# FAULT TOLERANT DEPLOYMENT, SEARCH, AND TASK COOPERATIVE CONTROL OF ROBOT/SENSOR NETWORKS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

MIDDLE EAST TECHNICAL UNIVERSITY

BY

BERKANT AKIN

IN PARTIAL FULFILLMENT OF THE REQUIRMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2005

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet Erkmen

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Aydan Erkmen

Co-Supervisor

Prof. Dr. İsmet Erkmen

Supervisor

Examining Committee Members

Prof. Dr. Erol Kocaoğlan (Chairman),(METU, EE) _____

Prof. Dr. İsmet Erkmen, (METU, EE) _____

Prof. Dr. Aydan Erkmen, (METU, EE) _____

Assist. Prof. Dr. Afşar Saranlı, (METU, EE) _____

Dr. Tolga Sönmez, (TÜBİTAK-SAGE) _____

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last name : Berkant Akın

Signature :

# ABSTRACT

## FAULT TOLERANT DEPLOYMENT, SEARCH AND TASK COOPERATIVE CONTROL OF ROBOT/SENSOR NETWORKS

Akın, Berkant
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. İsmet Erkmen
Co-Supervisor: Prof. Dr. Aydan Erkmen

September 2005, 182 pages

This thesis focuses on developing of a distributed, efficient and fault tolerant multiresolutional architecture for sensor networks. For demonstrative purpose, a powerful simulation environment using 3D environment model has been developed. The robot network is composed of autonomous robots capable of working cooperatively equipped with single typed simple sensor. The developed layered control architecture is hybrid including both subsumption and motor schema control strategies. In this proposed control method, behaviors in different or in same layer are coordinated with an evaluator unit that overcomes the difficulties of subsumption based architectures in terms of behavioral coordination. The final coordination between these layers is achieved cooperatively. We performed many simulation experiments to test robot deployment, search and task execution. It is shown that some important parameters such as target reaching time, energy consumption, and

communication range can be optimized if an approximate prior information about the environment is known. Robots executes task based on a task allocation algorithm. Market based auction method is used as a task allocation algorithm with completely different robot fitness evaluation method allowing a distributive problem solving. Six non-linear fitness functions are developed to increase the fairness, and fault tolerance of task allocation. These functions have been tested to represent the successes and failures of robots in a compact form. Performance analyses test results have shown that fairness increases two times more in task allocation when these fitness functions are used, compared to the results existing fitness evaluation methods used in the market based auction algorithms. Moreover, fault tolerance is increased by using fitness functions devoted to failure conditions.

Keywords: Robot/sensor network, behavior based robotics, market based auction method, layered-hybrid control

# ÖZ

## ROBOT/ALGILAYICI AĞLARINDA HATA TOLERANSI YÜKSEK KONUŞLANDIRMA, ARAMA VE KOOPERATİF İŞ YAPMA DENETİMİ

Akın, Berkant
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. İsmet Erkmen
Yardımcı Tez Yöneticisi: Prof. Dr. Aydan Erkmen

Eylül 2005, 182 sayfa

Bu tez dağınık, robot / sensör ağları için etkin, hata toleransı yüksek ve çok çözünürlüklü mimari tasarımını içermektedir. Gösterim amacı için etkin, 3 boyutlu çevre modeli kullanan bir benzetim ortamı geliştirilmiştir. Ağdaki robotlar otonom ve birlikte görev yapabilecek yetenektedirler. Ayrıca robotlar sadece bir tür basit sensörlerle donatılmıştır. Kontrol mimarisi olarak katmanlı-melez bir yapı geliştirilmiştir. Melez yapı hiyerarşik-öncelik tabanlı (subsumption) ve motor yaklaşımlı (schema) mimarilerinin birlikte kullanılmasından oluşmaktadır. Hiyerarşik-öncelik tabanlı mimarinin davranış koordinasyonuna ilişkin dezavantajlarını aşmak için farklı ya da aynı katmadaki davranışlar, değerlendirici denilen yeni bir birimle koordine edilmiştir. Farklı katmanların koordinasyonu ise kooperatif bir şekilde yapılmaktadır. Benzetimci kullanılarak, önerilen kontrol

mimarisi bir çok kapsamlı deneye tâbi tutulmuştur. Konuşlandırma, arama, davranışlar bir çok durum için test edilmiştir. Benzetim sonuçları, bazı önemli parametrelerin, örneğin hedef yakalama süresi, enerji tüketimi, haberleşme menzili, optimum değerlerinin bulunabileceğini göstermiştir. Bunun için tek koşul, ortam hakkında bazı yaklaşık verilerin bilinmesidir. Robotlar, görevleri belirli bir görev paylaşımı algoritmasına göre çözerler. Görev paylaşım yöntemi olarak dağınık problem çözmeye izin veren pazar tabalı açık artırma algoritması kullanılmıştır. Robot uygunluk hesabı tamamen farklı bir metoda göre yapılmaktadır. Adil görev dağılımı ve hata toleransı artırımı için 6 değişik uygunluk fonksiyonu geliştirilmiştir. Bu fonksiyonlar, başarı ve hata durumlarını oldukça yalın bir formda ifade etmektedirler. Geliştirilen uygunluk fonksiyonları, şu an pazar tabanlı açık artırma metodunda kullanılan uygunluk fonksiyonuna göre 2 kat daha adil görev dağılımı sağlamaktadır. Sistemin hata toleransı da hata durumları için ayrılan uygunluk fonksiyonları kullanımıyla artırılmıştır.

Anahtar Sözcükler: Robot/sensör ağları, davranış tabanlı robotic, pazar tabanlı açık artırma, katmanlı melez kontrol

# ACKNOWLEDGEMENTS

I wish to express my special thanks my supervisor Prof. Dr. İsmet Erkmen, and my co-advisor Prof. Dr. Aydan Erkmen for their guidance, support and suggestions.

I also wish to thank my colleagues at TÜBİTAK SAGE, especially to Dr. A. Pınar Koyaz, Dr. Tolga Sönmez, and A. Galip Yıldırım for their support and advices.

Beside, I would like to thank my family for their love and support. I also would like to thank my friends for their encouragements and support.

Finally, the support provided by TÜBİTAK-SAGE to this thesis is greatly acknowledged.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# Introduction

## 1.1 An Emerging Field in Robotics: Robot Networks

Multi-robot systems (MRS) have led to challenging contemporary research field of robot networks where intelligence is the recent focus. An important reason behind this popularity is the acute attention brought to the field by military surveillance needs and the capabilities of highly distributed systems. Developments in multi-robot systems are depended on many parameters.

Uncertainly handling was necessitated by application in unstructured environment and understanding and modeling intelligence in nature was the recent focus if MRS and robot networks.

The need for MRS systems more recently that of robot networks rise from technological advancements, modular, distributed architectures and equipment [1]. These advancements have occurred in

- Autonomous modular robotics
- Computational capabilities
- Flexible architecture developments
- Communication capabilities
- Analyses of complex systems

Multi-robot systems generally based on architectures enabling collective behaviors are preferred [2] over others because:

1. Tasks may be inherently too complex (or impossible) for a single robot to accomplish, or performance benefits can be gained from using multiple robots

2. Building and using several simple robots can be easier, cheaper, more flexible and more fault tolerant than having a single powerful robot for each separate task

3. The constructive, synthetic approach inherent in cooperative mobile robotics can possibly yield insights into fundamental problems in the social sciences (organization theory, economics, cognitive psychology), and life sciences (theoretical biology, animal ethology, biological inspirations).

Progress can be also seen in research all around the world. The number of articles about the MRS increased significantly. In Figure 1, number of articles vs. year is shown [8].

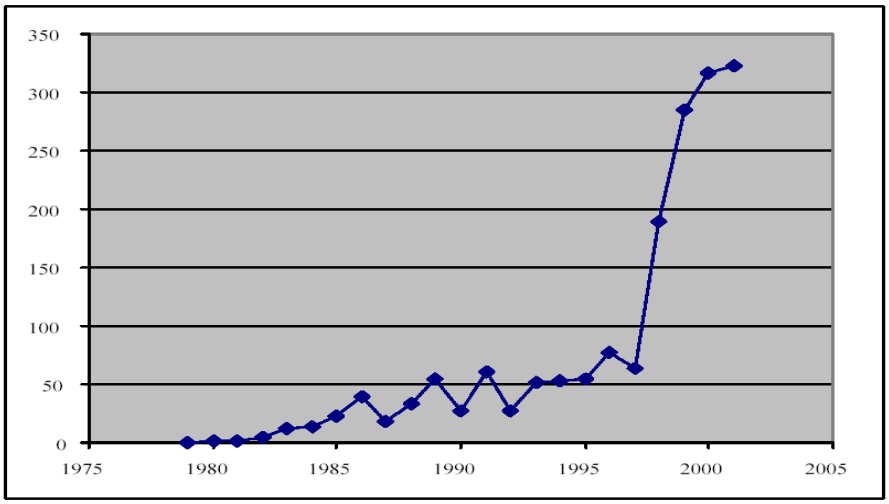


**Figure 1, Number of articles versus year [8].**

MRS covers the following applications:

- Tasks requiring corporative working
- Space Exploration

- Mapping
- Surveillance & Reconnaissance
- Hazardous waste clean up and mine removal

Actually targeted future of MRS applications gives a live feeling about the future impact of this field. Within 30 years some people aim at football matches between robot players and human players. For some of us, this future guess may be too overwhelmed but trend in the robotics applications is towards this kind of hard to believe robots. Soldiers, future workers will probably be robots.

## 1.2 Motivation of Thesis

Sensor network as a hyper multi-robot system is a recent focus in robotics, driven by the motivation that a network of intelligent simple agents can do many works more rapidly, and precisely based on “divide and conquer concept”. Moreover sensor networks can be used for executing tasks having potential danger or impossible for humans. Robustness is easily achieved by graceful degradation due to hyper redundancies of network modules where if one fails others can assume its role within the mission.

There are many examples of fields for which sensor networks should be used. Space exploration in unknown environments (on planets) where teams of autonomous robots explore, covering very large terrain, and send the exploratory data back to the base station. Surveillance & Reconnaissance (S&R) applications are also popular and progressing application field. For some S&R applications availability of humans may be very difficult, and in some situations even very dangerous for humans. Cleaning up, and mine removal are other application of sensor networks.

There are many topics left open in the MRS and sensor network research fields. Although there exist so many representational and architectural problems in MRS a powerful metric for measuring intelligence or any other types of quantities is

still lacking. Currently modeling of the MRS and especially sensor networks applications, evaluation of system performance and deployment of the intelligence are the main open issues.

Motivations behind this thesis can be listed as follows:

- In sensor network applications, there are architectural issues. Effective, stable, and robust architectures may increase performance of sensor network significantly. For this purpose a new architecture is designed by inspired from existing methods.

- Modeling of a multi-robot system is an extremely difficult task. Because there exist many parameters, and uncertainties crossly coupled with each other. Moreover, environmental conditions may not be predicted. Therefore statistical approaches are more suitable. To analyze the nature of a desired sensor network a simulation environment is necessary. This thesis also covers the implementation of a simulator. Using realistic simulators, non-linear models for optimization of system parameters can be developed. Optimal or sub-optimal models can increase efficiency considerably. We designed a 3D simulation environment to increase the reliability of simulations.

- Task allocation is another important open issue in achieving powerful cooperation among agents of sensor network. Efficient metrics for performance measurement is quite important in task allocation. This is another motivation of this thesis. We aimed to generate an analytical fault tolerant performance metric for task allocation.

## 1.3 Objective, and Goals

The main objective of thesis is designing a fault-tolerant, behavior based architecture for team of heterogeneous or homogenous agents acting in a sensor network. The proposed sensor network architecture is a platform for defining, and solving tasks cooperatively or autonomously. Robots' tasks are detection, executing

specific missions, and undertaking information fusion in a bounded, dynamic, obstacle-dense, and unstructured environment. Our generated architecture can be used for various tasks requiring cooperative working such as hazardous waste detection & cleaning, reconnaissance & surveillance, mapping, and exploration.

Efficiency, robustness, and fault tolerance are the main goals of our approach. These goals are achieved in the performance analyses conducted on the developed architecture. Since system components are highly correlated with each other and thus any change in any component affects the global performance significantly.

Moreover, issues regarding modeling are also considered. Modeling may include modeling of a behavior of an entire system or just a single parameter affecting performance significantly. For this purpose a simulation environment is developed. By using this simulator, proposed cooperative sensor network architecture can be tested for various conditions for very long duration. Simulation results are used to generate appropriate non-linear models for important system parameters such as range of communication.

Another important aspect of simulator is that the environment is modeled in 3 dimensional space enabling robots to undergo more realistic motion in the simulated environment. The 3D environment modeling allows implementation of the entire dynamics of robot as well as the dynamics of the robot team.

There are drawbacks of designing, and testing the proposed architecture in a simulation environment, because it is extremely difficult to model the entire physical world. There are many parameters to be considered about the physics of the robot, and environment. Poor modeling may change correct evaluation of proposed architecture. However, for high level analysis of the architecture in terms of task allocation efficiency, fault tolerance performance, simulation environments can be preferred.

## 1.4 Methods

A sensor network implementation encompasses many topics that have to be modeled, developed, and implemented, which can be classified as follows:

- System architecture
- Path planning
- Communication
- Task allocation

### 1.4.1 Architecture

Architectural issues cover how each robot in the network achieves their basic functions, and interact with other robots. Moreover, the implementation of such robot functions, and the coordination of these functions should also be investigated. For instance, a robot should avoid obstacles, and other robots to wander securely in the environment. Therefore the architecture provides not only the implementation of basic behaviors, but also their coordination.

To design a robust, efficient, and fault tolerant architecture, a behavior based approach is preferred. Reader is referred to [1] for basic information about behavior based robotics. Efficiency, robustness, and fault tolerance as design parameters can be achieved by developing capability measures of the behaviors compatible with this goal. For instance, efficiency of target detection is increased by adaptive wander behavior, whereas robustness of target detection process increased by heuristic wander behavior.

Our developed behavior based architecture is a hybrid of both motor schema, and subsumption based architecture. There is an additional unit called evaluator which is used to coordinate

- The equal priority behaviors in subsumption layer.
- The behaviors in motor schema layer with behaviors in subsumption layer

There are two types of behaviors in our proposed system: external and internal behaviors. External behaviors evaluate the information coming from external world via sensors, whereas internal behaviors monitor the activity of external behaviors to coordinate them accordingly. Internal behaviors behave as coordinator and interpreter of external behaviors. Coordination of behaviors is as crucial as implementation of behaviors. The nature of implemented coordination makes the system hybrid. Choosing this hybrid architecture is mainly due to the need for efficiency, and robustness. Well-known control strategies show poor performance in complex environments. Our hybrid architecture is aimed to generate an efficient and robust behavioral coordination even in very complex environments without degrading the simplicity and reactivity of the system.

### 1.4.2 Path Planning

In group behavior, path planning is another critical issue in sensor networks. Each robot needs to plan its path to reach a desired location rapidly, and securely while sensory system of robot detects the set of obstacles, $O$, other robots $R$, and target points, $T$. Robot path planning strategy considers all of these sets to reach target point $t_1$ while considering its role in the global task allocation.

There are different path planning approaches in the literature. The most popular ones being:

- Graph based approaches
- Potential fields

In this thesis, we preferred potential fields, because it is

- Suitable for both 2D & 3D environments
- Easy to compute, it does not require any search process
- Fast

Each behavior generates a 3D dimensional force field representing its desired direction in the 3D environment.

There is an important drawback of potential field used for path planning which is that of local minima problem. To avoid this problem, many techniques exist in literature. In our proposed architecture, we avoid local minima by adding noise continuously to the sum of potential field of behaviors in the architecture.

### 1.4.3 Coverage

Coverage is a part of performance of the search process. In this thesis, coverage is aimed to be increased while searching or exploring targets in the environment. There are two behaviors implementing search methods: Heuristic and adaptive wander behavior. Heuristic wander behavior is completely random. It does not take any information form the sensors. On the other hand adaptive wander uses coverage map, and detected obstacles' location information to generate next wander point. This behavior is simple, and also contains randomness but it requires additional memory for maps.

Coverage map contains last location of robot within coverage map timeout $TO_{\text{cov}}$. Obstacle map contains location of obstacles avoided within a predefined time frame called obstacle map timeout. Adaptive wander behavior generates a next wander point which is not covered within $TO_{\text{cov}}$. which further helps to decrease obstacle avoidance rate.

Time varying maps are used because

- Dynamic environment assumption is not violated. Reliability of information is time-limited.
- Memory demanded is decreased for these maps

We determine a value of the time frame that optimizes the coverage and memory usage experimentally using our simulator.

### 1.4.4 Communication

Communication is one of important issue in multi-robot system. Agents in MRS should communicate with each other in three ways:

- Interacting via sensing
- Interacting via the environment
- Interaction via direct communication (for instance short range radio communication)

Interacting via sensing, and environment can be grouped together as implicit communication. On the other hand, robot can explicitly communicate with each other using communication hardware.

Detection of other robots or some signs related with tasks in the environment is another way of communication. This kind of communication can be very effective if the nature of task is convenient.

In proposed architecture, explicit communication is used. Assumptions about communication are:

- Robots can communicate within a user defined range.
- Infinite communication bandwidth is available.
- Usage of communication depends on the task allocation algorithm
- Communication is perfect except for error simulations

In real world, usage of communication brings considerable cost. It requires higher energy consumptions. Communication cost increases exponentially as communication range is increased. So, unnecessary usage of communication decreases the system performance in case of limited energy. For this purpose, extensive numbers of simulations are done to find optimal communication range. Target reaching time is measured for various numbers of targets, tasks and communication ranges. These simulation results are fed to a neural network to generate an acceptable model.

### 1.4.5 Task Allocation, Definitions

Task allocation is the heart of the collaborative behavior. Task allocation algorithm depends on the type of control system which can be centralized, distributive, and hierarchical. In centralized systems, tasks allocation management is determined by a specific agent and all computations including task allocation is achieved by this agent. On the other hand, in distributed systems, there is no central control, computations or controls are achieved by individual team members distributively. In a hierarchical system, control of the system is achieved in a prioritized order. For instance, roles of the agents can be elected by entire team members, and roles can be altered by repeating elections within some time interval generated by vote number priority order.

There are advantages, and disadvantages of different control methods. Centralized approach enables development of global solutions, but its fault tolerance is poor. If the robot responsible for central control is corrupted then entire system may fail. Central control requires long communication ranges which cannot be accepted.

Current trend in the world is towards to decentralized systems. Although, distributed system may not reach global solutions, its fault tolerance is very high as compared with that of centralized systems. Moreover, problems having high computational complexities can be achieved distributively among the team members. This may decrease the need for powerful hardware for complex computations.

Our proposed architecture is a distributive system. Task allocation is achieved by a market based auction algorithm [51]. Each robot has an ability of allocating, and executing tasks if its fitness is sufficient. If a robot detects a task requiring collaborative work of different robots, then it starts the task allocation process where tasks are allocated among robots with respect to their fitness value.

Goals of task allocation algorithm are to archive fair task allocation among robots functioning perfectly, and making the system fault tolerant. These goals are tried to be achieved by fitness calculation methods. In the proposed system, there are two types of time dependent exponential fitness functions

- Function for success situation
- Function for failure situation

There are 6 fitness functions implemented. Functions for success situations are target reaching frequency, obstacle avoidance success frequency, coverage, and distance fitness. Functions for success situations are communication, and obstacle avoidance failure frequency. It is showed that, introduction of these fitness functions enables fair task allocation, and fault tolerance of system. Fault tolerance is tested by artificial error simulations.

Representation of tasks in simulation environment is also an important issue. For this purpose, a formal way is developed for representation of tasks. Agents should know how to decompose a task into sub-tasks to be solved cooperatively.

In the proposed system there are four different tasks:

- Uncorrelated tasks solved by a single robot.
- Correlated tasks with separate robot asynchronously solving the task
- Synchronously correlated tasks, done by inseparably by multi-robots in a synchronous manner.
- Sequentially correlated tasks, done inseparably by multi-robot doing the task sequentially.

Any task can be represented as a combination of the above task types. This kind of task decomposition enables the defining tasks in the simulator even for very complex tasks.

## 1.5 Contributions of Thesis

In this thesis, the main contributions are in the architecture, behavior fusing nodes called evaluators, environment simulation, and task allocation fitness functions. Contributions can be listed as follows

- Hybrid, fault tolerant, robust and efficient behavior based architecture is developed. In this architecture, fault tolerance, efficiency, and robustness are achieved by implementing appropriate behaviors, and behavioral coordination.

- Hybrid architecture is composed of two layers: subsumption layer and motor schema layer. Evaluators are designed to coordinate the behaviors in subsumption layer, and behaviors in different layers. Evaluator concept helps for overcoming classical subsumption architecture problems by increasing the behavioral coordination.

- A powerful simulator is developed. Using this simulator, many situations in a 3D environment model are simulated for different system parameters. Path planning is achieved in 3 dimensional spaces. This is not common in robotic simulators.

- Six important fitness parameters are developed for measuring performance of the robot. Fitness parameters are evaluated using exponential time-dependent fitness functions. Rate of increase & descent and maximum value of fitness parameter are the most important parameters of the fitness functions. Main goals of these fitness functions are obtaining fair, fault tolerant task allocation.

## 1.6 Outline of Thesis

This thesis contains 6 chapters. Chapter 1 is introduction to thesis. In Chapter 2, literature survey about sensor networks is presented. Simulation environment is described in Chapter 3. Chapter 4 is devoted to the proposed method. This chapter contains description, and implementation of proposed behavior based architecture, behaviors, task allocation algorithm, and fitness calculations. In Chapter 5, results of experiments are given. Experiments are done to analyze the individual performance of behaviors, coverage & target reaching performance, communication range estimation, task allocation performance regarding fault tolerance, and fairness. Conclusion, and future studies are given in Chapter 6.

# CHAPTER 2

## Literature Survey

Multi-robot systems applications can be decomposed into many sub-fields. The important subfields are:

- **Architecture, task planning, task allocation and control**
- **Biological inspiration**
- **Localization, mapping, and exploration**
- **Path planning or motion planning**
- **Learning**
- **Communication**
- **Motion coordination, and formations**
- **Reconfigurable robotics**

All of the fields listed above have many open research areas. In Figure 2, number of papers versus years is shown according to citation index for physics, electronics, and computing (INSPEC). For recent year publications numbers are growing with increasing speed.

In this part, literature survey will mainly focus on thesis subjects.

- Sensor Networks
- Architectural issues
- Learning
- Multi-robot task allocation
- Path planning, coverage

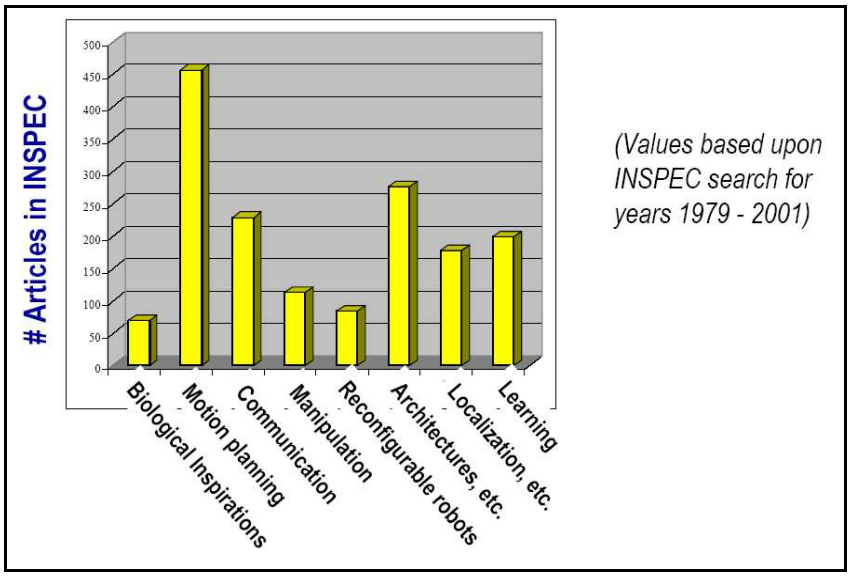


**Figure 2, Articles distribution between 1979 -2001**

## 2.1 Sensor Networks

Sensor network is an emerging field of research where robots equipped with single typed simple sensors are deployed for search, surveillance, and rescue. Sensor nodes have been designed as small as micro electro mechanical system (MEMS) and this has boosted the development of the sensor network applications considerably [2], [3]. Mass production of sensor nodes based on MEMS technology allows realization of sensors networks having a huge number of nodes with relatively small cost, making such networks extendable, fault tolerant, and self organizing. With such a network of nodes fusion of information becomes un-crucial issue to be dealt with high reliability. Application of sensor networks can be classified as follows:

- Military applications
- Environmental applications
- Home applications
- Health applications

Military application covers military command & control, intelligence, surveillance, reconnaissance, environment monitoring. Sensor networks have found

ardent interest because of their fault tolerance and scalability becoming the main drive of military application.

Environmental application is another hot application area of sensor networks. Meteorology, flood detection, agriculture, fire detection, pollution analysis, are animal observations are main topics in this area. In [4], long term analysis of flood based on a distributive approach using mobile sensor networks. CORIE (Columbia River) for vessel transport and ALERT (automated local evaluation in real time) [5] for air rainfall and water analysis are popular examples of sensor networks.

Sensor network topology, fault tolerance, cost, scalability, communication medium, power consumption are the main design constraints of an sensor network application. Deployment of sensor network is one the important issue because it affects the quality of collected information, fault tolerance, and speed of the fusion of information. Deployments of the nodes are generally fixed or variable. Variable deployment enables self organizing networks. Deployment should be based on some statistical data, preserving the network topology [6], [7].

## 2.2 Multi-Robot Systems and Their Architectural Issues

Multi-robot system defines how robot makes its functions, and how robot interacts with other robots, and environments. MRS is inherently a distributed system. The differences of MRS from other distributed systems are that MRS is highly dynamic, spatially limited or bounded and interacting with many environmental or physical conditions [1]. But MRS developers can examine other distributed systems, architecture to adapt or imitate the convenient parts of those to the MRS systems. In Figure 3, group architecture for MRS is given [1].

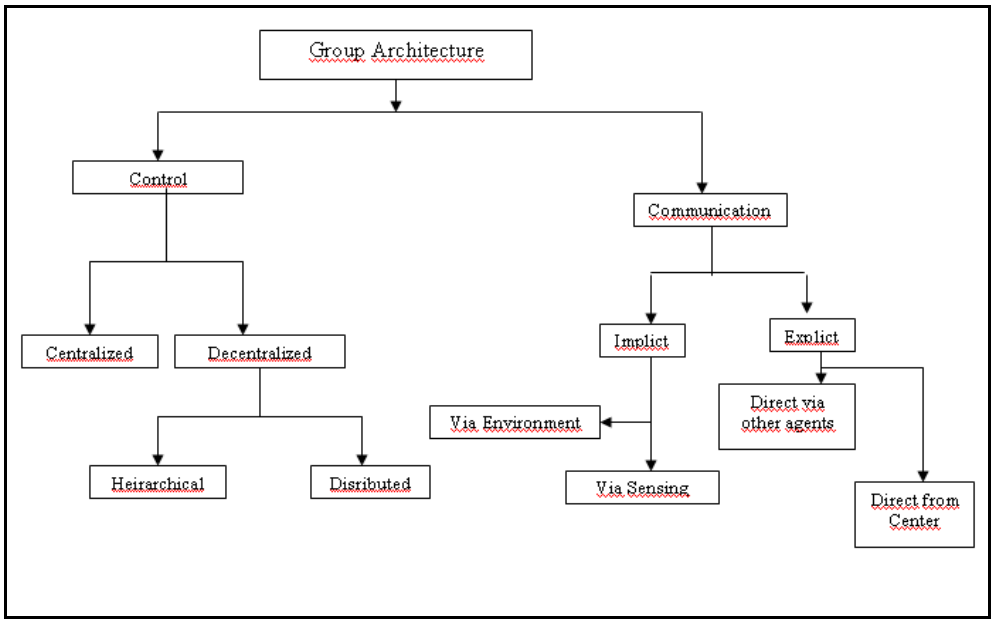


**Figure 3 General group architecture diagram**

There are valuable surveys on multi-robot systems with considering many sub-fields. In [1], a good survey on multi-robot system is given in terms of group architecture. Moreover [8], [9] are valuable references for current state art in multi-robot systems

In literature, robot control architectures can be grouped as **reactive, deliberative, hybrid, and behavior based control.** Depending on speed, and planning requirements, one of these control methods can be used [14].

Behavior based architectures are very popular in robotics applications. It was first developed by Rodney Brooks in 1980. Reasons behind this architecture are that it does not require any a priori external knowledge of world, and it is highly reactive. Reactivity can be expressed as "*planning is just a way of avoiding figuring out what to do next*" [12], i.e. in reactive systems planning of the next situations is considered as little as possible.

Behavior based robotics' architecture has three main units [10]:

1. Sensory Inputs

2. Behaviors
3. Motor Actions.

**Behavior** is the mapping of sensory inputs to a pattern of motor actions that are used to achieve a task. “A behavior is a reaction to a stimulus” [10], it wraps perceptual and motor actions, i.e. it should both include behavior triggering conditions and motor actions if it is activated.

There is a famous behavior based approach called subsumption architecture. Subsumption architecture defines layers of the augmented finite state machines (FSMs). Some behaviors can suppress, reset or inhibit other behaviors [11]. This architecture is a revolutionary approach in mobile robotics. It enables an incremental design strategy like object oriented approach in software engineering.

Motor schema as hybrid reactive approach also makes inspiration from brain theory, and psychology [13]. In motor schema there is no subsumption, suppression or inhibition. Response is generated as a sum of all active behaviors’ responses, which is reason for reactivity. Schematic of motor schema is given Figure 4 respectively.

Subsumption architecture defines a priority among the behaviors. At any instant, only one behavior can be active. On the other hand, in motor schema cooperation among behaviors exists. More than one behavior can be active at a time. This may be an advantage of motor schema approach [17].

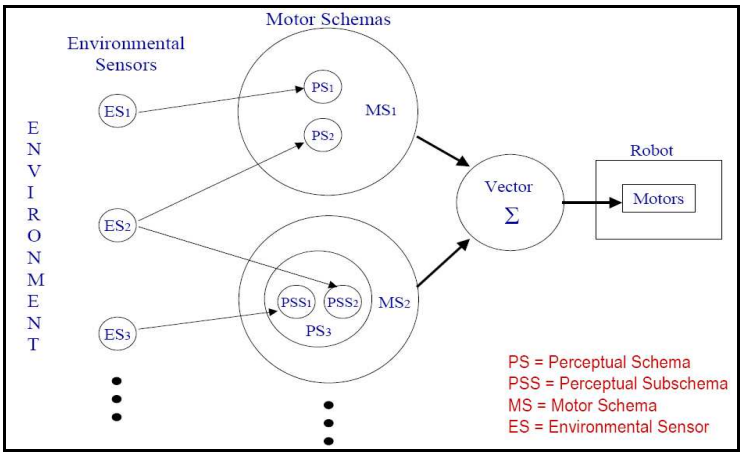


**Figure 4, Motor Schema architecture**

It is difficult to design complex system based on strict behavior based approach. To overcome this difficulty, hybrid architectures are developed [15], [16], [18].

There are architectures being hybrid of motor schema, and deliberative control. AuRA [19] is one of these architectures connecting motor schema and planners modules (high level mission planner, spatial reasoner, and plan sequencer). System is also reactive for changes requiring fast responses. Moreover AuRA [19] shown in Figure 5 contains parameter learning in run time.

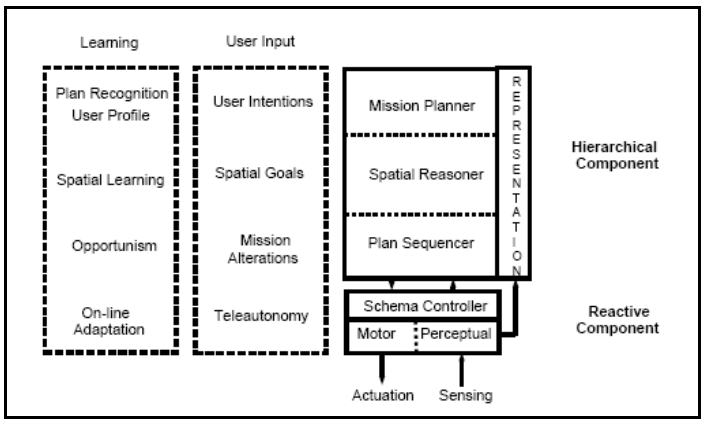


**Figure 5, AuRA high level architecture [19]**

Three layer architecture described in [18] is another example of hybrid control where layer are that of controller, sequencer, and deliberator. The controller layer implements reactive behaviors. The sequencer selects the behavior to be executed by the controller. Function of the deliberator is to evaluate time consuming algorithms using internal states.

Designing **multi-robot architectures** is also a hot topic in robotics. Multi-robot system is desired instead of single-robotics system due to many reasons. Some of them are [20]

- Distributed action to increase fault tolerance.
- Some tasks can be divided into different parts through task decomposition which can be handled by single robots.
- Single robot may be too complex having wide range of capabilities.

CEBOT is an early version of heterogonous multi agent architecture. Agents are capable of forming different assemblies. CEBOT can reconfigure the whole system depending on given tasks and environments and organize collective or swarm intelligence [21] [22].

ALLIANCE is a fault tolerant multi-robot architecture for heterogonous robots team. There are set of behaviors composed of high and low level behaviors. Motivational approach is used to activate different set of behaviors. This approach brings fault tolerance [23] [24]. L-ALLIANCE is another version of ALLIANCE. It allows adaptive update of internal parameters from past experience of robots. Modeling of other robots is also considered in case of cooperative work [25].

Examples of other multi-robot architectures are ACTRESS, SWARM, and GOFFER. Short descriptions about these architectures can be found in [1].

**Swarm intelligence** is an emerging field in multi-robot systems. Biological inspiration is a basis for behavior based robotics. Application of biological phenomenon to distributed robotics is preferable since biological organization is real

and working. Moreover it is known that most of the animals forming group behavior are not so clever to achieve observed tasks. But as a team, they get in complex dexterity. Another reason for biological inspiration is that a group of simple "non-intelligent" low level agents can show group intelligence. This is the key point behind social animals.

**Stigmergy** is the application of swarm intelligence to the MRS. It is used very effectively in much loosely coupled system. Stigmergy is "term used by some biologists to describe influence on behavior due to persisting the environmental effects of previous behavior" [26] [[27].

Many tasks are achieved by ants based on stigmergy. Ants secrete some pheromones. For example in finding shortest path between a point and nest can be solved by stigmergy. Ones an ant find this path, its pheromones will be distributed along this path. Since this is the shortest path among the possible paths, pheromone concentration along this path will be higher than that of other paths. Statistically, there will be a positive feedback to select this path. Ant colony approach was applied to mine detection problem with minimum mission completion time and successful results are obtained [28].

Fundamental works based on the biological inspiration are basically on insect colonies [29], [30], flocking, dispersing, aggregating, foraging [31], selfishness showing emergent behavior [32], predator-prey based systems [33].

## 2.3 Learning

**Learning** of parameters from interaction with the environment or other agents in the system increases the adaptivity of the system considerably. Any system is accepted as intelligent system if there is some kind of learning or adaptation. Popular learning methods are:

- Reinforcement Learning
- Neural Networks

- Evolutionary Learning

**Reinforcement learning (RL)** method is widely used in most of the robotic systems. In RL, a reinforcement signal, reward, is fed back to the system indicating the result of action. If a desired condition is occurred and also agent is behaved accordingly then positive reward is given, otherwise punishment is fed back. Reinforcement learning has been studied deeply and applied many robotic systems [34], [35], [36].

An important component of the RL based control system is the component responsible for evaluating the fitness of the response which is called "c**ritic**". Critic function generates the reinforcement signal, i.e. critic has learning capabilities. General structure of RL based system is shown in **Figure 6**. There are two types of dominant RL algorithm present: adaptive heuristic critic (AHC), Q-Learning

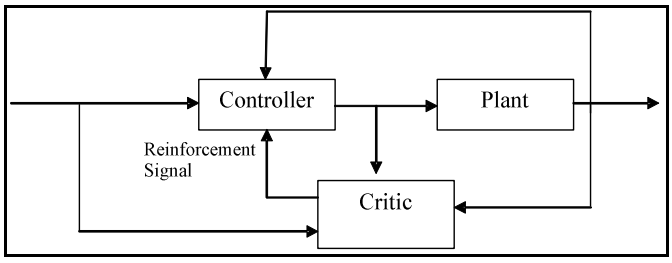


Figure 6, Reinforcement Learning System

**Neural networks** (NN) have been widely studied and applied many systems. NN is trained in a supervised or unsupervised fashion that synaptic weights are computed optimally so that difference between response to a set of stimulus and actual output is minimized. Synaptic weight update rule is different for distinct architectures. There are three main learning schemas in neural networks: classical conditioning, adaptive heuristic critic learning, and learning with associative memory.

Many researches have been made about application of NN to MRS. Fine and smooth adaptive action selection mechanics obtained using **Extended Kohonen**

**Maps**. Using this kind of mechanism is advantageous than other methods such as potential fields [37], [38].

Among the systems based on multi-layer perceptron (MLP), most impressive system is ALVINN (Autonomous Land Vehicle in a Neural Network) [39]. ALVINN can drive car autonomously at a speed of 60 Mph in a highway by a MLP based neural controller. In [40], a relatively simple single layer perceptron is used. As a training method **Widrow-Hoff** rule is selected. Depending on connection of neurons, behavior of the system is also analyzed. In [41] Hybrid evolutionary recurrent neural network controller is designed for secure navigation in the obstacle dense environments. A fitness space is defined and various fitness functions are evolved. Moreover fitness function contains both internal (behavioral) and external parameters.

## 2.4 Multi-Robot Task Allocation (MRTA)

Optimality of allocation determines local and global performance of the system. MRTA decides that which robot will perform which task in an optimal fashion. Moreover in order to increase immunity of system to faults, **distributed** problem solving is preferred to **centralized** approaches. MRTA should encourage the distributed problem solving.

In the MRS literature, formal analysis of architectures regarding the task allocation is highly incapable. Authors try to prove their system performance empirically. There is little work about formal analysis of the task allocation issues. In this part, definitions, taxonomy, and formal analysis of MRTA problems will be given based on [43] and [44]. Good surveys about MRTAs can be found in [1], [8], [42], and [48].

### 2.4.1 Definitions and Formal Analysis of MRTA

Task allocation deals with the assignment of tasks to the robots by minimizing a cost function or maximizing a utility function.

- **Task** is a sub-goal and also independent of other sub-goals. By independence, it cannot be a decomposed into any other sub-goals.
- **Utility** is a measure of specific action. When a task is assigned to a robot, utilty is the value of expected income from execution of this task. It is defined as formally difference between expected quality of task execution and cost of the resources and other things.

$U_{RT}$ : The utility of the assignment of task T to Robot R.

$Q_{RT}$ : The income of assignment of task T to robot R.

$C_{RT}$ : The cost of assignment of task T to robot R.

$$U_{RT} = \begin{Bmatrix} Q_{RT} - C_{RT} & Q_{RT} > C_{RT} \\ 0 & , Otherwise \end{Bmatrix}$$

Above equations show that a robot can be assigned to a task if expected income is larger than resource cost. Difficulty in MRTA is defining appropriate utility functions. Aim of the MRTA is given n robots and m tasks

$$R = \{r_1, r_2 ... r_n \}$$
$$T = \{t_1, t_2 .. t_m \}$$

Assign tasks to robots such that following metric should be maximized.

$$U_{total} = \sum_{i=1}^{n} \sum_{k=1}^{m} u_{ik}$$

Where

$u_{ik}$ is the utility of execution of task k assigned to robot i.

Types of MRTA problems are:

- **Single-task Robots (ST)** means robots can execute at most one task at a moment whereas **Multi-task Robots (MT)** can do multiple tasks at a moment.
- **Single-robot task (SR)** means each task can be achieved by only one robot whereas **Multi-robot task (MR)** requires more than one robot to execute a task.
- **Instantaneous assignment (IA)** stands for the no planning about the task allocation whereas **time extended assignment (TA)** a kind of planning exist.

MRS may have one of the following MRTA problems

- ST-SR-IA , ST-SR-TA
- ST-MR-IA, ST-MR-TA
- MT-SR-IA, MT-SR-TA
- MT-MR-IA, MT-MR-TA

Many MRS has a MRTA mechanism ST-SR-IA or ST-SR-TA, it is the easiest type of **optimal assignment** (OAP) allocation problems. **Greedy** algorithm generates optimal solution for this kind of allocation problems. ST-MR-IA, ST-MR-TA problems can be reduced to the **set partitioning problem** (SPP). Moreover MT-MR-IA, MT-MR-TA can be classified as **set covering** problems (SCP) and it is the most difficult problem among the MRTAs and its complexity is **NP hard**. . Solution of various types of MRTA problems can be solved with the help of literature of optimization theory and set theory. Some of those problems can be reduced to equivalent linear programs.

**Optimal Assignment Problem** is formulated [43]. as find mn negative integers $\alpha_{ij}$ maximizing

$$U = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} U_{ij} w_j$$

subject to

$$\sum_{i=1}^{m} \alpha_{ij} = 1, \quad 1 \leq j \leq n$$

$$\sum_{j=1}^{n} \alpha_{ij} = 1, \quad 1 \leq i \leq m.$$

Constraints force the assignment problems to be achieved as single assignment problem. This is suitable for first type (ST-SR-IA, ST-SR-TA) of MRTA problems.

**Set Partitioning Problem** can be defined as: Given a finite set E, a family F of acceptable subsets of E, and a utility function u : F → $R_+$, find a maximum-utility family X of elements in F such that X is a partition of E. Moreover complexity of the SPP is strongly NP-hard.

**Set Covering Problem**: Given a finite set E, a family F of acceptable subsets of E, and a cost function c: F → $R_+$, find a minimum-cost family X of elements in F such that X is a cover of E.

### 2.4.2 Task Allocation Methods

There are five types of task allocation methods: **auctions based methods**, **motivation-based methods**, **mutual inhibition, and no allocation**.

**Auctions methods** are based on negotiations among the robots. Negotiation is achieved by a process called bidding. A robot (manager robot) sends requests to other robots (worker robots candidates) for their helps. Manager robot assigns the tasks according to a metric called fitness of the robots. There are two famous auction based methods.

**Contract net protocol (CNP)** [49] [50] is the oldest version of auction based methods. It implements a central bidding method.

**MURDOCH** [51] is market inspired negotiation based algorithm [9]. The task announcement and bidding procedure almost the same with that of contract net protocol but there is no central manager. System is completely distributed. Task is always assigned to the most capable robot, i.e. MURDOCH has greedy based task allocation mechanism. Task can be announced any time this one of the advantageous of MURDOCH. MURDOCH is well applied to team of heterogonous robots. Hierarchical task structure is used, i.e. a task is composed of a tree of subtasks. In this thesis, task allocation mechanism is based on MURDOCH but fitness calculation is much different than that of original. M+ [52] and CEBOT [53] are also using auction based task allocation methods.

ALLIANCE [23] [25] [24] uses **motivation-based** method for task allocation. Robots have internal motivation parameters for task allocation. There are two motivational parameters: impatience, and acquiescence. These motivations control task allocation by defining desire, and impatience. Moreover these mechanisms make ALLIANCE fault tolerant. Another motivational based task allocation method called “Affective” described in [48]. Main aim of the method is decreasing communication overheads.

In **mutual inhibition** [55], ”robots directly inhibit those around them being chosen for a task” [48]. It requires more commutation messages to make inhibition process. In [56] task allocation is archived dynamically based on swarm intelligence. Dynamic role assignment is implemented in [57]. Agents exchange their roles depending on the conditions. Computational complexity analysis of different types of task allocation methods can be found in [51].

## 2.5 Path planning, Coverage, and Exploration

**Path planning** defines an efficient way of navigation to a desired location without avoiding collision from obstacles or other robots. There are different path planning methods in literature:

- Graph based approaches
- Potential field

In this thesis, potential field method is taken as path planning method because of the earlier reasoning that we mentioned. Although graph based approach can reach optimum path planning, computational complexity of these algorithm is high as compared with that of potential field. Moreover, graph based approaches are prone to error in case of noisy data. Reader is referenced to works [58], [59] for approaches, and current state art in the graph based approaches.

### 2.5.1 Potential Fields

Potential field approach is widely used in many fields especially in path planning. In this proposed architecture, potential fields are used generating appropriate paths with avoiding collisions. Robot does not globally plan its path and is well suited for reactivity; it simply generates a path using recent information about environment via its sensors. Some kind of emergent path generation is obtained.

A potential field function can be any kind of function depending on the system. Field function defines eqi-potential lines in the space with equal distances from the center of reference point. If 3D space is assumed then potential field function will define a surface in 3D. Potential field force is computed by taking the gradient of the field function at a specified point. If a ball is located around a potential field surface then ball will move in the direction of the gradient at current point. How fast the robot will move is determined by the magnitude of gradient.

Below some of potential field functions and associated force function with respect to position are given.

$$\left.\begin{matrix} U = C \\ F = \nabla U = 0 \end{matrix}\right\}, \text{Constant potential filed}$$

$$\left.\begin{matrix} U = aX^T + c \\ F = \nabla U = a \end{matrix}\right\}, \text{Linear potential field}$$

$$\left.\begin{matrix} U = aX^T X + bX + c \\ F = \nabla U = 2aX + b \end{matrix}\right\}, \text{Quadratic potential field}$$

Potential field function should not necessarily be continuous functions, depending on application; even non-linear field functions should also be preferred. In Figure 7, force lines are shown.

$$F = a_i \frac{(x - x_{oi})}{|x - x_{oi}|^2}$$

where $x_{oi}$ the position of the reference point; $a_i$ is the factor determining sign of the field that is whether it is repulsive or attractive and relative strength.

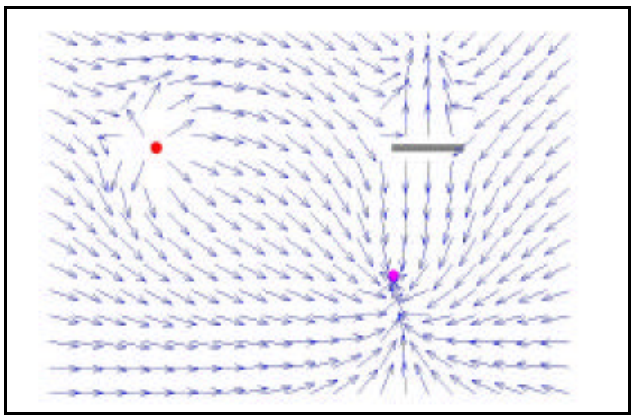

**Figure 7, Potential fieled forces generated by one attractive point, one repulsive point and one obstacle.**

Red point is evaluated as obstacle where as pink point is taken as target point. Potential field generates repulsive force with increasing magnitude around red circle. The potential field function can be obtained by integrating of force function.

In the literature, there are many works about potential fields. Potential field approach is well suited with the issue of coverage. In [60], [61], [62], [63], by using appropriate field functions coverage of entire system is increased. In [62], local minima problem of potential field is prevented by selecting the activity of field functions in run time. A new potential field is proposed such that target point is kept in the global minimum of total potential field function by adjusting the parameter carefully [64]. Moreover, in [65], potential field based path planning is implemented for autonomous underwater vehicle guidance.

### 2.5.2 Coverage

In path planning strategies, aim of the planning is reaching from start point to goal point. Coverage of the robot is not considered as a parameter to be optimized in path planning strategies. In this thesis, path planning is considered with coverage issue. A robot explores the environment by increasing coverage. Reader is referenced to [66] for taxonomy of coverage problems in the literature. There are works both considering path planning and coverage. A good survey on coverage algorithms is described in [67]. There are methods based on heuristic or non-heuristic (complete methods). Heuristic is a powerful tool if the robot does not know much thing about environment. Localization unit is not necessary if heuristic based search algorithm is adopted. Heuristic based approaches are more appropriate if cost effective and simple robots are used [68] [69] [70] [71].

If robot has a localization unit, then other type of methods can be utilized. These are

- Approximate cellular decomposition
- Semi approximate decomposition
- Exact cellular decomposition

Works and their summaries are given in [67]. Moreover, there are multi-robot versions of these coverage algorithms. In [72], an interesting solution is proposed for coverage, coverage, and sensor deployment. Robots have no information about environment. They are routed by sensor network to least visited cells. Potential field

approach can also be incorporated with coverage problem for both single and multi robot cases. In [73], constrained coverage is achieved for static sensor network node deployment. Works in [74], and [75] social potential function is implemented for robot formations by considering the coverage as an emergent behavior.

# CHAPTER 3

# Our Proposed Behavior Based System Architecture

With this chapter we begin to introduce the system architecture that we developed for a robot/sensor system.

## 3.1 Robot Network

Proposed multi-robot system is formed by heterogeneous robots. Each robot has one primary sensor and many auxiliary sensors. The primary sensor is used to detect a task source in the environment. For example for hazardous waste detection and clean up the robot in the system should have only one of the following primary sensors: chemical sensor, nuclear sensor, infrared sensor, spectrometer, etc. The primary sensor determines the role of that robot in the multi robot system. Auxiliary sensors which do not affect the role of the robot are general purpose sensors used for secure navigation and detection of other robots. For instance robot should have sonar to detect and avoid obstacles; light or motion sensors to detect other robots.

Communication capability requires additional communication hardware. In this architecture the robot may or may not have a communication unit. A robot without communication unit will therefore cooperate effectively and deliberately during task achievement will not be able to request of other robots.

In this simulation, each robot has the following capabilities:

- **Robots have the same control architecture**
- **Each robot senses the task source with its primary sensors**

- **Each robot senses the environment and the other robots to avoid collision and navigate securely.**
- **Each robot monitors its status and generates appropriate actions.**
- **Robot can request the help of other robots and respond the incoming requests coming from other robots if it has a communication unit.**
- **Each robot can be master (manager) or slave (worker) in a cooperative task execution phase.**
- **Each robot can evaluate its fitness for task allocation**
- **Each robot generates optimal or near optimal wander direction to decrease target detection time and energy consumption.**

The properties listed above can be related to:

- The robot behavior based control
- The communication structure and algorithms
- The optimality issues, concerning energy consumption and source detection and task execution time

## 3.2 Proposed System Architecture

### 3.2.1 The General Architecture

Robots in our proposed system are heterogeneous and capable of working cooperatively based on a market-based auction task allocation algorithm assigning co-occurring, sequential or individual tasks.

The layered, hybrid control system is designed based on subsumption and motor schema control strategies. In classical subsumption control, there is a priority based hierarchy between behaviors such that only one behavior can be active at a time. It is not being possible to separate behaviors, i.e. a behavior having high priority can cross couple with a behavior having low priority. Moreover, in complex

systems, there should be behaviors devoted to reactive action and be active at all time.

To surpass these difficulties, we developed the hybrid controller which contains two main layers: the **subsumption layer**, and the **motor schema layer**.

The subsumption layer contains priority based behaviors, whereas in motor schema layer, there are behaviors having equal priority. Behavior coordination is achieved in motor schema style which is devoted to behaviors requiring reactive actions. Final response is generated by summing responses from subsumption and motor schema layers.

We also introduce in the architecture a new control unit called **evaluator** which is used for defining priorities of behaviors in run-time, and the loosely coupled coordination between behaviors in subsumption and motor schema layers. The evaluator takes the state and output information of other behaviors as additional input parameters. By using an evaluator, priorities of behaviors in the subsumption layer can be changed dynamically depending on the state of behaviors. Another benefit is that behaviors in different layers can be fused within a function or filtered out, generating a coordination among the behaviors.

The proposed system has 4 types of units shown in **Error! Reference source not found.**: External behaviors, internal behaviors, planner, and behavioral coordination buses. Behaviors are divided into two parts, external and internal behaviors. Behaviors triggered by external sensory inputs or virtual input generated by internal behaviors are called external behaviors. In our applications they are selected as listed below.

- **Target Reaching Behavior**
- **Obstacle Avoidance Behavior**
- **Robot Separation Behavior**
- **Heuristic Wander Behavior**
- **Adaptive Wander Behavior**

- **Communication Behavior**
- **Environment Monitor Behavior**

These behaviors implement the main behavioral backbone. They translate the information gathered form sensors to an appropriate motor control vector. On the other hand, internal behaviors take the outputs of external behaviors and interpret them and control and coordinate the external behaviors. Outputs of these behaviors are either system state transition or virtual inputs to the external or internal behaviors. Internal behaviors implemented in the proposed system are:

- **State Evaluation Behavior**
- **Physical Situation behavior**

Behavior control bus includes necessary lines for subsumption style behavioral coordination. These are: suppress bus, inhibit bus, and reset bus. State evaluation behavior uses these buses to control and coordinate external behaviors. Outputs of the external behaviors are transported over behavior output bus.

There is a planner unit responsible for task allocation and execution. Moreover, it arranges data structures, and maps used by other behaviors.

#### 3.2.1.1 Proposed Control Architecture

In Figure 9, general structure of proposed architecture is shown with subsumption, and motor schema layers. Final response is generated by coordinating the subsumption and motor schema layers cooperatively, and sent to the actuators. A behavior may have or not have an associated evaluator posterior to it. If a behavior in the subsumption layer has not an associated evaluator then it is coordinated with other behaviors in the same layer just classical subsumption control strategy. The proposed control strategy is flexible enough when making complex actions without degrading the reactivity.

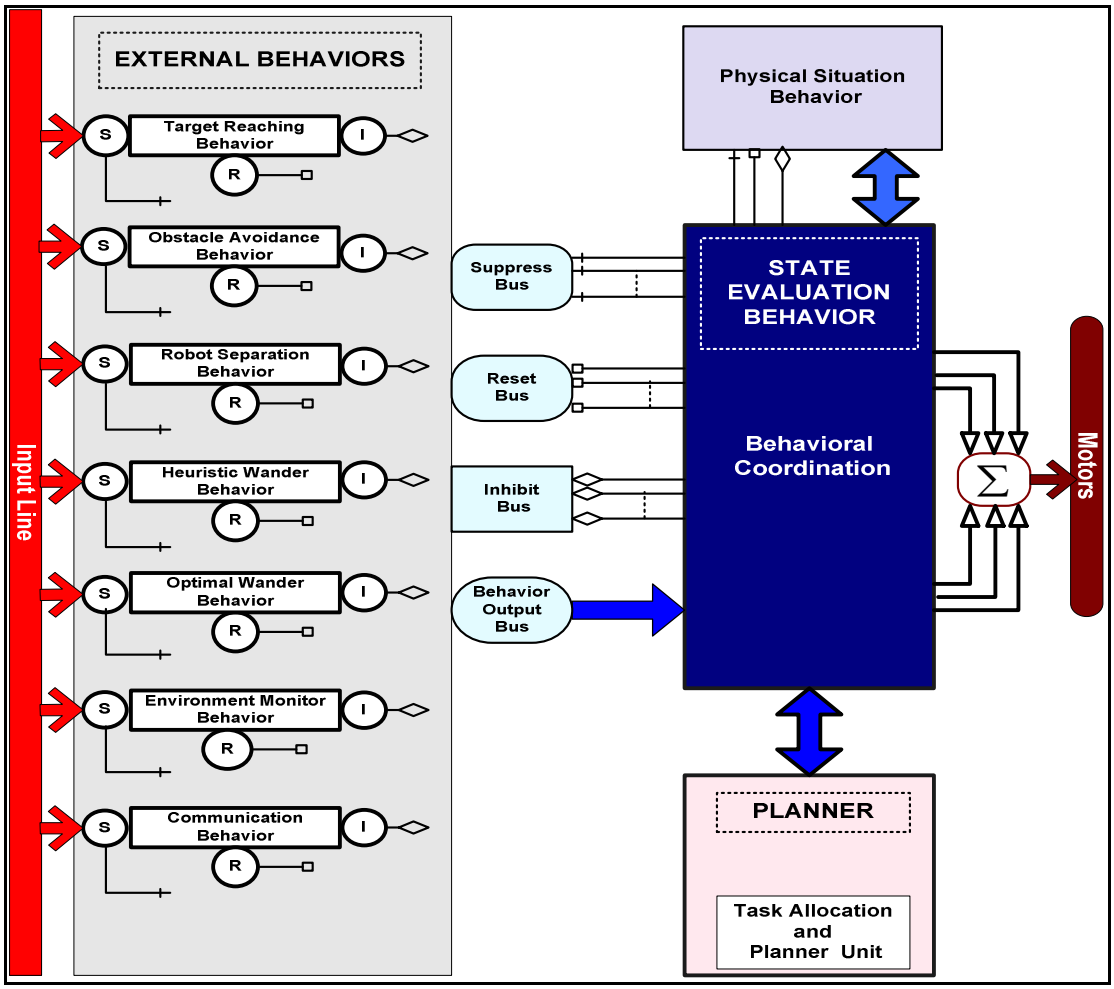


**Figure 8 Proposed system architecture**

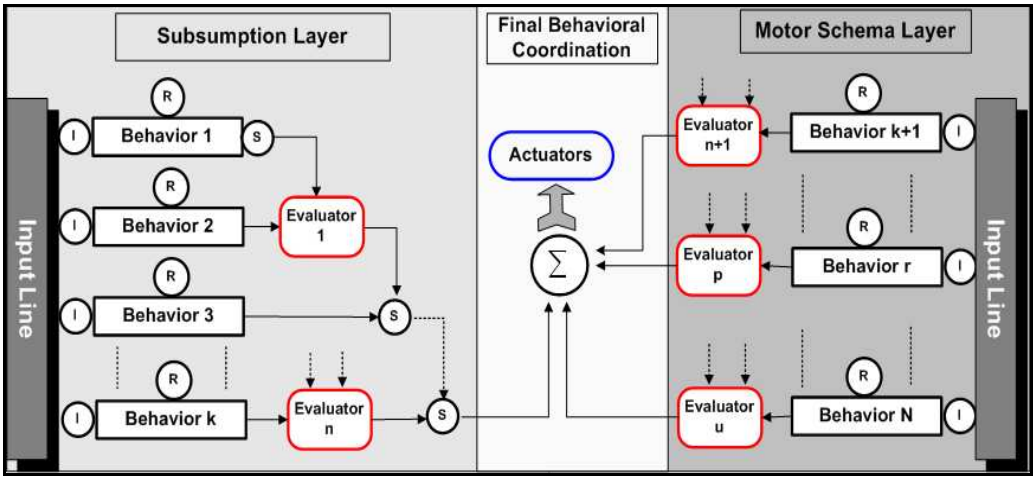


**Figure 9 Proposed control architecture**

Obstacle avoidance and environment monitor behavior have the highest priority in the subsumption layer. Priorities of these behaviors are equal to each other. They are coordinated in the subsumption layer by using evaluator1 in cooperative coordination style. If one of these behaviors is active then, activities of target reaching, communication, adaptive wander behaviors are inhibited. For this case, subsumption layer outputs the sum of responses of obstacle avoidance and environment monitor behavior.

Communication and target reaching behavior have the second highest priority in the subsumption layer. Evaluator 2 coordinates these behaviors as shown in Figure 11. Communication behavior has a priority over target reaching behavior if certain conditions are satisfied. As shown in Evaluator 2, if communication behavior is active at previous step then it inhibits the target reaching behavior, otherwise depending on the location of target, and communication event, activity of one of the behavior is inhibited according to the minimum distance criteria. Adaptive wander behavior has the lowest priority in subsumption layer. Moreover it is coordinated with heuristic wander behavior in motor schema layer via Evaluator 3. Even if adaptive wander behavior is active, it may not generate an appropriate next wander point. Heuristics wander behavior resides in the motor schema layer; it is almost active all the time if adaptive wander behavior is not active. This behavior is not implemented in the subsumption layer because of the performance issues. Activity of obstacle avoidance behavior inhibits the activity of adaptive wander behavior but it is not desired to inhibit the activity of heuristic wander behavior. As a result heuristic wander behavior is designed in motor schema layer. But it is coordinated with other behaviors as shown in Evaluator3.

Robot separation behavior is active for all time, so it resides in motor schema layer. It can be though that this behavior can also be deployed in the subsumption layer on top of obstacle avoidance behavior. But this will make the system inefficient because main goal of the system is implementing efficient collaborative work. Robot should execute target reaching behavior at the same time avoiding the collision with other robots. This behavior should reside in motor schema layer.

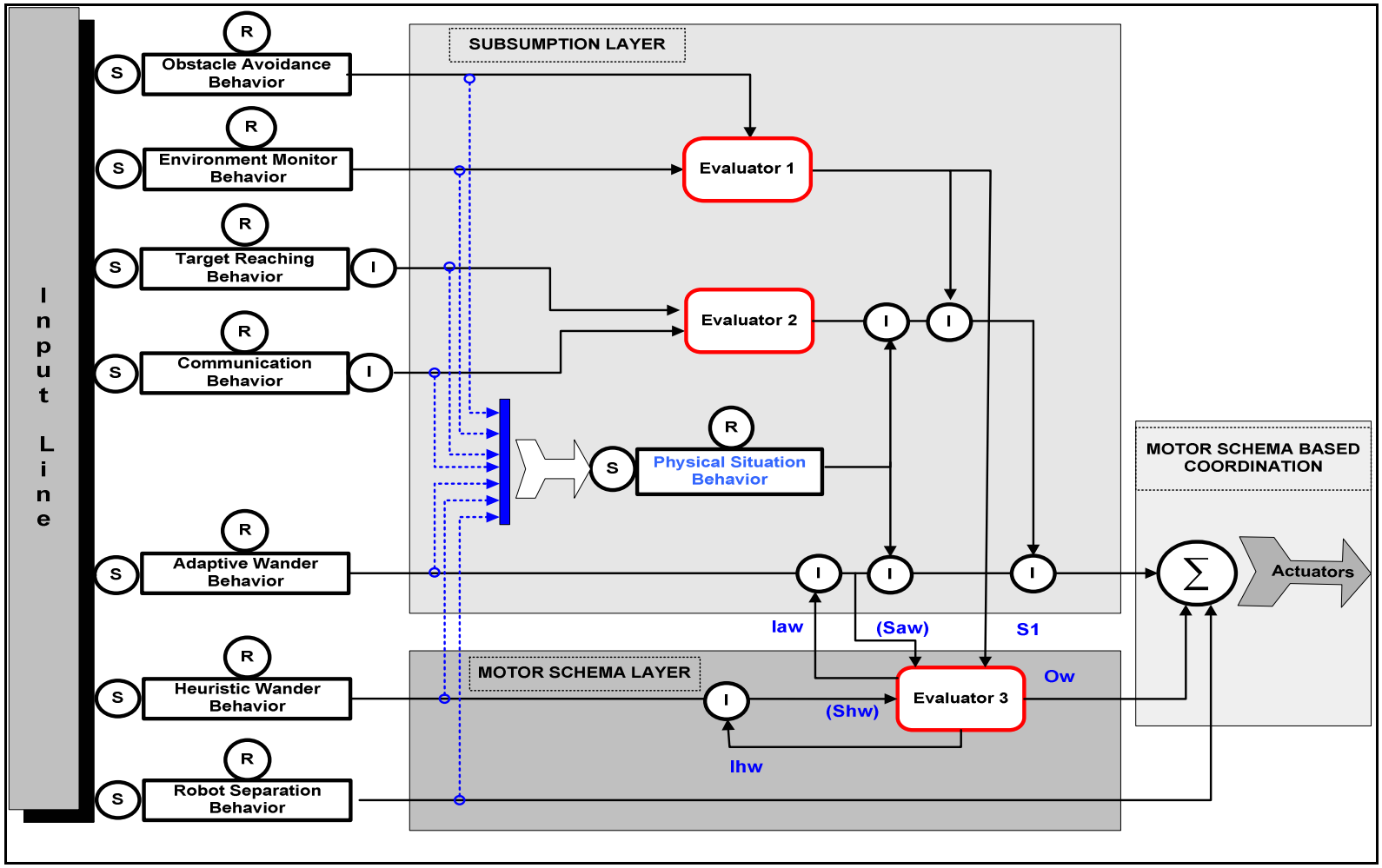


**Figure 10 Behavioral coordination made by state evaluation behavior**

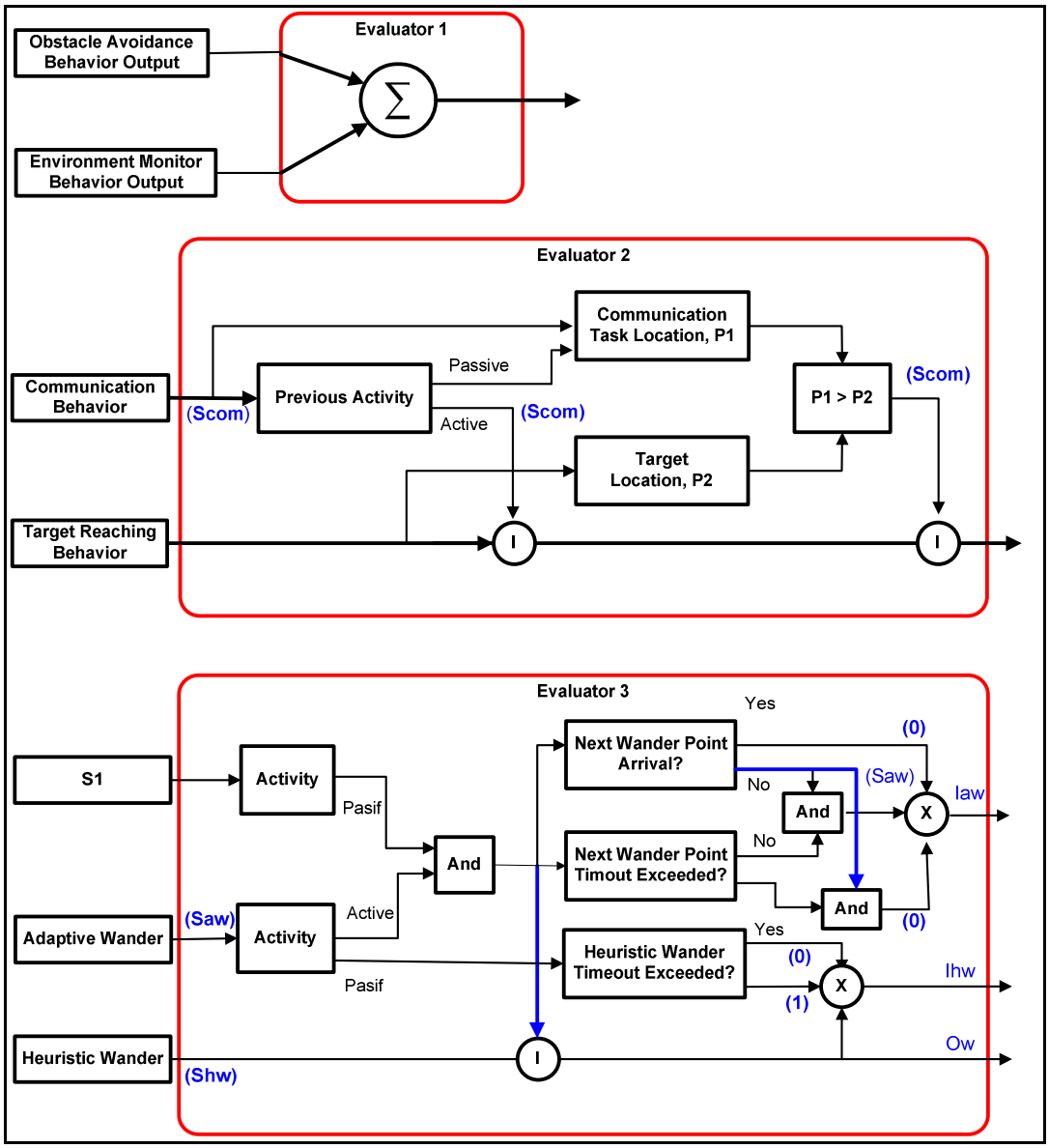


**Figure 11 Evaluator used in behavioral coordination**

## 3.2.2 Behaviors in the Proposed System

A behavior encapsulates both its perceptual and motor schema. By perceptual schema, triggering conditions of behavior is meant. Motors schema determines the action of behavior to the stimulus. In this section, for each behavior both of the schemas will be analyzed. Motor schema is implemented with potential field methods. Each behavior's motor schema generates a motor action represented with

potential field forces or equivalently each motor schema generates a potential field. Motion planning or potential field approach will be analyzed subsequent sections.

A behavioral unit is shown in Figure 12. As being compatible with subsumption and motor schema architecture, behavior both encapsulates its perceptual schema and motor schema. More specifically each behavior has

- Input port
- Output Port
- Inhibit Line
- Suppress Line
- Reset Line

**Input port** transfers necessary input to the perceptual schema, whereas **output port** sends output of motor schema to the actuators. Suppress line is used to suppress the inputs of the behavior, on the other hand inhibit line is used to inhibit the output of this behavior. Reset line is used to reset behavior to its initial state.

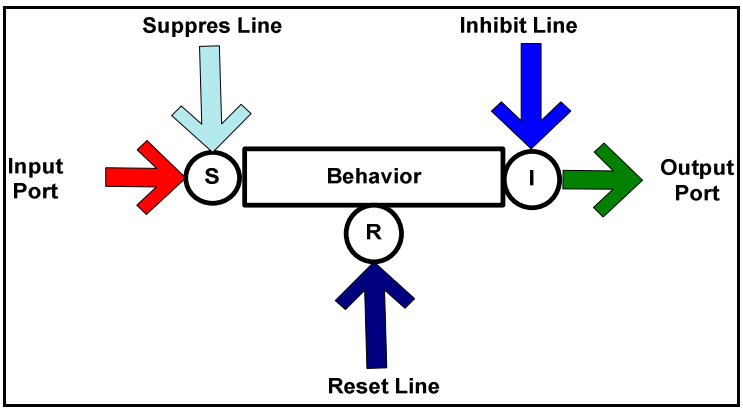


**Figure 12 A behavioral Unit**

Evaluators are used with behaviors for

- Defining priorities of behaviors in run-time
- Defining loosely coupled coordination between behaviors in subsumption layer and motor schema layer. Evaluator takes state of other behaviors as additional input parameters

In Figure 13, structure of an evaluator is shown. Evaluators take different signals from behaviors in the same or different layers. It generates an output by considering the state and output information of other behaviors as additional input parameters.

By using an evaluator, priorities of behaviors in the subsumption layer can be changed dynamically depending on the state of behaviors. Another benefit is that behaviors in different layers can communicate with each other.

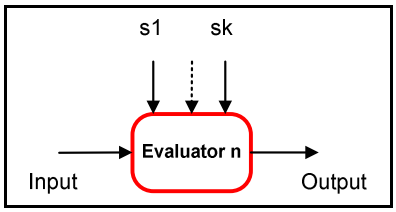


**Figure 13, Evaluator structure**

The output of evaluator k, $e_{k,Out}$, is a function of both input $e_{k,In}$, state and output information, S, of other behaviors, i.e.

$e_{k,Out} = F(e_{k,In}, S)$

where

$e_{k,In}$ : Input of evaluator k

$e_{k,Out}$ : Output of evaluator k

$S = \{s_1, s_2, s_u\}, \quad s_j = \{e_{j,Out}, \zeta_j\}$

$S$ : Set of information of other behavior,

$s_j$ : State information of behavior j

$e_{j,Out}$ : Output of behavior j

$\zeta_j$ : The state of behavior j.

### 3.2.2.1 Target Reaching Behavior

Target reaching behavior is responsible for detection and reaching the target. Target detection is achieved with the primary sensor of robot. In the case of multiple

of targets that are detected then the robot selects the target which minimizes energy consumption, thus arrival time. The behavior related to target reaching generates attractive **target reaching** force to reach target after it is detected. This behavior is not all the time active and depending on the state of robot, it may be suppressed.

#### 3.2.2.2 Obstacle Avoidance Behavior

This behavior implements obstacle avoidance. Obstacle avoidance behavior is a crucial behavior since performance of this behavior directly affects the performance of the robot and the whole robot team. This behavior generates repulsive and tangential obstacle **avoidance** force to avoid obstacles. Inputs of this behavior are from secondary sensors.

Obstacle avoidance field is the sum of two vectors shown in **Figure 14**. The first one is a repulsive force from the obstacle and the second is the tangential force. Function of repulsive force is to avoid collision whereas tangential force with respect to obstacle is used to navigate around the obstacle for exploratory purposes. Tangential force increases the performance of the system considerable in obstacle avoidance sense. This behavior is active whatever the robot state, i.e. no inhibition or suppression is applicable for this behavior.

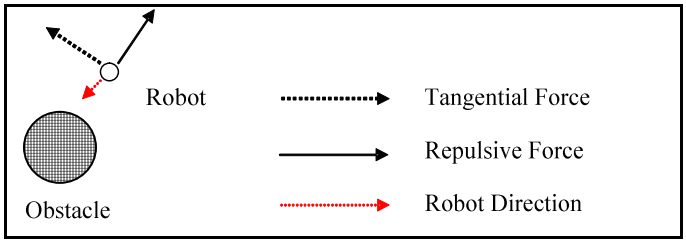


**Figure 14, Obstacle Avoidance Forces**

Obstacles are detected by the algorithm listed in the obstacle detection procedure. Generally robot detects lots of obstacle objects in an instance of time. Total obstacle avoidance field is evaluated as a vector sum of each obstacle avoidance field of detected objects.

#### 3.2.2.3 Robot Separation Behavior

Robot separation behavior detects other robots and avoids the collision. This behavior is necessary for secure navigation and task execution. Other robots are detected with secondary sensors. Robot knows the identification of other detected robots. This behavior generates repulsive **robot separation field** force. This behavior is active all the time whatever the robot state. There is no behavior inhibits or suppress this behavior.

Robot separation behavior may be implemented with obstacle avoidance behavior but this will lead to the degradation of cooperative working of robots. Obstacle avoidance behavior generates additional tangential force that does not allow robots to work close enough to each other.

#### 3.2.2.4 Heuristic Wander Behavior

This behavior is one of the wander behaviors. The main function of robot is wandering around the environment and detecting targets or involving in communication with others when needed. It generates an attractive **heuristic wander** force to wander in the environment.

The random nature of this behavior increases the reactivity. A deterministic wander algorithm may be prone to trapping in unplanned situations and its designer should be aware of everything exhaustively prior to the design and every possible situations about the environment. This means that extensive information fusion is required for optimal solutions [67], [68]. This heuristic wander field enables the robot wander very securely around the environment even under unexpected situations. Heuristic wander behavior is not active all the time; its activation condition depends on activity of other behaviors and it was discussed in 3.2.1.1.

#### 3.2.2.5 Adaptive Wander Behavior

Adaptive wander behavior is designed to increase the wandering performance of robot. Since a robot detects its target through its primary sensors, wander performance directly affects the target detection time, and consequently team performance. This behavior is implemented as a attractive **adaptive wander force** which will discussed in section 3.2.3.5.

This behavior generates wander maps to fuse this information adaptively. Maps are time varying, carrying a forgetting factor for each map. Time varying property is compatible with **dynamic** environment assumption. Periodically maps are updated, i.e. some data are extracted from maps and this period is an adjustable parameter that naturally affects the performance.

When robot wanders around the environment and when it detects targets, obstacles, it generates its own wander map, detected target maps and detected obstacle maps. Information fusion is made through these maps to generate the new wander direction. The generated wander direction is optimum in the sense that:

- Robot prefers the location which is not visited previously.
- Robot avoids obstacle dense regions. If a wander direction crosses a wall previously detected, then robot should not prefer this direction.
- Overall coverage time is decreased as compared to the heuristic wander.

This behavior is not active at all the times since then, the adaptive wander algorithm may generate false alarm which in this case, the heuristic wander behavior is activated.

#### 3.2.2.6 Communication Behavior

Communication behavior is responsible for reaching locations generated by the task allocation and execution planner suitable for communication.

Communication behavior is active all the time expect when the robot is executing a task. This behavior is implemented with an attractive **communication force.**

#### 3.2.2.7 Environment Monitor Behavior

Environment monitor behavior is implemented for wandering securely in a 3D environment. In some cases robot may not be allowed to some regions. In this case this behavior tracks and controls this undesired regions through a repulsive **environment monitor field** force. In this 3D environment robots have to explore surrounding and execute every command where the environmental conditions should put constraints

This environmental monitor behavior detects the following:

- Unreasonable regions that robot dynamics will not be allowed to enter. For example, robot may not be allowed within regions having a steepness over a maximum allowed value.
- Banned region avoidance. Some regions may be allowed for navigation only for some amount of time. Outside of this time limit, robots will be prohibited to wander to in those premises.

Environment monitor behavior is active all the time making the system definitely makes the system more realistic environmental changes.

#### 3.2.2.8 State Evaluation Behavior

State evaluation behavior is an internal behavior and monitors, coordinated the external behaviors using suppress, reset, and inhibit buses. Actually this behavior generates the logic of robot and a primary control of behavioral coordination among other behaviors. Since this behavior is an internal behavior, it does not generate a field force as other external behaviors. Coordinated motor control signal is sent to motors.

#### 3.2.2.9 Physical Situation Behavior

Physical situation behavior is an also internal behavior that controls the situation of a robot in the physical environment by interpreting the outputs of the eternal behaviors. It detects stuck conditions of robots and environmental conditions. This behavior continuously updates the environmental conditions. For instance, if robot is running at rainy environment then it will not be able to perform at a desired speed due to slippery ground. So a kind of calibration is needed for internal parameters. The overall function makes this behavior behave as a

- Physical stuck detector, used as local minima detection or real physical stuck
- Environment modeler from buffered internal data. This property is not implemented for the time being and will be addendum of future works

This behavior is different from environment monitor behavior because it analyzes the outputs of the external behaviors and tries to detect unusual things. On the other hand environment monitor behavior just detects the locations having excessive slopes. This behavior is active the all times.

#### 3.2.2.10 Behavior Output with Potential Fields

Vectors are generated as forces using the derivative of derivative of potential field force function with respect to distance in 3D space will be presented. The generated force according to the proposed potential field satisfies the following conditions:

- High magnitude around target point.
- Quadratic decrease in magnitude with increasing distance between target point and robot point
- Constant in magnitude after some distance.

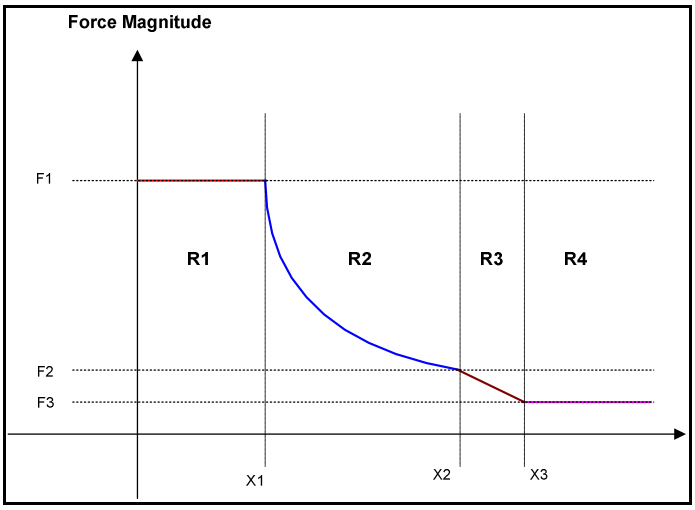


**Figure 15, Potential field force magnitude with respect to distance**

In Figure 15, magnitude of the proposed highly nonlinear force with respect to the distance is shown for the four different regions:

**R1**: Constant high force region

**R2**: Quadratic force region, force decreases quadratically with respect to distance

The general form of the force in this region is $F = \frac{m}{x^2} + n$

**R3**: Transient region from R2 to R4.

**R4**: Constant low region

Introduction of these regions is necessary in:

- Preventing sharp robot movements, smoothing down the motion.
- Obtaining reasonable force values. Force values become bounded within some max. and, min. values.
- Generating quadratic decrease or increase in region 2 enables the quick decrease or increase around target point. Robot is more reactive to position change.
- After some distance potential force begins to be ineffective. It is in effect within a region and zero outside.

In Figure 16, 2D plot of the gradient of proposed potential field giving the force is given. This plot is obtained relative to the origin by simply rotating force function along z-axis.

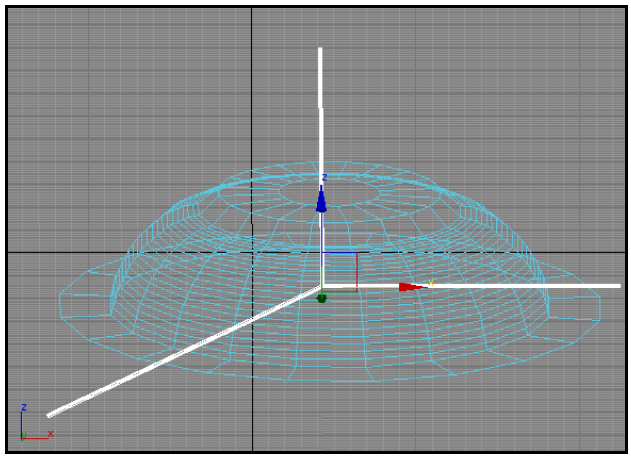

**Figure 16, Potential filed force in 3D**

The magnitude of force function is given for each region as:

$$|F| = \begin{cases} F_1 & , R_1(0 \le X < x_1) \\ \dfrac{(F_1 - F_2)(x_1 x_2)^2}{x_1^2 x_2^2}\dfrac{1}{X^2} + F_1 - \dfrac{(F_1 - F_2)(x_2)^2}{x_1^2 x_2^2} & , R_2(x_1 \le X < x_2) \\ \dfrac{(F_2 - F_3)}{(x_2 - x_3)}X + F_2 - \dfrac{(F_2 - F_3)}{(x_2 - x_3)}x_2 & , R_3(x_2 \le X < x_3) \\ F_3 & , R_4(x_3 \le X < \infty) \end{cases} \quad (1)$$

**Each behavior in the proposed architecture has $F_1, F_2, F_3, x_1, x_2, x_3$ values. By changing these values relative strength and relative effective distance of each behavior is adjusted. In**

Table 1, the current force function parameter for each behavior is shown. We choose this set of parameters through simulation experiments.

**In**

Table 1, some values are shown as **NA (∞)**. This is because of some behaviors takes only one behavior in reasonable distances. Heuristic wander and adaptive Wander

behavior are this kind of behaviors. The logic behind their formulation gives rise to this result. Later, details of these behaviors will be discussed

**Table 1, Field function parameters for each behavior**

| Behavior | $F_1$ | $F_2$ | $F_3$ | $X_1$ | $X_2$ | $X_3$ |
|---|---|---|---|---|---|---|
| **Target Reaching** | 50.0 | 20.0 | 0.0 | 2.0 | PSR - 2.5 | PSR - 2.0 |
| **Obstacle Avoidance, Repulsive Field** | 100.0 | 5.0 | 0.0 | 1.0 | ODSR – 1.5 | ODSR – 1.0 |
| **Obstacle Avoidance, Tangential Field** | 10.0 | 1.0 | 0.0 | 1.0 | ODSR/2– 1.5 | ODSR/2 – 1.0 |
| **Robot Separation** | 40.0 | 11.0 | 10.0 | 2.0 | RSDSR– 2.5 | RSDSR – 2.0 |
| **Heuristic Wander** | 40.0 | NA(∞) | NA(∞) | NA(∞) | NA(∞) | NA(∞) |
| **Adaptive Wander** | 40.0 | NA(∞) | NA(∞) | NA(∞) | NA(∞) | NA(∞) |
| **Communication** | 50.0 | 20.0 | 0.0 | 2.0 | CR - 2.5 | CR - 2.0 |
| **Environment Monitor** | 40.0 | 2..0 | 0.0 | 2.0 | 3.8 | 4.0 |
| **Physical Situation** | NA | NA | NA | NA | NA | NA |
| **State Evaluation** | NA | NA | NA | NA | NA | NA |

Abbreviations:

PSR : Primary Sensor Range

ODSR : Obstacle Detector Sensor Range

RSDSR : Robot Separation Detector Sensor Range

CR : Communication Range

NA(∞) : Not Applicable or Infinite

## 3.2.3 Implementation of Behaviors

### 3.2.3.1 Target Reaching Behavior

Target reaching behavior gets input from primary sensors of the robot and generates a direction to which a robot is directed. Number of targets determined can be any number. Since robot can only lock and go to only one target, it should make a decision between sensed targets.

For any time list of sensed targets, $T = \{t_1, t_2, ..., t_n\}$

The robot selects the target at minimizing distance between robot and target. Robot evaluates the metric below for each target and selects the winning target.

$$\min\left(\sqrt{\left\|P_R - P_{t_i}\right\|}\right), 0 \le i \le n$$

$P_R$: Robot Position, $P_{t_i}$ : i[th] target position

Target reaching behavior generates an attractive force from robot to target which had a magnitude evaluated according to equation (1).
The force direction of the force:

$$\vec{f} = \left(\frac{P_T - P_R}{|P_T - P_R|}\right)$$

where,

$P_R$: Robot Position in 3D

$P_T$: Target Position in 3D

Overall force vector will be:

$\vec{F} = \vec{f}|F|$

### 3.2.3.2 Obstacle Avoidance Behavior

As mentioned earlier, this behavior is composed of two basic forces: a repulsive force and a tangential force. The repulsive one being responsible for avoiding collision whereas the tangential force is responsible for generating a direction for the robot to wander around obstacle. The magnitude of each force is evaluated with equation (1).

In Figure 17 obstacle particles detected by robot (blue circle) is shown. Red circle represents the range of robot obstacle detector sensors. Robot detects many obstacle particles for a medium size obstacle. The overall force generated by obstacle particles over the robot is a linear combination of each particle force.

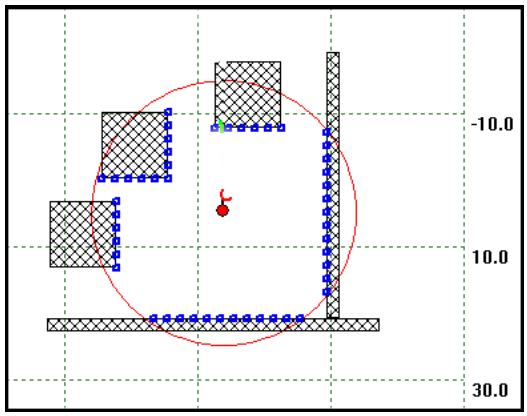


**Figure 17, Obstacle particles detected by robot**

Set of obstacles, *O,* detected by robot $O = \{o_1, o_2, ..., o_P\}$

Direction of repulsive force for a particle is

$$\vec{f}_i = \left( \frac{P_R - P_{O_i}}{\left| P_R - P_{O_i} \right|} \right)$$

$P_R$ : Robot Position in 3D

$P_{O_i}$ : i$^{th}$ Obstacle Position in 3D

Overall repulsive force direction will be

$$\vec{f}_{\,repulsive} = \sum_{i=0}^{P} \vec{f}_i$$

Tangential force evaluation is a little bit different. A set boundary points is chosen around the particle on the obstacle nearest to the robot. These objects are used to generate non-linear obstacle curve to calculate the relative direction of the robot with respect to nearest obstacle. The algorithm is given below.

***Algorithm, Tangential Force Calculation***

---

**1**. Evaluate the nearest obstacle particle to the robot, $O_{nearest}$

**2**. Generate a set, $R\{O\}_N$, N points on the boundary of obstacle $O_{nearest}$

**3**. Find the 3rd order polynomial curve, $C$ passing points in the $R\{O\}_N$ set, minimizing least square error.

**4.** Find the nearest point, $P_{O_{nearest}}$, in $R\{O\}_N$ to the robot, or simply use the position of $O_{nearest}$ particle.

**5.** Evaluate the surface, $S_{O_{nearest}}$ passing through $P_{O_{nearest}}$

**6.** Evaluate the tangential force direction, $\vec{f}_{\tan}$ by projecting direction of robot, $\vec{d}$, on to the surface, $S_{O_{nearest}}$

---

In Figure 18, mentioned curves, sets and points are shown. In this case there are two different obstacles sensed by the robots. Robot executes the above algorithm to find the nearest obstacle particle and set of neighborhood particles. Then it evaluates the curve passing through this particle points (curve C shown in the figure). Surface passing through the nearest point on the curve to the robot is found. Finally tangential force is evaluated by projecting the direction vector of robot onto this surface.

The net force will be

$$\vec{F} = \vec{f}_{repulsive1} \left|F\right|_{repulsive1} + \vec{f}_{repulsive2} \left|F\right|_{repulsive2} + \vec{f}_{\tan} \left|F\right|_{\tan}$$

Robot will move in the direction of $\vec{F}$.

### 3.2.3.3 Robot Separation Behavior

Robot separation behavior is implemented for preventing collision of the robot with each other. Each robot is capable of sensing other robot regardless of its primary sensor type. A robot can sense any number of robots within its robot separation detector sensor range (**RSDSR**).

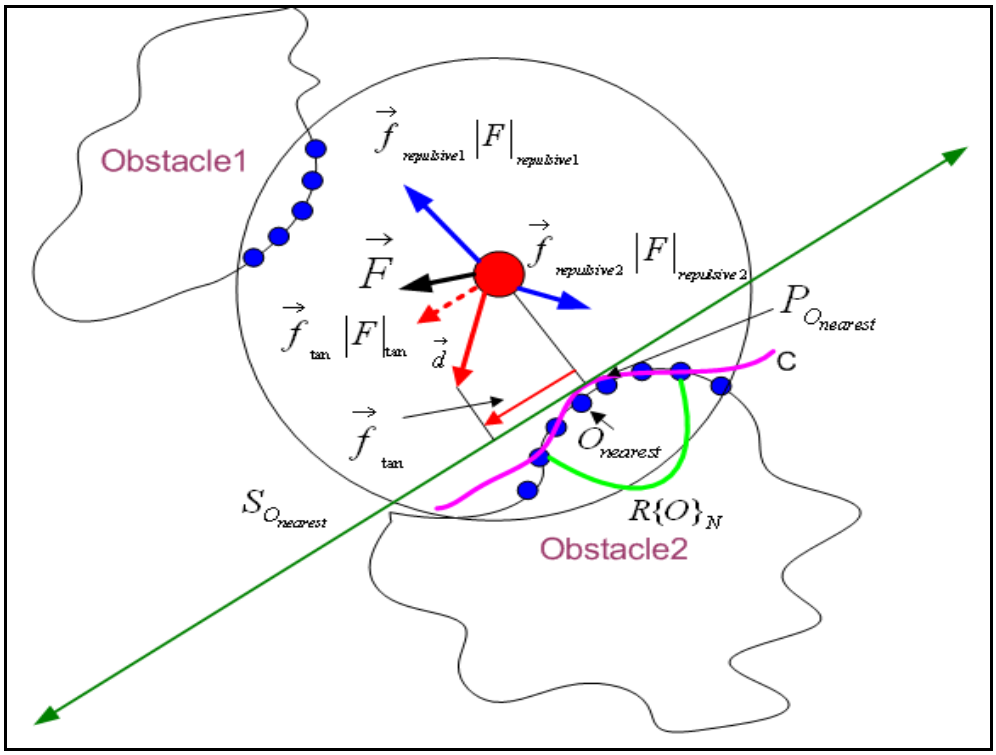


**Figure 18, Tangential and repulsive force diagram.**

The magnitude of robot separation force is calculated as described in equation (1). Robot separation force is a repulsive force; the direction of which is from the detected robot to one that detected. If more than one other robots are detected then the total force is evaluated by summing each robot separation force.

Set of robots detected by robot

$$R = \{r_1, r_2, ..., r_M\}$$

Direction of repulsive force for a detected robot is

$$\vec{f}_i = \left( \frac{P_{R_i} - P_R}{|P_{R_i} - P_R|} \right)$$

$P_R$ : Robot Position in 3D

$P_{R_i}$ : i$^{th}$ Robot Position in 3D

Overall repulsive force direction will be

$$\vec{F}_{net} = \sum_{i=0}^{M} \vec{f}_i$$

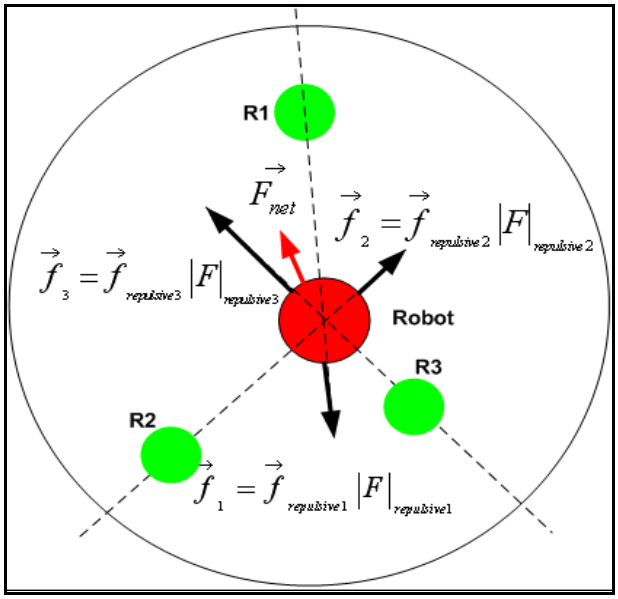


**Figure 19, Robot Separation Forces**

### 3.2.3.4 Heuristic Wander Behavior

Heuristic wander behavior is one of the basic behaviors. Robot wanders around the environment and searches the targets heuristically with the help of this behavior. Heuristic wander behavior is not active all the time. It is active only when adaptive wander behavior is not active.

Heuristic wander behavior generates a new wander direction at the beginning of the behavior. Behavior does not generate a new direction until some time is elapsed. The reason behind this is that robot is allowed to wander a certain time. Generating new wander direction repeatedly will force the robot to wander around the same point which is undesirable. Since physical robot will not turn immediately, robot will not generate sharp wander direction. Maximum wander angle is limited by a limit angle $\theta_{\max}$ . Choosing small $\theta_{\max}$ will decrease dead reckoning error due to

the rotation sensors. Moreover there will be some resolution that for θ we denote as of turning angle $\theta_{res}$ .

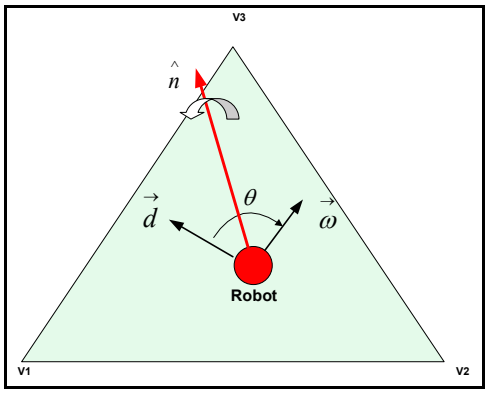


**Figure 20, robot generated a direction randomly by rotating it direction around normal vector of current grid that robot resides.**

In Figure 20, a robot on a 3D plane is shown. Vertices of the plane where the robot is currently located are $v_1$, $v_2$, and $v_3$ respectively. These vertices are obtained from terrain map represented as a DirectX mesh structure. Algorithm of heuristic wander direction generation is listed below.

***Algorithm, Heuristic Wander Direction Generation***

---

1. Determine the vertices of the plane where the robot is located from 3D terrain mesh.
2. Evaluate the normalized plane normal, $\hat{n}$ as

$$\hat{n} = \frac{(v_2 - v_1) \times (v_3 - v_1)}{|v_2 - v_1||v_3 - v_1|}$$

3. Generate an angle $\theta$ randomly (uniformly distributed) between $-\theta_{\max}$ and $\theta_{\max}$
4. Rotate robot direction vector, $\vec{d}$ , about plane normal by an angle $k\theta_{res}$ where

$k = \left\lfloor \frac{\theta_{\max}}{\theta_{res}} \right\rfloor$ , new direction is $\vec{\omega}$ .

5. Wait for some time, and continue from step 1.

**The overall force generated by this behavior is calculated using values from**

Table 1 and as shown there, the magnitude of the force is constant.

### 3.2.3.5 Adaptive Wander Behavior

Adaptive wander behavior is designed for generating optimal or near optimal wander direction. It is more optimal as compared to heuristic wander behavior. This behavior considers the following issues to increase the wander performance:

- Current coverage map
- Obstacle map
- Target map(optional)

This behavior makes a data fusion among the generated maps and determines the next wander direction by:

- Increasing temporary coverage by wandering through unvisited region
- Decreasing obstacle crash rate by wandering through region that is not dense with obstacles
- Increasing target detection rate by wandering through region having lower target density (estimated density).

**Coverage map** is a 2D $MxM$ grid map. Each entry of the map is a **linked list** having the following data structure in C++ style:

```
struct {
        _TIME time;
        _POSITION pos;
}MAP_COVERAGE_ELEMENT;
```

**time** is the system time

**pos** is the 3D position of robot

The entire map has also the following C++ deceleration:

```
CList<MAP_COVERAGE_ELEMENT, MAP_COVERAGE_ELEMENT> **MapCoverage;
```

The number of times that the robot visited a region is determined by simply finding the length of the linked list.

The resolution of map is important because it determines the memory to be allocated for the coverage map and is determined by the range of the primary sensor of robot. In Figure 21, a grid of coverage map is shown. Theoretically, robot can sense the target within its primary sensor range shown as a dashed circle. Due to the noise or other undesired effect, the resolution of the coverage map is taken as smaller than the diameter of the robots sensing range. Size of the coverage map grid is then formulated as

$2\rho r$ , where $\rho$ is the discount factor. Currently $\rho$ is set to 0.8.

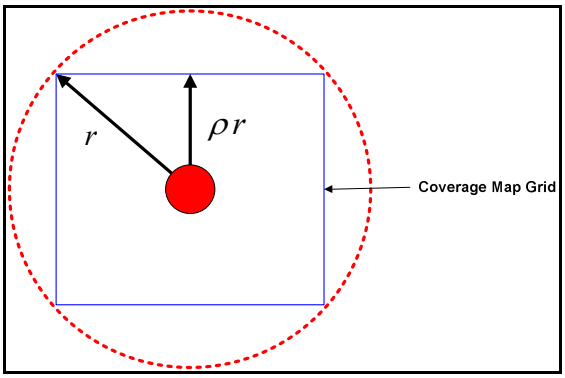


**Figure 21, Coverage map resolution calculation diagram**

To satisfy the dynamic environment assumption, data in the coverage map cannot be used forever. After a timeout limit, $TO_{\text{cov}}$ , entries at the linked list are deleted if the time of an entry is too old. Let the entry time be $T_1$ and the current time is $T_{curr}$ . Entry is deleted if

$$T_{curr} - T_1 > TO_{\text{cov}} .$$

Moreover at each simulation time the coverage map is refreshed to decrease its size. In Figure 22 and Figure 23 coverage map at time T1 and T2, T2 >T1, is shown. The height value corresponds to coverage value of current grid, i.e. how many times that the robot has visited a grid cell. Due to the timeout, the vertex value of the grid map

shown in the region 1 of Figure 22 are deleted at time T2 as shown in Figure 23. Red line in the figures points the next wander point.

Value of $TO_{\text{cov}}$ is important: If the it is selected too high then robot will memorize the environment and dynamic environment assumption will fail. However if it is too small then it will not use past coverage information and its performance will approach to that of heuristic wander behavior. In this system it is set to a time that is required to

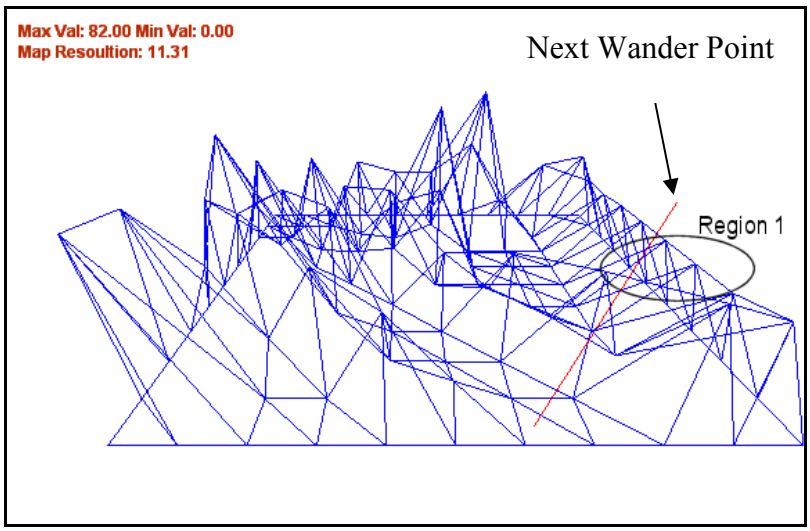


**Figure 22, Coverage Map at Time T1**

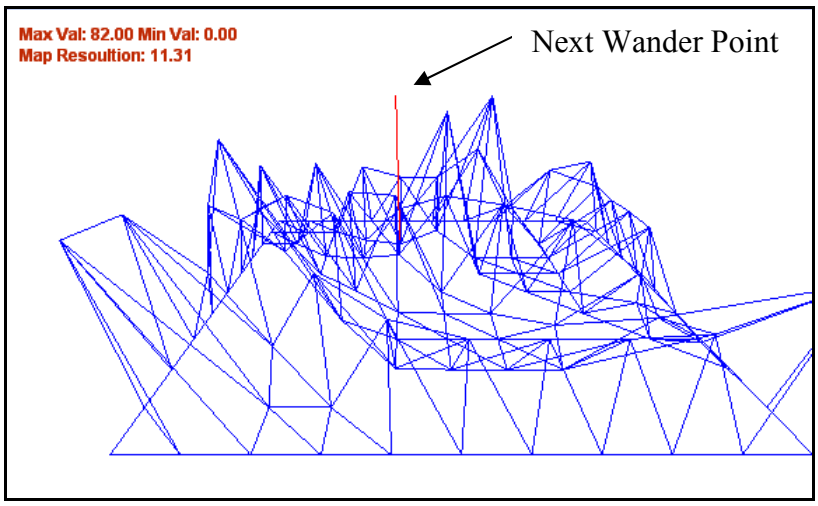


**Figure 23, Coverage Map at Time T2**

**Obstacle map** stores the information about the obstacle in the environment. Its primary usage is to decrease the obstacle avoidance rate. Implementation is similar to coverage map except that its resolution is different. Data structures are same with that of coverage map. The entire obstacle map is also has following C++ deceleration:

**CList<MAP_OBSTACLE_ELEMENT, MAP_ OBSTACLE _ELEMENT> **MapObstacle;**

Resolution is calculated as $2\rho r_{obs}$ where

$\rho$ is discount factor $r_{obs}$ is obstacle detector sensor range.

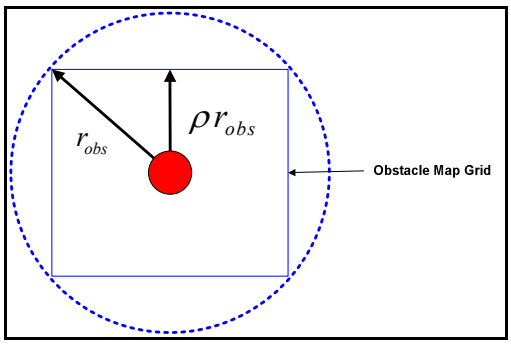


**Figure 24 Obstacle map resolution calculation diagram**

Another critical parameter is the timeout time, $TO_{obs}$, for obstacle map entry removal to satisfy dynamical environment assumption. It is set to $3TO_{\text{cov}} / 2$.

**Target map** is used to store the location sensed targets. Concept behind the target map implementation is the same as those of obstacle map and target map. Resolution of this map is equal to resolution of coverage map but the timeout time, $TO_{tar}$ is set to $TO_{\text{cov}} / 4$.

### *3.2.3.5.1 Memory Requirements for Maps*

Usage of maps increases the performance in terms of coverage and target detection time with better obstacle avoidance but there is a main disadvantage: memory allocation for maps. Memory space required for maps is determined by:

- Resolution of map

- Timeout time

Resolution of map depends on the sensorial range of the robot. High sensorial range results in lower memory demand. By decreasing the values of above parameters memory demand can be controlled but performance of the system can be decreased. Total space required can be calculated at steady state. Robot should have enough physical memory to meet the memory demand.

In Figure 25 normalized memory demand for three timeout values are shown. After some time, each situation reaches a saturation point. Of course reaching saturation will take different time for each case. For $TO_{\text{cov}} = 3000$ amount of memory needed at steady state is 1.5 times of that of $TO_{\text{cov}} = 2000$ and 3 times of $TO_{\text{cov}} = 1000$.

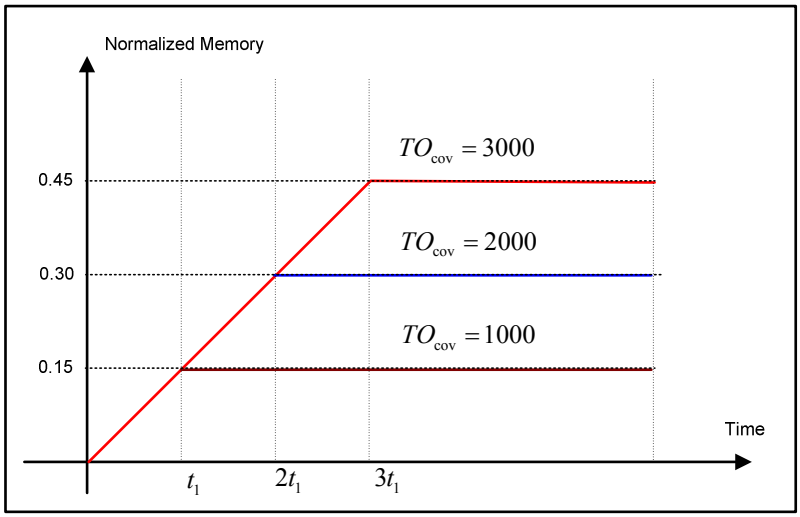


**Figure 25, Normalized memory requirement versus time for coverage map. Each line corresponds to different timeout time.**

### *3.2.3.5.2 Fusion of Maps Generating Next Wander Direction*

Fusion of maps is based on the finding free or unvisited location in the coverage and obstacle maps. The first step is searching the coverage and obstacle

maps for free grid cells. Of course, each location in the physical environment corresponds to a cell in the map with limited resolution. Searching the coverage map is achieved in an expanding radius search fashion, i.e. coverage map is searched starting from the cells having distance (in the index sense) 1 and at each step, it is increased by one.

In Figure 26, the search method is shown graphically. The robot location is shown as red circle. Start point of the path is in circular shape whereas end point is depicted as arrow. Search path having depth 1 is shown as "Depth 1". Shape of the search path can be square if the path does not intersect with the boundary. Depth 5 and Depth 6 is not in square shape because of the reasoning above.

To decide that a cell in the coverage map is free, corresponding cell in the obstacle map should also be free. Search procedure is continued up to enough free points are found in specified minimum search depth. If any free point is not found then adaptive wander behavior should be disabled for the current simulation time.

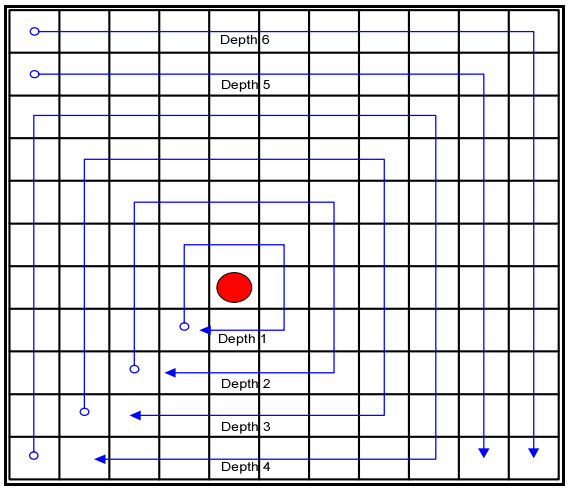


**Figure 26, Coverage map search path**

Frequently, search of the maps generates more then one free cell. So a decision should me made among these points. Decision making procedure is based on optimizing following issues:

- Minimizing obstacle crash

- Minimizing the angle between current wander direction and next wander direction

Depending on the above criterion some of the wander points are filtered out.

Some wander points should result in obstacle crash as shown in Figure 27. Even if the wander point does not contain any obstacle but robot should avoid the obstacle in its path. Obstacle avoidance will force the robot lose time and energy. In some cases robot should generate a wander point which is absolutely unreachable as shown in Figure 28. Robot should filter out this kind of wander points. So in any case some kind of post obstacle crash check should be made.

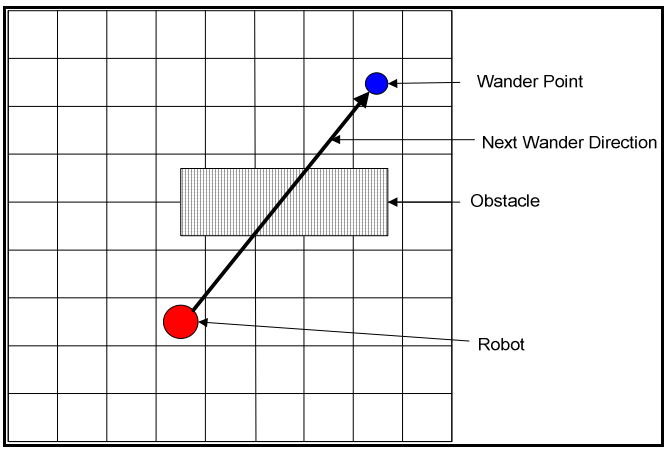


**Figure 27, Obstacle crash situation, partly reachable situation**

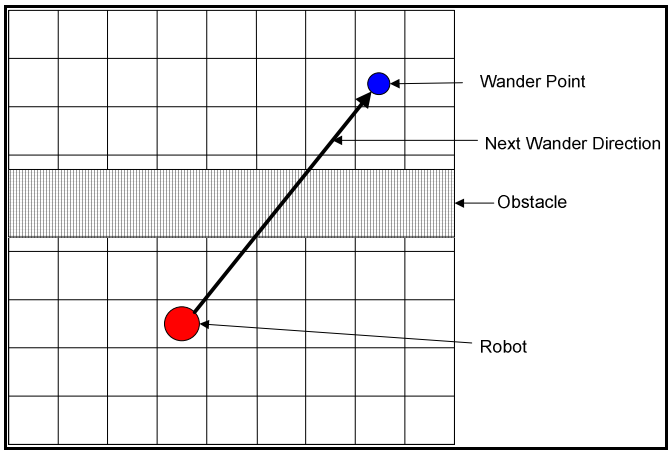


**Figure 28, Obstacle crash situation, absolutely unreachable situation**

#### 3.2.3.5.2.1 Obstacle Crash Check

Obstacle crash check is achieved finding density of obstacle in the region defined by a rectangle whose corners formed by robot location and wander point location. If obstacle density $D_{obs}$ is higher than a threshold then wander point is filtered out otherwise it is used.

In Figure 29 "Robot-Wander Point Rectangle" ( RWPR) is shown. Obstacle density estimation is calculated within this rectangle in horizontal direction. For each row in the RWPR obstacle presence is checked from obstacle map. If no obstacle is present along the row then this row is said to be obstacle crash free row.

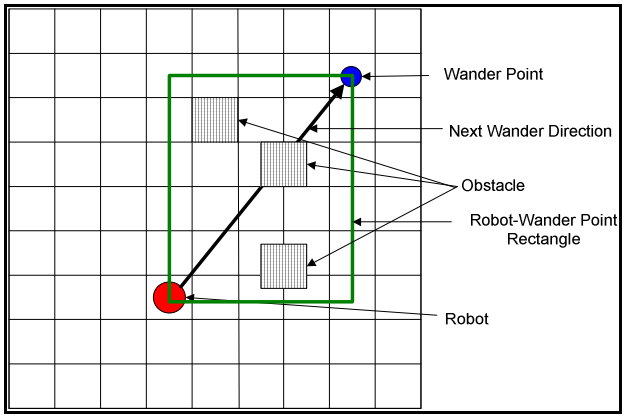


**Figure 29, Robot-Wander Point rectangle for obstacle crash check**

But if an obstacle exists in a point in the row then this row is said to be obstacle crash susceptible row. For each row above procedure is executed. If the ratio of the number of "obstacle crash susceptible rows" and "total number of rows" exceeds a threshold then wander point under consideration is filtered out. Depending on the value of this threshold "Partly Reachable" wander points may or may not be filtered out.

In Figure 30 obstacle density for two different situations are shown. The first case has obstacle density 0.57 since 4 of 7 rows are filled by obstacles. However in second case there are 5 rows filled with obstacles.

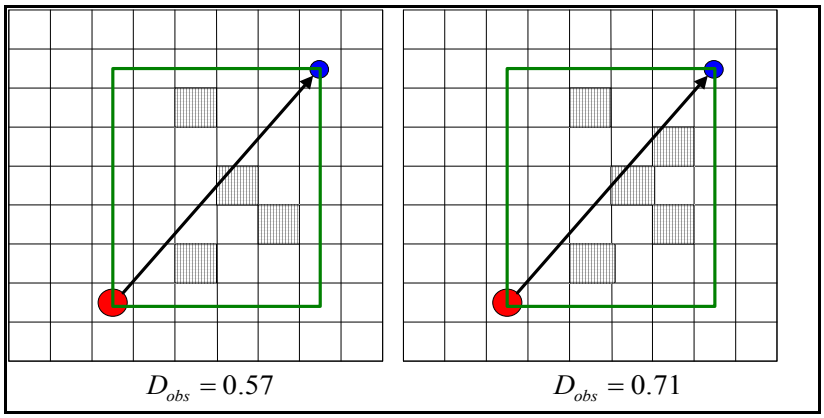


**Figure 30, Obstacle Density for two situations. Image at the left hand side has obstacle density 0.57 whereas right hand side situation has 0.71 obstacle density.**

#### 3.2.3.5.2.2 Angle Check

Another important criteria in the adaptive wander point generation is that angle check algorithm. There can be wander point candidates. Some of them may be filtered out by obstacle crash check. Remaining points are checked for angle test. Logic behind angle test is that robot should not be turned with large angles relative to its current direction. Large angle turn blockage will result in more continuous motion and less dead-reckoning errors. In Figure 31 angles of wander point direction relative to robot current direction is shown.

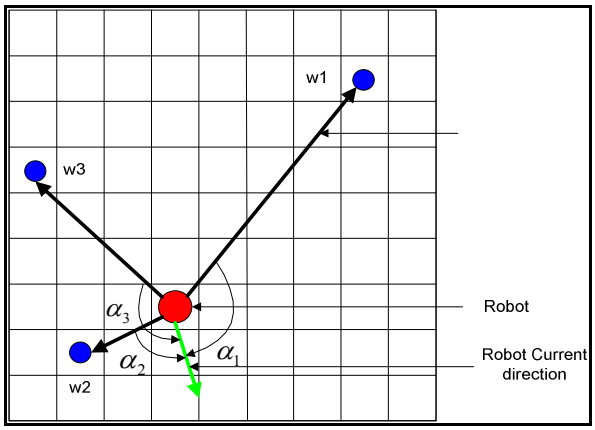


**Figure 31, Wander point angle with respect to robot current direction**

Proposed angle check is achieved as follows:

Set of wander point candidates points $W = \{w_1, w_2, ..., w_M\}$

For each point in set W, there is a set of angles A relative to robot current direction

$A = \{a_1, a_2, ..., a_M\}$

Final wander point candidates are selected whose angles are smaller than a threshold angle $\alpha_{w\_thr}$ .

#### 3.2.3.5.2.3 Final Decision of Wander Point

The final phase of the adaptive wander point generation is that making a decision between the wander points candidates filtered out by angle and obstacle crash check. Decision is made heuristically, i.e. wander point is determined among the candidate points randomly. This random nature is desired since it will allow robot to wander more freely within some degree. Moreover above deterministic algorithm becomes a little bit stochastic to filter out unconsidered situations. Below overall algorithms are shown.

***Algorithm, Adaptive Next Wander Direction Generation (without Target Map fusion)***

---

1. Generate set of free (unvisited) locations $F = \{f_1, f_2, ..., f_M\}$ by searching the coverage map and obstacle map up to a depth. If enough free cells are found then stop the searching. If no free point is found then return.

2. Apply angle test for points in $F$ for an angle $\alpha_{w\_thr}$ . Generate a new set formed by points that are passed angle check test. New set is "angled check passed" set $F_{acp} = \{ff_1, ff_2, ..., ff_K\}, K \le M$

   Where

   $ff_u = f_d, u \le M, d \le M$

3. Apply obstacle crash test for points in $F_{acp}$. Generate a new set formed by points that are passed obstacle crash test. New set is "angle check passed obstacle crash check passed" set $F_{acpocp} = \{ fff_1, fff_2, \ldots, fff_L \}\ L \leq K$

   $fff_u = ff_d, u \leq K, d \leq K$

   i.e. ***u*** and ***d*** are unique numbers.

4. Find the final wander point from set $F_{acpocp}$ by selecting a point randomly(uniform distribution), say $P_{wp} = fff_0$

5. Estimate time, $t_{est}$, necessary to visit from robot current position, $P_r$ to adaptive wander point, $P_{wp}$. Estimated time will be

   $$t_{est} = \frac{\left| P_r - P_{wp} \right|}{E\{v_r\}}$$

   $E\{v_r\}$ is the average speed of robot.

6. Wait $t_{est}$ and continue from step 1

---

### 3.2.3.6 Communication Behavior

Since proposed multi-robot system is strongly a multi-robot system, some kind of explicit communication is needed. Depending on the situation there may be need of help of other agents in the environment, and then robot tries to gather necessary number of agents with enough capabilities.

Each robot can only execute only one communication task. A communication task is created by a communication requester agent in the environment. Requester robot commands the robot to go a location in the environment. Communication behavior is responsible for navigating through this communication task location

securely. Logic behind the communication behavior is the same as target reaching behavior except that field force settings are different and there can be only one communication task location.

Set of communication task contains only one active communication task, $C = \{c_1\}$

Direction of attractive force for a communication event

$$\vec{f}_i = \left( \frac{P_R - P_{C_1}}{\left| P_R - P_{C_1} \right|} \right)$$

$P_R$ : Robot position in 3D

$P_{C_i}$ : Position of i$^{th}$ communication task in 3D

Over all force will be

$$\vec{F}_{net} = \sum_{i=0}^{M} \vec{f}_i$$

### 3.2.3.7 Environment Monitor Behavior

Function of environment monitor behavior is collecting information from environment and generating appropriate motor responses. Since robot has physical constraint, it cannot achieve all the maneuvers. Physical constraints may be:

- Slope, robot cannot drive at a surface having big slope
- Water or mud level of soil
- Temperature or humidity of weather

Environment monitor behavior considers only slope constraints. Robot does not navigate a region having a slope over a threshold level, $\alpha_{s\_thr}$. Force magnitude is computed using force parameter values from

Table **1**. Calculation of net force is presented below.

Set of slope over limits detected by robot

$S = \{s_1, s_2, ..., s_T\}$

Direction of repulsive force for a detected slope over limit is

$$\vec{f}_i = \left( \frac{P_{S_i} - P_R}{\left| P_{S_i} - P_R \right|} \right)$$

$P_R$ : Robot position in 3D

$P_{S_i}$ : i[th] Slope over limit position in 3D

Overall repulsive force direction will be

$$\vec{F}_{net} = \sum_{i=0}^{M} \vec{f}_i$$

### 3.2.4 Physical Situation Behavior

Primary function of physical situation behavior is detecting stuck conditions. In case of physical stuck, target reaching, communication, and adaptive wander behavior is inhibited. This behavior adds white noise to final response, the maximum value of which is taken to be $(10, 10, 10)$.

# CHAPTER 4

## Task Abstraction and Task Allocation Algorithm

Proposed system is based on executing tasks which can be resolved with multi-agent effort, it is necessary to define task executed by robots clearly. At this stage, a powerful abstraction is needed. Any task can be reduced into a well defined execution of sequence of sub tasks. Abstraction encapsulates the definition and execution of a task.

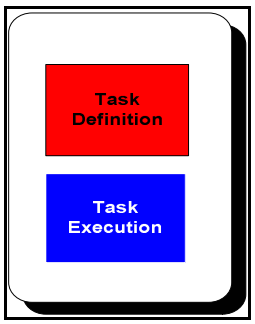


**Figure 32, Task Abstraction**

## 4.1 Task Definitions

In this proposed system there are four different type of tasks:

- Uncorrelated Tasks
- Correlated Tasks
- Synchronously Correlated Tasks
- Sequentially Correlated Tasks

Any robot has a primary sensor and depending on the primary sensor, its role differs. Moreover each robot is capable of sensing the nature of the task, i.e. whether the task is uncorrelated, correlated, synchronously correlated or sequentially correlated or not.

### 4.1.1 Uncorrelated Tasks

Uncorrelated task is a single task which can be resolved by a single robot. There are two important parameters of an uncorrelated task $t$ .

$t_i = \{\tau, \kappa, \Lambda\}$

$\tau$ : Task Type

$\kappa$ : Number of robots necessary executing for task type $\tau$

$\Lambda$ : Task Execution time, constant for all subtasks

for uncorrelated tasks $\kappa$ is always equals to 1

- Type of task: Robot senses the type of task using its primary sensors. Type determines which robot executes the task
- Task Time: Time is necessary for executing task

If a robot detects an uncorrelated task then it tries to execute it within task time. Robot does not generate any communication event since it can execute task by itself. Task symbol in simulator is shown below.

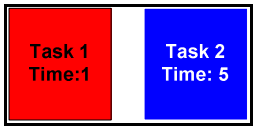


**Figure 33, Uncorrelated task representation simulator. Red rectangle stands for task type 1 with task time 1, whereas blue rectangle stands for task type 2 with task time 5.**

**Uncorrelated Task Execution Algorithm**

1. Robot executes task in task time.
2. Robot switch to free-state

### 4.1.2 Correlated Tasks

Correlated task is composed of at least two different subtasks and these subtasks are solved by different robots asynchronously at independent times.

$t_i = \{\tau, \kappa, \Lambda\}$

$\tau$ : Task Type

$\kappa$ : Number of robots necessary executing for task type $\tau$ , it is always 1. Because 1 robot can execute task in task execution time.

$\Lambda$ : Task Execution time, constant for all subtasks

Separated tasks can be executed independently. Let a correlated task, T, be composed of two tasks:

$T = \{t_1 \cup t_2\}$

$t_1$ can be executed by robot $r_1$ at time $T_1$ in $\Delta T_1$

On the other hand

$t_2$ can be executed by robot $r_2$ at time $T_2$ in $\Delta T_2$

If $r_1$ finds the source of task first then it generates communication event to invite a robot with type 2 based on task allocation protocol.

Correlated task execution does not require any kind of synchronization between executions of separable tasks. In Figure 34 a correlated task is shown. Relative height of different color rectangles shows the relative task time.

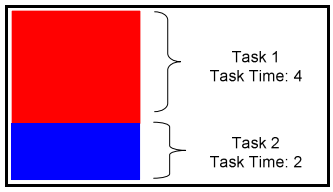


**Figure 34, Correlated task representation in simulator. Correlated task is composed of separable two subtasks: Task1 and Task2 with specified task time.**

***Correlated Task Execution Algorithm***

1. Robot $r_i$ separates the task into subtasks

   $T = \{t_1 \cup t_2 .. \cup t_M\}$

2. Robot $r_i$ sends communication request and distributes task

   $t_j, \quad 1 \le j \le M, \; j \ne i$

   to appropriate robots based on communication protocol.

3. Robot $r_i$ executes task $t_{i\Lambda}$ in task time
4. Robot switch to free-state.

---

### 4.1.3 Synchronously Correlated Tasks

Synchronously correlated task is used to define a task composed of at least two different subtasks but execution of these subtasks is strictly synchronous at the same time. Task execution processes is managed by the task manager. Manager decides for:

- Communicating with other robots to invite to join task execution
- Gathering enough number of robots with appropriate type
- Execution of task
- Release of task

Details of roles for the robot will be explained in detail in the communication section.

A synchronously correlated task is represented as follows

$$T = \{t_1 \leftrightarrow t_2 \ldots \leftrightarrow t_M\}$$

Task $T$ is composed of M subtasks $\leftrightarrow$ stands for synchronous task execution.

A synchronously correlated subtask has three parameters.

$$t_i = \{\tau, \kappa, \Lambda\}$$

$\tau$ : Task Type

$\kappa$ : Number of robots necessary executing for task type $\tau$

$\Lambda$ : Task Execution time, constant for all subtasks

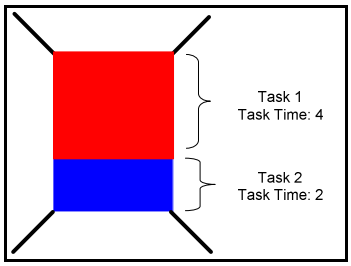


**Figure 35, Synchronously correlated task representation in simulator. Synchronously correlated task is composed of separable two subtasks: Task1 and Task2 with specified task time.**

In Figure 35 Synchronously correlated task representation is shown. Relative height of different colored rectangle stands for relative task execution time.

***Synchronously Correlated Task Execution Algorithm***

---

1. Robot $r_i$ separates the task into subtasks $T = \{t_1 \leftrightarrow t_2 .. \leftrightarrow t_M\}$
2. Robot $r_i$ sends communication request for tasks $t_j$, $1 \le j \le M$, $j \ne i$ for number of robots $t_{j\kappa}$ based on communication protocol.
3. If enough number of robots is joined, robot $r_i$ sends execute command for all robots.

   Robot $r_i$ monitors the task execution process.
4. If all tasks are executed synchronously then robot $r_i$ sends "go to free-state" command for all robots. If an error occurred then $r_i$ restarts the task from the beginning of the error. If the task is not completed within a timeout time for any reason then manager robot sends release command to all robots.
5. All robots switch to free-state.

---

### 4.1.4 Sequentially Correlated Tasks

Sequentially correlated task is used to define tasks requiring sequential execution process. There is an order for task execution; for starting each task has to wait the completion of its predecessor task

A synchronously correlated is represented as follows

$T = \{t_1 \rightarrow t_2 ... \rightarrow t_L\}$

Task $T$ is composed of L subtask $\rightarrow$ stands for sequential task execution.

A sequentially correlated subtask has for parameters.

$t_i = \{\tau, \kappa, \Lambda, Q\}$

$\tau$ : Task Type

$\kappa$ : Number of robots necessary executing for task type $\tau$

$\Lambda$ : Task Execution time, constant for all subtasks

$Q$ : Order of task.

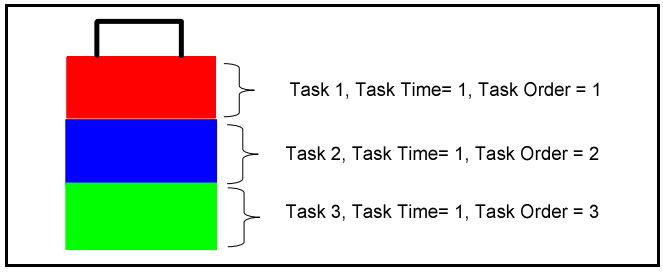


**Figure 36 Sequentially correlated task representation in simulator. Synchronously correlated task is composed of separable three subtasks: Task1, Task2, Task3 with specified task time and task order.**

***Sequentially Correlated Task Execution Algorithm***

---

1. Manager robot $r_i$ separates the task into subtasks as $T = \{t_1 \rightarrow t_2 ... \rightarrow t_L\}$.
2. Robot $r_i$ sends communication request for tasks $t_j$, $1 \le j \le M$, $j \ne i$ for number of robots $t_{j\kappa}$ based on communication protocol.
3. If enough number of robots is joined, robot $r_i$ sends execute command in the order of task execution for tasks, and monitors the execution of task. If

current task is completed successfully then robot sends execute command for the next task. In case of any error, manager robot restarts the current task. If task is not completed within a timeout then manager robot sends "release" command to all robots and all robots switch to free-state.

4. All Robot switch to free-state

---

### 4.1.5 Task Combinations

Any task can be formed by a combination of **uncorrelated tasks, correlated tasks, synchronously correlated tasks, sequentially correlated tasks**. Manager robot senses the type of task and separates it accordingly. In our system, a robot can sense only one type of task source, naturally it is expected that any robot in the system cannot achieve task separation but for simulation purposes each robot knows the type of the task and therefore how to decompose a task in to sub-tasks.

General form of set of task R is

$$R = \left\{ T_1, \overset{\cup}{T_2}, \overrightarrow{T_3}, \overset{\leftrightarrow}{T_4}, ...., T_L \right\}$$

Where

$T_1$ : Uncorrelated Task

$\overset{\cup}{T_2}$ : Correlated Task

$\overrightarrow{T_3}$ : Sequentially Correlated Tasks

$\overset{\leftrightarrow}{T_4}$ : Sequentially Correlated Tasks

Each task $T_i$ **is** also a task and formed by a combination of other subtask. Execution of set of task will from $T_1$ to $T_L$**.**

## 4.2 Task Allocation Algorithm (TAA) and Communication Protocol

Task allocation is one of the central issues of any multi-robot system. It determines how tasks are allocated among the participants of a robot/sensor network. TAA can be very different depending on the system architecture If a centralized system architecture is applied then completely different TAA is used as compared with decentralized systems. Task allocation algorithm depends also on the communication type and communication type is also dependent on task allocation algorithm, i.e. TAA and communication type should be compatible.

### 4.2.1 Communication Protocol

In the proposed system within communication range, it is assumed that all robots can communicate each other without any fault for exchanging information. Important issue regarding communication protocol is the addressing of messages, i.e. how messages will be distributed correctly to robots. In this system there are two addressing modes:

- Explicit addressing
- Broadcast addressing

In explicit addressing mode, messages are sent to a specific robot whereas broadcast addressing is used for sending message to robots within communication range. Explicit addressing is used for node to node communication.

We provide here as an example of communication message structure as follows:

**<Addressing Mode><Message ID><Robot ID>< Word Count><Word1>…<WordN><CS>**

**Table 2, Communication message structure**

| | |
|---|---|
| Addressing Mode | Mode of addressing, explicit or broadcast addressing |
| Message ID | Identification number of communication message |
| Robot ID | Identification number of robot to message sent. This is meaningful if addressing mode is explicit addressing. |
| Word Count | Total number of words in the message including word count and checksum. |

| | |
|---|---|
| Word1 | Data Word1 |
| WordN | Data WordN |
| CS | Checksum |

### 4.2.2 Task Allocation Algorithm (TAA)

TAA is based on known **decentralized market-based auction** algorithm described in [51] with some differences. Fitness calculation is completely different in this thesis. It is assumed that robots live in a virtual liberal system. Actions of the robots are based on their reliability and their fitness for tasks. Robots send their requests and responses via a communication infrastructure.

There are two basic roles that a robot can have: the role of a manager or that of a worker. The manager role is being manager of a multi-robot task, where management covers:

- Distributing tasks to the workers according to defined criterion
- Starting task &Ending task
- Monitoring execution of task

A Worker is responsible for the execution of a specific task. Worker robots should do following items:

- Execution of task
- Reporting of execution of task to manager
- Reporting its current status to manager

Roles of the robots are completely determined by a decentralized fashion. A robot can be a manager or a worker at different times. A robot is free for accepting its role, i.e. robot has not to be a worker or a manager even if it is requested. For instance, a robot can choose wandering around the environment instead of answering for a requested help depending on the requirements of requested help.

### 4.2.2.1 General Aspect of TAA algorithm

TAA has four phases:

- Task Request Phase
- Task Response Phase
- Task Acknowledge Phase
- Task Execution Phase

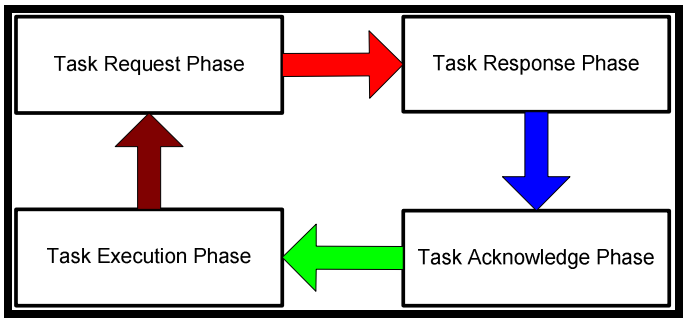


**Figure 37, Phases of task allocation algorithm**

### 4.2.2.2 Task Request Phase

In this phase, manager robot evaluates the current task and generates appropriate communication packet to request the help of other robots as worker robots.

TAA is achieved according to the fitness of each robot. Fitness is the minimum required reliability of robot to achieve the task. Each task has a fitness value evaluated by manager robot. Since task in the proposed system is a linked list of subtasks then each subtask has, value for different robots, a different or equal fitness which is determined by the conditions evaluated by manager robot. Fitness calculation is one of the important aspects in this TAA.

After the calculation of the fitness value for a task the manager robot transmits a call for gathering enough number of robots for that task execution using the communication infrastructure. It starts to send a task request packet (TRP) to a

range allowed by its communication hardware. Parameters of TRP, and their explanations are given in Table 3.

**Table 3, Communication task request packet parameters**

| **Parameter Name** | **Explanation** |
|---|---|
| PacketId | Unique packet ID of communication task request packet. Simulator interprets the TRP according to its identification number. |
| OwnId | Unique identification number of manger robot requesting. A response is sent to correct manager robot using this OwnId number. |
| OwnType | Type of manager robot's primary sensor. It is used by the robots candidates of being worker robot for request under consideration. |
| RequestedType | Type of robot's primary sensor. If requested type and robot's type is identical then robot should respond the manger robot. |
| RequestedTaskType | Type of task, i.e. whether task is uncorrelated, correlated, synchronously correlated or sequentially correlated. A robot request a communication task if its type is identical to TaskType and its fitness is enough. |
| SourceId | Unique identification number of source for which the request is to be generated. Simulator uses this parameter. |
| RequestTime | Time of request made. Request time is the time relative to manager robot. In this system, it is assumed that robots are not time synchronized. A robot synchronizes itself with manager robot with this parameter by setting the value of a counter dedicated to time synchronization to RequestTime value. |
| ExpirationTime | Time of expiration of request. Response made by worker robot candidates is accepted only time before ExpirationTime. A robot knows the expiration date and of course it will not make any response. |
| OwnFitness | Fitness of the owner robot. Worker robot candidates use this parameter before responding. |
| RequestedFitness | Required fitness of a worker robot candidate. Robots having fitness value greater then this value can be a worker robot candidate. |
| OwnPosition | Position of manager robot in 3D environment. A worker robot candidate knows the distance to the manger robot using this parameter. |
| CommunicationRange | Range of communication hardware owed by manager robot. |

For each subtask a TRP is generated and sent. After sending enough number of request for each subtask then the manager robot gets the responses from the worker robot candidates. The entire task request phase algorithm is listed below.

***Task Request Phase Algorithm***

---

1. Manager robot detects a task in general form shown below. Each task has also subtasks.

   $$R = \left\{ T_1, \overset{\cup}{T_2}, \overrightarrow{T_3}, \overset{\leftrightarrow}{T_4}, \ldots., T_L \right\}$$

2. For each subtask of $T_j$, $1 \le j \le L$ calculate fitness, generate TRP and sent TRP.
3. Evaluate responses within some time specified by expiration time. If no response is received and a timeout value is not exceeded then decrease the fitness value and re-announce the task, go to step 1. If enough responses are received then go to step 4. If the expiration time is reached then release task and go away from current location.
4. Switch to task acknowledge phase.

---

#### 4.2.2.3 Task Response Phase

In this phase, response is generated for incoming task request packets. This phase is only applicable for worker robots. In TRP, requested worker robot type (RequestedType) and requested fitness (RequestedFitness) are specified. If the robot type is identical to RequestedFitness and fitness is equal to RequestedFitness in TRP then that robot can respond this TRP in case of no limiting conditions, i.e. that robot may not be free then it cannot respond the incoming event even if it satisfies the requirements of current TRP.

In this phase, a robot as a worker robot candidate generates a task response packet (TRSP) and sends it. Parameters of TRSP and their explanations are given in

Table 4. A worker robot candidate can only respond in the form of one task respond packet. A worker robot candidate is a resource in the manager robot side. A resource cannot be shared by different manager robots having different positions. Moreover if a worker robot candidate sends two response packets to two different manager robots and if these manager robots acknowledge the response then of course there will be a collision. The worker robot candidate should select one of the managers as a master. There should be an additional re-acknowledge phase from worker robot candidate to the manager robot. By limiting response packet generation count to one avoids this problem and as well as any other problems.

**Table 4, Task response packet parameters and their explanations**

| Parameter Name | Explanation |
|---|---|
| PacketId | Unique packet ID of communication task response packet. It is equal to the identification number at corresponding TRP. |
| OwnId | Unique identification number of worker robot candidate. |
| ManagerId | Identification number of manager robot whose TRP is received. Simulator sends the response packet to correct robot by checking both PacketId and ManagerId. |
| OwnType | Type of worker robot candidate. |
| ResponseTime | Time at which response is generated. This is necessary for synchronizing with manager robot. A manager robot synchronizes itself with worker robot candidate with this parameter by setting the value of a counter dedicated to time synchronization to ResponseTime value. |
| OwnFitness | Fitness value of worker robot candidate relative to corresponding task request packet. |
| OwnPosition | Position of worker robot candidate in 3D environment. |

***Task Response Phase Algorithm***

1. If robot is available for communication request then task request packets are buffered into a linked list for some time. Let list of received task request packet be $L\{TRP\}$ shown below.

$$L\{TRP\} = \{TRP_1, TRP_2, \ldots, TRP_N\}$$

2. For each task request packet not expired in $L\{TRP\}$, generate a relative fitness list as

   $$L\{F\} = \{F_1, F_2, \ldots, F_N\}$$

3. Find the maximum of fitness, $F_k, 1 \le k \le N$, in the fitness list $L\{F\}$.
4. Generate and send task response packet for task request packet, $TRP_k$.

#### 4.2.2.4 Task Acknowledge Phase

This phase responses from worker robot candidates are evaluated and worker robots are selected. After task request phase, manager robot waits for some time for completion of task response phase. Within this time, worker robot candidates send responses to manager robot. In task acknowledge phase, depending on the value of fitness in the task response packets, received from worker robot candidates, acknowledges are generated.

For a given task request packet, there will be a list of responses sent by worker robot candidates. Manager robot acknowledges the response having the maximum fitness. For each task or subtask only one worker robot candidate is acknowledged. Since a worker robot candidate sends response to only one task request then there will be no resource allocation conflict. In Table 5, description task acknowledge packet (TAP) is given.

**Table 5,Task acknowledge packet parameters and their explanations.**

| Parameter Name | Explanation |
|---|---|
| PacketId | Unique packet ID of communication task acknowledge packet. |
| TaskResponsePacketId | Task response packet ID for which acknowledge is generated |
| ManagerId | Identification number of manager robot |
| WorkerRobotId | Identification number of worker robot |
| AcknowledgeTime | Time of acknowledge time. This time is used for synchronization purposes. |
| ExpirationTime | Time after which acknowledge is invalid. Worker robot should reach to its destination pointed by task request packet |

**Task allocation** among the worker robots are described below. Manager robot's task request packet list $L\{TRP\}$ is shown below.

$$L\{TRP\} = \{TRP_1, TRP_2, \ldots, TRP_N\}$$

For each element of this list, at least one response is received from the worker robot candidates.

For each task request packet there will be a list of task response packets

$$L\{TRSP\}_j = \{TRSP_1, TRSP_2, \ldots, TRSP_{S_j}\}, 1 \leq j \leq N$$

Where

$S_j$ is the size of task response packet list $L\{TRSP\}_j$ for task request packet $TRP_j$

For each task request packet $TRP_i, 1 \leq i \leq N$ in $L\{TRP\}$ maximum of fitness value in the task response packet list $L\{TRSP\}_i$ is found. A task acknowledge packet is generated and sent for task response packet maximizing the fitness. Overall acknowledge phase algorithm is described below.

***Task Acknowledge Phase Algorithm***

---

1. For manager robot's task request packet list $L\{TRP\}$

   $$L\{TRP\} = \{TRP_1, TRP_2, \ldots, TRP_N\}$$

   and corresponding task response packet list

   $$L\{TRSP\}_j = \{TRSP_1, TRSP_2, \ldots, TRSP_{S_j}\}, 1 \leq j \leq N$$

   If for each task request packets in $L\{TRP\}$ is responded by at least one worker robot candidates then apply task allocation algorithm described above.

   Otherwise clear all of the buffers and exit from task acknowledge phase algorithm.
2. Generate and send task acknowledge packet to worker robots
3. Wait worker robots arrival for some time

---

#### 4.2.2.5 Task Execution Phase

Task execution phase is the final phase of the task allocation algorithm. In this phase worker robots inform the manager robot on their arrival to location. Execution of task is started by the command of the manager robot if all of the worker robots are in the region of task execution. Periodically worker robots report the current status of the task execution. Once the task is completed then the manager robot sends release command to all robots.

Overall task execution phase algorithm is described below.

***Task Execution Phase Algorithm***

---

1. For manager robot's task acknowledge packet list $L\{TAP\}$

   $$L\{TAP\} = \{TAP_1, TAP_2, \ldots, TAP_N\}$$

   Received list of inform messages from worker robots $L\{TAIP\}$

   $$L\{TAIP\} = \{TAIP_1, TAIP_2, \ldots, TAIP_P\}$$

   If for all elements in $L\{TAP\}$ is matched with the elements in the $L\{TAIP\}$ one to one then all workers robots are in the vicinity of task execution region. If one to one match is not achieved then some of workers robots are missing then manager robot sends release command for all robots.
2. Send execute command for all robots.
3. Receive and evaluate the task execution report from the worker robots. If an error occurred then send release command for all robots.
4. If all tasks are completed successfully then send release command for all robots.

---

#### 4.2.2.6 Complexity of Task Allocation Algorithm

The manger robot has to make many relative fitness calculations. The complexity of the task allocation algorithm is the complexity of the task

acknowledge phase algorithm. In the worst case the manger will be obliged to make n comparisons to evaluate the maximum fitness value of received response packets. Complexity of task allocation for manager is $O(n)$. On the other hand, worker robot needs only calculate its fitness, so the complexity of task allocation for worker robot (bidder) is $O(1)$ [46].

#### 4.2.2.7 Fitness Evaluation

The fitness calculation is an important issue of the task allocation algorithm. Depending on the value, task allocation fairness can be affected, it is necessary to define the criterion of fitness calculation. In this system, there is no single fitness calculation but fitness is calculated relative to some conditions and we will consider each of conditions separately in subsections 6.5.2.

The fitness value depends on the following items and all conditions stem for those dependencies:

- Target reaching frequency
- Communication failure frequency
- Obstacle avoidance success frequency
- Obstacle avoidance failure frequency
- Coverage
- Distance from task location

**Target reaching frequency** defines how frequent is the target reaching. If time interval between target reaching is small then robot's target detection procedure is working fine. Target reaching frequency includes tasks allocated via communication.

**Obstacle avoidance success and failure** also shows the current status of the robot's obstacle detector sensors. If there are problems with these sensors then obstacle avoidance, which has a deep impact on robot overall performance will not be done accurately.

**Communication failure frequency** is also an important parameter. Communication task failures show a problem about robots communication hardware, or some cross-couplings with other structures.

**Coverage** is a valuable parameter since it shows how robot navigates, and its physical situation such as motors. Coverage is computed using a time-varying coverage map. High coverage value is always more preferable as compared with low coverage values. This fitness measure enables fair task allocation.

**Distance from a task location** is used to describe effect of distance. Distance is important since it means time and energy consumption. A worker robot candidate will calculate its fitness by also taking into account its distance from the manager robot.

#### *4.2.2.7.1 Fitness Calculation*

Fitness of the robot is determined by the sum of individual fitness values for target reaching, target reaching frequency, obstacle avoidance success and failure rate, communication task success and failure rate, coverage, distance from task location. So it is necessary to develop a well defined fitness calculation for the above parameters. For this purpose two different fitness function types are adopted. A Function for type 1 is used to measure the fitness of success parameters:

- Target reaching frequency
- Obstacle avoidance success frequency
- Coverage fitness
- Distance fitness

On the other hand a function for type 2 is used to measure the fitness of failure fitness:

- Communication failure frequency
- Obstacle avoidance failure frequency

The reason behind the selection of two different fitness functions is the speed of ascent or descent: error fitness functions should decrease more rapidly as compared to success fitness functions.

#### *4.2.2.7.2 Fitness Function for Success Situations*

For target reaching frequency, obstacle avoidance success frequency, coverage fitness, and distance fitness calculation the following fitness function is proposed.

$$F_s(x) = u(x)F_1(1-e^{-f_1 x}) + u(-x)F_2(1-e^{f_2 x})$$

where,

$F_1$ and, $F_2$ are boundary values, $f_1$ is increase rate1 and $f_2$ are time constant or increase rates.

$x$ is defined as total number of events / time elapsed

$$F_1 > 0, F_2 < 0$$
$$f_1 > 0, f_2 > 0$$

$u(t)$ is unit step function, i.e.

$$\left\{\begin{matrix} u(x) = 1 & x \geq 0 \\ u(x) = 0 & x < 0 \end{matrix}\right\}$$

The above fitness function is preferred since it is bounded, i.e.

$$\infty \leq x \leq \infty$$
$$F_1 \leq F(x) \leq F_2$$

Moreover the rate of increase and decrease can be controlled by adjusting the values of $f_1$ and $f_2$.

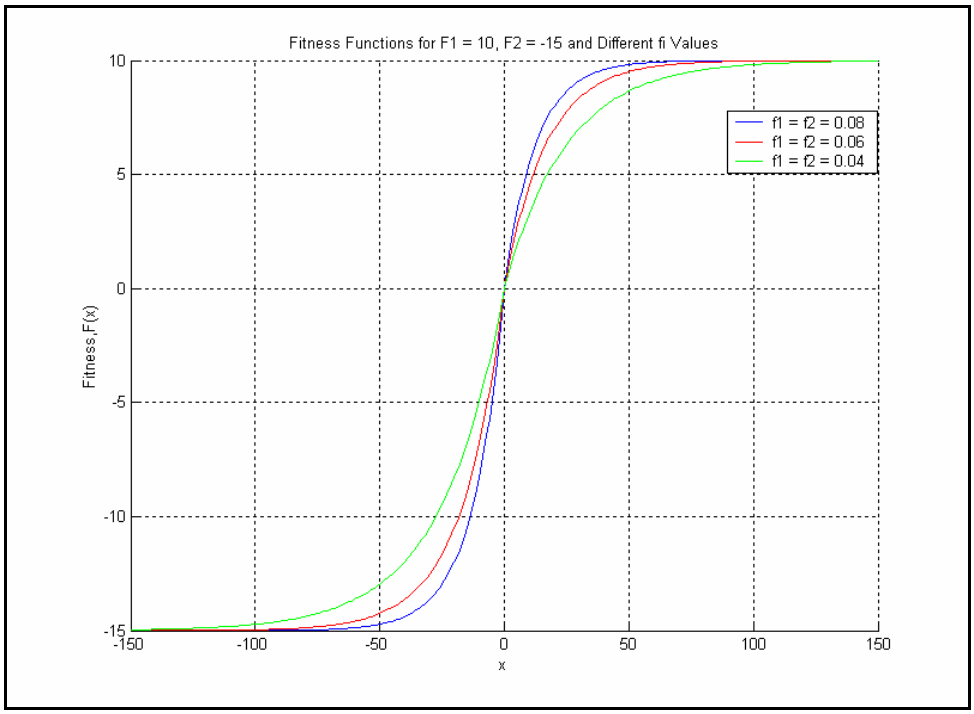


**Figure 38, Fitness functions for $F_1 = 10,\ F_2 = -15$ and different increase rate values.**

In Figure 38 different fitness functions are plotted for $F_1 = 10,\ F_2 = -15$ and different increase rate values. The Increase rates $f_1 = f_2 = 0.08$ generate the maximum increase. However for $f_1 = f_2 = 0.04$ the increase rate is slow as compared to the first case. As a result, the introduction of this fitness function gives the flexibility of adjusting

- Two limit values on both side of the fitness parameter axis
- Increase rates to reach boundary fitness values.

Selection of values $F_1, F_2, f_1, f_2$ is important issues among all these values determine the performance of task allocation among robots, since task allocation is made according to fitness values of robots as mentioned earlier. If a robot overestimates its fitness then the manager robot will allocate the task for that robot. Proper adjustment of these parameters will be achieved based on extensive simulations. Selection of the fitness function parameters can be evaluated by making some observations. For instance to select $F_1, F_2, f_1, f_2$ for target reaching, it is very

important to know the maximum number of targets that a robot can face statistically in a bounded region. These values can be adaptively updated by the robots. Another method can be based on selecting $F_1, F_2$ values large such that robot fitness function never saturates. Control of rate of ascent and descent to boundary values is also important. $f_1$, and $f_2$ values can be selected in cooperation with $F_1, F_2$ such that fitness parameter value reaching at 50% and 90% of boundary values should be acceptable.

$$F(x) = \begin{cases} F_1(1 - e^{-f_1 x}), & x \geq 0 \\ F_2(1 - e^{f_2 x}), & x < 0 \end{cases}$$

For positive values of fitness parameters, fitness parameter getting $0.5F_1$ and $0.9F_1$ are

$$x_{0.5}^{+} = \frac{ln(2)}{f_1},\ x_{0.9}^{+} = \frac{ln(10)}{f_1}$$

For negative fitness values

$$x_{0.5}^{-} = -\frac{ln(2)}{f_2},\ x_{0.9}^{-} = -\frac{ln(10)}{f_2}$$

By choosing only $x_{0.5}^{+}$ and $x_{0.5}^{-}$, $f_1, f_2$ values can be selected easily.

#### *4.2.2.7.3 Fitness Function for Failure Situations*

For the communication failure, and obstacle avoidance failure fitness calculation the following function is proposed.

$$F_f(t) = u(t)F_1(e^{-f_1(t-t_0)}),$$

where $F_1$ is the maximum value of the fitness, $f_1$ is the descent factor, $t_0$ is the time of last failure event, and $t$ is time.

In this case $f_1$ is selected depending on the speed of descent. $f_1$ is selected to obtain a half fitness within a time interval. A $t_{0.5}^{+}$ is selected such that it satisfies the following equality.

$$F_f(t_{0.5}^{+}) = \frac{F_1}{2}$$

For this condition, value of $f_1$ will be

$$f_1 = \frac{\ln(2)}{t_{0.5}^{+} - t_0}$$

Large descent factor means that effect of failure will be forgotten in a few seconds, whereas high values means that effect of failure will continue very long time.

#### *4.2.2.7.4 Parameters Settings*

Parameter settings of fitness items are the central issue of fitness calculation. In multi-robot system it is hard to find an objective function to represent the cost of an action. Parameters are selected by prior estimates about environment and relative importance of fitness items. In Table 6, parameters of different fitness items are given. For all fitness functions, F2 is equal to zero, therefore $x_{0.5}^{-}$, $x_{0.9}^{-}$ is not meaningful for this case.

The unit of $x_{0.5}^{+}$ is number of target / time elapsed, F1 is a unitless parameter.

**Table 6, Fitness items parameters**

| **Fitness Functions** | Abbreviation | F1 | $x_{0.5}^{+}$ | $t_{0.5}^{+}$ |
|---|---|---|---|---|
| Target Reaching Frequency | $F_{trf}$ | 20 | 0.01 | |
| Communication Failure Frequency | $F_{cff}$ | 50 | NA | $t_0 + 500$ |
| Obstacle Avoidance Success | $F_{oas}$ | 5 | 0.20 | |
| Obstacle Avoidance Failure | $F_{oaf}$ | 50 | NA | $t_0 + 100$ |
| Coverage | $F_c$ | 5 | 0.15 | |
| Distance from task location | $F_d$ | 5 | 25 | |

**The Target reaching frequency fitness** is used to measure how frequent a robot reaches targets. This fitness item is an additional fitness to target reaching

fitness. It is necessary because it shows the average target reaching. If the frequency is high then robot finds its targets in a small amount of time and saves energy and time.

$$\text{Target reaching frequency} = \frac{N}{t_{elep}}$$

$N$ : Total Number of target reached up to current time

$t_{elep}$ : Time elapsed since deployment of robot

The frequency of target reaching is computed by dividing the total number of targets reaches to time elapsed from the beginning robot deployment. In this system, it is assumed that average target detection time is 100 seconds. For this value, target reaching frequency fitness reaches half of the maximum. As time goes on, the value of this fitness item decreases.

**The Communication failure frequency fitness** is used to represent the effect of communication task failures. Higher values of this fitness show that robot is not successful in execution of communication tasks. In case of failure of a communication task, robot is punished. Its overall fitness is decreased accordingly.

Whenever a communication failure takes place, the robot is temporally not allowed to assigned task allocation for some time. This is achieved by adjusting the F1 value of the robot greater than sum of maximum of other positive fitness functions.

The sum of success fitness functions excluding distance fitness is $F_{trf} + F_{oas} + F_c = 30$. Since total fitness is calculated by subtracting failure fitness functions from success fitness functions, this kind of F1 setting will guarantee that the total fitness is less than zero so that robot cannot make any communication, i.e. F1 > 30, i.e. robot is punished so that it cannot get involved in a task allocation after the occurrence of an communication error. $t^{+}_{0.5}$ is selected as 500 seconds. This means

that if success fitness functions is not greater than 25 (F1/2), the robot will not make any task allocation within 500 seconds.

**The Obstacle avoidance success fitness** is used to monitor the obstacle avoidance performance of the robot. A fitness function should exist to represent the effect of the obstacle-dense region since it is difficult to navigate in obstacle dense environments. If robot avoids obstacles, its fitness is increased the maximum obstacle avoidance success fitness is set to5 and it is small compared to other fitness values.

The obstacle avoidance fitness frequency is calculated by dividing the total number of obstacles avoided by time elapsed.

**Obstacle avoidance failure frequency fitness** is important since it shows the status of sensors used for obstacle avoidance or any other side sensors. In case of obstacle avoidance failure, robot is punished since obstacle avoidance failure is an unacceptable error. F1 value is selected such that when an obstacle avoidance error is occurred then the robot cannot get involved in task allcoation. It is again selected as 50, and $t_{0.5}^{+}$ time is 100 seconds. Obstacle avoidance failure is forgotten more rapidly as compared with communication failure fitness descent speed which is because obstacle avoidance failures are considered as occurring more frequently as compared to communication failures.

**Coverage fitness** is also important because it shows the ability of wandering around the environment. Coverage fitness is estimated over temporal coverage, such that a time varying coverage map is used. Temporal coverage is computed from the position information belonging to the last 80 seconds of data. Map timeout is kept small to increase the reactivity to short time events such as target reaching. A robot making more target reaching will result in a decrease in coverage fitness, because robot will loose time for execution of tasks. Coverage fitness function is implemented to adjust the fairness of task allocation. While the robot's target reaching is increasing, its coverage fitness will decrease, and as a result, the sum of

success fitness functions will be adjusted. Moreover this fitness function also takes a role in fault tolerance of robots.

**Distance from task location** is a measure of distance from a location. In a communication task, if a robot is far from a task location then fitness of this robot is decreased. Robots near the task locations are preferred to those far from the task location. This is very logical since it leads to time and energy consumption reduction if other fitness values are compatible. F1 value of this fitness function is taken as small in order not to exaggerate the effect of distance from task location. Exaggeration will result in poor fault tolerance.

The total fitness is sum of all fitness values.

$$F_{success} = F_{trf} + F_{oas} + F_c$$

$$F_{failure} = F_{ctf} + F_{oaf}$$

$$F_{total} = F_{success} - F_{failure} - F_d$$

In Figure 39, Figure 40, and Figure 41 plot of each fitness functions are shown. For failure case a single failure at $t_0 = 200$ seconds is simulated.

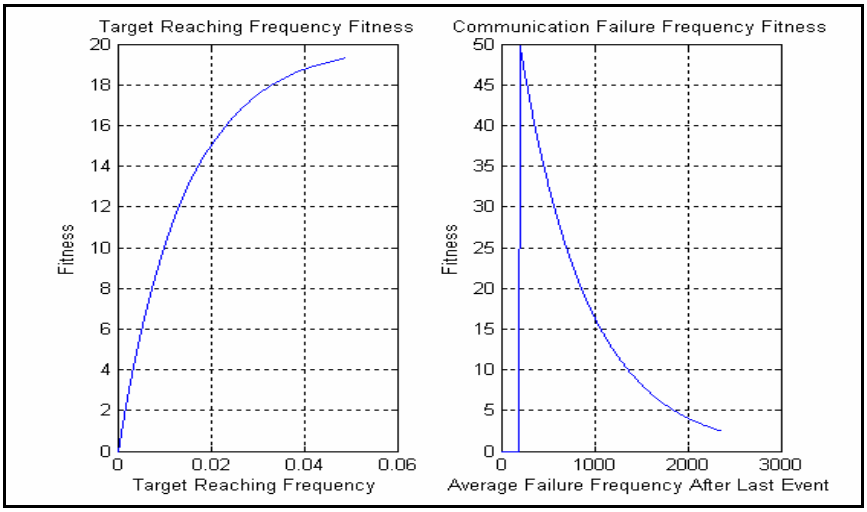


**Figure 39, Target reaching frequency, and communication failure frequency fitness plot with parameters in Table 6. At $t_0 = 200$ seconds a communication failure is occurred.**

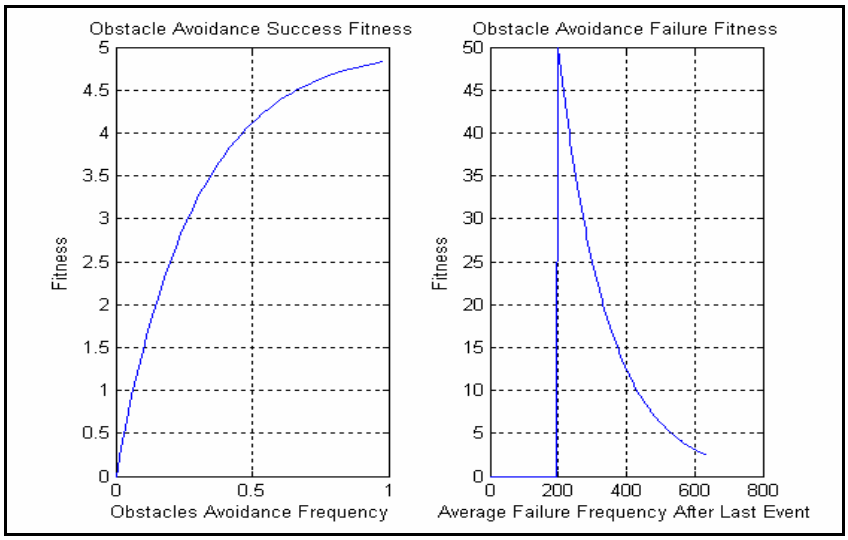


**Figure 40 Obstacle avoidance success, and failure frequency fitness plot with parameters in Table 6. At $t_0 = 200$ seconds a obstacle avoidance failure is occurred.**

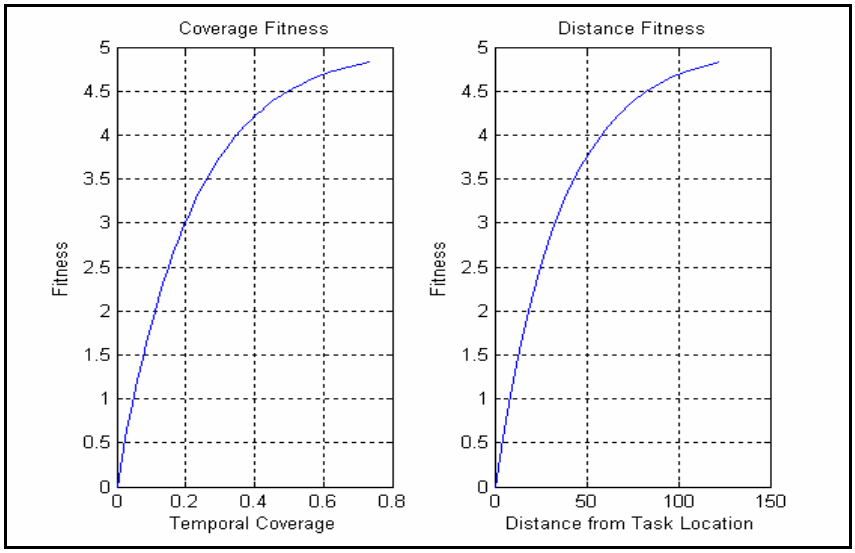


**Figure 41 Coverage & distance from a task location with parameters in Table 6**

# CHAPTER 5

# Developed Simulation Environment

## 5.1 Introduction

A Simulation software (SS) is required to be implemented, and testing the proposed cooperative multi-robot system architecture. For this purpose, we design a visual simulator in Windows XP environment. SS is developed using Microsoft Visual C++ 6.0 compiler using MFC's (Microsoft foundation classes) single document interface (SDI) model. Since object oriented approach is used, simulation environment is quite modular. Moreover, it is easy to use the simulator with its friendly graphical user interface (GUI). The reader is provided by a CD at the end of this thesis to experiment with the generated simulator. Total number of codes of the simulator exceeds 31.000 lines. The simulator is documented with Doxygen 1.4.3, a software documentation tool which is also provided in the CD added to this thesis. Matlab R13 is also used as a data analysis, and modeling tool to interpret simulation results.

Although many simulation environments are based on 2D environment model, implemented software is using 3D environment model. Robots are deployed in a 3D environment. Usage of 3D environment enables the realistic physical environment. Even if the proposed system does not cover dynamic model of robots, user can easily integrate the physics of robot with the simulation environment. The proposed system uses line of sight (LOS), and slope check in 3D environment.

3D infrastructure is designed using Microsoft's DirectX technology. DirectX offers a powerful 3D software development kit for users. DirectX 9.0 is used in this

simulator. Necessary transformations (rotation, translation, and scaling) are also done with DirectX.

Robots are moving in a 3D base. Base is a 3D mesh structure defined by triangular strips. Since it is not easy to design a 3D mesh manually, auxiliary software is used. Discreet's 3DS Max 6.0 software is an excellent 3D software for mesh design. Meshes designed with 3DS Max 6.0 are exported to the DirectX environment using an additional plug-in software. PandaDXExport6 is used to convert 3DS Max 6.0 binary file to DirectX's X file format. After loading designed mesh to the simulator, it is preprocessed to increase efficiency in terms of computational time. Mesh information in DirectX is represented with vertex buffers, and index buffers, it is not suitable for fast search, and deployment operations. A prepossessing is applied after loading the raw mesh information.

The main properties of this simulation software are:

- Modular object oriented software approach using MS Visual C++ 6.0, and DirectX technology.
- Simulation is achieved over single thread.
- 32 bit floating point representation
- Friendly graphical user interface (GUI). User can easily modify, and monitor system.
- 3D Environment. This type of model rarely used in many robot simulators.
- User can use this simulator as a software development kit. It is easy to add new components, and environment models. Surface of moon can be simulated. It is up to user's programming talent.
- Realistic 3D environment model can be used to integrate the robot's real dynamics to the system. More realistic simulation may be achieved.
- Adding robots to the system with desired primary, and auxiliary sensors
- Modifying robot settings even in run time

- Adding uncorrelated, correlated, sequentially correlated, synchronously correlated task to the environment
- Adding obstacles to environment. User can design complex environments using obstacles. Inside of a factory or a very large terrain formed by houses or small hills can also be modeled.
- Error simulations for fault tolerant analysis

Screen shoot of the simulation software is show in Figure 42. In this figure, robots, tasks, obstacles are shown in 3D environments. In Figure 43, top view editor of the simulator is shown. A top view editor is implemented since it is difficult to deploy components in a 3D environment. Moreover, rendering object in 3D is time consuming. Even if 3D rendering is disabled, user can monitor system using 2D editor.

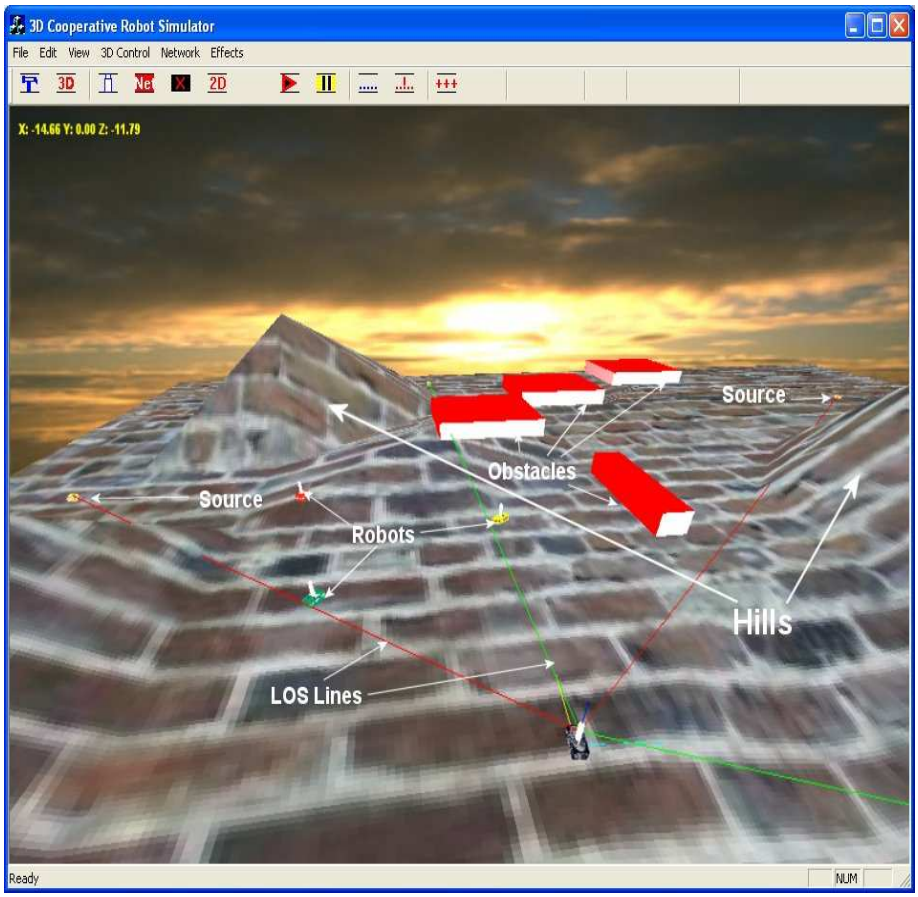


**Figure 42, 3D view of Simulator**

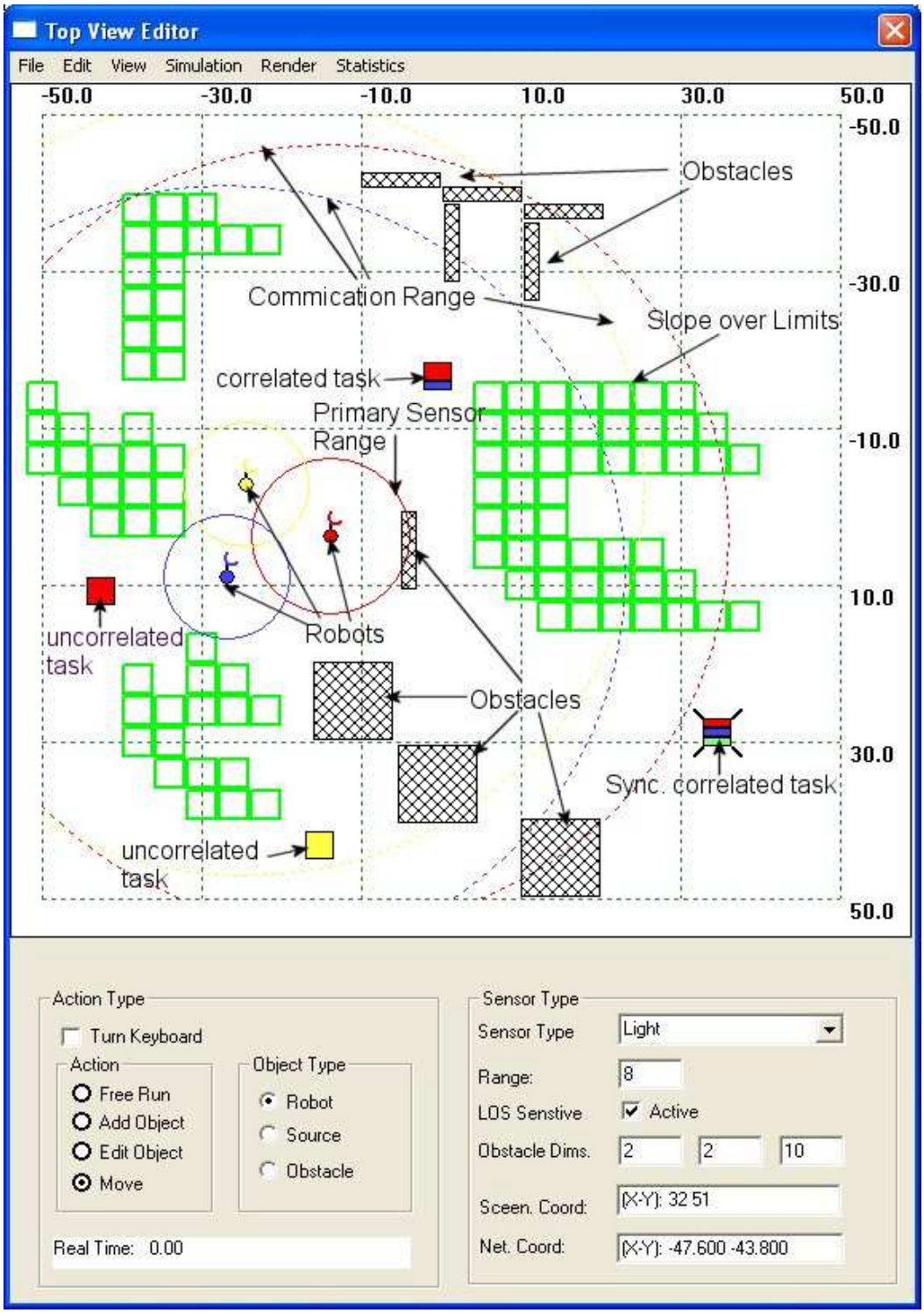


**Figure 43, Top View Editor of Simulator**

## 5.2 Classes and Class Hierarchy of Simulation Environment

There are many classes implemented for satisfactory simulation. List of classes and the class hierarch are shown in Figure 44. Classes can be categorized into 3 main categories:

- Environment, and DirectX classes
- Sensor network objects (robots, tasks, obstacles, communication) classes
- Display classes

Main application class, CMyDirect3DApp, is derived from CDirectDevice which is devoted to DirectX device generation. The purpose of this class is construction of 3D environment from 3D meshes. CSensorSoureNet class is used to represent sensor network. This class implements a virtual sensor network platform for the robots. All of robots, tasks, and obstacles are contained as attributes in this class. Simulation of sensor network is achieved incrementally using this class. CRobot is used to represent robots. It includes a complete description of a robot. CSource is used to implement 4 different types of tasks. Obstacles are described using CObstacle class. CCom class encapsulates all of list required for communication. Moreover, it is used to implement communication protocol. There are also classes, CEnvEditor, CTopEdit, CStats, implemented for display purposes.

There are other classes which is not included hierarch figure. Total number of classes, and structures used in simulator is 171.

## 5.3 Flow of Simulation

There are 9 phases of simulation listed as below.

- Initialization & environment loading phase
- Deployment of objects phase
- Object Detection phase

- Behavioral evaluation and coordination phase
- Updates phase
- Robot status generation phase
- Task allocation phase
- Movement of robot in 3D environment phase
- Rendering phase

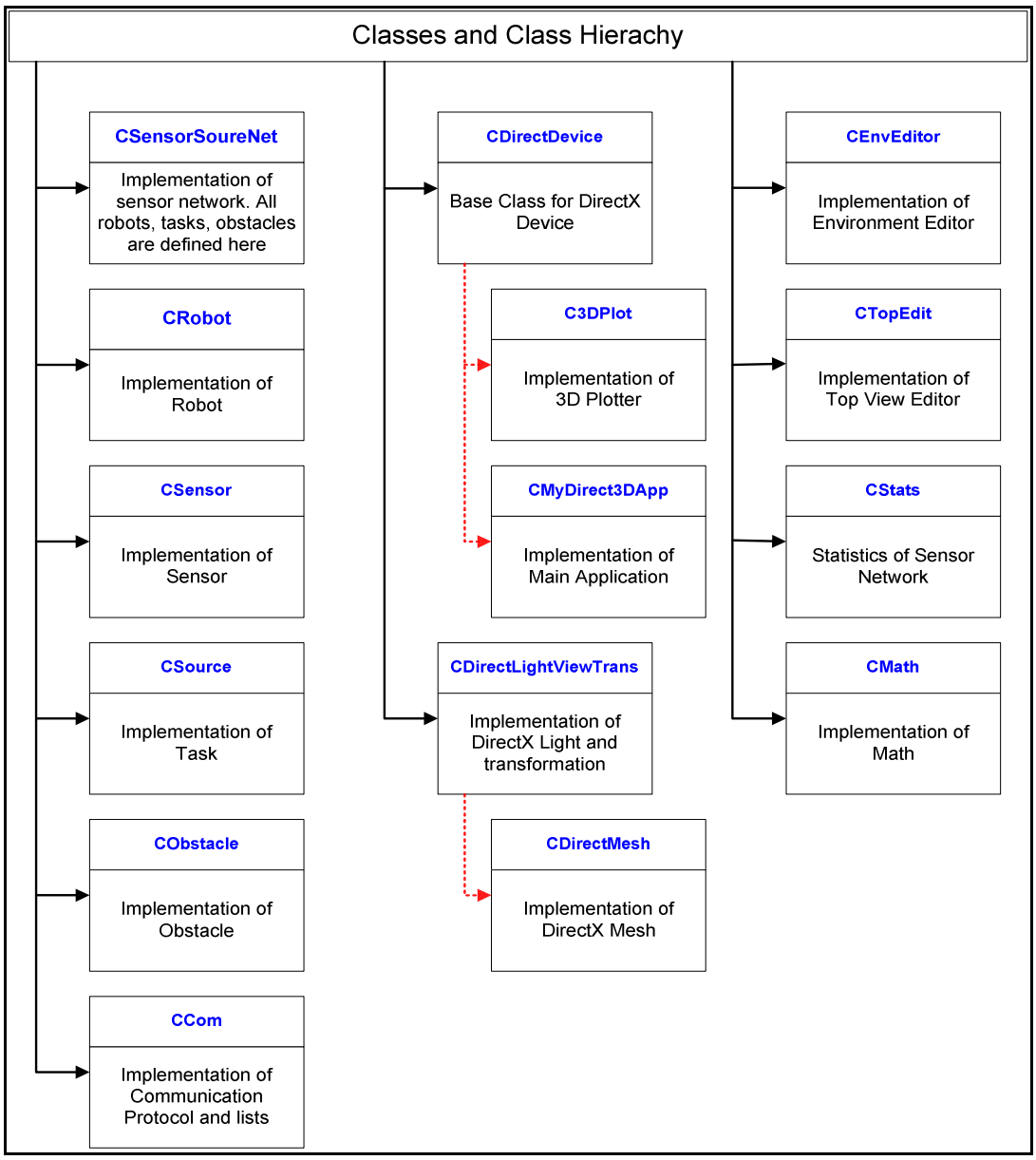


**Figure 44, Classes, and class hierarchy used in simulator**

For each simulation instant these phases are executed. Flow of simulation is given in Figure 45.

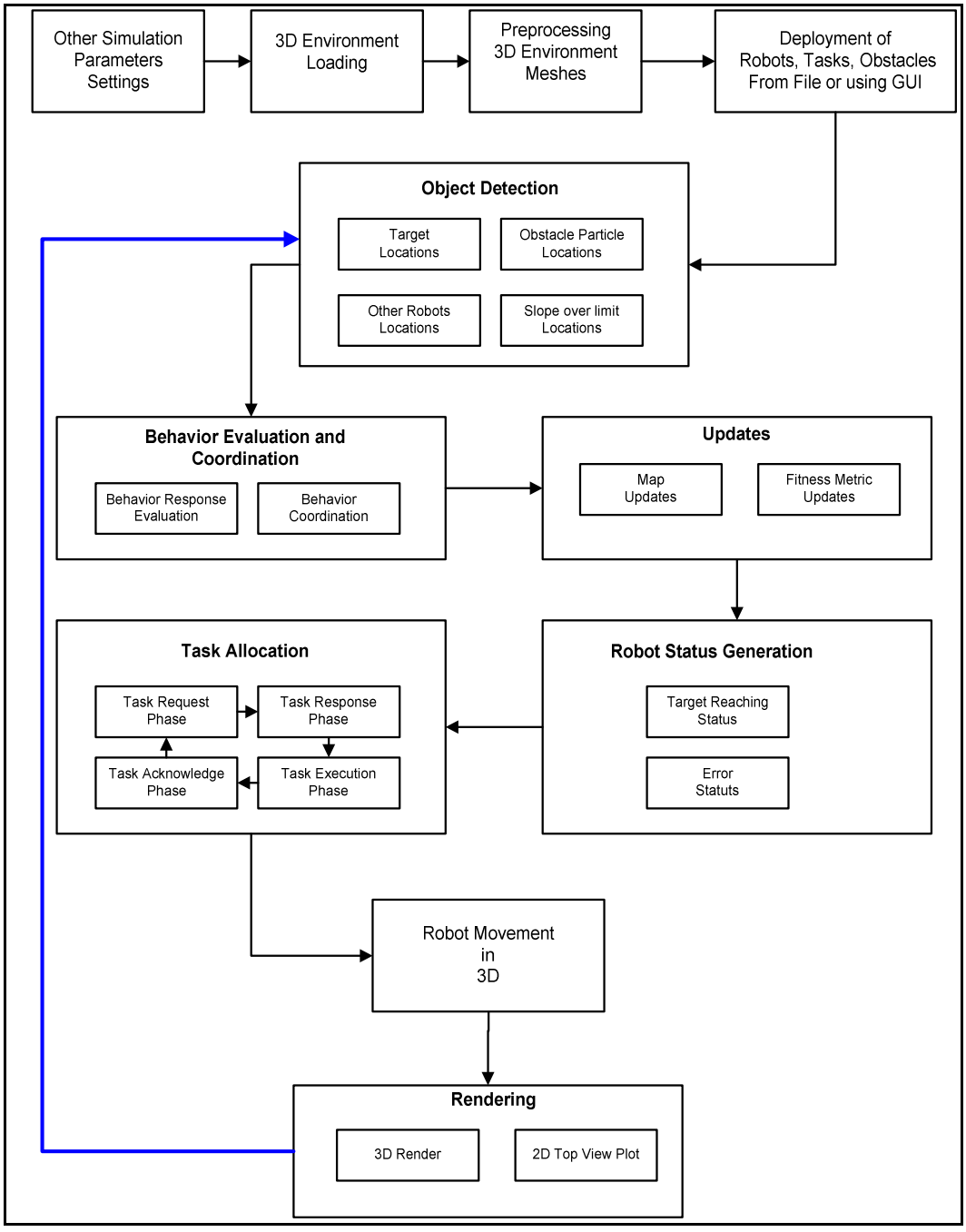


**Figure 45 Flow of simulation for each robot**

**Initialization & environment loading phase: G**lobal environment variables such as simulation step size, environment size, limits are set to the desired values. The 3D environment is loaded and preprocessed according to the simulation settings.

**Deployment of objects phase**: Robots, tasks, and obstacles are deployed to the desired locations in the environment with desired settings. Objects can be deployed either from files saved previously or from graphical user interface.

**Object detection phase**: At each simulation instant deployed robots detect the objects in the environment. Detectable objects are targets, tasks, obstacle & locations having excessive slope.

**Behavior evaluation and coordination phase**: Each robot makes its behavior response evaluation using detected objects. Commands for actuators are generated. Behavioral coordination is achieved depending on the control strategy. Default robot control strategy is our proposed hybrid control architecture, but user can set the control strategy of robots to the pure subsumption or motor schema strategies.

**Updates phase**: Robot's maps and fitness metrics are updated in this phase. Updates phase is one of the time consuming phase because depending on the resolution of maps, the size of the maps change considerably.

**Robot status generation phase**: Robot updates its status. Default status of each robot is set to free. If a robot detects a task source then it switches its status to from free to not-free state, and does not respond any help requests. Moreover, error checks are made regarding the data consistency of robot.

**Task Allocation Phase**: If status generated in previous phase requires a task allocation process then, task allocation process is started, and continued according to task allocation algorithm. Response to a task allocation request is also made here for worker robot candidates.

**Movement of robot in 3D environment phase:** Robot moves on the 3D terrain along the specified direction with specified speed. In this phase the next location of the robot is found in the 3D environment. If a crash condition, which is an extraordinary error is occurred then it is reported.

**Rendering Phase:** 3D structures are rendered moreover top view editor is updated. Since rendering is a quite time consuming process, for fast simulations, rate of rendering can be decreased or it can be completely disabled.

## 5.4 Communication Medium

Messages coming from robots are transmitted over a virtual medium. Each robot has a class "CCom" containing the linked lists for communication protocol, and events. If a robot decides to call another robot for its help then it starts to transmit the communication request, and adds a request package to its corresponding communication request list. At each simulation instant, "CSensorSourceNet" class analyzes the lists of the robots, queues, and sends to the other robots appropriately. If robot A is in the communication range of robot B then, a message from robot B is transmitted to the robot A. It the proposed system it is assumed that virtual communication medium is perfect with infinite bandwidth.

## 5.5 Robot Sensorial Structure in Developed Simulator

Sensors have some limited ranges and limited accuracy. Depending on the quality of sensor, accuracy and range may differ substantially. Auxiliary sensor range is generally shorter than primary sensor ranges. Since sensors are not perfect, to model the uncertainty some noise should be added on top of expected measurements. In this simulation 5% of desired range is added as Gaussian noise.

$$S_R = \bar{R} + \Delta R$$

$$\Delta R \sim N(0.05\,\bar{R})$$

Where

$S_R$ : Real Sensor Range

$\bar{R}$ : Desired Sensor Range

$\Delta R$ : Gaussian Noise

There are three important concepts regarding sensors:

- Range of sensor
- Accuracy of sensor
- Line of sight (LOS) dependence

**LOS** dependence may differ for different sensors. LOS check algorithm is simple but its computational complexity is high. LOS check is achieved by searching intersection of LOS line and terrain.

In Figure 46, 2D polar plot of robot sensor range is given. Red line shows the desired range, blue line show realistic range. In this case 10% percent noise added to the desired range, 10 units. This range plot is given in 2D but in actual case, robot lives in 3D world and real range plot will be a sphere added some noise on top of it.

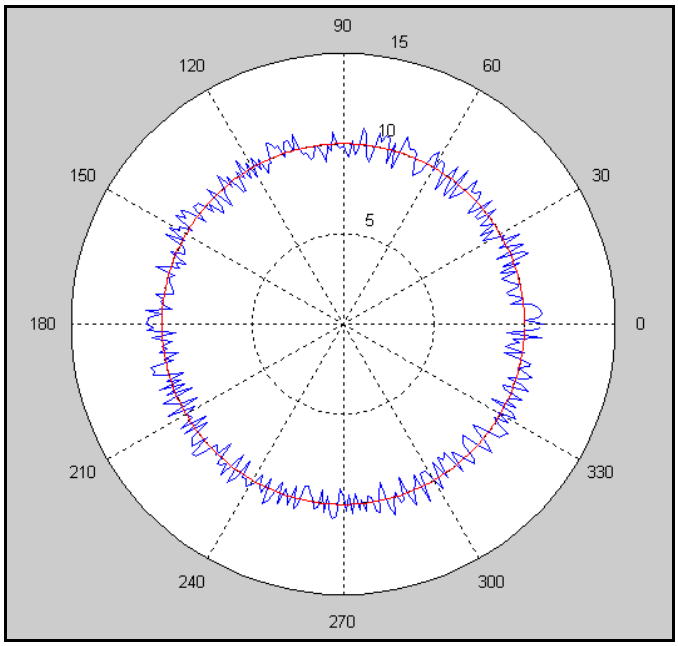


**Figure 46, 2D Noisy Range Drawing**

# CHAPTER 6

# Experiments & Results

This part will cover the experiments on following issues

- Motion control of robots
- Coverage
- Effect of communication
- Single and multi-robot task execution
- Effect of fitness calculation on fair and fault tolerant

## 6.1 Robot Motion Control of Robots

In this part, results will present the implementation of basic behaviors. This part will include graphical illustrations of following behaviors:

- Obstacle Avoidance Behavior
- Robot Separation Behavior
- Heuristic Wander Behavior

Other behaviors are not included because they will be discussed in more detail in different sections.

### 6.1.1 Obstacle Avoidance Behavior

Obstacle avoidance behavior is one of the important behaviors. Performance of this behavior affects the performance of the system directly. Obstacle avoidance behavior is evaluated for different basic obstacle shapes which are convex, concave, partially concave, strongly concave shapes. The avoidance algorithm is tested in robots reaching a target while avoiding obstacles.

In Figure 47, obstacle avoidance path from a rectangular shaped obstacle is shown. Actually rectangular shaped obstacles can be classified as partially concave obstacles. It is clear that avoidance is successful. Figure 48 shows the path of avoidance from partially concave shaped obstacle which is a difficult task to avoid. The Proposed obstacle avoidance algorithm handles avoidance process successfully. In Figure 49 and Figure 50, avoidance from concave and strongly concave shaped obstacles are shown respectively. The most difficult avoidance situation is avoiding from strongly concave shapes due. However the algorithm successfully avoided this obstacle and reached the target successfully.

Our proposed method easily handles this kind of almost closed concave shaped obstacles owing to its behavioral integration. Using wandering concept together with other basic behaviors enables fast and secure avoidance.

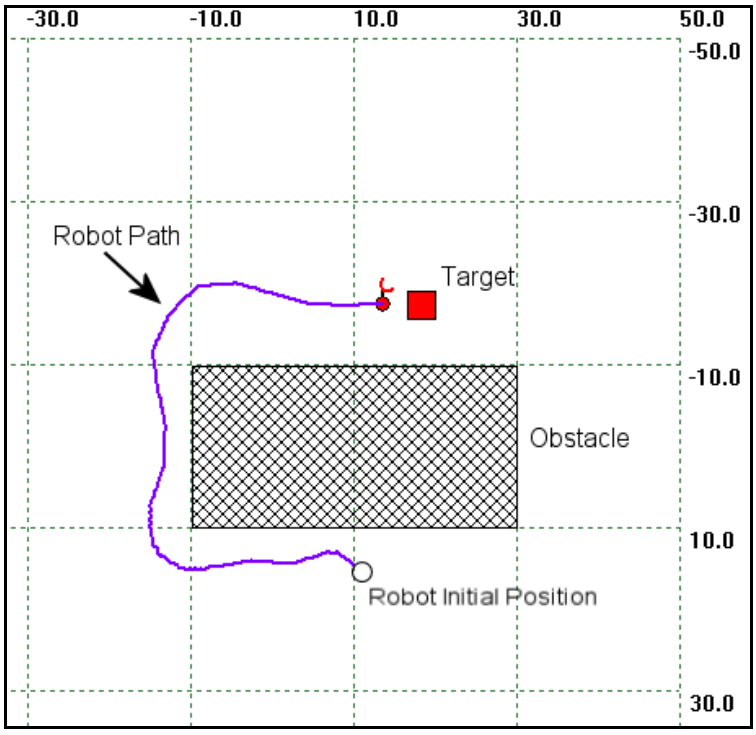


**Figure 47, Final position of obstacle avoidance from rectangular shaped, convex obstacle. Robot avoided obstacle successfully by following shown path.**

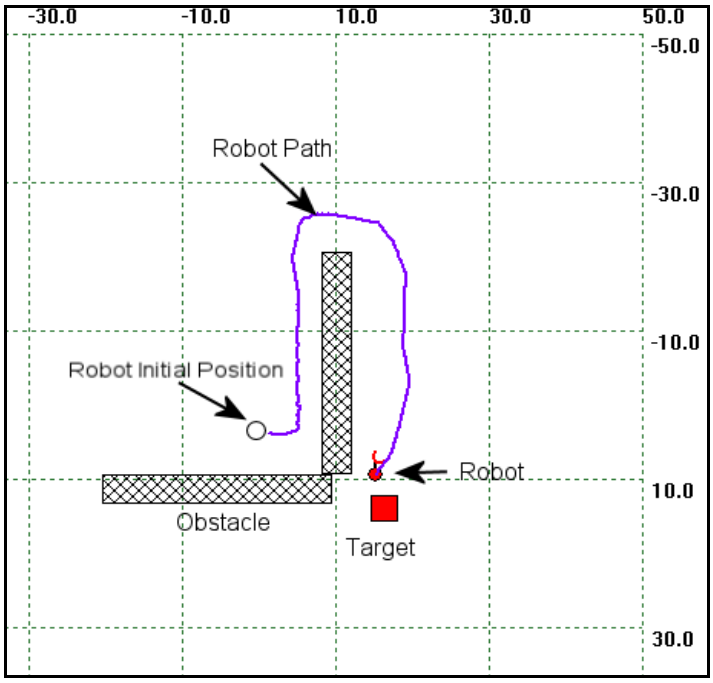


**Figure 48, Final position of obstacle avoidance from a partially concave obstacle. Robot avoided obstacle successfully by following shown path.**

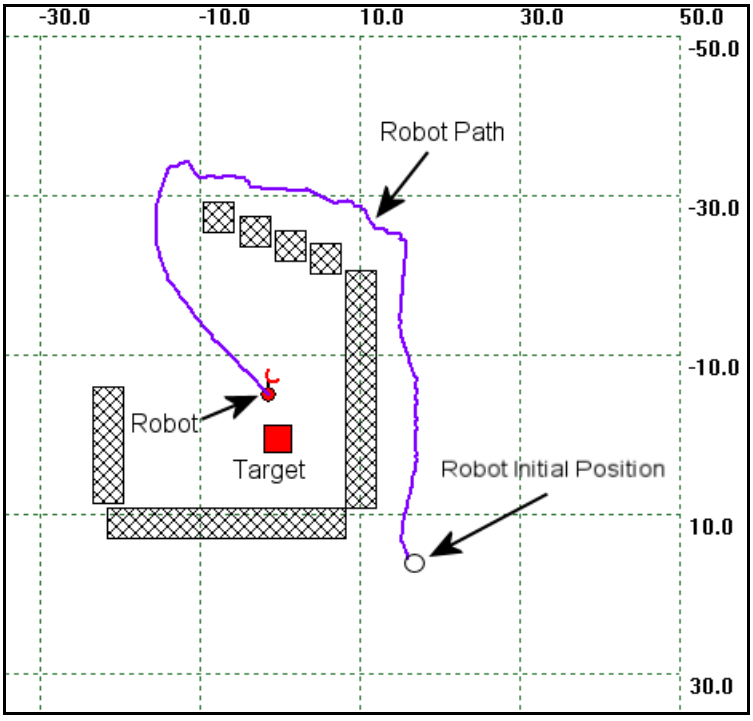


**Figure 49, Final position of obstacle avoidance from a concave obstacle. Robot avoided obstacle successfully by following shown path.**

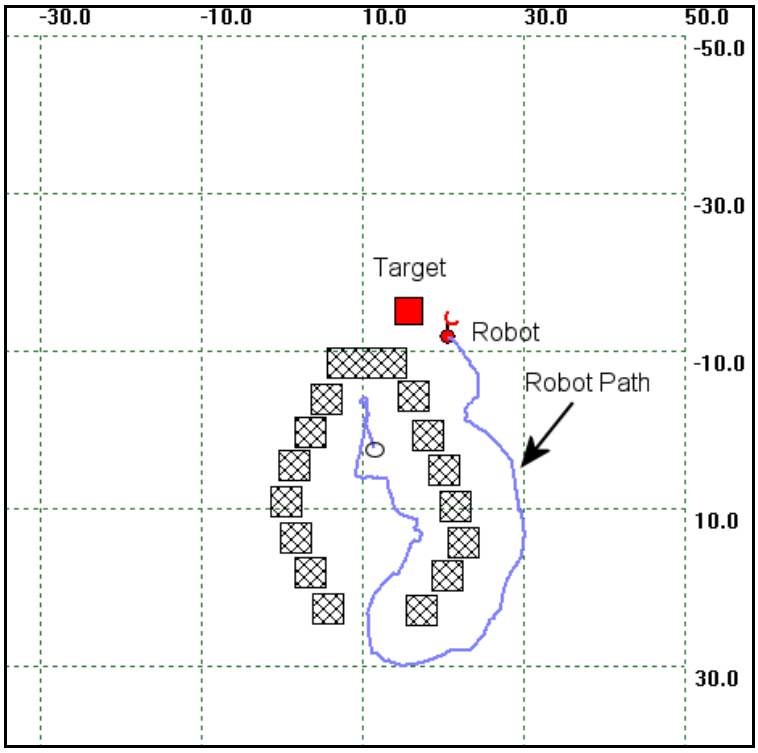


**Figure 50, Final position of obstacle avoidance from a strongly concave obstacle. Robot avoided obstacle successfully by following shown path.**

### 6.1.2 Robot Separation Behavior Analysis

Robot separation behavior can be thought as obstacle avoidance behavior because robot tries to avoid collision with other robots.

Figure 51 gives the initial positions of the robots and sources. A robot tries to go to a target source having the same color, i.e. robot 1 will go to source1 whereas robot 2 will go to source2. In Figure 52, it is clear that robot separation is achieved. In the separation region where robots detected existence of each other a repulsive robot separation force is exerted between each other and robots pass at clear distance from each other

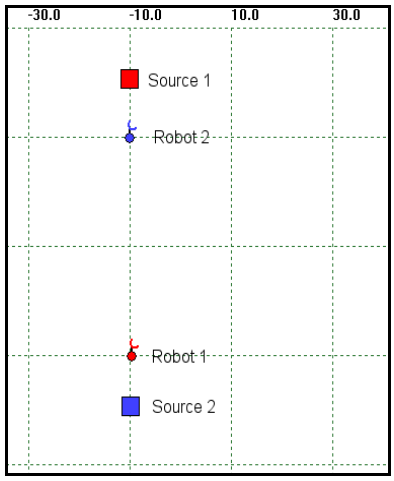


**Figure 51, Initial position of robots and sources**

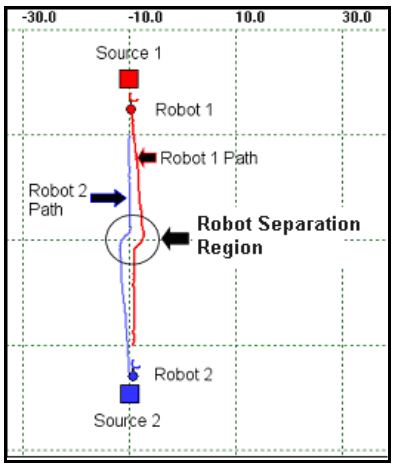


**Figure 52, Robots having initial positions shown in Figure 51 avoided collision each other in robot separation region successfully.**

### 6.1.3 Heuristic Wander Behavior Analysis

Heuristic wander behavior is used in conjunction with adaptive wander behavior. In Figure 53, Figure 54, Figure 55, Figure 56, and Figure 57 paths (blue curves) generated by heuristic wander behavior are shown for time intervals 0-100, 0-200, 0-300, 0-400, 0-900 respectively. These plots are given to show the path generation of the heuristic wander behavior. Since this behavior generates the wander direction randomly, robot passes at the same locations within some time intervals which is not high.

As far as any wander behavior is concerned, coverage is an important measure of the wander performance. In Table 7, the coverage versus time table is tabulated. Even if it takes 300 seconds to reach 50% of entire terrain, it takes 900 seconds to cover the entire region because heuristic wander behavior implements a random search strategy and naturally linear coverage with respect to time is not guaranteed. Moreover, it can take very long times to cover the entire region 100 percent based on this kind of strategies.

**Table 7, Coverage performance table of heuristic wander behavior**

| Time, seconds | Coverage, % |
|---|---|
| 100 | 27.55 |
| 200 | 55. 76 |
| 300 | 76.543 |
| 400 | 82.71 |
| 900 | 100.00 |

Despite the coverage results, heuristic wander behavior increases the reactivity in dynamic environment. Heuristic wander behavior helps avoiding unexpected obstacles because it gives some degree of flexibility of movement. By wandering around the environment based on random strategies increases the probability of getting free in some regions filled with complex shaped obstacles. Deterministic wander algorithm has many advantages but they may fail in highly dynamic environments. In this system main wander algorithm is implemented by adaptive wander behavior. Heuristic wander behavior is an auxiliary wander behavior.

Heuristic wander behavior can be used for mapping purposes. In Figure 59, the mapping of environments given in Figure 58 is shown where mapping is achieved with collaboration of two robots. Robots have mapped the entire region except for obstacles. The shape of the obstacles can be distinguished easily from the

mapping, although mapping is well beyond the scope of the heuristic wander behavior.

## 6.2 Coverage

Coverage is one of the most important concepts in any kind of wander algorithms. In this section, the performance of both heuristic and adaptive wander algorithm will be discussed. Measure of coverage is defined as follows

$$C = \frac{c_n}{G_{MxM}}$$

Where, $c_n$ : Total number of grid covered up to now, and $G_{MxM}$ : Total number of grid of entire region.

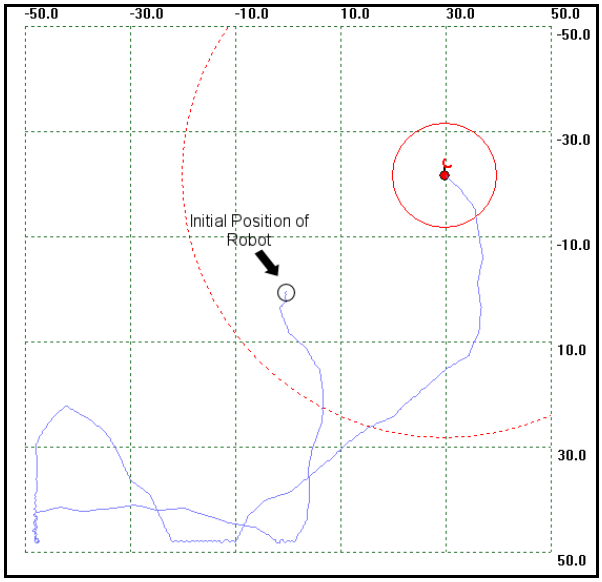


**Figure 53, Robot path generated by heuristic wander behavior after 100 seconds from startup. Coverage is 28 percent.**

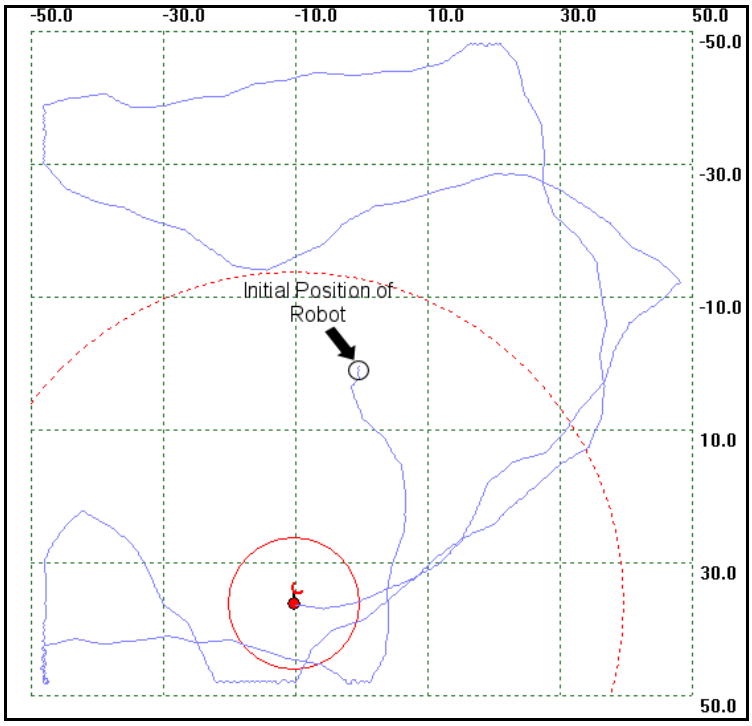


**Figure 54, Robot path after 200 seconds. Coverage is 56 percent.**

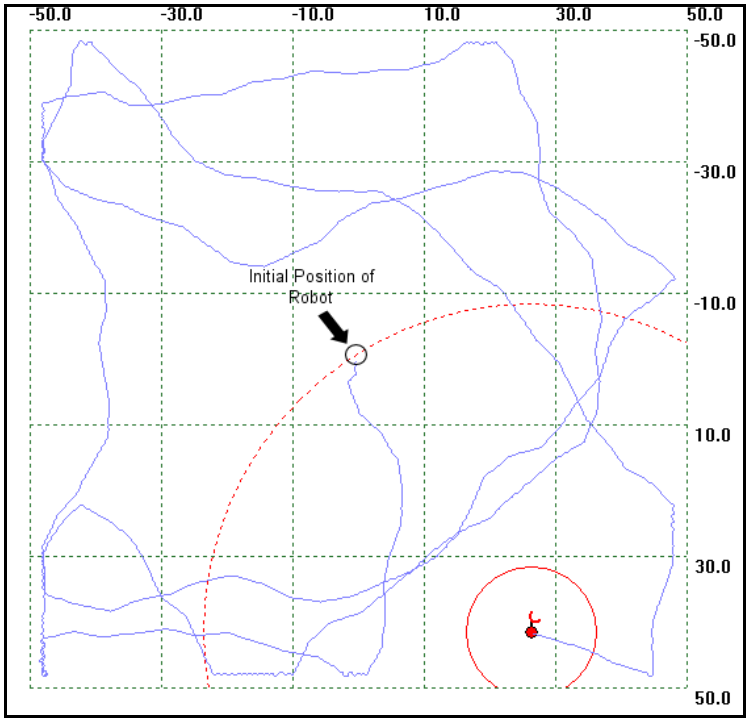


**Figure 55, Robot path after 300 seconds. Coverage is 77 percent.**

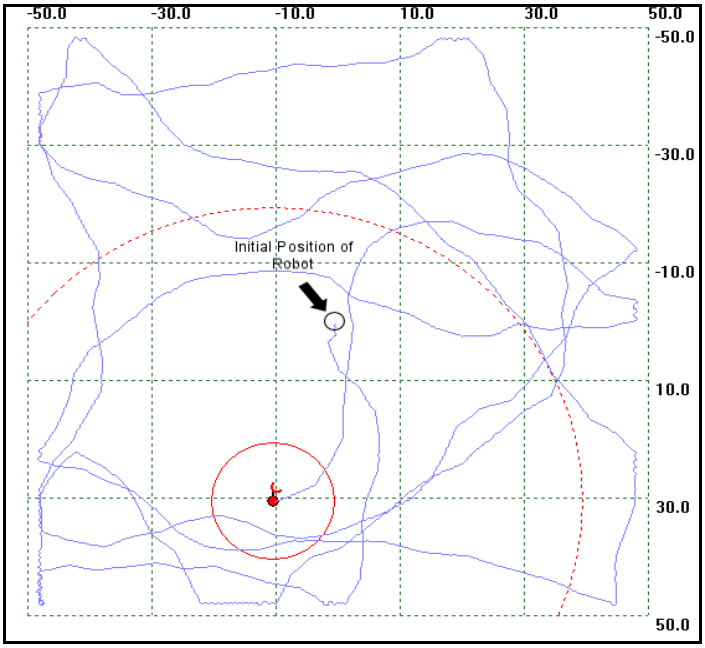


**Figure 56, Robot path after 400 seconds. Coverage is 83 percent.**

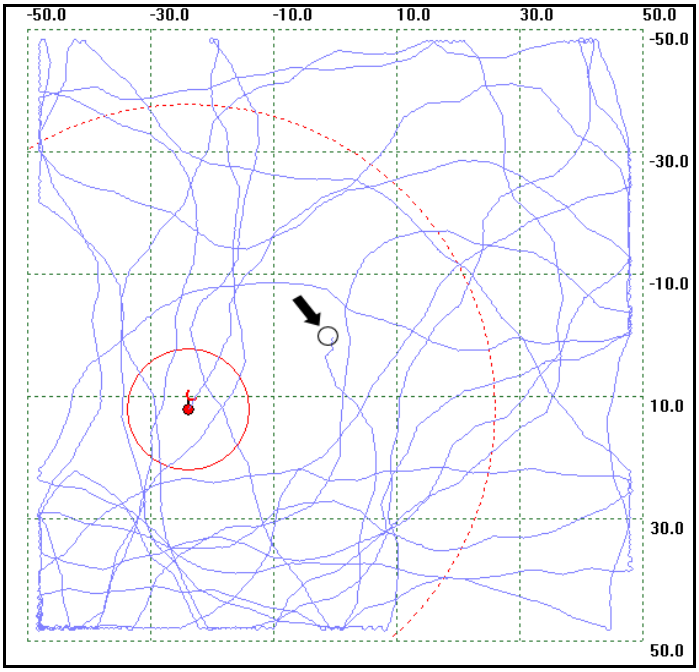


**Figure 57, Robot path after 900 seconds. Robot has covered the entire region.**

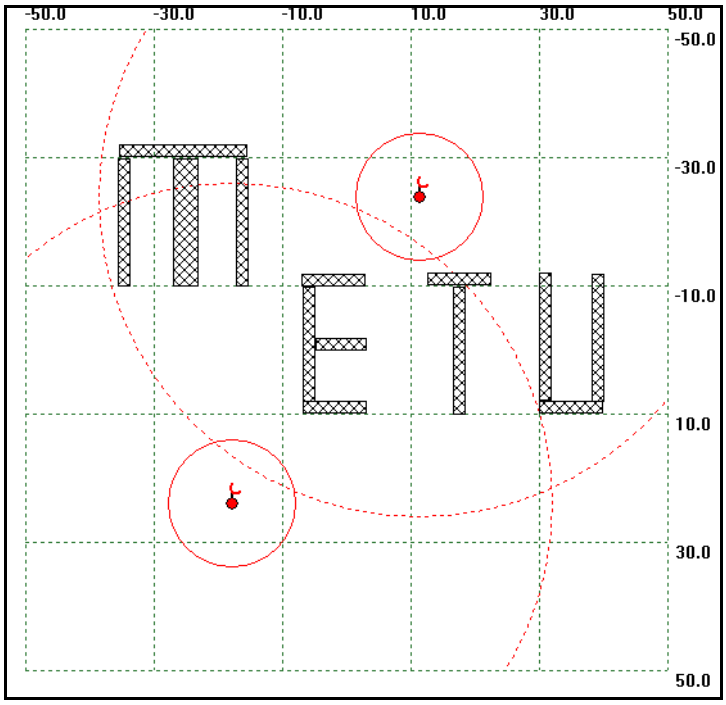


**Figure 58, Initial situation of two robot mapping of a region filled by obstacles.**

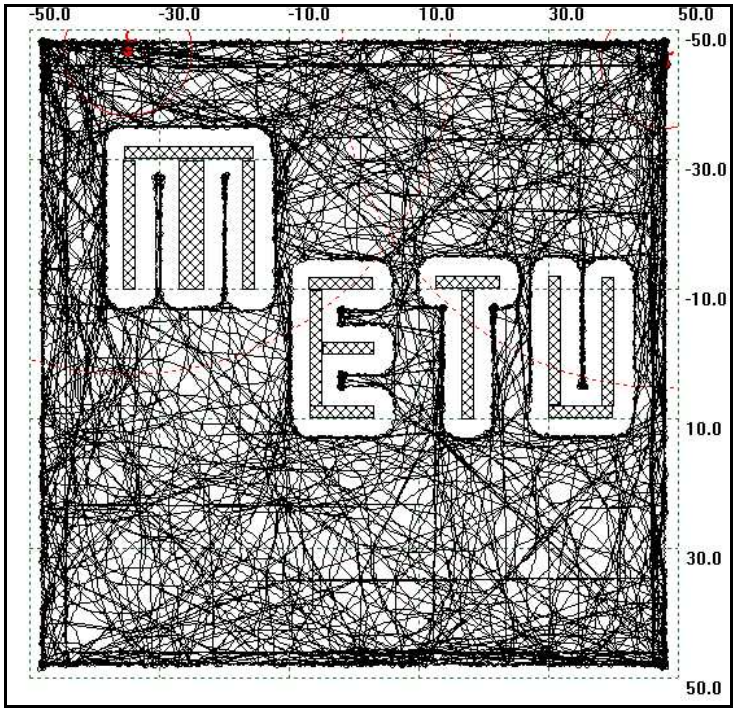


**Figure 59, Final situation of two robots mapping. Black curves are the path of the robots.**

A sample coverage map is shown in Figure 60. In this case $c_n$ is 12, and total number of grids are 81. Coverage is 15%, $C = \frac{12}{81} = 0.15$

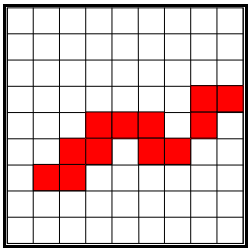

**Figure 60, Coverage map**

In Figure 61, the coverage performance of the heuristic wander behavior is shown. Data is collected for 30 runs in an obstacle and target free environment. Vertical axis corresponds to the required coverage time for which a robot covers 95% of an entire 100x100 $m^2$ region. As shown in the figure, coverage time varies between 4529 and 11243 system time. In real time these correspond to 452.9 and 1124.3 seconds respectively. Environment settings and results are listed in Table 8.

**Table 8, Environment setting and simulation results of heuristic wander behavior**

| **Environment** | **Target and obstacle free** |
|---|---|
| Simulation Frequency, Hz | 10 |
| Robot Primary Sensor Range, m | 10 |
| Robot Maximum Speed, m/sec | 2.5 |
| Region Dimension , $m^2$ | 100x100 |
| Maximum Coverage Time, sec. | 1124.3 |
| Minimum Coverage Time, sec. | 452.9 |
| Mean Coverage Time, sec. | 704.3 |
| 1σ of Coverage Time, sec. | 242.1 |

Results shows that variance of coverage time of heuristic wander behavior is about 0.3 of average coverage time. In worst case, detection time of target will take 1124 seconds. Mean time is 704.3 seconds.

**Table 9, Environment setting and simulation results of adaptive wander behavior**

| **Environment** | **Target and obstacle free** |
|---|---|
| Simulation Frequency, Hz | 10 |
| Coverage Map Timeout Time, $TO_{cov}$ , sec. | 200 |
| Region Dimension , $m^2$ | 100x100 |
| Maximum Coverage Time, sec. | 818.2 |
| Minimum Coverage Time, sec. | 357.0 |
| Mean Coverage Time, sec. | 500.6 |
| 1σ of Coverage Time, sec. | 114.3 |

In Figure 61, the coverage performance of the adaptive wander behavior is shown. In Table 9, the performance of the adaptive wander behavior is found to be better than that of the heuristic wander behavior, about 50%. The mean coverage time is 500.6 seconds which is lower than that of the heuristic wander about 200 seconds. In addition, standard deviation of coverage time is 114.3 seconds which is a half that of the standard deviation of heuristic wander behavior.

In Figure 61 coverage time of heuristic and adaptive wander is given conceptually. These results show that the adaptive wander behavior gives better results than heuristic wander behavior in obstacle free regions. It is clear that probability of detecting a target using adaptive wander behavior is higher than that of heuristic wander behavior. This shows that design and implementation of adaptive wander behavior is successful.

The performance of adaptive wander behavior is determined by the coverage map timeout time, $TO_{cov}$ which as mentioned earlier determines the required memory space. Figure 62 gives, the time required for 95% entire region coverage versus coverage map timeout time is shown. As $TO_{cov}$ is increased then required coverage time decreases. This is logical since robot has more data about covered & uncovered cells to evaluate next wander point and it covers entire region in small amount of time since robots try to wander to the locations previously uncovered. After some timeout value (500sec) there is no effect of increasing $TO_{cov}$ value.

This is because of the saturation of coverage map, i.e. robot covers the entire region before first timeout occurs in coverage map.

High timeout value of the coverage map requires large memory spaces to be allocated. In Figure 63, memory allocation needed for 95% coverage is shown. When changing the timeout in an increasing manner, beyond a certain value the memory demand does not increase much this is because after that value of timeout, the robot has already covered the entire region. Thus it is not needed to allocate unnecessary memory space.

The selection of $TO_{\text{cov}}$ is important here. The selection criterion should be based on the comparison between memory requirements for 95% coverage and the steady state coverage. Steady state coverage map memory usage is the amount of memory needed at least to contain all the data within time interval, $TO_{\text{cov}}$ . If for the same coverage timeout value these memory requirements differ highly then choosing the timeout value under consideration is not meaningful.

In Figure 63, the steady state memory allocation needed for different timeout values is depicted. For $TO_{\text{cov}}$ = 500 seconds, 15 Kb memory should be allocated to satisfy the above timeout value. But in Figure 63, we see that for 95% coverage 12 Kb memory is needed, 3Kb is wasted up. Above a certain value of the timeout value, steady state memory requirements exceed the memory requirements for 95% coverage value. So very high $TO_{\text{cov}}$ is not useful.

In Figure 63, it is seen that memory requirements for $TO_{\text{cov}}$ up to the value of 400 seconds is almost the same. But for $TO_{\text{cov}}$ greater than 400 seconds, the difference between steady state coverage and 95% coverage memory requirements increases dramatically. Therefore it is logical to choose $TO_{\text{cov}}$ as 400 seconds.

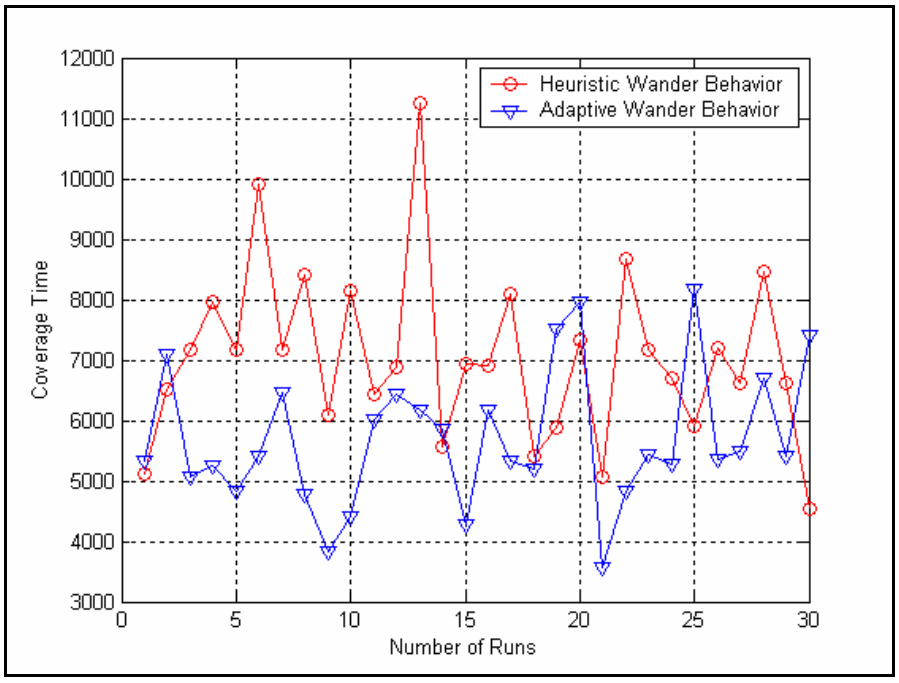


**Figure 61, Coverage time of both heuristic and adaptive wander behavior for various runs.**

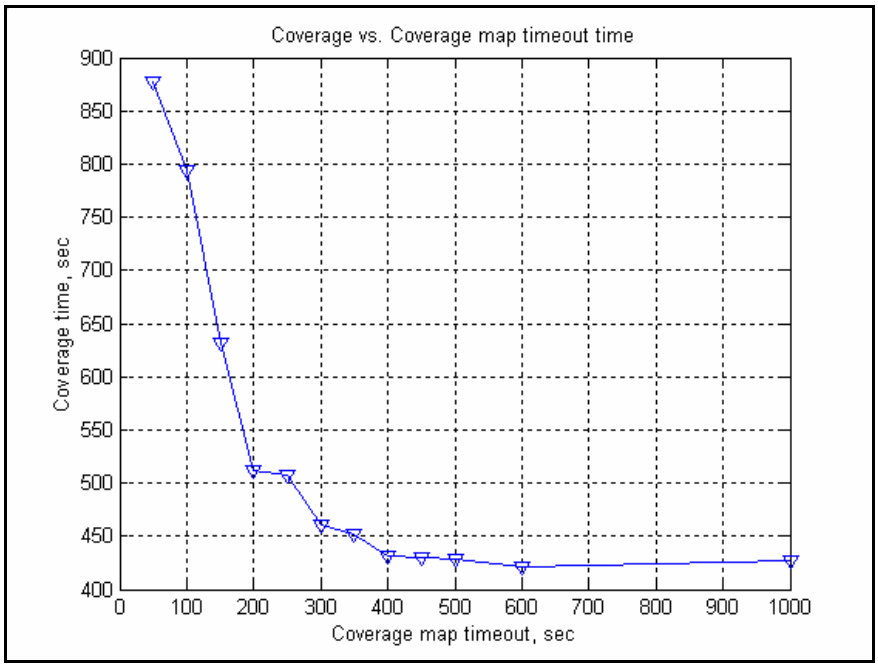


**Figure 62, 95% Coverage for different coverage map timeouts, $TO_{\text{cov}}$. There is an inverse proportionality between coverage time and $TO_{\text{cov}}$.**

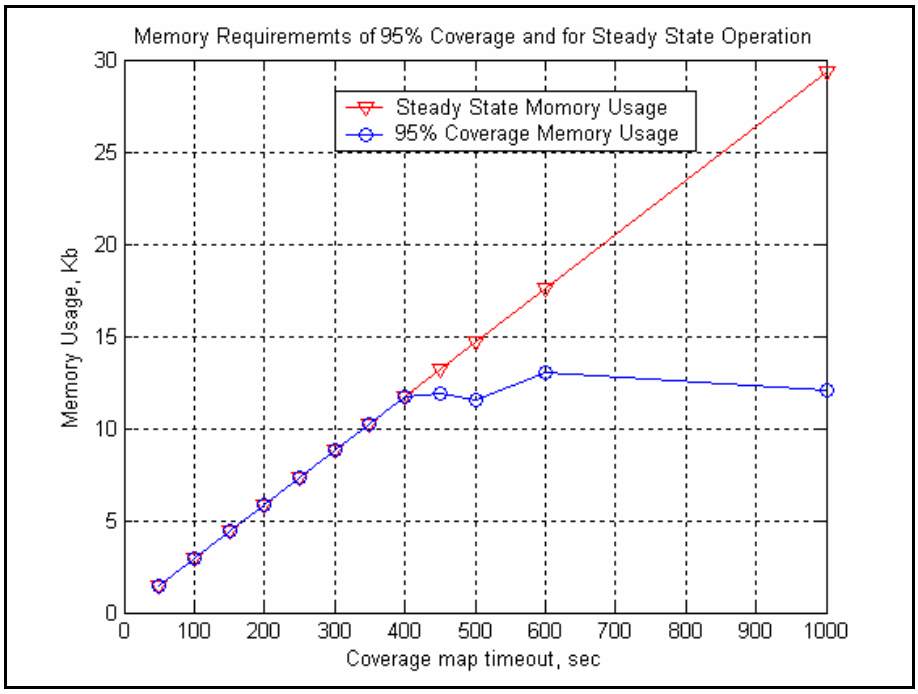


**Figure 63, Memory usage at steady state and memory usage for 95% coverage. After $TO_{cov}$ = 400 seconds, difference between memory usage increases considerably. Appropriate $TO_{cov}$ value should be selected by considering this observation.**

## 6.3 Target Reaching

Target reaching is an essential component of the proposed cooperative control architecture for robot/sensor network. Target reaching behavior generated a motion motor control input whenever a target is detected. In this section, the performance of target reaching behavior in conjunction with system parameters will be discussed.

In Figure 64 and Figure 65, the target reaching in obstacle dense region is shown. The aim of the robot is to reach the target depicted as red square at the same time avoiding collision with many obstacles. In these cases both target reaching and obstacle avoidance behaviors are active and the coordination between these behaviors is done as described in the subsection state evaluation behavior 3.2.1.1.

The Environment in Figure 64 is filled by long and small rectangular obstacles. Avoiding these kind of obstacles are easy and the robot is found to reach the target easily without any collision. The environment in Figure 65 is filled by

large and small rectangular obstacles but depending on the robot direction to the target, these obstacles form convex and concave shaped general obstacles. Avoiding this kind of obstacles is relatively difficult using behavior based approach as described in subsection **Error! Reference source not found.**. Because in behavior based system reactivity is an important design constraints. Modules should be as simple, and reactive as possible. Avoidance from complex shaped obstacles requires careful design of behavior without degrading the simplicity of the system. Target reaching is successful as shown in the figure.

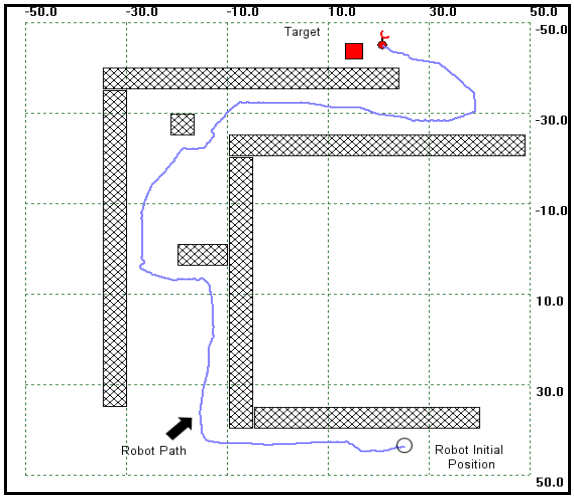


**Figure 64, Target reaching in an environment filled by long and small rectangular obstacles.**

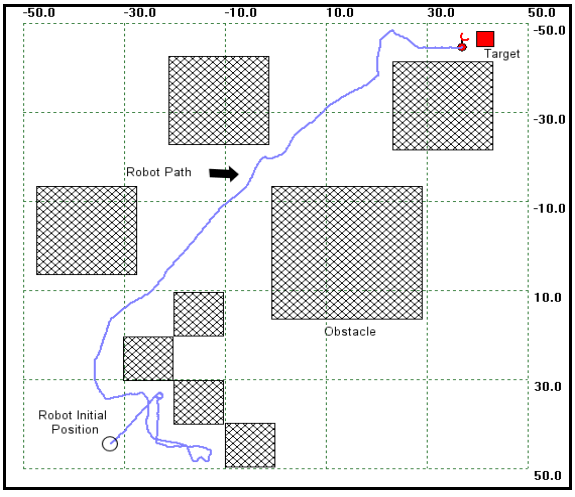


**Figure 65, Target reaching in an environment filled by dense concave and convex shaped obstacles.**

In Figure 66 gives 8 targets distributed around the environment. The corresponding average target reaching time versus coverage map timeout, $TO_{\text{cov}}$ is in Figure 67. As $TO_{\text{cov}}$ increases, the average target reaching time is seen to decrease. This result is compatible with results in coverage section. It is seen that as we increase the map timeout value, then coverage time decreases. To reach all of targets, robot should cover the entire terrain. Therefore, the coverage time and the target reaching time are related with each other.

In Figure 68, target reaching time with respect to number of randomly deployed targets is shown. Targets are deployed around the region uniformly using uniform random number generation. The minimum distance between the sources is adjusted to 10m. The average target reaching time increases almost linearly after 20 targets. Non-linearity up to 20 targets can be explained based on target density and the nature of wander behaviors. In case of low target density (less than 20 targets for this situation), robot non-linear effects of wander behavior becomes important, because target detection rate is low for low target densities. To detect all targets, the robot speeds much time for wandering, on the other hand for high target densities, robot finds the targets without wandering much thus the random nature of the wander behaviors becomes relatively less effective.

The average target reaching time with respect to randomly deployed robots is shown in Figure 69 where the environment is filled with randomly deployed 25 targets. From 0 to 10 robots average target reaching time differs substantially, but after this point, the rate of decrease of the average target reaching time does not decrease much. This is because of two reasons:

- As the number of deployed robots increases then target reaching time probability increases until a certain value beyond which probability does not change much. So rate of decrease lessens.

- Reaching to the target takes some time because the robot should be very close to be accepted as having reached it. But when the number of robot deployment tends to infinity, zero target reaching is found because everywhere, the terrain will be filled by robots.

For above 25 randomly distributed targets, the energy consumption of robots with respect to number of deployed robots is shown in Figure 70. It is interesting that even if the target reaching time is high, the energy consumption of 1 robot is the lowest, i.e. we can deploy 1 robot in the environment instead of many robots if we are aiming to decrease the energy consumption of the robots. Depending on the aim of system, time or energy consumption can be optimized. Average target reaching time versus energy plot for different number of robots is shown in Figure 71. Each triangle in the plot corresponds to a triplet (energy/time/number of deployed robots). If it is desired to operate at 670, 150 average energy consumption and average target reaching time then 10 robots should be deployed in the region. These plots are important because they give a statistical analysis of important constraints.

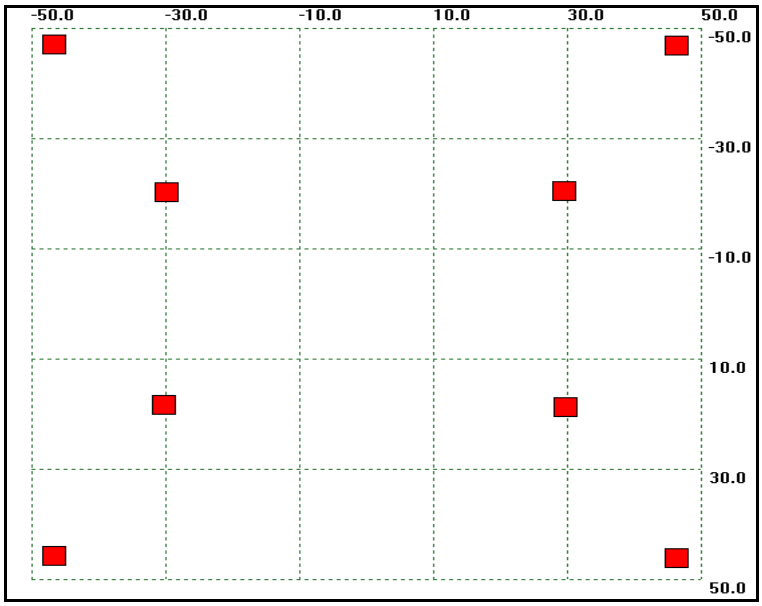


**Figure 66, Target distribution in 100x100 $m^2$ region**

.

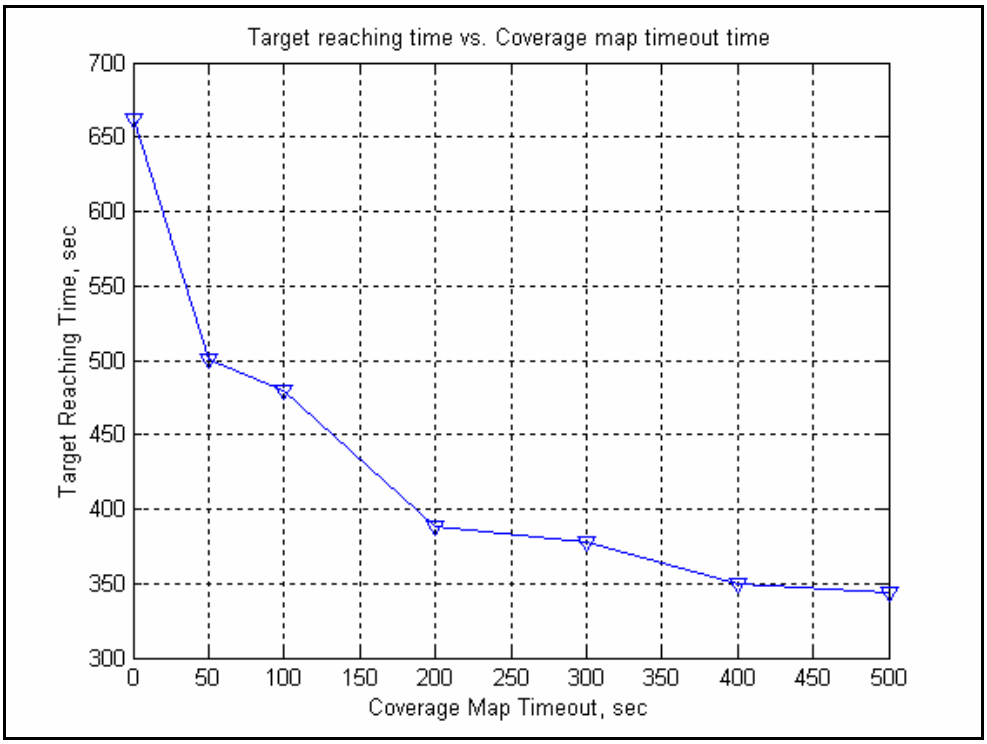


**Figure 67, Target reaching time with respect to $TO_{\text{cov}}$ for regularly deployed 8 targets shown in Figure 66.**

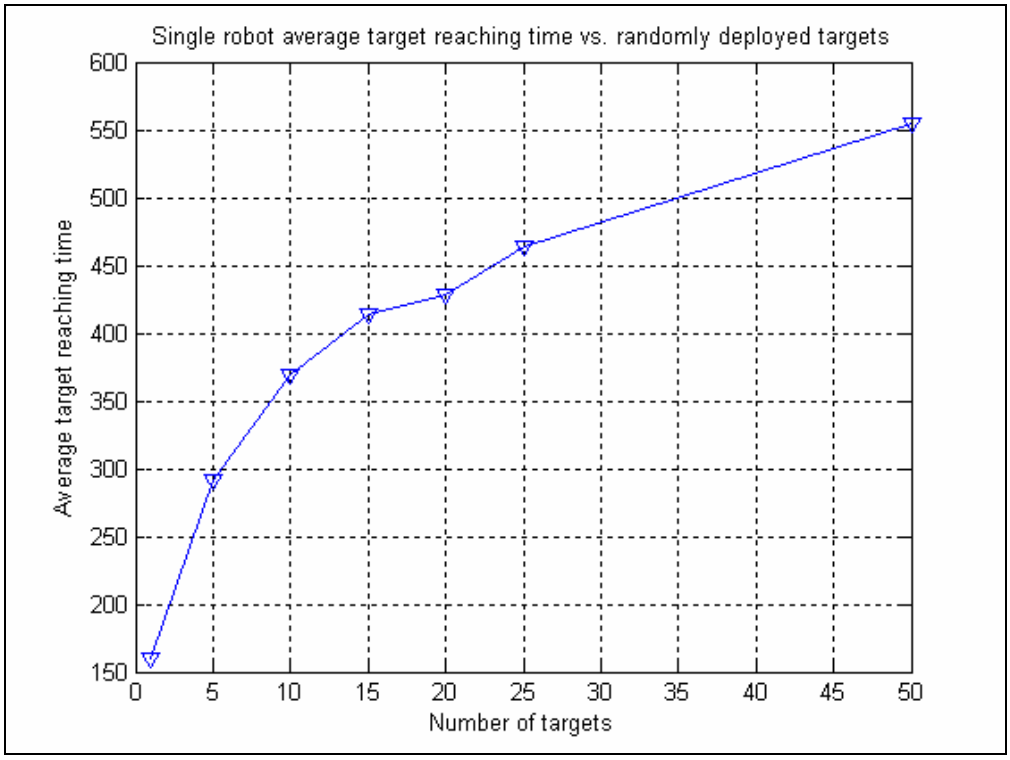

**Figure 68, Average target reaching time of single robot with respect to different number of randomly deployed targets.**

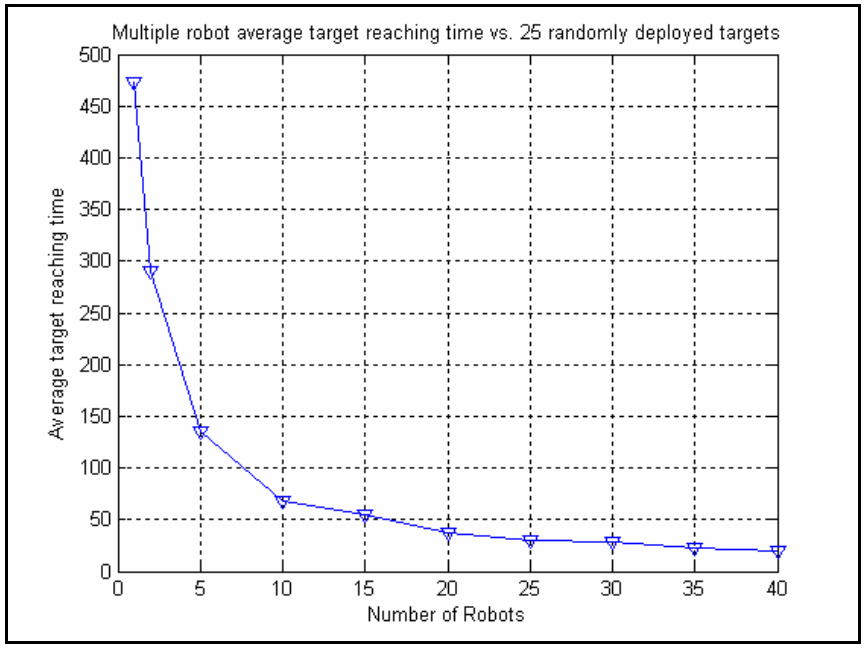


**Figure 69, Average target reaching time of different number of randomly deployed robots to randomly deployed 25 targets.**

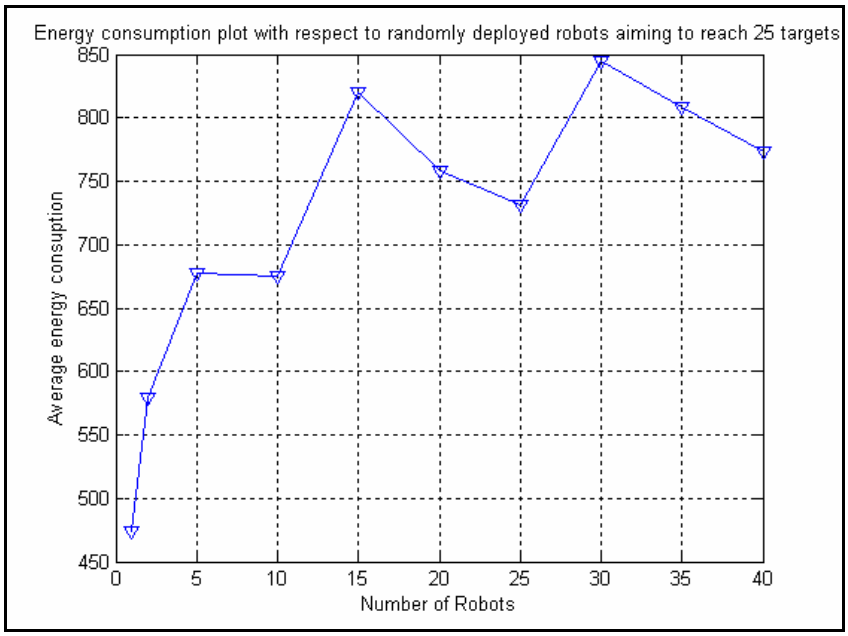


**Figure 70, Average energy consumption of different number robots reaching to randomly deployed 25 targets.**

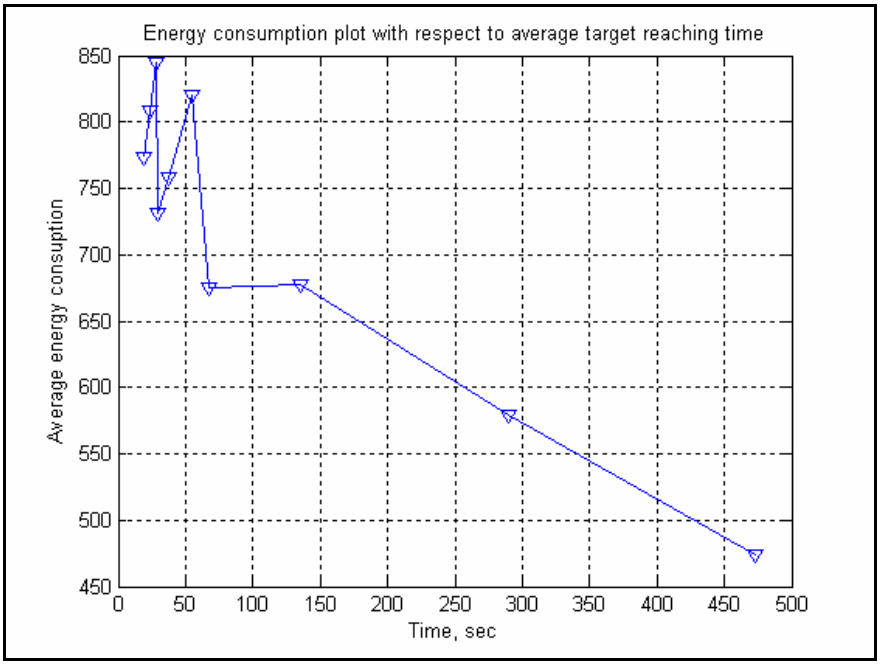


**Figure 71, Average energy consumption of different number of robots with respect to time. Each triangle corresponds to an energy-time-number of robots triplet.**

So far, targets were deployed uniformly or very regularly in the environments. One of the basic assumptions of this system is the dynamic environment assumption. In the above cases dynamism is all around the environment. In some cases, probability of existence of targets may be high in some regions. System dynamism should include this kind of situations. We will analyze the target reaching time when target is not deployed uniformly around the environment. For some regions, density of target is high as compared to the other regions.

In Figure 72, average target reaching time of single robot with respect coverage map timeout, $TO_{\text{cov}}$ is given for 5 targets which are deployed randomly in right quarter of the entire region. The plot is given with respect to $TO_{\text{cov}}$ because this parameter directly determines the reactivity to target deployment region. As shown in this figure, for $TO_{\text{cov}}$ 300 and 400 seconds target reaching time is the minimum. As it increases beyond 400 seconds, average target reaching time gets

increased. This is because, once a robot has visited a region it takes a long time to revisit it because of high timeout value. Low $TO_{cov}$ values give the worst results because it forgets the past rapidly. For the system conditions listed in Table 8, $TO_{cov}$ can be selected as $270 \leq TO_{cov} \leq 400$. This is compatible with target reaching time and memory requirements analysis.

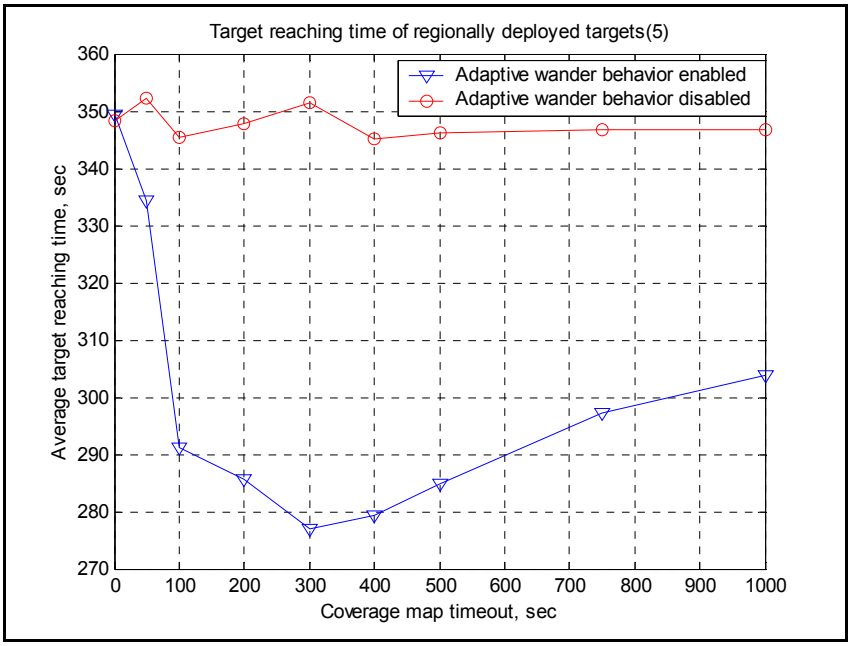


**Figure 72, Target reaching time of a single robot with respect to $TO_{cov}$. If Adaptive wander is enabled target reaching time decreases considerably in case of regionally deployed targets. There is a local minima optimizing dynamic response to regionally target deployments.**

## 6.4 Effect of Communication on Target Reaching

This section is devoted to the discussion on communication range effect on target reaching time. Communication is a key aspect in task allocation. Usage of communication enables an efficient multi-robot task solution. Despite its advantages, there are many issues to be considered as drawbacks, for example the usage of communication brings in an extra cost. There should be a measure of range in order to optimize the communication cost. In some cases which will be emphasized in below, increase of communication dose not increase the system performance considerably.

Since multi-robot systems are highly non-linear systems, it is difficult to model the entire system. There are many parameters affecting the system and also the coupling effects of these parameters. Moreover, since behaviors in the architecture contain randomness, it is difficult to develop appropriate models and we aim to identify the model using an appropriate designed neural network using simulated data.

There are parameters affecting communication performance and cost, these are:

- Number of robots
- Number of targets
- Communication range of robots
- Primary sensorial range of robots
- Nature of task

Effective communication range can be found by making experiments for different values of the above parameters. Communication range performance is evaluated using total target reaching time (TAR), that is if 20 tasks are deployed, TAR is taken as the total time elapsed to reach zero task in the environment..

Primary sensorial range is kept constant as 10 meters. For a constant number of robots, targets numbers are changed. For each target number, TAR time is obtained for communication ranges 0, 25, 50, 75, 125, 150 meters. For each range, 100 simulations are performed. Communication range 0 means no communication. For 150 meters communication range, a robot can communicate with any robot in the terrain. The upper limit is determined by the terrain size. In this case the terrain syurface is 100x100 meter$^2$.

***Analysis of the Effect of Communication Range over Target Reaching Procedure***

1. Repeated step 2 for number of robots 4 and 8. Deploy robots in the center.

2. Repeat step 3 for number of targets 5, 10, 15, 25, 30, 35. Distribute the target randomly around the environment. Each target should not close to each other less than robots sensorial range.
3. Repeat step 4 for communication ranges for 0, 25, 50, 75, 100, 125, and 150.
4. Simulate environment 100 times and measure the TAR time for each simulation. Take the average of TAR times. This averaged time is saved. Each simulation is continued up to all of the tasks are executed.

---

Analysis of the effect of the communication range over target reaching time procedure is listed above. Targets are correlated tasks formed by two independent sub-tasks. If a robot determines the existence of a correlated task, it executes a suitable sub-task compatible with its primary sensors. Communication request is generated for other sub-task, i.e. a robot having capability of solving this subtask is invited to handle it. If 4 robots are to be deployed, robot team is divided into equal two parts since there exist only 2 sub-tasks. Robots in the same team have the same primary sensor.

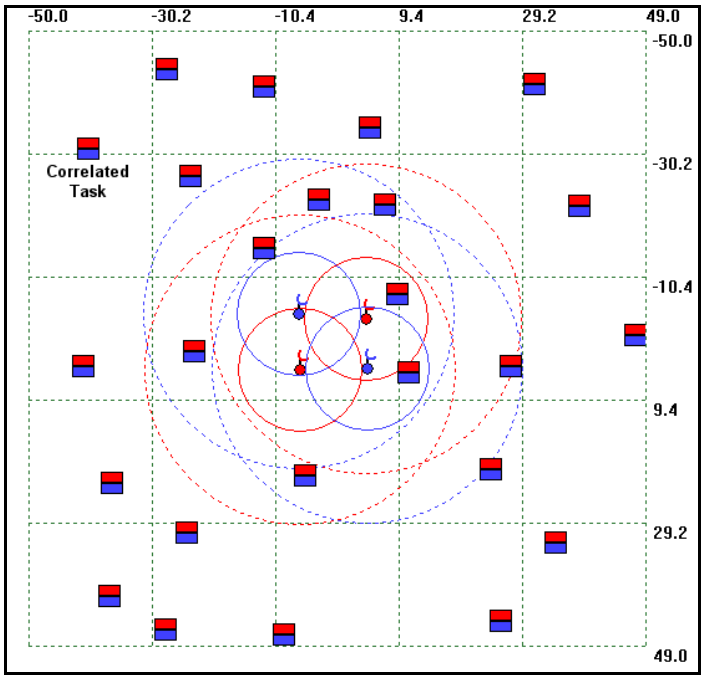


**Figure 73 A screen shot from simulation environment for 6 robots and 25 correlated tasks. Communication range is 25 for each robot.**

In Figure 73, initial screen shot of the simulation for 4 robots, 25 targets and 25m communication range is given. The range of primary sensors is 10m. Table 10 covers the information about the environment and simulation.

**Table 10, Simulation and environment properties.**

| **Environment** | **Obstacle free** |
|---|---|
| Simulation Frequency, Hz | 10 |
| Robot Primary Sensor Range, m | 10 |
| Robot Maximum Speed, m/sec | 2.5 |
| Region Dimension , $m^2$ | 100x100 |

In Figure 74, target reaching time (TRT) for 4 robots is shown. As the density of task increases, increase in communication range does not increase the TRT. Up to 10 tasks increase of communication range decreases the TRT. But for tasks more than 10 and less than 30, there is a limit in their effect to TRT changes: for these task numbers, there is an optimum point around 50-75 meters communication range. As

the density of target increases beyond 30 then communication range increase does not change the performance. In this case (4 robots) for 30 and 35 target numbers, it is unnecessary to use communication. Since target density is so high, robots do not need to communicate with each other. But for lowest task density, increase in communication range directly affects the target reaching time. For 35 targets, situation gets worse when communication range is increased.

The above discussion is also valid for simulation results of 8 robots shown in Figure 75. But since the robot density is high, increase in the communication range more rapidly saturates the TRT. After 15 tasks, communication range increase does not result in a considerable TRT performance increase. Results clearly show that, there is a limit to the communication range for creating an increase in TRT performance. For low task densities, communication covering wide range of regions is more preferable.

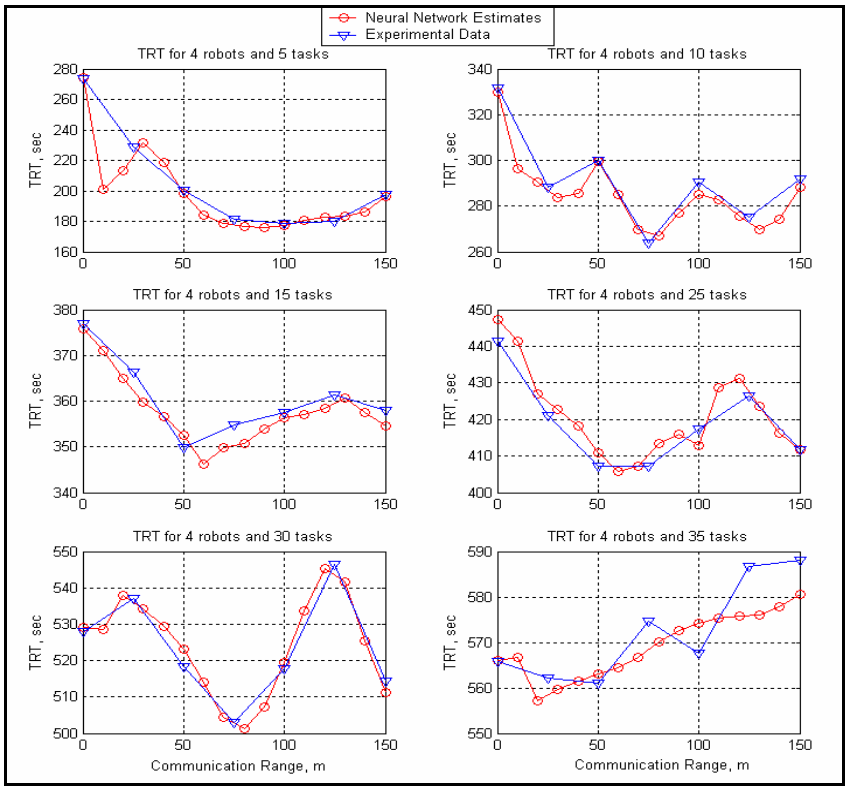


**Figure 74, Target reaching time (TRT) for 4 robots and different number of task and communication ranges. Neural network estimates are also shown as red lines with more resolution.**

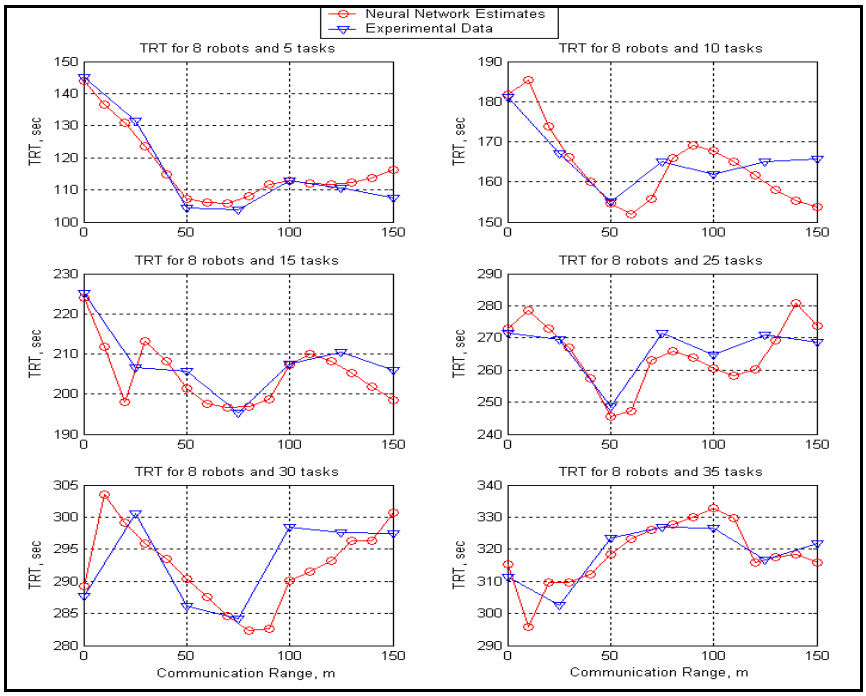


**Figure 75, Target reaching time (TRT) for8 robots and different number of task and communication ranges.**

## 6.4.1 Neural Network Implementation

A 4 layered neural network is designed using MATLAB R13 to estimate the effect of communication range for different conditions. Inputs to the neural network are number of robots, number of tasks and communication range. The output is estimated target reaching time.

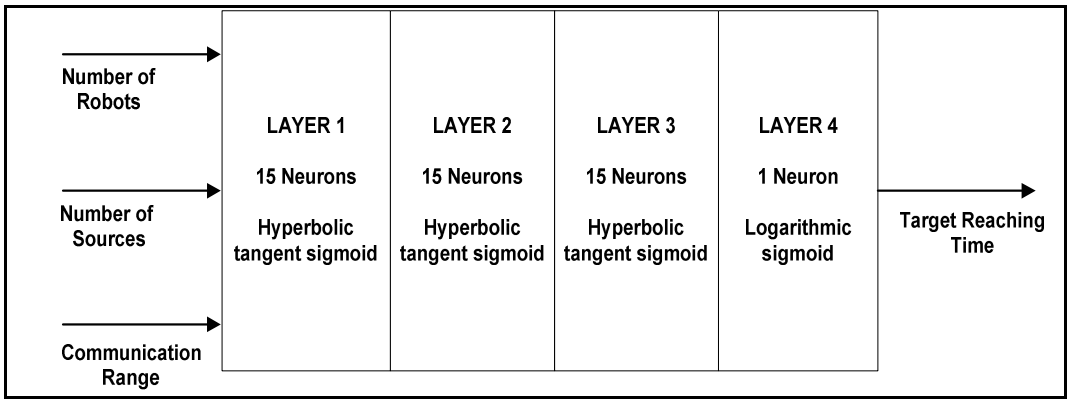


**Figure 76, Implemented neural network architecture**

In Figure 76, neural network structure is shown. There are two different non-linearities are used: hyperbolic tangent sigmoid and logarithmic sigmoid functions. Neural network is trained using data collected from simulations of 4 and 8 robots. Neural network is trained with simulation data having a mean square error up to 0.0005, using Levenberg-Marquardt method. To avoid over training, the network is trained using both training data and test data. In Figure 74, and Figure 75 both experimental data and trained neural network estimates for these data are shown. These plots show that the training of neural network give better results.

To test the performance of neural network, simulation results and neural network outputs are compared in Figure 77. Described procedure is performed for 6 robots. Since neural network is not trained with 6 robots data, this experiment can be used for comparative purposes. Even if the shapes of the curves are similar, the neural network does not give correct experimental results. The reason behind mismatch between results in experimental data, and neural network output, may be due to the lack of simulation data or due to system nature. As mentioned earlier, simulated system is highly stochastic and non-linear. Correct curve shape estimate is a benefit. This information can also be used to decide upon increasing the range of communication.

The 3D plot of Neural network TRT estimates for 2, 4, 6, and 8 robots are shown in Figure 78, Figure 79, Figure 80, and Figure 81 respectively. X and Y axis are corresponding to the number of targets (tasks), and communication range. Since implemented model generated with neural network is highly non-linear, for some input values sharp changes can exist. These plots are given to show fitted model is meaningful because it does not have sharp changes, and compatible with the experimental data. There are small deviations, hills, peaks in plots which are due to the non-linear nature of fitted model.

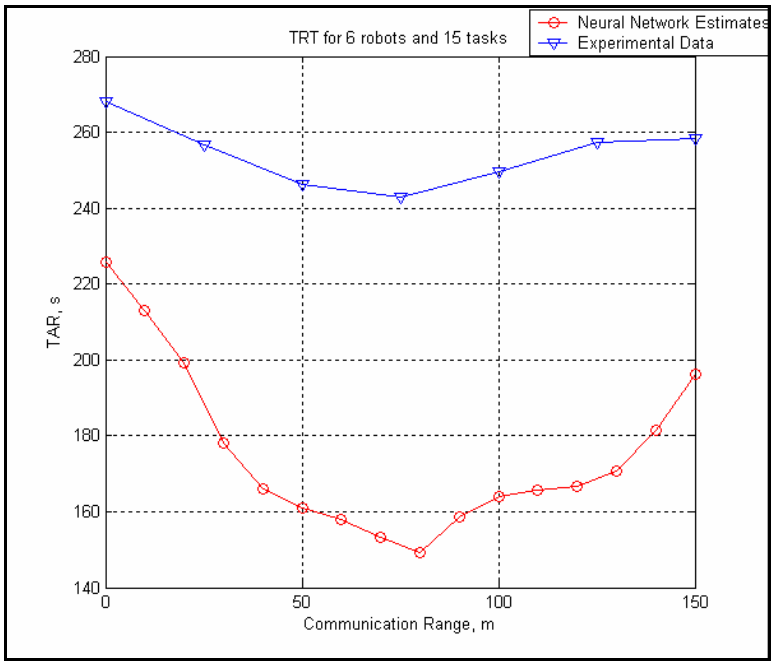


**Figure 77, Experimental data and expected neural network output for 6 robots.**

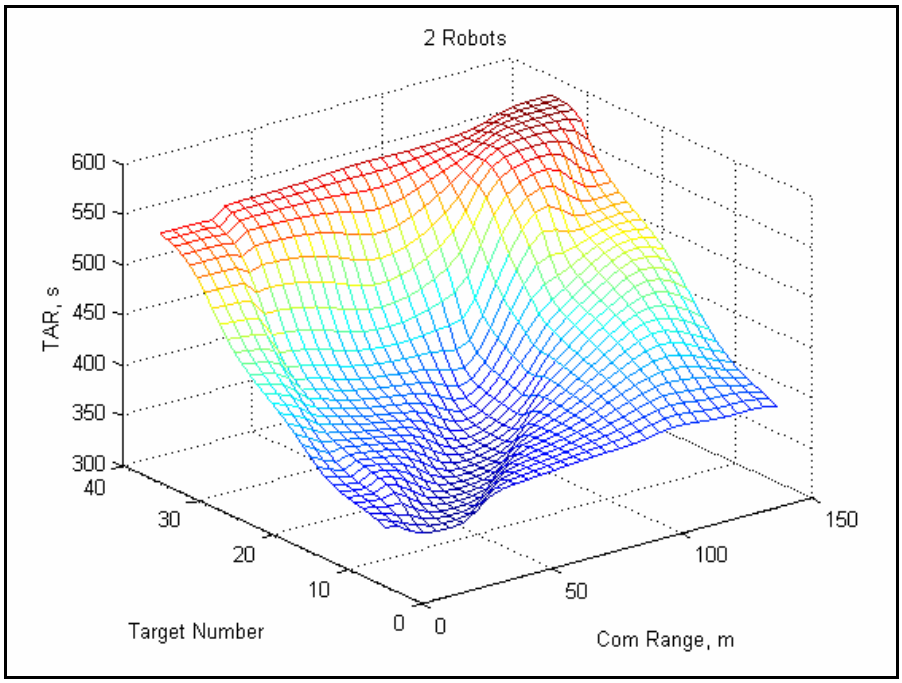


**Figure 78, Neural network TRT estimates for 2 robots for different target numbers and communication ranges**

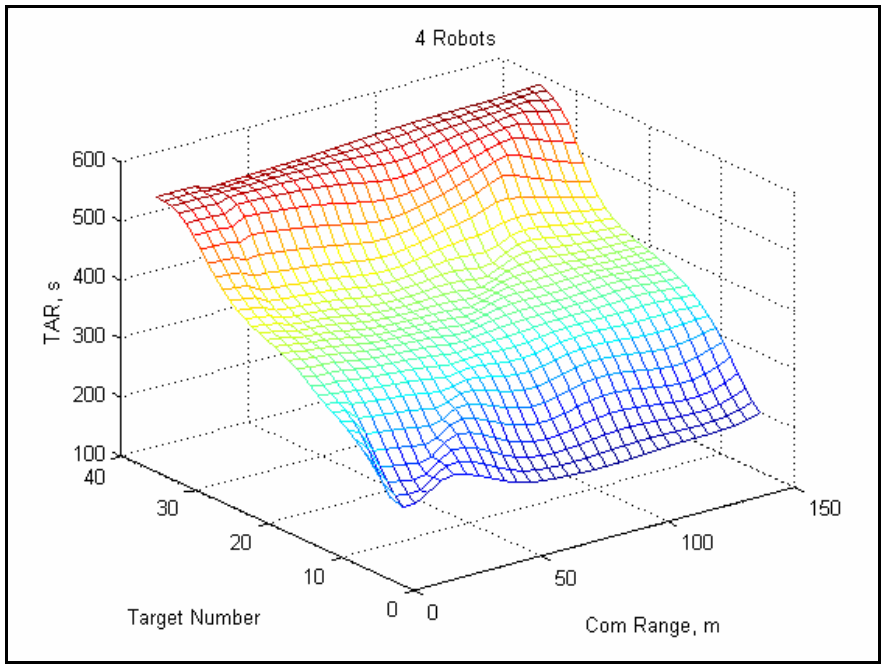


**Figure 79 Neural network TRT estimates for 4 robots for different target numbers and communication ranges**

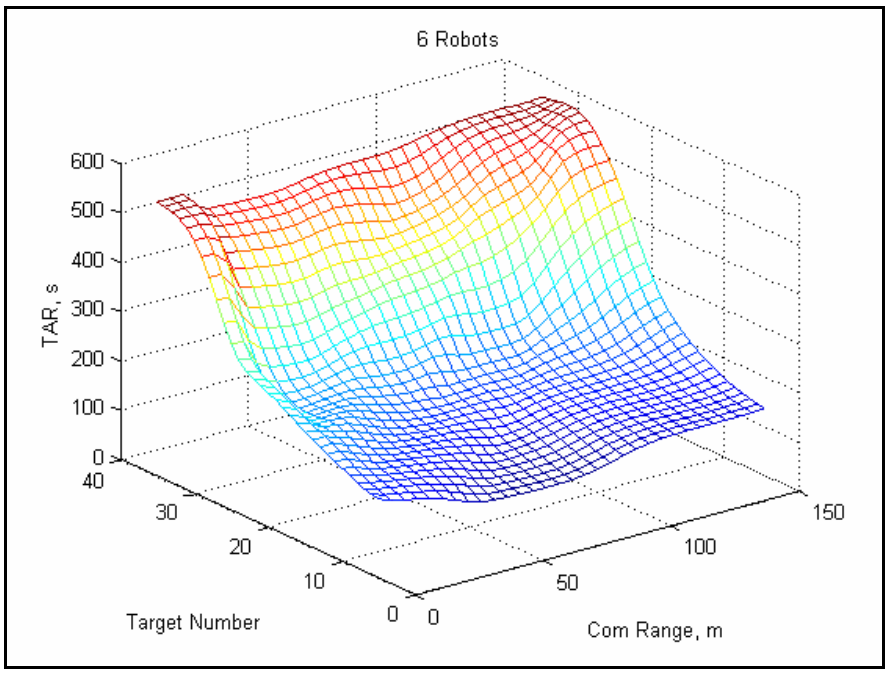


**Figure 80 Neural network TRT estimates for 6 robots for different target numbers and communication ranges**

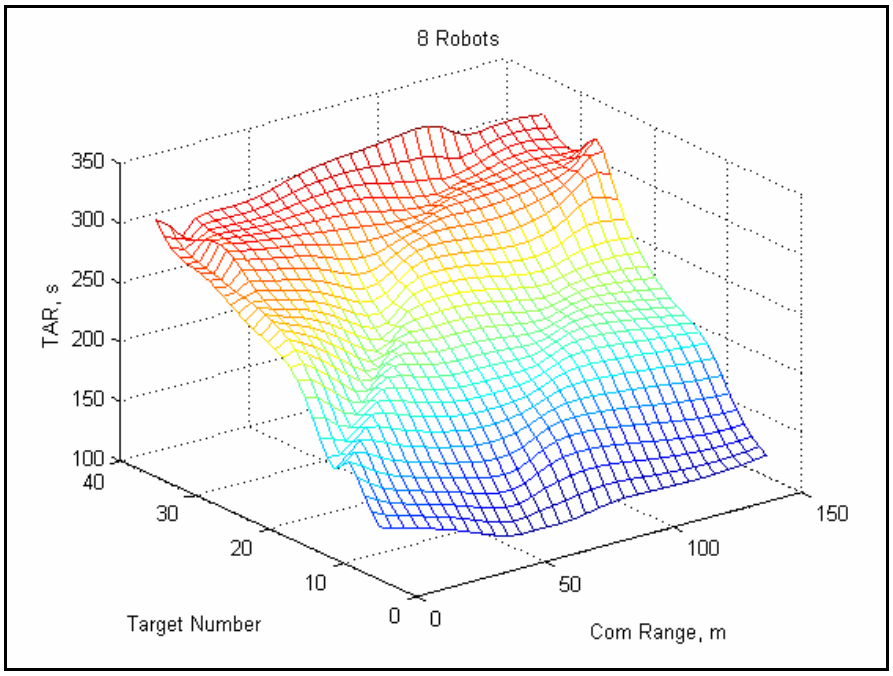


**Figure 81 Neural network TRT estimates for 8 robots for different target numbers and communication ranges**

## 6.5 Task Allocation Performance Evaluation and Fault Tolerance

In this section, the performance of task allocation algorithm (TAA) presented in previous section will be analyzed. Task allocation process enables the robots to solve tasks requiring cooperation of robots. Therefore the performance of task allocation directly affects the performance of robots and task execution.

Task allocation is based on a performance criterion called fitness. Each robot is capable of evaluating its own fitness described in fitness sections. This section will cover the experimental results for

- Fair task allocation in obstacle dense, and obstacle free regions
- Selection of fitness functions
- Fault tolerance analysis

### 6.5.1 Fair Task Allocation

Fair task allocation is a performance criterion regarding selection of fitness functions. Fair task allocation means that robots having no faults in bounded regions should have equal probability for task allocation. If the fitness criterion is not selected appropriately, fitness of some robots may become very high as compared to others. As result, un-fair task allocation can be obtained for some robots. For instance, in case of fair task allocation, if 5 robots are deployed in region then at steady state it is expected that each robot should execute 20 percent of the total executable tasks executed. In some un-fair allocation one robot may executes 40%, and others 15% of the total tasks.

Fairness of task allocation can be measured using the following metric [48]:

$$\rho_i = \begin{cases} |f_i - \mu|, & f_i > \mu \\ 0, & f_i \le \mu \end{cases}$$

where

$\rho_i$ : Fairness of current robot

$f_i$ : Normalized number of task allocated for the robot under consideration

$\mu$ : Average number of normalized task allocation

Since $f_i$ , and $\mu$ are normalized, both of them less than or equal to 1.

Total fairness, $\rho$ , is

$$\rho = \sum_{i=1}^{N} \rho_i \text{ , N is number of robot}$$

The proposed fitness functions are found to eliminate the un-fair task allocation situations based on the above metric. Results are tested both in obstacle free and obstacle dense regions.

#### 6.5.1.1 Task Allocation for 5 Robots, 5 tasks at Obstacle-Free Region

Task allocation performance evaluation is done for the test conditions presented in Table 11. 5 robots are deployed in 100x100 $m^2$ environment to find and to solve 2-robot synchronously correlated tasks, and uncorrelated tasks. As mentioned earlier, the synchronously correlated task requires task allocation process; where as uncorrelated task does not require any task allocation. In this test, 5 robots are deployed on a terrain where three 2-robots synchronously correlated tasks, and two 1-robot uncorrelated tasks exist. As robots execute the tasks, the number of tasks decreases but this is not allowed. Task deployment strategy is based on keeping target density constant in the environment. Whenever a task is finished then a new task is created randomly in the region. In Table 12 fitness parameters of fitness functions are given.

**Table 11, Environment, robots, tasks, and simulation properties**

| | |
|---|---|
| Behavioral status | No behaviors are inhibited |
| Fitness evaluation status | All fitness functions are enabled |
| Environment | Obstacle free |
| Simulation Frequency, Hz | 10 |
| Simulation time, sec | 3600 |
| Region Dimension , $m^2$ | 100x100 |
| Robots | 5 robots, all robots are the same type. |
| Robots fault status | All robots are functioning perfectly |
| Robots' initial position | At the center of region |
| Tasks | 3 2-robots synchronously correlated task, and 2 uncorrelated tasks |
| Task deployment Strategy | Task density is kept constant. Targets are randomly. |
| Target Density, targets/ $m^2$ | 5/(100x100) |
| Robot main sensorial range, m | 10 |
| Communication range, m | 50 |
| Coverage map timeout, sec | 400 |
| Obstacle map timeout, sec | 300 |

**Table 12, Fitness parameters**

| **Fitness Item** | F1 | F2 | $x_{0.5}^{+}/t_{0.5}^{+}$ |
|---|---|---|---|
| Target reaching frequency | 20 | 0 | 0.01 |
| Communication failure frequency | 10 | 0 | 0.01 |
| Obstacle avoidance success frequency | 5 | 0 | 0.2 |
| Obstacle avoidance failure frequency | 10 | 0 | 0.5 |
| Coverage | 5 | 0 | 0.15 |
| Distance | 5 | 0 | 25 |
| Coverage fitness timeout, sec | 80 | | |

In Figure 82, the fitness functions for different robots are shown. In this case, the dominant fitness function is the target reaching frequency. But since all robots are functioning perfectly, error fitness functions, communication failure, and obstacle avoidance failure frequency are not activated. It is shown that robots execute task at 0.02 Hz, i.e. robots' average target reaching time is 50 seconds.

Obstacle avoidance success frequency fitness functions are activated only at the borders of the environment. Its contribution is very small as compared to the other non-zero fitness functions.

Coverage fitness function is highly reactive for the coverage of robot. Since fitness coverage map time out is set to 80 seconds, a target reaching will decrease coverage fitness function since average target execution of a task is 30 seconds. This fitness function is one of the tuning functions for fair task allocation process. Therefore it fluctuates more rapidly. In Figure 82, distance fitness function is shown as zero but this is not the case. Distance fitness function is evaluated relative to task location. It is not meaningful to draw the fitness function with respect to time. Distance function, and coverage function enables fair task allocation operation.

In Figure 83, number of tasks executed by each robot, total number of tasks executed, percentage of task executed by different robots, and sum of fitness functions excluding distance fitness functions are given with respect to time.

Sum of fitness functions excluding distance function is shown. It is clear that number of targets reached by each robot is proportional to the sum of fitness functions. For fair task allocation sum of fitness functions excluding distance fitness should be close enough to each other. Since 5 robots are deployed, it is expected that each robot should reach the 20% of the total targets reached. As shown in Figure 83, each robot reaches nearly 20% of the total number of tasks reached. Fairness of the task allocation is 0.020

Another important parameter is **time of convergence** to 20 percent task allocations. In Figure 83, it is shown that after 750 seconds from the beginning, task allocation percentage reaches almost steady state task allocation percentage.

There are fluctuations over sum of fitness functions. This is because of fluctuations over coverage fitness function. Whenever a task is reached, coverage fitness is decreased otherwise robots coverage fitness increases. Actually, these fluctuations tune the sum of fitness functions for fair task allocation.

Another important aspect is the rate of task allocation/execution. In a normal situation, it should be linear with respect to time. The first figure with dashed lines in Figure 83 shows that task allocation rate is almost constant because total number of target reaches increases linearly with time.

Simulation results show that task allocation is fair enough, and linear for conditions on system and fitness parameters listed in Table 11, and Table 12 respectively.

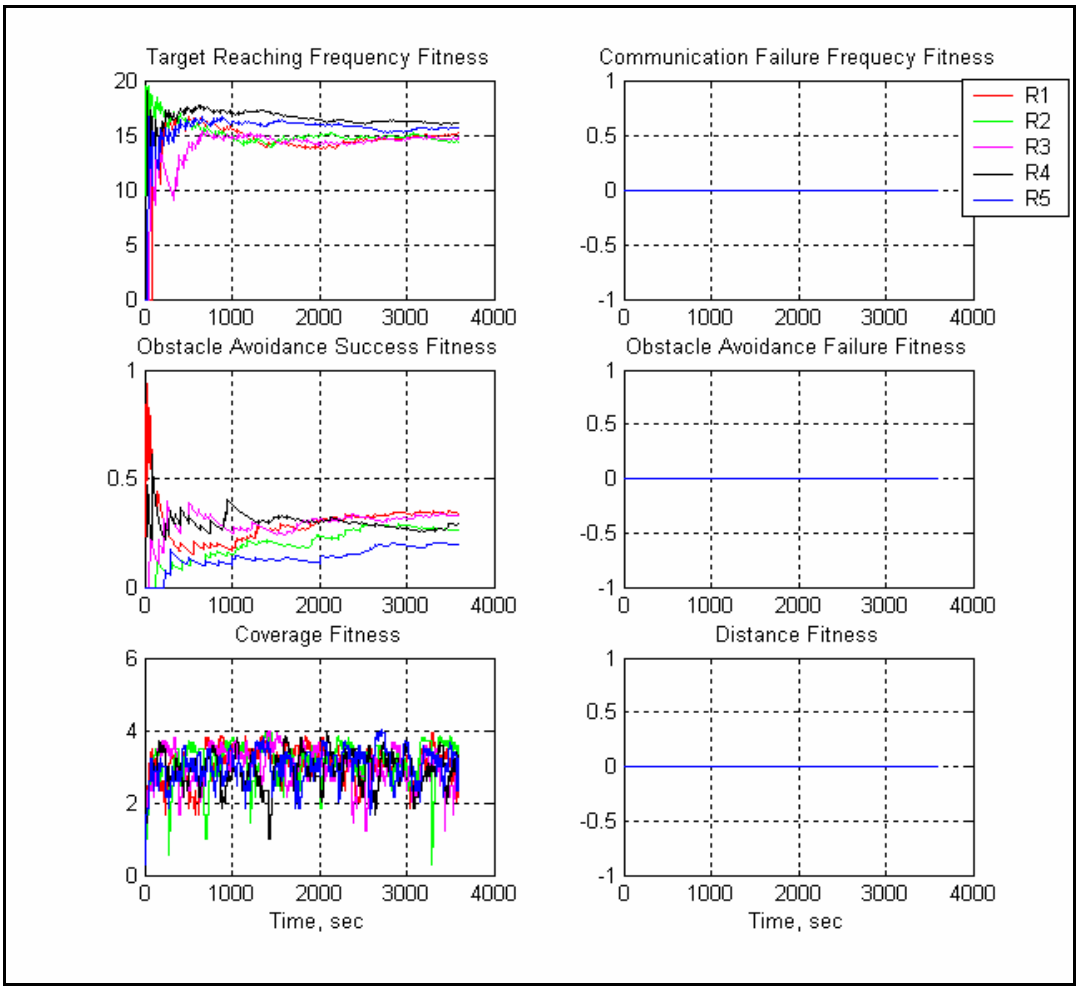


**Figure 82 Fitness functions of 5 robots in obstacle-free region for 5/(100x100) task/m$^2$ task density.**

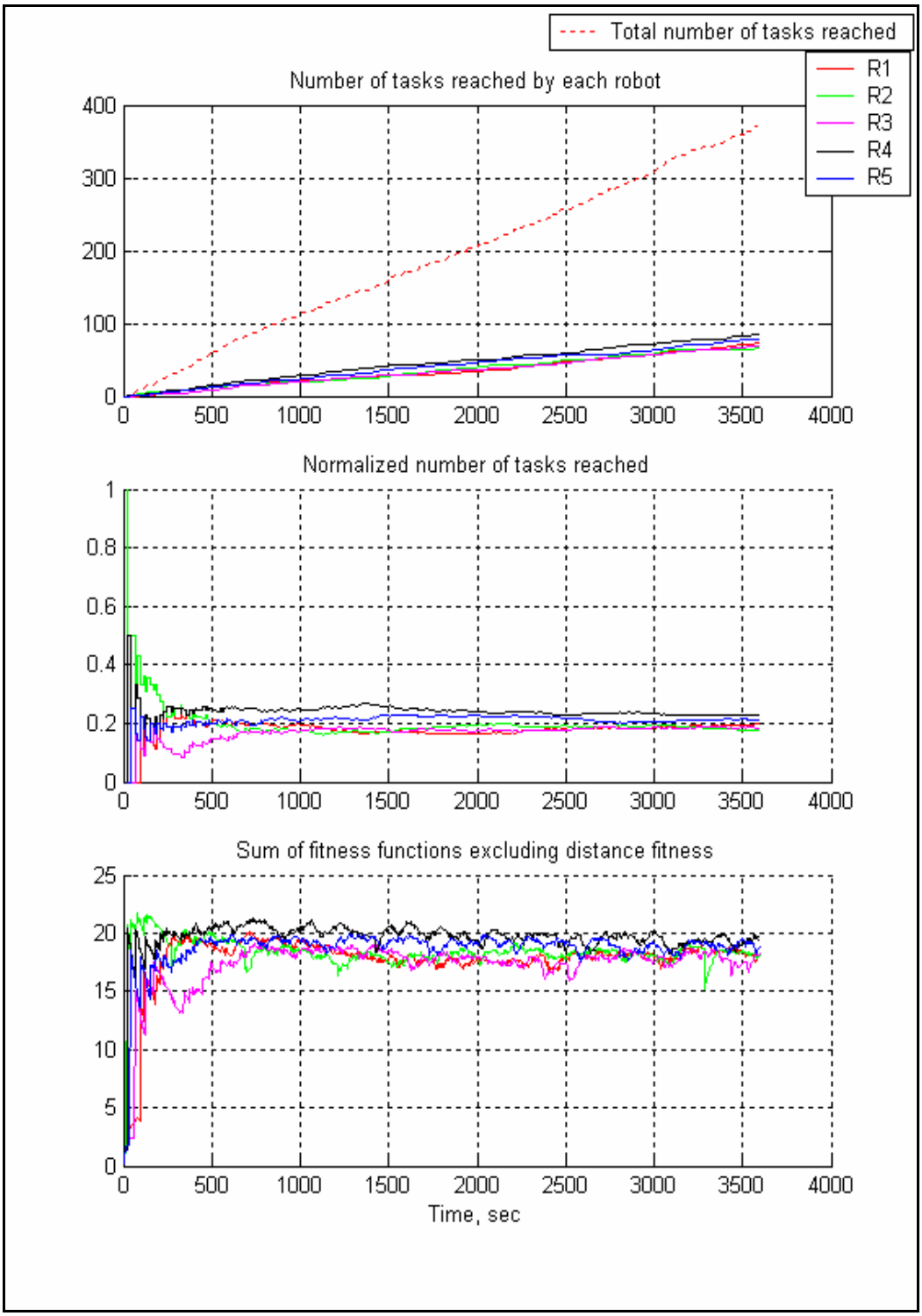


**Figure 83 Number of tasks reached statistics, and sum of fitness function excluding distance of 5 robots in obstacle-free region for 5/(100x100) task/m$^2$ task density.**

#### 6.5.1.2 Task Allocation for 5 Robots, 5 tasks at Obstacle-Dense Region

Simulation environment settings are same as listed in Table 11 except that environment is filled with obstacles. Obstacle density is 10.5%. Obstacle distribution is the same for each run. Fitness functions settings are also same with parameters listed in Table 12.

Simulation results are shown in Figure 84, and Figure 85. Since the environment is filled by obstacles, obstacle avoidance fitness functions is more active for this case. Moreover coverage fitness function fluctuates much more than that of obstacle-free case.

Discussion about for the case of obstacle-free environment with the same number of robots, and tasks are valid for this case but convergence time to almost 20% steady state task allocation is increased because of obstacles in the environment. It is nearly 1000 seconds. Results presented in Figure 85 show that fair task allocation with linear characteristic is also obtained for this case. Total task allocation fairness is 0.0217.

#### 6.5.1.3 Task Allocation for 5 Robots, 10 tasks at Obstacle-Dense Region

The aim of this simulation is to investigate the task allocation performance with respect to task density. In this case task density is doubled.

Simulation environment settings are same as with the setting listed in Table 11 except that environment is filled with obstacles, and task number is increased by 5. In this case robots are deployed in an environment having 8 2-robots synchronously correlated, and 2 1-robot uncorrelated tasks are deployed. Again task density is kept constant. Moreover, obstacle density is also10.5%. Fitness function settings are also the same as in Table 12. Simulation results are shown in Figure 86, and Figure 87. Almost perfect task allocation is obtained. Task allocation fairness is 0.0189. Since fairness metric is defined as bias in task allocation, the smaller this value is the fairer becomes the task allocation. Time to 20 percent convergence time

is around 600 seconds. If these results are compared with 5 tasks case, it is clear that convergence time is decreased and more satisfactory task allocation percentage is obtained. This is clearly due to the increase in task density. High task density increases the probability of reaching targets. In this case, robot competes less for targets because it is not difficult to find a free-robot.

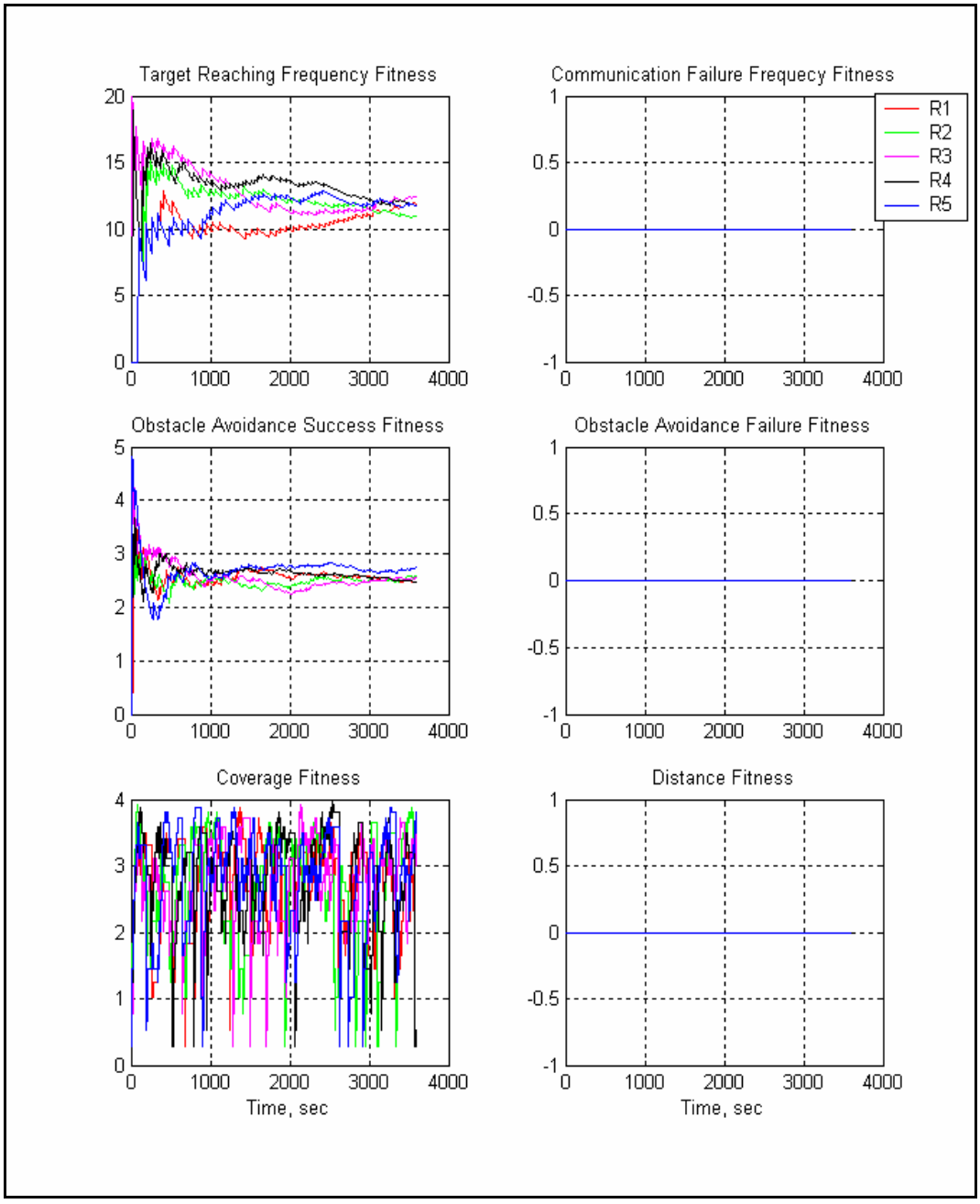


**Figure 84, Fitness functions of 5 robots in obstacle-dense region for 5/(100x100) task/m$^2$ task density**

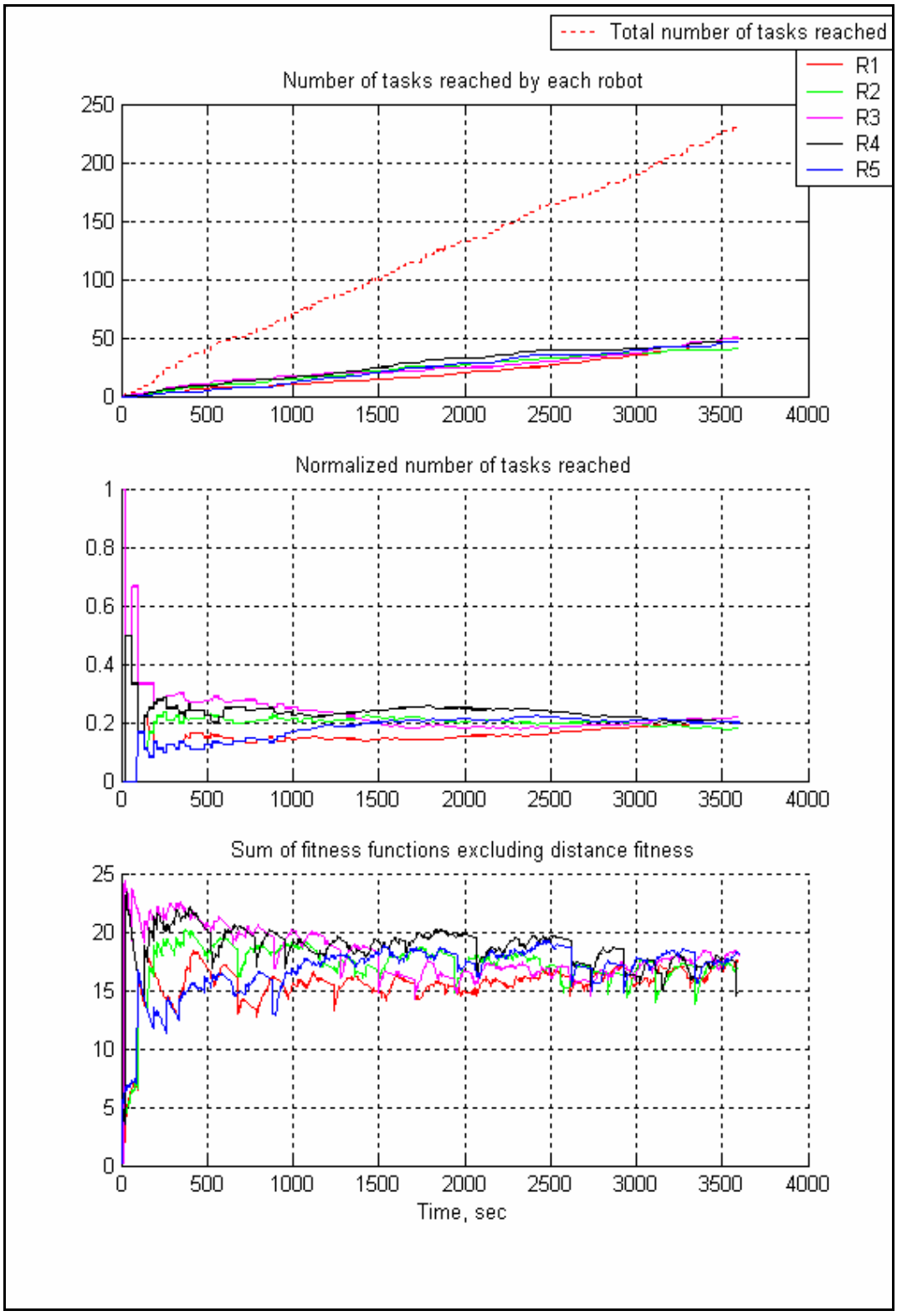


**Figure 85 Number of tasks reached statistics, and sum of fitness function excluding distance of 5 robots in obstacle-dense region for 5/(100x100) task/$m^2$ task density.**

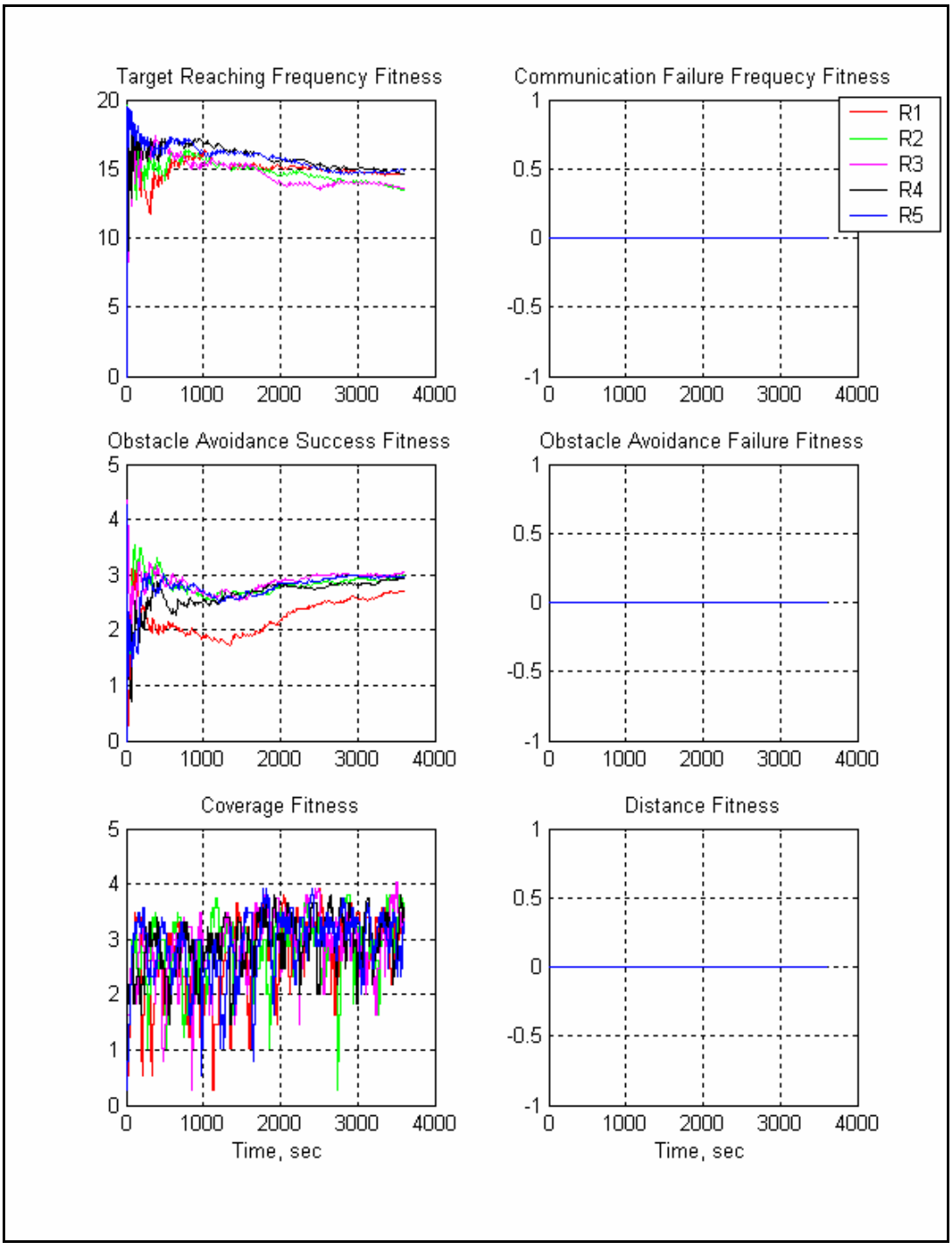


**Figure 86 Fitness functions of 5 robots in obstacle-dense region for 10/(100x100) task/m$^2$ task density.**

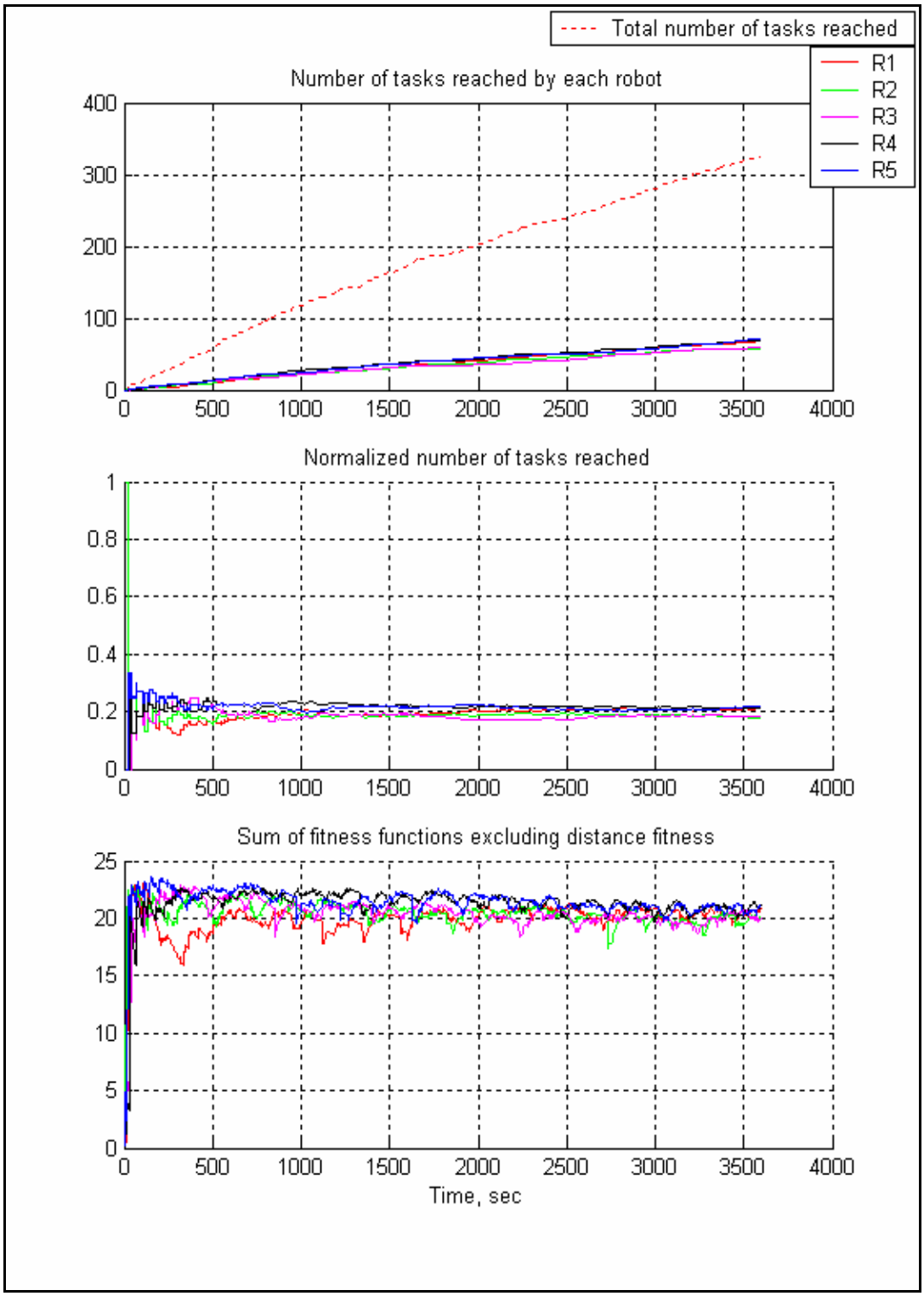


**Figure 87 Number of tasks reached statistics, and sum of fitness function excluding distance of 5 robots in obstacle-dense region for 10/(100x100) task/m$^2$ task density.**

### 6.5.2 Selection of Fitness Functions

Choice of fitness functions is very important because it directly determines the fairness of task allocation, and fault tolerance. All of the proposed fitness functions were active in fairness analysis experiments. In this subsection, only one fitness function that of target reaching will be activated to observe its individual effect of fitness function on fairness and the same analysis will be repeated for coverage and distance fitness function.

It is not meaningful to analyze the obstacle avoidance success fitness function because the robot may operate in obstacle-free regions. Other fitness functions, communication failure, and obstacle avoidance failure functions are not suitable for task allocation purpose alone. They are designed for fault tolerance.

Simulation environment settings are adjusted to be the same with setting listed in Table 11 except that environment is filled with obstacles. Obstacle density is 10.5%.

#### 6.5.2.1 First Fitness Analysis: Target Reaching Frequency

It is very meaningful to use target reaching frequency as a fitness parameter for task allocation. In Figure 88, target reaching percentages are shown for this case. By taking only target reaching frequency as fitness criterion, task allocation is not so satisfactory. The task allocation percentages at the steady state are given in Table 13. Almost 30% of all tasks are allocated to Robot1, whereas it is 13.5% for Robot4. Total task allocation fairness is 0.0847. It is clear that fair task allocation is not obtained as compared with the experimental results where all of the fitness functions were enabled. Moreover the linearity at steady state of task allocation got worsen.

Simulation results show that, usage of target reaching frequency as fitness function alone is not a good choice.

**Table 13 Task allocation percentages for 5 robots when only target reaching frequency is activated.**

| Robot | Steady state task allocation percentage |
|---|---|
| Robot1 | 27 |
| Robot2 | 21.5 |
| Robot3 | 19.5 |
| Robot4 | 18.5 |
| Robot5 | 13.5 |

#### 6.5.2.2 Second Fitness Analysis: Coverage

Coverage can also be used as task allocation fitness criterion for missions requiring some search procedures. Coverage fitness has two important characteristics:

- It represents how good robots wander.
- It gives information about how frequent target reaching is done.

If robot target reaching is very frequent then coverage fitness value decreases. This gives a kind of fairness. If a robot reaches a target then its coverage fitness will decrease but other robots coverage will remain relatively high.

In Figure 89, target reaching percentages is shown for this case. At steady state, task allocation percentages are given in Table 14. Task allocation fairness is 0.0293. Task allocation results are fair enough as compared with the case only target reaching frequency is activated. Task allocation linearity is also satisfactory.

Results show that coverage fitness can be used as fitness parameter but this is only valid if all robots are functioning perfectly. In case of any fault, this parameter will not filter out some kind of faults such as robots primary sensor errors or communication errors.

**Table 14 Task allocation percentages for 5 robots when only coverage fitness is activated.**

| Robot | Steady state task allocation percentage |
|---|---|
| Robot1 | 22 |
| Robot2 | 21 |
| Robot3 | 20 |
| Robot4 | 19 |
| Robot5 | 18 |

### 6.5.2.3 Third Fitness Analysis: Distance

Distance to task location can also be used in a very simple manner as fitness criterion allocating the task to the robot nearest to that task location. Of course this simplicity can not do anything about the fault tolerance issue. In [48], and [51], distance based metric evaluation is made

In Figure 90, target reaching percentages is shown for this case. In Table 15, task allocation percentages are shown. Task allocation fairness is 0.0439. Task allocation is almost fair, and task allocation rate is almost constant.

The fairness analysis of different metrics shows that only coverage metric usage enables fairer task allocation as compared to using only target reaching, and only distance fitness function.

**Table 15 Task allocation percentages for 5 robots when only distance fitness is activated.**

| Robot | Steady state task allocation percentage |
|---|---|
| Robot1 | 24 |
| Robot2 | 21 |
| Robot3 | 19 |
| Robot4 | 18 |
| Robot5 | 18 |

In Table 16, task allocation fairness metric for different fitness metrics are given. Proposed fitness functions usage gives total 0.0217 task allocation fairness metric. On the other hand, distance based fitness calculation gives 0.0439 fairness showing that our proposed method enables almost 2 times fairer task allocation than distance based fitness calculation method used in the literature. Usage of coverage, obstacle avoidance and distance fitness measures at the same time enables fairer task allocation.

**Table 16 Task allocation fairness metric for different fitness metrics**

| **Fitness Metric** | **Fairness metric** |
|---|---|
| Six fitness parameters are activated | 0.0217 |
| Distance | 0.0439 |
| Coverage | 0.0293 |
| Target reaching frequency | 0.0847 |

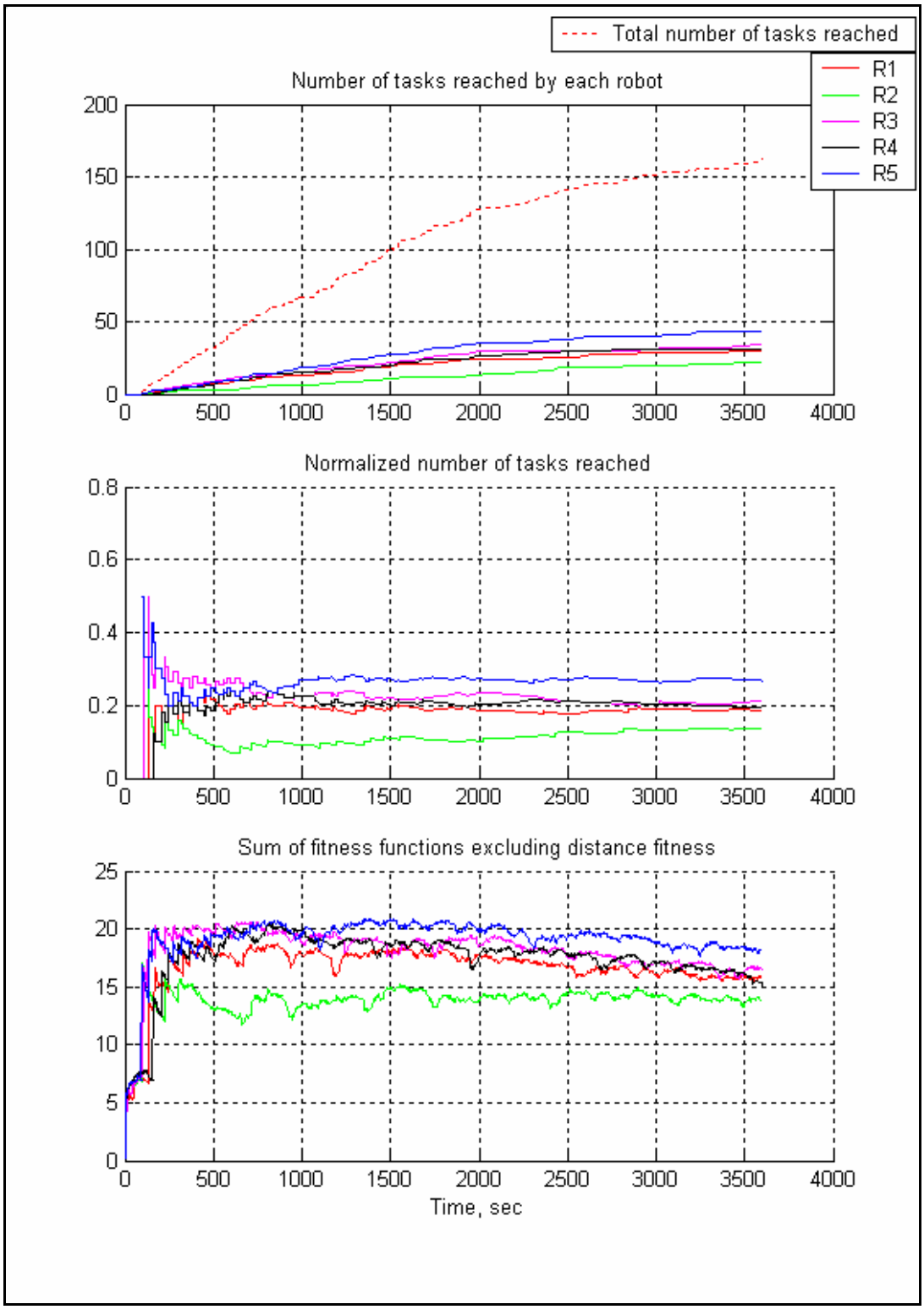


**Figure 88 Only target reaching frequency fitness function is activated. Number of tasks reached statistics, and sum of fitness function excluding distance of 5 robots in obstacle-dense region for 5/(100x100) task/m$^2$ task density.**

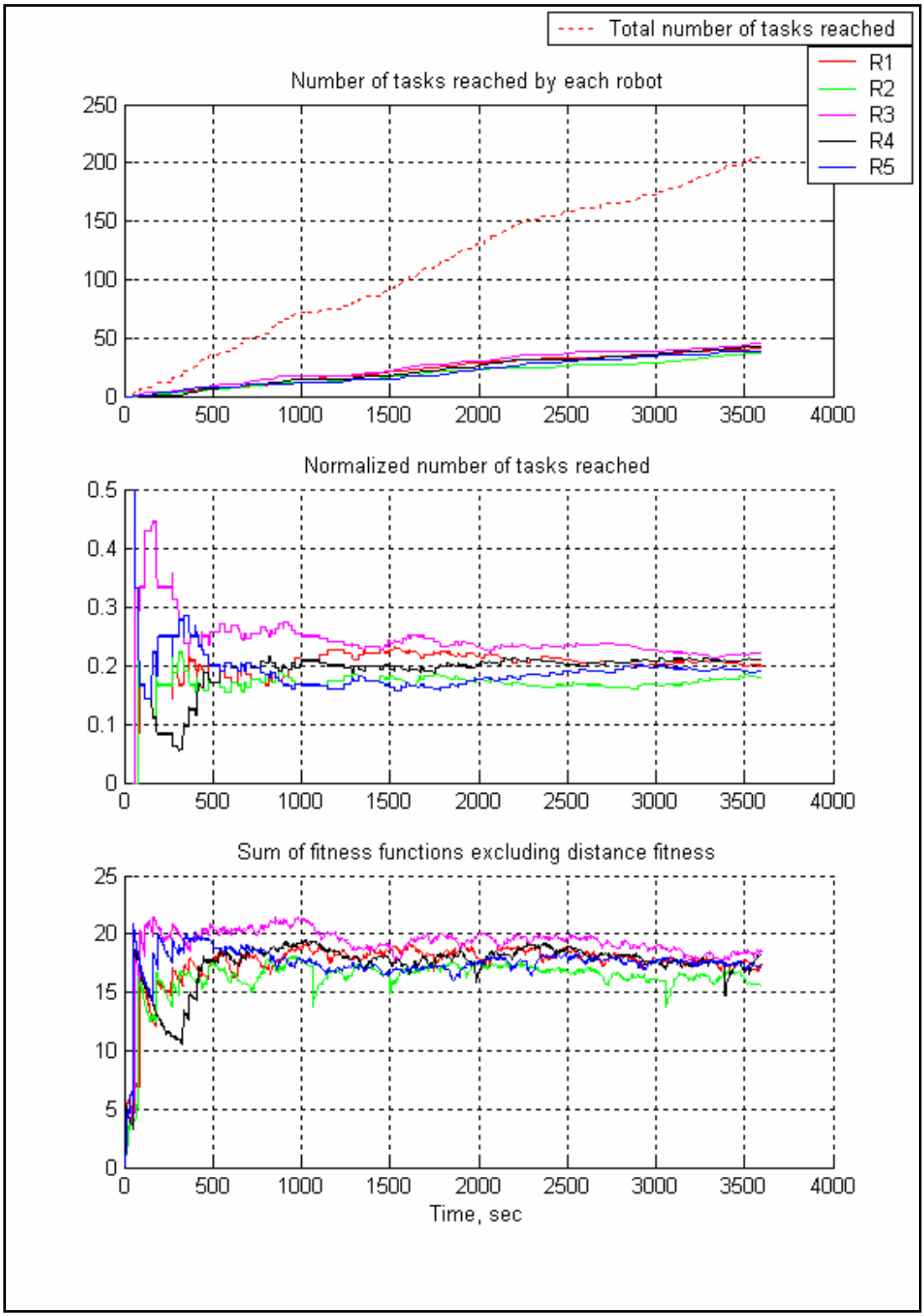


**Figure 89 Only Coverage fitness function is activated. Number of tasks reached statistics, and sum of fitness function excluding distance of 5 robots in obstacle-dense region for 5/(100x100) task/m$^2$ task density**

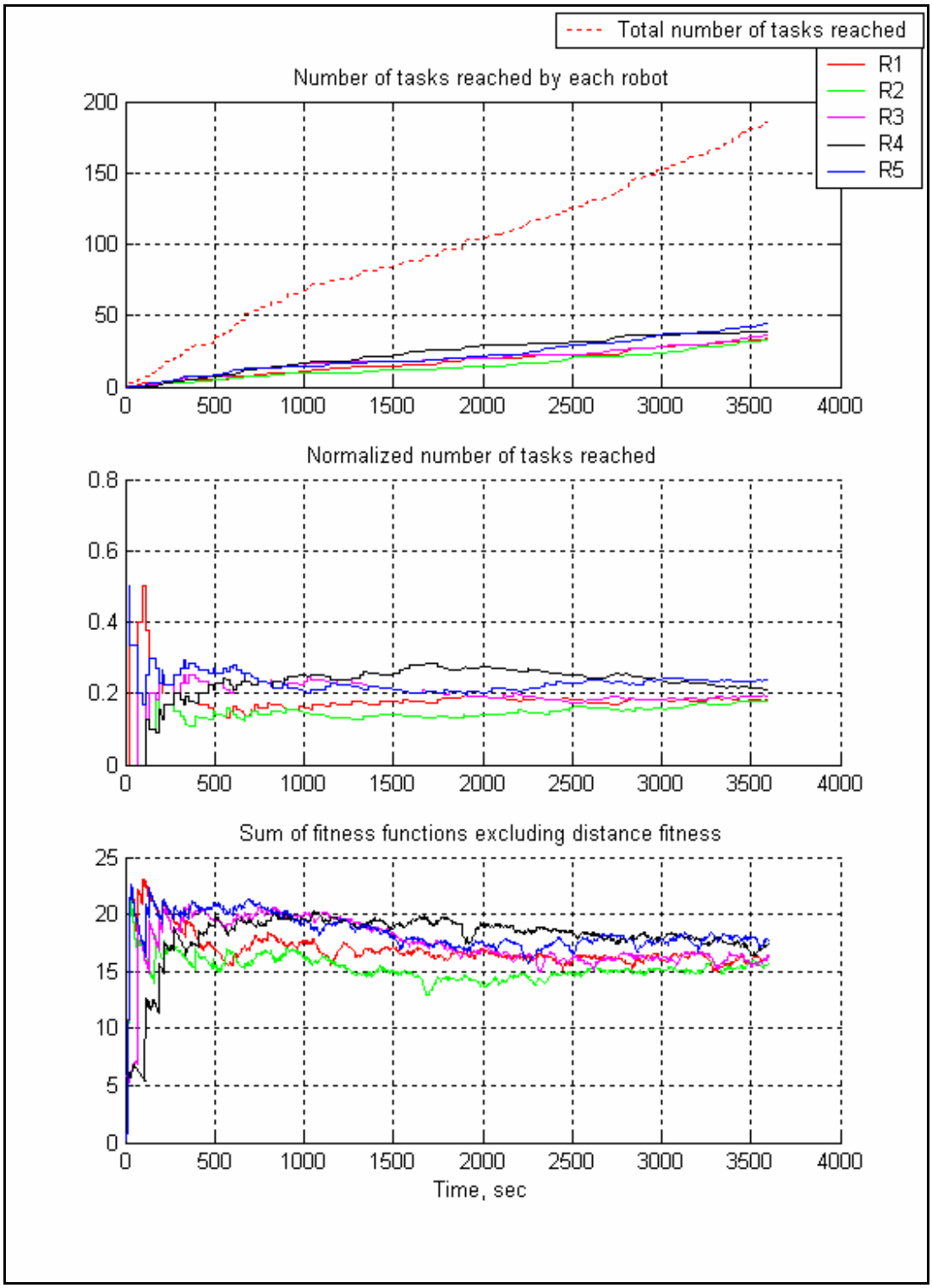


**Figure 90 Only Distance fitness function is activated. Number of tasks reached statistics, and sum of fitness function excluding distance of 5 robots in obstacle-dense region for 5/(100x100) task/m$^2$ task density**

### 6.5.3 Fault Tolerance Analysis

In this section fault tolerance of the proposed system will be analyzed. For this purpose, an artificial error situation is simulated. Some of the deployed robots are made not to functioning perfectly, i.e. they are corrupted partly or completely. Consequently the following error situations are simulated:

- Target reaching errors
- Obstacle avoidance error
- Physical integrity error.

In case of above errors, task allocation to partly corrupted robots will be analyzed. It is expected that fitness of this partly or completely corrupted robots should be very low so that task allocation should not be made to them. In Table 17, simulated fault types are given. Moreover, reasons, effect and simulation methods of these faults are also given.

**Table 17 Simulated fault types, and information about these faults**

| **Fault Type** | **Main Reason** | **Effect** | **Simulation Method** |
|---|---|---|---|
| Target reaching errors | Fault in robot's sensors or communication hardware | Robot cannot make efficient target reaching. | Robot cannot lock targets |
| Obstacle avoidance error | Fault in robot's secondary sensors | Robot cannot make efficient obstacle avoidance. It can crash into obstacle and destroy its physical integrity. It can stuck and loose time. | Robot average speed is decreased to 33% of average speed of normal robot while obstacle avoidance behavior is active. |
| Physical errors | Fault in robot motors or any kind of physical problem | Robot cannot move appropriately. Robot may fail to execute the tasks. | Robot average speed is decreased to 10% of average speed |

### 6.5.3.1 Target Reaching Errors

Target reaching error is simulated using communication errors. When a robot is allocated to execute a task, it will not lock on to the task, and therefore it will make communication faults. So communication failure frequency fitness function will be activated. It is expected that the fitness of a robot having this kind of fault will decrease as compared with other robots.

Simulation is done at obstacle-free environment. Simulation environment settings are also same with setting listed in Table 11 except for task properties listed in Table 18. In this error simulation, only one of deployed robots is defective. Target reaching error simulation start time, and stop time are given in Table 19. After stop time, robot begins to function again properly by not doing any faults.

**Table 18 Robot, and task properties for target reaching error simulation**

| | |
|---|---|
| Robots | 5 robots, all robots are the same type. |
| Robots fault status | Robots are functioning perfectly except for Robot1 |
| Robots' initial position | At the center of region |
| Tasks | 3 2-robots synchronously correlated task |
| Task deployment Strategy | Task density is kept constant. Targets are randomly. |
| Target Density, targets/ $m^2$ | 3/(100x100) |

**Table 19, Target reaching fault simulation timing**

| **Fault Simulation Method** | **Start Time, sec** | **Stop Time, sec** | **Duration, sec** |
|---|---|---|---|
| Robot cannot lock to task allocated via communication | 1000 | 2000 | 1000 |

In Figure 91 all of fitness functions are given. In Figure 92, target reaching counts for each robot, and sum of fitness functions with respect to time are given. **Red line** in each subplot belongs to the robot making target reaching error. It is shown in Figure 91 that the faulty robot has made a target reaching error at time 1113 second. At that instant robot total fitness is reached to -28.5 excluding distance

fitness. Since task allocation algorithm requires that sum of fitness functions excluding distance should be greater or equal to zero, this robot should not undergo any task allocation. The robot should wait for the sum of fitness to exceed zero. As it is shown in Figure 92, defective robot fitness reaches zero near time 2000. After 2000, no error is simulated so robot is functioning with no fault.

In Figure 92, in subplot 2, normalized task allocation for each robot is shown. After the time communication error occurred, number of tasks allocated to defective robot decreased considerably due to the decrease in fitness functions. Since tasks are not allocated to the defective robot, task allocation linearity is not violated. In Figure 92, time interval, where robot1 could not make any task allocation, is marked. In this interval, the slope of task allocation line is decreased because effective number of robots decreased to 4 from 5, i.e. the total number of task allocated is decreased. After this time interval, robot 1's fitness becomes greater than zero starts again to get allocated tasks after time 1960 seconds. Above results shows that task allocation is performed well in case of target reaching errors.

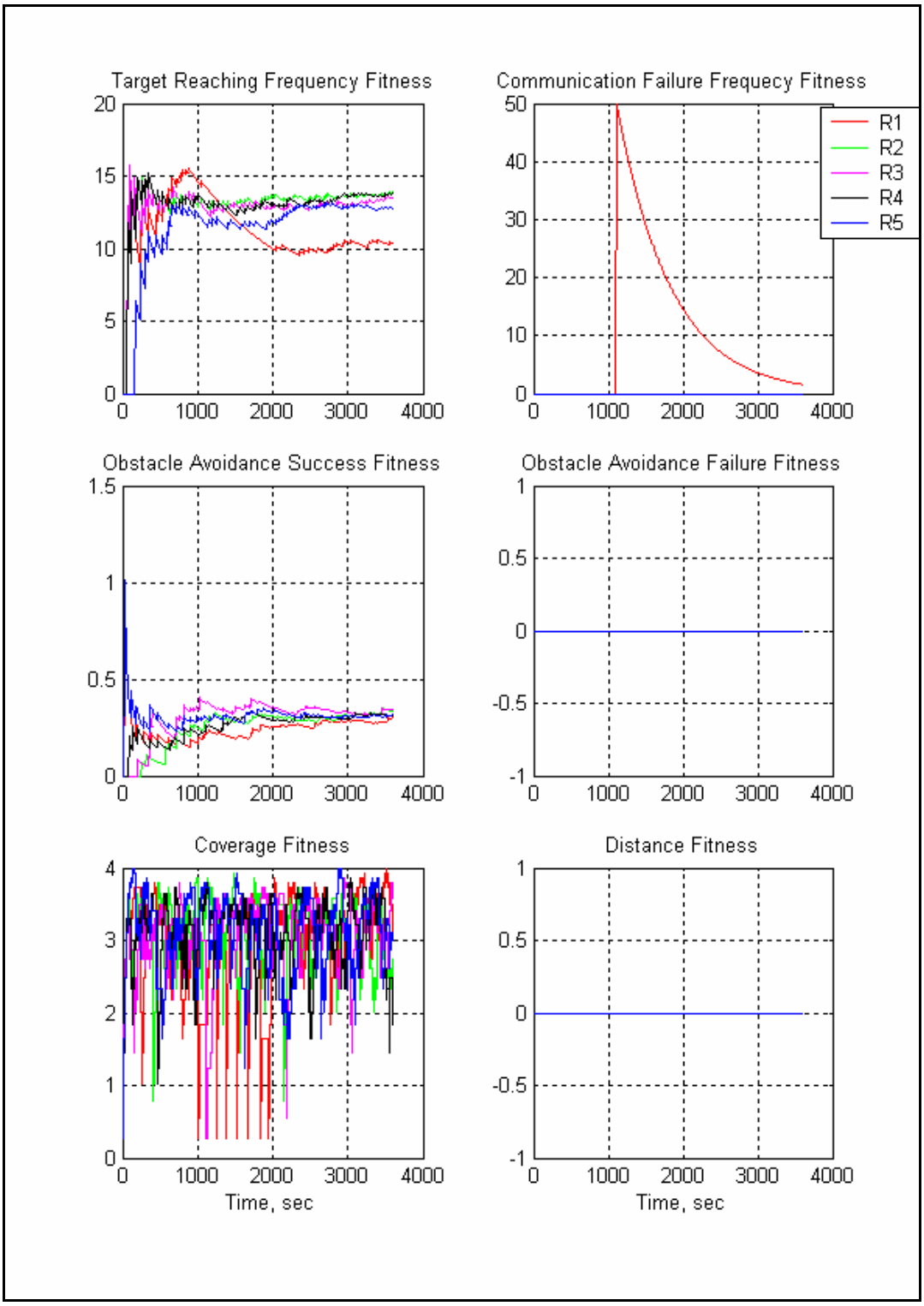


**Figure 91 Fitness functions for all robots. Red line is belonging to defective robot making target reaching errors.**

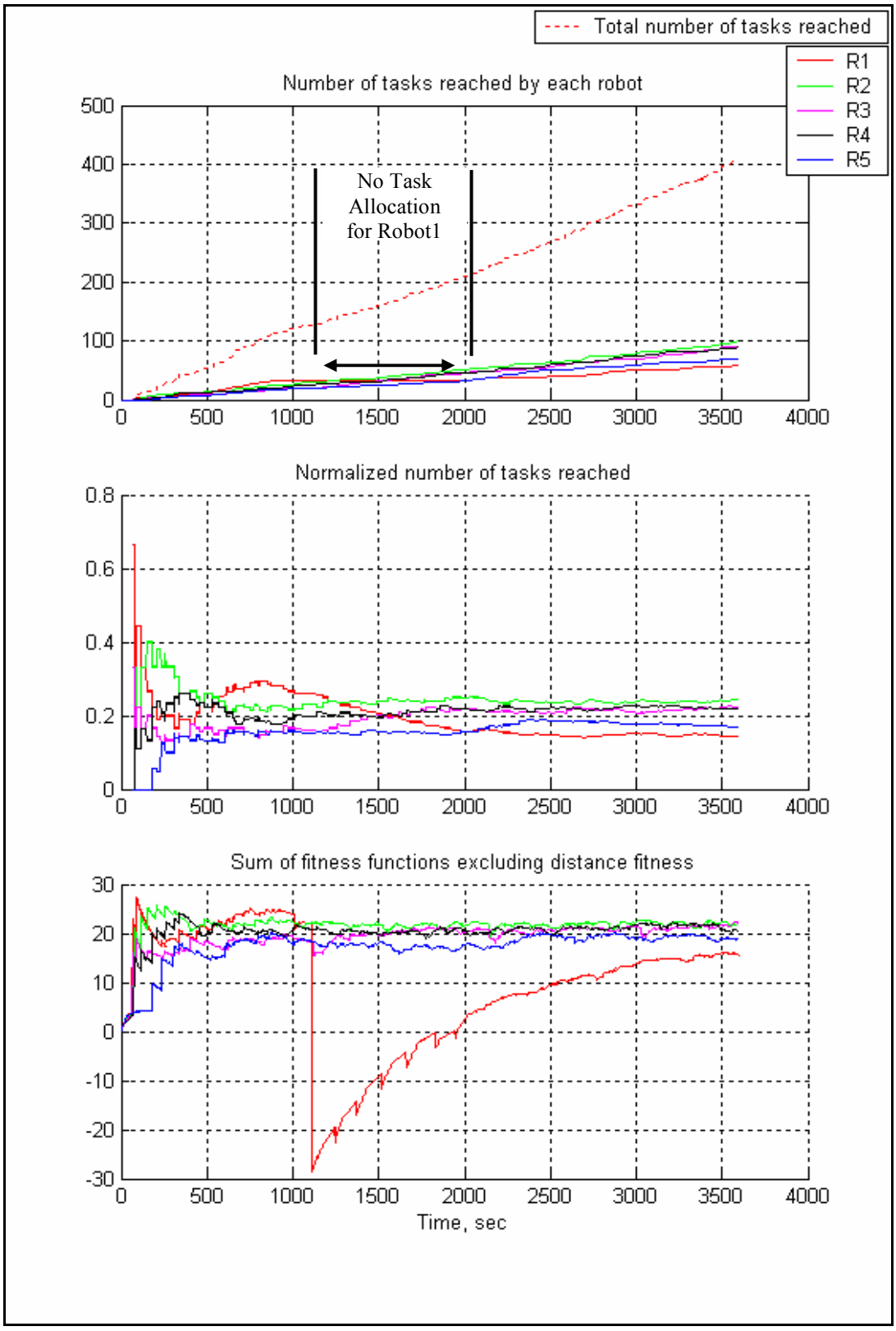


**Figure 92 Number of tasks allocated, and sum of fitness functions for all robots. Red line is belonging to defective robot making target reaching errors.**

### 6.5.3.2 Obstacle Avoidance Errors

Obstacle avoidance error is simulated in a %10.5 obstacle filled environment. Error is simulated by decreasing the average speed of the robot described in Table 17. Properties of robots and task are given in Table 20. In this case, average task number is increased to 10.

**Table 20 Robot, and task properties for obstacle avoidance error simulation**

| | |
|---|---|
| Robots | 5 robots, all robots are the same type. |
| Robots fault status | Robots are functioning perfectly except for Robot1 |
| Robots' initial position | At the center of region |
| Tasks | 10 2-robots synchronously correlated task |
| Task deployment Strategy | Task density is kept constant. Targets are randomly. |
| Target Density, targets/ $m^2$ | 10/(100x100) |

**Table 21 Obstacle avoidance fault simulation timing**

| Fault Simulation Method | Start Time, sec | Stop Time, sec | Duration, sec |
|---|---|---|---|
| Robot average speed is decreased to 33.3% | 1000 | 2000 | 1000 |

Obstacle avoidance fault is simulated according to timing given in Table 21. Fitness functions are given in Figure 93. Task allocation statistics and sum of fitness functions are given in Figure 94. Red lines in the plots belong to the defective robot (robot 1). Robot1 has started to make obstacle avoidance error starting at time 1014. At this point, robot 1's overall fitness goes to -28.8. Since total fitness excluding distance fitness is less than zero, this robot cannot be allocated any task. Robot1 continues to make obstacle avoidance error up to time 2000 but its fitness reaches zero after 2168 seconds, i.e. even if there is no obstacle avoidance error after time 2000, due to the decaying effect therefore due to the transient effect of the obstacle avoidance failure fitness function, the robot cannot make any task allocation for 168

seconds after error simulation is removed. This is the confidence interval. Since robot did not make any errors, its fitness increases gradually, but it takes some time to reach other robots having no fault.

Results for target reaching error are also valid for this case. Task allocation linearity is preserved for time intervals in which error is simulated or not simulated. Slope of total number of task allocation at error simulation interval is slightly less than the slopes at other instants. It is clear that task allocation is done effectively in case of obstacle avoidance faults.

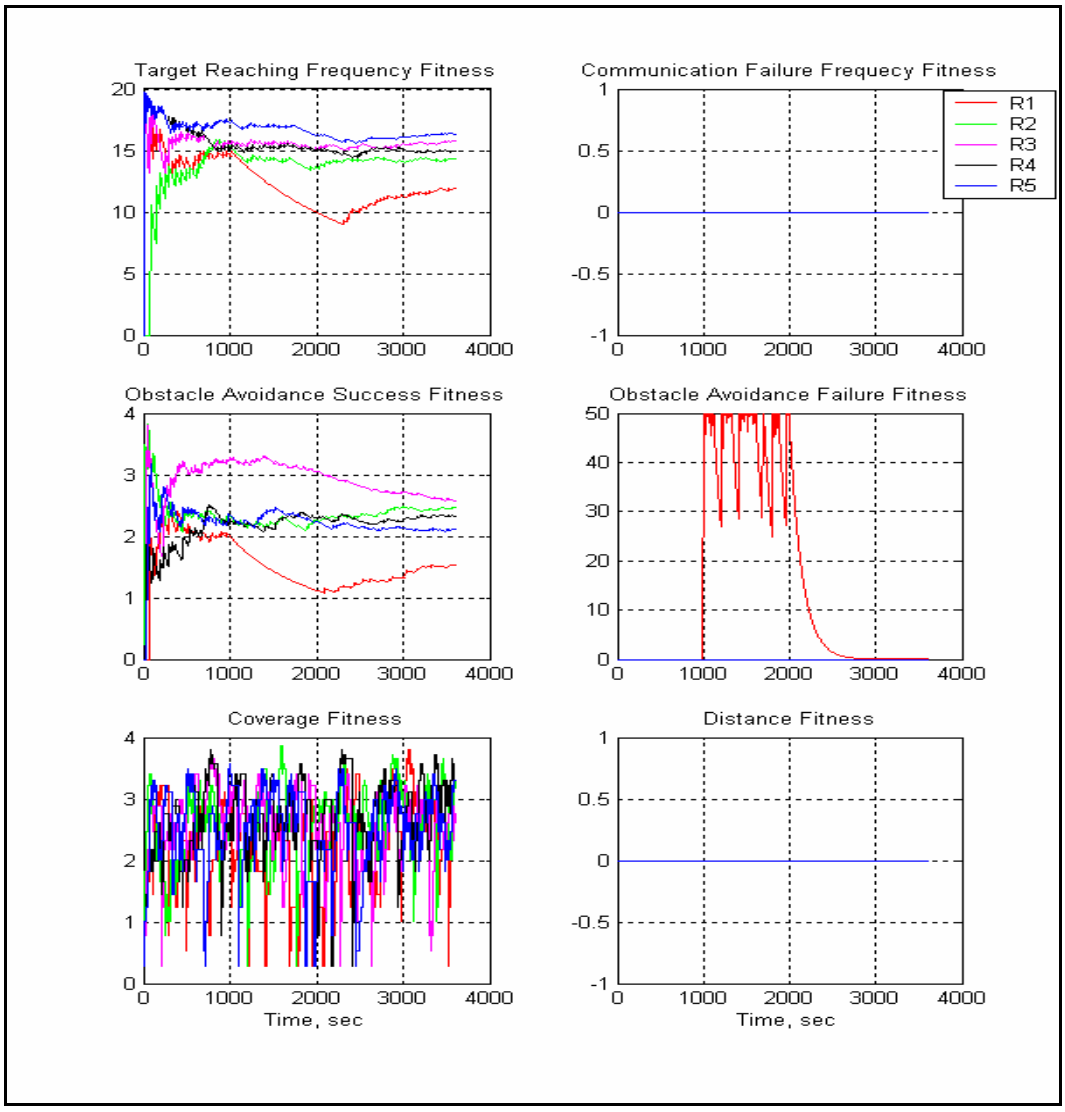


**Figure 93, Fitness functions for all robots. Red line is belonging to defective robot making obstacle avoidance errors.**

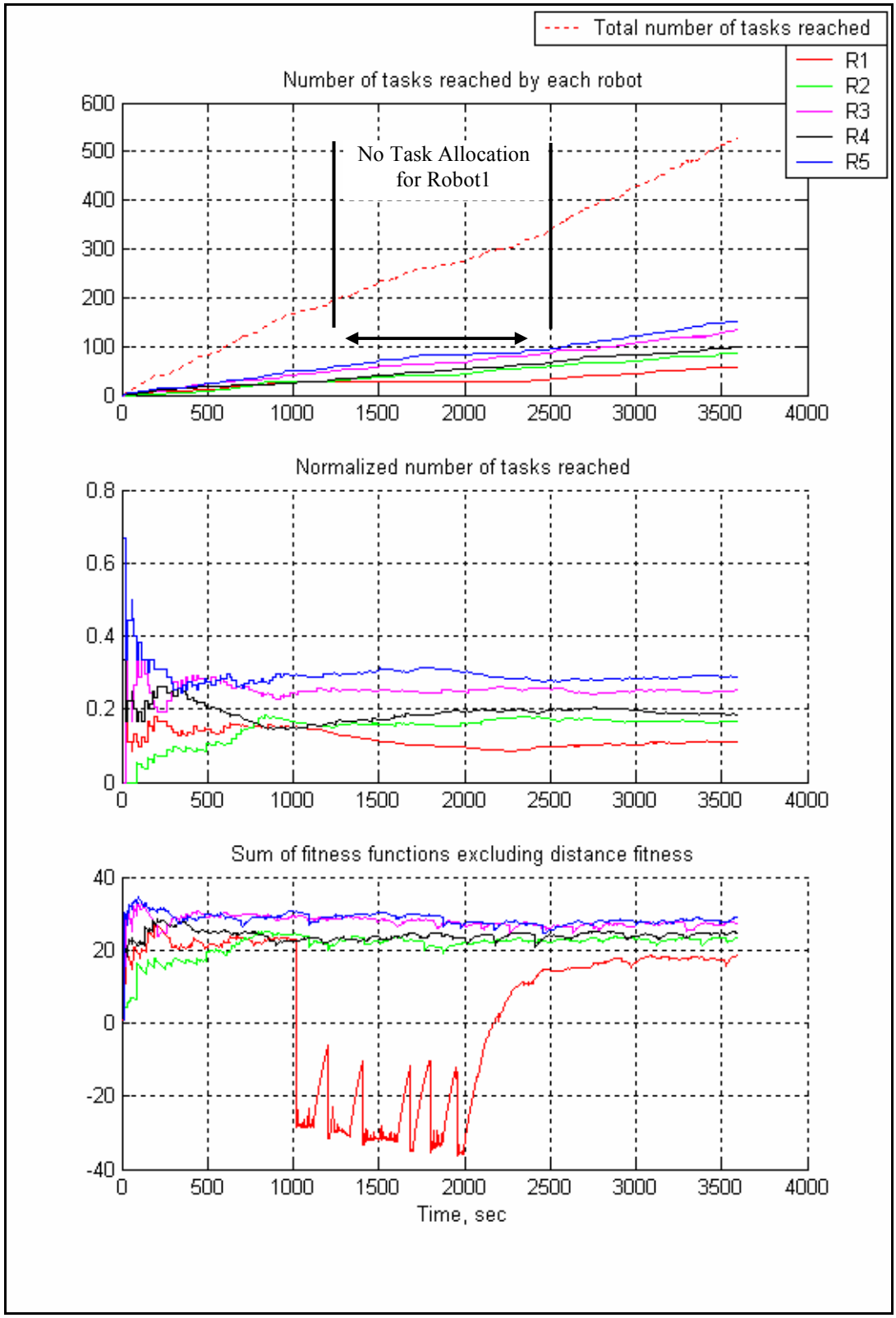


**Figure 94 Number of tasks allocated, and sum of fitness functions for all robots. Red line is belonging to defective robot making obstacle avoidance errors.**

#### 6.5.3.3 Physical Errors

Physical errors include robot physical structural errors such as faults in motors. These kinds of errors result in speed degradation.

Physical error is simulated in an obstacle-free environment. Error is simulated by decreasing average speed of robot described in Table 17. Properties of robots and task are given in Table 22.

**Table 22 Robot, and task properties for obstacle avoidance error simulation**

| | |
|---|---|
| Robots | 5 robots, all robots are the same type. |
| Robots fault status | Robots are functioning perfectly except for Robot1 |
| Robots' initial position | At the center of region |
| Tasks | 3 2-robots synchronously correlated task |
| Task deployment Strategy | Task density is kept constant. Targets are randomly. |
| Target Density, targets/ $m^2$ | 10/(100x100) |

**Table 23 Physical fault simulation timing**

| Fault Simulation Method | Start Time, sec | Stop Time, sec | Duration, sec |
|---|---|---|---|
| Robot average speed is decreased to 10% | 1000 | 2000 | 1000 |

Physical fault is simulated according to timing given in Table 23. Fitness functions are given in Figure 95. Task allocation information and sum of fitness functions are given in Figure 96. Red lines in the plots again belong to the defective robot (robot 1).

Robot 1 starts to make this error beginning at time 1000. From this instant, coverage fitness function decreases to almost zero. But the actual fitness drop is obtained at instant 1615. Robot 1 makes a communication error because of a task allocated to this robot. But since it did not reach the target point then its task execution has failed. This communication error decreases the total fitness of robot 1

to -40, so it could not make any task allocation from then on. It is only after time 2910 that the robot fitness reaches zero and robot1 can make task allocation. After time 2000, coverage fitness again starts to increase taking normal values.

Number of allocated task is also linear for this case for each simulation interval whether error simulation is active or not. This shows that task allocation is performed well among the healthy robots. As presented in this case, physical errors have serious deeper side effects affecting different fitness functions depending on the type of error.

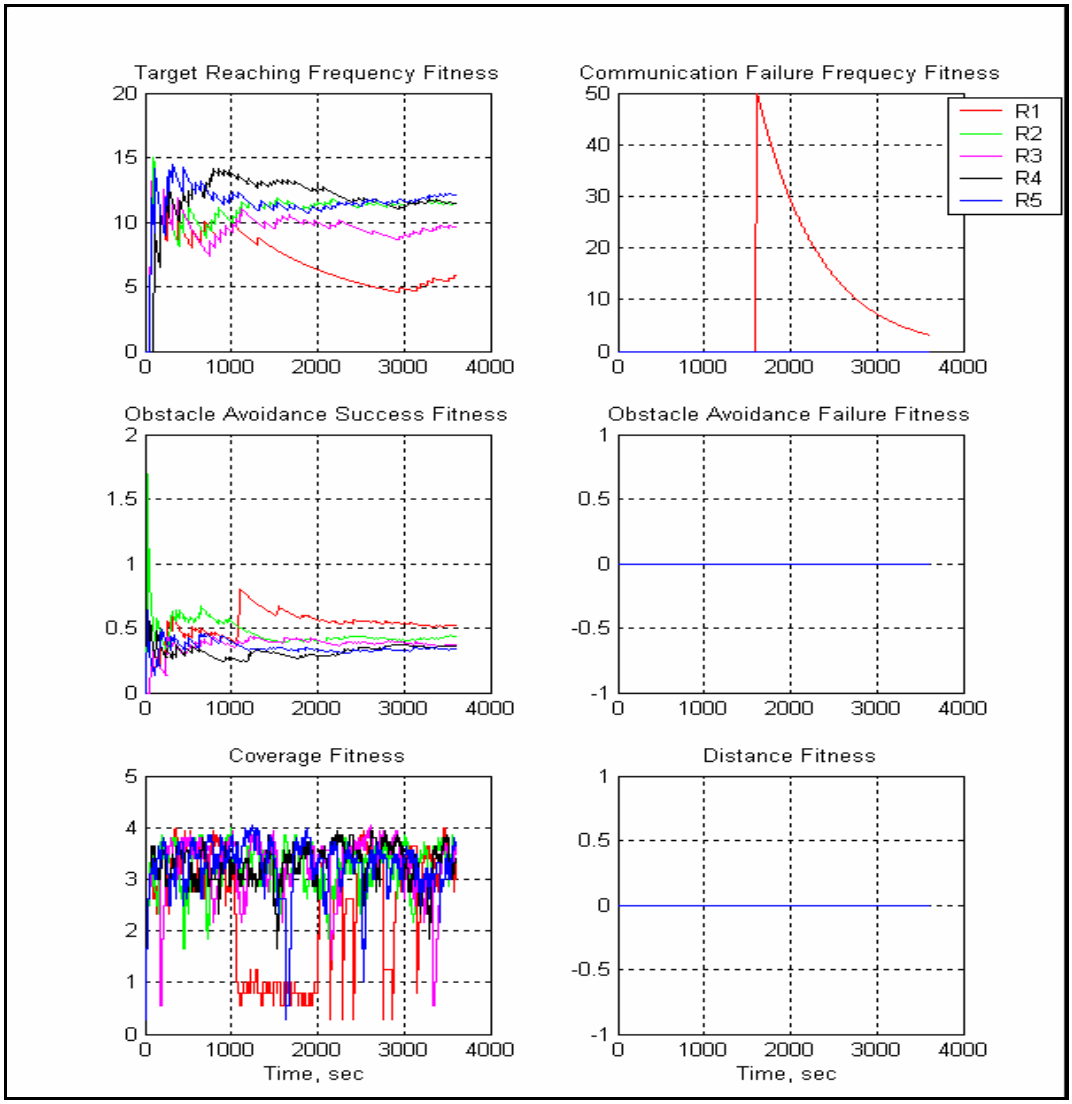


**Figure 95 Fitness functions for all robots. Red line is belonging to defective robot making physical errors.**

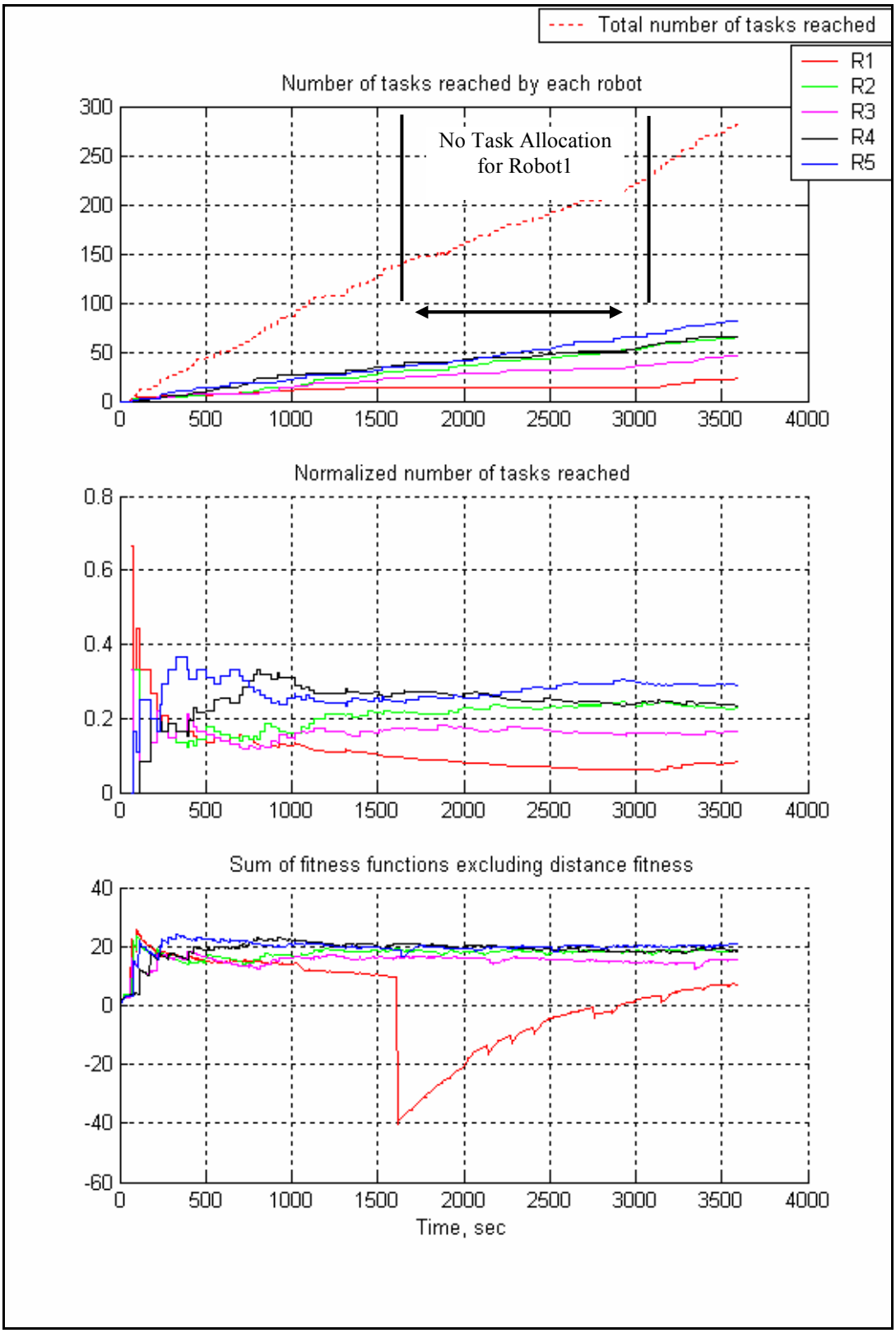


**Figure 96 Number of tasks allocated, and sum of fitness functions for all robots. Red line is belonging to defective robot making physical integrity errors.**

### 6.5.4 Fitness Summary

In sections from 6.5.1 to 6.5.3, fitness calculation of robots for task allocation has been analyzed with extensive number of simulations. Fairness, and fault tolerance are the main inherent issues of in fitness calculation. Moreover there is an important parameter, linearity of task allocation with respect to time. It is expected that, for constant number of robots, and task densities, the number of tasks allocated or executed should be constant regardless of the simulation instant.

Usage of 6 fitness functions enables the robots to allocate tasks fairly, and fault tolerantly. Fairness is tested for both obstacle-dense, and obstacle free regions for various robot numbers, and task densities. Fairness metric for these case ranges from 0.0189 to 0.0217. Results are satisfactory about fairness, and task allocation linearity.

Moreover, selections of fitness functions are also important. If the usage of fitness function is unnecessary it should be used. For this purpose, target reaching frequency, coverage, and distance fitness functions are tested individually as an only fitness functions.

For these cases, fairness metric is not as good as the cases in which all fitness functions are enabled. Fairness of task allocation is almost 3 times less fair than those of the case when all functions are enabled. Linearity of task allocation is not as linear as expected. Moreover, fault tolerance will not be satisfied if these fitness functions are used individually.

Fault tolerance analyses are held for three error situations:

- Target reaching error
- Obstacle avoidance error
- Physical integrity error

Simulation results show that these kinds of errors are filtered well: no task is allocated for/ defective robots. If the task allocation to defective robots is inhibited,

the long time of operation is needed to reach fair task allocation. This is because of fitness of defective robots gradually increases but it takes some time to reach other robots' fitness value. Fair task allocation requires more time in case of fault.

When an error has occurred, fitness functions (communication failure frequency fitness, obstacle avoidance failure fitness) dedicated to error situations is triggered. These fitness functions decreases more rapidly as compared with others. Selection of parameters of fitness functions is important. Overestimation of these parameters may worsen the fault tolerance. Decaying factors should be selected by considering average target detection, and execution time.

# CHAPTER 7

# Conclusion, and Future Works

In this thesis, a hybrid behavior based architecture for a multi-robot system is implemented. The thesis covers

- Designing a flexible, modular, scalable, as much as realistic 3D simulation environment
- Designing a hybrid of both subsumption and motor schema architecture
- Implementation of external and internal behavior
- Implementation of task allocation based on the well known market based auction algorithm
- Designing powerful task allocation metric allowing fair and fault tolerant task allocation
- Extensive experiments for performance analysis both in low level (behavior level) and high level (system level)

## 7.1 Simulation Environment

Simulation environment design is a challenging and time consuming work. Existing simulation environments can be used instead of designing a new environment. But this brings many limitations to the users. Many simulation environments are not based on 3D environment modeling. Once a base simulation environment is designed, it is not difficult to extend this environment to more realistic levels. Main advantage of designing own simulation environment is that every point of environment is open for designer. Designed environment are

satisfactory for simulation and test purposes except for realistic sensor, and robot dynamical model.

## 7.2 Behavioral Architecture

Control and interaction of behaviors are achieved in a hybrid style. Behavior coordination is based on both subsumption, and motor schema type. A layered control strategy is implemented. One layer is devoted to subsumption type control, and other is devoted to motor schema type control. Behaviors in the subsumption layer deal with the complicated tasks. On the other hand, behaviors in motor schema layer deal with tasks requiring reactivity.

Behaviors having relative priority with respect to each other reside in the subsumption layer, whereas behaviors having equal priority reside in the motor schema layer. These layers are coordinated cooperatively like in motor schema architectures. The reason behind the hybrid architecture is to take advantage of flexibility of controlling robots more accurately and effectively without violating reactivity.

In classical subsumption architecture, behaviors do not know anything about the state of each other. Behaviors are coordinated via lines (suppress, inhibits, and reset lines) in a priority based style. This coordination enables incremental and modular design. In complex systems this may not be easy. Because, there will be need for interaction of some low and high level behaviors. To overcome this difficulty, **evaluators** are introduced. In this thesis, evaluators can be used as

- Defining priorities of behaviors in run-time
- Defining loosely coupled coordination between behaviors in subsumption layer and motor schema layer. Evaluator takes state of other behaviors as additional input parameters

Introduced control architecture does not degrade modularity, and reactivity. It gives an additional coordination mechanism, and flexibility. High, low, and equal priority behaviors can be controlled effectively by additional information about state of behaviors. Moreover, introduced system can be reduced to subsumption style control if evaluators are disabled.

Motor control of robots as an evaluation of external behaviors is tested, and simulation results are given sections from 6.1 to 6.3. Simulation results show that, external behaviors are functioning well enough individually.

Obstacle avoidance behavior is one of the basic and most important behaviors. Implementation of this behavior is very critical. For effective obstacle avoidance both repulsive and tangential potential field forces are generated. These two kinds of forces enable the robot to avoid obstacles more rapidly, and securely. Obstacle avoidance behavior is tested for convex, concave, partially concave and strongly concave shapes. Avoiding from strong concave obstacles is difficult using potential field because there is a probability of falling into local minima. But the implemented obstacle avoidance algorithm minimizes this probability with the help of the behavior integration.

The main task of the robots is first finding tasks, and if multi-robot task allocation is required then robot tries to gather enough number of other robots. So detection phase of the tasks is the first thing that a robot should achieve. For this purpose two kinds of behaviors are implemented:

- **Heuristic wander**
- **Adaptive wander behavior**

Searching environment by considering coverage is not a new idea. There are graph based approaches generating optimal solutions. But these algorithms generate sub-optimal solutions in dynamic environments. Advantages of adaptive wander behavior with respect to existing methods are time-varying map usage for dynamic environment assumption and computational simplicity.

It is assumed that the probability of occurrence of a task in any point of terrain is equal. Due to the equality, robot may go to locations previously wandered if enough time is elapsed. Adaptive wander behavior makes fusion of time varying coverage map, and obstacle map to generate next wander point. This point generated by this behavior is optimal or near optimal because

- It is guaranteed that the next wander point is not visited previously within coverage map timeout
- Path to the next wander point contains fewer obstacles, and then the robot will undergo less obstacle avoidance. As a result of this, time, and energy is saved.

Time varying maps are utilized in all maps used in adaptive wander behavior implementation. The reasons behind this are to increase coverage, and adaptivity to dynamic environment. Coverage is a good measure of wander behaviors. The performance of two wander behaviors is tested in terms of coverage by changing the parameters of the wander behaviors.

The performance of adaptive wander algorithm is tested by comparing the coverage times of heuristic wander behavior. It is shown that adaptive wander behavior is 50 percent better than heuristic one. The value of map timeout is critical in terms of memory usage. As coverage timeout, $TO_{\text{cov}}$, is increased, the coverage of the entire terrain decreases exponentially. But this increase brings considerable memory requirements. Simulation results show that there is an optimum memory requirement depending on the terrain dimensions. Analysis of memory requirements for steady state, and 95% coverage gives that there is an optimum $TO_{\text{cov}}$ in terms of memory usage. This is because for a given terrain dimensions, large $TO_{\text{cov}}$ is not necessary. For terrain size of 100x100 m2, 400 seconds $TO_{\text{cov}}$ is enough.

There are many side effects in implementing two kinds of wander behaviors working together. In obstacle dense environments, adaptive wander behavior fails to generate the next wander point, and then heuristic wander takes the wander control. Since heuristic wander is more dynamic, it enhances the obstacle avoidance process. Results show that implementation of wander behaviors are satisfactory. It is not observed that a robot get stuck in the terrain. Adaptive wander behavior increases coverage time considerably. Moreover memory usage can be optimized using the simulator. Moreover, these behaviors help obstacle avoidance behavior indirectly.

Many experiments are done about **target reaching time** for various environmental conditions, team and task sizes. The main results of these simulations are that:

- It is extremely difficult to model a multi-robot system. There are many parameters to be considered for appropriate model.

- In general, optimum operating points can be found by adjusting parameters such as team size, main, sensorial range, and communication range.

- If there is prior knowledge about the environment, different parameters should be optimized. For instance, if approximate task density is known, energy consumption of the robot team can be optimized. Number of robots optimizing target reaching time, and energy consumption can be found using the simulator.

- Simulation can decrease unnecessary resource usage, i.e. if 20 meters communication range is enough, 30 meter communication range will increase cost considerably.

**Communication cost** is one of the major constraints in the system. In this thesis, it is aimed to model the optimal communication range with respect to team size, task density, and main sensorial range. Many experiments are done to optimize

target reaching time with respect to communication range with the different parameters of are mentioned above. Simulation results are fed to a 4 layered neural network to train it. This neural network is asked to compute an optimum communication range for input data not uses for training. Even if the neural network estimated the correct shape, the error between estimated and simulated result is high for the time being.

## 7.3 Task Allocation and Description

Task allocation is the central issue in multi-robot systems. It affects performance of the team directly. In this thesis, market based auction algorithm is implemented. Task allocation algorithm is almost the same with MURDOCH described in [51], except for fitness calculations.

Tasks are allocated to robots using their fitness values. In MURDOCH, fitness of robots is determined by the relative distance to the task location. In this work, there is a different method is developed calculating fitness. The main aims of the new fitness functions are to obtain:

- A compact description of fitness of robot without dealing with the reasons
- Fair task allocation
- Fault-tolerant task allocation

There are two types of non-linear fitness functions, one for the success conditions, and one for the failure conditions. Fitness functions are non-linear, and time dependent functions. By using this non-linear fitness functions, following benefits are obtained

- Rate of increase with respect to fitness function parameters is determined
- Upper bound for fitness values is specified
- Time effect is inserted. Fitness is not absolute, it is time-dependent. As time goes on fitness function decreases.

Fitness are exponential functions, and more linear for small fitness parameters. As parameters increase, fitness function goes to more non-linear region. Meaning of this is that if a robot reaches 100 targets, and another has reached 105 targets, fitness of these robots will differ slightly.

Fair task allocation is achieved with success fitness functions: target reaching frequency, obstacle avoidance success frequency, coverage. Fault tolerant task allocation is achieved using failure fitness functions: communication failure frequency, obstacle avoidance failure frequency. In addition to these an instantaneous distance fitness function is also used to measure the fitness relative to task location.

In 6.5, effect of fitness functions on task allocation is analyzed. Results are quite satisfactory. Task allocation fairness is higher than the results for auction based task allocation algorithm given in [48], [51]. Three times fair task allocation is achieved using introduced fitness functions. Fairness of task allocation is achieved with coverage, and obstacle avoidance success frequency fitness functions.

Fault tolerance analysis is also satisfactory. To test the fault tolerance analysis, three types of error situations are simulated:

- Communication errors
- Obstacle avoidance error
- Errors in the robots physical structure for instance faults in motors.

Task is not allocated to a robot having faults. Robot fitness decreases dramatically in case of fault but if any other fault has not occurred, then fitness of the robot increases exponentially with a specified time constant so that the robot can again be allocated and execute tasks.

These fitness functions cover compact descriptions of success and failure conditions. For instance, a robot can determine the obstacle avoidance error, but it may not know the reasons of error. Fitness functions are intended to analyze the

results rather than reasons. Another beneficial side effect is the communication demand reduction in case of errors.

Method of **task description** used in this simulator is also satisfactory. User can define 4 different types of tasks. By nesting these 4 types of tasks, any task can be obtained. But even if four types of task description exist in the simulator, nesting is not implemented for the time being.

## 7.4 Future Works

All results are obtained using the simulator implemented. Physical robot implementation is a meaningful and necessary future work. Proposed behavioral control architecture should be tested with embodied robots. This is the final validation of this thesis. Benefits of the simulator should not be forgotten.

Simulation environment can be developed further. Models of the sensors and the environment can be enhanced to obtain more realistic simulations. Currently, there is no physical model of the robot dynamics in the simulator. Robots are simply assumed to be box shaped with perfect kinematics. Dynamical model of robots with correct physical structure can be integrated with the simulation environment.

Motion control of the robot is achieved with potential fields. Magnitudes of different force parameters are found using the simulator for safe operations. But these values can be optimized further, or they can be updated in real time adaptively. Fitness functions can also be improved in terms of convergence time to fair task allocation. New paradigms can be developed in task allocation issues to decrease the communication demand. Currently, auction based algorithms need high communication resources.

Understanding the nature of intelligence as an ultimate goal of the multi-robot system researches will never end. Many hot topics can be tackled in the future: If

robots do not perfectly fit to tasks that have to be achieved, relevance measures need to be evaluated besides fitness such that fused relevance of many robots will make that group fit for a task. Information need to be fused at different levels of resolution. Security issues can be investigated like sensing enemy, surveillance.

# REFERENCES


[1] Cao, Fukunaga, and Kahng , “Cooperative Mobile Robotics: Antecedents and Directions” *Autonomous Robots* 4(1): 7-27, 1997.

[2] I.F. Akyildiz, W. Su*, Y. Sankarasubramaniam, E. Cayirci, “Wireless sensor networks: a survey”, Computer Networks 38 393–422, 2002

[3] N. Xu, “A Survey of Sensor Network Applications”, 2003

[4] T. Imielinski, S. Goel, “DataSpace: querying and monitoring deeply networked collections in physical space”, ACM International Workshop on Data Engineering for Wireless and Mobile Access MobiDE 1999, Seattle, Washington, 1999, pp. 44–51.

[5] http://www.alertsystems.org

[6] C. Intanagonwiwat, R. Govindan, D. Estrin,” Directed diffusion: a scalable and robust communication paradigm for sensor networks”, Proceedings of the ACM Mobi-Com’00, Boston, MA, 2000, pp. 56–67.

[7] S. Meguerdichian, F. Koushanfar, G. Qu, M. Potkonjak,” Exposure in wireless ad-hoc sensor networks”, Proceedings of ACM MobiCom’01, Rome, Italy, 2001, pp. 139–150.

[8] T. Arai, E. Pagello, and L. E. Parker “Guest Editorial: Advances in Multi Robot Systems”, *IEEE Transactions on Robotics and Automation* 18(5): 655-661, 2002

[9] Parker, L. E., Current State of the Art in Distributed Robot Systems, *Distributed Autonomous Robotic Systems 4,* Lynne E. Parker, George Bekey, and Jacob Barhen (eds.), Springer, 2000: 3-12.

[10] R. Arkin, “Behavior Based Robotics”, The MIT Press, 1998.

[11] Brooks, R. A., "A Robust Layered Control System for A Mobile Robot”, IEEE Journal of Robotics and Automation, Vol. 2, 1986

[12] Brooks, R. A., "Planning Is Just A Way Of Avoiding Figuring Out What To Do Next", MIT AI Lab Working Paper 303, September 1987.

[13] R. Arkin, "Motor Schema-Based Mobile Robot Navigation"Internationa Journal of Robotis Reasearch, 1989

[14] Mataric, M. J., "Situated Robotics", Invited contribution to the Encyclopedia of Cognitive Science, Nature Publishing Group, Macmillan Reference Limited, Nov 2002

[15] Monica N. Nicolescu and Maja J. Mataric, "A Hierarchical Architecture for Behavior Based Robots" In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems Bologna, ITALY, July 15-19, 2002

[16] R. P. Bonasso, R. J. Firby, E. Gat, D. K. D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive systems. Journal of Experimental and Theoretical Artificial Intelligence, 9(2–3):237–256, 1997.

[17] R. Arkin, Reactive robotic systems, The handbook of brain theory and neural networks 793 – 796, 1998

[18] E. Gat., "On three-layer architectures", Artificial Intelligence and Mobile Robotics, pages 195–210. AAAI Press, 1998.

[19] R. Arkin and Tucker Balch, "AuRA: Principles and Practice in Review", *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3), pages 175-189, April 1997.

[20] Arkin, R.C. and Balch, T Cooperative Multiagent Robotic Systems., in *Artificial* Intelligence and Mobile Robots *, D. Kortenkamp, R.P. Bonasso, and R. Murphy (eds), MIT* Press, 1998.

[21] Fukuda, T., Nakagawa, S., Kawauchi, Y., and Buss, M.,. Structure Decision for Self Organizing Robots Based on Cell Structures - CEBOT. IEEE International Conference on Robotics and Automation, Scottsdale Arizona, 1989.

[22] http://www.mein.nagoya-u.ac.jp/activity/2001_e/ads.html

[23] L. E. Parker, "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation", IEEE Transactions on Robotics and Automation, Vol. 14, No. 2, April 1998

[24] L. E. Parker. "Heterogeneous Multi-Robot Cooperation" PhD thesis, MIT EECS Dept., 1994.

[25] L. E. Parker, "L-ALLIANCE: Task-Oriented Multi-Robot Learning in Behavior-Based Systems, Advanced Robotic"s, Special Issue on Selected Papers from IROS '96, 11 (4) 1997: 305-322

[26] L. E. Parker, "Lecture Notes on Distributed Intelligence in Autonomous Robotics", University of Tennessee, Department of Computer Science, 2003.

[27] Holland, Melhuish "Stigmergy, Self-Organization, and Sorting in Collective Robotics", by, *Artificial Life* 5: 173-202, 1999.

[28] V. Kumar, F. Sahin "Foraging in Ant Colonies applied to the Mine Detection Problem", IEEE International Workshop on Soft Computing in Industrial Applications, June 23-25 2003

[29] K. L. Doty, R. E. Van Aken. Swarm robot materials handling paradigm for a manufacturing workcell. In IEEE ICRA, volume 1, pages 778–782, 1993.

[30] M. Mataric. Interaction and Intelligent Behavior. PhD thesis, MIT, EECS, May 1994.

[31] J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario. "Selforganizing collection and transport of objects in unpredictable environments". Symposium on Flexible Automation, 1990.

[32] D. McFarland. "Towards robot cooperation." Proceedings of the Third International Conference on Simulation of Adaptive Behavior, pages 440–444. MIT Press, 1994.

[33] T. Haynes and S. Sen. "Evolving behavioral strategies in predators and prey" In Gerard Weiss and Sandip Sen, editors, Adaptation and Learning in Multi-Agent Systems, pages 113–126. Springer, 1986.

[34] P. Maes & R. A. Brooks "Learning to Coordinate Behaviors" , National Conference on Artificial Intelligence, 1990

[35] L. P. Kaelbling, M. L. Littman, A W. Moore "Reinforcement Learning: A Survey" Journal of Artificial Intelligence Research, 1996.

[36] W. D. Smart, L Pkaelbling, "Effective Reinforcement Learning for Mobile Robots" Journal of Artificial Intelligence Research, 2002

[37] K. H. Lowy, W. K. Leowy, M H. Ang, Jr.z "Action Selection for Single- and Multi-Robot Tasks Using Cooperative Extended Kohonen Maps" *Proc. 18th IJCAI'03*, pages 1505-6, Aug 9-15, 2003,

[38] C. Versino, L. M. Gambardella, "Learning Fine Motion by Using the Hierarchical Extended Kohonen Map", Proceedings ICONIP96, International Conference on Neural Information Processing, 1996

[39] Pomerleau, D.A., "Efficient Training of Artificial Neural Networks for Autonomous Navigation", Neural Network Perception for Mobile Robot Guidance, Kluwer Academic Publishers 1993

[40] Wyeth G., Neural Mechanisms for Training Autonomous Robots, 1997

[41] D. Floreano and F. Mondada. "Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot." Proceedings of the Third International Conference on Simulation of Adaptive Behavior, pages 421--430, Cambridge, MA, 1994.

[42] K. R. Baghaei, A. Agah. "Task allocation and communication methodologies for multi-robot systems" Autosoft – Intelligent Automation and Soft Computing Journal, Vol. 9, No. 4, 2003

[43] B. P. Gerkey, M. J. Mataric, "A Formal Analysis and Taxonomy of Task Allocatiom in Multi-Robot System", Intelligent Journal of Robotic Research , July 2003.

[44] B. P. Gerkey, M. J. Mataric, "On Multi-Robot Task Allocation, PhD thesis, Center for Robotics and Embedded Systems", University of Southern California, Los Angeles 2003

[45] B. P. Gerkey, M. J. Mataric´, "A Formal Framework for the Study of Task Allocation in Multi-Robot Systems", International Journal of Robotics Research, 23(9), Sep 2004, 939-954.

[46] B. P. Gerkey, M. J. Mataric´, “Multi-Robot Task Allocation: Analyzing the Complexity and Optimality of Key Architectures”. IEEE ICRA 2003

[47] L. E. Parker, “Evaluating Success in Autonomous Multi-Robot Teams: Experiences from ALLIANCE Architecture Implementations”, Journal of Theoretical and Experimental Artificial Intelligence, 13, 2001

[48] Gage, A., and Murphy, R.R., "Affective Recruitment of Distributed Heterogeneous Agents" Proceedings of the Nineteenth National Conference on Artificial Intelligence, San Jose, CA, July 25-29, 2004, pp. 14-19.

[49] Smith, “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", *IEEE Transactions on Computers* , 1980.

[50] R. Davis and R. G. Smith, “Negotiation as a metaphor for distributed problem solving,” Artificial Intelligence, vol. 20, no. 1, pp. 63–109, 1983.

[51]B. P. Gerkey, M. J. Mataric “Sold! Auction Methods for Multi-robot Coordination”, IEEE Transactions on Robotics and Automation, 18(5): 758-768, October 2002.

[52] S. Botelho and R. Alami, “M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement,” in Proc. IEEE ICRA, May 1999.

[53] A. Cai, T. Fukuda, A. Arai, K. Yamada, and S. Matsumura, “Path planning and environment understanding based on distributed sensing in distributed autonomous robotic system,” in 4th Int. Workshop on Advanced Motion Control Proceedings (AMC ’96-MIE), vol. 2, 1996,

[54] B. P. Gerkey and M. J. Mataric, “Pusher-watcher: an approach to fault-tolerant tightly-coupled robot coordination,” in Proc. IEEE ICRA, 2002

[55] B. B. Werger and M. Mataric, “Broadcast of local eligibility for multi-target observation,” in Distributed Autonomous Robotic Systems 4, L. E. Parker, G. Bekey, and J. Barhen, Eds. Springer-Verlag, 2000, pp. 347–356.

[56] K.H Low, W. K. Leow, M. H. Ang, “Task Allocation via Self-Organizing Swarm collations in Distributed Mobile Sensor Network” In Proc. 19th AAAI-04, pp. 28-33, Jul 25-29, 2004.

[57] L. Chaimowicz, M. F. M. Campos, V. Kumar, “Dynamic Role Assignment for Cooperative Robots”, IEEE ICRA 2002

[58] Hwang, Y. Ahuja, N. “Gross Motion Planning: A Survey” ACM Computing Surveys, 24(3)., 1992

[59] Latombe, J. C., “Robot Motion Planning” Kluwer Academic Publishers, Boston, MA, 1991

[60] D. O. Popa, C. Helm, H. E. Stephanou, A. C. Sanderson, “Robotic Deployment of Sensor Networks Using Potential Fields”, Proceedings of the 2004 IEEE International Conference on Robotics & Automation , April 2004.

[61] B. H. Krogh, “A generalized potential field approach to obstacle avoidance control,” in Robotics Research: The Next Five Years and Beyond, Society of Manufacturing Engineers, 1984.

[62] G. A. S. Pereira, M. B. Soares, M. F. M. Campos, “A potential field approach for collecting data from sensor networks using mobile robots”, Proceedings of the IEEE/RJS International Conference on Intelligent Robots and Systems, 2004.

[63 ]Andrew Howard and Maja J Mataric´, "Mobile Sensor Network Deployment Using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem", Proceedings, 6th International Symposium on Distributed Autonomous Robotic Systems *(DARS)*, Japan, 2002

[64] S. S. Ge and Y. J. Cui, “New Potential Functions for Mobile Robot Path Planning”, IEEE Transactions on Robotics and Automation, Vol. 16, NO. 5, October 2000

[65] Horner, D. P., Healey, A. J., "Use of Artificial Potential Fields for UAV Guidance and Optimization of WLAN Communications", Proceedings of the IEEE AUV2004 Conference, Maine, June 2004.

[66] Gage, D.W. "Command Control for Many-Robot Systems", AUVS-92, the Nineteenth Annual AUVS Technical Symposium, Huntsville AL, 22-24 June 1992.

[67] Choset, H., "Coverage for robotics - A survey of recent results", Annals of Mathematics and Artificial Intelligence 2001.

[68] Balch, T. "The case for randomized search", Work Shop in Sensor and Motion, IEEE, ICRA, 2000

[69] B. Yamauchi, "Frontier-based approach for autonomous exploration," in In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation, 1997, pp. 146–151.

[70] A. Zelinsky, "A mobile robot exploration algorithm," in IEEE Transactions on Robotics and Automation, vol. 8, 1992

[71] Gage, D. W., "Randomized search strategies with imperfect sensors", SPIE VIII, 1993

[72] Matalin, A. M., Sukhatme, G. S., "The Analysis of an Efficient Algorithm for Robot Coverage and Exploration based on Sensor Network Deployment" Proceedings of the 2005 IEEE ICRA, 2005

[73] Sameera, P. and Gaurav, S., "Constrained Coverage for Mobile Sensor Networks", IEEE ICRA, 2004

[74] J. H. Reif and H. Wang, ."Social potential fields: A distributed behavioral control for autonomous robots",. Robotics and Autonomous Systems, vol. 27, pp. 171.194, 1999.

[75] T. Balch and M. Hybinette, ."Social potentials for scalable multi-robot formations" IEEE ICRA, 2000.