The Dissertation Committee for Jaeho Hur Certifies that this is the approved version of the following dissertation:

MULTI-ROBOT SYSTEM CONTROL USING ARTIFICIAL IMMUNE SYSTEM

Committee:

Benito R. Fernández, Supervisor

Michael D. Bryant

Richard H. Crawford

Alfred E. Traver

Risto Miikkulainen

MULTI-ROBOT SYSTEM CONTROL USING ARTIFICIAL IMMUNE SYSTEM

by

Jaeho Hur, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2007

Dedication

I dedicate this dissertation to my parents.

MULTI-ROBOT SYSTEM CONTROL USING ARTIFICIAL IMMUNE SYSTEM

Jaeho Hur, Ph.D.

The University of Texas at Austin, 2007

Supervisor: Benito R. Fernández

For the successful deployment of task-achieving multi-robot systems (MRS), the interactions must be coordinated among the robots within the MRS and between the robots and the task environment. There have been a number of impressive experimentally demonstrated coordinated MRS. However it is still of a premature stage for real world applications.

This dissertation presents an MRS control scheme using Artificial Immune Systems (AIS). This methodology is firmly grounded in the biological sciences and provides robust performance for the intertwined entities involved in any task-achieving MRS. Based on its formal foundation, it provides a platform to characterize interesting relationships and dependencies among MRS task requirements, individual robot control, capabilities, and the resulting task performance.

The work presented in this dissertation is a first of its kind wherein the principles of AIS have been used to model and organize the group behavior of the

MRS. This has been presented in the form of a novel algorithm. In addition to the above, generic environments for computer simulation and real experiment have been realized to demonstrate the working of an MRS. These could potentially be used as a test bed to implement other algorithms onto the MRS.

The experiment in this research is a bomb disposal task which involves a team of three heterogeneous robots with different sensors and actuators. And the algorithm has been tested practically through computer simulations.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 From Single to Multi-Robot Systems[1]

The field of multi-robot systems (MRS) has received increased attention since the mid 1990's. Earlier research efforts had concentrated on either single robot systems (SRS) [Cao el al., 1995] or distributed problem-solving systems that did not involve robotic components [Carver et al., 1991]. This is not surprising as continually improving technology and infrastructure have made the deployment of MRS consisting of increasingly larger numbers of robots possible. With the growing interest in MRS comes the expectation that, at least in some important respects, multiple robots will be superior to a single robot in achieving a given task. Some potential advantages of MRS over SRS are summarized below:

- Total system cost may be reduced by utilizing multiple simple and cheap robots as opposed to a sophisticated and expensive robot.
- Some tasks may be accomplished more efficiently by a group of robots by decomposing the particular task into subtasks and performing each in parallel.
- The inherent complexity of some task environments may require the use of multiple robots as the required capabilities are too substantial to be met by a single robot.

[1] [Jones and Mataric, 2005; Arai et al., 2005].

- Multiple robots are often assumed to increase system robustness by taking advantage of inherent parallelism and redundancy. Therefore, negative effects on task performance caused by individual robot failure or the dynamic addition or removal of individual robots can be minimized.

1.2 DERIVING INSPIRATION FROM NATURE

Since the age of reason, man has looked to nature for inspiration to solve problems. Biologically-inspired computing is one area of research that inspired fields such as Genetic algorithms (GA) [Goldberg, 1989], Artificial Neural Networks (ANN) [Haykin, 1994], Evolutionary Programming, Artificial Immune Systems, and etc. These tools are the most popular to solve difficult, sometimes hard to model engineering problems.

Other fields of study that seek to mimic intelligent behavior of organisms and incorporate them in machines are Artificial Life (Alife) and swarm intelligence. An example of Alife is animats [Kodjabachian and Meyer, 1996] and they are artificial animals. Researchers have talked about scenarios where animats are capable of independent learning about their environment through application and evolution of pattern matching rules. Swarm intelligence is a technique based around the study of collective behavior in decentralized yet organized systems. Examples of systems like this can be found in nature, including ant colonies [Dorigo et al., 1996], bird flocking[2], animal herding [Schultz et al., 1996], and

[2] http://www.red3d.com/cwr/boids/

fish schooling[3]. In swarm intelligence interactions among *entities* lead to an emergence of a common global system behavior.

An Artificial Immune System (AIS) is a type of optimization algorithm inspired from the principles and processes of the vertebrate immune system [Manning, 1979]. The algorithms typically exploit the immune system's characteristics of *learning* and *memory* to solve a problem. The field of AIS is fairly new and began in the mid 80`s with Farmer, Packard and Perelson`s papers on immune system networks [Farmer et al., 1986]. Further details are enunciated in the relevant chapters.

1.3 RESEARCH MOTIVATION

Even though MRS may produce robust solutions, the utilization of MRS poses potential disadvantages and additional challenges that must be addressed if MRS are to present a viable and effective alternative to SRS in an important subset of domains. A poorly designed MRS, with individual robots working toward opposing goals, can be less effective than a carefully designed coordinated SRS [Jones, 2005]. The opposing goals can be analogous to conflicting objectives in multi-objective optimization, e.g. in a gear train design problem, higher gear ratio and cost are opposing objectives.

Many researchers have come to realize that the design of MRS is in many critical respects a very different challenge from the design of single robot systems. In most cases just taking a suitable SRS design and scaling it up to

[3] http://freshaquarium.about.com/cs/beginnerinfo/a/schooling.htm

multiple robots is not adequate [Arai et al., 2005]. A paramount challenge in the design of effective MRS is managing the complexity of a group's control introduced by multiple, interacting robots as well as conflicting individual goals.

To date, the design of MRS has remained ad hoc. and, as such, few formal methodologies have been devised. This lack of design procedures has limited the usefulness of MRS and has prohibited the growth of MRS solutions for many potential domains.

In the mean while, Artificial Immune Systems (AIS) have appeared as a new computational approach for the computational intelligence community. Like other biologically inspired techniques, it tries to extract ideas from a natural system, in particular the vertebrate immune system, with an aim to develop a computational platform for solving engineering problems. There are many things in common between AIS and MRS in terms of distributed entities and processing, which is discussed later. In this research, AIS's distributed learning structures will be studied and applied to orchestrate the group of robots.

1.4 RESEARCH CONTRIBUTION

The work presented in this dissertation is a first of its kind wherein the principles of AIS have been used to model and organize the group behavior of MRS. This result is presented in the Chapter 3 in the form of a novel algorithm. In addition to the above, generic environments for the virtual and real experiments were realized to demonstrate the working of a MRS. This could potentially be used as a platform to implement other algorithms onto the MRS. The proposed

algorithm has been implemented experimentally. The test bed scenario that has been proposed for final experiments involves a bomb disposal task which has been successfully performed by three robots with different sensors and actuators.

1.5 DISSERTATION OUTLINE

The remainder of this dissertation is organized as follows. The current chapter discussed a brief introduction and outline.

Chapter 2 presents a brief literature review of background concepts and previous work related to MRS and AIS.

Chapter 3 proposes the control architecture which is used to control MRS using AIS.

Chapter 4 presents the software and hardware environments to realize and verify the proposed algorithm.

Chapter 5 shows the results and discussion of the virtual environment experiment.

Chapter 6 draws conclusion of this dissertation with a summary of contributions and directions for future work.

Chapter 2: Background Concepts and Related Work

2.1 MULTI-ROBOT SYSTEMS (MRS)

2.1.1 System Description

Figure 2.1 shows an example of a MRS, where many robots are searching for food and taking it home. They have sensors to find food and to detect obstacles. They can communicate with each other to exchange information. Also, there is a home area where they can rest (battery recharge, maintenance) and store food.

Cooperative autonomous robot groups may have a number of advantages over a single complex robot system. Robot groups can readily exhibit the characteristics of structural flexibility, reliability through redundancy, simple hardware, adaptability, reconfigurability, and easy maintainability [Liu et al., 1998]. Real-life applications of such autonomous robot groups can be found in literature. Some examples include: explosive ordinance disposal [McLurkin, 1996], welfare robots [Yamaguchi et al., 1998], etc.

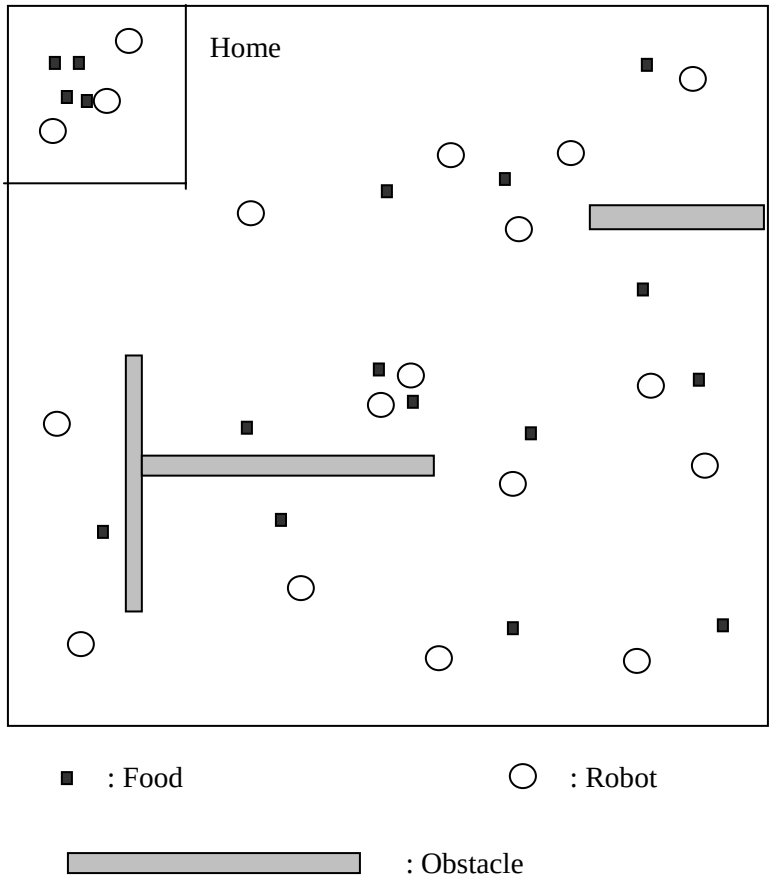


Figure 2.1: Example of a Multi-robot system (MRS) searching for food

There are robots (white circle) looking for foods (black squares). The robots collect food and carry it to the area denoted as "Home".

- **Typical system requirements and constraints**

In MRS, robots sometimes must give up attaining the best possible individual solutions in favor of "collective efficiency" of the group of robots. In general, MRS are designed such that, at least in theory, it is in the interest of each

of the individuals to try and attain the group objective since, on the average; their individual efficiency will be improved as well.

However, since the connection between individual and collective benefit is not always obvious, the problem of implementing a common overall goal is a difficult one. Usually there is no central unit to orchestrate the overall movement or plan of each robot, therefore, their individual decision depends totally on distributed resources and restricted communication. The communication is in some instances limited to a local area or by bandwidth. The global position of each robot and the environmental model are sometimes not usually available for each robot. The conditions above can be easily found in many applications of multi-agent robot systems. For example, when a robot colony is to explore another planet, there are not enough computational resources; hence local memory should be utilized with minimal overload.

This kind of system is called an autonomous decentralized system (ADS) [Ishida and Adachi, 1996]. Self-organizing economic systems of free market, organization of nations, and development of enterprises are good examples of ADS, not to mention self organizing biological systems such as the human immune system. The difference is, in MRS, the agents (robots) are equipped with relatively poor intelligence (as compared to huge computational resources of a company) and limited sensors (as compared to well distributed sensory organs of animals).

- **Implementation Barriers**

When scaling the robot system from single-agent to multi-agent domain, the dimension of the global state space $\mathbb{G}$ grows exponentially with the number of robots: $|\mathbb{G}| = s^a$, where, s is the size of the state space of each robot and a is the number of robots [Mataric, 1994]. Here s is assumed to be equal for all robots or at worst the maximum for all robots[4]. This makes the problem of on-line planning intractable for all but the smallest group sizes. Furthermore, since global planning requires communication between the robots and the controller, the bandwidth requirements grow with the number of robots. Additionally, the uncertainty in perceiving states grows with the increased complexity of the environment. All of these properties conspire against global planner-based approaches[5] for problems involving multiple robots acting in real-time in dynamic, noisy environments.

2.1.2 MRS Taxonomy

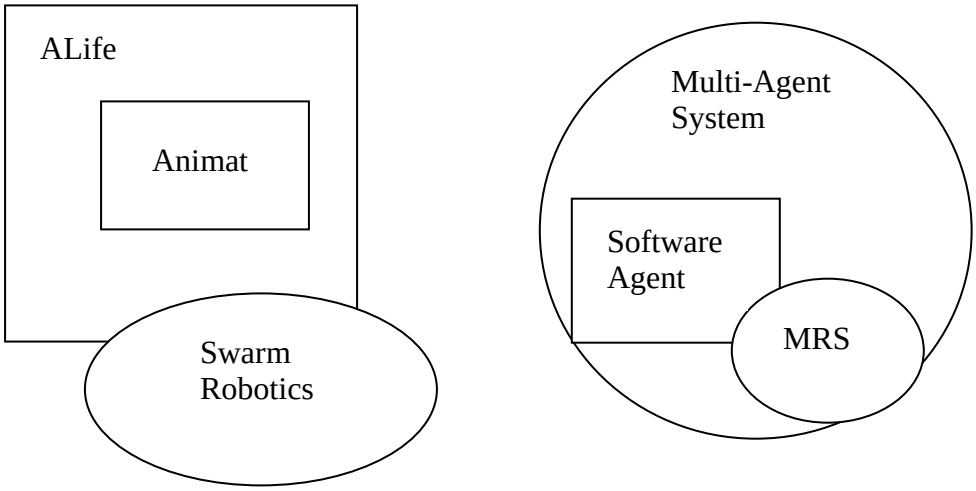
2.1.2.1 MRS Research Communities

There have been many researchers from many different research communities who have studied MRS from different perspectives. It is difficult to distinctly identify those research domains as they greatly overlap with each other. Fig 2.2 is a brief diagram on research communities about MRS. The figure 2.2a depicts ideas that have been derived from nature and implemented in the form of Animats and Swarm robotics. The figure 2.2b shows the analogy of figure 2.2a in the engineering domain.

---

[4] $s = \max_i \{s_i\}$ where s_i is the size of the state space for some robot i.

[5] Use a centralized mathematical world model for verifying sensory information and generating actions in the world.



(a) (b)

Figure 2.2: Research communities of MRS

2.1.2.2 Overall Structure of MRS [6]

Multi robot systems are defined by several key characteristics. Most of these have been described below:

- **Cooperative vs. Non-cooperative**

  In cooperative MRS individual robots assist each other in reaching the overall common goal while this is not true of non-cooperative MRS.

- **Coordinated**

[6] Summarized and rephrased from [Iocchi et al., 2001] and http://www-scf.usc.edu/~csci445/

This feature essentially means that there is a clear cut internal hierarchy among the individual robots of a MRS. This hierarchy influences properties such as decision making, social roles etc.

- **Centralized vs. Distributed**

  MRS can be centralized similar to the central nervous system in a human body under the command of a brain or distributed similar to the distributed immune system of the human body.

- **Direct communication vs. Indirect communication**

  The individual robots in a MRS may communicate directly with each other using some standard protocol or they may upload and download data from a centralized server like blackboard.

- **Homogeneous vs. Heterogeneous**

  A homogenous MRS usually has all robots which are exactly the same in size, shape and functionality. A heterogeneous MRS has individual robots which may differ from each other in size, shape or some other feature such as processing power, control algorithms, sensors/actuators, locomotion, etc.

- **Reactive vs. Deliberative**

  A reactive MRS is one where the individual robots perform actions based upon information from sensors, and in a deliberative MRS the individual robots work based both upon sensor data and either historic data (state machine) and/or data from other robots and/or commands. This is performed by the behavioral algorithm imposed upon it.

- **Hybrid Control vs. Behavior Based Control**

Finally MRS can be categorized depending on how to combine reactive and deliberative schemes: hybrid or behavior based.

2.1.2.3 Action Selection Mechanism (ASM)[7]

An Action Selection Mechanism (ASM) is a decision making process based on sensory information, hierarchy or other variables. ASM's ability to accommodate multiple behaviors simultaneously divides ASM's into two main groups: Arbitration and Command fusion as in Figure 2.3.

Arbitration ASMs (a-ASMs), allow one behavior at a time to take control for a period of time until another set of behaviors is activated. In priority-based a-ASMs, an action is selected by a central module based on priorities that have been assigned a priori. Thus, behaviors with higher priorities are allowed to take control of the robot superseding behaviors lower in the priority rank. State-based a-ASMs select the set of behaviors that is adequately competent of handling the situation corresponding to the given state. Winner-take-all arbitration action selection results from the interaction of a set of distributed behaviors that compete until one behavior wins the competition and takes control of the robot.

[7] Summarized and rephrased from [Pirjanian, 2005].

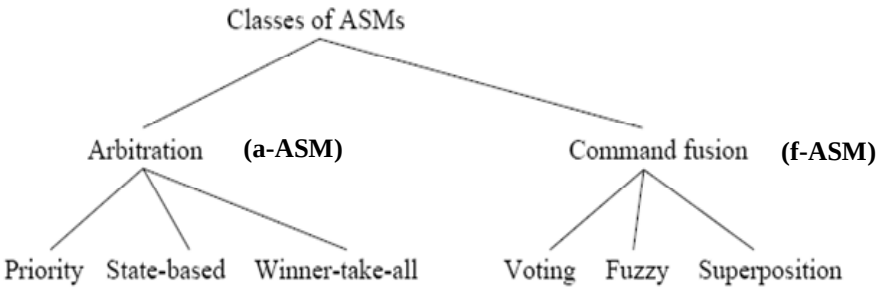


Figure 2.3: ASM classifications[8]

Command fusion ASMs (f-ASMs), allow multiple behaviors to contribute to the final control of the robot. Voting f-ASM techniques interpret the output of each behavior as votes which are combined by tallying the votes and then utilize this vote ranking. Fuzzy f-ASMs are very similar to voting techniques however fuzzy inferencing techniques [Ross, 1995] are used to implement fusion among possible actions. Finally, superposition f-ASM techniques combine behavior recommendations using linear combinations.

2.1.3 Related Work

Social and ecological systems exhibit a structure that is desirable for multi robot systems. For example, insect colonies achieve organized group behaviors in

[8] [Pirjanian, 2005]

complete absence of centralized control and an environmental model. Therefore, many ideas inspired from biological and social systems are applied to MRS.

Mataric [Mataric, 1995] extended Brooks' [Brooks, 1987] work to the multi-robot case. First, she defined basic primitive social behaviors for the robots, and made them learn social behaviors with a reinforcement learning algorithm. Her approach basically is that each robot can gradually understand its social rules (defined by a human).

McLurkin [McLurkin, 1996] applied behavior-based approach to small micro-robots. Liu et al. [Liu, 1998] developed Mataric's work further by adding evolving group behavior function using a genetic algorithm.

Jun and Sim [Jun and Sim, 1997] used fuzzy inference based reinforcement learning and a distributed genetic algorithm for behavior learning. Their interests were learning processes not social behaviors. Maeda [Maeda, 1997] developed a very simple evolutionary algorithm, which mimics genetic algorithms, to train collision avoidance and target reaching behaviors. The study of Goldberg [Goldberg, 1996] was on heterogeneous robot group behavior.

Kelly and Keathy [Kelly and Keathy, 1998] used reinforcement learning and showed that sharing robots' experiences results in faster and more repeatable learning of each robot's behavioral parameters. Ashiru and Czarnecki [Ashiru and Czarnecki, 2002] focused on the communication between robots, and used genetic algorithm to evolve communication protocol. Floreano and Noli [Floreano and Noli] modeled competing co-evolving species (e.g., prey and predator).

Yamaguchi et al. [Okura et al., 2003] took chaotic dynamics as an evolutionary computational method for robot learning.

In this section, an overview of the work done in MRS related areas has been briefly covered. This overview is not exhaustive rather it summarizes some of the major efforts in the field of robotics which are related to action selection and uncertainty handling of MRS.

2.2 ARTIFICIAL IMMUNE SYSTEMS (AIS)

2.2.1 Immune System Basics

- What is an Immune system (IS)?

The IS is a very complex biochemical system with several mechanisms for defense against pathogenic organisms, toxins and other foreign molecules, collectively known as antigens. The main function of the IS is to recognize all cells (or molecules) within the body and categorize those cells as self or nonself. The nonself cells are futher categorized in order to induce an appropriate type of defensive mechanism. The IS learns through evolution to distinguish between dangerous foreign antigens and the body's own cells or molecules.

The atomic or fundamental constituents of the IS are the lymphocytes which circulate throughout the body, mainly of two types, namely B-lymphocytes (B-cells) and T-lymphocytes (T-cells). B-lymphocytes are the cells that mature in the bone marrow and T-lymphocytes are the cells maturing in the thymus. Each of them has a distinct molecular structure and functionality.

- How does the IS work[9]?

There are two kinds of immunities: innate and adaptive immunity. The Innate IS is a primitive system of defense against the pathogens. It is nonspecific; that is, it is not directed against specific invaders but any pathogens that enter the body. On the other hand, the adaptive IS (performed by cooperation of B-cells and T-cells) is an additional and more sophisticated system of defense mechanism, and can recognize and destroy specific substances. The way B-cells and T-cells can identify specific substances is called a key and key hole relationship.

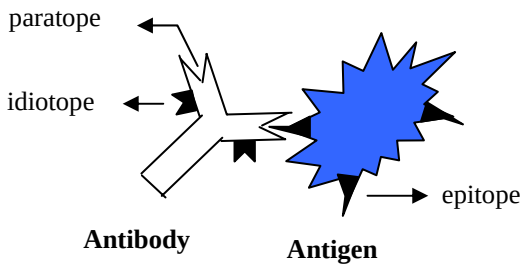


Figure 2.4a: Immune system working

[9] Summarized and rephrased from [Dasgupta and Attoh-Okine, 1997], [Jun and Sim, 1997], [Kondo et al., 1998] and [Mitsumoto et al., 1997].

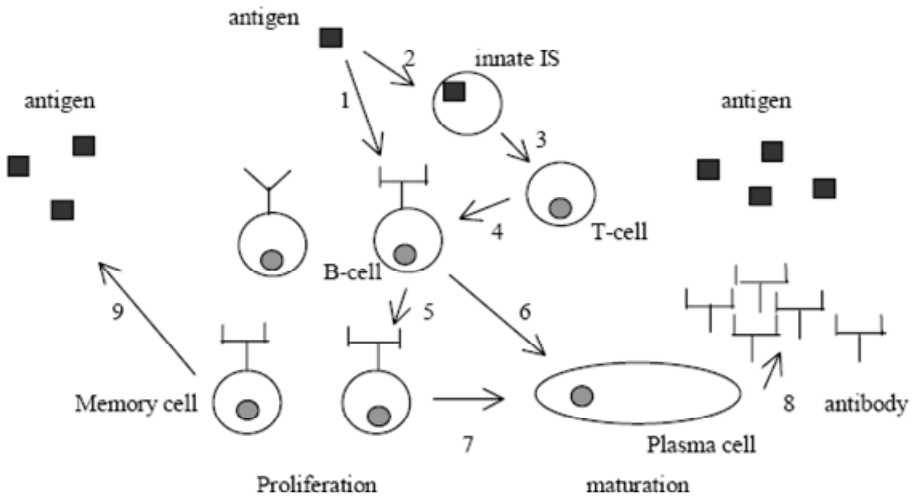


Figure 2.4b: Immune system working

This figure shows various stage of immune system (IS) process when antigen invades the IS. The numbers on the arrows represent reaction process.

Figure 2.4a shows the receptor areas at the surface of each cell. The shape of the receptors uniquely define the kind of other antibody cells or antigens it can identify. Each process of IS (the numbered arrow in Figure 2.4b) will be mentioned in detail.

When an antigen invades the human body, innate IS will try to neutralize it first (2). If the receptor shape of the antigen matches with the shape of the B-cell, B-cell production is stimulated and through processes (6) and (8), a lot of antibodies are generated. If the shape of the antigen does not match perfectly with the receptor of the B-cells, a strong learning process called hypermutation can make B-cells match the antigen. The role of the T-cells is to regulate the

stimulation of B-cells (4). After the learning process, some of B-cells are “stored” as memory cells to prepare antibodies upon appearance of the same antigen (5). If the same antigens show up again in the body, IS can react very fast via process (7).

2.2.2 Models on AIS

There are three major models on IS for use with engineering applications [Dasgupta and Attoh-Okine, 1997]: Negative Selection, Clonal Selection and Immune Network. Each of them will be described in this section.

2.2.2.1 Negative Selection[10]

The human immune system makes use of gene libraries from two types of organs called the thymus and the bone marrow. When a new antibody is generated, the gene segments of different gene libraries are randomly selected and concatenated in a random order. The main idea of this gene expression mechanism is that a vast number of new antibodies can be generated from new combinations of gene segments in the gene libraries.

However, this mechanism introduces a critical problem. The new antibody can bind not only to harmful antigens but also to essential self cells. To help prevent such serious damage, the human immune system employs negative selection. This process eliminates immature antibodies, which bind to self cells passing by the thymus and the bone marrow. From newly generated antibodies,

[10] Summarized and rephrased from [Kim and Bentley, 2001].

only those which do not bind to any self cell are released from the thymus and the bone marrow and distributed throughout the whole human body to monitor other living cells.

Inspired by this idea, Forrest et al. [Forrest et al., 1994] developed an anomaly detection algorithm based upon the negative selection of T-cells within the thymus and applied it for computer security systems. The interesting aspect of this algorithm is that it can be used to perform tasks like pattern recognition by storing information about the set of patterns that are unknown to the system.

2.2.2.2 Clonal Selection[11]

When the antibodies on a B-cell recognize an antigen with a certain affinity (degree of match), the B-cell will be stimulated to proliferate (divide) and eventually mature into terminal (non-dividing) antibody secreting cells, called plasma cells. Proliferation of the B-cells is a mitotic process whereby the cells divide themselves, creating a set of clones identical to the parent cell. The proliferation rate is directly proportional to the affinity level, meaning that B-cells with higher affinity levels will be more readily selected for cloning and cloned in larger numbers compared to others. More specifically, during asexual reproduction, the B-cell clones experience somatic hyper-mutation; a random structural change.

[11] Summarized and rephrased from [Ong et al., 2005] and [Engin and Döyen, 2004].

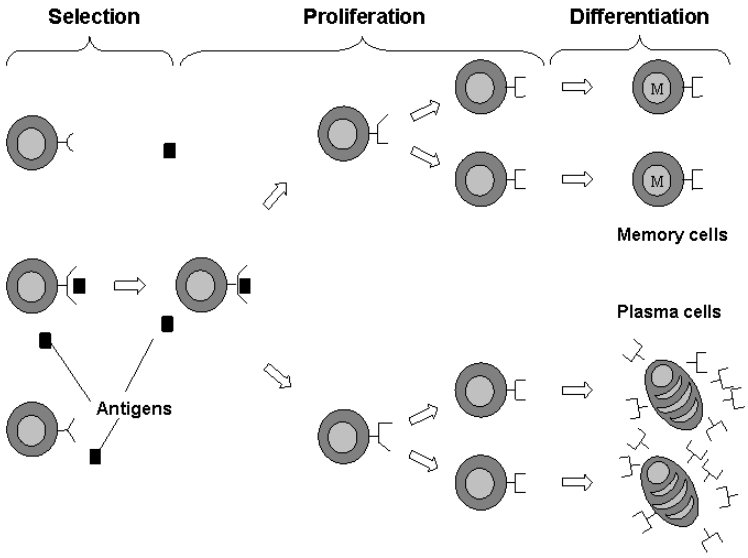


Figure 2.5: Clonal selection process[12]

The mutation on the cloned cells occurs at a rate which is inversely proportional to the antigen-affinity. Clones of higher affinity cells are subjected to less mutation compared to those from cells which exhibit lower affinity. This process of constant selection and mutation of only the B-cells with antibodies which can better recognize specific antigens is known as affinity maturation. Though the repertoire of antibodies in the immune system is limited; through affinity maturation, it is capable of evolving antibodies to successfully recognize and bind with known and unknown antigens, leading to their eventual elimination.

[12] [de Castro and Von Zuben, 1999]

The immune system also possesses memory properties as a portion of the B-cells will differentiate into memory cells, which do not produce antibodies but instead remembers the antigenic pattern in anticipation of future re-infections. These memory cells circulate within the host body. In response to a second antigenic stimulus, they differentiate into plasma cells to produce antibodies which have high affinity. This feature of the clonal selection is not integrated into the proposed algorithm, so it will no longer be mentioned in the paper. The whole clonal selection principle has been shown as in Figure 2.5.

These immunological processes of clonal selection (and affinity maturation) have been used for inspiration in AIS, the most common abstraction being Clonalg [de Castro and Von Zuben, 1999]. Clonalg currently exists in two similar but distinct forms—one for optimization and one for pattern matching—but in both cases the B-cell is implemented as a single real-valued vector and no two B-cells are allowed to interact.

2.2.2.3 Immune Network[13]

In 1974 Jerne [Jerne, 1974] proposed the immune system network hypothesis as a mechanism for regulating the antibody repertoire, although it has not gained wide acceptance within the field of immunology partly because of the implementation complexity. The hypothesis is based on the fact that similar to paratopes (for epitope recognition), antibodies also possess a set of epitopes and so are capable of being recognized by other antibodies even in the absence of

[13] Summarized and rephrased from [Whitbrook, 2005].

antigens. Under the clonal selection theory all immune responses are triggered by the presence of antigens, but under the network theory antibodies can be internally stimulated. (Experiments have shown that the number of activated lymphocytes in germ free mice is similar to that of normal mice, which supports the argument.)

Paratopes and epitopes (Figure 2.4a) are complimentary and are analogous to keys and locks. Paratopes can be viewed as master keys that may open a set of locks (epitopes), with some locks being opened by more than one key (paratope). Epitopes that are unique to an antibody type are termed idiotopes and the group of antibodies that share the same idiotope belong to the same idiotype.

When an antibody type is recognized by other antibodies it is suppressed that is, its concentration is reduced, but when an antibody type recognizes self antibodies or antigens it is stimulated and its concentration increases. The theory explains the suppression and elimination of self-antibodies and presents the immune system as a complex network of paratopes that recognize idiotopes and idiotopes that are recognized by paratopes (Figure 2.6). This implies that B-cells are not isolated, but are communicating with each other via collective dynamic network interactions.

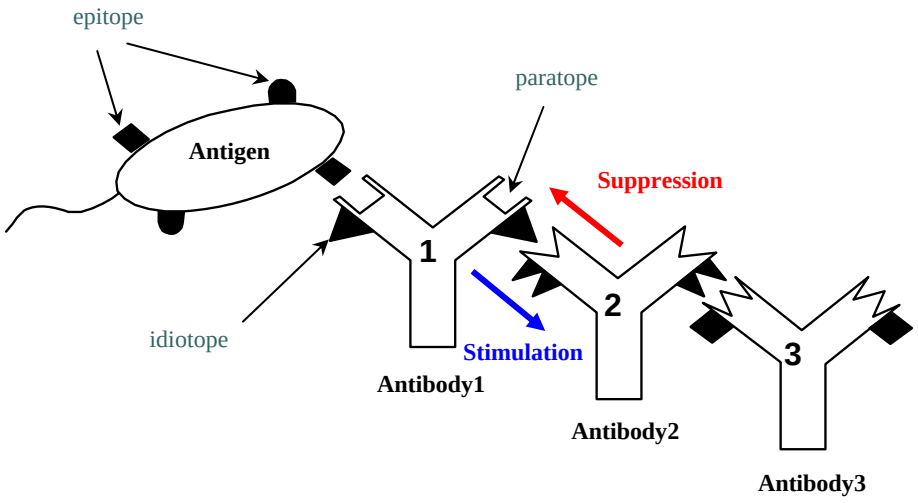


Figure 2.6: Idiotyping network[14]

2.2.3 Related Work

Luh and Liu [Luh and Liu, 2004] used a reactive immune network for robot obstacle avoidance, trap escapement and goal reaching in an unknown and complex environment with both static and dynamic obstacles. Their architecture consisted of a combination of previously observed behavior based components and an adaptive component modeled on the immune network theory.

Krautmacher and Dilger [Krautmacher and Dilger, 2004] applied Farmer's immune network model to robot navigation in a simulated maze world in which a

[14] [Whitbrook, 2005]

building had collapsed due to an earthquake. The robot's task was to find victims, determine their situation and location and record the information on a data sheet. No a priori knowledge of the maze or object locations was given; fuzzy identification of objects was achieved through image processing and comparison with stored information. Location and identification of a given object was analogous to the presence of an antigen, and its type and location were used as epitopes. Many potentially useful antibodies representing basic behaviors were used and as the system evolved new antibodies emerged and were added to the system.

Vargas et al. [Vargas et al., 2003] constructed a hybrid robot navigation system (CLARINET) that merged ideas from learning classifier systems, (introduced by Holland in the mid seventies, see [Holland, 1986]) and the immune network model of Farmer et al. [Farmer et al., 1986].

Learning classifier systems have been linked to artificial immune systems by Farmer et al. [Farmer et al., 1986] and Vargas et al. [Vargas et al., 2003]. Antibodies can be thought of as classifiers with a condition and action part (the paratope) and a connection part (the idiotope). The action part must be matched to a condition (antigen epitope) and the connections show how the classifier is linked to others. The presence of environmental conditions causes variations in classifier concentration levels in the same way that antigens disturb antibody dynamics.

Learning classifier systems have frequently been used to solve mobile robotics problems. Stolzmann [Stolzmann, 1999] applied them to robot learning

in a T-shaped maze environment and Carse and Pipe [Carse and Pipe, 2004] used a fuzzy classifier system. Webb et al. [Webb et al., 2003] used classifiers with reinforcement learning for the autonomous navigation of simulated mobile Khepera robots[15] that were required to find and travel to target locations.

[15] http://www.k-team.com/kteam/index.php?rub=3&site=1&version=EN&page=3

Chapter 3: Proposed Architecture for MRS

There can be many factors influencing MRS control architecture design like other design problems. If there are specific robot mission requirements, detail design procedures should be focused to meet those requirements. Also, in some case, there can be a need for an exhaustive benchmarking among specific MRS control algorithms.

However, the proposed architecture in this research is based on how to implement a control architecture using current AIS model and verify it while considering our research institute resources. And it is not designed for any specific mission.

3.1 STRUCTURE FOR THE MRS

In Chapter 2, various aspects of MRS taxonomy were discussed. In this section, MRS structure will be discussed keeping in mind scalability issues and combining the AIS for the *learning* role.

3.1.1 Overall Structure

To improve the robustness of the MRS for a particular mission, distributed organization instead of a central organization is preferred vis-à-vis the group behavior. This scheme offers physical and logical redundancy. For cooperative and robust group behavior, a distributed control algorithm will be adopted. To reduce the gap between reactive and deliberative decision making, a behavior-based control scheme will be used.

Most behavior-based systems are also reactive, which means they use relatively little internal variable states to model the environment, most of the information is gleaned from the input of the robot's sensors. The robot uses that information to react to the changes in its environment. Behavior-based robots (BBR) usually show more biologically analogous actions than their computing intensive counterparts, which are very deliberate in their actions.

3.1.2 Internal Structure

Figure 3.1 shows the block diagram of the proposed MRS control scheme. There are sensors to detect the environment and other robots, and a communication agent to share information with other robots. A blackboard [Carver and Lesser, 1994] has been added to help the robot to share previous experiences with other robots. All robots write their knowledge on this board that is shared. The robot’s current condition depends on the sensor readings, communication information and the contents of the blackboard. There are basic behaviors prepared in advance for the whole mission of the robots (BBR), and actuators move or act depending on individual behaviors.

Considering the scalability issue to control many robots, condition and behavior mapping has the merit to reduce the number of states to train a robot, for example, there are many sensor inputs to define each robot current state. If those are considered for many robot cases, the sensor state space will be dramatically increased. However, current approaches use conditions that lump the sensor state space reducing the number of conditions when considering other robots.

In this research, a lookup table (Table 3.1) is used to map conditions to behaviors. From the taxonomy view point developed in the previous chapter, the proposed control architecture can be categorized as a behavior based and state based ASM structure. This lookup table is tuned by using AIS during robot learning process. And learning process will be discussed in the next section.

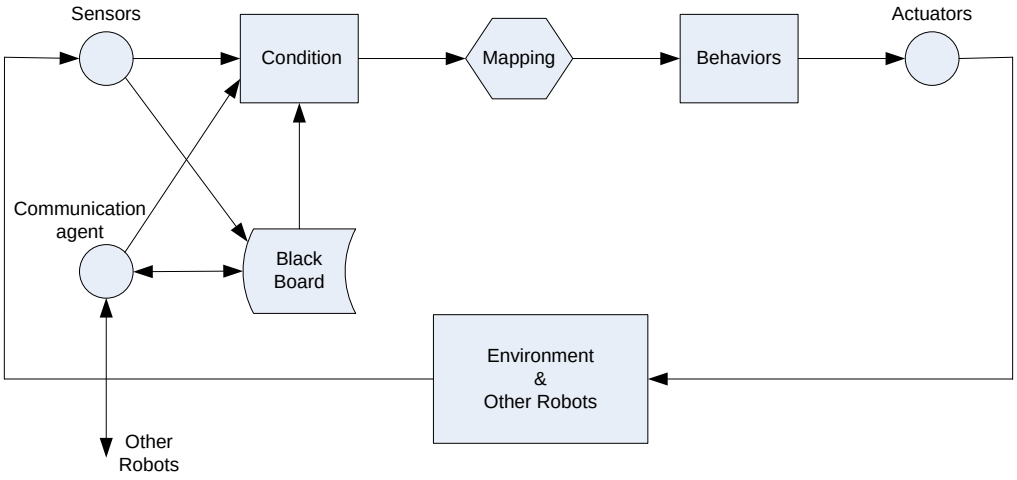


Figure 3.1: MRS internal control schema

Condition	Basic Behaviors				
	BB1	BB2	BB3	BB4	BB5
000000	1	0	0	0	0
000001	0	0	1	1	0
000010	0	0	1	0	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

111101	0	1	0	0	0
111110	0	0	0	1	0
111111	0	6	1	0	0

Table 3.1: Mapping between condition and basic behaviors

Condition is based on the current robot states. Robot has 5 basic behaviors in this table.

3.2 LEARNING ALGORITHM FOR THE MRS

Three main streams of AIS research has been covered in the previous chapter. From the paper survey for this research, clonal selection is the most adequate solution for the organization of each robot behavior. That is because the negative selection is naturally suited for fault detection or virus detection application. On the other hand, the immune network theory is still early stage of its development.

3.2.1 Applying Clonal Selection[16]

In order to apply Clonal Selection to control the MRS, an antibody is represented as a vector. The elements of the vector are deduced from the lookup table. The following are the steps (modeled after [He et al., 2005]) integral to the learning algorithm (Figure 3.3):

- **Initialization (1)**

[16] Modified from [He et al., 2005].

The initial population containing a set of feasible solutions, or antibodies is created randomly regardless to their affinity measurement value.

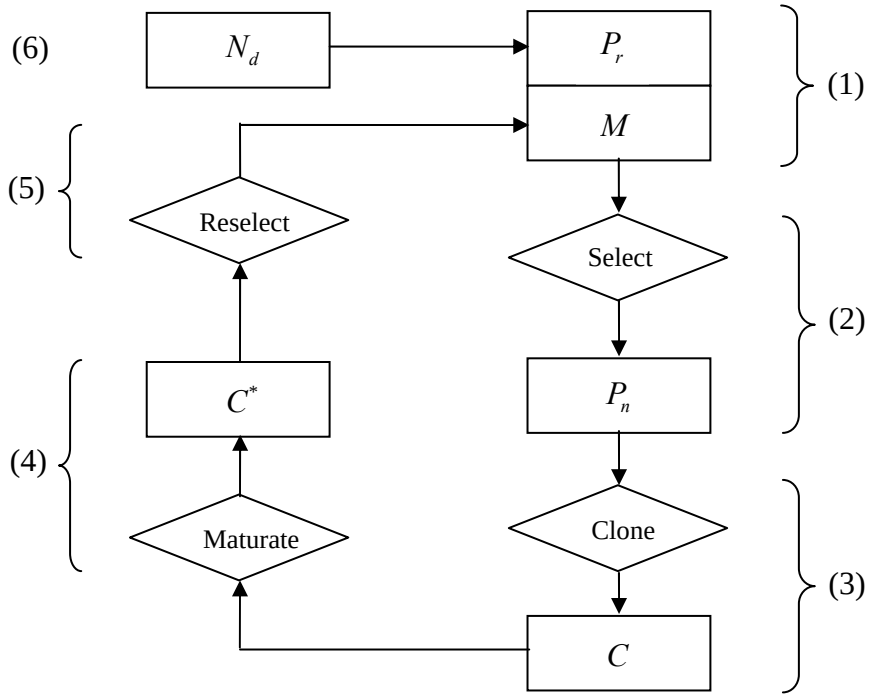


Figure 3.3: Computational procedure for clonal selection[17]

- **Selection (2)**

The selection process by running robots during computer simulation or experiment begins with the evaluation (running robots in this research) of the affinity of each antibody: These antibodies are then sorted increasingly according

[17] [He et al., 2005]

to the affinity calculated. The first antibody in the sorted list has the lowest affinity and the last one has the highest affinity. The affinity measurement function is defined as

$$f = 1/(1+\sum_{i=1}^{k} w_i \cdot n_i) \qquad \text{eq (3.1)}$$

where;

k = total number of the soft constraints defined.

n_i = number of a certain kind of soft constraints within a particular antibody.

w_i = attached penalty or weight.

After the ordering of antibodies, the n highest affinity antibodies are selected to produce a new population P_n. If we choose $n = N$, that is, the number of highest affinity individuals equals to the number of candidates, each member of the population will constitute a potential candidate solution locally, characterizing a greedy search. In addition, if all the individuals are accounted locally, their clones will have the same size. The value of the parameter n is in general determined empirically.

- **Cloning (3)**

Antibodies in the population will be duplicated proportional to their affinity and enter the clone population C (Figure 3.2) of size N_c, which is computed by equation (3.2)

$$N_c = \sum_{i=1}^{n} round(\frac{\beta \cdot N}{i}) \qquad \text{eq (3.2)}$$

where;

N_c = total amount of clones generated.

β = multiplying factor.

N = total amount of antibodies.

round(·) = operator that rounds its argument towards the closest integer.

Each term of this sum corresponds to the clone size of each selected antibody, for example, for N = 100 and β = 1, the highest affinity antibody (i = 1) will produce 100 clones, while the second highest affinity antibody produces 50 clones, and so on.

- **Maturation (4)**

The mutation rate of a cell is inversely proportional to the affinity of the cell. It gives the chance for low affinity cells to “mutate” more in order to improve their affinity. Since the mutations can result in better affinity antibodies, the immune system searches to climb up the hill towards higher affinity antibody, leading to local optima.

A lookup Table 3.1 picked from the selection process will be mutated with mutation rate inversely proportional to its affinity function measure.

- **Reselection and Diversity Introduction (5)**

The *n* highest affinity antibody clones will then be selected after running robots during simulation or experiment to compose the new population of P_n, and low affinity antibodies are to be replaced by the diversity introduction process.

Figure 3.2 shows a block diagram how a robot can be learned while interacting with other robots. The clonal selection generates antibodies during learning and they are evaluated in the robot environment to evaluate affinities for further generation of antibodies depending on the stage of the learning.

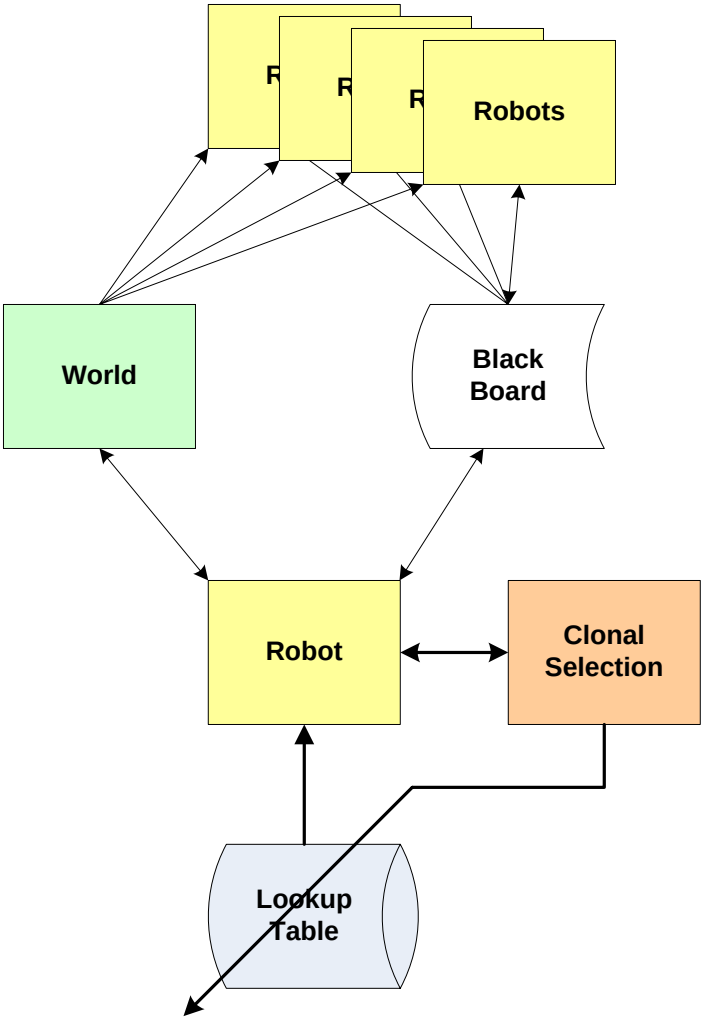


Figure 3.2: MRS overall control schema

Robot is interacting with other robots and the world. Robot communication is done via blackboard. A learning algorithm (clonal selection) tunes the lookup table during robot learning.

3.2.1 Comparison of the proposed algorithm using Clonal Selection (CS) with Genetic Algorithm (GA)[18]

[He et al., 2005] used a clonal selection algorithm to solve the university timetabling problem which is very similar to the lookup table problem of this research. They also benchmarked with GA. Their preliminary experimental results indicate that the CS performs better than GA when tested on the university timetabling benchmark data. From the data, it is noticed that CS maintains a diverse set of local optimal solutions, while GA tends to polarize the whole population of individuals towards the best one. This is mainly due to the selection and reproduction schemes adopted by the CS.

From [de Castro and Von Zuben, 1999], compared the decoded average value of a multi-modal sinusoidal function, for the whole population, evolved by the GA and the CS algorithms. The GA approach presented a greater average value, indicating a less diverse set of individuals. Both strategies successfully determined the global optimum.

While the GA uses a vocabulary borrowed from natural genetics and is inspired in the Darwinian evolution, the CS makes use of the shape-space formalism, along with immunological terminology to describe antigen antibody interactions and cellular evolution. The CS performs its search, through the mechanisms of somatic mutation and receptor editing, balancing the exploitation of the best solutions with the exploration of the search-space. Essentially, CS's

[18] [He et al., 2005] and [de Castro and Von Zuben, 1999]

encoding scheme is not different from that of GA but their evolutionary search differs from the viewpoint of inspiration, vocabulary and fundamentals.

Next chapter describes how to implement the proposed algorithm for simulation and experiment to verify the algorithm.

Chapter 4: Test Environment Setups

In this chapter, two different setups to test the proposed algorithm will be explained. A group of heterogeneous robots were developed. And infrastructures, such as communication and positioning systems, to support their evolutionary learning were made.

Five computers (named: Cyberspace, DefuserBot, ScannerBot, InspectorBot and Satellite) are used for the experiments. They are running Microsoft Windows XP Professional 2002 as an operating system and specifications are as followings:

- Cyberspace: AMD Athlon, 1.2 GHz, 640 MB RAM
- DefuserBot: Intel Pentium 4, 1.7 GHz, 512 MB RAM
- ScannerBot: Pentium 4, 1.49 GHz, 512 MB RAM
- InspectorBot: Intel Pentium III, 863 MHz, 512 MB RAM
- Satellite: Intel Pentium 4, 1.7 GHz, 512 MB RAM

4.1 SIMULATION SETUP

4.1.1 Overview

To prove the efficiency and robustness of the proposed AIS algorithm for the MRS, this section describes a computer simulation environment as shown in Figure 4.1. There are five computers (named: Cyberspace, DefuserBot, ScannerBot, InspectorBot and Satellite) connected to a local area network (LAN)

via Ethernet. In the Cyberspace, there is a 3D visualization tool called Webots[19] to simulate the situated robots. Each robot is controlled independently by the individual personal computers (PC) assigned to it. Matlab is used to make the robots learn and evolve, while C language is used for communication among PC's (robots in other words) and between Webots and Matlab. Robots can share their knowledge via blackboard using the network connection. The Satellite is in charge of collecting position information of all the robots from the supervisor in the Webots and can pass that to other PC's upon their requests. Satellite can also reset each robot to its initial position during learning and the blackboard. Figure 4.2 is a screen shot of Webots with three robots in the simulation environment.

[19] http://www.cyberbotics.com/

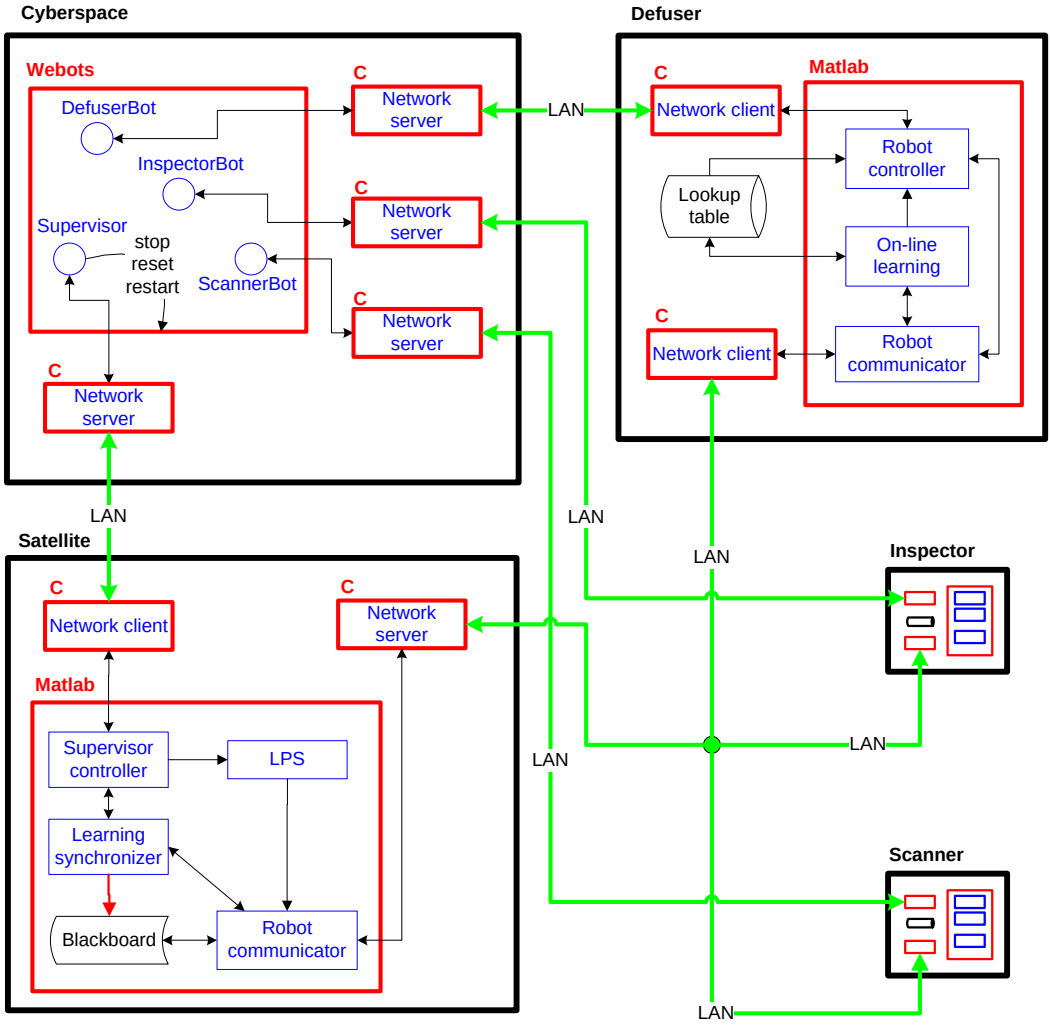


Figure 4.1: Overall control diagram of MRS

In the Figure, the PC for DefuserBot is exploded to show its internal details. All robots are similar in terms of blocks. They differ in their functionality.

- **Webots**

Webots is a robot simulation software from Cyberbotics, Ltd. It contains a virtual design tool allowing the user to create 3D virtual worlds complete with the physics. The user can add simple inert objects such as an obstacle or active objects. These robots can have different locomotion mechanisms and they can be equipped with a number of sensor and actuator devices, such as distance sensors, motor wheels, cameras, servos, touch sensors, grippers, emitters, receivers, etc. Finally the user can program each robot individually to exhibit a desired behavior.

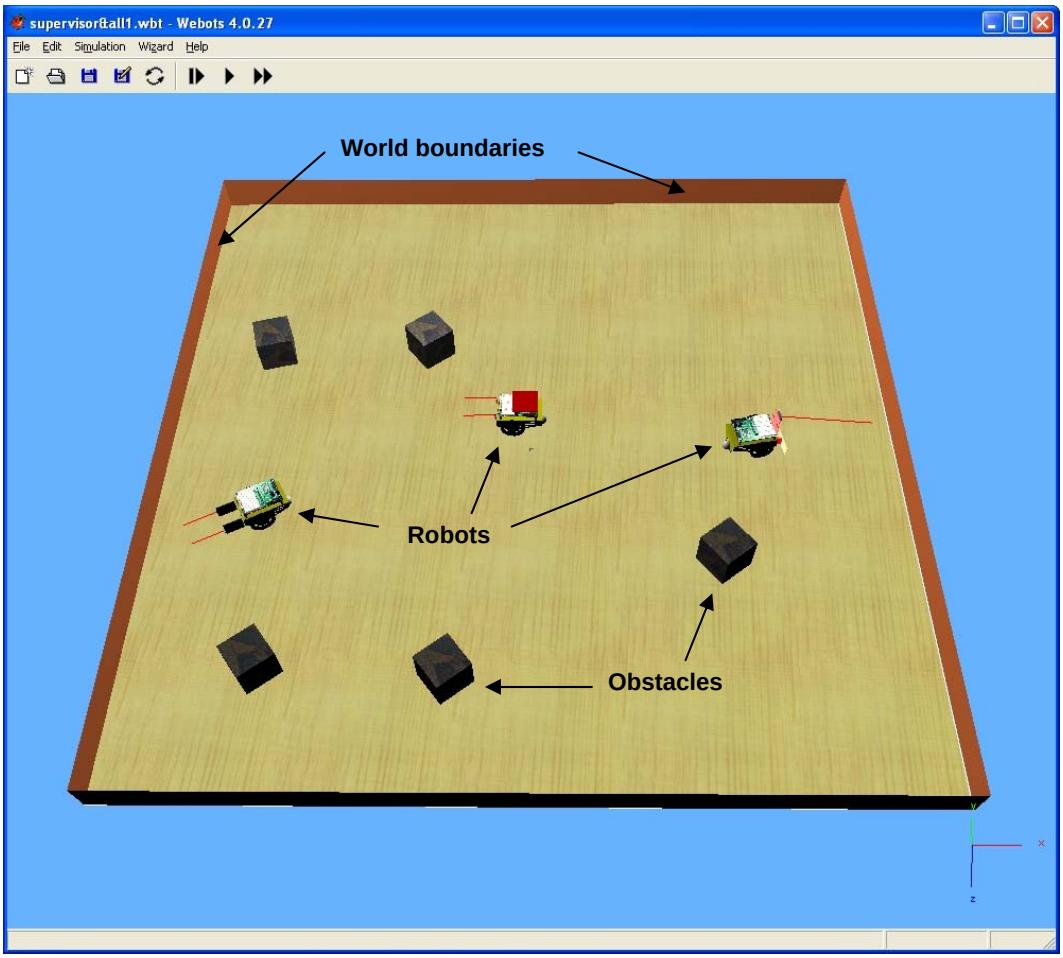


Figure 4.2: Screen shot of Webots for MRS

4.1.2 Entities

4.1.2.1 Robots

There are three different robots built and modeled in Webots for the purposes of this doctoral research (Figure 4.3). They are DefuserBot, ScannerBot and InspectorBot. They have basically the same actuator to drive two wheels for navigation. Also, they are equipped with communication systems to access the server: Satellite. However, their sensors are different from each other. The goal was to create a group of heterogeneous robots such that to perform some tasks would require cooperation.

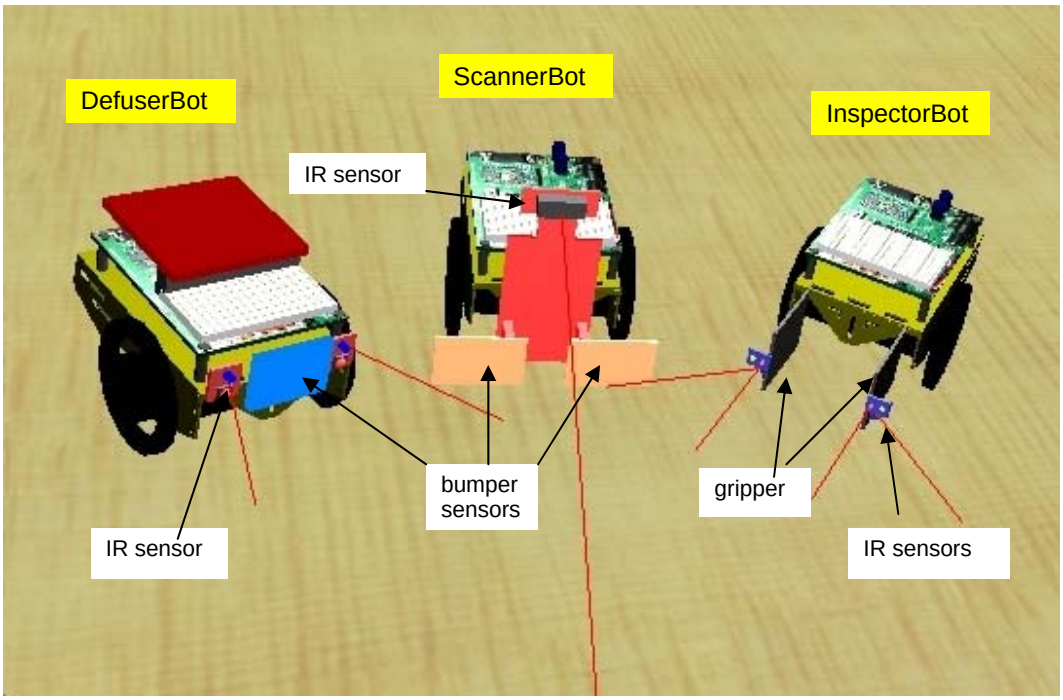


Figure 4.3: DefuserBot, ScannerBot, InspectorBot

- **DefuserBot**

DefuserBot depicted in Figure 4.3 is equipped with two short range infra red sensors and one bumper sensor to detect objects at the front of the robot.

- **ScannerBot**

ScannerBot shown in Figure 4.3 has two bumper sensors at the front. It also has a long range infra red sensor (IR) that can measure distance from the object. This IR sensor has far more coverage than the IR sensor of the DefuserBot.

- **InspectorBot**

InspectorBot has a mechanical gripper at the front. At the gripper front tip, there are four short range infra red sensors. In Figure 4.3 there are four red rays representing the IR sensor directions and the ranges that they can detect.

4.1.2.2 Objects

There are three types of objects other than robot itself that robots can detect with their built-in sensors. Those are world boundaries, dummies and bombs. Figure 4.4 shows screen shot of a dummy and a bomb.

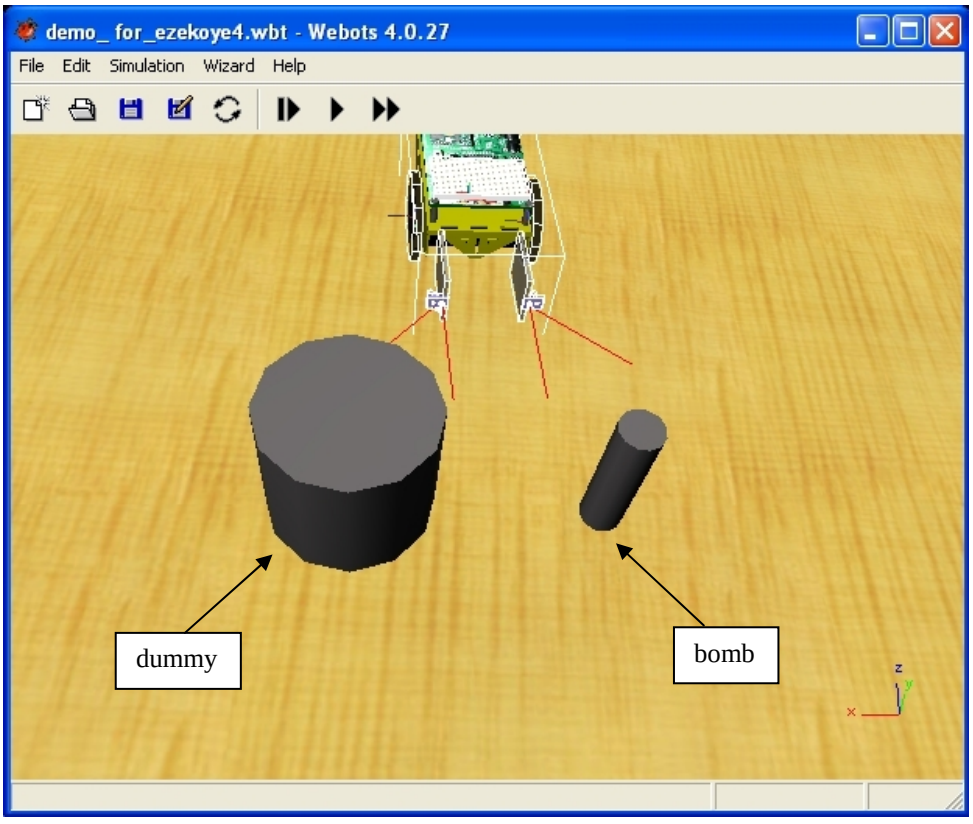


Figure 4.4: Screen shot of a dummy and a bomb.

4.1.2.3 Supervisor

Supervisor is a Webots node that enables to keep track of any solid nodes such as robots in the Webots environment. Therefore, robot positions can be monitored continuously. This Supervisor is controlled by matlab programs in the Satellite PC so that it can run a local positioning system server. Supervisor can also move or rotate any object in the scene which is essential for learning process.

4.1.3 Communication

There are two kinds of communication involved in this simulation. The first one is communication among PC's to exchange information about the current situation, sensor information or their knowledge. The second one is to interface two different software packages which are Webots and Matlab.

There is a common standard communication mechanism called *socket* provided by programming environments of the respective entities in their own style. Therefore, processes can seamlessly exchange their messages via this interface whether they are located in the same machine or distributed over the network, regardless of their operating systems and programming languages.

4.1.3.1 Socket Interface

Socket is a name given to the package of subroutines that provide access to TCP/IP (Transmission Control Protocol/Internet Protocol) on the system. It is a network programming interface and a collection of library functions that request TCP/IP operations of the underlying operating system via system call mechanisms. Figure 4.5 represents the sequence and architecture for PC communication between the server and client. In the communication, one process plays a role of server that invokes *bind, listen*, and *accept* calls in addition to the common calls of *socket, close*, and data exchange functions. Oppositely, the client process invokes *connect* system call to establish a reliable connection to the server process. In the data exchange part, each process sends or receives data via corresponding *read* and *write* calls.

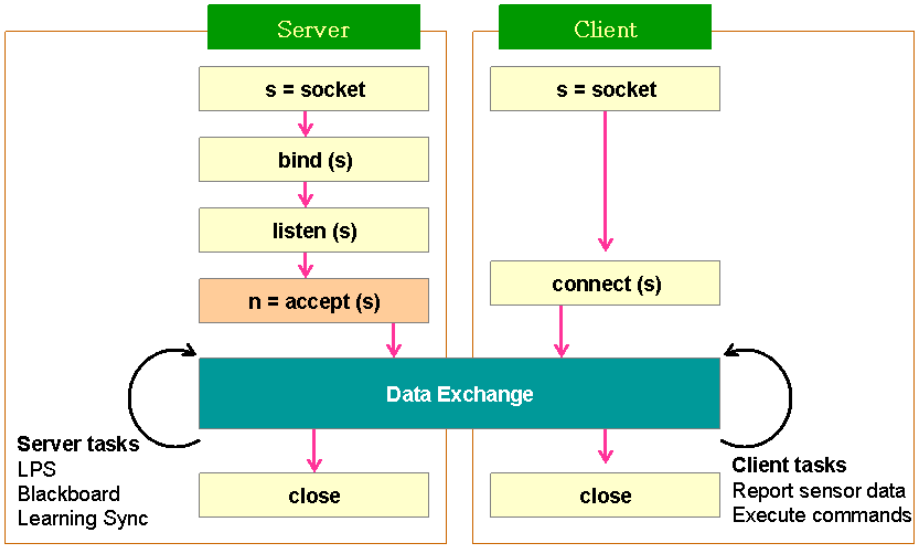


Figure 4.5: Server/Client communication between PC's

4.1.3.2 TCP/IP Communication among PC's

There are three PC's involved in this research for three robot communication. Each PC runs Windows XP as an operating system and Matlab as a primary platform for numerical calculation. Besides the command-line instructions, Matlab provides an *external programming language extension* including C/C++ or Java. A new Matlab command can be implemented by coding a C source program and then compiling to generate a DLL (dynamic link library) using MEX utility. After all, the communication functions can be developed using socket API and then called from the Matlab command line.

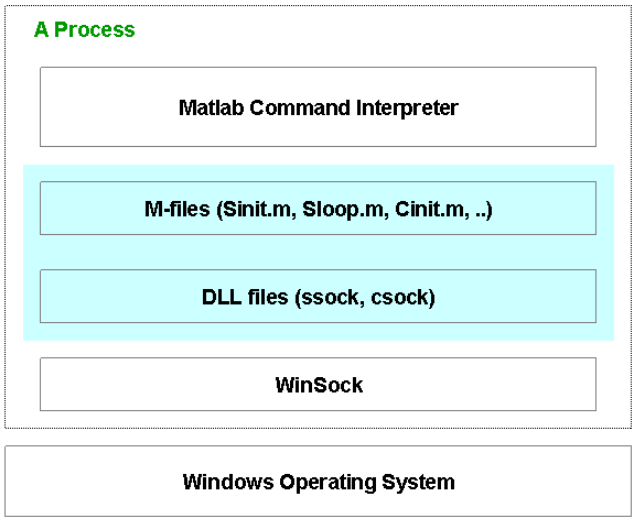


Figure 4.6: Components for Server/Client communication between PC's

Shaded components are realized for this research.
Ssock.dll, csock.dll : complied with Matlab mex utility

With socket, diverse communication topologies are possible, for example, one-to-one architecture or one-to-many architecture as shown in Figure 4.7. One-to-one architecture needs connections between every pair of processes, so its scalability is much limited. On the other hand, one-to-many communication architecture needs only (n-1) connections, where n is the number of processes. Correspondingly, this model is used for this research to accommodate blackboard described in previous chapter and for the scalability toward many robots. There is one process relaying communication to the rest. In terms of socket primitive, this process will function as a socket server, and the rest of them function as socket

clients in the connection setup procedure. As the server has to listen to multiple clients, it blocks on the *select* function to switch among the three different clients. Figure 4.6 shows components realized for the PC-to-PC's communication. And they are for the sequences as shown in the Figure 4.5.

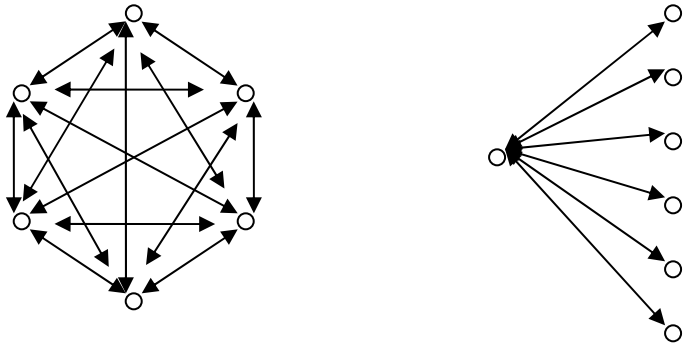


Figure 4.7: Topology for sever and client connection

Finally, Matlab uses column-major matrix with additional information fields to represent data structure, which is so different from that of C language. Therefore, the DLL's developed in this research execute some conversion when they are called and return to the caller. It is assumed that each dimension of the data matrix to communicate with is known in priori.

4.1.3.3 TCP/IP Communication between Matlab and Webots

The TCP/IP communication between Matlab and Webots is similar to the communication among PC's as described in the previous section. The difference is that Webots is fixed as a server and Matlab is fixed as a client and only two parties are involved in this communication. After Webots side first creates a socket server and binds to the port, it listens for any connection and accepts a connection request from the Matlab side. Another difference is that since the Webots software is implemented as a form of super loop, one process without multithreading, the blocking call in the loop can not be used. Therefore, the arrival of data is checked periodically via *select* function and then the data is exchanged asynchronously. Figure 4.8 is the diagrams for the communication protocol.

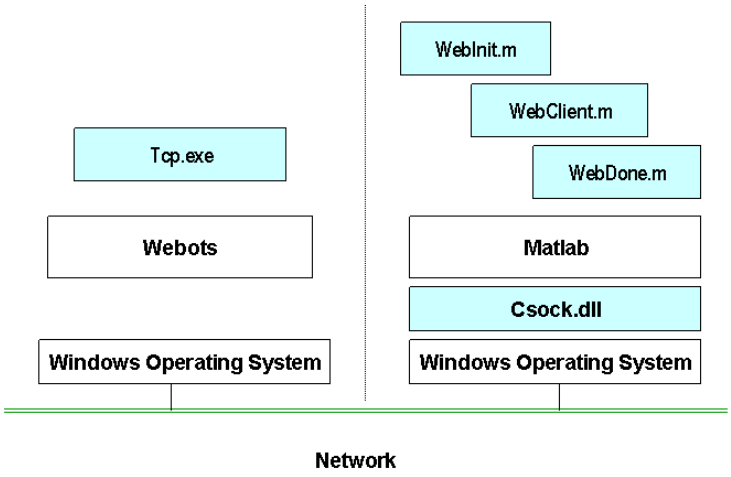


Figure 4.8: Components for Matlba and Webots communication

Shaded components are realized for this research.

Tcp.exe : compiled with visual C++ 6.0
Csock.dll : complied with Matlab mex utility

4.2 EXPERIMENT SETUP

4.2.1 Overview

To demonstrate the proposed AIS algorithm for the MRS, a hardware environment has been set up as depicted in Figures 4.9 and 4.10. It is very similar in structure to the computer simulation environment so as to minimize the transition effort from the simulation to the experiment. There are four computers (named: Satellite, DefuserBot, InspectorBot and ScannerBot) connected to a local area network via Ethernet in this setup.

There is an arena with walls for robots to navigate as in Figure 4.10. Each robot is controlled independently by a PC connected via radio frequency (RF) wireless communication. There is an embedded control system in each robot which uses a BASIC Stamp[20] and programmed in BASIC language. TCP/IP communication among PC's (which essentially represent individual robots) is identical to the simulation case.

The Satellite is achieved with the aide of a bird's eye view of the field (containing the position of robot) obtained via an overhead camera. Image data is then transmitted back to the PC and processed by LabVIEW. LabVIEW extracts necessary information and passes the data to Matlab via an ActiveX server application. Matlab performs the control algorithm calculations and sends

[20] http://www.parallax.com/html_pages/products/basicstamps/basic_stamps.asp

instructions via wireless RF communication to the robot's servos to move accordingly. Robots send sensory information back to the PC's and vice versa.

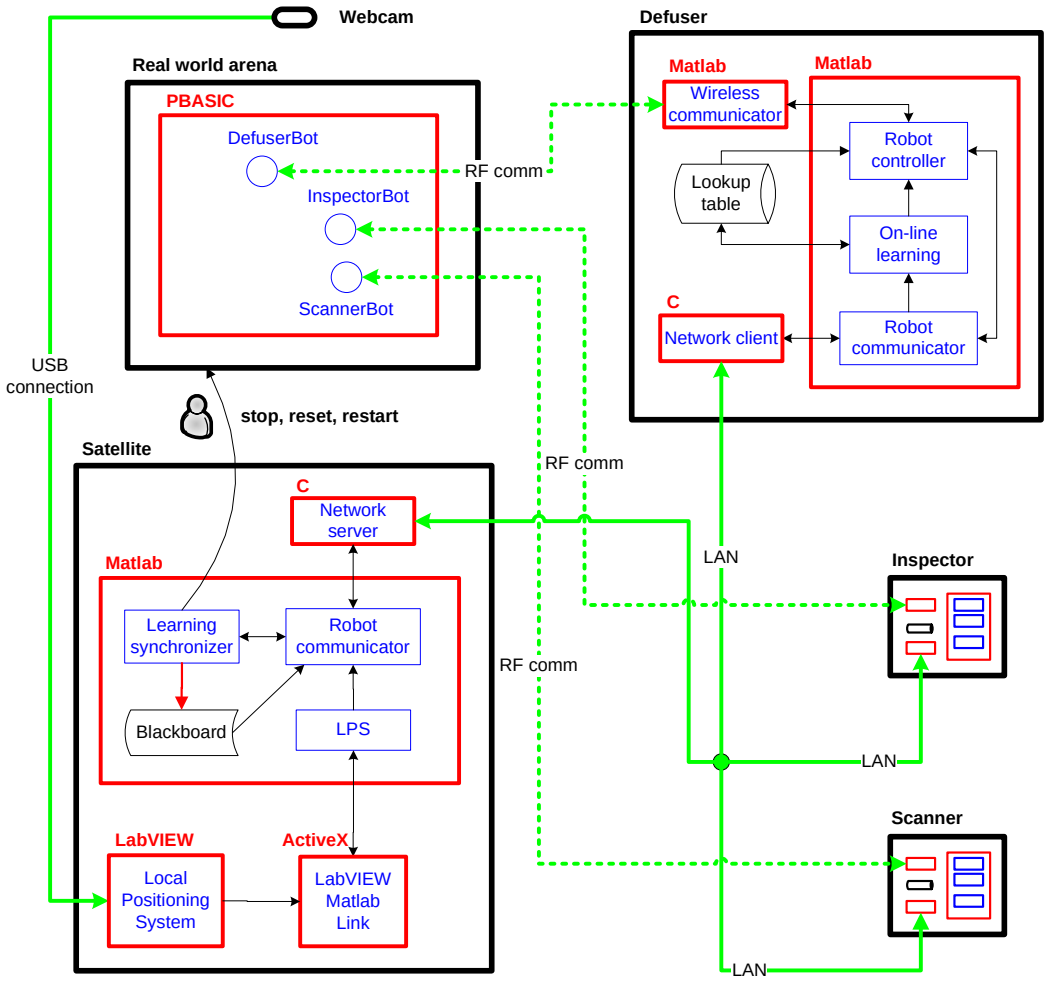


Figure 4.9: Experimental setup block diagram

The PC for DefuseBot is exploded to show its internal details. All robots are simillar in terms of blocks. They differ in their functionality.

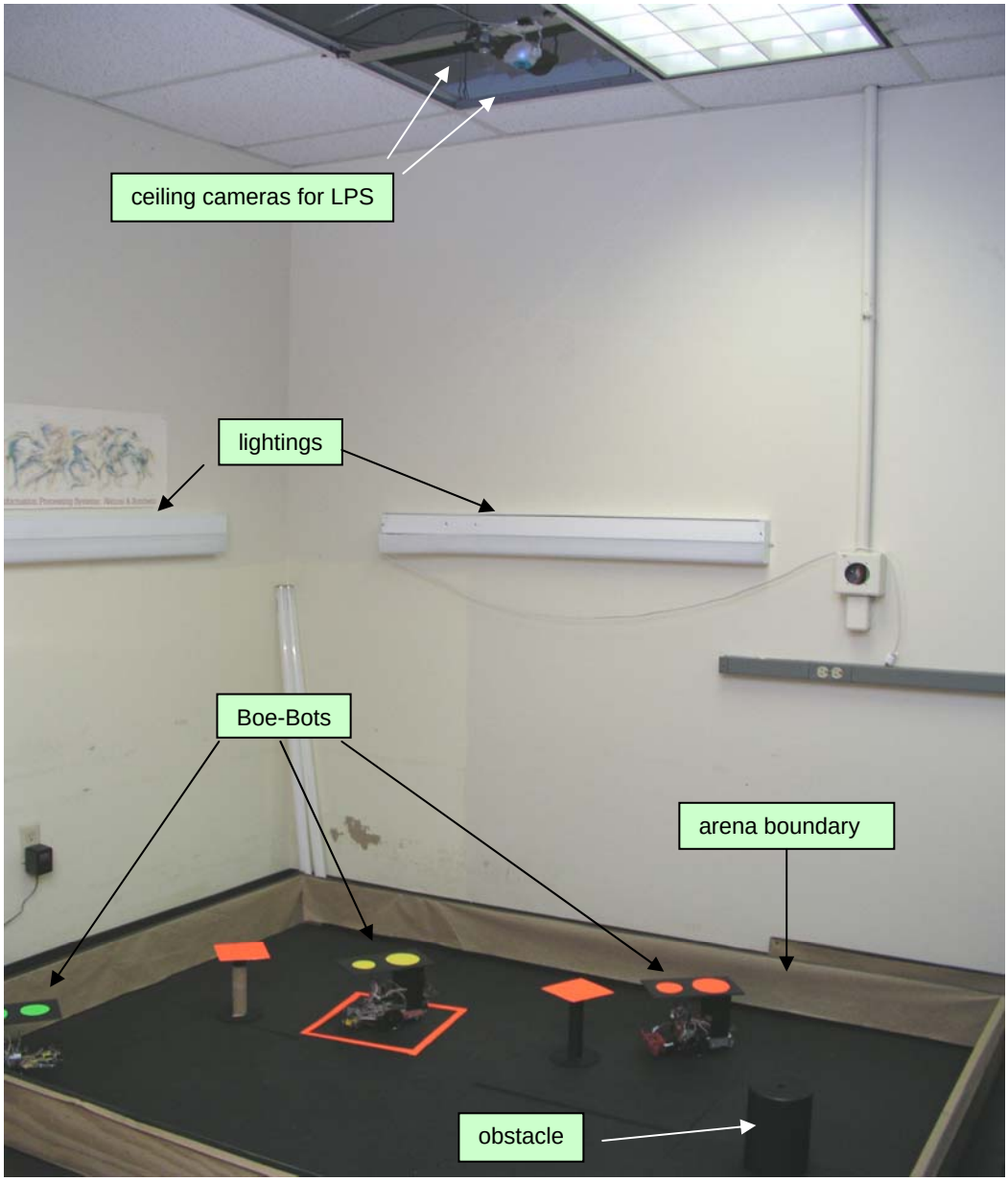


Figure 4.10: Experimental setup side view (NERDLab)

There are Boe-Bots and obstacle inside the arena boundary. Ceiling cameras are installed to pin-point the positions of robots.

- **Boe-Bot**

The robots used in this experiment are Board of Education Robot (*Boe-Bot)* a simple mobile robot kit developed by Parallax Inc.[21], with a BASIC Stamp microcontroller as its processing unit (Figure 4.11). The base model comes with the *Board of Education* (BOE) Rev. C carrier board, the BASIC Stamp 2 (BS2) microcontroller, and two servomotors. A BASIC Stamp microcontroller is a single-board computer that runs the Parallax PBASIC language interpreter in its microcontroller. The developer's code is stored in an EEPROM, which can also be used for data storage. The BS2 is programmed using Parallax BASIC (PBASIC). There is a DB-9 serial port that allows the microcontroller to communicate with a computer. The BASIC Stamp 2P (Figure 4.12) microcontroller module is an extremely fast chip (20 MHz, 12000 instructions/sec) that allows the Boe-Bot to perform relatively complex decision-making capabilities and this is an upgrade from the previously used BASIC Stamp 2 (20 MHz, 4000 instructions/sec).

[21] http://www.parallax.com

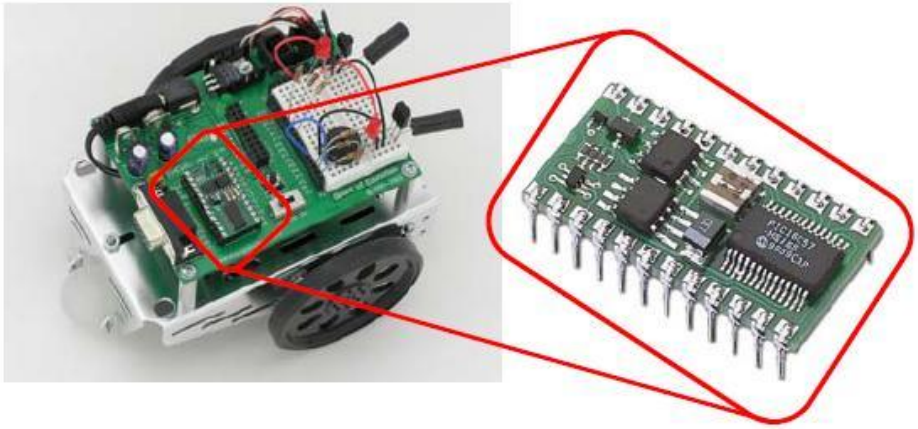
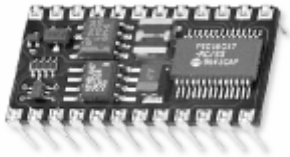


Figure 4.11: Parallax Boe-Bot and BASIC Stamp 2[22]

[22] http://www.parallax.com



BS2

Microcontroller	PIC16C57 SMD
Clock	20 MHz
EEPROM	2K Bytes
Length of program	500 Lines PBASIC Code
RAM (Variable)	6 I/O, 26 Variable
Input/Outputs	16
Output current(Source/Sink)	20 mA / 25 mA
Current consumption	7 mA Run, 50 µA Sleep
PC Interface	Serial Port
Package	24-Pin DIP Module (green PCB)
Dimensions	30 mm L x 16 mm W x 9 mm H

BS2p

Microcontroller	Ubicom SX48AC
Clock	20 MHz Turbo
EEPROM	8 x 2K Bytes
Length of program	4000 Lines PBASIC Code
RAM (Variable)	12 I/O, 26 Variable
Input/Outputs	32 Bytes (6 for I/O and 26 for Variables) plus 32 Byte Scratch Pad RAM
Output current(Source/Sink)	30 mA / 30 mA
Current consumption	40 mA Run, 60 µA Sleep
PC Interface	Serial Port
Package	24-Pin or 40-Pin DIP Module (gold PCB)
Dimensions	24-Pin: 30 L x 16 W x 9 H (mm) 40-Pin: 53 L x 16 W x 9 H (mm)

Figure 4.12: Parallax BASIC Stamp2 and BASIC Stamp2P[23]

[23] http://www.parallax.com

4.2.2 Entities

4.2.2.1 Robots

There are three different robots built with Boe-Bots for the purposes of this doctoral research (Figure 4.13). They are named as DefuserBot, ScannerBot and InspectorBot. They have basically the same actuators to drive two wheels for navigation. There is a radio RF transceiver for the wireless communication between each robot and corresponding host PC. Also, the PCs are equipped with communication systems to access the Satellite. However, their sensors are different from each other depending on their roles of their mission. The goal is to create a group of heterogeneous robots such that to perform some tasks would require cooperation.

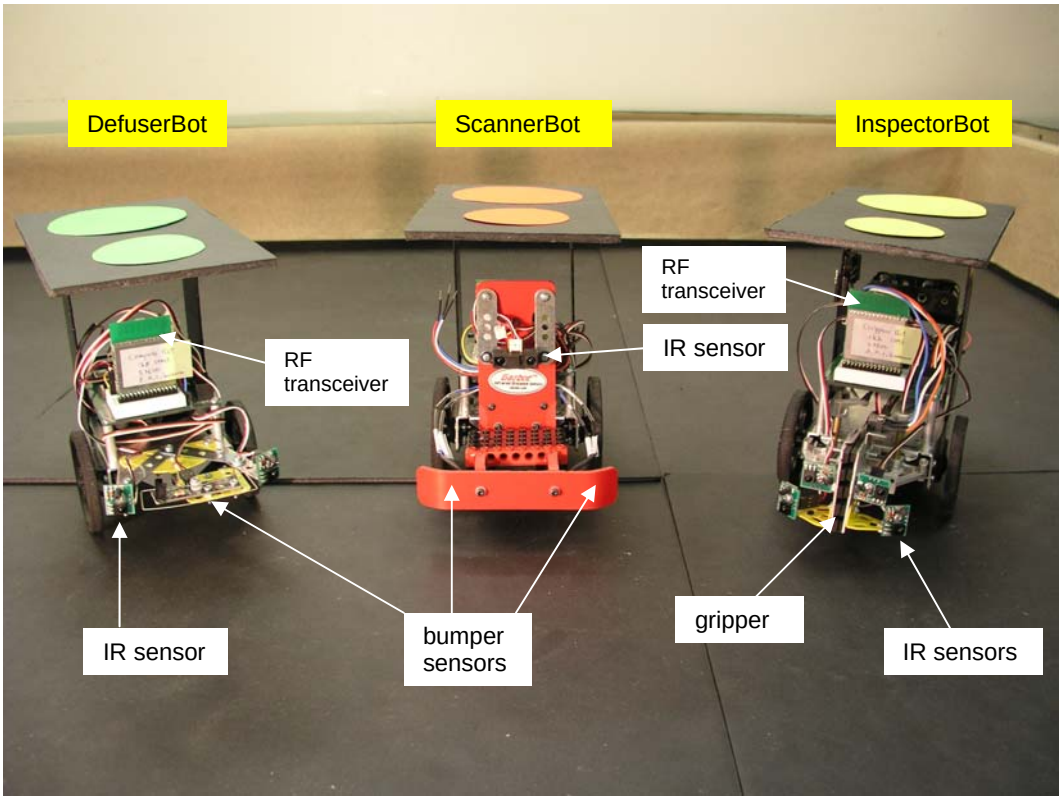


Figure 4.13: DefuserBot, ScannerBot, InspectorBot

The logic loop of PBASIC programs, as shown in Figure 4.14 for servo actuation downloaded onto the BASIC Stamp microcontroller, starts with the servos being at an idle state. Next, the microcontroller invokes for input data from the pin which is assigned to communicate with the wireless device. If no actuation command is sent by Matlab at that moment, the servos will remain idle. Otherwise, the input data will be processed to determine rotation speed and direction. Servo rotation will take place continuously until a new input is received

by the RF receiver. Next, the Stamp reports sensor data to the matlab running in the PC.

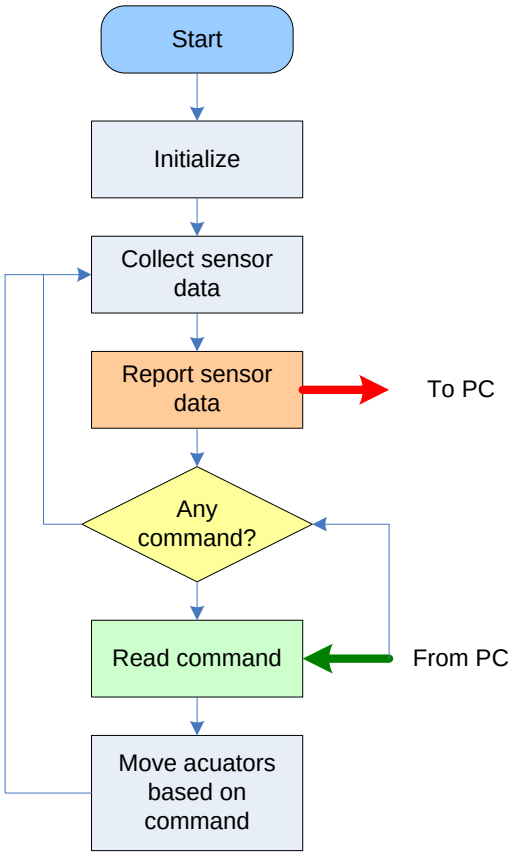


Figure 4.14: Logic loop of Boe-Bots controller

- **DefuserBot**

This robot has two IR sensors for the front right and front left direction obstacle detection purpose. The IR detectors are 15.8 mm x 18.2 mm boards that

incorporate both an IR LED and a 40 kHz IR receiver. The detectors work by having the LED emit light, and if an object is close enough, the light will reflect back to the detector. When the IR is not detecting an object it reads a value of 1, and when it is detecting an object it reads a value of 0. Regardless, the IR detectors are very good at detecting objects several inches in front of the robot, allowing it to avoid them quickly.

In the front middle part of the robot, a bumper sensor is installed to cover the center area that IR sensors can not detect. And this bumper sensor emulates defuse of a bomb. It was modified from the Twinkle Toes Bumper Sensor by Parallax, Inc.[24] for this research.

- **ScannerBot**

This robot has two bumper sensors and one distance sensor, and both of them are facing forward. This distance sensor is used for the obstacle avoidance during navigation and the scanning the arena for bomb search. It is Sharp GP2D12[25], an analog distance sensor, that uses infrared to detect an object between 10 cm and 80 cm away. It provides a non-linear voltage output in relation to the distance an object is from the sensor.

The GP2D12 is wired to an ADC0831[26], 8-bit analog to digital converter, as shown in the circuit of Figure 4.15. Two resistors of 217 Ohm connected to the Vref pin on the ADC0831 are a voltage divider to set the reference voltage to 2.55

[24] http://www.parallax.com/detail.asp?product_id=27312
[25] http://www.acroname.com/robotics/parts/SharpGP2D12-15.pdf
[26] http://robotics.me.jhu.edu/~llw/courses/me530420/lab/adc0831.pdf

volts. On the ADC0831 this will give a value of 0 to 255 for an input voltage of 0 to 2.55 volts. Figure 4.16 shows calibration result of the distance sensor by measuring the output of the GP2D12 at given fixed distances, in centimeters.

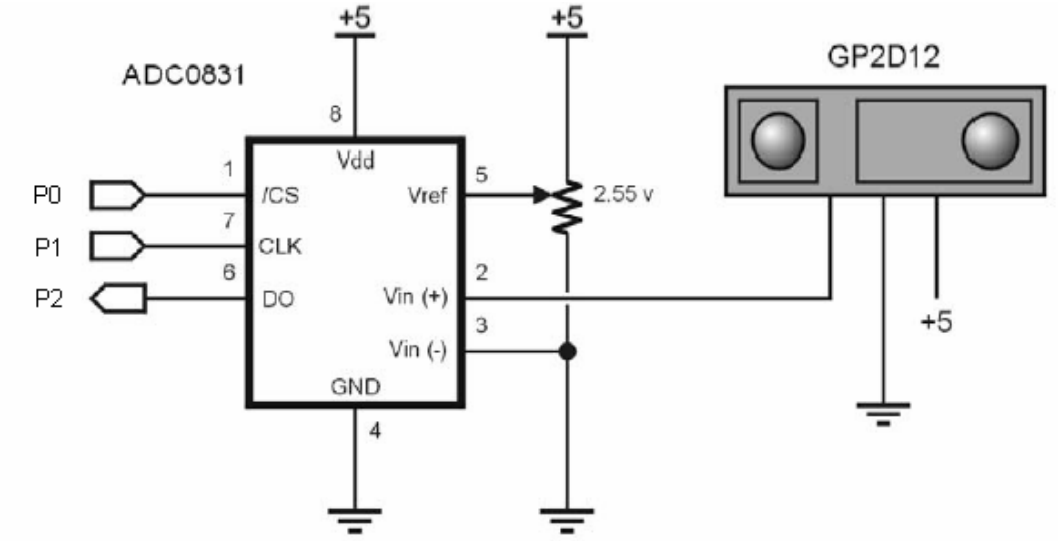


Figure 4.15: Wiring diagram of ADC0831 and GP2D12 [27]

[27] http://www.parallax.com/dl/docs/prod/acc/SharpGP2D12Snrs.pdf

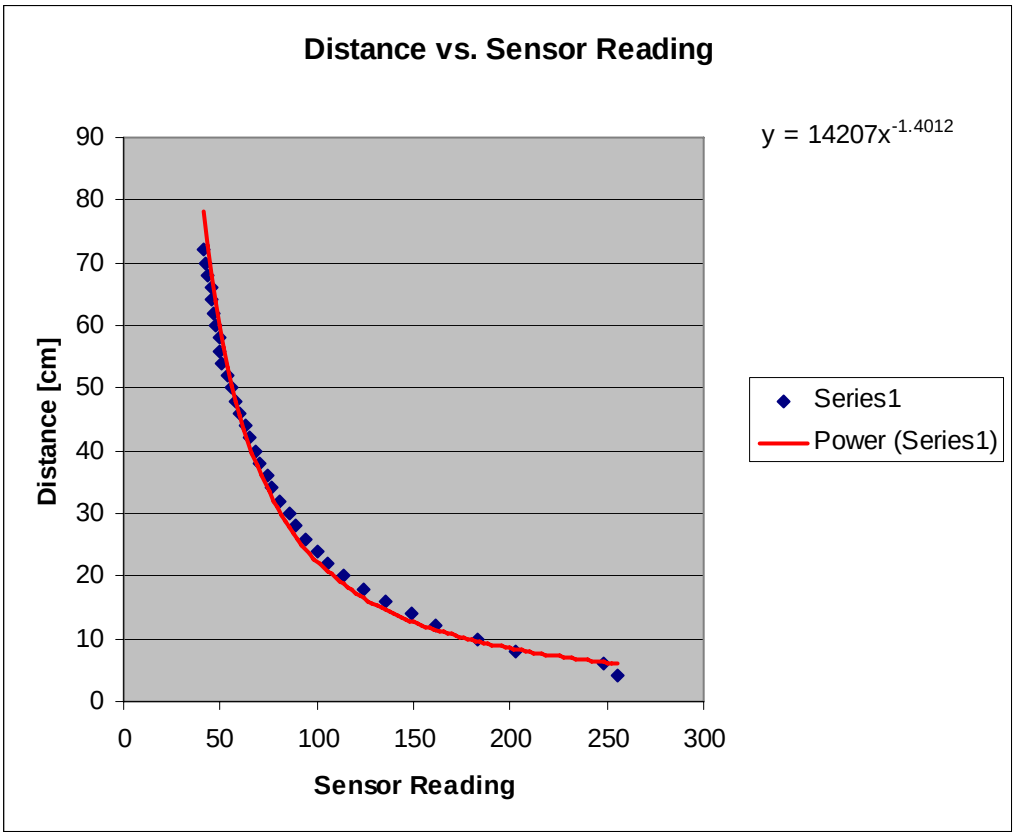


Figure 4.16: Distance sensor reading vs. Distance

Finally, if an object in front of the robot is out of the distance sensor's field of view, two bumper sensors in front of the robot can detect it whenever it is bumped. They are originally from Gazbot[28] and modified for this research.

- **InspectorBot**

[28] http://www.gazbot.com/products.asp

This robot's gripper, produced by Parallax, Inc., allows the Boe-Bot to pick up and move objects. The assembly consists of two arms controlled by a standard servomotor to allow the gripper to hold its open or closed position without continuously needing a power supply.

It was modified for this research by adding two more IR sensors and a touch sensor to fulfill the task of inspection of two different types of objects. A total of four IR sensors were installed on the gripper. Front-left and front-right IR detectors detect objects in front of the robot. Side-left and side-right IR detectors detect objects at the side of the robot. And they operate exactly as the IR detectors on the DefuserBot. Also, a touch sensor to detect whether the gripper is closed or not is additionally installed.

4.2.2.2 Objects

There are three types of objects other than robot itself that robots can detect with their built-in sensors. Those are world boundaries, dummies and bombs. Figure 4.17 shows a dummy and a bomb.

The arena boundary walls are made with plywood and top areas are painted in dull black for better image processing purposes. Inner sides are covered with gray papers for better robot IR sensor object sensing purposes.

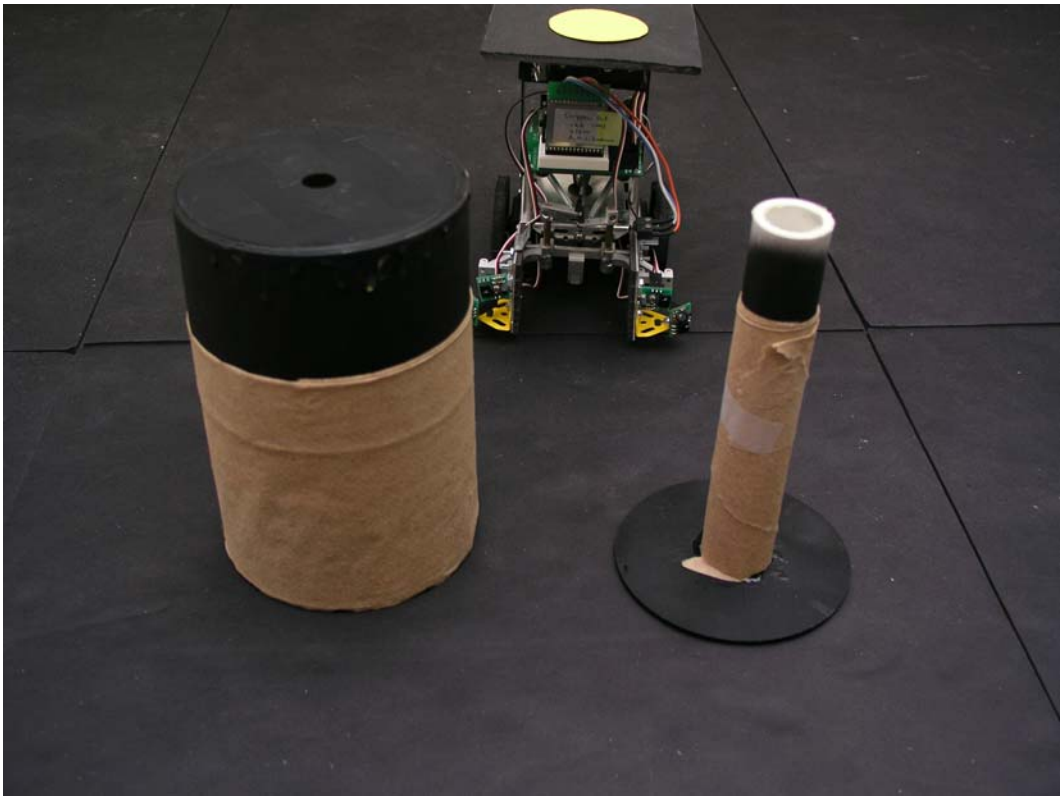


Figure 4.17: A dummy and a bomb (experiment)

4.2.2.3 Webcam

The overhead camera (Figure 4.18), installed 10 ft above ground, is connected to the PC through the Universal Serial Bus (USB) port. The USB port provides serial bus standard for device connection from one to another. The overhead camera used in the experiment is the *Creative NX Pro webcam*[29].

[29] http://www.creative.com/welcome.asp?bypass=1

Figure 4.18: Creative NX Pro webcam

4.2.3 Local Positioning System (LPS)

The visual data is acquired by the data acquisition software, LabVIEW, and processed by image filtering LabVIEW add-on software named IMAQ Vision[30], to determine the position of the individual robots. By communicating via ActiveX, LabVIEW passes the state information to Matlab[31] upon request.

To determine the relative position of robots in real-time and have the data ready for input to the control system, the following tasks need to be accomplished:

- Image acquisition: capture continuous streams of images using compatible cameras.

[30] http://www.ni.com/http://www.ni.com/
[31] http://www.mathworks.com/

- Image processing: filter raw images, cancel noise, and extract useful information.
- The system must be calibrated with the robot scale and the accuracy determined.

4.2.3.1 Image Acquisition

Vision of the field is obtained via overhead cameras. The camera was selected for its sufficiently high video resolution and suitable video format, necessary to ensure fine image quality captured by the camera. High frame rate of 15 fps is also needed to reduce time delay and ensure real-time data processing. Automatic exposure control and color balance options allows less filtering to be done by the image processing software, thus reducing computation time. Lastly, a 40^{o} field-of-view allows optimal capture of the 5' x 7' field below the camera.

Figure 4.19 describes the architecture of application for LPS. NI-IMAQ for USB Cameras is a free software driver for acquiring images from any DirectShow imaging device into LabVIEW. These devices include USB cameras, webcams, microscopes, scanners, and many consumer-grade imaging products. NI Vision Assistant, a software product by National Instruments, is a configurable, interactive prototyping application, which allows machine vision software development easier.

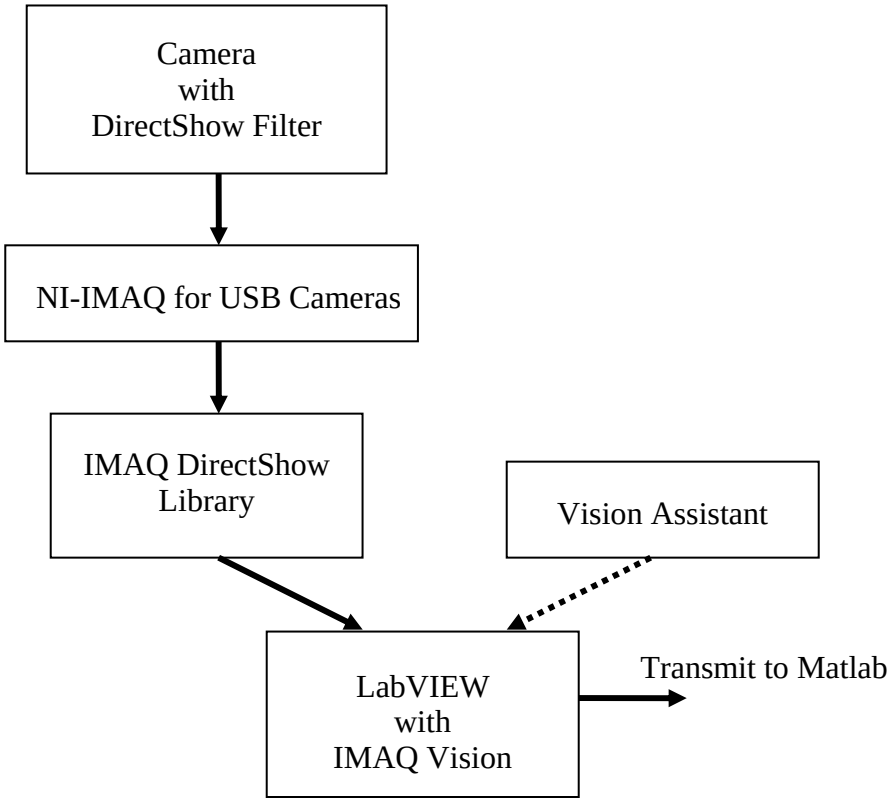


Figure 4.19: Architecture of applications for LPS

A camera with DirectShow Filter (USB webcam) is connected to the Satellite.`LabVIEW and add-on software IMAQ Vision can capture the camera images. Vision Assistant is used for off-line software development purpose.

4.2.3.2 Image Processing

NI Vision Assistant software is used to create a LabVIEW compatible script containing sequential steps to filter noise from the image and eventually determine the object's position within a desired accuracy. The main difficulty

encountered while programming the LabVIEW algorithm with Vision Assistant was the enormous number of variables that have to be taken into account. Many factors affect image quality and the robot's ability to distinguish objects in an image. Figure 4.20 shows a snapshot of the original image as captured by the camera. The primary image operation is to perform exposure control which applies brightness, contrast, and gamma correction to each color plane separately to remove initial noise from the raw image.

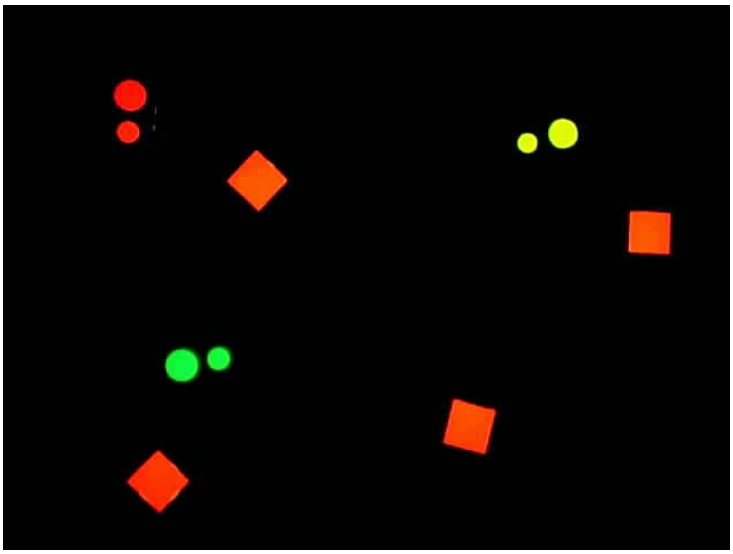


Figure 4.20: Snapshot of an original, untouched image, showing four different markers (paired shapes of different sizes) with different colors

Figure 4.21 below shows the transformation sequence of the raw image in Figure 4.20 to processed images as the image undergoes filtering steps by IMAQ Vision. Figure 4.22 shows the flow chart of the image process.

.

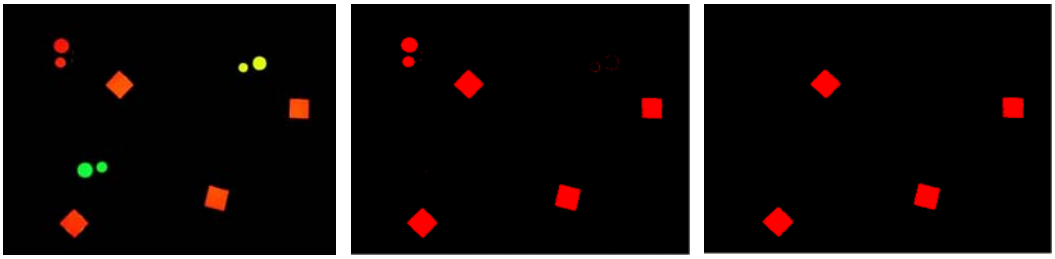


Figure 4.21: Filtration steps to remove noise and isolate red objects in the image and filter circular objects

Shown from left to right are: original image after noise removal; yellow and green objects filtered out; circular shape filtered out.

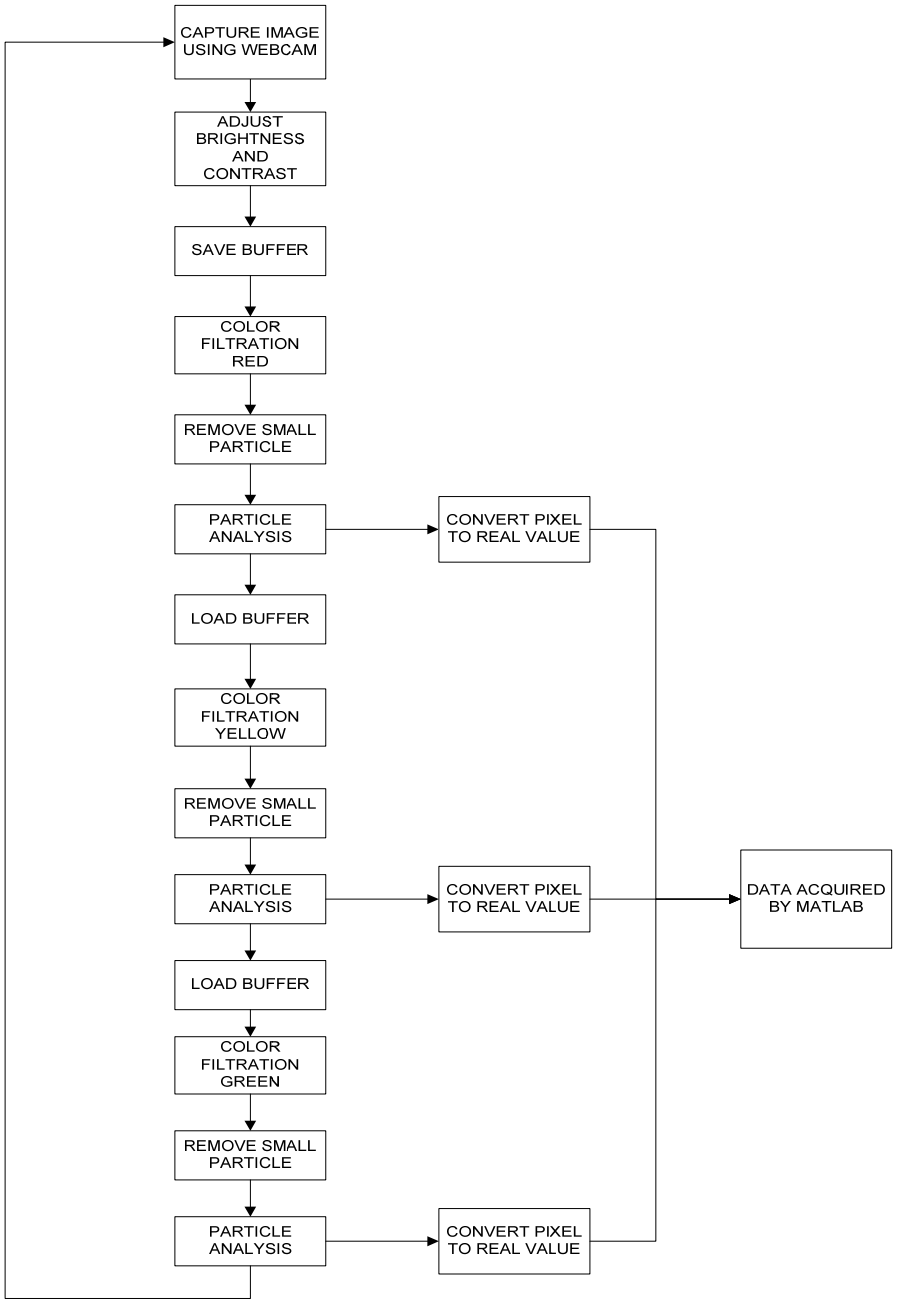


Figure 4.22: Filtration steps to remove noise and isolate red objects in the image and filter circular objects

4.2.3.3 Data Interpretation for Coordinate Determination

Using *IMAQ Particle Analysis* VI[32], LabVIEW is capable of determining the pixel coordinate (x,y) of objects seen through the camera. When an overhead camera is used, the robot is identified by placing a marker (two circular colored papers of different sizes) on the robot. Particle analysis tool can sort binary image by its size. Bigger size is number one and smaller size is number 2. The smaller paper is situated "in front" of the larger one, allowing to determine the robot's direction; size differentiation is necessary to allow determination of robot orientation.

From the pixel coordinate of each object outputted by LabVIEW, the real-world coordinates of robots are as follows:

$$x_{robot,real} = \frac{\frac{1}{2}\left[x_{small,pixel} + x_{large,pixel} \right]}{k_{Cx}} \qquad \text{eq (4.1)}$$

$$y_{robot,real} = \frac{\frac{1}{2}\left[y_{small,pixel} + y_{large,pixel} \right]}{k_{Cy}} \qquad \text{eq (4.2)}$$

where k_C is the calibration coefficient which indicates the number of pixels that corresponds to the length in feet in real world measurement. For the

[32] http://zone.ni.com/devzone/cda/tut/p/id/3169

overhead camera, k_{Cx} *for the x direction = 3.04 pixels/ft and* k_{Cy} *for the y direction = 3.14 pixels/feet* which were deduced by calibration.

Also, the heading of a robot can be calculated as follows:

$$\theta = a\tan(x / y) \qquad \text{eq(4.3)}$$

where θ is the heading in radians.

4.2.4 Communication

There are three kinds of communication involved in the experimental setup. The first is communication among PC's to exchange information about the current situation, sensor information or their knowledge. The second is communication between a Boe-Bot and a corresponding PC. The one is communication between Matlab and LabVIEW.

4.2.4.1 TCP/IP Communication among PC's

The communication protocol among PC's is identical to the case for the computer simulation described in Section 4.1.3.1.

4.2.4.2 Serial Communication between Boe-Bot and PC

Serial communication (RS-232C)[33] is the most common low level protocol for communicating between two or more devices. As the name suggests, the serial port sends and receives bytes of information in a serial fashion: one bit at a time. These bytes are transmitted using either a binary format or a text (ASCII) format. The serial data format includes one start bit, between five and eight data bits, and one stop bit. A parity bit and an additional stop bit might be included in the format as well. Figure 4.23 illustrates the serial data format.

Figure 4.23: Serial data format

There are two pins for bi-directional data transfer (Tx, Rx) and the other pins for control of the communication flow at a DB9 serial connector. Because a sender and receiver can't always process data at the same rate, some technique of negotiating when to start and stop transmission is required. One method relies on the serial port hardware; the other is implemented in software. Both methods are types of flow control. The hardware flow control uses two of the serial port lines to control data transmission.

[33] http://www.taltech.com/TALtech_web/resources/intro-sc.html

For the wireless communication between the Matlab of a PC and the on-board PBASIC program of a Boe-Bot, a set of *SureLink 900 MHz RF Modules* and *QuickLink Demo Board* all by Parallax Inc. are used. Figure 4.24 shows a block diagram of wireless communication between Boe-Bot and PC. One RF module is directly connected to the Basic Stamp on the robot side, and another RF transceiver module is slotted into the QuickLink Demo Board unit, which is connected to the PC via serial cable.

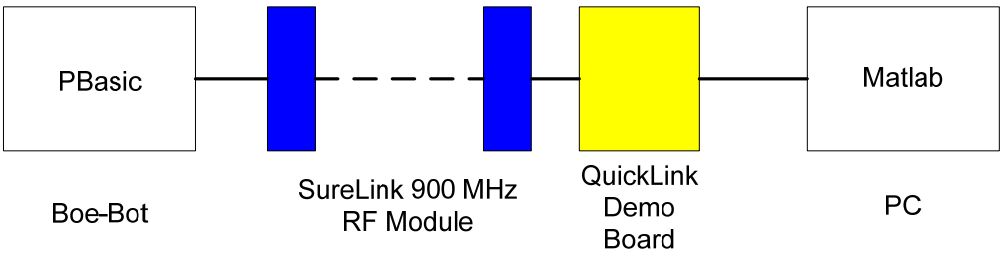


Figure 4.24: Communication between Boe-Bot and PC

Serial Baud Rate (data communication speed between the SureLink RF module and host that can be Basic Stamp or PC) is ranged between 1200 Baud and 115k Baud. The RF Data Rate, the speed at which the RF data is transmitted between SureLink RF modules wirelessly, is from 48 kbps to 76.8kbps. Higher RF data rates will decrease the distance of operation, while lower RF data rates will increase it. The maximum distance is up to 1000 feet. For this research three pairs of wireless communication with different channels are built for each of the three independent robot navigations.

Many trial and error attempts have been made to fine tune the software to maximize the communication performance almost reaching hardware upper limits. However, difficulties with timing on the serial ports were one of the most frustrating aspects of developing stamp applications.[34] It is partly because of the intrinsic characteristics of this distributed robotic system where each device uses its own internal clock. Therefore, the communication protocol is asynchronous. Due to additional overhead in the BASIC Stamp, and the fact that the BASIC Stamp has no hardware receive buffer for serial communication, received data may sometimes be missed or garbled.

To optimize the serial communication, BASIC Stamp chips have been upgraded to the BASIC Stamp 2P for faster communication speed. Also lowering the baud rate, RF data rate, adding extra stop bits, not using formatters in the SERIN command, and using simple variables (not arrays) increased the chance that the BASIC Stamp can receive the data properly. A type of data flow control called hardware handshaking is used to prevent data loss during transmission. For this experiment, serial baud rate of 57.6 kbps, RF data rate of 76.8 kbps, eight data bits, no parity bit, and one stop bit has been used.

Figure 4.25 shows a schematic sketch of serial communication between two different software platforms.

[34] http://www.emesystems.com/BS2rs232.htm

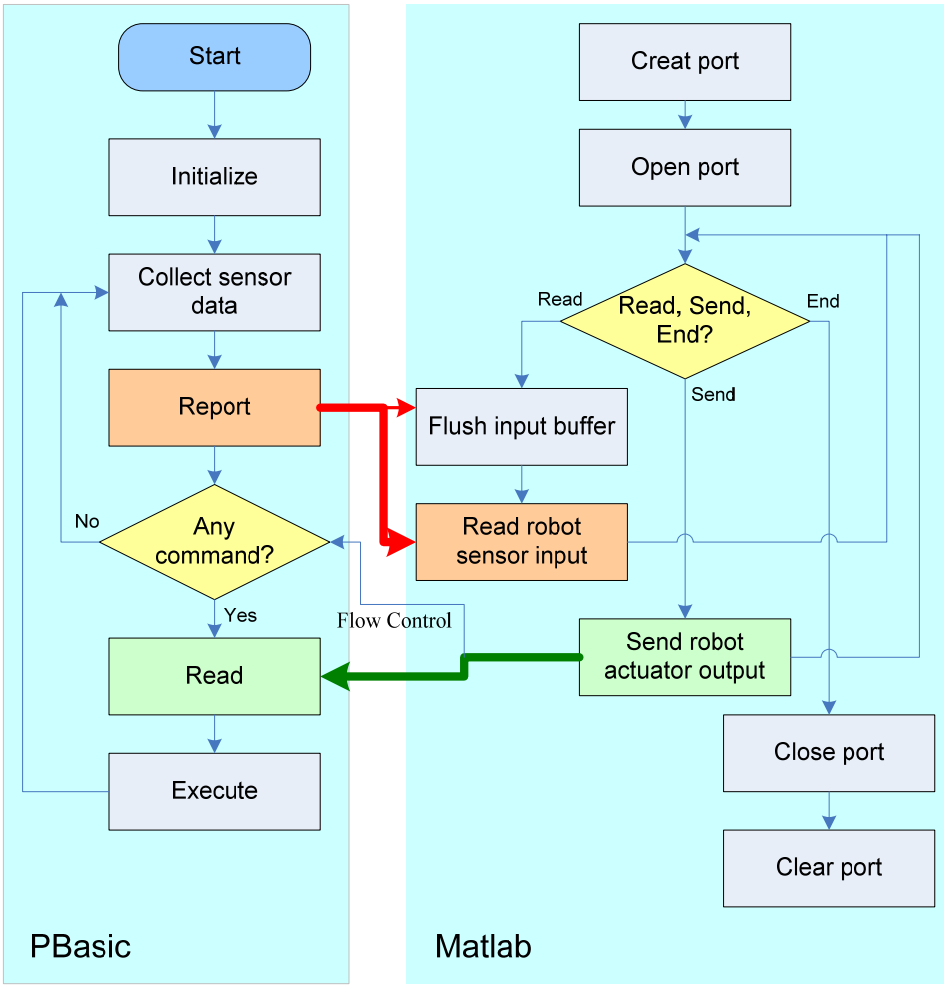


Figure 4.25: Serial communication flow chart

4.2.4.3 Communication between Matlab and LabVIEW via ActiveX

After each completion of image processing LabVIEW, *IMAQ Particle Analysis* outputs measurement values of requested parameters about the object,

such as *object pixel area*, *object dimension* (width and height), and *object coordinate* within the image. These parameter values are essential inputs to the control algorithm which runs in Matlab. For Matlab to receive this data from LabVIEW, ActiveX technology needs to be incorporated. ActiveX is a distributed object system and protocol used to manage compound documents and data transfer between applications, which is accomplished via *Object Linking and Embedding* (OLE).

LabVIEW software has integrated *ActiveX Automation* functionality that allows other programs to use and control the LabVIEW VI. Commands and data can be sent to different applications in a single format by means of invoking and getting and setting properties. Matlab can access LabVIEW VI through its *ActiveX Server*, whereby an *Application Object* exports the properties of LabVIEW [Johnson, 2004].

Chapter 5: Test Conditions, Results and Discussion

5.1 ROBOT MISSION DESCRIPTION

In order to validate the power of the AIS algorithm proposed in this thesis, a bomb disposal scenario was developed. The mission that has been simulated for the purposes of this doctoral research is for a group of heterogeneous robots to detect and dispose of bombs in a robot arena. The disposal unit consists of a team of three robots labeled by their expertise: ***DefuserBot***, ***ScannerBot*** and ***InspectorBot***. These robots will explore the environment and, inspired by the immune system, evolve from experiences creating a set of behaviors such that they can cooperate with each other to detect, inspect, defuse and dispose of the bombs.

Although *bomb disposal* is the "main" reason for existence, they have other individual goals such as the energy needed for operation and survival. Therefore, each robot in the group should meet the global (group) goals and the local (individual) goals at the same time. Each robot has a physical space designated as its *"home"* and in order to promote periodic visits, the "home energy" decreases with time and is replenished when visited by the owner robot. Also, each robot loses its own "robot energy" as time goes by and actions are performed.

To replenish the robot's energy, there is an energy source (*charging station*) that each robot can use to get charged one at a time. The robot is able to

transfer some of this energy to its home for home usage. Figure 5.1 shows a graphical representation of the arena and the different areas described above.

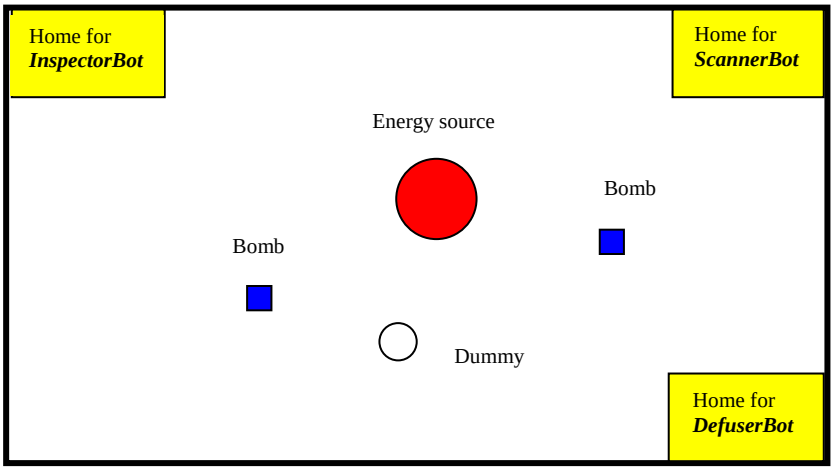


Figure 5.1: Mockup of the arena used by the robot unit

Shown are the important locations: each robot's home (robots need to visit regularly), charging station (energy source where robots get charged to survive and provide energy to their homes), bombs (that robots need to detect, inspect, defuse, and dispose), and a dummy (obstacle, diffused bomb or dummy). Not shown are the robots.

The ScannerBot uses its range finder sensor to scan the arena and locates the potential bombs. When it finds a bomb, it writes the location to the *blackboard*. The InspectorBot has an ability to discern a bomb from other objects of a similar shape, i.e., dummies. It can get information on objects found, which was left in the blackboard by the ScannerBot. The InspectorBot can also request a rescan job to the ScannerBot, if it is not a bomb after the inspection. The

InspectorBot posts the location of the bomb to-be-defused in the blackboard (for the DefuserBot to read and take action) when the (up to that moment unknown) object is identified as a bomb. Finally, the DefuserBot defuses and disposes the bombs listed in the blackboard. The DefuserBot can request a rescan job to the ScannerBot if it can not defuse it. This information flow is explained in detail later (Figure 5.18).

There are two sources of these rescan orders from the DefuserBot and the InspectorBot. One is from the scanning error. The other one is due to the movement of the bomb itself. The bombs are designed to absorb the impact energy during the collision against the robots to protect the robot hardware. Therefore, this internal feedback loops between robots work as a mutual error correction mechanism for the robot mission. Figure 5.2 shows the information flow for the bomb disposal mission. It also shows the flow of energy among the different actors.

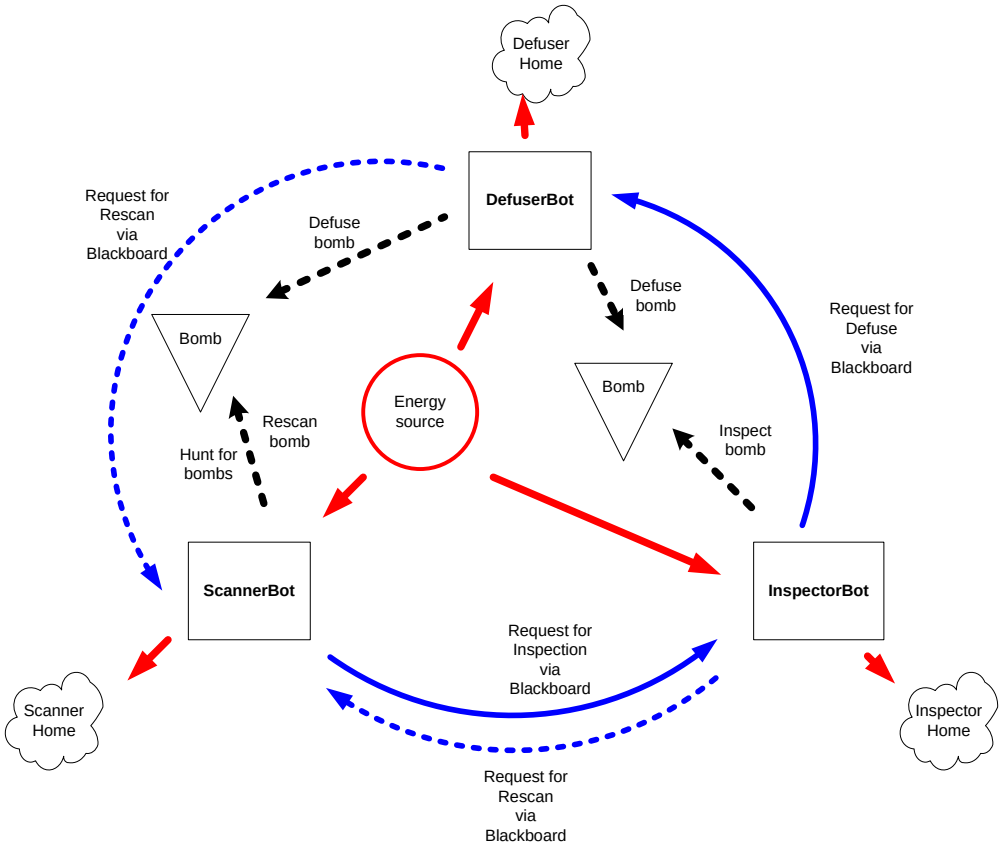


Figure 5.2: Energy and information flow of the robot group

Messages (information) are shown as blue lines and energy flow as red lines. The ScannerBot searches for objects and marks an entry in the blackboard of the object's location and is listed as "unknown" (basically requesting the InspectorBot to inspect the object – adds the object to the InspectorBot queue). The InspectorBot, after completing its current task, checks the blackboard for unidentified objects, goes to their location and inspects the object. If the object is a bomb, it updates the blackboard labeling the object as "bomb" (this is basically sending a message to the DefuserBot to defuse the bomb). The DefuserBot, after done with its current task, looks in the blackboard for bombs-to-be-defused, goes to their location, and defuses them. Robots go to the Charging Station and take energy from this source. They use the energy to move and perform tasks. Each robot should go periodically to its home and give some of its energy to the home supply.

5.2 ROBOT LEARNING CONDITIONS

A simulation environment was developed using Webots software. The basic idea of the simulator was to reproduce the arena environment and simulate different scenarios, learning and evolution algorithms before deploying the robots in the real arena. This virtual world was also used as a teaching laboratory for the different robot brains to initialize their behaviors. Figure 5.3 shows a screen shot of the computer simulation of a typical bomb disposal mission and Figure 5.4 shows the experimental setup for a similar scenario. There are two different types of objects, bombs and dummies.

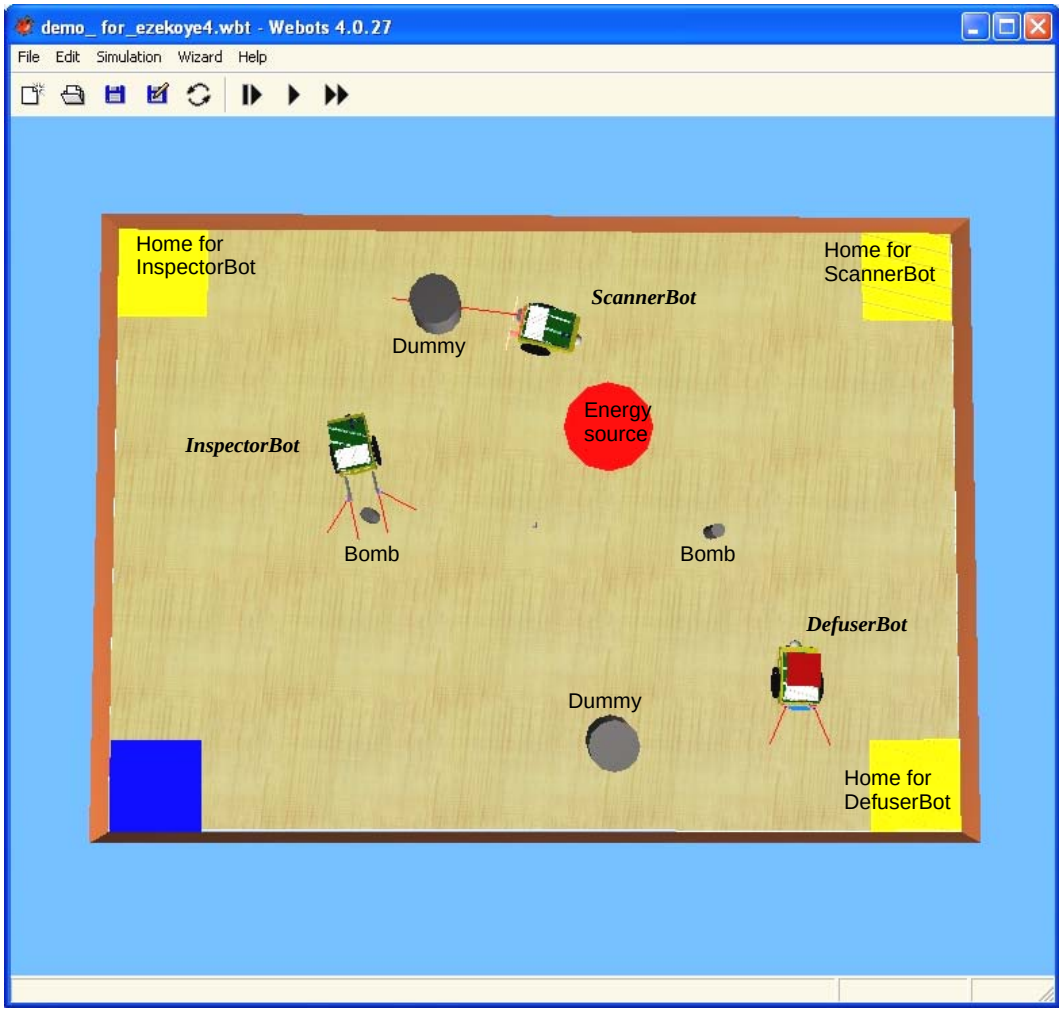


Figure 5.3: Computer simulation of a bomb disposal mission

The virtual world (developed using Webots® software) mimics the real experimental setup (see Figure 5.4). Avatars of the physical robots were created in the virtual arena. Simulated are also the different areas and objects found in the real arena.

Figure 5.4: Experimental setup showing a typical bomb disposal mission

The arena shows the different areas (Homes for each robot, charging station) and objects (bombs, dummy). A virtual world (developed using Webots® software, see Figure 5.3) was developed to mimic the real experimental setup.

5.2.1 Robot Basic Behaviors

In this section, basic behaviors (BBs) for the robots, which are essential for Behavior Based Robotics (BBR), are discussed. BBs for the experiment have the same functionalities as described in simulation case, but are tuned for the real

experiment case. These BBs are coded in a Matlab environment and tested in advance in the virtual world arena (VA).

Figure 5.5 shows the robot detection range for different entities. Locations farther than their range limit can not be detected. When robot battery level gets low, the sensor coverage is reduced. Therefore, each robot should maintain battery level to accomplish the mission.

Since the range that robot can detect is limited, each robot uses the *wander* behavior. Given the robot's mobility, by wandering, the robot is able to cover more terrain and effectively explore the whole physical arena (PA). The robots can go to any location in the arena using information of their location from the Satellite server (a LPS – Local Positioning System) and the target location if it is within robot detection range.

Robots can not differentiate between different objects such as bombs, dummies and wall boundaries with their built-in sensors. When the ScannerBot logs possible bomb locations, robots can then interpret them as locations of interest to them.

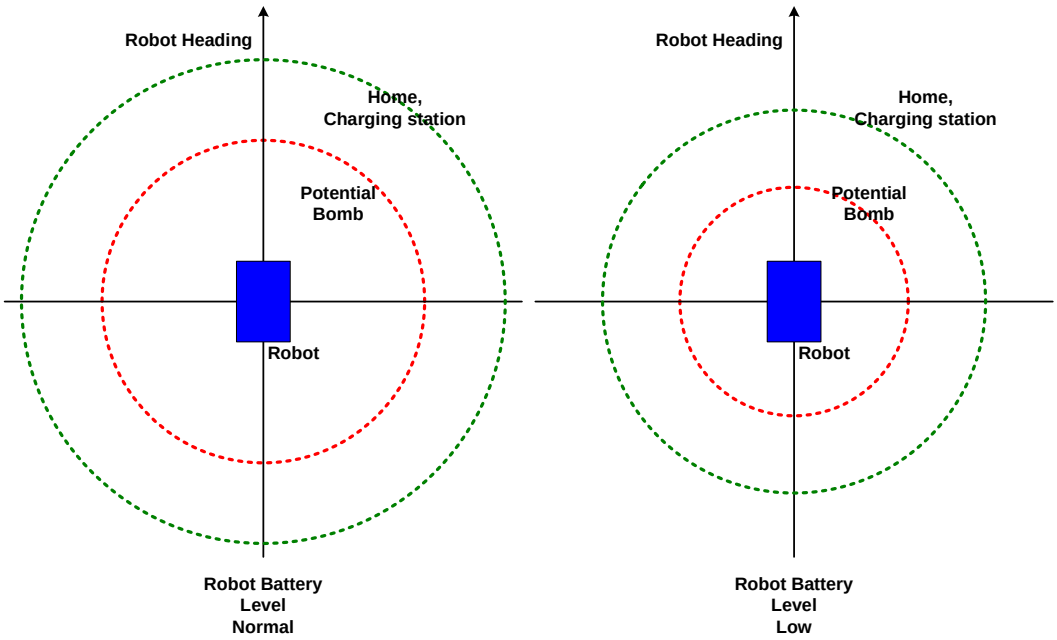


Figure 5.5: Robot detection range

It is assumed that the on-board sensors of the robots have limited range. And the range depends on robot battery level. Shown are two different ranges for a threat (bomb) and home and charging station. The difference in range is intended to be qualitatively proportional to the relative entity sizes. When robot battery level gets low, there is a decrease in the range. The range is assumed unidirectional.

- **Common Behaviors**

  - ***Wander*** **(**`time_limit`**,** `detect_range`**)**

    Since the robots' sensors detection range is limited, robots should navigate the arena first until they find something of interest while avoiding collisions with any other objects (including other robots). This *wander* behavior is embedded into other basic behaviors. It uses their built-in sensor information to detect anything in the way for navigation.

However, their sensors are not good enough to detect other robots especially in the experiment case. Routines to check the distances from other robots are added by looking up the robot position information from the Satellite server. Figure 5.6 shows zones such as left, front and right to represent an other robot within detect range. The front zone has a sector of 60-degree range. Each side zone (right, left) has a 30-degree range of coverage. *Wander* has two input arguments: `time_limit` for the wander time limit and `detect_range` for detecting an other robot. Both can be changed when it is called by other behaviors.

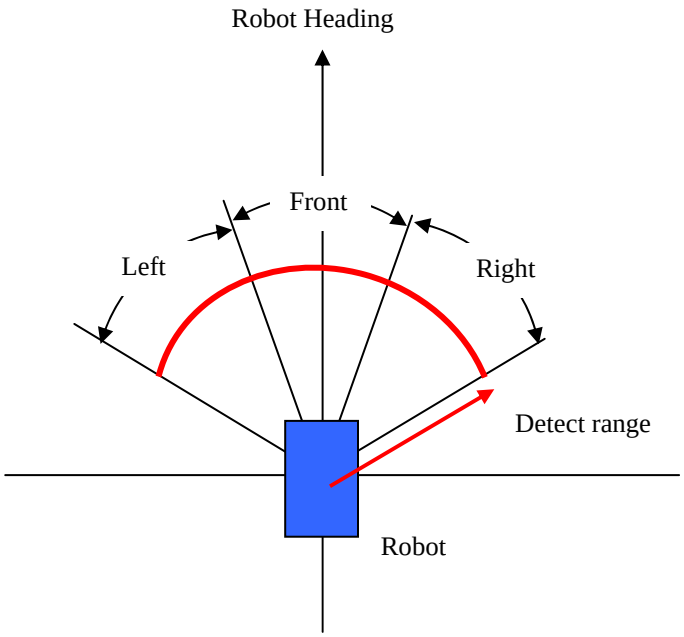


Figure 5.6: Robot detection range to avoid collision

The sensors are frontal, left and right. By detecting presence in either or both, the sector (Left, Front, Right) may be discerned. The actual

value of the proximity sensor estimates the distance. After a certain distance, the sensor input doesn't change.

- ***goForEnergy&getEnergy*** **(**`destination`**,** `time_limit`**)**

Each robot internal energy level decreases with time. For a Robot to survive, it needs to find the energy source and charge itself. When, the charging station is occupied by another robot, the robot should wait until the charging process of the current occupant is over. The dynamics of charging process is modeled as a linear time delay for this research. Figure 5.7 shows the robot's internal energy change before and after the charging station visit. *goForEnergy&getEnergy* has `time_limit` to break this behavior loop after `time_limit` has elapsed. Robot energy level is bounded by its maximum battery capacity.

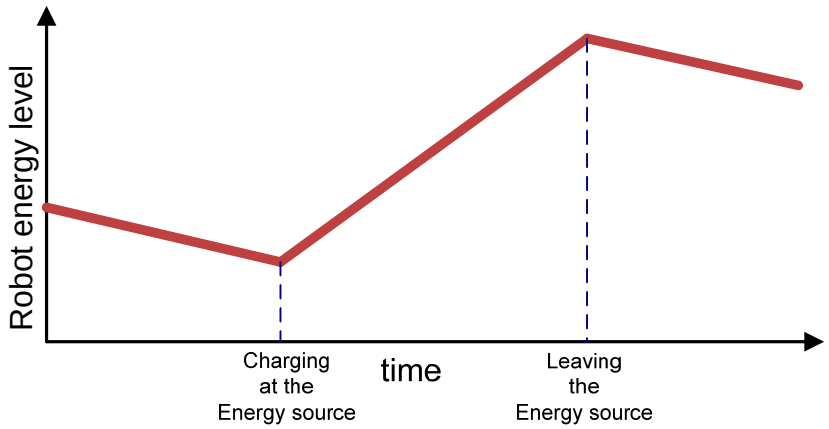


Figure 5.7: Robot charging at the charging station

The robot's internal energy decreases continuously until visiting the energy source. While at the charging station, the robot replenishes its energy fast. The function *goForEnergy&getEnergy* has as input arguments the `time_limit` which represents the time to break the loop after `time_limit` has passed.

- ***goHome&chargeHomeORgetEnergy*(`destination`, `time_limit`)**

Robots may have two reasons to go home. One is to charge its home and the other one is to charge itself. Figure 5.8 shows two cases of robot and home energy interactions. The rate at which a robot loses energy is modeled much faster than that of a home. When a robot reaches home, the resulting energy of home and robot will be the mean value of before the robot reaches home. The dynamics of the charging process is modeled as a time delay for this research. Home energy level is bounded by its maximum battery capacity.

Robots take *wander* behavior to find their home when the destination can not be detected. *goHome&chargeHomeORgetEnergy* has an input argument of `time_limit` to break the loop after time_limit has passed.

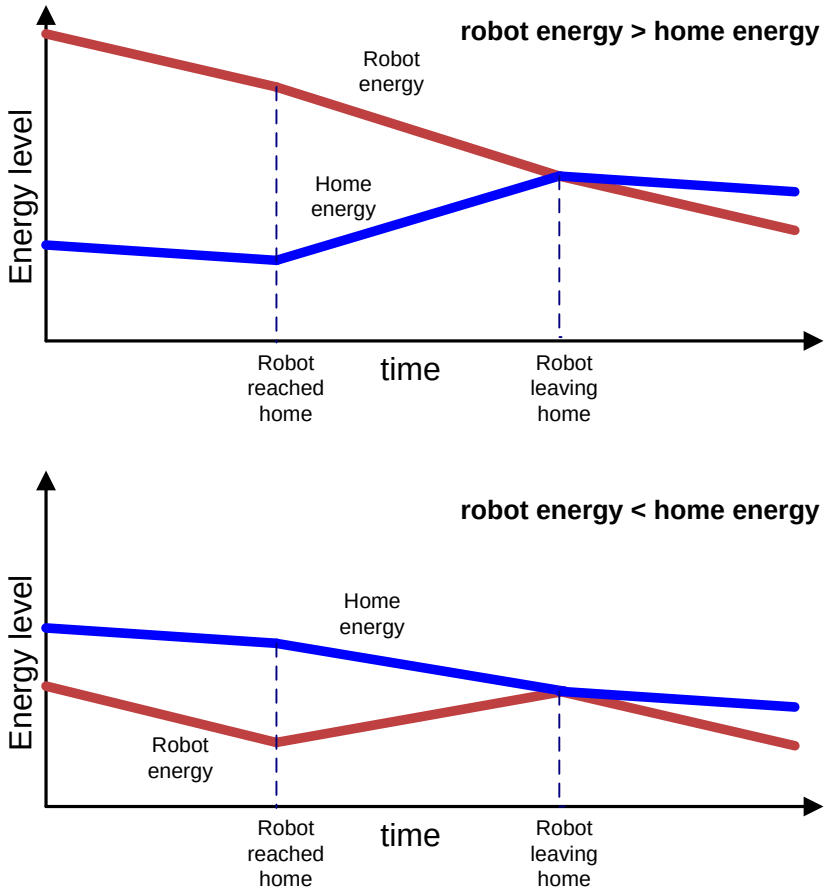


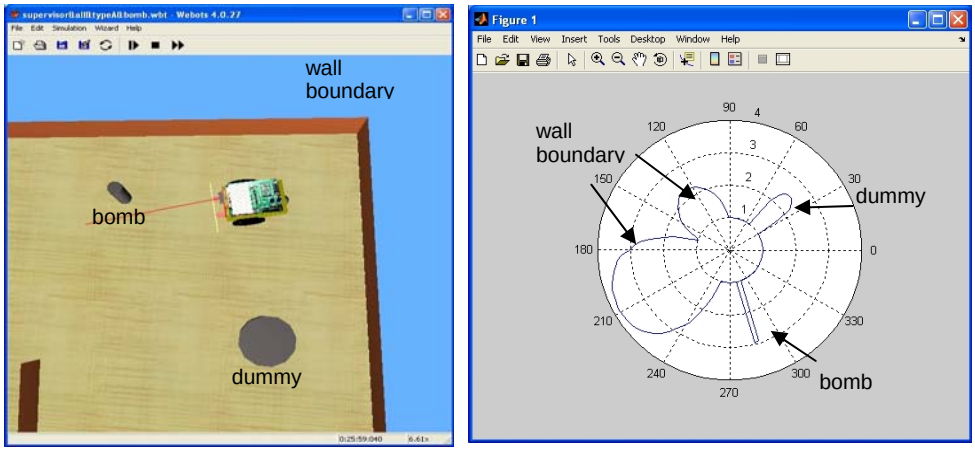
Figure 5.8: Robot/Home energy exchange

Shown are energies before and after home visit. It is assumed that the resulting energy after the exchange is the average of the energies at encounter. The rate of energy consumption of the home is slower than that of the robot.

- **ScannerBot**

  - ***scanField&logPosition***`(time_limit)`

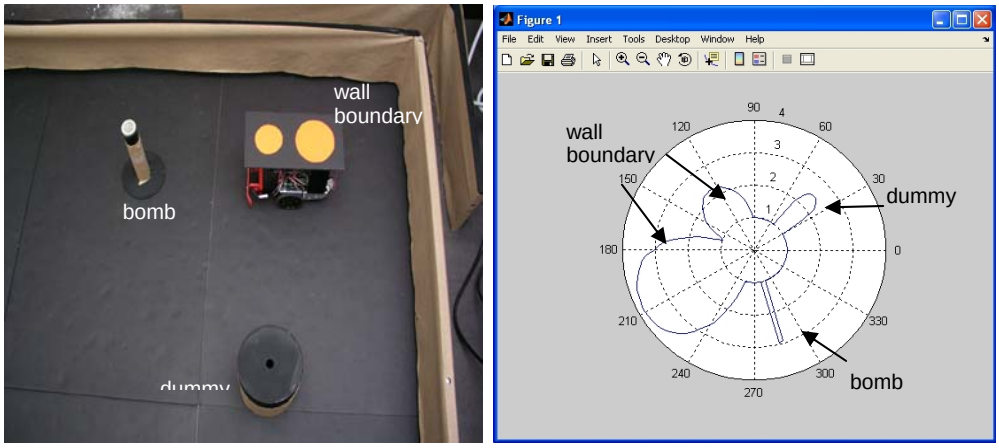
This is a unique behavior of the ScannerBot. It will randomly *wander* the arena and scan the field to find possible bomb locations. Figure 5.9 (a) shows the ScannerBot scanning the environment to locate possible bomb locations. The polar plot in Figure 5.9 (b) enables finding out the possible bomb location based on scan signatures. Figure 5.10 shows an experimental version of Figure 5.9. A numeric filter was designed to filter out all the other signals (with signatures different than bombs) based on the slope and height versus width ratio of the signal's features. Figures 5.11 and 5.12 show the filter results for sequences of scanned signals for both cases.



(a) ScannerBot scanning surrounding (b) Polar plot scan of environment

Figure 5.9: ScannerBot scanning behavior (simulation)

On the left (a), a typical scenario is shown, where the robot is near a corner with a bomb and a dummy in its viewing area. On the right (b), a polar plot showing the sensor output. The signatures of different features are shown. The wall boundaries signature shape is different than the ones for dummies and bombs. A similar scenario for the experimental case is shown in Figure 5.10.



(a) ScannerBot scanning surrounding (b) Polar plot scan of environment

Figure 5.10: ScannerBot scanning behavior (experiment)

The experiment shown is similar to the simulation case of Figure 5.9. On the left (a), the robot is near a corner with a bomb and a dummy in its viewing area. On the right (b), the polar plot showing the sensor output with the different.

If the robot finds something, it posts the object's location on the *blackboard* for other robots to access it. The *scanField&logPosition* function has one input argument, `time_limit`, to break the loop after the time_limit has passed. This routine can filter out objects already posted in the blackboard by calculating the distance between new findings and locations in the list.

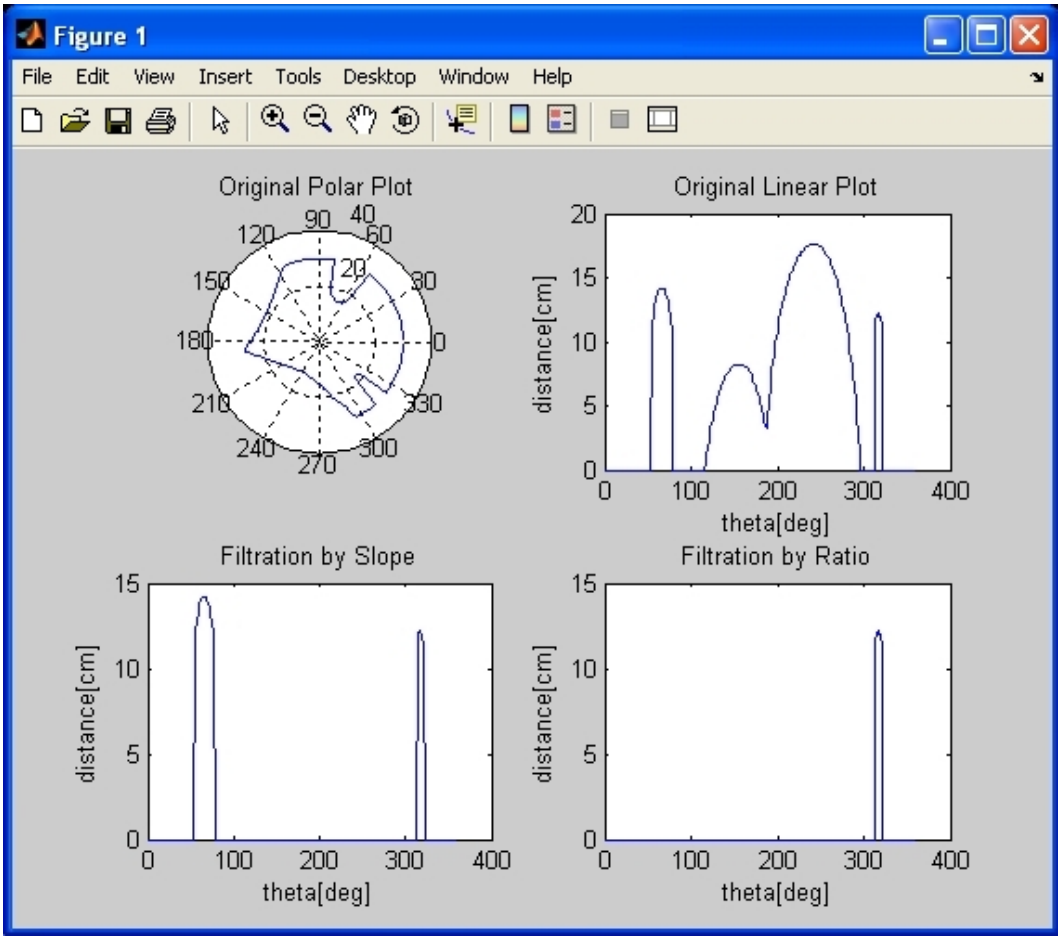


Figure 5.11: ScannerBot numeric filter result for a simulation sequence

The polar plot scanned data is unwrapped linearly (top-right) and then filtered (bottom-right). The filter is basically a high-pass filter that searches for the bomb's signature that is a sharp peak. Walls and dummies are shallow or "fat" features. A similar result for the experimental setup is shown in Figure 5.12.

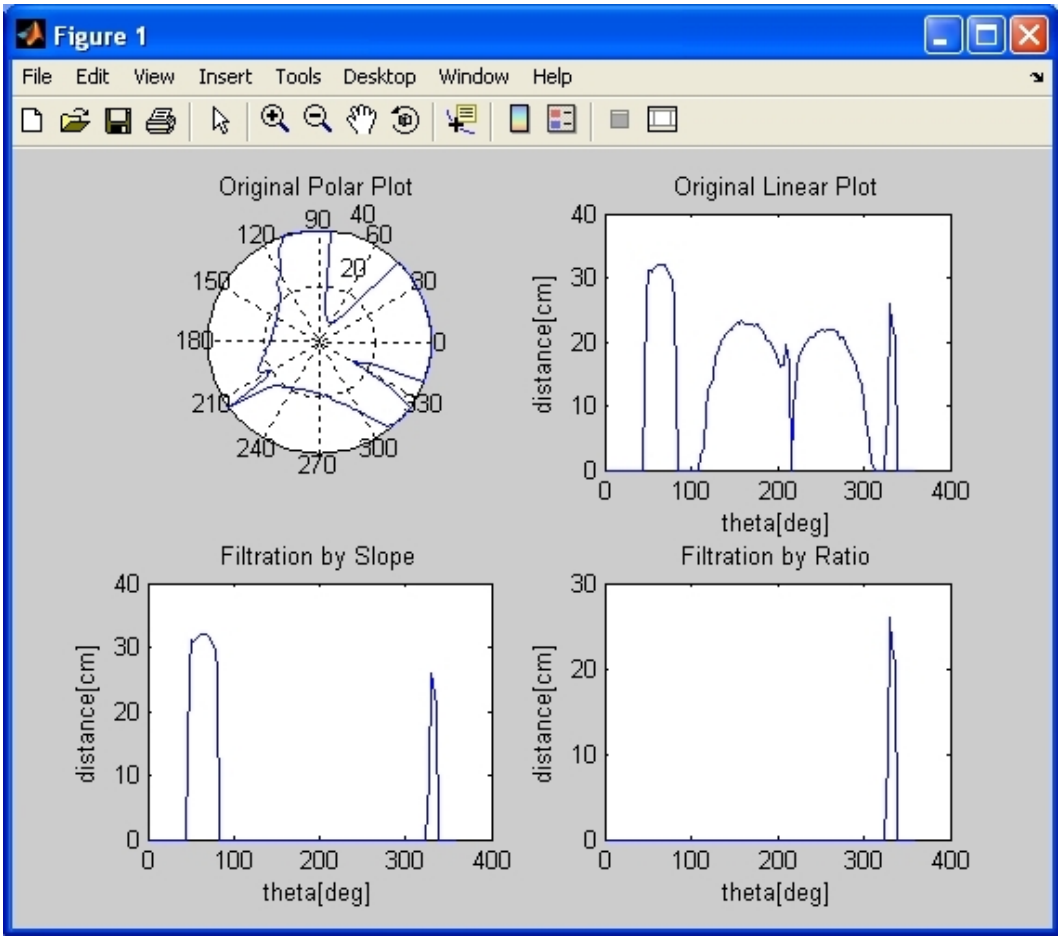


Figure 5.12: ScannerBot numeric filter filtration sequence (experiment)

The experiment is similar to the simulation shown in Figure 5.11. The polar plot scanned data is unwrapped linearly (top-right) and then filtered (bottom-right).

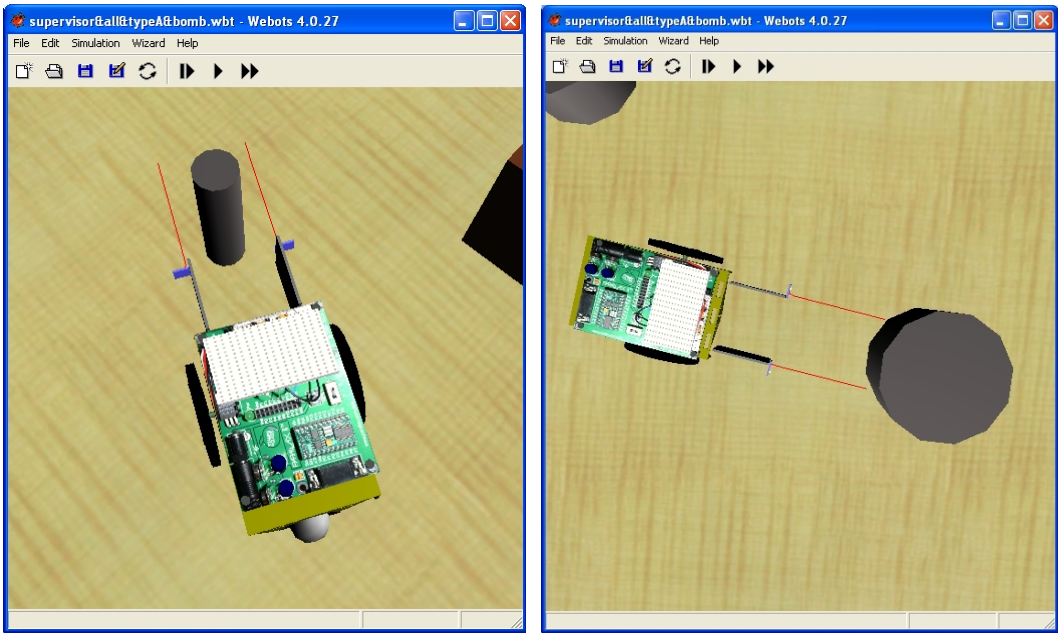
- ***goForBomb&rescan*(`destination`, `time_limit`)**

The ScannerBot may post wrong information on the blackboard. In this case, the InspectorBot posts a rescan request on the board. After the rescan, the ScannerBot can correct the initial posting (for example, update the

position or delete the position in the black board) or can ask the InspectorBot for another work (inspection) order. The *goForBomb&rescan* function has input arguments of `destination` to find the location and `time_limit` to break the loop after time_limit has passed.

- **InspectorBot**
  - ***goForBomb&inspect(***`destination`**,** `time_limit` **)**

    This is a unique behavior of the InspectorBot. It checks the blackboard for postings on new items found that have not been classified and are considered candidates for bombs. The InspectorBot finds the locations (posted by the ScannerBot) in the blackboard and goes to the tagged location to validate (inspect) whether the object is a bomb or not. Figure 5.13 and 5.14 show the behavior of the InspectorBot while inspecting objects. Based on the diameter of the objects, it can differentiate between those objects. If its inspection result is a bomb, it will post a request to the DefuseBot on the blackboard. If it is not a bomb, it will request a rescan to the ScannerBot.



(a) bomb (b) dummy

Figure 5.13: InspectorBot inspecting potential bombs (simulation)

The InspectorBot finds the locations (posted by the ScannerBot) in the blackboard and goes to the tagged location to validate (inspect) whether the object is a bomb or not. Based on the diameter of the objects, it can differentiate between bombs and dummy objects. If its inspection result is a bomb, it will post a request to the DefuseBot on the blackboard.

(a) bomb (b) dummy

Figure 5.14: InspectorBot inspecting potential bombs (experiment)

This is an experimental equivalent to Figure 5.13. Based on the diameter of the objects, it can differentiate between those objects.

- **DefuserBot**
  - ***goForBomb&defuse*(**`destination`**,** `time_limit`**)**

    This is a unique behavior of the DefuserBot. It will find the locations posted in the blackboard by the InspectorBot and defuse the bomb at the location. Figures 5.15 and 5.16 show the behavior of the DefuserBot while defusing a bomb. It will approach the bomb until the bumper sensor at the front detects it, which emulates bomb defusing process. The bomb becomes a dummy after this behavior is successful. If the DefuserBot can not defuse the bomb, it will request a rescan order to the ScannerBot for updated position information.

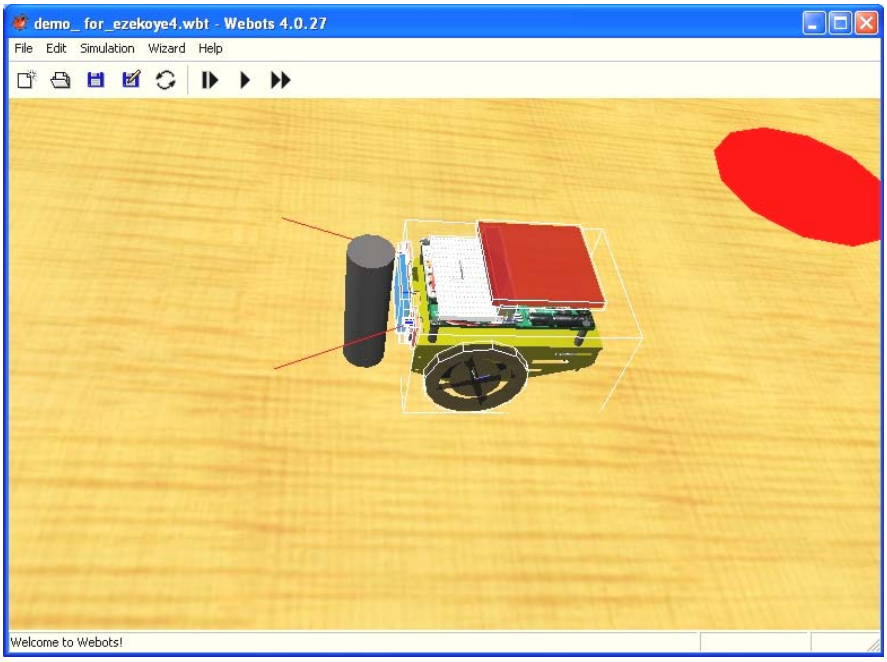


Figure 5.15: DefuserBot defusing a bomb (simulation)

It will find the locations posted in the blackboard by the InspectorBot, go to that location and defuse the bomb.

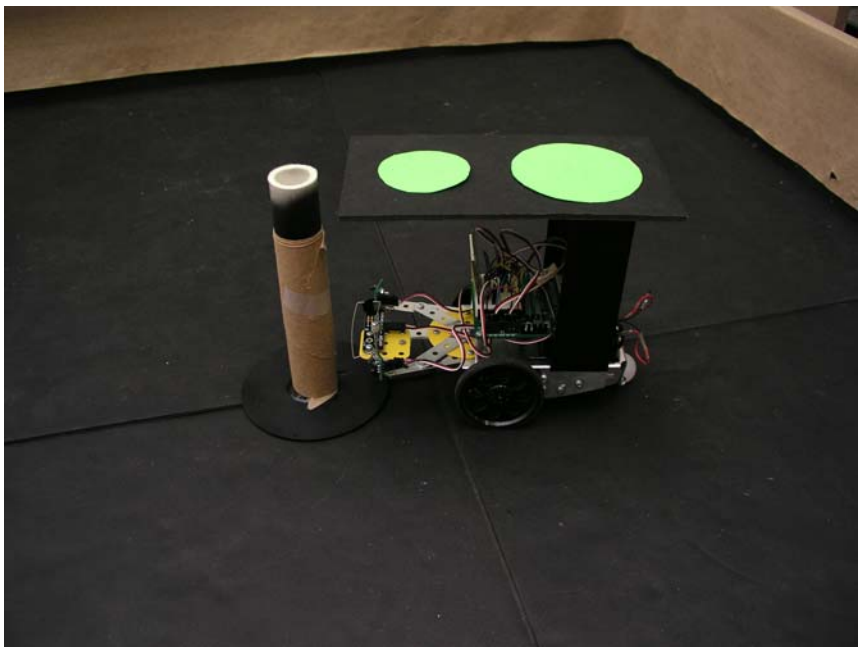


Figure 5.16: DefuserBot defusing a bomb (experiment)

This is the experiment of the equivalent simulation in Figure 5.15. It will find the locations posted in the blackboard by the InspectorBot, go to that location and defuse the bomb.

- **Basic Behaviors in summary**

The basic behaviors explained so far can be categorized as in Table 5.1. Figure 5.17 shows flow charts of two types of behaviors. The type1 BB is a time-limited where the robot navigates avoiding obstacles until the time_limit is reached at which point the behavior is executed. The type2 BB has a destination as a parameter. The robot wanders until the destination is reached or the time_limit is exceeded. If the location is reached, the behavior is executed, but if the time_limit is exceeded, the behavior is aborted.

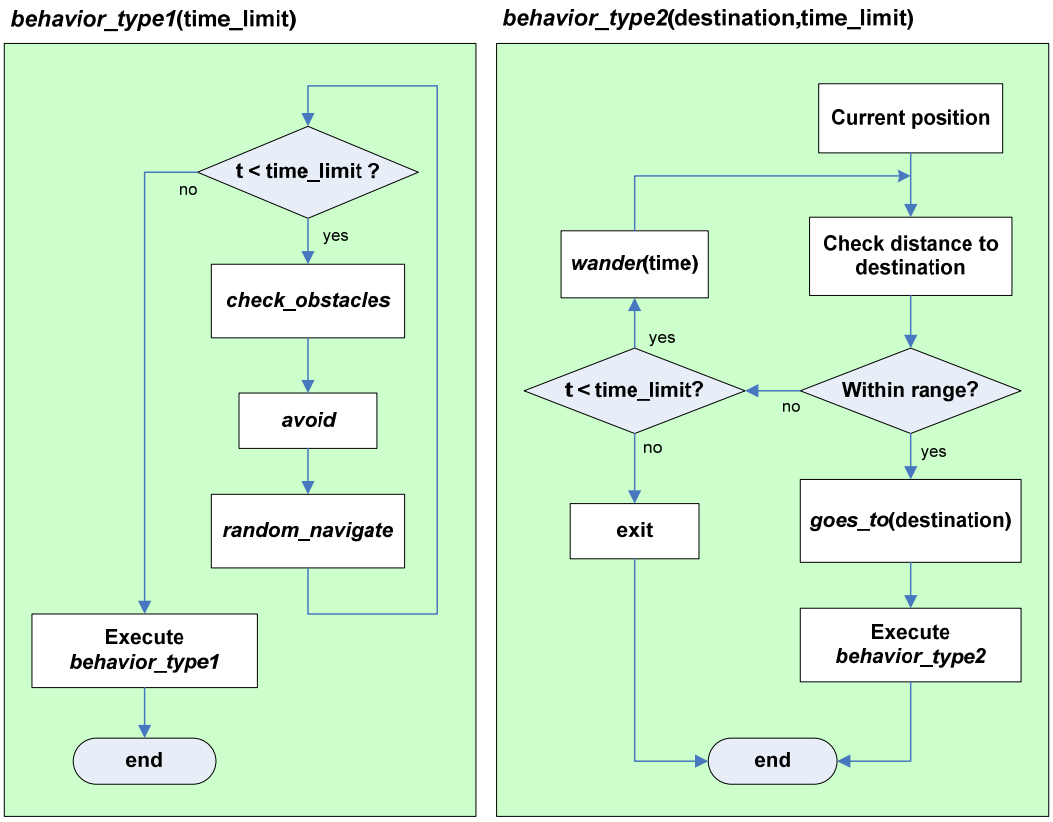


Figure 5.17: Two types of basic behaviors

The behavior on the left is time-limited where the robot navigates avoiding obstacles until the time_limit is reached at which point the behavior is executed. The other behavior has a destination as a parameter. For this type of behavior, the robot wanders until the destination is reached or the time_limit is exceeded. If the location is reached, the behavior is executed, but if the time_limit is exceeded, the behavior is aborted.

Robot name	Basic Behaviors	type
ScannerBot	*wander*(`time_limit, detect_range`)	1
	goForEnergy&getEnergy(`destination, time_limit`)	2
	goHome&chargeHomeORgetEnergy(`destination, time_limit`)	2
	scanField&logPosition(`time_limit`)	1
	goForBomb&rescan(`destination, time_limit`)	2
InspectorBot	*wander*(`time_limit, detect_range`)	1
	goForEnergy&getEnergy(`destination, time_limit`)	2
	goHome&chargeHomeORgetEnergy(`destination, time_limit`)	2
	goForBomb&inspect(`destination, time_limit`)	2
DefuserBot	*wander*(`time_limit, detect_range`)	1
	goForEnergy&getEnergy(`destination, time_limit`)	2
	goHome&chargeHomeORgetEnergy(`destination, time_limit`)	2
	goForBomb&defuse(`destination, time_limit`)	2

Table 5.1: Robot basic behaviors

5.2.2 Learning in Detail

- **Robot states**

Robots have their own way of describing their world using states. Table 5.2 shows each robot's states. The states are represented using Boolean logic.

A state is defined as 'robot battery low' when its battery level reaches below 30. Robot battery level can vary from 0 to 100 depending on robot's visit to home and charging station. When the 'robot battery low' is '1', the robot's detection ranges for other locations (for example home, charging station and bomb locations) are decreased as described in Figure 5.5.

There are five states representing location availability information for the robots. When these states are '1's, robot can reach the locations directly without

executing *wander* behavior while searching the location. And the last three states in the Table are from the blackboard and represent requests among robots during the bomb disposal process.

States	ScannerBot	InspectorBot	DefuserBot
robot battery low	1/0	1/0	1/0
ScannerBot Home near	1/0	N/A	N/A
InspectorBot Home near	N/A	1/0	N/A
DefuserBot Home near	N/A	N/A	1/0
charging station near	1/0	1/0	1/0
location on the board near	1/0	1/0	1/0
request to rescan exists	1/0	N/A	N/A
request to inspect exists	N/A	1/0	N/A
request to defuse exists	N/A	N/A	1/0

Table 5.2: Robot states

Shown are the possible values of the different variables. Most of the values are binary {0,1}.

- **Blackboard**

As described in Chapter 3, the blackboard is used for the robots to share their information collected during their navigation in the arena. The blackboard is managed in the Satellite PC by running a server. Table 5.3 shows data structure of the blackboard. Each bomb position is represented using Cartesian coordinate system. And each disposal process is represented in Boolean logic.

Figure 5.18 shows information transition of the blackboard depending on each process of the bomb disposal. Shaded areas (SCAN and RESCAN by the

ScannerBot, INSPECT by the InspectorBot, DEFUSE by the DefuserBot) represent various stages of bomb disposal process. And each process is done by the robot in charge of each process. Solid arrows indicate a bomb disposal and broken arrows show internal feedback loops in the group to correct their postings. When experiment starts, the server at the Satellite initialize the blackboard matrix with rows of [999 999 0 0 999].

bomb	x	y	inspected	defused	rescanned
1	…	…	1/0	1/0	1/0
2	…	…	1/0	1/0	1/0
3	…	…	1/0	1/0	1/0
4	…	…	1/0	1/0	1/0
.	…	…	.	.	.
.	…	…	.	.	.

Table 5.3: Data structure of the blackboard

Shown are the possible values of the different variables. Most of the values are binary {0,1}.

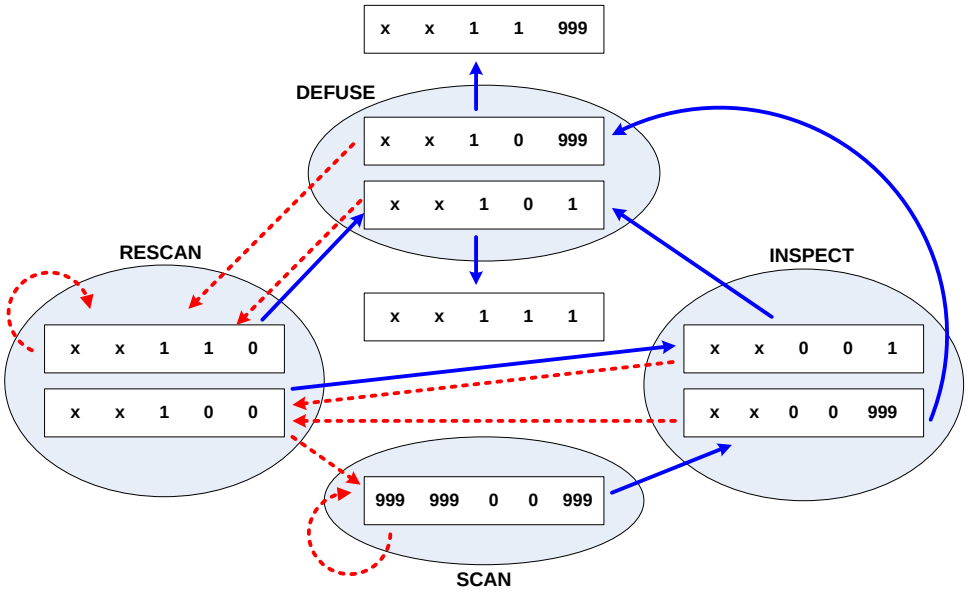


Figure 5.18: Bomb information transition in the blackboard

Shaded areas represent various stages of bomb disposal process. And each process is done by the robot in charge of each process. Solid arrows indicate a bomb disposal. Broken arrows show internal feedback loops in the group to correct their postings. The postings in the blackboard are changed depending on the stage of the disposal by robots.

- **Robot learning**

Figure 5.19 shows a block diagram of a robot during the learning sequence. It checks its own states continuously, which are affected by other robots and the world. During learning, the lookup table will be changed as the antibodies in the clonal selection evolve. Table 5.4 shows a lookup table. If the lookup table generates a solution with more than one behavior (for example, condition 000001 and 111111 in Table 5.4) for the robot, it will engage the *wander* behavior.

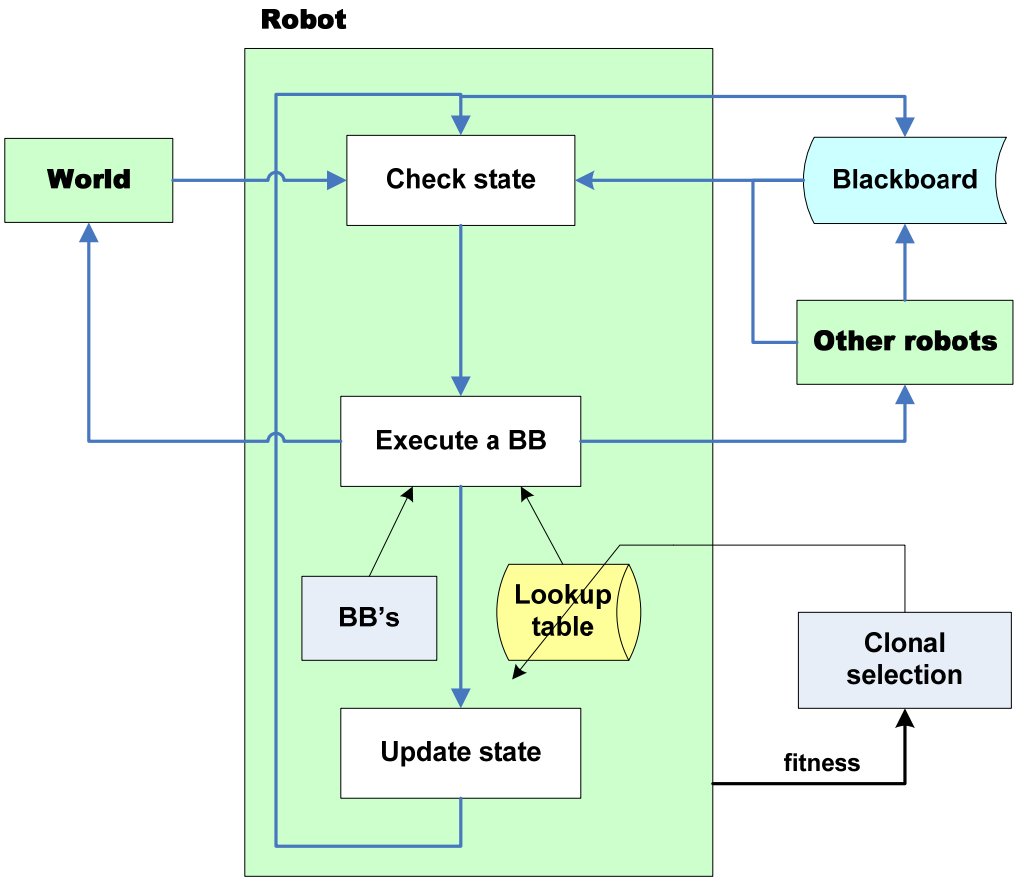


Figure 5.19: Block diagram of a robot during learning

The robot checks its own states continuously, which are affected by other robots and the world. Given its current state, the robot selects a BasicBehavior (BB) from the Lookup Table and executes it. As a result, the state will change. During learning, the lookup table will be changed as the antibodies in the clonal selection evolve.

Condition	Basic Behaviors				
	BB1	BB2	BB3	BB4	BB5

000000	1	0	0	0	0
000001	0	0	1	1	0
000010	0	0	1	0	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
111101	0	1	0	0	0
111110	0	0	0	1	0
111111	0	1	1	0	0

Table 5.4: Lookup table

First column represents condition reflecting current robot states. The rest of columns are Basic Behaviors (BB).

Figure 5.20 shows a block diagram where all three robots are involved. The blackboard is a media for them to communicate. During learning, there should be a synchronized way to reset the blackboard and robot positions every time, when the selection or reselection process of the clonal selection starts, as shown in Figure 5.20. In this research, the server running in the Satellite PC performs this function. During the real environment experiment, this synchronization routine can be used to replace the robot's batteries.

During the learning, if robots try BB's of type2 with null destination, they are engaged to the *wander* BB. If the ScannerBot can not find potential bombs during the time window of the learning session (the posting of the initial bomb information totally rely on the ScannerBot's random search), there will not be any opportunities for the rest of them to develop their learning to inspect and defuse

bombs. Therefore, locations of bombs with false information are initially posted in the blackboard to speed up the learning process.

Table 5.5 summarizes the items that are used for the affinity evaluation of each generation of antibody. Affinity can be measured by assigning weight to each item and combining them in various ways (for example, linear combination). The bottom line is how to reflect the global and local interest of the robot group with these items in the table.

To be used for Affinity	DefuserBot	ScannerBot	InspectorBot
robot energy	maintain level	maintain level	maintain level
home energy	maintain level	maintain level	maintain level
no. of request executed	maximize bomb defused	maximize bomb rescanned	maximize bomb inspected
bomb found	N/A	maximize	N/A

Table 5.5: States or other conditions for affinity evaluation functions

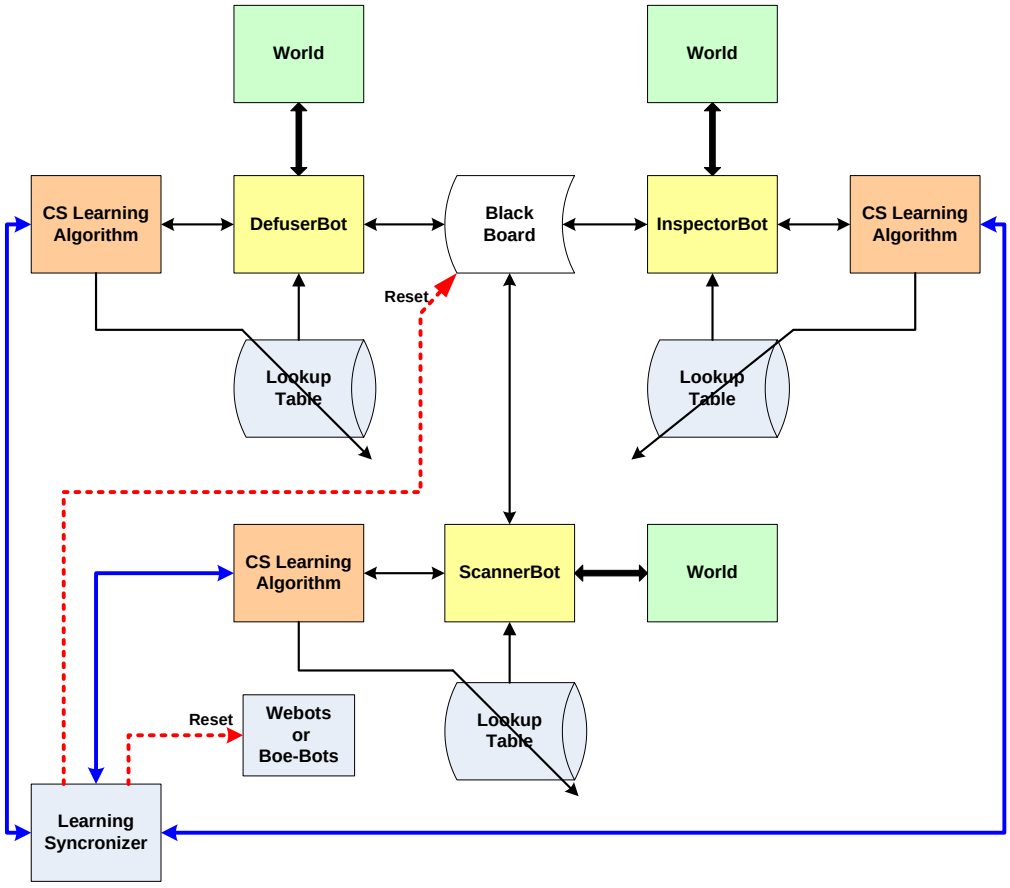


Figure 5.20: Three robot learning

From the signals from the clonal selection affinity evaluation routine of each robot, the server controls the execution of each learning process. At the same time, the blackboard is reset after each affinity evaluation process ends. Webots or Boe_Bots are reset depending on the experiment.

5.3 RESULTS AND DISCUSSION

5.3.1 Simulation Results and Discussion

Figure 5.21 shows a simulation screen shot. There was false bomb information put on the blackboard for each robot on purpose, which worked as vaccines during AIS learning to speed up the learning. Considering simulation time, following assumptions are made:

robot knows all the locations

robot can take batteries one by one at the charging station and bring home.

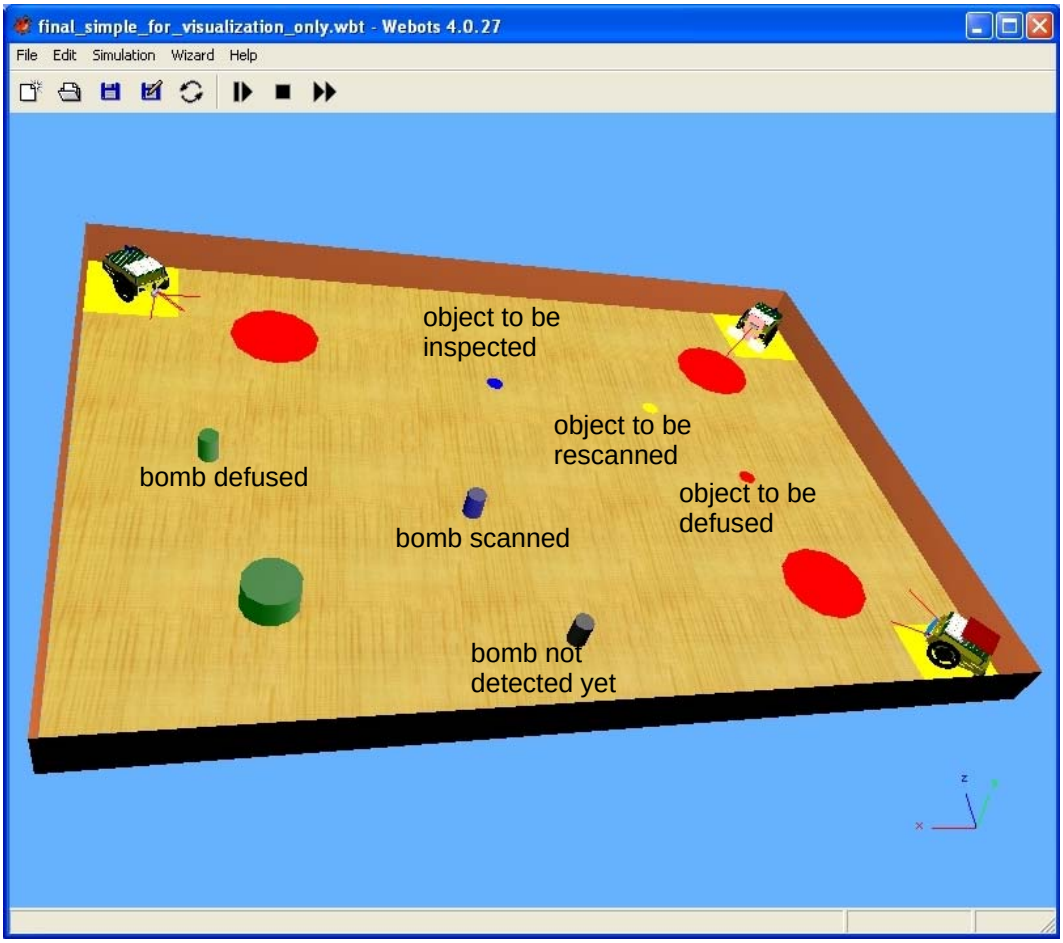


Figure 5.21: Simulation condition

Conditions for the learning algorithm are:

population size: 30

number of generation: 25

hypermutation probability: 0.1

lookup table size: 4 x 3

vaccines:

1 false bombs information for the DefuserBot

1 false bombs information for the InspectorBot

1 true bombs information for the InspectorBot

1 false bombs information for the ScannerBot

weights for the selection process:

ScannerBot

battery picked: 50

battery brought home: 150

rescan: 200

InspectorBot

battery picked: 50

battery brought home: 150

inspection: 300

DefuserBot

battery picked: 50

battery brought home: 150

defuse: 300

robot	antigen		antibody		
	carrying battery	work order exists	go for battery	bring battery home	rescan
ScannerBot	0	0	0	1	1
	0	1	1	0	0
	1	0	0	0	0
	1	1	0	0	1
			go for battery	bring battery home	inspect
InspectorBot	0	0	1	0	0
	0	1	0	0	1
	1	0	1	1	0

	1	1	1	0	0
			go for battery	bring battery home	defuse
DefuserBot	0	0	1	0	0
	0	1	1	0	0
	1	0	0	0	1
	1	1	0	1	1

Table 5.6: Antigens and antibodies for robots during simulation

Table 5.6 shows antigens and antibodies during robot learning. Antigens represent robot condition and antibodies are equivalent to the lookup table. Each robot has three basic behaviors. Tables 5.7, 5.8 and 5.9 represent actual robot behaviors during affinity measurement for corresponding antibodies during learning.

InspectorBot			
antigen	antibody	equivalent action	affinity
(0 1)	0 0 1	inspect	300
(0 1)	0 0 1	inspect	300
(0 0)	1 0 0	go for battery	50
(1 0)	1 1 0	waste time	0
(1 0)	1 1 0	waste time	0
time expired			650

Table 5.7: Change of InspectorBot behaviors during learning

ScannerBot			
antigen	antibody	equivalent action	affinity
(0 1)	1 0 0	go for battery	50
(1 1)	0 0 1	rescan	200
(1 1)	0 0 1	rescan	200
(1 0)	0 0 0	waste time	0
(1 0)	0 0 0	waste time	0
(1 0)	0 0 0	waste time	0
time expired			450

Table 5.8: Change of ScannerBot behaviors during learning

DefuserBot			
antigen	antibody	equivalent action	affinity
(0 1)	1 0 0	go for battery	50
(1 1)	0 1 1	waste time	0
(1 1)	0 1 1	waste time	0
(1 1)	0 1 1	waste time	0
(1 1)	0 1 1	waste time	0
(1 1)	0 1 1	waste time	0
(1 1)	0 1 1	waste time	0
time expired			50

Table 5.9: Change of DefuserBot behaviors during learning

Figure 5.22 shows ScannerBot evolution of antibodies during learning. Maximum and mean affinities were calculated during the leaning. Figure 5.23 shows how ScannerBot processed bombs and batteries and represented in average numbers during learning.

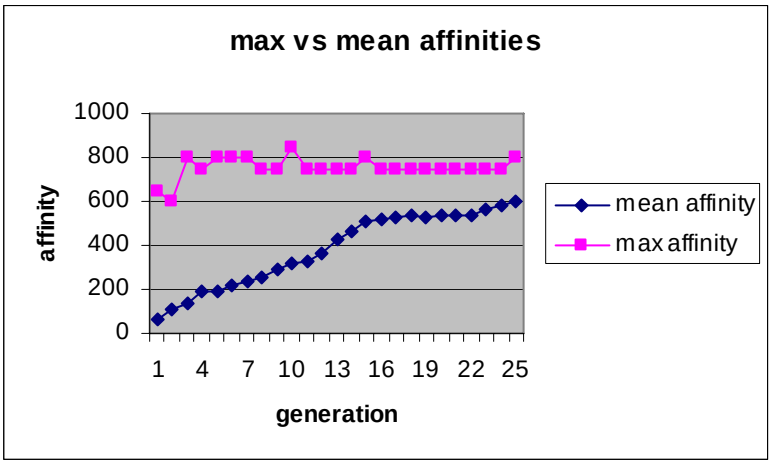


Figure 5.22: ScannerBot affinity evolution during simulation

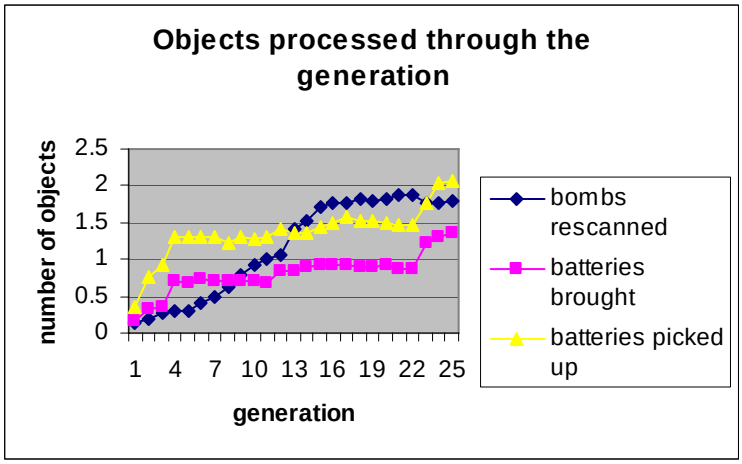


Figure 5.23: ScannerBot behavior evolution during simulation

Table 5.10 shows ScannerBot's antibodies representing maximum affinity for each generation. Table 5.11 is showing average antibodies for each generation. These tables show how actual learning has been done.

1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	0	1	0
1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	0	1	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	0	0	0	0	1	0	0	1	1	0	1
1	0	1	1	0	0	1	0	1	0	1	0

1	1	1	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	0	1	1	0	0	1	1	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
0	1	0	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0

Table 5.10: ScannerBot best antibodies for the generation during simulation

0.6	0.56667	0.63333	0.7	0.56667	0.4	0.36667	0.53333	0.46667	0.4	0.56667	0.5
0.56667	0.5	0.66667	0.66667	0.46667	0.33333	0.4	0.43333	0.43333	0.36667	0.46667	0.46667
0.6	0.46667	0.56667	0.6	0.46667	0.3	0.33333	0.43333	0.36667	0.43333	0.53333	0.5
0.56667	0.53333	0.6	0.63333	0.36667	0.36667	0.36667	0.36667	0.43333	0.46667	0.56667	0.46667
0.5	0.46667	0.6	0.7	0.33333	0.4	0.4	0.5	0.4	0.46667	0.56667	0.5
0.53333	0.43333	0.63333	0.63333	0.33333	0.46667	0.46667	0.5	0.4	0.53333	0.53333	0.43333
0.56667	0.4	0.66667	0.6	0.3	0.43333	0.46667	0.56667	0.43333	0.5	0.53333	0.4
0.5	0.4	0.73333	0.56667	0.26667	0.4	0.46667	0.5	0.46667	0.5	0.53333	0.4
0.56667	0.4	0.73333	0.56667	0.23333	0.46667	0.5	0.46667	0.46667	0.46667	0.5	0.4
0.53333	0.43333	0.66667	0.56667	0.2	0.46667	0.56667	0.5	0.56667	0.5	0.46667	0.36667
0.6	0.43333	0.56667	0.53333	0.16667	0.43333	0.56667	0.46667	0.6	0.4	0.43333	0.36667
0.63333	0.46667	0.46667	0.53333	0.13333	0.43333	0.5	0.43333	0.56667	0.43333	0.4	0.33333
0.63333	0.5	0.43333	0.5	0.13333	0.5	0.6	0.43333	0.53333	0.43333	0.36667	0.36667
0.7	0.46667	0.43333	0.46667	0.066667	0.5	0.53333	0.43333	0.6	0.43333	0.36667	0.36667
0.73333	0.5	0.43333	0.46667	0.033333	0.53333	0.6	0.43333	0.5	0.4	0.43333	0.43333
0.7	0.43333	0.5	0.43333	0.033333	0.56667	0.6	0.5	0.53333	0.36667	0.43333	0.46667
0.73333	0.43333	0.5	0.43333	0.033333	0.56667	0.6	0.53333	0.56667	0.36667	0.43333	0.5
0.73333	0.5	0.5	0.46667	0.033333	0.53333	0.56667	0.53333	0.6	0.46667	0.43333	0.5
0.7	0.53333	0.46667	0.46667	0.033333	0.53333	0.5	0.53333	0.56667	0.46667	0.43333	0.5
0.73333	0.5	0.46667	0.46667	0.033333	0.53333	0.46667	0.5	0.46667	0.5	0.4	0.46667
0.7	0.46667	0.53333	0.46667	0	0.56667	0.5	0.43333	0.5	0.46667	0.46667	0.5
0.66667	0.53333	0.53333	0.46667	0	0.56667	0.53333	0.43333	0.43333	0.36667	0.46667	0.5
0.66667	0.5	0.5	0.46667	0	0.53333	0.43333	0.46667	0.4	0.33333	0.56667	0.36667

0.73333	0.5	0.46667	0.46667	0	0.53333	0.46667	0.43333	0.43333	0.33333	0.6	0.43333
0.7	0.5	0.46667	0.46667	0	0.53333	0.43333	0.4	0.4	0.33333	0.6	0.43333

Table 5.11: ScannerBot mean antibodies for the generation during simulation

Figure 5.24 shows InspectorBot evolution of antibodies during learning. Maximum and mean affinities were calculated during the leaning. Figure 5.25 shows how InspectorBot processed bombs and batteries and represented in average numbers during learning.

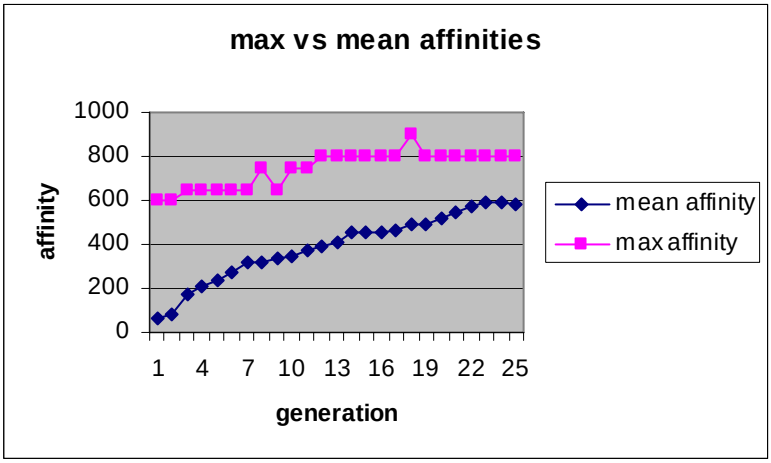


Figure 5.24: InspectorBot affinity evolution during simulation

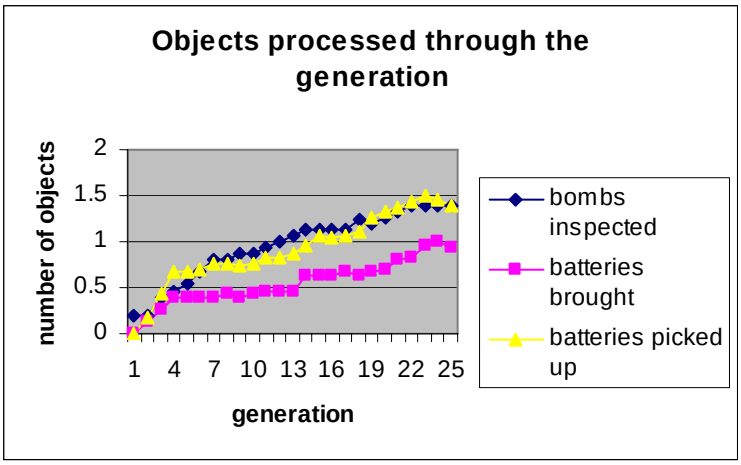


Figure 5.25: InspectorBot behavior evolution during simulation

Table 5.12 shows InspectorBot's antibodies representing maximum affinity for each generation. Table 5.13 is showing average antibodies for each generation. These tables show how actual learning has been done.

1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	0	1	0
1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	0	1	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	0	0	0	0	1	0	0	1	1	0	1
1	0	1	1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	0	1	1	0	0	1	1	0	0	1	0

1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
0	1	0	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0

Table 5.12: InspectorBot best antibodies for the generation during simulation

0.6	0.56667	0.63333	0.7	0.56667	0.4	0.36667	0.53333	0.46667	0.4	0.56667	0.5
0.56667	0.5	0.66667	0.66667	0.46667	0.33333	0.4	0.43333	0.43333	0.36667	0.46667	0.46667
0.6	0.46667	0.56667	0.6	0.46667	0.3	0.33333	0.43333	0.36667	0.43333	0.53333	0.5
0.56667	0.53333	0.6	0.63333	0.36667	0.36667	0.36667	0.36667	0.43333	0.46667	0.56667	0.46667
0.5	0.46667	0.6	0.7	0.33333	0.4	0.4	0.5	0.4	0.46667	0.56667	0.5
0.53333	0.43333	0.63333	0.63333	0.33333	0.46667	0.46667	0.5	0.4	0.53333	0.53333	0.43333
0.56667	0.4	0.66667	0.6	0.3	0.43333	0.46667	0.56667	0.43333	0.5	0.53333	0.4
0.5	0.4	0.73333	0.56667	0.26667	0.4	0.46667	0.5	0.46667	0.5	0.53333	0.4
0.56667	0.4	0.73333	0.56667	0.23333	0.46667	0.5	0.46667	0.46667	0.46667	0.5	0.4
0.53333	0.43333	0.66667	0.56667	0.2	0.46667	0.56667	0.5	0.56667	0.5	0.46667	0.36667
0.6	0.43333	0.56667	0.53333	0.16667	0.43333	0.56667	0.46667	0.6	0.4	0.43333	0.36667
0.63333	0.46667	0.46667	0.53333	0.13333	0.43333	0.5	0.43333	0.56667	0.43333	0.4	0.33333
0.63333	0.5	0.43333	0.5	0.13333	0.5	0.6	0.43333	0.53333	0.43333	0.36667	0.36667
0.7	0.46667	0.43333	0.46667	0.066667	0.5	0.53333	0.43333	0.6	0.43333	0.36667	0.36667
0.73333	0.5	0.43333	0.46667	0.033333	0.53333	0.6	0.43333	0.5	0.4	0.43333	0.43333
0.7	0.43333	0.5	0.43333	0.033333	0.56667	0.6	0.5	0.53333	0.36667	0.43333	0.46667
0.73333	0.43333	0.5	0.43333	0.033333	0.56667	0.6	0.53333	0.56667	0.36667	0.43333	0.5
0.73333	0.5	0.5	0.46667	0.033333	0.53333	0.56667	0.53333	0.6	0.46667	0.43333	0.5
0.7	0.53333	0.46667	0.46667	0.033333	0.53333	0.5	0.53333	0.56667	0.46667	0.43333	0.5
0.73333	0.5	0.46667	0.46667	0.033333	0.53333	0.46667	0.5	0.46667	0.5	0.4	0.46667
0.7	0.46667	0.53333	0.46667	0	0.56667	0.5	0.43333	0.5	0.46667	0.46667	0.5
0.66667	0.53333	0.53333	0.46667	0	0.56667	0.53333	0.43333	0.43333	0.36667	0.46667	0.5
0.66667	0.5	0.5	0.46667	0	0.53333	0.43333	0.46667	0.4	0.33333	0.56667	0.36667
0.73333	0.5	0.46667	0.46667	0	0.53333	0.46667	0.43333	0.43333	0.33333	0.6	0.43333
0.7	0.5	0.46667	0.46667	0	0.53333	0.43333	0.4	0.4	0.33333	0.6	0.43333

Table 5.13: InspectorBot mean antibodies for the generation during simulation

Figure 5.26 shows DefuserBot evolution of antibodies during learning. Maximum and mean affinities were calculated during the leaning. Figure 5.27 shows how DefuserBot processed bombs and batteries and represented in average numbers during learning.

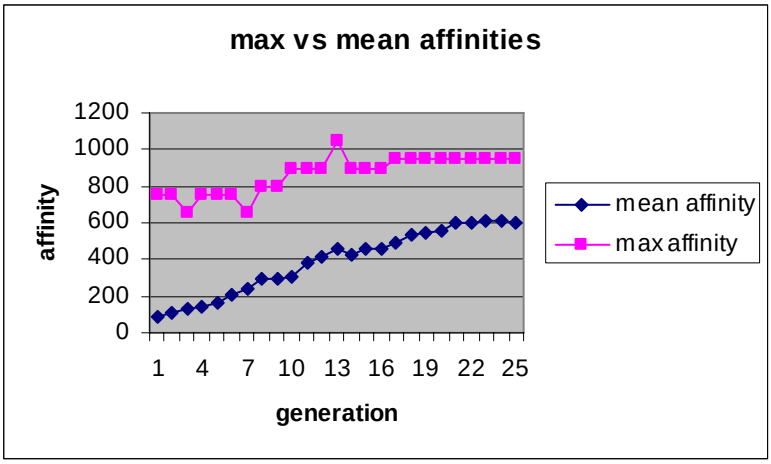


Figure 5.26: DefuserBot affinity evolution during simulation

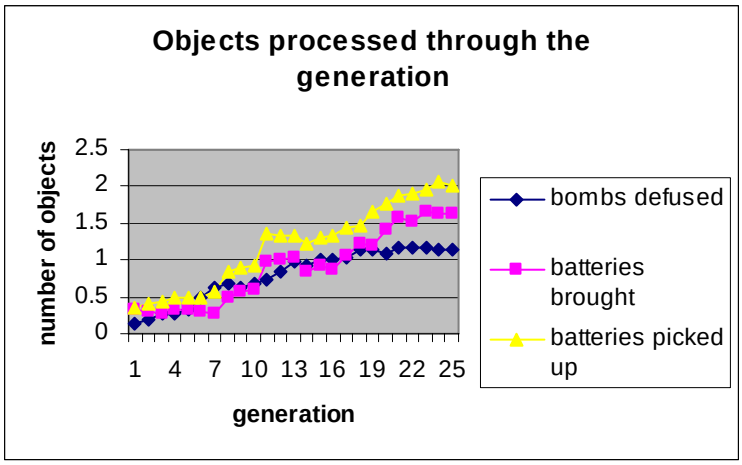


Figure 5.27: DefuserBot behavior evolution during simulation

Table 5.14 shows DefuserBot's antibodies representing maximum affinity for each generation. Table 5.15 is showing average antibodies for each generation. These tables show how actual learning has been done.

1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	0	1	0
1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	0	1	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	1	0	1	0	1	0
1	0	0	0	0	1	0	0	1	1	0	1
1	0	1	1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	0	1	1	0	0	1	1	0	0	1	0

1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
1	1	0	1	0	0	1	0	0	0	1	0
0	1	0	1	0	0	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	1	0

Table 5.14: DefuserBot best antibodies for the generation during simulation

0.6	0.56667	0.63333	0.7	0.56667	0.4	0.36667	0.53333	0.46667	0.4	0.56667	0.5
0.56667	0.5	0.66667	0.66667	0.46667	0.33333	0.4	0.43333	0.43333	0.36667	0.46667	0.46667
0.6	0.46667	0.56667	0.6	0.46667	0.3	0.33333	0.43333	0.36667	0.43333	0.53333	0.5
0.56667	0.53333	0.6	0.63333	0.36667	0.36667	0.36667	0.36667	0.43333	0.46667	0.56667	0.46667
0.5	0.46667	0.6	0.7	0.33333	0.4	0.4	0.5	0.4	0.46667	0.56667	0.5
0.53333	0.43333	0.63333	0.63333	0.33333	0.46667	0.46667	0.5	0.4	0.53333	0.53333	0.43333
0.56667	0.4	0.66667	0.6	0.3	0.43333	0.46667	0.56667	0.43333	0.5	0.53333	0.4
0.5	0.4	0.73333	0.56667	0.26667	0.4	0.46667	0.5	0.46667	0.5	0.53333	0.4
0.56667	0.4	0.73333	0.56667	0.23333	0.46667	0.5	0.46667	0.46667	0.46667	0.5	0.4
0.53333	0.43333	0.66667	0.56667	0.2	0.46667	0.56667	0.5	0.56667	0.5	0.46667	0.36667
0.6	0.43333	0.56667	0.53333	0.16667	0.43333	0.56667	0.46667	0.6	0.4	0.43333	0.36667
0.63333	0.46667	0.46667	0.53333	0.13333	0.43333	0.5	0.43333	0.56667	0.43333	0.4	0.33333
0.63333	0.5	0.43333	0.5	0.13333	0.5	0.6	0.43333	0.53333	0.43333	0.36667	0.36667
0.7	0.46667	0.43333	0.46667	0.066667	0.5	0.53333	0.43333	0.6	0.43333	0.36667	0.36667
0.73333	0.5	0.43333	0.46667	0.033333	0.53333	0.6	0.43333	0.5	0.4	0.43333	0.43333
0.7	0.43333	0.5	0.43333	0.033333	0.56667	0.6	0.5	0.53333	0.36667	0.43333	0.46667
0.73333	0.43333	0.5	0.43333	0.033333	0.56667	0.6	0.53333	0.56667	0.36667	0.43333	0.5
0.73333	0.5	0.5	0.46667	0.033333	0.53333	0.56667	0.53333	0.6	0.46667	0.43333	0.5
0.7	0.53333	0.46667	0.46667	0.033333	0.53333	0.5	0.53333	0.56667	0.46667	0.43333	0.5
0.73333	0.5	0.46667	0.46667	0.033333	0.53333	0.46667	0.5	0.46667	0.5	0.4	0.46667
0.7	0.46667	0.53333	0.46667	0	0.56667	0.5	0.43333	0.5	0.46667	0.46667	0.5
0.66667	0.53333	0.53333	0.46667	0	0.56667	0.53333	0.43333	0.43333	0.36667	0.46667	0.5
0.66667	0.5	0.5	0.46667	0	0.53333	0.43333	0.46667	0.4	0.33333	0.56667	0.36667
0.73333	0.5	0.46667	0.46667	0	0.53333	0.46667	0.43333	0.43333	0.33333	0.6	0.43333
0.7	0.5	0.46667	0.46667	0	0.53333	0.43333	0.4	0.4	0.33333	0.6	0.43333

Table 5.15: DefuserBot mean antibodies for the generation during simulation

InspectorBot and DefuserBot show good evolution results of mean affinity. However, considering the maximum possible fitness values based on the weight conditions, it is still premature stage and it needs more generations of evolution.

The result of ScannerBot shows too much fluctuation and it is due to the weighting of excessive bomb found reward.

5.3.2 Preliminary Experiment Results

The proposed hardware setup was tested for actual robot basic behavior development. Since the architecture for both the simulation and experiment are very similar, the codes for the simulation could be transferred to the experiment without any difficulties. Machine vision system was good enough to pick up robot position information and communication systems seamlessly bridged between different modules and platforms. However, due to fast battery drain, intensive experiment could not be done.

Chapter 6: Conclusions and Future Work

6.1 Summary and Conclusions

This research has shown that the proposed distributed control architecture using CS of AIS can provide a suitable methodology for the autonomous solution of highly confined goal-seeking problems targeted in this research. This document has also highlighted some of the factors such as communication using blackboard involved in robot communication.

Computer simulation has shown the feasibility of AIS as a possible measure for control of a group of robots. The methodology is firmly grounded in the biological sciences and provides robust performance for the intertwined entities involved in most task-achieving MRS. Based on its formal foundation, it provides a platform to characterize interesting relationships and dependencies among MRS task requirements, individual robot control, capabilities, and resulting task performance.

In this research, we do not advocate that the CS performs better than the GA in any application, instead we demonstrate that it is also composed of a biologically inspired algorithm, which performs learning and multi-modal search along the space. Like the GA, the CS algorithm is highly parallel.

The architecture proposed enables a robot group to navigate in an unknown environment. The implementation results are still in a very early state, so it is not advisable to draw major conclusions; the simulation results obtained so far show that based on the proposed modeling of the environment an AIS promises to be a good candidate solution to the problem of robot navigation in

unstructured and unknown environments. We did not exhaust all features that are usually defined in AIS, offering ways of extensions to the proposed approach.

6.2 RESEARCH CONTRIBUTIONS

The work presented in this dissertation is a first of its kind wherein the principles of AIS have been used to model and organize the group behavior of the MRS. This has been presented in the form of a novel algorithm. In addition to the above, generic environments for computer simulation and real experiment have been realized to demonstrate the working of an MRS with considerably low budget. These could potentially be used to implement other algorithms onto the MRS. Therefore can provide a valuable test-bed for AIS ideas, and a useful tool for MRS research.

6.3 FUTURE WORK

Possible future research can be categorized and summarized as follows:

- Learning algorithm

There should be a sensitivity analysis to tune the clonal selection parameters affecting the convergence rate. And to speed up the learning, reinforcement learning methods should be studied.

- Experiment

After the learning algorithm has been improved especially to speed up the learning, experiment should be performed using the setup developed in this research.

- MRS formalism

The basic behaviors for this research are all static and prepared separately for the mission. For real-world deployment, several additions that have been used before in our lab [Westlake, 2005] are suggested. For example, implementing some evolutionary methods that make these behaviors evolve in real-time will be a more practical goal for real world applications. Also, agent technology can be applied to utilize the mobility of the robots.

- Benchmarking

Comparison with other evolutionary methods especially for multi-objective optimization purpose should be followed after this research to compare the performance of the proposed algorithm.

- Hardware

For more flexible robot missions, robots with better built-in sensors and actuators; autonomous charging system should be developed.

Bibliography

Arai, T., Pagello, E. and Parker, L. E., 2002, "Editorial: Advances in Multi-Robot Systems", *IEEE Transactions ON Robotics and Automation*, Vol. 18, No. 5, pp 655-661.

Ashiru, I. and Czarnecki, C. A., 1998, "Experiments in Evolving Communicating Controllers for Teams of Robots," *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pp. 3498-3503.

Brooks, R. A., 1987, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, pp. 264-271.

Brooks, R. A., 1991, "Intelligence without Representation," *Artificial Intelligence*, Vol.47, pp.139-159.

Burnet, F.M., 1959, *The Clonal Selection Theory of Acquired Immunity*, Cambridge University Press, Cambridge.

Carver, N. and Lesser, V., 1994, "Evolution of blackboard control architectures. Expert Systems with Applications," *Expert Systems with Applications*, pp. 1-30.

Carver, N., Cvetanovic, Z. and Lesser, V., 1991, "Sophisticated cooperation in FA/C distributed problem solving systems," *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 191-198.

Cao, Y., Fukunaga, A. S., Kahng, A. B., and Meng, F., 1995, "Cooperative Mobile Robotics: Antecedents and Directions," *IEEE/TSJ International Conference on Intelligent Robots and Systems*, pp. 7-23.

de Castro, L. N. and Von Zuben, F. J., 1999, *Artificial Immune Systems: Part I-Basic Theory and Applications*, FEEC/Univ. Campinas, Brazil.

Dasgupta, D. and Michalewicz, Z., 1997, *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag, Berlin, Heidelberg.

Dasgupta, D. and Attoh-Okine, N., 1997, "Immunity-based systems: A survey," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Vol. 1, pp. 369-374.

Dorigo, M., Maniezzo, V. and Colorni, A., 1996, "The ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics--Part B*, vol. 26, No. 2, pp. 29-41.

Dutta, I., Bogobowicz, A. D. and Gu, J. J., 2004, "Collective Robotics-A Survey of Control and Communication Techniques", *Proceedings of International Conference on Intelligent Mechatronics and Automation*, pp. 505-510.

Engin, O. and Döyen, A., 2004, "New approach to solve hybrid flow shop scheduling problems by artificial immune system," *Future Generation Computer Systems*, Volume 20, Issue 6, pp. 1083-1095.

Farmer, J. D., Parkard, N. H. and Perelson, A. S., 1986, "The Immune System, Adaptation, and Machine Learning", *Physica*, 22D, pp. 187-204.

Farinelli, A., Iocchi, L. and Nardi, D., 2004, "Multirobot systems: a classification focused on coordination", *IEEE Transactions on Systems, Man and Cybernetics*, Part B, Volume 34, Issue 5, pp. 2015 – 2028.

Floreano, D. and Nolfi, S., 1997, "Adaptive Behavior in Competing Co-Evolving Species," *Proceedings of the Fourth Conference on Artificial Life*, pp. 378-387.

Forrest, S., Perelson, A. S., Allen, L. and Cherukuri, R., 1994, "Self-Nonself Discrimination in a Computer", *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp. 202-212.

Goldberg, D. E., 1989, *Genetic Algorithms in search, Optimization and Machine Learning*, Addison-Wesley, Reading, Massachusetts.

Goldberg, D., 1996, "Heterogeneous and homogeneous robot group behavior," *Proceedings of the National Conference on Artificial Intelligence*, Vol. 2, p. 1390.

Haykin, S., 1994, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York.

He, Y., Hui, S. C. and Lai, E. M., 2005, "Automatic Timetabling Using Artificial Immune System," *AAIM 2005*, LNCS 3521, pp. 55-65.

Holland, J. H., 1986, “Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule Based Systems,” *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, pp. 593-624.

Iocchi, L., Nardi, D. and Salerno, M., 2001, “Reactivity and Deliberation: A Survey on Multi-Robot Systems”, *Balancing Reactivity and Deliberation in Multi-Agent Systems* (LNAI 2103), pp. 9--32.

Ishida, Y. and Adachi, N., 1996, “Active Noise Control by an Immune Algorithm: Adoption in Immune System as an Evolution,” *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pp.150-153.

Jerne, N. K., 1973, “The Immune System”, *Scientific American*, 229, pp. 52-60.

Jerne, N. K., 1974, “Towards a network theory of the immune system”, *Ann. Immunol*., 125C, pp. 373-389.

Jones, C. V., 2005, *A Principled Design Methodology for Minimalist Multi-Robot System Controllers*, PhD Dissertation, Univ. of Southern California, California.

Jones, C. V. and Mataric, M. J., 2005, "Behavior-Based Coordination in Multi-Robot Systems", *Autonomous Mobile Robots: Sensing, Control, Decision-Making, and Applications*, Sam S. Ge and Frank L. Lewis, eds., Marcel Dekker, Inc..

Jun, H. and Sim, K., 1997, “Behavior Learning and Evolution of Collective Autonomous Mobile Robots based on Reinforcement Learning and Distributed Genetic Algorithms,” *Proceedings of the 1997 IEEE International Workshop on Robot and Human Communication*, pp.248-253.

Kelly, I. D. and Keating, D. A., 1998, “Increased Learning Rates Through the Sharing of Experiences of Multiple Autonomous Mobile Robot Agents,” *Proceedings of the 1998 IEEE International Conference on Fuzzy Systems*, pp.129-134.

Kim, J. and Bentley, P., 2001, “An evaluation of negative selection in an artificial immune for network intrusion detection,” *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1330-1337.

Kodjabachian, J. and Meyer, J., 1995, “Evolution and development of control architectures in animats”, *Robotics and Autonomous Systems*, 16(2-4).

Kennedy, J. and Eberhart, R. C., 1995, "Particle swarm optimization", *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942-1948.

Kondo, T., Ishiguro, A., Watanabe, Y., Shirai, Y. and Uchikawa, Y., 1998, "Evolutionary Construction of an Immune Network-Based Behavior Arbitration Mechanism for Autonomous Mobile Robots," *Electrical Engineering in Japan*, Vol.123, No.3, pp.865-873.

Krautmacher, M. and Dilger, W., 2004, "AIS Based Robot Navigation in a Rescue Scenario," Lecture Notes in Computer Science, Volume 3239, pp. 106-118.

Lee-Johnson, C. P., 2004, "The Development of a Control System for an Autonomous mobile Robot" University of Waikato.

Liu, J., Wu, J. and Tang, Y.Y., 1998, "On Emergence of Group Behavior in a Genetically Controlled Autonomous Agent System," *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp.470-475.

Luh, G. and Liu, W., 2004, "Reactive Immune Network Based Mobile Robot Navigation," Lecture Notes in Computer Science, Volume 3239, pp. 119-132.

Maeda, Y., 1997, "Behavior Learning and Group Evolution for Autonomous Multi-Agent Robot", *Proceedings of the 1997 IEEE International Conference on Fuzzy Systems*, pp.1355-1360.

Manning, M. J., 1979, "Evolution of the vertebrate immune system," *J R Soc Med*, 72(9), pp. 683–688.

Mataric, M. J., 1994, *Interaction and Intelligent Behavior*, Ph.D. Dissertation, MIT, Massachusetts.

McLurkin, J. D., 1996, "Using Cooperative Robots for Explosive Ordnance Disposal," Technical Document, MIT, Artificial Intelligence Lab, MA.

Mitsumoto, N., Fukuda, T., Arai, F. and Ishihara, H., 1997, "Control of the Distributed Autonomous Robotic System based on the Biologically Inspired Immunological Architecture," *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pp.3551-3556.

Okura, M., Ogura, A., Matsumoto, A., Ikeda, H., Islam, M.M. and Murase, K.,

2003, “Chaotic dynamics in evolutionary autonomous mobile robot and fitness evaluation with complexity measure,” *SICE 2003 Annual Conference*, Volume 3, Issue 4-6, pp. 2847-2851.

Ong, Z. X., Tay, J.C. and Kwoh, C.K., 2005, “Applying the Clonal Selection Principle to Find Flexible Job-Shop Schedules,” Lecture Notes in Computer Science, Volume 3627, pp. 442-455.

Pipe1, A.G. and Carse1, B., 2002, “ First Results from Experiments in Fuzzy Classifier System Architectures for Mobile Robotics,” Lecture Notes in Computer Science, Volume 2439, pp. 578-587.

Pirjanian, P., 1998, Mutiple Objective Action Selection Behavior Fusion Using Voting, Ph.D. Dissertation, Aalborg University, Denmark.

Ross, T. J., 1995, *Fuzzy Logic with Engineering Applications*, McGraw-Hill, New York.

Schultz, A., Grefenstette, J. and Adams, W., 1996, “RoboShepherd: Learning Complex Robotic Behaviors,” *In Proceedings of the International Symposium on Robotics and Automation*, pp. 763-768.

Stolzmann, W., 1999, “Latent learning in Khepera robots with Anticipatory Classifier Systems,” *International Workshop on Learning Classifier Systems (2.IWLCS) on the Genetic and 22 Evolutionary Computation Conference*, pp. 290-297.

Sycara, K., Decker, K., Pannu, A., Williamson, M. and Zeng, D., 1996, “Distributed intelligent agents,” *IEEE Expert*, 11(6), pp. 36-46.

Vargas, P. A., de Castro1, L. N., Michelan, R. and Von Zuben, F. J., 2003, “An Immune Learning Classifier Network for Autonomous Navigation,” Lecture Notes in Computer Science, Volume 2787, pp. 69-80.

Webb, A., Hart, E., Ross, P. and Lawson, A., 2004, “Controlling a Simulated Khepera with an XCS Classifier System with Memory,” Lecture Notes in Computer Science, Volume 2801, pp. 885-892.

Vita

Jaeho Hur was born in Seoul, Korea on May 14th, 1965, the son of Mr. Baik Hur and Mrs. Hyunkyung Song. He was admitted to the undergraduate program in Mechanical Engineering at the Seoul National University, Seoul, Korea in March, 1983 and received the degree of Bachelor of Science in February, 1987. He entered the graduate program in Mechanical Design and Production Engineering at the Seoul National University, Seoul, Korea in March, 1987 and received the degree of the Master of Science in February, 1989. After he finished his military service as a second lieutenant in 1990, he worked as a research scientist at the Korea Institute of Science and Technology, Seoul, Korea. In August, 1992, he started the program for Doctor of Philosophy in Mechanical Engineering at the University of Texas at Austin.

Permanent address: 108-307, Mi-Do, Dae-Chi 2 Dong, Gang-Nam Gu, Seoul, Korea, 135-307.

This dissertation was typed by Jaeho Hur.