

A Genetically Evolved Neural Network for an Action Selection Mechanism in Behavior-based Systems

THIS THESIS IS PRESENTED TO THE
SCHOOL OF ELECTRICAL, ELECTRONIC & COMPUTER ENGINEERING
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
OF
THE UNIVERSITY OF WESTERN AUSTRALIA



By
Saufiah Abdul Rahim
May 2016

Abstract

The world of robotics has grown so much that it has reached a state where it can be trusted with many real-world applications, especially those that involve a high safety risk for human effort. From its ‘humble’ beginnings operating in a static and controlled environment, robot platforms are now required to operate in dynamic and unknown environments, which traditionally require human intelligence for real-time decision making. Development of a control system for a robot platform to handle such scenario can be very demanding. The system requires complex decision making capabilities in order to be sufficiently robust and responsive to the dynamics of its environment.

One possible approach is to implement a behavior-based system, which ‘reacts’ to its environment rather than using preprogrammed rules of engagement. However, other than the accuracy of its sensors, the success of a behavior-based system relies largely on its Action Selection Mechanism (ASM) module, which is basically a behavior coordination method. Common implementations of behavior coordination method can be categorised into two: arbitration and command fusion. Consequently, deciding on a suitable coordination method for a particular task in an unknown environment presents a similar complex issue. To handle this, the more popular approach is to use Artificial Intelligence (AI) in the development of ASM modules.

In this thesis, a Genetic Algorithm (GA) has been used to evolve a neural network engine that is used as an ASM module for a behavior-based system. The

proposed control architecture implements a basic GA to train the synaptic weights of a simple Multi-Layered Perceptron (MLP) feed-forward Artificial Neural Network (ANN) in identifying a suitable formulation of ASM. A simple, yet found to be sufficiently adequate, fitness function has been formulated in order to ensure the effectiveness of a GA in evolving the system. The proposed fitness function is defined as such that it can be generalised and applied to any robot control tasks. The proposed system has been tested using simulation software in two common robot mission scenarios involving unknown environments: search and exploration, and target tracking.

Simulation results show that the proposed Genetically Evolved ASM (GEASM) can dynamically manage the behavior coordination method that enables the system to achieve mission objectives in both test scenarios. For the search and exploration mission, the GEASM managed to achieve a 93% success rate compared to other architectures, with the nearest competitor at 67%. As for the target tracking mission, the GEASM achieved a stunning 100% success rate, compared to the next best at 75%. Since the test environment is actually different from the one used in training the proposed system, it can be projected that the GEASM can actually enable a system to perform in an unknown environment with a significantly high probability of success.

Acknowledgements

First , I would like to thank my supervisor, Professor Thomas Bräunl for giving me the opportunity to pursue this research at CIIPS, UWA.

Next, I would like to thank UniMAP and KPT for awarding me a three and a half-year scholarship.

I would also like to thank my friends - Haniza Yazid, Norazian Abdul Razak and others, who have helped me in many ways.

An unlimited thanks to the whole family who encourage me, and giving me a continuous support until the completion of the thesis. To my parents, this thesis is specially dedicated for you. Thanks for your bless and doa. "Ayah dan ibu, itu lah permulaan kami..".

Warmest and most enduring thanks to my beloved husband, for being a great motivator, and an excellent mentor. Thank you so much for guiding me and sharing me some of your knowledge. A special dedication also go to my beloved children who make my life more fun, more memorable, and more worthwhile. I will always love all of you.

List of Abbreviation

AI	Artificial Intelligence
AHC	Adaptive Heuristic Critic
APOC	Activating-Processing-Observing-Components
ASM	Action Selection Mechanism
BEP	Back Error Propagation
CoSyNE	Cooperative Synapse Neuroevolution
DAMN	Distributed Architecture for Mobile Navigation
EA	Evolutionary Algorithm
ER	Evolutionary Robotic
FDES	Fuzzy Discrete Event Systems
FSA	Finite State Acceptor
FSM	Finite State Machine
GA	Genetic Algorithm
GEASM	Genetically Evolved Action Selection Mechanism
GPS	Global Positioning System
iB2C	integrated Behavior-Based Control
IR	Infrared
MLP	Multi Layered Perceptron
MSE	Mean Squared Error
PB	Priority-based
PSO	Particle Swarm Optimization
RL	Reinforcement Learning

SNN	Spiking Neural Network
SR	Stimulus-Response
TS	Time Step
VS	Vector Summation
WF	Wall Following

Contents

Abstract	ii
Acknowledgements	iv
List of Abbreviation	v
1 Overview of the Thesis	1
1.1 Introduction	1
1.2 Project Overview	5
1.3 Thesis Outline	6
2 Related Work	8
2.1 Robot Control Philosophy: An Overview	8
2.2 Behavior-based Control System	13
2.2.1 Principles of a Behavior-based System	13
2.2.2 Action Selection Mechanism	18
2.2.3 Emergent Behavior	29
2.3 AI Techniques for Robot Control	31
2.3.1 Evolutionary Robotics: An Overview	32
2.3.2 Combining Neural Network with Genetic Algorithm	34

2.3.3	Trends of Application in Evolutionary Robotic	37
2.3.4	Evolutionary Approach in Behavior-based System	40
2.4	Summary	43
3	Genetically Evolved Action Selection Mechanism (GEASM)	44
3.1	Overall View	45
3.2	Behavior-based Control	46
3.3	Core Design	49
3.3.1	Multi-Layered Perceptron	49
3.3.2	Genetic Algorithms	53
3.3.3	Using GA to Evolve a Neural Network	57
3.4	Summary	59
4	Implementation of Robot Control	61
4.1	Experiment Platform	62
4.1.1	Simulation Platform	63
4.1.2	Robot and Sensor Model	64
4.1.3	Pre-processing of the Sensory Data	67
4.1.4	Simulation Environment	69
4.2	From Theory to Implementation	71
4.3	Justification of Selected Design Parameters	76
4.3.1	Size of Hidden Layer	76
4.3.2	Error Term Contribution	79
4.3.3	Weighting Fitness Components	80
4.4	Considerations for Performance Comparison	83

4.5	Summary	87
5	Results and Discussions	88
5.1	Search and Exploration	89
5.1.1	Performance on Exploration Coverage	93
5.1.2	Performance of Search and Exploration Mission	98
5.2	Target Tracking	107
5.2.1	Performance of Classical ASM Approaches	108
5.2.2	Performance of Evolutionary Neural Network Approaches	114
5.3	Performance of the GEASM System under Image Noise	117
5.4	Re-utilising the GEASM System	124
5.5	Discussion	130
5.6	Summary	134
6	Summary and Future Work	135
6.1	Thesis Summary	135
6.2	Key Findings	137
6.3	Limitation and Future Work	139
	References	142
	Appendices	160
A	Initial Robot Position and Direction	161
B	Best Path Traces for Conventional ER Approach	164
C	Worst Exploration Coverage	166

D Environmental Setup for Search and Exploration Mission	169
E Wall Following Fails in Search and Exploration Mission	176
F Comparison of Robot Control Architectures	178

List of Tables

2.1	A brief review on action selection mechanism as classified by Pirjanian [9]	19
4.1	Examples of data extraction from the camera image	70
4.2	Initial parameter settings for optimising the weights of a neural network	75
5.1	Performance comparison of exploration coverage in two different environments	94
5.2	Area ratio of testing environment over training environment	100
5.3	Time steps to complete a search and exploration task under salt-and-pepper noise	119
5.4	Summary on percentage of success in the execution of robot missions	131

List of Figures

2.1	Block diagram of four basic robot control architectures	11
2.2	Basic architecture of a behavior-based system	14
2.3	A stimulus-response diagram, redrawn after Arkin [1]	14
2.4	A stimulus-response diagram for target tracking task	15
2.5	Class of action selection mechanism as categorised by Pirjanian [9] .	20
2.6	An SR diagram of a simple behavior-based system, employing Sub- sumption Architecture	21
2.7	An FSA for a door traversal task, copied from Pirjanian [9]	22
2.8	Behavior voting in DAMN, copied from [49]	24
2.9	Fuzzy automaton for robot navigation with two fuzzy states, copied from Jayasiri et al. [51]	25
2.10	Basic structure of APOC, copied from Scheutz et al. [20]	27
2.11	Basic structure of IB2C, copied from Proetzsch et al. [21]	28
2.12	Distributed neural network for hexapod robot, copied from Beer et al. [89]	39
2.13	An emergent modular neural network architecture, copied from Nolfi [101]	42
3.1	The proposed control architecture using GEASM	45
3.2	Multi-layered perceptron, redrawn after Haykin [78]	50

3.3	Model of artificial neuron, redrawn after Haykin [78]	51
3.4	Process of crossover	55
3.5	Range of random values	56
3.6	Chromosome representation	58
4.1	Snapshots of a real and its equivalent simulated EyeBot platform	65
4.2	Robot is facing a targeted ball with obstacles around the robot	66
4.3	Pre-processing from camera image	68
4.4	A snapshot of a program to encode the IR readings	71
4.5	Analysis of hidden node for feed-forward neural network	78
4.6	The effect of the error suppressor	80
4.7	Best fitness value when training with different weighting fitness components	81
4.8	Path traces of GEASM when training the system with various weight ratios	82
4.9	Block diagrams of action selection mechanisms	84
4.10	A snapshot of a program for implementing the priority-based method	85
4.11	A snapshot of a program for implementing the vector summation method	86
5.1	The simulation training environment for search and exploration mission	89
5.2	Fitness value during the training phase	90
5.3	Search and exploration in a simple environment	91
5.4	Best path traces for exploration in 10-paths environment	95
5.5	Best path traces for exploration in 10-rooms environment	96
5.6	Comparing worst path traces of the WF system with GEASM in 10-rooms environment	98

5.7	Comparing worst path traces of the WF system with GEASM in 10-paths environment	99
5.8	Number of successes for search and exploration missions in various environments	99
5.9	Search and exploration: Environment 1	102
5.10	Search and exploration: Environment 2	103
5.11	Search and exploration: Environment 3	104
5.12	Execution of GEASM in Environment 2	105
5.13	A scenario where the WF approach is unable to complete a mission	106
5.14	Fitness value during the training phase	108
5.15	Path traces for GEASM, PB, VS, conventional ER: simple environment	109
5.16	Path traces for GEASM, PB and conventional ER: Scenario 1 . . .	109
5.17	Path traces for VS: Scenario 1	110
5.18	Path traces for GEASM, PB, VS and conventional ER: Scenario 2 .	110
5.19	Path traces for GEASM, PB, VS and conventional ER: Scenario 3 .	111
5.20	Active behavior during the execution of the PB system: Scenario 1	111
5.21	Active behavior during the execution of the PB system: Scenario 2	112
5.22	Active behavior during the execution of the PB system: Scenario 3	112
5.23	MSE between the output of behavioral module and the GEASM system: Scenario 1	115
5.24	MSE between the output of behavioral module and the GEASM system: Scenario 2	116
5.25	MSE between the output of behavioral module and the GEASM system: Scenario 3	116
5.26	The interface of error model in the EyeSIM simulator	118

5.27	The maximum percentage of image noise that GEASM can tolerate	119
5.28	Robot's view without noise, with salt-and-pepper noise, and with Gaussian noise	120
5.29	Search and exploration: execution under salt-and-pepper noise. Path traces on the left side are at 25% salt-and-pepper noise. Path traces on the right side are at 40% salt-and-pepper noise. . . .	121
5.30	Target tracking of the GEASM system under 25%, and 40% of salt-and-pepper noise	122
5.31	Target tracking of the GEASM system under 60%, and 90% of Gaussian noise	122
5.32	Search and exploration: execution under 90% of Gaussian noise . . .	123
5.33	Search and exploration performance when changing the target object's criteria	125
5.34	Implementing a foraging system using FSM	126
5.35	Initial environmental setup for a foraging application	128
5.36	All target objects have been successfully placed at the home area . .	128
5.37	Implementing an autonomous battery recharging system using FSM	129
A.1	Three initial positions in 10-paths environment	162
A.2	Three initial positions in 10-rooms environment	163
B.1	Conventional ER approach in 10-paths environment	165
C.1	Worst exploration coverage in 10-paths environment	167
C.2	Worst exploration coverage in 10-rooms environment	168
D.1	Initial position: Environment 1	170
D.2	Initial position: Environment 1 (... continued)	171
D.3	Initial position: Environment 2	172

D.4	Initial position: Environment 2 (... continued)	173
D.5	Initial position: Environment 3	174
D.6	Initial position: Environment 3 (... continued)	175
E.1	Path traces of GEASM and WF in an environment where WF has failed.	177
F.1	A comparison of three robot control architectures	179

Chapter 1

Overview of the Thesis

1.1 Introduction

Advancements in robot control technology incline towards the development of autonomous mobile robot platform with sufficient intelligence so that it may operate in any given dynamic environment even without prior information. In such environments, due to the existence of various potential sources of uncertainties, identifying a function to produce a desired control output for a specific sensory inputs is almost impossible. Instead of pushing the limits of processing elements on robot controllers, an alternative can be found in the biological-inspired systems that has been around since the late 1940s.

In 1953, W. Grey Walter successfully implemented a tortoise-like hardware platform based on the principles that machines with proper feedback control systems exhibits natural behaviors [1, 2]. Walter's tortoise was designed as a mobile platform (i.e. fitted with motor-driven wheels) that is always exploring its environment except when it needs to feed (i.e. recharge). Other than making sure it does not run into obstacles, it reacts on light sensor inputs (i.e. fitted with proximity sensor and a photocell) as its motivation during exploration. Implementing these

simple reactive circuits, Walter’s tortoise managed to exhibit a more complex natural behavior — safely exploring an environment and recharging itself as required.

This early example of the capabilities of reactive systems in generating a moderately complex behavior has spurred many subsequent works that attempt to enhance the system—mainly by implementing a more complex reactive behavior. This led to the transition from simple reactive system design to a more deliberate behavior-based system. With its roots in reactive systems design, it is easy to see why many publications managed to prove that a behavior-based system is very responsive to unknown, dynamic environments [3, 4, 5, 6, 7, 8].

One example of a behavior-based system that has been successfully executed in unknown dynamic environment was presented by Parker through ALLIANCE architecture [4]. Parker has successfully tested her proposed method by implementing it in a team of three robots operating in a hazardous waste cleanup mission. From her experiments, which have been conducted on both simulated and physical robots, it is interesting to note that the team of robots was able to respond to unexpected obstacles, which have been intentionally added into the mission environment. Another example of the responsiveness of a behavior-based system has been presented by Huq et al. in [8]. Their proposed approach was able to achieve a 100% success rate on a robot’s navigation task, which has been executed in the presence of unpredictable obstacles and within unknown environments.

In simple reactive systems, using basic logical arbitration, in allowing which reaction gets control of the system, is sufficient in most cases. With more deliberate behaviors existing in a behavior-based system, the need for a proper mechanism to select a dominant or resulting behavior becomes rather critical in nature. In fact, the success of a behavior-based system implementation relies greatly on this action selection mechanism (ASM) module. The role of ASM is to coordinate actions suggested from each behavior module in order to produce an appropriate system response at any given instance. Several classifications of ASM have been suggested, but in [9], ASM has been classified into two main categories: arbitration and command fusion.

The arbitration category consists of mechanisms that allow the behaviors to compete for control of the system's resources for a period of time [10, 11, 12]. Priority-based (PB) is one example from this category. Depending on a predefined priority for each behavior module, which is usually based on the nature of the input parameter, a reliable control signal is determined from the selected behavior output. However, methods from this category can be considered inappropriate when dealing with multiple reliable control signals from active behaviors. This is obvious because the 'losing' signals will be completely decimated and their information will be lost. Also, in order to implement this method, appropriate priorities should be assigned to each behavior. In many cases, this is rather difficult to do because the priorities may change depending on mission scenario and environment, and some implementations have to use AI methods to overcome this problem [13, 14, 15, 16]. A more detailed insight on this is covered in Chapter 2.

Alternatively, the methods in the command fusion category allow all behaviors to contribute towards producing a control output in a cooperative manner as proposed by Arkin, Rosenblatt, and Riekkki and Ronning in [17], [18], [19] respectively. This method makes an effort to retain all information provided by each behavior. The central issue here is to find a fair representation of information for all contributing behaviors. A simple example for this is to take the average of all the behavior outputs, like vector addition in the Motor Schema [17]. Unfortunately, this can produce a control signal which does not comply with any of the behavior outputs. Furthermore, it can suffer from the local minima problem [18].

Due to the fact that both methods have specific limitations, the logical next step needed in progressing towards a better ASM is to actually come up with a combination of both methods, by focusing on the strength of each method while minimising the limitation. It is imperative that a certain behavior module be given 'control' but this has to be done while considering 'opinions' from other behavior modules. This is actually more complex than it seems. Quite a number of works have provided a platform to implement the two classes of ASM on the same control

architecture. For example, as presented by Scheutz and Andronache, and Proetzsch et al. in [20], and [21], their proposed techniques have implemented ASM methods from both categories, and are able to switch from one method to another. However, using these techniques requires another mechanism to decide which ASM method should be executed. As for the work presented in [8], a fuzzy logic technique has been used to generate activity states for each behavior. Based on the activity states, ASM methods from both command fusion and behavior arbitration categories will be applied in order to produce an effective control signal. However, as reported, this technique suffers from handling a large number of variables and tuning parameters in order to achieve a successful operation of the system.

The trend in implementing an integrated (combination of arbitration and command fusion) solution clearly shows that using AI in obtaining optimal performance for ASM has become the most important part, given the increasing complexity of a system's operating environment. An ASM that can dynamically adapt to its environment would be the best solution for a behavior-based system to perform in an unknown environment. A study needs to be done on suitable AI methods in order to find the ideal candidate in generating such dynamically-adaptive ASM. This should be hugely beneficial for mobile robot applications that intend to implement behavior-based control systems.

Evolutionary robotics (ER) is one of the most popular research fields that have employed AI techniques (i.e. evolutionary algorithms) in modelling a robot control system. It is usually implemented as a black box, where the function of the control system will be automatically adapted from the input information of the environment by extracting the data stream accessible from the robot. This technique provides a mechanism for a robot to move and react to its environment and at the same time will adjust its behavior according to real situations. In addition, unknown nonlinearities will be taken into account during the adaptation process. This probably being the main contribution towards several success works of ER approach in executing their robot control system under uncertainties.

Quite a number of works in ER have proven that their techniques can successfully adapt to various unknown, changing environments [22, 23, 24]. For instance, Knudson and Tumer in [22] have implemented a neuro-evolutionary approach for adaptive navigation of a mobile robot. In their research, an evolutionary algorithm is used to configure the parameter space of a neural network to find the path qualities for a robot to travel in an environment. Based on the selected configuration, the neural network is run for several potential paths from the robot's surrounding in order to find the greatest path quality. From the experiments conducted, a successful navigation has been achieved in various changing environments. In fact, the same control system has been successfully transferred from simulations to a real robot without any modification.

Since an ER approach offers automated design to develop a robot control system with adaptive capability, the use of this technique to ease the problem in finding an adaptive ASM for a specific robot application could be a great option. Furthermore, as stated in [25] a behavior-based system and an ER approach share many common characteristics. With this fact, the idea of employing the two techniques on the same platform could be possible, perhaps with a high chance of success in enhancing the capability of a behavior-based system to be executed in various kind of environmental conditions.

1.2 Project Overview

The purpose of this research is to investigate the use of Artificial Intelligence (AI) techniques in order to generate a suitable ASM based on a system's mission scenario. With an assumption that at every decision making cycle, when a suitable coordination method has been applied to a behavior-based system with appropriate parameters, rapid interactions of its behavior modules with the environment can be fully utilised. This may contribute towards a successful execution of a behavior-based system for achieving the overall system's objective even if it is executed in the existence of uncertainties.

This research presents the Genetically Evolved Action Selection Mechanism (GEASM), as an alternative method to overcome the problem of formulating an effective ASM in a behavior-based system. The main idea is to use a dynamic and adaptive model in the process of finding the appropriate ASM. In this case, a Genetic Algorithm (GA) has been used to provide the capability in traversing the search space for the optimum solution. The search space is provided by the neural network engine in the form of an array of synaptic weights. This solution focuses on constructing low-level components of the behavior-based systems, which usually feature rapid interaction between a robot platform and its environment. In the implementation, the GEASM will produce a low level action (e.g. turn left, turn right, go forward) that can be used to control the robot in accomplishing its mission.

The proposed method is a generalised approach that can be implemented for various robot applications. In this thesis, the feasibility of the method has been tested through two main test applications: search and exploration mission, and target tracking task. The performance of the proposed method is investigated in various experiments. This includes the capability of the system to deal with uncertainties from various sources. Moreover, the capability of the GEASM system to be utilised for a higher level of robot applications through a simple arbitration type of ASM will also be explored.

1.3 Thesis Outline

This thesis is organised as follows. In Chapter 2, research in areas related to this work is reviewed. Two main areas are covered. First, a general introduction to behavior-based systems is presented. Second, various aspects of implementing AI techniques for various robot applications are discussed. Focus of the discussion is given to the works that have used Neural Network, and GA techniques. The theory of the GEASM architecture, including some of the fundamental design choices is described in Chapter 3. Chapter 4 reviews the full implementation of the system for

search and exploration tasks and target tracking missions. Experimental platforms, environments and mission scenarios are also detailed in this chapter. Chapter 5 presents the detailed experimental descriptions, results and discussion. Chapter 6 concludes with a summary of the approach and discussion of results, including the contributions of this work and suggested future research directions.

Chapter 2

Related Work

There are two main areas of robotics research that are relevant to this thesis: behavior-based systems, and evolutionary robotics. Therefore, this chapter intends to establish a comprehensive foundation in these areas, and to review all important related work of the two fields. This includes an overview of robot control philosophy, a discussion on the fundamentals of a behavior-based system, and a survey on research works that have employed artificial intelligence (AI) techniques in the development of their robot control system.

2.1 Robot Control Philosophy: An Overview

Generally, a robot control can be defined as a process of mapping the robot sensory information to produce an appropriate action in an environment. In [26], the control philosophies have been categorised into four basic classes: deliberative, reactive, hybrid and behavior-based. The obvious differences among the approaches can be seen not only at the the system structure but also by the amount of computation performed and the degree of real-time responsiveness upon changes in the world [27].

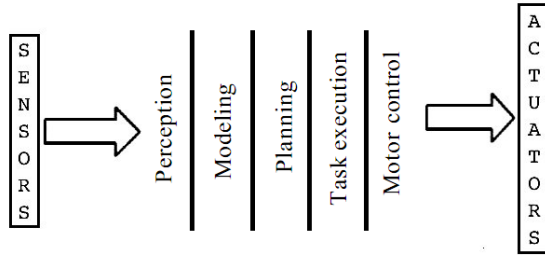
Using a deliberative approach, the Sense-Plan-Act paradigm [1, 28] is applied. This method is computationally intensive due to the use of explicit reasoning. It requires a search of possible state-action sequences and their outcomes. It also relies on a centralised world model and usually uses symbolic representation where the planning for actions is made based on this model. As shown in Figure 2.1(a), information from sensors will be filtered through several intermediate stages of interpretation before it can be used to control the actuators. The method has been criticised as it is obviously not capable of coping with changing environments and uncertainty [3]. This limitation will lead to a ‘qualification problem’ where a continual planning process is required [1]. To illustrate this issue, let’s say a robot has been programmed to deliver a medicine to a patient using a deliberative approach. Using this approach, the robot would have the hospital’s map to plan all possible paths from its current position to the desired patient’s room. Since the world representation of the hospital is available to the control system, the robot is able to search the shortest path to its destination.

However, working in a busy hospital environment is not an easy task. What would happen if on its way to the room using the shortest path, the route is closed due to some construction works in the hospital? In this case, the robot control needs to update its map, re-plan, and search for a new path to the desired destination. Developing a deliberative robot control system for a large hospital will definitely involve a large number of possible states in order to plan the execution of the robot’s task. Clearly, the control system may require a very long time to browse the large state spaces before generating an appropriate action based on its current situation. This is one of the main reasons why the deliberative approach is not able to make a fast decision when dealing with changing environment and uncertainty. For instance, if the robot needs to avoid an unexpected danger, the robot may risk collisions before the planning process has completed. Therefore, the deliberative method is usually applied to control a robot for a very specific task that operates in a structured and highly predictive environment. As stated

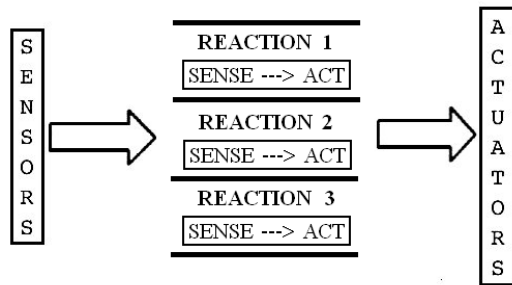
in [2], a robot surgery is one good example of an application that is suitable to employ a deliberative approach. Using this method, not only can the perfect plan be calculated for the robot to follow (e.g. drilling the patient's skull at a specific position), but the environment can be kept perfectly static in order to precisely execute the robotic task.

Reactive systems, on the other hand, have an extremely different approach from the deliberative method. They provide a tight coupling between sensing and action by allowing rapid real-time responses. Usually, a reactive system is constructed using several basic behaviors (refer Figure 2.1(b)). Each of the behaviors will be implemented using a simple rule-based method involving a minimal amount of computation which consists of a simple sensors-to-effectors pair [1]. As stated in [2], to design a reactive system, all possible states should be uniquely coupled to appropriate actions. This offers the system a great run-time efficiency. However, a reactive system has usually exhibit an unsatisfactory performance in complex environments and tasks. Due to lack of memory, the system is less effective in integrating world knowledge and unable at learning over time [29, 30]. Moreover, a reactive system has a limited representational capability. Even though the system is suited to dynamic and unstructured worlds, it cannot rely on a world model for more complex reasoning processes.

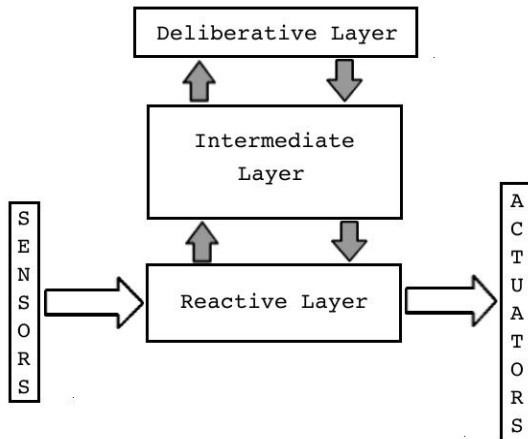
This limitation can be further described using a similar delivery robot task executed in a hospital environment as presented earlier. In this case, the reactive approach is employed for a safe navigation system for the delivery robot. Let's say, two behaviors have been used to develop the reactive system. One is to make sure the robot is moving forward smoothly in the environment. The other one is turning away from a detected obstacle. More behavior can be added to the reactive system to fulfil the other system's requirement. However, the two behaviors are sufficient enough to allow the robot to move around the hospital safely. Clearly, by using the two behaviors, the reactive system is capable of fast response to any unexpected obstacles. Reaching a destination using this safe navigation system



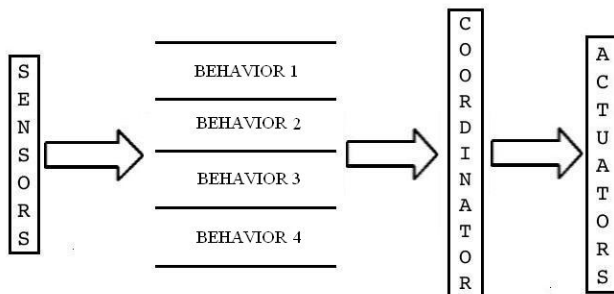
(a) Deliberative, redrawn after Brooks [28]



(b) Reactive, redrawn after Mataric [2]



(c) Hybrid system, redrawn after Mataric [2]



(d) Behavior-based system, redrawn after Mataric [26]

Figure 2.1: Block diagram of four basic robot control architectures

is possible when it is executed in a small unknown environment. However, the possibility to efficiently reach the target destination within a range of time gets lower when the system is executed in a large hospital environment. Since a reactive system does not rely on a world model, it is not possible for the robot to find the shortest path to deliver the medicine to the specific patient's room. In fact, in this large environment, there is a possibility that the robot is not able to find the target destination at all.

Hybrid systems attempt to compromise between the two approaches. They employ a reactive system for a low-level control to deal with a robot's immediate need (fast time-scale). They also have a planner (deliberative) for high-level decision making that operates at a longer time-scale. In this case, using this approach, the delivery robot is able to manipulate the world representation in planning the shortest path to its desired destination (i.e. deliberative system). At the same time, it is able to avoid any unexpected obstacles (i.e. reactive system) during the execution of its task. However, as the two systems have distinct time-scales, an intermediate module is essential to allow the deliberative component to communicate with the reactive component (Figure 2.1(c)). This is the most challenging part of implementing the hybrid system where the two control systems need to interact with each other without creating any conflicts at the output. Mataric [2] has discussed a number of methods to manage the interaction of the three layers in a hybrid control system. However, as mentioned in the text, the intermediate layer is hard to design and implement. In most cases, the layer will be designed for a very specific-purpose, and a new design of this layer will be required when implementing the control system on a different robot or task. Moreover, due to the difficulties of integrating the layers, the hybrid system has been widely implemented in single robot control domain.

Behavior-based systems are an alternative to hybrid systems as they may include the deliberative and reactive components in their architecture. Unlike the

hybrid control structure, behavior-based systems are composed from a set of independent modular components called behaviors that are executed in parallel (as shown in Figure 2.1(d)). The parallel structure of the behaviors provides a fast real-time response to the control system. This structure and the capability to maintain real time couplings between sensing and action always lead to a misconception between behavior-based systems and reactive systems. However, unlike in a reactive system (where the development of each behavioral unit is based on sense-act pairs), a behavior-based system may use representations in developing its behavioral unit. This allows the behavior-based system the ability to store the system state in a distributed fashion. For this reason, behavior-based systems can be more powerful than the reactive systems as they can have representation that enables reasoning, planning and learning in their control structure. Behavior-based control is best suited for a system that is situated in an environment with significant dynamic changes. In this kind of environment, fast response and adaptation are crucial. This is one of the reason why behavior-based systems are widely utilised, even for multi-robot control applications [31, 32, 33, 34].

2.2 Behavior-based Control System

This section will discuss in more detail the theory of a behavior-based system.

2.2.1 Principles of a Behavior-based System

Behavior-based control was originally developed for robots that operate in a dynamic, unpredictable environment without using an abstract representation [26, 35]. In principle, a behavior-based control works through the integration of a set of interacting behaviors in order to achieve a desired system objective. Figure 2.2 shows the basic architecture of a behavior-based system. From the figure, it can be seen

that the system is organised from several behavioral modules. Each of the behaviors will manipulate the data from the sensory inputs (e.g. camera, ultrasound, infra-red, tactile) to produce an output for robot effectors (e.g. wheels, grippers, arm, speech). As defined by Arkin [1], a behavior is a reaction to a stimulus. The relation between the input and the output of each behavior can be also expressed using a stimulus-response (SR) diagram, illustrated in Figure 2.3.

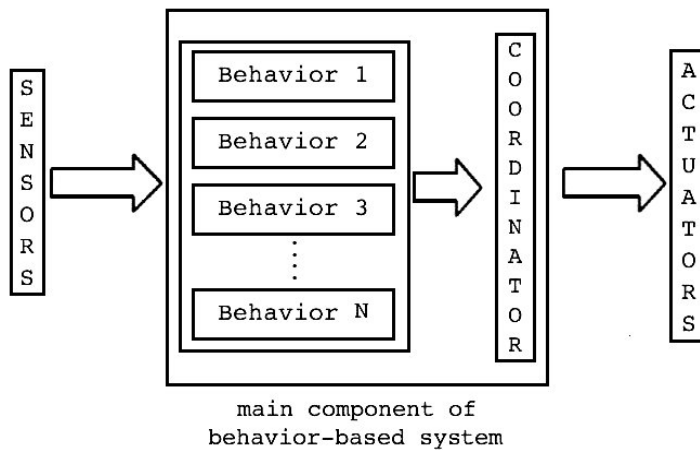


Figure 2.2: Basic architecture of a behavior-based system

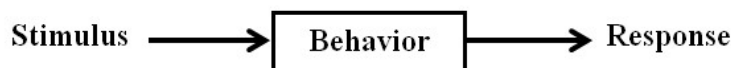


Figure 2.3: A stimulus-response diagram, redrawn after Arkin [1]

As mentioned by Pirjanian in [9], the development of the behavioral modules in a behavior-based system can be characterised by dividing the overall system's objective into simpler and smaller tasks. Therefore, each of the behaviors will be designed so that it can generate an action in order to reach or maintain its own goal [30]. For example, **avoid obstacle** behavior will utilise the sensory information to produce an output that can keep the robot safe from obstacles. It is important to note that this behavioral output can be used as additional information to the other behaviors as well.

The response generated from each behavioral module will then be used as control signals for controlling a robot to reach the overall system objective. One advantage of this modular structure is the contribution to the robustness of the system. This structure allows each of the behaviors to interact actively with the environment and with the other behavior components as well. Interestingly, the side-effect of these interactions can usually generate a useful high-level behavior that is not specified by the robot's program. This behavior is called an emergent behavior [36].

Figure 2.4 shows an example of an SR diagram for a target tracking task. As reflected from the figure, the behavioral components will play an important role to observe the current environment. Based on a given stimulus, each of the behaviors will suggest an appropriate action according to its own goal [30]. For example, the **Avoid obstacle** behavior in a target tracking task will always suggest an action so that the robot will not run into obstacles. On the other hand, a **Track** behavior will always suggest an action to move towards its target. It is also important to note that the proposed actions of each behavior, may or may not be aligned with the robot's overall objective.

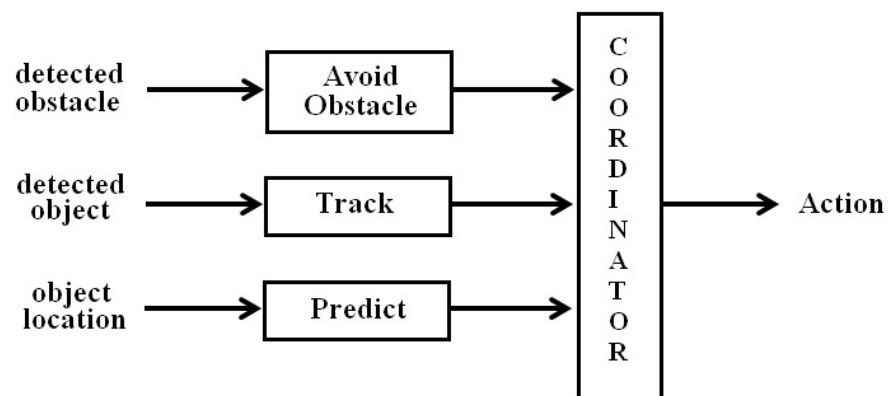


Figure 2.4: A stimulus-response diagram for target tracking task

According to Arkin [1], the overall robotic response can be expressed as below:

$$\rho = \mathbf{C}(\mathbf{G} * \mathbf{B}(\mathbf{S})) \quad [1] \quad (2.1)$$

where,

ρ is the overall robotic response.

\mathbf{C} is the behavioral coordination function.

\mathbf{S} is a vector of all perceivable stimuli s_i for each behavior β_i at time t .

\mathbf{B} is a vector of all active behaviors β_i at time t .

\mathbf{G} is a gain vector g_i that specifies the relative strength of each behavior β_i .

Since each of the behaviors β_i will produce its own specific response \mathbf{r}_i based on detected stimuli s_i ($\beta : S \rightarrow R$), the above expression can be alternatively written as:

$$\rho = \mathbf{C}(\mathbf{G} * \mathbf{R}) \quad [1] \quad (2.2)$$

where,

\mathbf{R} is a vector of all responses \mathbf{r}_i generated from each active behavior.

Let us say, two behaviors are involved in the development of a behavior-based robot control system. The first one is a **homing** behavior where it will produce a response in order to drive a robot towards a home location. The second one is an **avoid obstacle** behavior that always produces a response to ensure the robot moves away from any obstacles. Assuming the coordination function \mathbf{C} in Equation 2.2 is implemented using a vector summation method. In this case, the response from each behavior has been encoded as action vector (i.e. a vector of all

responses \mathbf{R}), consisting of orientation and magnitude components. If the behavior-based system has been developed so that both behaviors are equally important to the system, then, in this case, the gain value of 0.5 can be applied to each of the two behaviors representing a similar relative strength of each response (note that the gain value will be set using gain vector \mathbf{G}). Therefore, in a situation where both behaviors are generating a response to moving forward, the output of overall robotic response ρ will also produce a similar action as suggested by the behaviors (i.e. moving forward).

From Equation 2.2, it is clear that the gain vector and the coordination function are the two important elements that ensure the success of a behavior-based system. Note that the responses will be multiplied by the gain \mathbf{G} before the coordination function is applied. This is to set the relative importance of each of the behavioral components (i.e. if required), which will provide useful information to the coordination function. As an example, if the overall response has been chosen based on the highest-ranked component of an active behavior, the ranking of each of the behavioral modules will be set using the gain g_i . However, to set an appropriate value of the gain vector is not as simple as it seems. A series of questions may arise in order to do this. First, what is the right initial gain vector that should be assigned to the behavior-based system in executing a particular robot mission? Then, which behaviors are more important than the other behaviors? If an appropriate initial gain vector has been successfully determined by a trial and error process, should the value for each gain remain static along the execution of its task? Will the static value of the gain vector be able to cope with various uncertainties especially when the system is executed within an unknown environment? Unfortunately, there are no specific rules to answer these questions.

It is also important to note that the coordination function \mathbf{C} in Equation 2.2 should be properly developed. This is to ensure a single, stable output control signal can be generated to achieve a successful mission of the behavior-based system. This is because, at any given time, the behaviors may produce

conflicting control actions, which will lead to system instability. This is where the coordination function \mathbf{C} plays its major role. There are various techniques that have been proposed to encode the coordination function. Some of them have been discussed in the introduction chapter. Detailed coverage of this topic will be further discussed in the following sub-section.

2.2.2 Action Selection Mechanism

The coordination function \mathbf{C} in Equation 2.2 is also known as an action selection mechanism (ASM). ASM is a central issue in the design of a behavior-based control system. This involves choosing a particular action from a set of possibilities from behavior components in order to select the most appropriate robot's next action. Depending on how the behavioral module is developed, actions in this case may be considered at two levels: high-level and low-level. High-level action selection may be defined as choosing between abstract activities (e.g. go to a location, analyse target, etc). This usually involves more than one robot's movement to execute the selected action. One example of ASM that implements high-level actions, has been presented in [4]. As for the low-level action, it deals with the next physical action (e.g. go forward, turn left, stop, etc). Obviously, for this case, at every decision cycle the robot will make one step each in order to execute the selected action (e.g. as presented in [37, 38]).

Several classifications of ASM have been suggested, but as proposed by Pirjanian in [9], ASM has been classified into two main categories: arbitration and command fusion (Figure 2.5). Table 2.1 summarises some of the existing techniques that have appeared in the literature. It is important to note, no matter which category it belongs to, there are no constraints or rules in generating a method for behavioral organisation (i.e. ASM). This has led many researchers to investigate a number of ways to coordinate the behavioral module effectively. The following sub-sections give more comprehensive reviews on ASM according to the mechanisms used in selecting an appropriate robot's next action.

Table 2.1: A brief review on action selection mechanism as classified by Pirjanian [9]

Main Classes	ASM Technique	Description	Related Work
Arbitration	Priority-based	Action is selected based on priorities that are assigned to each of the behavioral units. Behaviors with higher priorities will take control of robot.	[10, 37]
	State-based	Behavior selection is done using state-transition. It will select a set of behaviors that is adequately competent at handling the situation upon the given state.	[39, 12]
	Winner-take-all	Behaviors will compete among them until one behavior wins the competition and the winner will take control of the robot.	[40]
Command Fusion	Voting	Behaviors generate votes for actions. Action that receives the maximum number of votes will be chosen.	[18, 41]
	Superposition	This method uses a linear combination of behavior outputs (such as vector summation) to create command signals for actuators.	[17, 42]
	Fuzzy	This is performed by combining the fuzzy output that encodes the desirability of each action by using fuzzy inferencing. Defuzzification is then applied to select final action that best satisfies the decision objectives.	[43, 44],
	Multiple Objective	Each behavior calculates an objective function over a set of permissible actions. Multiple behaviors are then blended into single complex behaviors to select an action that satisfies all objectives as well as possible.	[45, 46]

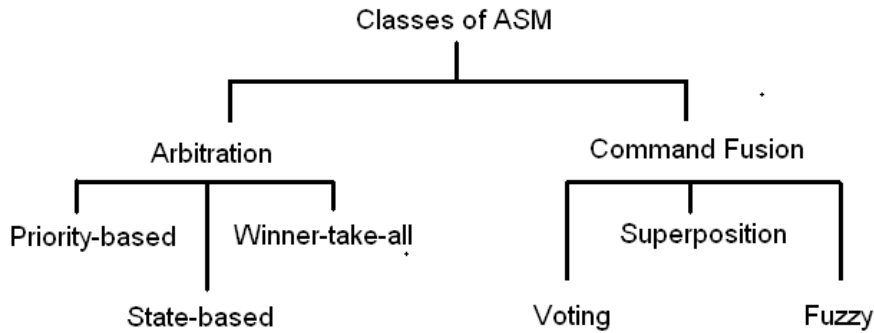


Figure 2.5: Class of action selection mechanism as categorised by Pirjanian [9]

Arbitration

As stated earlier, arbitration is an ASM approach category where each of the behaviors in the behavioral module will compete with each other in order to control a robot’s actuator. In this case, the action (i.e. response) suggested from a single behavior component will be selected by the coordinator (i.e. ASM module). It is also important to note, in some cases, other than the selected behavioral module, a non-conflicting behavior can be allowed to operate in parallel.

One well-known method in arbitration category has been presented by Brooks in [10] through his Subsumption architecture. Using his approach, a fixed-priority hierarchy has been assigned to each of the behaviors. Selection of one behavior (from many) to generate a robot’s next action is made based on the highest priority of active behaviors. This is done by allowing a higher-level (i.e. higher priority) behavior to suppress the input and override the output of a lower-level behavior. Figure 2.6 shows an example of an SR diagram of a simple behavior-based system, employing Subsumption architecture. As shown in the figure, two behaviors have involved in the development of the system: **Wander**, and **Drive To Target**. By default, **Wander** behavior is active, and the robot will be executed based on the action suggested by this behavior. However, when the robot senses a target object, **Drive To Target** behavior will be activated and suppresses the

response from **Wander** behavior. This is done through inhibition and suppression mechanisms as described in [10] and [1].

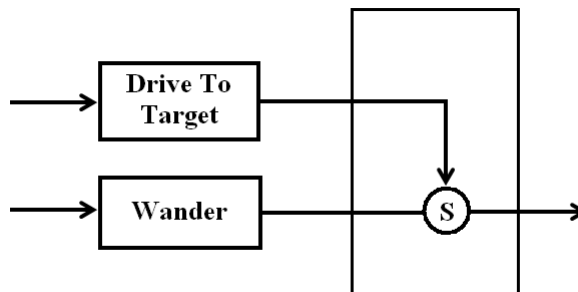


Figure 2.6: An SR diagram of a simple behavior-based system, employing Subsumption Architecture

Temporal Sequencing [47] is another method of ASM in this category. An illustration of the overall system can be represented as a Finite State Acceptor (FSA). Figure 2.7 shows an FSA representing flow of control for a door traversal task (an example which has been discussed in [9]). Employing this technique, the transition in executing a response from one behavior to another is determined by a triggered signal. As shown in Figure 2.7, there are five states (i.e. behaviors) in the FSA diagram. From the figure, it can be seen that the robot will start its mission once receiving a user command signal, and thus will select the **Find Door** behavior for controlling robot's actuator to search a door. While searching, if the robot finds the door, a signal will be triggered to allow a transition from executing **Find Door** behavior to **Traverse Door** behavior. During this state (i.e. **Traverse Door**), whenever the robot senses an obstacle in its path, the **Obstacle Avoidance** behavior will be invoked until a clear path has been achieved. A signal to end the robot mission will only be sent when the door has been successfully traversed. Clearly, similar to Subsumption architecture, only one behavior is taking control of the robot at any one time. However, as for the Temporal Sequencing technique, it will employ an FSA to set the sequencing between a series of behaviors. In this case, each of the behaviors will be executed when its corresponding signal has been triggered.

The work presented by [48] has also employed an arbitration type of ASM in their proposed control architecture. However, before a coordination function is

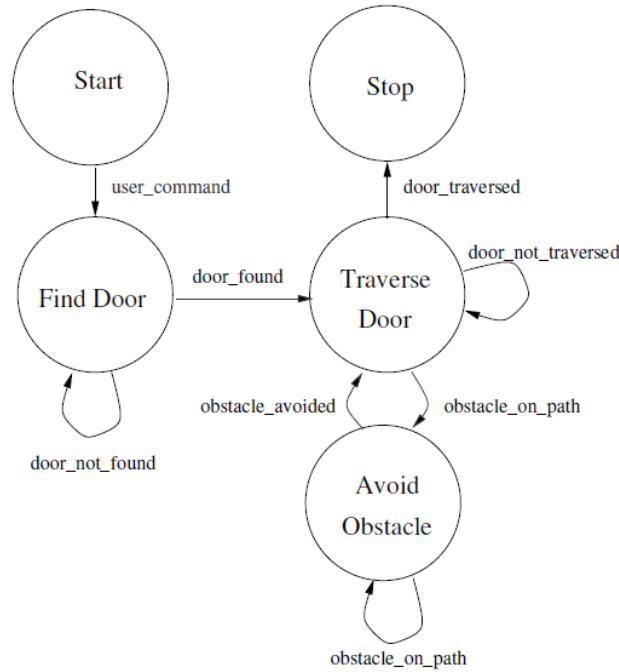


Figure 2.7: An FSA for a door traversal task, copied from Pirjanian [9]

applied, the response of each behavioral module (i.e. \mathbf{R} in Equation 2.2) will be adapted using the evolutionary technique. A neurocontroller have been evolved for each behavior component incrementally, starting with the basic behaviors (bottom layers). When the evolution of the particular behavior component has reached a suitable fitness score (i.e. the behavior is able to produce an appropriate response for the environment), a more complex behavior (upper levels) will be created. At this stage, the behavior from the lower level will be frozen. A simple behavior coordination (which is nearly similar with the Subsumption architecture approach) will then be applied. However, instead of selecting a behavior based-on a triggering condition, a control signal from a higher level behavior will be used to determine which response should be executed by comparing it with a predefined threshold. As reported, their proposed method has been successfully applied for a mobile robot navigation.

Command Fusion

Unlike arbitration, the command fusion approach will blend actions from multiple behaviors. The coordinator will apply a method that takes two or more behavioral responses to generate an output to control the robot. For instance, a voting-based method is used in Rosenblatt [18] through his behavior-based control architecture known as DAMN (Distributed Architecture for Mobile Navigation). Even though this approach is classified under command fusion, it is quite similar to an arbitration concept. However, it reduces the effect of suppressing other behavior outputs by introducing weights and votes in its *preference combining* algorithm.

Figure 2.8 illustrates a scenario of how votes are issued in DAMN. In this example, it is assumed that two behaviors are active: **Obstacle Avoidance** behavior and **Goal Seeking** behavior. Each of these behaviors will suggest its preference response based on these pre-defined possible actions. As shown in the figure, there are five possible options to control the robot movement (i.e. *Hard Left*, *Soft Left*, *Straight Ahead*, *Soft Right*, and *Hard Right*). The magnitude of a vote ranges from -1 to +1, indicating the behavior representing against that particular action (i.e. -1), or representing the most favourable action (i.e. +1). In Figure 2.8, the size of the circles illustrates the magnitude of the votes. A positive value of the magnitude is described by the unfilled circle, while the negative value is represented by the striped circle. As shown in the figure, the **Goal Seeking** behavior gives the highest vote (e.g. +1) for the most preferred *Straight Ahead* response. The next preference is the *Soft Left* response with a possible vote value of 0.5. Other than the two responses mentioned earlier, the **Goal Seeking** behavior is against the *Hard Left* response and the right turn responses. The large striped circle illustrated in the figure indicates that the behavior is against mostly to the *Hard Right* response. Note that, in this example, the smallest striped circle may be represented by a negative value which is near to 0. Other than the magnitude of a vote, each behavior is assigned with a weight value in order to show the importance of each behavior towards the overall system. As described in [49], since avoiding an obstacle is more

important than seeking for the shortest path to the goal (which is suggested by **Goal Seeking** behavior), the **Obstacle Avoidance** behavior is assigned with a higher weight. This is indicated by thicker arrows in Figure 2.8. An arbiter in DAMN then sums up the weighted votes received from the two behaviors, for every possible actions. The final action is selected by the arbiter based on the highest value of the weighted sum. As shown in the figure, *Soft Left* action will be selected.

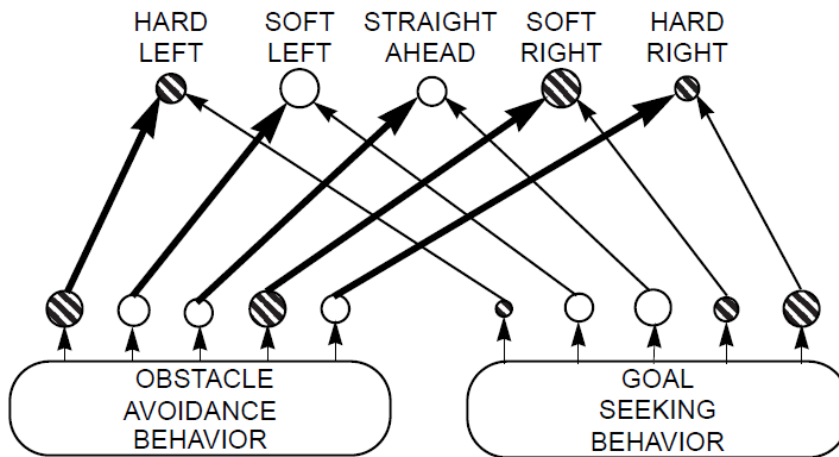


Figure 2.8: Behavior voting in DAMN, copied from [49]

Another popular approach of ASM in this category has been proposed by Arkin [17] through his Motor Schema behavior-based architecture. ‘Motor Schemas’ in this case are the behavioral modules shown in Figure 2.2. Based on a perceived stimuli (i.e. information used to compute the reaction for the respective behavior), each of the motor schema will produce an action in the form of a vector (consisting of magnitude and orientation components of a robot’s movement). The action vector from each motor schema will be multiplied with its associated gain value before applying the coordination function. Using this technique, a simple, straightforward vector addition is employed for its coordination function. The command resulting from this behavior coordination will then be normalised to ensure the control signal is executable on the robot. As stated in [1], it is also important to note that even the action vector of each motor schema is generated based on a potential field approach (refer [50]), only a single vector at a robot’s current

position will be used in determining the output of the behavior coordination (i.e. robot's next action). For example, assume a robot is on its way to a home that is located to the right side of a robot's current position. Also, at this position, the robot has detected an obstacle at the right side between the robot and the home. In this situation, motor schema from **Avoid Obstacle** may produce an action vector so that the robot will make a left turn in order to get away from the obstacle. However, **Goal Seeking** behavior will definitely produce an action vector to drive the robot towards the home location, which is located at the right side of the robot. Clearly, in this scenario, the control signal produced from the overall process described earlier will make a forward movement with the turning angle between the one suggested by **Avoid Obstacle** and **Goal Seeking** motor schemas. How much the robot is turning will depend on several parameters, such as the gain value assigned to each of the motor schema, and the distance between the robot and the obstacle.

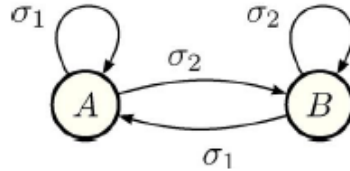


Figure 2.9: Fuzzy automaton for robot navigation with two fuzzy states, copied from Jayasiri et al. [51]

There are quite a number of works that have employed a fuzzy logic technique in the implementation of their behavior-based robot control systems [51, 8, 52, 43, 53, 54]. Jayasiri et al. [51] present an approach to control a behavior-based system using ‘supervisory control’ of Fuzzy Discrete Event Systems (FDES). In their proposed method, each of the behavioral modules is represented by a fuzzy state. Figure 2.9 is an example of a system developed using this method with two fuzzy states: A—representing **Avoid Obstacle** behavior, and B—for **Go to Target** behavior. As described in [51], two fuzzy events are defined for the transition from fuzzy state A to fuzzy state B, and vice versa. Depending on a robot’s sensory

information, these fuzzy events may be triggered by a module known as ‘supervisory control’. For example, assume a robot is initially executed in a clear path environment where no obstacles will be detected on a robot’s surroundings. In this scenario, only a fuzzy event labelled as σ_2 (refer Figure 2.9) will be triggered. However, in a situation where a robot sensed an obstacle, besides the fuzzy event σ_2 , the supervisory control will also enable fuzzy event σ_1 . In the proposed method, both fuzzy events must comply with each other to allow the robot to reach the target safely. This is done by computing a new set of fuzzy events which represent the weights associated to its corresponding behavior (i.e. state). Using these weights, recommendations from the two behaviors will be combined, thus, a command fusion type of behavior coordination is achieved.

Other Approaches

An effective ASM is expected to generate an action that enables a robot to move towards the satisfaction of the current situation. In doing so, the robot control system should be able to observe and utilise the information from the environment and consider the suggestion from all behaviors in order to achieve the overall system objective. However, to develop an ASM that can generate appropriate actions to be executed in various environmental conditions is not an easy task. Even ASM strategies that are developed from a fixed mathematical model may require parameter tuning through a series of experiments before a given target mission can be successfully implemented on a behavior-based system. Furthermore, when a reliable parameter setting is discovered, still it is never certain that the ‘best’ settings will be valid for all mission scenarios. This may be due to a number of constraints imposed by the real world, such as incorrect or incomplete data from sensor readings, the dynamic and non-deterministic environment, and the robot’s goal that may vary over time as the current situation changes during the execution of the mission.

Therefore, there are implementations that have highlighted the importance of having both arbitration and command fusion methods during the execution

of a behavior-based system [21, 20, 8]. Figure 2.10 and Figure 2.11 show the basic structure of an APOC (Activating-Processing-Observing-Components) component and the basic iB2C (integrated Behavior-Based Control) behavior module, as proposed by Scheutz et al. and Proetzsch et al. in [20] and [21] respectively. In both implementations, they have used their proposed basic structure to compose existing behavior coordination methods (e.g. Subsumption architecture, Motor Schema, DAMN, ALLIANCE), where special links are used to integrate various behavior coordination methods on the same control architecture. As stated in their reports, switching among different behavior selection strategies will be carried out through another mechanism. A different approach has been presented by Huq et al. in [8]. Instead of switching the ASM method, they have applied both arbitration and command fusion methods in their control architecture of a behavior-based system.

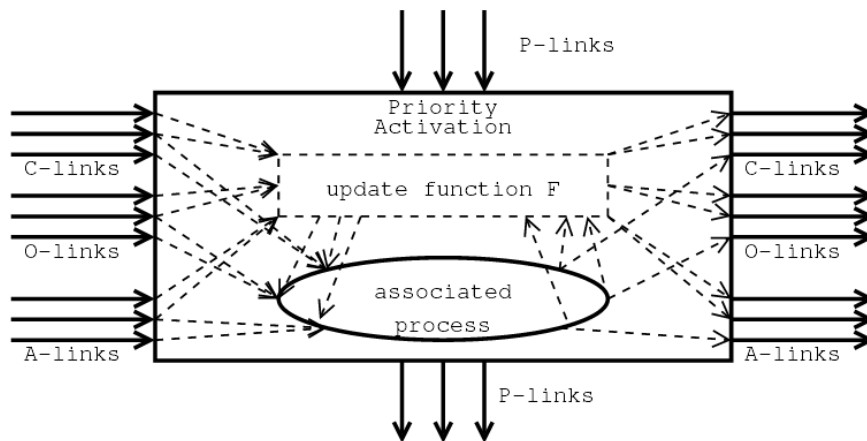


Figure 2.10: Basic structure of APOC, copied from Scheutz et al. [20]

There are also implementations that adapt the parameters in the coordination function in order to improve ASM techniques [5, 55, 56, 57, 14]. Adapting in this case can be an adjustment of internal parameters to cope with the environmental changes during the run-time of the system. For example, Parker [4] has proposed a mechanism that can activate a behavior set depending on motivation, which is computed during the execution of a robot's task by several components (i.e. the sensory input, activity of other behavior sets, and explicit communication

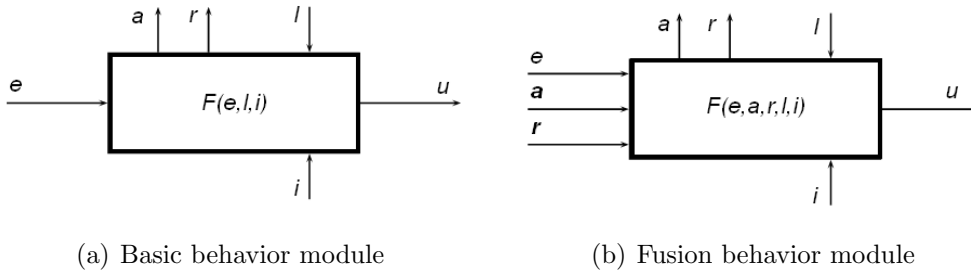


Figure 2.11: Basic structure of IB2C, copied from Proetzsch et al. [21]

among the robots). Whenever the activation level (which is defined by the motivation of each behavior set) exceeds a given threshold, the corresponding behavior set becomes active. This method has been successfully applied to hazardous waste clean-up missions which were executed by a team of robots.

Various works in behavior-based system have employed AI techniques to allow their ASM strategies to be adaptable with changing environments. One example has been presented by Ram et al. in [55]. In their work, AI has been applied to modify internal parameters that affect the strength (i.e. gain) of active behaviors. They have employed a genetic algorithm (GA) to automatically fine-tune parameters of three schemas (i.e. move-to-goal, avoid-static-obstacle, and noise) for a robot's navigation. Another example has been presented by Farahmand et al. in [13]. They have adopted three AI techniques in the development of their behavior-based system: an evolutionary algorithm, a reinforcement learning, and a culture-based memetic algorithm. The proposed method have used an evolutionary algorithm to generate several sets of behavioral modules. For example, if a behavior-based system requires two behaviors in the development of its system (e.g. **avoid obstacles**, and **go to target** behaviors), the evolutionary algorithm will generate several candidates for each of the respective behavior modules. A reinforcement learning on the other hand, will be used to learn the organisation of the behavior modules. Since a purely parallel Subsumption architecture has been employed for

its ASM, the role of reinforcement learning in this case is to find an appropriate ordering for each behavior module. Remember that the higher level of an active behavior module in Subsumption architecture can inhibit and suppress the behavior at the lower level. To sum up, employing evolutionary algorithm and reinforcement learning in the proposed method enables the system to find an appropriate set of behavioral modules and organises those behaviors in the architecture according to their received reinforcement signal. This is done by maximising the received rewards value when the agent interacts in the environment. Other than the two algorithms described earlier, Culture-based memetic algorithm has been introduced and used in the proposed method as a medium for sharing previous experiences of the other agents to accelerate the learning process. Note that the proposed method has been successfully implemented for multi-robot object-lifting tasks, which has been tested on a simulation platform. More examples of work that employ AI techniques to construct ASM strategies can be found in [14, 15, 16].

In short, an ASM is the most challenging issue in developing a behavior-based system. However, the successful system has shown a great potential for a robot execution especially in a real, unknown, dynamic world. This is one of the reasons why the research in this field has been continuously explored.

2.2.3 Emergent Behavior

As stated by Arkin in [1], most action selection mechanisms are developed from algorithms in which theoretically they will produce a deterministic and computable output. However, due to the dynamic properties and uncertainty of a real world, the output of coordinated action has usually generated unpredictable output. The unpredictable output is known as ‘emergent behavior’. It is one of the main features of a behavior-based system where the system will generate interesting behavior without explicitly being programmed to do so.

A simple example of an emergent behavior has been discussed in [2] and [26]. In the example, three behaviors have been constructed in the development of a behavior system:

1. **Avoid obstacle:** To ensure the robot does not get too close to the other robot or obstacles
2. **Stay close to the group:** To make sure the robot does not get too far from other robots
3. **Keep moving:** To allow the robot to keep moving in the environment, if possible.

This is assuming that the above behavior-based system is executed in parallel by a number of robots in the same environment. Note that each of the robots is executing the system independently without having any communication among them. Interestingly, by allowing interactions between the robot's controller and its environment, it is observed that the group of robots is performing a flocking behavior, which is none of the three behaviors that have been initially constructed for the system. In this example, the flocking behavior is an emergent behavior, as it is only arises at the execution time of the system (not pre-programmed using rules!).

However, it is also important to note that not all behaviors that emerge from the system's dynamics are desirable. For example, a robot with obstacle avoidance behavior may interact with the other behaviors and its environment, causing the robot to oscillate and get stuck in a corner. This unpredictable output is also an emergent behavior but will be treated as an undesirable rather than a desirable feature [2]. This is the reason why the development of a behavior-based system will usually involve a lot of trials and errors.

2.3 AI Techniques for Robot Control

A conventional way for designing a robot control system is based on a predefined set of rules (i.e. sensing to action pairs). Such a control design approach is suitable for robot tasks that are executed in structured and predictable environments. Robot arms used for manufacturing applications is a good example of a robot control system in this category. Since the positions and orientations of the robot hardly vary, and the robot is performing precise motions and repetitive processes, developing a program for such a system is not very difficult. This is because the rules to perform its tasks can be clearly defined.

Today, robots have been used for various challenging missions. Space exploration, processing radioactive ore in the nuclear industry and rescuing tasks are some of the risky jobs in which the use of robots can be advantageous. Reducing the risk to human lives is one obvious benefit of replacing humans with robots in such missions. However, these missions are usually executed in an unstructured environment. In consequence, it demands a robot control system to be able to modify its action appropriately when responding to any unpredictable and unexpected situations. Developing such a system can be very complex. This is because it is impossible to pre-program the control system to handle every possible situation.

Adopting an AI technique can be very useful in this case. Many research studies have proven that the technique is able to overcome some of the difficulties in designing a robot control system. This has been done in several ways. Parameter tuning is one example. For instance, Mehdi and Boubaker in [58] used AI techniques for parameter tuning. In their work, Particle Swarm Optimization (PSO) is applied to fine tuning the controller parameters of a robot manipulator. Apart from parameter tuning, AI techniques are also being used to manipulate a complex function to develop a robot control system for a specific application. For instance, Batllori et al. in [59] have employed Spiking Neural Network (SNN) to train a robot for a light seeking task. Using their approach, data from three infra-red sensors

and two lights sensors have been used to control a robot's movement by controlling the left side and right side of the robot's wheels. In order to train the neural network, a set of input-output data has been generated using a rule-based method for a similar task. In this case, the neural network will act like a black box where the function of the control system is not known. The function of the system will be approximated during the training process by mapping the input-output data. Another example of work that has utilised the AI technique for robot control has been reported in [60]. They have employed a reinforcement learning method to allow a robot to learn through its experience when interacting with the environment. Their proposed method has been successfully tested on a real robot (i.e. NEO) for backward docking at a charging station.

In short, various AI techniques have been explored to aid robot control designers in the development of their system for various applications. Fuzzy logic, reinforcement learning, GAs and neural networks are among the popular techniques that have been implemented in the literature. Some examples have been discussed previously. However, the focus of this section will be given to approaches under evolutionary robotics (ER) field which is closely related to the work presented in the thesis.

2.3.1 Evolutionary Robotics: An Overview

Inspired by the Darwinian principle, ER uses evolutionary computation to develop robot controllers. Algorithms in ER frequently operate on populations of candidate controllers which will evolve and repeatedly modified according to a fitness function. The performance of each candidate controllers will be evaluated using the fitness function, which can be further used to determine how well each of the candidate controllers to achieve the global objective. GA is one popular evolutionary algorithm implemented in ER.

The reason of employing a GA for robot control is mainly due to its adaptive capabilities [61, 62, 63]. This allows a robot to work in a dynamic environment where the goal or constraint can be changed over time. Another reason is the ability of the algorithm to develop an effective controller without having full understanding of the relationship between the robot and its environment [25, 64, 65].

However, the evolutionary process requires a few iterations before it can achieve a desired solution. Applying this method (testing individuals) on a real robot will require a huge amount of power. Furthermore, the fact that the solution could be some random solution (especially at the early generations) increases the possibility of exposing its platform to physical damage.

A robot simulator can overcome such potential hazards. As mentioned by Nolfi and Floreano in [25], using a simulator can also reduce the time required to evolve a satisfactory controller. Nowadays, there are many robot simulators available and many of them have the option to execute the simulation in fast mode. In fact, most robotics researchers tend to completely evolve the controller within the simulation environment until a desired system response has been achieved, before transferring to a real robot [66, 67].

Alternately, the controller can be evolved partly using a simulator until an approximately good result has been obtained. The following evolution will then be continued on a real robot for fine tuning until the preferred system is found [68]. Even though there is a huge amount of work that has been carried out using simulators within the evolutionary robotics community, there are still several projects presented where the experiments have been fully conducted on a physical robot [69, 70]. More works related to the methods of evolving a robot controller have been discussed in [25, 64, 65].

In any case, the success of an approach using GAs will greatly rely on the definition of its fitness function. Nelson et al. have presented a survey and analysis on the usage of fitness functions in [71]. The surveyed research has compared the

amount of prior information being used to successfully evolve a specific controller to perform a specific task. For a more practical approach, fitness functions should be defined using data that is only available in a real environment. Without relying on data that is only available from the simulator, the system will be more adaptive and more robust to a changing environment where further evolution can be carried out (if necessary) on a real robot platform.

Another research area that is very similar to ER would be developmental robotics (DevRob). However, rather than trying to evolve a population of robots (social development), DevRob focuses more on evolving a single entity (individual development). In other words, the evolution is focus on a single robot's control system, similar to how a child develops through experience, over time.

Although it may seem that the implementation need not keep huge data (e.g. gene pool), it needs instead to retain as much memory of previous experience as possible. This memory of accumulated experience needs to be available for the robot controller to evolve over time. This fact may dampened the desire for hardware implementation as it needs an expandable data storage to keep DevRob platform experiencing new things.

2.3.2 Combining Neural Network with Genetic Algorithm

Many success works in ER have employed neural network controller in their implementations. The ability of a neural network to learn and generalise a non-linear function is what makes it attractive for many tasks. Instead of having to go through lengthy mathematical analysis, complex functions can be approximated by an artificial neural network through some pattern recognition mechanisms on a given set of data. Neural networks have been successfully applied to various areas of robot control [72, 73, 74]. Navigation and target tracking are among attractive applications [75, 76, 77].

The success of a neural network highly depends on how well it adapts during the learning process. For a basic multi-layered perceptron (MLP) feed-forward neural network with back-error propagation (BEP) learning algorithm, this is done by altering the weights of its connections using the value Δw_{ij} calculated in Equation 2.3.

$$\Delta w_{ij} = \eta \delta_j y_i \quad [78] \quad (2.3)$$

where,

Δw_{ij} is the value used to update W_{ij} (the weight connection from node i to j)

η is the learning rate

δ_j is the value to be back-propagated

y_i is the output of input neuron i

and,

$$\delta_j = \begin{cases} y'_j(d_j - y_j), & j \text{ is an output node} \\ y'_j \sum_k \delta_k W_{jk}, & \text{otherwise} \end{cases} \quad [78] \quad (2.4)$$

As can be seen from Equation 2.4, the term δ_j for an output node is slightly different from that of a hidden node because of the fact that a desired output, d_j , is available for the output nodes. This error-based term will be propagated backward beginning from the output layer to the previous layers. Using this approach, the combination of weights that minimises the error value will be the solution for the learning task. Noticing the term y'_j (derivative of the output value y_j) in δ_j , the BEP algorithm strives to minimise the error term $(d_j - y_j)$ using the gradient descent method. However, a problem with this technique is the possibility of being trapped in a local minimum where the minimum error found is not the global minimum of the search space.

Another problem of using the BEP algorithm, or any other supervised learning methods, is the need to supply data for the learning process. As shown in [79], the BEP algorithm has been applied to train a recurrent neural network for motion control of a mobile robot. Notice that a huge series of input-output pairs generated from simulation data have to be provided for the training before the network can converge into a desired solution (i.e. learn the pattern). Thus, a problem of such learning algorithms is the requirement of huge training data sets, which are usually proportional to the complexity of the input-output patterns.

An alternative learning method that can be used is known as Reinforcement Learning (RL). In terms of machine learning, it defines an *action* that an *agent* should take in an *environment* in order to achieve a global objective. The *environment* is usually represented by a finite number of *states* and each *state* will have a set of possible *actions*. Every *action* selected by the agent (based on a given *policy*) will generate a feedback signal (a.k.a. reinforcement signal) that will be the basis for the learning process. This makes it suitable to learn problems where the *agent* does not know the so-called ‘correct’ *action*, which is quite common in a real-world scenario. This learning paradigm emphasises the trade-off between exploitation (using what we already know to optimise results) and exploration (try new things that may or may not be beneficial).

The problem with RL approach is that we need to define the *states* of the *environment* and its *action* paths, as well as evaluating the *rewards* (i.e. reinforcement signal) related to it. In addition to that, because of limited perception (i.e. sensor features), it is virtually impossible to discretely define the current *state* of the environment. Consequently, this affects the coverage of *state-action* pairs of the environment. This can be seen in [80], where a robot control application using a neural network and reinforcement learning has been presented. In their work, they have used a special topology of a neural network called Adaptive Heuristic Critic (AHC) Neural Network. As stated in the text, the solution lacks the number of input sensors to properly classify the environment characteristics.

Similarly, the evolutionary algorithm is able to train a neural network without a predefined set of input-output pairs [81, 82, 83]. However, the primary advantage of using an evolutionary algorithm over RL can be highlighted when the availability of state information is little or none. In other words, the evolutionary algorithm does not introduce any constraints on what can be the part of self-organisation process. In [84], a method called Cooperative Synapse Neuroevolution (CoSyNE) has been used to train a neural network based on an evolutionary algorithm. This method has also been compared to a wide range of RL through some experiments on a pole balancing problem. In the study, CoSyNE has been shown to be more reliable and efficient in comparison to RL. Although some texts may classify the neuro-evolution method under reinforcement learning approach, it is actually another form of machine learning that utilises an evolutionary algorithm like a GA.

2.3.3 Trends of Application in Evolutionary Robotic

The earliest success in ER has been presented in [85] through a navigation task. It is important to highlight that to hand design a navigation control system even for a simple environment can be very difficult [86]. To obtain a good trajectory, the designer should fine tune many parameters in order to successfully control the direction of robot's motion and its velocity. In addition, to implement a similar navigation task on different robots and/or in different environments may require different sets of carefully chosen value of parameters.

The experiment conducted by Floreano and Mondada [85] revealed that the evolutionary approach could find a solution for straight navigation and obstacle avoidance without a prior knowledge on robot's sensors, motors, and environment. In the experiment, Florano and Mondada employed a simple GA to evolve the synaptic strength of a recurrent neural network. The neural network was composed by eight input neurons and two output neurons. The input neurons were attached

with infrared sensors (placed at eight different positions) and the output neurons were connected to each wheels of the robot. The performance of each individual (i.e. fitness function) was based on three components: speed, straight motion, and distance from objects. It should be noticed that even though the fitness function did not specify the direction for the robot to navigate, the evolved neural network can automatically generate a navigation control system by exploiting the interactions between robot and its environment. As the results, the evolved controller was observed being capable to avoid deadlock situations and resume with smooth trajectory. It is also claimed that the evolved controller can self-adjust the maximum cruising speed to ensure safe navigation. Other works on navigation using ER approach has also been reported in [87, 88].

Another popular platform being used by ER researchers to test their techniques in this field is legged robots. Early works in this area was inspired by insects, where a neural controller was evolved for simulated hexapod robots to walk in simple environments [89]. Figure 2.12 shows the proposed distributed neural controller used for hexapod movement. Using a similar distributed neural network, the research was further explored to implement the controller on a real robot, which has been successfully executed on a flat surface. This has been tested even with the existence of obstacles in the environment. In order to achieve a ‘stick’ insect gait controller, a more advanced approach was proposed by using a different distributed architecture of a neural network, combined with a more sophisticated mechanical design. Results from the experiment were promising as the robot controller successfully generated a relatively smooth locomotion over an extremely irregular terrain.

Various versions of legged robots have been developed. However, the study to coordinate movements of a biped robot (i.e. a robot with two legs) could be the most challenging research area in this field. Even a slight unevenness of the floor can cause serious instability to the biped robot. Since the dynamics of a biped system is nonlinear and difficult to analyze, it is not surprising that quite a number

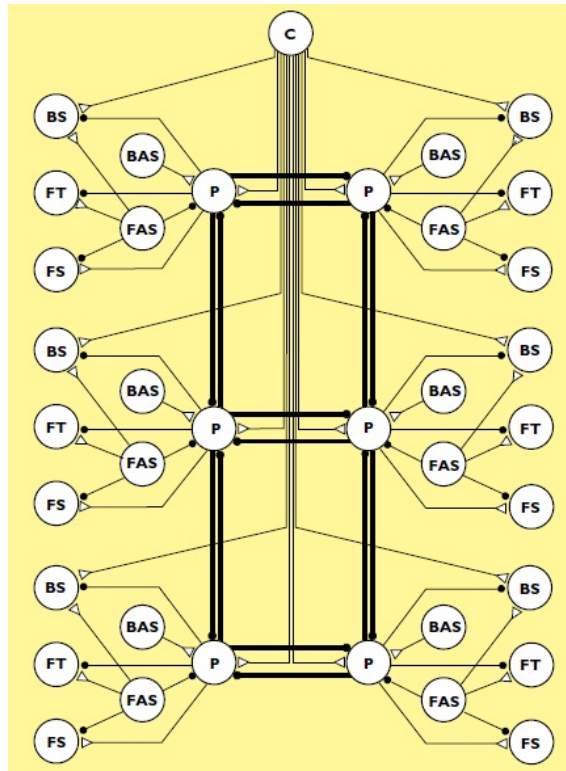


Figure 2.12: Distributed neural network for hexapod robot, copied from Beer et al. [89]

of research have adopted various AI techniques to generate biped gait [90, 91, 92]. One examples has been presented by In-sik L. et al. where a GA is used to generate a stair walking trajectory of a biped robot which is similar to a human [93]. This was done by optimizing seven design variables (which have been defined for human gaits) to generate optimal locomotion patterns for a biped robot to go up and down a stair. The effectiveness of their proposed method has been tested on a simulation platform.

Multi-robot systems offer better reliability compared to single-robot systems based on the fact that it offers data redundancy and greater coverage of a solution space. However, the main issue in the development of a multi-robot system is to ensure collaboration of each individual system working together in the same environment towards accomplishing a single task. Quite a number of research have employed evolutionary algorithm in order to help in finding an optimal team-coordination solution. An interesting example has been presented in [94], where

a team of Unmanned Aerial Vehicles (UAVs) are required to perform an environmental survey mission across an unknown region. In their implementation, each UAV is developed using a neural controller. The weights of each neural controller were evolved to develop a policy (i.e. a function for the robot controller). In this research, N populations of neural network controllers were coevolved (i.e. multiple populations evolved simultaneously) for N number of vehicles. In order to measure the fitness function, the proposed approach has implemented ‘difference evaluation functions’ where it quantifies each individual agents contribution to the team. This way, the relative usefulness of each robot in the team can be determined. Based on the experiment results, the proposed method (i.e. coevolution with different evaluation functions) outperformed the other two benchmark systems (i.e. coevolution with global evaluation function, and hand-coded patrolling algorithm).

2.3.4 Evolutionary Approach in Behavior-based System

As shown in Figure 2.2, behavior modules and coordinator (i.e. action selection mechanism) are the main components in the development of a behavior-based system platform. Therefore, this section reviews several works that have employed EA to evolve an elementary behavior of a robot system, and those that evolved action selection components of a behavior-based system.

As discussed in Section 2.2, the behavioral modules of a behavior-based system are usually composed by breaking down the overall systems objective into simpler and smaller tasks. Therefore, each behavior is designed so that it can generate an action in order to reach or maintain its own goal. This is done by manipulating data from the sensory inputs in order to produce an output for robot effectors. In general, each of the behavior modules is usually developed by hand coded algorithm which requires many trials of parameters tuning. As an alternative, ER researchers have explored an automatic way to develop the behavior modules. Several publications have shown that a neural network and an EA can be employed

to evolve an elementary robot's behavior such as obstacle avoidance, edge following and homing. Most implementations usually feed sensor data directly to the input of the neural network, and the EA will alter either the weights, or both the weights and the structure of the network until the robot exhibits the required behavior [95, 96, 97, 98]. Another possible approach has been presented in [99]. In this work, a GA is used to evolve 64-bit strings that encoded the mapping from eight possible sensor-states to motor actions. Results from the experiments show that a real LEGO Mindstorm robot was able to perform a tactile wall-following behavior efficiently.

In the field of behavior-based robotic, many researches have been focused on finding an efficient method to implement the ASM component. Deciding what action to take at every time steps is crucial to ensure a behavior-based system is able to work in its environment. Thus, quite a number of researches have explored the use of EA in the development of ASM. A few examples have been discussed in Section 2.2.2. As for the work presented in [100], the author has employed GA to evolve behavior coordination based on an FSA (a description of using an FSA for ASM can be found on Section 2.2). To implement this, important parameters to design an FSA have been used as the genotype representation. The FSA has been evolved incrementally, where a functional controller is extended to a new functionality until the desired system objective has been achieved. The proposed method has been tested on a high lifting fork robot where the controller has performed well and resulted in reliable cargo delivery behavior.

Rather than focusing on a development of a specific component in a behavior-based system, Nolfi proposed a different approach. Nolfi claims that the whole process in the development of a behavior-based system (i.e. the process of breaking down the required behavior into subcomponents, the development of each behavior modules, and the development of action selection mechanism) should be accomplished by using learning or an adaptation technique. As presented in [101], a type of a neural network known as 'emergent modular architecture' was trained

for a mobile robot to clean an arena. Figure 2.13 shows the connections of a neural network to implement the task.

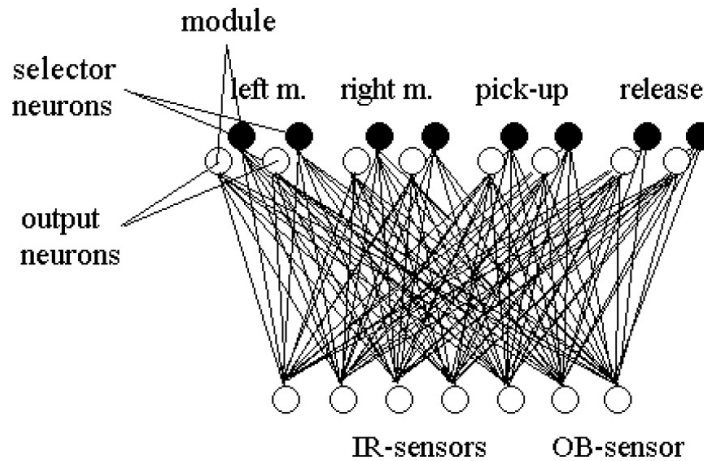


Figure 2.13: An emergent modular neural network architecture, copied from Nolfi [101]

In this implementation, seven input neurons were used to feed data from six IR sensors, and a barrier light sensor. The input neurons were connected directly to 16 output neurons. The first four pairs of output neurons (represented by empty circles) were coded for the speed of the left and right motors and for the triggering of the ‘object pick-up’ and ‘object release’ procedures. The other four pairs (represented by full circles) were used to determine which two output neurons had control over the corresponding robot’s effector (responsible for the selection mechanism). The neural network was trained using a GA, and was found to perform efficiently for picking up and releasing five target objects outside the arena without displaying any incorrect behavior.

The work presented in [94] used a behavior-based architecture to develop a multi-robot system. To do this, four Fuzzy Logic Controller (FLC) are utilized for four different levels of hierarchy in a behavioral architecture for a multi-robotic system (i.e. basic behaviors, behavior coordination, role building, and role assignment). Since the design of an FLC usually require a lot of ‘trial and error’ fine tuning, EA was used to optimize related fuzzy logic parameters. As reflected from the presented results, the evolved fuzzy behavior-based system has performed an efficient robot soccer system which is capable to attack and defend aggressively.

2.4 Summary

The work in this thesis shares motivations and goals with a number of related fields that can be categorised in two main areas: robot control and AI. This chapter reviewed the most related lines of research from each of these fields in preparation of introducing a new methodology proposed in this thesis.

Chapter 3

Genetically Evolved Action Selection Mechanism (GEASM)

The proposed Genetically Evolved Action Selection Mechanism (GEASM) architecture presents an alternative solution for behavior-based systems operating in an unknown or dynamic environment. The main idea of this control architecture is to utilize the versatility of an artificial neural network (ANN) to imitate any transfer function that may represent an effective action selection mechanism (ASM). This is actually a huge advantage since ANNs are known to be able to find a solution even if it is not linearly separable (e.g. XOR gate function) [78].

With an ANN engine at its core, the architecture requires a learning algorithm that can browse through the huge pattern search space created by the array of synaptic weights in the ANN for a solution that represents the most efficient ASM functionality. For this, GEASM employs a genetic algorithm (GA), which is a robust search heuristic [102, 103, 104] that uses natural evolution techniques—hence the name.

3.1 Overall View

Referring to Figure 2.2 shown in the previous chapter, GEASM attempts to improve both the behavior coordination procedure (i.e. the ASM) and the formulation of that procedure. For the first part, an ANN is used as the behavior coordinator module in GEASM. An important point that should be highlighted here is the direct feed of sensor output into the ANN. Notice that this is somewhat similar to the architecture used in evolutionary robotics (ER), which has been discussed in Section 2.3. GEASM, on the other hand, maintains the need for behavior modules and also uses their respective outputs as inputs to the ANN. Therefore, GEASM could also be seen as a merger between behavior-based and ER approaches. A block diagram of GEASM in a behavior-based system is shown in Figure 3.1.

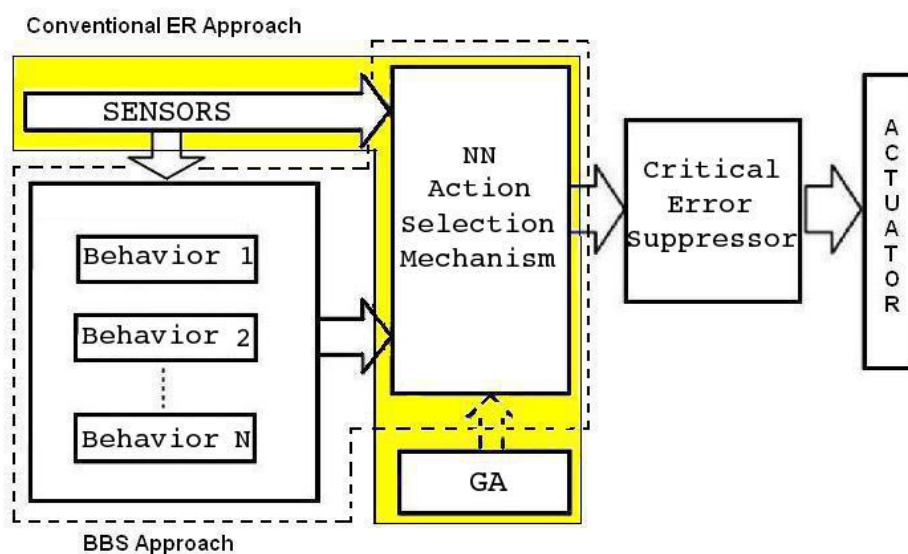


Figure 3.1: The proposed control architecture using GEASM

As for the second part, a GA is used mainly to evolve the ANN towards creating an effective ASM. In addition to that, due to the proposed connectivities in GEASM architecture, the GA may also evolve the system towards emergent behaviors [1], which are behaviors that were not thought of at design time, but turn out to be essential to the system. It should be pointed out that since GA is

mostly used in simulated offline training, it is not actually a ‘hardware’ component in the GEASM architecture.

In order to implement GEASM, all the connected components (e.g. sensors, actuators) need to have a clearly defined interface. The next section will attempt to explain the functions of each component shown in Figure 3.1.

3.2 Behavior-based Control

A behavior-based control works through the integration of a set of interacting behaviors in order to achieve a desired system objective. Each behavior manipulates data from sensory devices (e.g. camera, ultrasound, infra-red, tactile) to produce an output for robot actuators (e.g. wheels, grippers, arm, speech). The synthesis of behavior modules may differ between implementations depending on how a behavior unit is defined and how a response action is encoded.

As pointed out in [1] and [25], there is no universally accepted description of what a behavior module should be or how it should respond. This is only logical since such system is historically an evolution of a reactive system that primarily reacts to its environment, and since a system may have a different way of sensing its environment, reactions will most certainly be different.

There have been attempts to define and implement a complex behavior module that represents abstract behavior and produces high level action. In [4], a ‘go to home’ behavior has been defined, which produced a set of actions to allow a robot to move from its current position to a predefined ‘home’ location. Clearly, this complex behavior, known as motivational behavior, actually consists of multiple movement phases for a mobile robot to execute. This is rather obvious since a motivational behavior does not have any direct control over the actuators—instead, it simply suppresses or inhibits the outputs of low level actions to the actuators.

The most common method to implement behavior modules is to define and implement simple behavior modules that is capable of supporting a system's primary objective. These simple behavior modules are usually designed based on fundamental behaviors—for a mobile robot, those would be something like obstacle avoidance. Consequently, these modules usually produce low level actions [37] such as 'drive forward' and 'turn right'.

In the proposed control architecture, each behavior is designed to propose a robot's next movement in terms of distance (how far to go forward or backward) and angle (how much it should turn and which way). The distance parameter will represent the strength or magnitude of the response, while the angle parameter denotes the vector or direction of the response.

With multiple behavior modules modelled with different 'functionality', conflicting actions or output parameters may occur, which presents a fundamental issue in behavior-based systems. For example, let us consider a behavior-based mobile robot that is executing a task that requires it to move towards a certain target object. It then sensed both an obstacle and the target object, at the same time, to be in front of it. In this situation, the behavior module that encourages it to move away from obstacles may suggest to the robot to turn away, while another behavior module responsible for getting to the target object would suggest otherwise. In principle, the suggestions from both behaviors are very useful for the robot to make a decision on an appropriate action for controlling the robot's actuator—in this case, the motors that are controlling the robot's wheels. As in any similar decision making situation, to digest the suggestions from these two behavior modules in producing an optimal action is not an easy task. This is where the behavior coordinator or ASM is needed.

As pointed out in [1], the coordination function can be generally represented as Equation 2.2, which has been introduced in the previous chapter. Clearly, this is the function that needs to be infused into the ANN's neural connections. Various methods have been used to modify the weights of ANN's neural connections

(i.e. to train the ANN) [78, 105, 106]. In GEASM, GA is used, as will be explained further later on.

Note that there is a gain value that is applied to the output of each behavior modules before being fed into the coordination function. This is done so that each behavior can be assigned a relative importance factor that represents its significance towards the overall system response. However, in the proposed GEASM architecture, response from each behavior will be directly fed to the ANN without any scaling multiplier. This actually makes GEASM implementation a lot easier since the ANN is expected to be trained to adapt a suitable value for this purpose as well. Therefore, this eliminates the difficulties in assigning an appropriate gain value which is probably different for every decision cycle.

Somewhat of an issue that needs to be addressed when applying an ANN for robot control applications is the possibility of it operating in a random manner. This is even more so when a GA is used to train that ANN because the GA will generate a new candidate in every generation, which allows it to randomly traverse into new regions of the search space. Because of this, applying GA to generate new weights for the ANN controller during training produces an even higher possibility for that random action to happen. Such actions could result in the robot moving away from a clearly visible target or critically driving straight into obstacles. In order to ensure that the robot does not make such critical error during training, a suppressor module is introduced at the output of the ANN.

However, the main role of the critical error suppressor module is actually to provide information of action selection errors generated by the ANN, which will be used during training phase. It should be noted here that, while bumping into obstacles may not be that critical if the robot platform has a robust mechanical design, the error information gathered by this module would really help GA in determining the performance level of that particular chromosome. The use of this error information in GA will be discussed further in chapter 4. It is therefore important to note that even when this module is not really needed and can be

safely removed from the final system, it is a critical component in the training phase of GEASM.

3.3 Core Design

The proposed architecture mainly revolves around two main tasks—creating a simple ANN engine as the ASM core and implementing a basic GA to train that ANN. For the ANN engine, a Multi-Layered Perceptron (MLP) feed-forward neural network has been chosen because of its simple architecture. This ensures that the proposed architecture can actually be implemented on real robot platform. As for implementing a basic GA, only the most fundamental genetic operators has been selected for implementation and this will be discussed in the coming sub-section.

3.3.1 Multi-Layered Perceptron

Generally, an ANN is made up of highly interconnected processing elements called neurons (which are also known as nodes in some text). The way these neurons are interconnected with each other (i.e. its network topology) will define the type of that particular ANN. The most simple and most commonly used topology, the MLP feed-forward neural network, has been chosen to implement GEASM.

As shown in Figure 3.2 and as its name implies, an MLP neural network is simply an ANN with multiple neuron layers connected in a feed-forward manner (i.e. the output of each layer becomes input of the next layer). The neurons (also known as perceptrons, hence the name MLP) in each layer is connected to all the neurons in the next layer. An MLP normally consists of an input layer, an output layer and one or more hidden layers, with at least two of them are processing layers. The input layer is usually not a processing layer and will only be used to pass input signals to the next layer. This is the reason why there is at least one hidden layer in an MLP network.

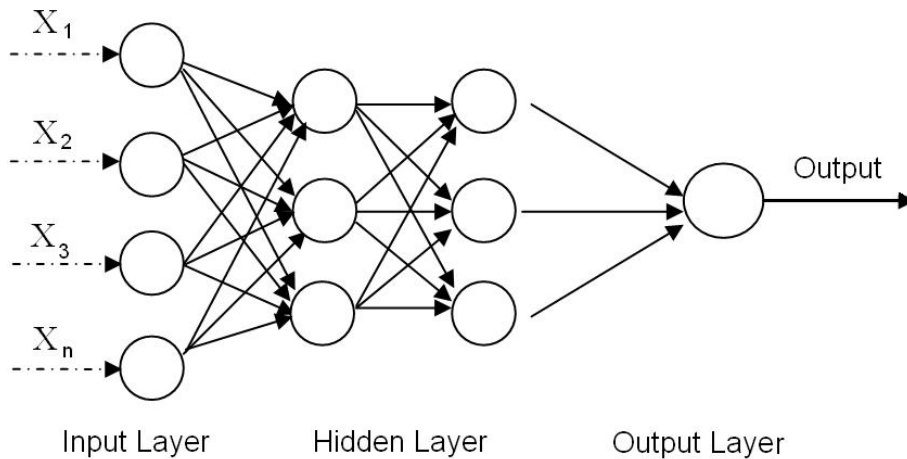


Figure 3.2: Multi-layered perceptron, redrawn after Haykin [78]

The number of neurons required in each layer may vary depending on the complexity of target applications. For GEASM architecture, the number of input neuron will be directly proportional to:

1. the number of behaviors multiplied by two (as each behavior will suggest a value for distance and angle to control an actuator), and
2. the number of data manipulated from sensors that are used to perceive the state of the environment.

As for the output layer, only two neurons are needed to produce the distance and angle parameters required to control the robot's movement.

Many works have proven that a single hidden layer with nonlinear activation function (e.g. sigmoidal function), is adequate in approximating any continuous functions with a good accuracy [107, 107, 108]. Since GEASM is going for simple implementation, only one hidden layer will be implemented. Deciding the number of neurons for the hidden layer is trickier than the other two because there are no rules for this. More neurons means better chances of adapting to complex functions, but this also means a higher complexity network interconnections. Even though there is no specific rule to determine the optimum number of hidden neurons, there are some suggestions in the literature that can be used as a guideline [109, 110, 111].

Further description related to number of hidden neurons will be discussed in the next chapter.

The operations of an ANN, no matter how complex it may be, are defined by the basic processing element in a neuron. Each neuron in a processing layer will process an incoming input signal as illustrated in Figure 3.3. Theoretically, the response or output of a biological neuron will only either fire or otherwise. Thus, it is believed that it is the rate of firing, instead of the amplitude, which really conveys the magnitude of the information. However, an artificial neuron normally uses the magnitude of an output signal to relay the information.

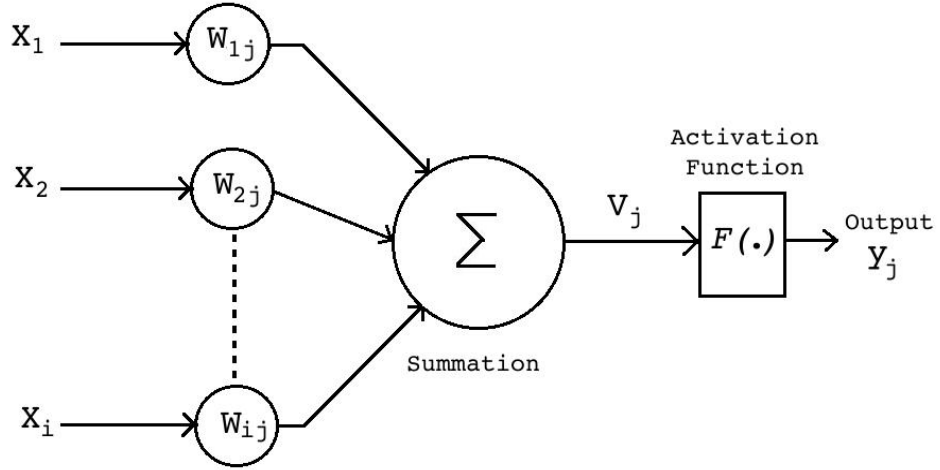


Figure 3.3: Model of artificial neuron, redrawn after Haykin [78]

Incoming input signal X_i will be scaled by a weight value W_{ij} (which represents the connection strength of a synapse) that has been assigned for every connection. Then, they are accumulated and fed into an activation function. The summations of weighted inputs are given by Equation 3.1.

$$v_j = \sum_{i=1}^n (X_i W_{ij}) \quad [78] \quad (3.1)$$

$$y_j = \frac{1}{1 + e^{-v_j}} \quad [78] \quad (3.2)$$

In this work, the sigmoid function (as shown in Equation 3.2) has been selected as the activation function of all neurons. This is simply due to the fact that it provides a continuous linear region between two limits (as opposed to a thresholding function), while acting as a limiting function that can filter noisy inputs (similar to a thresholding function) [112].

The most interesting feature of an ANN that makes it valuable in many systems is their ability to learn. This is done by modifying the weight value W_{ij} of interconnections between neurons using a learning algorithm. Learning algorithms can essentially be categorized into supervised and unsupervised learning. In supervised learning, samples of input-output pairs will be provided to the ANN during training phase. The learning algorithms in this category will approximate the target function that can map the available input data to its correct output. As stated in [79, 113, 114], such a learning approach requires a huge amount of training data. In addition to that, as discussed in [115, 116, 117], the training data must be carefully selected in order to ensure a successful learning process. In general, the training data should represent as many conceivable solutions as possible, so that it covers the whole spectrum of viable solutions.

On the other hand, unsupervised learning enables an ANN to learn without having specific desired system responses (no explicit target outputs). The only information provided to the neural network during its training are the observed input patterns. The learning systems in this category mostly work based on statistics-driven techniques that usually attempts to find key features of input data pattern. Therefore, such a learning system is usually suitable for applications such as data clustering, feature extraction, and similarity detection [118]. One thing that distinguishes the algorithms in this category from the ones in supervised learning is the fact that an evaluation of a potential solution has to be done without any external help.

For a robot control application that is executed in an unknown environment, the robot is exposed to various kinds of uncertainty. Clearly, for such an

application, it is impossible to provide a complete example (i.e. input-output samples) in order to allow the ANN to successfully learn a function that represents the desired functionality. Therefore, a variant of supervised learning method known as reinforcement learning would be a great option for implementing GEASM. Reinforcement learning is very similar to supervised learning in the sense that a potential solution can be evaluated to a certain extent. However, unlike supervised learning, it does not need any ‘correct’ input-output data pairs—a potential solution is instead evaluated using merit-based system. Each action taken during an evaluation phase will be rewarded (or sometimes penalized) accordingly based on criteria defined for a particular task. As an action is not directly corrected, reinforcement learning also involves finding a balance between using current knowledge (exploitation) and learning something new (exploration). Using the same approach, a GA is used in GEASM to modify the weights of ANN in order to obtain the desired function.

3.3.2 Genetic Algorithms

GAs are inspired by the principles of natural evolution techniques, which comprised of basic genetic operators such as selection, crossover and mutation [119]. Simplicity, robustness and the ability to search solutions for complex problems are the main reasons that contribute towards the popularity of GA as a tool for optimisation and machine learning. The process involved in GAs can generally be described as shown in Algorithm 1.

Algorithm 1 A genetic algorithm

- 1: Generate the initial population of chromosomes
 - 2: **repeat**
 - 3: Test every chromosomes and assign a fitness value
 - 4: Execute genetic operators to form a new generation
 - 5: **until** a satisfying solution has been found
-

As shown in Algorithm 1, a GA requires a population of chromosomes, each of which represents a possible solution to a particular problem. The initial

population usually consists of randomly generated candidate solutions. In order to evaluate these candidate solutions, an individual is created from each chromosome (candidate solution) and put through a test in which the individual's performance will be assigned with a fitness value.

The jungle rule—'Survival of the fittest'—will then be applied to the population in an evolution process based on predefined genetic parameters. The first parameter is the death rate, which specifies the percentage of chromosomes that will be discarded from current population. This should, in theory, eliminate chromosomes of individuals that did not 'perform' in the test. The second parameter is the birth rate, which specifies the number of new chromosomes that will be added into the population. Obviously, the birth process involves creating new chromosomes and thus presents a new issue—how should these new chromosomes be created? A 'natural' way would be to create a new chromosome by 'mating' two individuals, which are among the fittest in current population, through a crossover process. This will logically increase the chances of producing better chromosomes by randomly mixing the characteristics from each chromosome. Lastly, but certainly not the least, in order to increase diversity within the population and to inhibit premature convergence, a mutation process can be applied to a portion of the newly created chromosomes. The number of new chromosomes affected by this is specified by the third parameter—mutation rate. GEASM proposes that the mutation process is done by altering one or more—but not more than half—connection weights in a chromosome. It is worthwhile to note that, in GEASM, the mutation process is actually part of birth process, since it will be applied to the chromosomes created using crossover process. This is to stress that the occurring mutation is actually through natural evolution process and not one that is forced upon current population.

As illustrated in Figure 3.4, the crossover process takes two fit chromosomes (i.e. known as parents) and divides each gene-string into two portions at a randomly selected point inside the encoded gene-string. This produces two 'head'

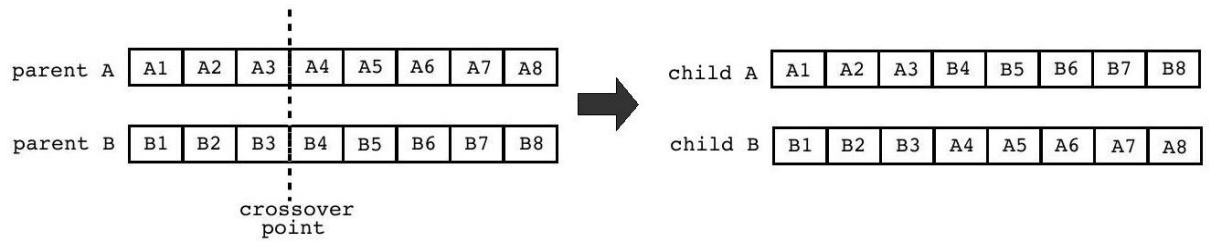


Figure 3.4: Process of crossover

segments and two ‘tail’ segments. The two tail segments for the chromosomes are then interchanged, resulting in two new chromosomes (i.e. known as child or newborns).

The mutation process in GEASM is done with the following rules:

1. A random number of genes (with at least one gene) will be chosen to mutate in each process. In order to ensure the information inherited from the parents will be preserved, the maximum number of genes allowed to mutate is limited to half of the gene size.
2. The mutation of the selected gene will be done by seeding it with a new random value (i.e. floating point number) with opposite polarity from the gene’s previous value (refer to Figure 3.5). This is to ensure that the new value assigned will represent a truly different weight strength (and polarity) of the respective connection.

Figure 3.5 shows a range of possible random values that will be generated for a mutated gene, which is between *min_value* and *max_value* (i.e. the values can be user-defined). The *median* of the two numbers will be used to separate the entire range of the value into two opposite polarities. For example, if the chosen range of possible random values are between -1 and 1, the *median* for the range would be 0. Based on this example, if the *new_value* generated is -0.25, and the *previous_value* is within the negative range, then the two values are in the same

polarity region. Therefore, the polarity of the *new_value* will be inverted using the procedure defined in Algorithm 2. In this case, the new value assigned to the gene would be 0.75.

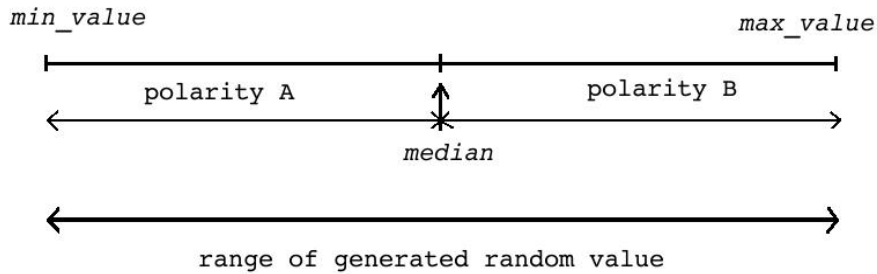


Figure 3.5: Range of random values

Algorithm 2 Range of random values

```

1: if new_value and previous_value < median then
2:   new_value = max_value - (new_value - min_value)
3: else
4:   if new_value and previous_value > median then
5:     new_value = min_value + (max_value - new_value)
6:   end if
7: end if

```

Do note that if the *new_value* and the *previous_value* of the gene is at the *median_value*, a new random value from either polarity could be assigned.

In brief, using the three genetic operators—namely selection, crossover, and mutation—a GA will be able to provide different results at different runs (i.e. stochastic). This allows a GA to have the ability to cover a huge search space (through random creations and crossovers) as well as being able to avoid the issue of local minima (through random creations and mutations) [120, 121, 122, 123]. In addition, instead of maintaining a single best solution applied by most conventional optimisation techniques, a GA maintains a population of solutions. Because of this, a better result may later be found among the population, possibly in other regions of the search space. As shown in Algorithm 1, this process will be repeated, generation after generation until acceptable solutions are found. Clearly, from the

procedures described, GA can be a very effective method to explore the solution space for optimising the weights of an ANN.

3.3.3 Using GA to Evolve a Neural Network

As mentioned earlier, in GEASM, the connection weights of the ANN are modified using GA in order to resolve the formulation of an ASM. Therefore, a chromosome has been encoded to represent a set of connection weights between neurons in consecutive layers (as illustrated in Figure 3.6), which are initially assigned with random values in the range $[-1.0 .. 1.0]$. In order to get a better dynamic range, the chromosomes need to be encoded with the connection weights values as it is—as floating-point numbers— instead of the more commonly used bit strings. Mathematically, the vector representing a chromosome can be written as:

$$U = (u_1, u_2, \dots, u_N),$$

where,

u_1, \dots, u_N are the connection weights of the ANN.

The chromosomes are then evaluated by creating individuals and putting them through a test, usually in the form of a simulated mission environment. A fitness value will be assigned to each individual and the population will be sorted based on these fitness values. For a more practical approach, a fitness function should be defined using data that is available in a real environment. Other than the fact that this is a more ‘natural’ way to implement a bio-inspired system, using this practical approach allows the possibility of this system to be trained in real-world environment. Without relying on data that is only available from the simulator, the system should be more adaptive and more robust to a changing environment where

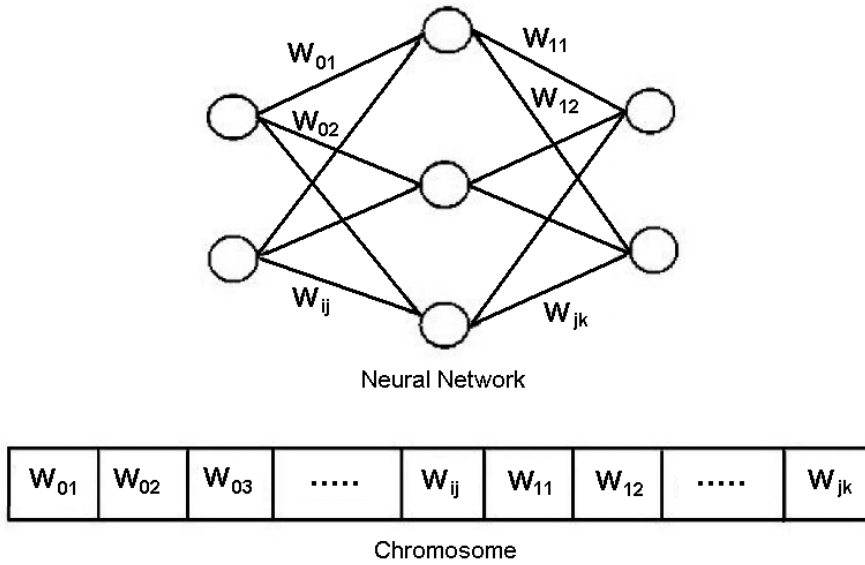


Figure 3.6: Chromosome representation

further evolution can be carried out if deemed necessary. This is consistent with the findings in a survey done in [71], where the effects of using prior information in creating a fitness function to be used in evolving robot controllers have been analysed.

In this thesis, a generic fitness function is proposed as follows:

$$fitness = f_{Err} * W_{Err} + f_{Eff} * W_{Eff} \quad (3.3)$$

f_{Err} is the component that keeps track of decision errors and f_{Eff} is the component that measures the performance of a solution to execute a given task.

Preliminary tests show that the f_{Err} term is vital, especially in the early generations, for a successful convergence of the genetic population towards a desirable solution. This is because performance measurements cannot reliably distinguish the candidate solutions when most (if not all) of them failed to complete their task. However, this component has to be carefully weighted so that it does not inhibit the evolution of the genetic population, which could affect healthy growth

towards target solution space. On the other hand, introducing f_{Eff} allows GA to guide the evolution of middle generations towards finding optimum solutions based on its performance for executing the given task. This is because when potential solutions start to dominate the population, the probability of making errors will be minimum and therefore the error component can no longer be reliably used to rate the fitness of a solution. Although performance measurement is the determining factor at later stages, it is important to maintain the error component for consistent fitness evaluation because new solutions will still be introduced to the population.

In GEASM, both f_{Err} and f_{Eff} are normalised values between 0 and 1. The normalisation is adapted to make the two variables comparable to each other. W_{Err} and W_{Eff} , weights for f_{Err} and f_{Eff} components respectively, are introduced into the equation in order to get a normalised fitness value. This also allows the system to rate the importance of each component based on on-going requirements. A more detail description of implementing GEASM on robot missions will be discussed in Chapter 4.

3.4 Summary

This chapter has discussed in detail the methodology to implement GEASM in a behavior-based system. Even though an MLP neural network and a basic GA have been employed in order to resolve the difficulties in developing a suitable ASM, several components of this framework are new. First, a control architecture is proposed to integrate an ANN and a GA in a behavior-based system. Second, a critical error suppressor has been introduced in the architecture in order to solve part of the problems when implementing a GA for robot control applications. Also, the output from the critical error suppressor will be used in the fitness function definition. Since designing a fitness function for a specific robot control application is not an easy task, a generic fitness function definition has been suggested in the thesis. Finally, a new rule for mutation operator has been proposed to improve the

search performance of the weights optimisation of a neural network. In order to test the feasibility of the proposed method, the implementation of GEASM in robot control will be discussed further in the following chapter.

Chapter 4

Implementation of Robot Control

Various aspects of implementing the proposed Genetically Evolved Action Selection Mechanism (GEASM) on a behavior-based mobile robot platform will be discussed in this chapter. That includes the selection of test environments and parameters, which need to represent an unknown or dynamic environment. In such environments, uncertainties may occur and might lead to problems associated with estimation and execution of the control system—this is exactly the reason why GEASM is proposed. In addition to that, the rationale of assigning suitable tasks to be performed in those environments for the evaluation of GEASM, will also be mentioned.

For testing in an unknown environment, quite a number of project opted for tests based on robot navigation tasks [124, 43, 52, 125, 126]. Robot navigation usually consists of three major competences: mapping, path planning and self-localization. However, in order to test a mobile robot in an unknown environment, these competencies are usually absent from implementation. This is done simply to prove that a particular robot control implementation, which is usually based on artificial intelligence (AI), can really perform navigation tasks without any ‘external’ guidance.

Two navigation-related test applications have been selected to evaluate GEASM: (1) search and exploration, and (2) target tracking. For the first test application (i.e. a search and exploration task), a robot is required to find a target object that may be located anywhere in the test environment. With randomly-placed obstacles in the test environment, the robots will also need to avoid those obstacles while searching for the target object. This should be sufficient in simulating an unknown environment for a mobile robot platform [8, 52].

Meanwhile, in the second test application (i.e. target tracking), a robot needs to continuously identify, locate, and follow a moving target object within an acceptable distance. Due to the fact that both robot and target object are free to move around the unknown environment, the target tracking mission can be considered as the best mission scenario to simulate a ‘dynamic’ world [127, 128, 129].

With these two test applications in mind, a suitable experimental platform needs to be determined next. Using genetic algorithm (GA) naturally requires offline training (due to the need to maintain many candidate solutions), which inherently enforces the use of simulation platform. However, the project also sets a future goal of implementing the system on real hardware mobile robot platform. Because of this, the chosen simulation platform used in the test phase must support hardware implementation.

4.1 Experiment Platform

As mentioned earlier, selecting a suitable test platform for GEASM will determine the approach that needs to be taken to implement the whole system. Since there is a need for hardware support, sensor selections will be limited by the ones offered by the selected platform. Therefore, data processing requirements will only be identified once a basic system has been finalized. This section will describe, in detail, the experiment platform that will be used in evaluating GEASM as behavior-based robot control system.

4.1.1 Simulation Platform

A robot simulator can actually facilitate the process of designing and experimenting a robot control system, sometimes by creating a very simple model of a platform and its environment. Using simple models usually means that either most parameters are abstract, or simple (and most of the time inadequate) mathematical models are used in order to allow the system to be simulated on low-end hardware. It is important to know that this will present a serious issue if and when the system needs to be implemented on a real robot hardware. As stated by Nolfi and Floreano [86], the problems in characterising the body of a robot and modelling the behavior of a sensor and an actuator are the main contributions towards the gaps between simulation systems and real robot systems.

In order to evaluate the effectiveness of GEASM as a robot control system, a robot simulator software called EyeSim [130] has been employed. EyeSim is a realistic mobile robot simulation system with 3D graphics, which provides adjustable error models and allows multiple robots to operate in the same environment. The main reason for using EyeSim is because the software exclusively supports EyeBot platform—a family of mobile robot implementation based on Eyebot controller board that is geared towards applications of image processing.

Using C/C++ programming language, the simulator has been designed to support running programs created using the exact same source code that is written for its equivalent robot hardware [130, 131, 132]. Quite a number of robot applications have been successfully transferred from simulation to a real robot using the EyeSim platform. One example has been reported in [133] where a robot control system has been implemented for a robot soccer application. Utilizing the power of object-oriented programming (OOP), the implementations for GEASM and the two selected test applications has been written in C++, targeting the EyeBot mobile robot platform. However, the experiments presented in the thesis will be mainly based on software simulations. It is worthy to highlight that an obvious advantage

of using a software simulation platform to experiment with the proposed method is the execution time. While it may take minutes to complete a task on a mobile robot, with proper modelling and abstractions, a software simulation system may complete a number of trials in the same amount of time with acceptable accuracy.

Another advantage of using a simulation platform, such as EyeSim, for this particular project is apparent due to the use of GA in training the artificial neural network (ANN) engine in GEASM. Since GA maintains a population of candidate solutions, the training process usually involves a huge number of trial runs before GEASM can learn a particular strategy. A fast mode simulation offered in EyeSim also adds an extra advantage to the testing phase, allowing more work on exploring the evolutionary technique for robot control with more efficiency. Since the aim of this research is to allow the overall control system to interact intelligently in a dynamic or unknown world, using EyeSim is the perfect choice because it allows a larger experiment space to be manipulated in creating various testing environments.

4.1.2 Robot and Sensor Model

The target robot platform for GEASM is based on a wheeled robot known as SoccerBot (shown in Figure 4.1). Similar to the other robots in the EyeBot family, it is mainly equipped with two DC driving motors with differential steering and an EyeCam camera for vision. Sensors such as shaft encoders, tactile bumpers or infra-red (IR) can be added to the robot platform depending on the immediate requirements of an assigned task. It should be noted here that, using EyeSim, each sensor can be coupled with a user-customizable error model that allows a simulation to be executed with a more realistic figures. This error model is actually used to test GEASM's system tolerance towards unexpected minor glitches within data stream—more details on this will be discussed in the following chapter.

Deciding upon which sensor to use for evaluating GEASM also requires some planning. It is easy to get trapped into the idea of simply putting every

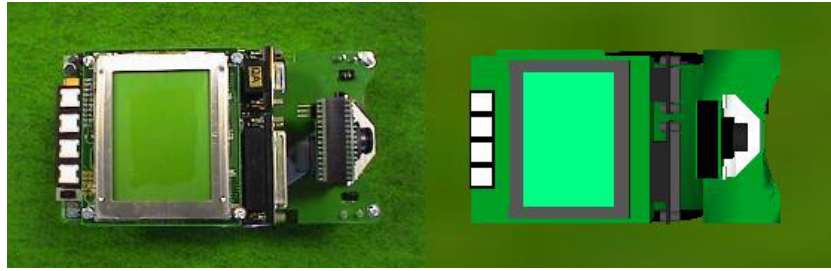


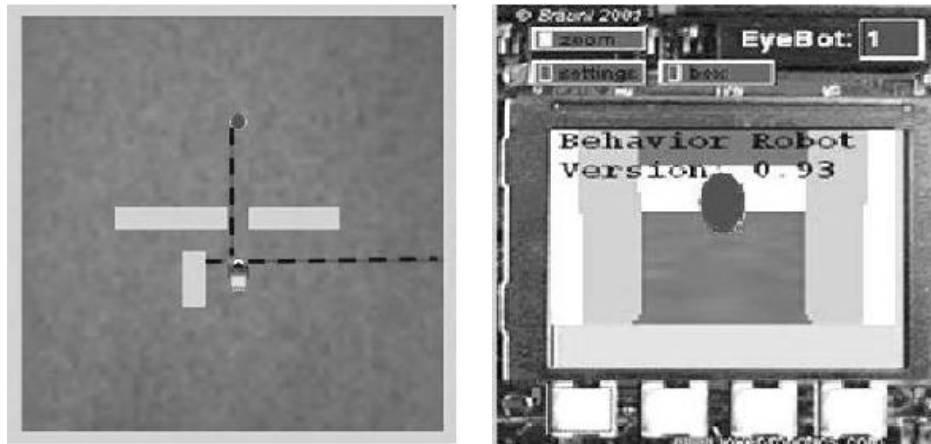
Figure 4.1: Snapshots of a real and its equivalent simulated EyeBot platform

available sensors onto the system, but other than costs involved, the data processing time also needs to be considered. So, it is always best if a robot is equipped with only sensors that are really needed by the task at hand. Since the tests mainly involves moving around or following objects, the sensors must be able to give information about its environment as well as any existing objects.

Consistent with the fact that the EyeBot platform promotes the use of image sensors (i.e. camera), the target robot platform for implementing GEASM will be fitted with a colour camera that should be able to provide any required information of its surroundings. For the purpose of this text, the customized platform for GEASM will be referenced as GE-Bot. The use of image sensors on GE-Bot will enable it to not only detect the presence of obstacles or target objects, but even monitor any dynamic changes in an environment.

However, similar to human vision, a camera has limited scope in terms of area coverage (i.e. field of view and line of sight). Because of this, GE-Bot will also be equipped with IR sensors that should help improve the system’s awareness towards physical obstacles that may cause the platform to behave erratically. Since IR sensors are directional, GE-Bot will be using three of them—front, left and right. Because of this, the resolution at which the IR sensors can detect obstacles is slightly at a disadvantage and therefore it is important both sensors can complement each other. A simple scenario that exhibits the importance of combining the two type of sensors can be illustrated in Figure 4.2.

As shown in the figure, consider a scenario where the robot platform is situated as such in the operating environment. Due to the directional nature of the



(a) Dotted lines illustrate the output from the IR sensors (b) Display from the camera view

Figure 4.2: Robot is facing a targeted ball with obstacles around the robot

IR sensor, the front sensor will be informing the platform that there are no obstacles in that direction—which is obviously false. Fortunately, with proper processing on the image information provided by the image sensor, the obstacles in front of the robot can be detected. On the other hand, due to the limited field of view inherent in the image sensor, the existence of an obstacle on the left of the platform was not detected by it—but was instead detected by the IR sensor on the left.

Some brief descriptions of the two sensors equipped on GE-Bot are as follows:

1. EyeCam Camera: EyeCam is a digital camera that provides an image with a resolution of 60x80 pixels with 24 bit colour per pixel. The EyeBot controller is capable of extracting an image sequence of up to 30 fps (frames per second) from the EyeCam. One advantage of using EyeSim is the ability to simulate this on-board camera available on EyeBot platform. This allows real-time image processing to be performed in the simulation platform.
2. IR Distance Measurement Sensor: This sensor consists of an IR transmitter and an IR receiver which are used to measure the distance of the nearest

obstacle based on the reflected IR beam. A look-up table will be used to transform the raw data values to the actual distance values. Since the sensor is cheap and the size is small, this sensor can be freely positioned and oriented around a robot. However, in this implementation, the sensors are mounted at three positions (i.e. front, right and left), which are considered to be adequate for the experiments required in this project.

4.1.3 Pre-processing of the Sensory Data

One of the reasons why many systems developers are not so keen on utilizing vision sensors is due to the fact that processing requirements for such systems are very demanding. In order to overcome this, some implementations opted for lower resolution or grayscale image processing, which clearly reduces the amount of input image data. Another method that can be used to ease up the processing requirements is to create abstract representations of the required information and use simple image analysis to translate captured image data [134, 135, 136]. The simplest and most commonly used information abstraction in image processing is by using colour to represent specific object in the environment.

In this experimentation, the implemented system will be using ‘Colour Object Detection’, which has been presented in [130], as the basis for GE-Bot’s image analysis sequence. Generally, the object detection method (which is based on hue-histogram algorithm) has been extended to gather more information on the robot’s environment. The overall image analysis is done based on defining specific regions on the input image to represent useful information of the surrounding environment. Figure 4.3 shows how this is done.

As shown in Figure 4.3, the image has been segmented to clearly define four main regions representing the left-side region, the right-side region, the near front-side region, and the centre front region of the robot’s view. The low horizontal line is used to define the boundary for ‘near’ front side region, which indicates

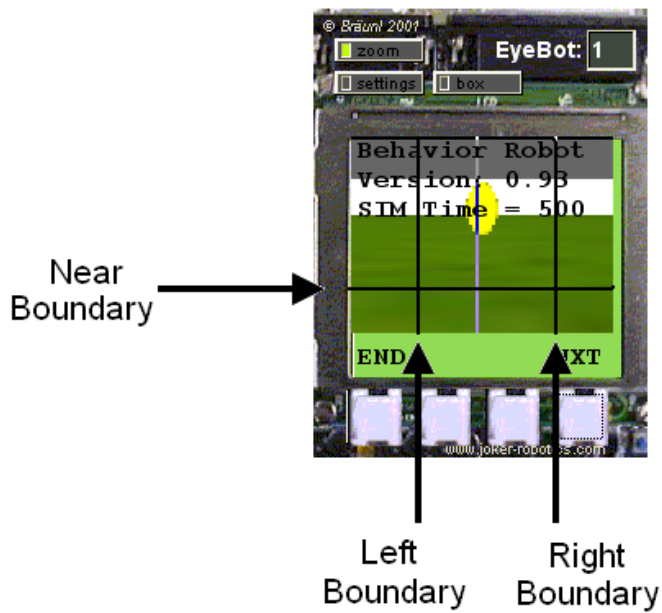


Figure 4.3: Pre-processing from camera image

whether the robot is near to the target object or an obstacle (walls). Similarly, the other two vertical lines close to both sides of the image define boundaries for the left, right, and centre regions. This segmentation method is used as the basis for image analysis sequence used by GE-Bot to create abstract representation of its surroundings.

The abstracted information used by GE-Bot is as listed below:

1. **bview**: This parameter represents the size of target object in the robot's view. This value is actually the maximum number of target object's pixels detected in a column of the camera's image.
2. **bpos**: This parameter represents the direction of a target object in the robot's view with reference to the centre point. Thus, value of 0 in this case indicates that the target object is exactly at the centre while negative and positive values indicate that the target object is on the left and right side of the robot's view respectively. It is an offset value of the column used for **bview**.

3. **bnear**: This parameter represents how near the target object is in the robot's view. It is the maximum number of target object's pixels detected in the near front region (under the horizontal line shown in Figure 4.3). In this case, the maximum number will be chosen among three possible values: the right-side, the centre, or the left-side within this region.
4. **sideL**: This parameter represents detected obstacle at the left-side of the robot. It is the number of white pixels (representing an obstacle) detected at the left side of the camera's view, in the 'near region'.
5. **sideC**: This parameter represents detected obstacle at the near front-side of the robot. It is the number of white pixels (representing an obstacle) detected at the near front-side of the camera's view.
6. **sideR**: This parameter represents detected obstacle at the right-side of the robot. It is the number of white pixels (representing an obstacle) detected at the right side of the camera's view, in the 'near region'.


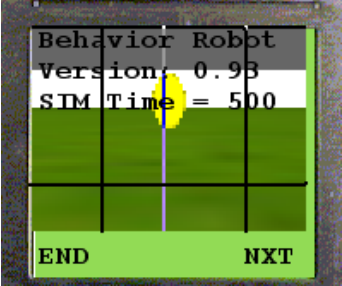

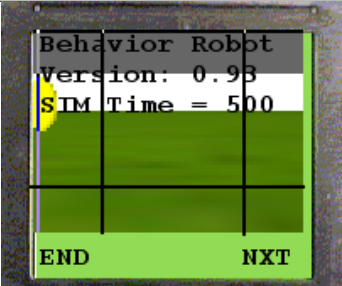

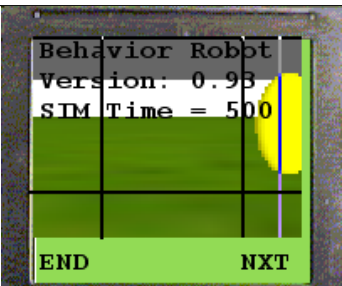

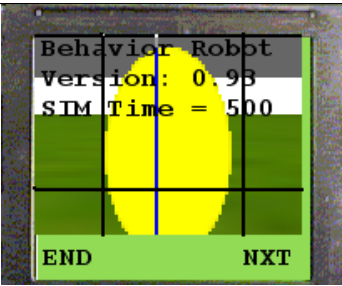

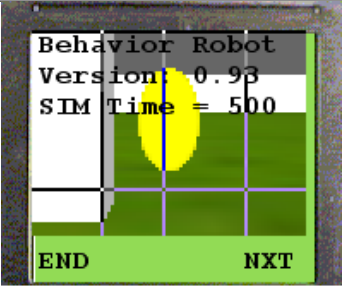
Table 4.1 demonstrates some examples of generated abstract data representing the existence of a target object at various locations within the robot's view.

In addition to the information from camera image, readings from the IR sensors (i.e. located at three positions) will be encoded with either '1' or '0', which represents the existence of an obstacle or otherwise. As shown in Figure 4.4, the encoding is based on a pre-defined safe range between a robot and an obstacle.

4.1.4 Simulation Environment

As mentioned in the previous sub-section, the implementation of GEASM on Eye-Sim revolves around data abstraction based on colour. The environment (i.e. world) for the two missions is a green carpeted arena surrounded by white walls. There can be other walls (or objects) within the arena to represent environmental obstacles, which also enable the creation of maze-like worlds. The size of operating

Table 4.1: Examples of data extraction from the camera image

Robot's Environment	Camera's View	Camera Sensory Information
		<p>bpos = 0; bview = 22; bnear = 0; sideL = 0; sideC = 0; sideR = 0</p>
		<p>bpos = -35; bview = 22; bnear=0; sideL =0; sideC=0; sideR=0</p>
		<p>bpos = 40; bview = 16; bnear= 0; sideL =0; sideC=0; sideR=0</p>
		<p>bpos = -3; bview = 56; bnear= 60; sideL =0; sideC=0; sideR=0</p>
		<p>bpos = -1; bview = 30; bnear=0; sideL =190; sideC=41; sideR=0</p>

```

void SVisBot::PSDMeasure(void)
{
    mSensor.valid = 0;
    if(PSDCheck()!=TRUE) /* check validity */
    {
        mSensor.valid = 1;
        /* get readings */
        mSensor.f_dist = PSDGet(mStat->front);
        if(mSensor.f_dist>mSensor.saferange) mSensor.f_safe = 1;
        else mSensor.f_safe = 0;
        mSensor.l_dist = PSDGet(mStat->left);
        if(mSensor.l_dist>mSensor.saferange) mSensor.l_safe = 1;
        else mSensor.l_safe = 0;
        mSensor.r_dist = PSDGet(mStat->right);
        if(mSensor.r_dist>mSensor.saferange) mSensor.r_safe = 1;
        else mSensor.r_safe = 0;
    }
}

```

Figure 4.4: A snapshot of a program to encode the IR readings

environment used by the robots may vary depending on mission objectives and required complexity. In all missions, the performance of GEASM-based mobile robot platform will be tested in various unknown environments where any information on surrounding environment and object locations will not be available to the system.

4.2 From Theory to Implementation

In this section, the implementation of GEASM control architecture shown in Figure 3.1 will be further discussed for search and exploration and target tracking mission. Among the first things to do in implementing the proposed system is to develop behavior modules that is consistent with the available sensors. Sensor data obtained from the robot's camera (colour image), and IR sensors (located at the left, right, and front of robot's body) will be processed and used by the behavior modules.

Three basic behavior modules have been defined separately for search and exploration and target tracking missions respectively:

Behaviors for Search and Exploration

1. **Avoid obstacle:** Manoeuvres the robot to avoid obstacles (based on input from IR sensors and camera image)
2. **Wander:** Allows the robot to move in a random direction (slightly influenced by results from camera image analysis)
3. **Drive to Target:** Drives the robot towards target object (based on the colour object detection method [130])

Behaviors for Target Tracking

1. **Avoid obstacle:** Manoeuvres the robot to avoid obstacles (based on input from IR sensors and camera image)
2. **Predict:** Predicts the next movement of the target object based on input from the camera image
3. **Track:** Drives the robot to keep the moving target object in front within view

There are no specific rules for designing behavioral modules for a behavior-based system. However, it is commonly defined in order to perform a particular activity, representing a sub-task of an overall robot mission. Therefore, the three defined behavior modules for each application are considered to be sufficient for a behavior-based system to perform the respective missions.

For GEASM implementation, the processed data (output of behavior modules) is then fed as input, along with direct sensor outputs, to the ANN, which will act as the action selection mechanism (ASM) engine. As described previously, the ANN will consider all information from the behavior modules and produces an appropriate action (i.e. distance and angle) for the actuator to control the movement of the robot. However, just like for any other system based on ANNs, a sufficiently suitable action can only be generated after the ANN has gone through

an acceptable amount of training. Without any prior information available to the system, the ANN in GEASM architecture will be trained using GA.

In order for the GA to train the ANN, a generic fitness function has been defined previously by Equation 3.3. For this specific implementation, the equation is further refined with the error component, f_{Err} , being defined by Equation 4.1. As mentioned earlier, this term is used to represent decision errors (e.g. moving in circle, driving into obstacles, stopping indefinitely) made by GEASM, which creates a seemingly harmless action, but in reality does not contribute to the success of overall mission. Such errors generated by the ANN engine in each decision cycle is accumulated into $\varepsilon(k)$. Naturally, the value for f_{Err} is normalized within range between 0 and 1.

$$f_{Err} = \frac{t - \sum_{0 < k \leq t} \varepsilon(k)}{t} \quad 0 < t \leq \tau \quad (4.1)$$

where,

- $\varepsilon(k)$ error penalty value function;
- k decision cycle;
- t total time steps to execute the mission;
- τ maximum time limit for robot to operate;

The defined fitness function can vary from one application to another via the component f_{Eff} , depending on how the performance of a test platform executing a particular task can be measured. The performance term, f_{Eff} , for search and exploration; and target tracking is described in Equation 4.2 and Equation 4.3 respectively.

$$f_{Eff} = \frac{\tau - t}{\tau} \quad 0 < t \leq \tau \quad (4.2)$$

The component f_{Eff} for the search and exploration mission has been defined as a measure of how fast the system can explore the environment and reach the target object within a given time τ . This is simply done by measuring the number of decision cycles needed to complete the task. Using decision-cycle count instead of real time value is actually an advantage since the results can be translated to platform with more processing power.

$$f_{Eff} = \frac{\sum_{0 < k \leq t} P(k)}{\tau} \quad 0 < t \leq \tau \quad (4.3)$$

where,

k	decision cycle;
t	total time steps to execute the mission;
τ	maximum time limit for robot to operate;
$P(k)$	tracking point function;

On the other hand, the component f_{Err} for the target tracking mission is defined by the ability of the system to keep the moving target that is being tracked within its field of view. This is done by accumulating reward and penalty values for $P(k)$ in every decision cycle k . A reward is given to the system when it manages to keep the target within its field of view—otherwise the system will be penalised.

Referring to Equation 3.3, weight values W_{Err} and W_{Eff} should be set to for each error and performance components respectively. In this thesis, the weights W_{Err} and W_{Eff} are both set to 0.5, which indicate that the contribution for fitness values from both components (i.e. f_{Eff} , and f_{Err}) are considered to be equally important.

Table 4.2 shows the parameters used in the genetic evolution for optimising the weights of the ANN. In general, there are no universal parameter settings for GA that can be used, as it may vary depending on the problems and approaches

Table 4.2: Initial parameter settings for optimising the weights of a neural network

Parameter	Value
Death rate	0.400
Birth rate	0.600
Mutation rate	0.400
Initial population	20
Gene type	float
Min gene value	-1.00
Max gene value	1.00

for implementing the GA itself. In this research, the roles for birth rate and death rate have been tuned to control the population growth. Many researchers have discussed the issue related to optimum population size and growth [137, 138, 139]. However, since the initial population consists of a set of randomly created chromosomes, having a large population at this stage may not be necessary (i.e. as it may lead to a longer training phase). However, when the population evolves, it is ideal to have a larger number of candidate solutions so that the search space can be sufficiently covered.

Based on tests for a simpler mission where a robot simply needs to move straight to a target, it is found that having a starting population of 20 chromosomes is adequate. This value is then used in a few other missions and has been found to be an adequate initial population size as well. As explained earlier, the population size is then allowed to continuously grow during the first ten generations by setting the value for death rate and birth rate with 0.4 and 0.6 respectively. This effectively translates into a 20% population growth. The size will then remain unchanged in the following generations by setting the same value for death rate and birth rate. Again, there are no rules for this and a value of 0.4 has been chosen based on common logic. A value of more than 0.5 would create a relatively whole new population set for each generation, and a value closer to zero would not generate a population that is dynamic enough to cover a wide search space. This value of

0.4 has been found to work well with many test missions. Similarly, a mutation rate of 0.4 has been chosen to introduce a more diverse search pattern, while still maintaining proven chromosome parts from fit parents in the crossover process.

To summarise, the GA presented here allows three groups of chromosomes to exist in the population after an evolution: chromosomes of the fittest individuals from the previous generation, unmutated newborns generated from the crossover process, and mutated newborns (also from crossover process). Using the parameter settings described earlier, the ANN implemented for each test applications has been trained for 30 generations.

4.3 Justification of Selected Design Parameters

This section presents an elaboration on some of the design decisions made in order to implement the proposed system. The first involves the determination of the number of neurons required by the hidden layer in the neural network. Also, the contribution of error component in both architecture and fitness evaluation will be given a closer look. While the final item will be on the effect of manipulating the weights of the parameters in the fitness function.

4.3.1 Size of Hidden Layer

This parameter basically determines the ability of the ANN to learn and perform the required task. The hidden layer can be seen as the core processing elements of an ANN, and therefore determines the complexity of the network. As with many intelligent systems, a simple system is usually unable to look beyond simple logic. In other words, if the number of hidden neurons is inadequate, the ANN may not be able to recognise complex patterns. This is also known as an underfitting scenario. At the other end of the spectrum, if the system tries to be too sophisticated it will attempt to look for patterns in pure random data and may ultimately report

false positives. This is expectedly known as overfitting. Based on this knowledge, a balanced number of hidden neurons needs to be determined in order to produce an ANN with satisfactory performance.

Some research has been proposing a dynamically adaptive neural network that is capable of growing its hidden layer as needed. Algorithms like Cascade Correlation [140] start with a minimal network, then add hidden nodes accordingly during training. This can make experimental setup simpler and theoretically produce better performance because the number of hidden nodes will always be optimised. However, one problem of using this technique is to decide when the ANN should stop increasing its size. Although there is a possibility that the ANN could be allowed to grow bigger for a better performance, it is also important to know when to stop before overfitting occurs.

Alternatively, a more rigorous method known as pruning can be applied, where the neural network is tuned during training by removing nodes whose weight values are approaching zero. This is consistent with the fact that a zero-valued weight represents a disconnected (and therefore, unneeded) synapse. This will understandably produce an even more optimised network but it comes with a huge training commitments.

A more practical solution that is used by many (and sometimes considered as the rule of thumb for this matter) is to use a trial-and-error method with logical considerations by looking at the number of nodes in the input layer (incoming signals) and those in the output layer (outgoing signal). A good starting point would be to choose a size that is between the size of the input layer and the output layer [141]. Other guesses include using a fraction of the total size of neighbouring layers and a value that is up to twice the number of input nodes.

For the proposed system, a simple analysis has been made using the search and exploration mission to find a suitable number of hidden nodes. The system has been trained using 1, 5, 15, and 50 hidden nodes. The performance

(best fitness) value of every generation in each scenario has been charted for 30 generations, as shown in Figure 4.5. The system with one hidden node demonstrates underfitting scenario (not reaching acceptable fitness) while the system having 50 hidden nodes shows signs of overfitting (not achieving steady state). Both systems with five and 15 hidden nodes show an acceptable fitness convergence pattern and this is consistent with the first choice in the ‘rule of thumb’ described above, which recommends taking a value between the size of the input layer and the output layer. It is worthy to note here that, from the figure it can be seen that the system with 15 hidden nodes managed to converge faster compared to the one with five hidden nodes.

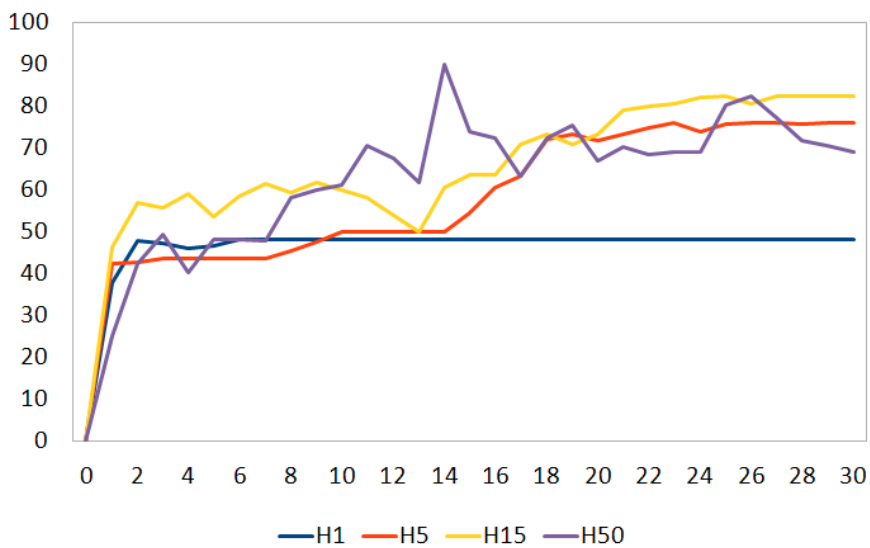


Figure 4.5: Analysis of hidden node for feed-forward neural network

Although the proposed system is implemented using 15 hidden nodes in most experiments, it should be highlighted here that having a smaller number of hidden nodes would mean a less complex system and therefore be more practical for a real hardware implementation.

4.3.2 Error Term Contribution

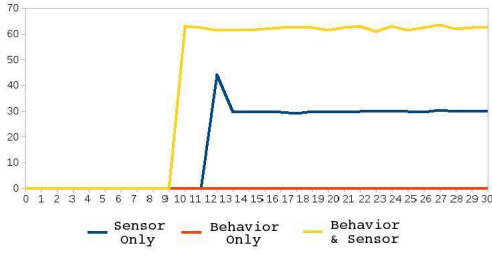
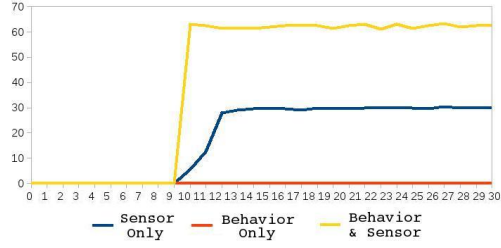
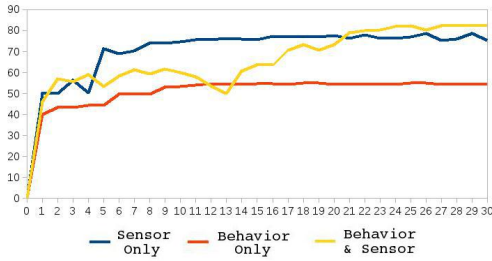
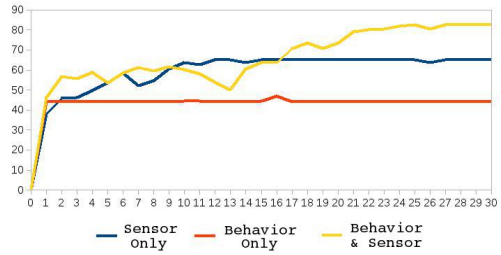
As mentioned earlier, the error term for fitness evaluation was deemed necessary and plays an important role in the success of genetic evolution of the proposed system. It is important to note that the decision errors flagged by the system do not change any output produced by the GEASM control architecture (except for critical errors that are suppressed by the suppressor module shown in Figure 3.1). This ensures that the fitness of a candidate solution is really valid with the platform running its course of action whatever it may be. In any case, this section will attempt to investigate the real contribution of the error term as proposed in the GEASM control architecture.

A simple experiment has been conducted based on the search and exploration mission training. The procedure is first done for the proposed ANN, while in the second run, the ANN is stripped from all error-related modules including the error suppressor module. In both instances, three sets of input data were used:

1. Sensory information (conventional ER approach);
2. Behavior suggestions (i.e. angle, and distance);
3. Sensory information and behavior suggestions (as proposed in GEASM architecture)

The experiment has been repeated using two different ANN hidden node counts. First, the ANN is built with 15 hidden nodes (i.e. $H=15$). Next, the number of hidden nodes is set with an equivalent number of the input nodes (i.e. $H=I/P$).

The results in Figure 4.6 show that the ANN with an error term can converge to a satisfactory fitness value faster compared to the one without. This is consistent with the fact that a learning process is usually more effective when a teacher is available to point out mistakes (errors). Notice that the network without an error term does not ‘perform’ at the early stage of genetic evolution (virtually

(a) Without the error suppressor ($H=15$)(b) Without the error suppressor ($H=I/P$)(c) With the error suppressor ($H=15$)(d) With the error suppressor ($H=I/P$)**Figure 4.6:** The effect of the error suppressor

zero fitness). This is also expected since it is quite impossible for a randomly-driven (no patterns in the first few generations) robot to achieve its goal. Without the error term to evaluate its fitness, almost all genetic population may be assigned to a virtually zero fitness value. From this experiment, it is clear how much the error term contributes towards the success of the propose GEASM control architecture.

4.3.3 Weighting Fitness Components

As mentioned earlier, the weights of both components (i.e. W_{Err} , and W_{Eff} in Equation 3.3) used in fitness evaluation have been set to 0.5 each, which indicate that both components are equally important. However, it is interesting to know the effect of using other weight ratios for fitness evaluation. Note that W_{Err} and W_{Eff} are introduced to get a normalised fitness value, which limits the total of the two weights to be within the range of 0 and 1. Figure 4.7 shows the trend of fitness values during the training phase over 30 generations with the following $W_{Err}:W_{Eff}$ weight ratio:

1. 9:1 (i.e. $W_{Err} = 0.9$; $W_{Eff} = 0.1$)
2. 7.5:2.5 (i.e. $W_{Err} = 0.75$; $W_{Eff} = 0.25$)
3. 5:5 (i.e. $W_{Err} = 0.5$; $W_{Eff} = 0.5$)
4. 2.5:7.5 (i.e. $W_{Err} = 0.25$; $W_{Eff} = 0.75$)
5. 1:9 (i.e. $W_{Err} = 0.1$; $W_{Eff} = 0.9$)

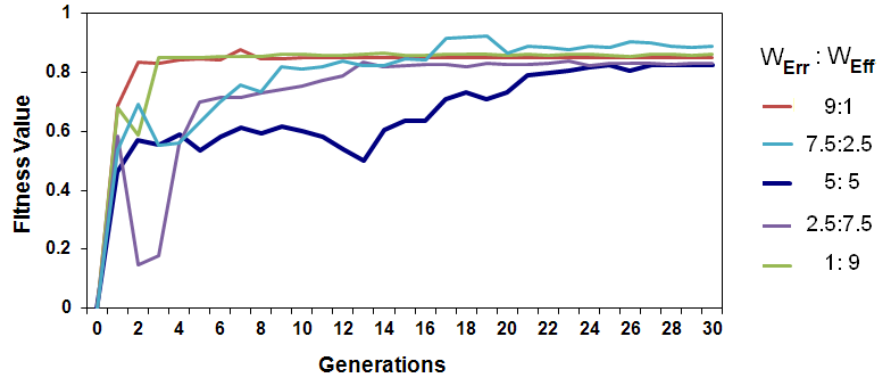


Figure 4.7: Best fitness value when training with different weighting fitness components

From the figure, it is clear that the best fitness values for all configurations have converged to good results of around 0.80 after 30 generations. A closer look into it shows that the systems with ‘unbalanced’ weight ratios (9:1 or 1:9) seem to converge much faster (takes less than five generations). The systems with slightly ‘unbalanced’ weight ratios (7.5:2.5 and 2.5:7.5) converge after about 15 generations, while the proposed balanced weight ratio only converges towards the end of the 30 generations’ evolution. This shows that the convergence rate of the best fitness value seems to be affected by changing the weight ratios. However, it should be noted that this is simply an evaluated fitness value, which may not indicate true performance if the evaluation is flawed.

Another way to look at the performance is to view the path traces of the robot while executing its mission scenario. This is shown in the Figure 4.8.

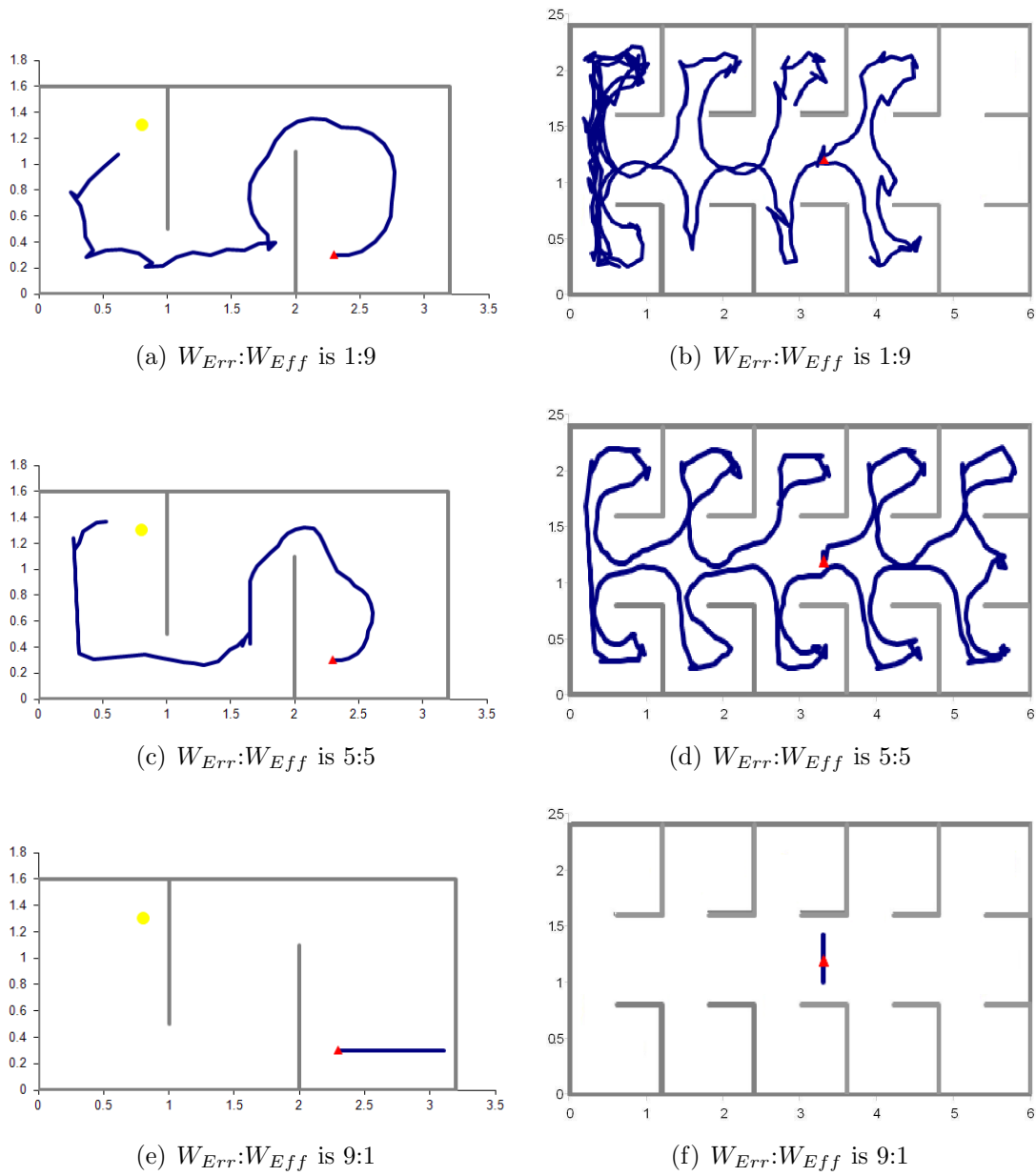


Figure 4.8: Path traces of GEASM when training the system with various weight ratios

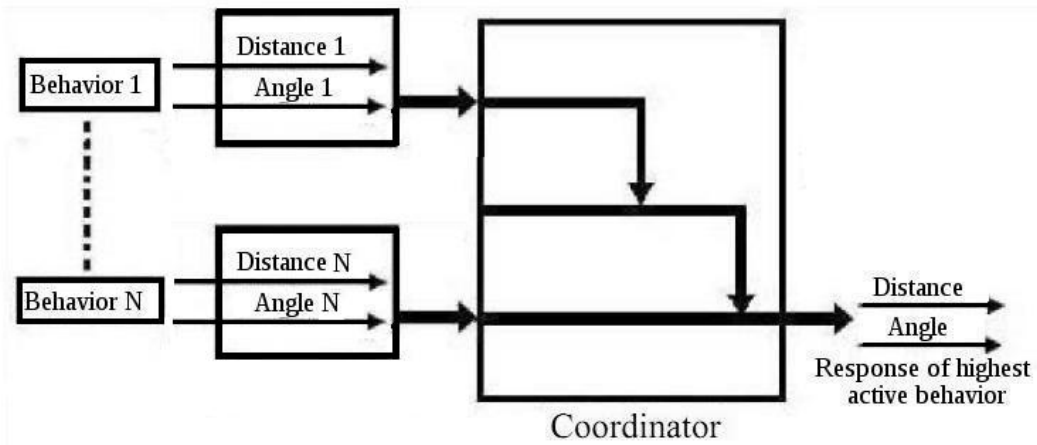
From this new point of view, it is clear that while the evaluated fitness value is achieved for weight ratios other than the balanced 1:1 ratio, the system failed to successfully execute its mission. They are actually stuck in a ‘local minima’ solution, a problem which is common in gradient-based learning systems (some texts may refer to this condition as ‘premature convergence’) [142, 143, 144, 145]. A quick assumption that can be made from this result is that in order for a GA to successfully evolve its population towards a global solution, a properly selected fitness evaluation method (in this case, its component) is of primary importance.

4.4 Considerations for Performance Comparison

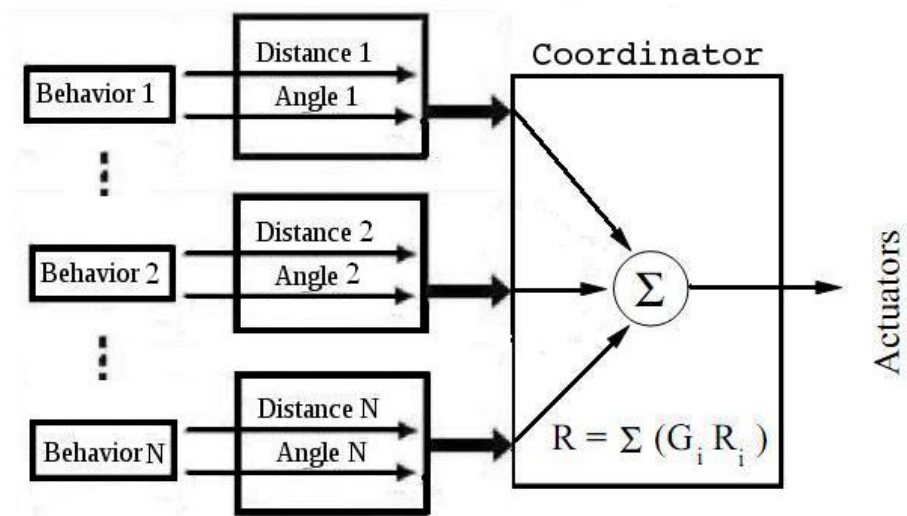
In order to examine the effectiveness of GEASM in coordinating behaviors, priority-based (arbitration type) and vector summation (command fusion type) methods have been chosen for a performance comparison. Subsumption logic [10] uses a priority-based (PB) method while motor schemas [146] architectures use a vector summation (VS) technique. Both priority-based and vector summation methods are among the classic, acknowledged techniques that are mostly discussed in the field of behavior-based system. Furthermore, the two techniques are suitable for the implementation of a low level ASM, which is part of the main focus of this research work.

The block diagrams of priority-based and vector summation are illustrated in Figure 4.9. Notice that the same behavior definitions have been used in all methods.

Using the priority-based method, each behavior has been assigned a priority value and an active flag. The active behavior with highest priority will control the system by overriding the other behaviors in a decision cycle. The box in the priority-based approach represents the override switch that could be implemented with multiple stage multiplexers, in which the active flag of a behavior (when activated) will take over control from a lower priority behavior.



(a) priority-based



(b) vector summation

Figure 4.9: Block diagrams of action selection mechanisms


```

int SBhvBot::Evaluate(void)
{
    robomove* cDrive = NULL;
    list<SBehavior*>::iterator pBehavior = mBhvList.begin();
    int cSelected = RBHV_ERROR;

    while(pBehavior!=mBhvList.end())
    {
        (*pBehavior)->Evaluate();
        cDrive = (*pBehavior)->GetDrive();
        cSelected = (*pBehavior)->GetBehaveID();
        if((*pBehavior)->IsActive()) break;
        pBehavior++;
    }
    SetDrive(*cDrive);

    return cSelected;
}

```

Figure 4.10: A snapshot of a program for implementing the priority-based method

In this experiment, a custom fixed-priority arbitration has been implemented. Figure 4.10 shows a snapshot of a program to implement the priority-based technique using C++ language. Even though a variable priority can be assigned to the behaviors during the execution of the system, several conditions should be considered. As stated in [147], to build a system that uses a variable priority, it requires another mechanism to determine the priority ordering. The issue here is, what is that mechanism? Another concern is how to ensure two or more different behaviors will not be having a similar priority at a particular decision making cycle? Since the priority can be shifted among the behaviors from time to time, building such a system will become more complicated especially when relevant conditions can not be outlined.

Unlike the priority-based method, vector summation will allow every behavior to contribute in producing the overall system output. At the robot's current location, each of the behaviors will produce an output vector indicating the response or direction where the robot should move to satisfy each of its goals. As shown in Figure 4.9(b), each of the output vectors (R_i) is multiplied with its behavior gain factor (G_i) and summed up with the rest to produce a single output vector. In this thesis, the gain factor for every behavior has been defined as *one per*

```
int SBhvBot::Evaluate(void)
{
    robomove* cDrive = NULL;
    robomove cDriveAverage;
    cDriveAverage.dist=0.0;
    cDriveAverage.angle=0.0;
    int cCount=0;
    list<SBehavior*>::iterator pBehavior = mBhvList.begin();
    int cSelected = RBHV_ERROR;

    pBehavior++;

    while(pBehavior!=mBhvList.end())
    {
        (*pBehavior)->Evaluate();
        cDrive = (*pBehavior)->GetDrive();
        cDriveAverage.dist+=(cDrive->dist);
        cDriveAverage.angle+=(cDrive->angle);
        cSelected = (*pBehavior)->GetBehaveID();
        LCDPrintf("D%d=%.2f;A=%.2f\n",cSelected,cDrive->dist,cDrive->angle);
        pBehavior++;
        cCount++;
    }
    cDriveAverage.dist/=cCount;
    cDriveAverage.angle/=cCount;
    LCDPrintf("Dav=%.2f;Aav=%.2f\n",cDriveAverage.dist,cDriveAverage.angle);

    SetDrive(cDriveAverage);

    return 0;
}
```

Figure 4.11: A snapshot of a program for implementing the vector summation method

number of behaviors. In other words, the produced output vector is actually the average response from all behaviors in the system. Therefore, for the case where all the behaviors are suggesting the same response (i.e. identical in magnitude and orientation), the output vector that controls the overall system is equivalent to the vector suggested by each of the behaviors. The snapshot of a program to implement vector summation is shown in Figure 4.11.

Applying GA for robotic application is closely related to the work in evolutionary robotics (ER). Therefore, the effectiveness of the proposed method will be also compared with a conventional implementation of ER (refer Figure F.1(b) in Appendix F). This has been carried out to investigate the relationship between the two techniques and to highlight any advantages of the proposed approach. In this thesis, the conventional ER has been implemented using the same sensory input and has been applied to the same fitness function definition during the training phase.

4.5 Summary

This chapter has laid out the details for implementing GEASM architecture in controlling a behavior-based robot for two different test applications: search and exploration task, and target tracking missions. This includes the description on how to define the fitness function for both test applications. An explanation on parameters used when employing GA and ANN have also been discussed. In addition, preliminary studies have been conducted in order to investigate some of the parameters used to evolve the ANN for GEASM. The outcome of these studies will be used as the basis for the experimental setup for both test applications presented in the thesis. The proposed approach is evaluated in various simulation scenarios, and the results will be presented in Chapter 5.

Chapter 5

Results and Discussions

Using experiments mentioned in Chapter 4, this chapter examines the performance of a Genetically Evolved Action Selection Mechanism (GEASM) system, which is basically a mobile robot system utilizing GEASM as part of its control architecture. The first two sections investigate the effectiveness of GEASM system in the aforementioned test applications—search and exploration, and target tracking. In both test applications, the mission environment is a finite-sized area but is completely unknown to the mobile robot system. For each test application, the measured performance of GEASM system will be compared against four other implementations—priority-based (PB), vector summation (VS), and conventional evolutionary robotic (ER).

The following section then investigates the capability of the GEASM system to execute a robot mission under the existence of noise within its input sensors. In addition to that, the use of the GEASM system as sub-modules in order to execute a higher level robot tasks is also explored.

5.1 Search and Exploration

Figure 5.1 illustrates the simulated training environment designed to train the artificial neural network (ANN) controller for GEASM, which will then be used in the search and exploration mission. Figure 5.2 shows the trend of fitness values during training, captured over 30 generations. The solid line and the dotted lines denote the best and average fitness values, respectively. In theory, the best or average fitness value of the population is expected to increase over generations, and converge to the point where an optimal solution can be found. From Figure 5.2, it is clear that the best fitness value has converged to a satisfactory value of around 0.83. Remember that the fitness value, in this case, will never reach 1.0 because the time needed to get to the objective, t can never be zero (refer to Equation 4.2) due to the nature of the task and initial mission environment.

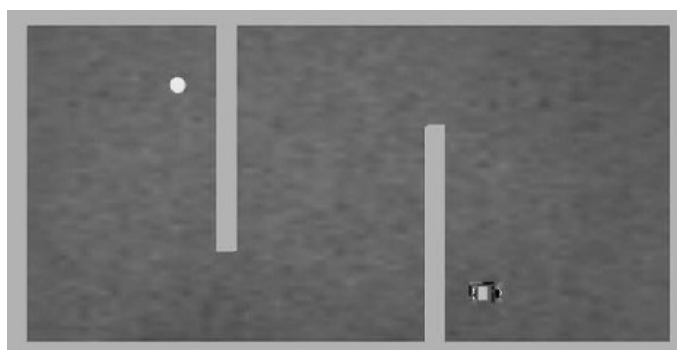


Figure 5.1: The simulation training environment for search and exploration mission

The fully evolved GEASM system is then tested for a search and exploration task in two simple environments as shown in Figure 5.3. When comparing the GEASM system with PB, VS, and ER systems, it is observed that all the four systems have performed the search and exploration task, approximately with a good performance.

One thing to notice about this result is the fact that the output of all control architectures seem to converge towards using wall following (WF) movement pattern. WF robots, which can be implemented using simple logic, are quite

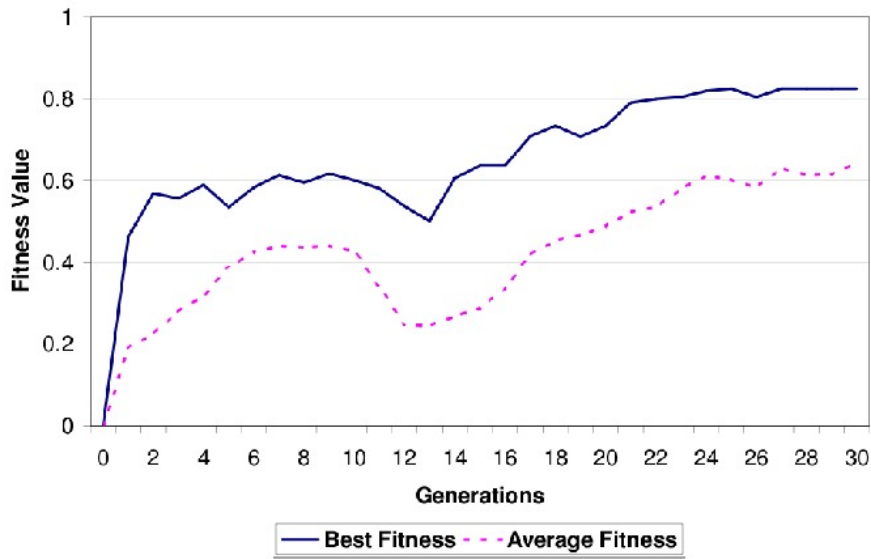


Figure 5.2: Fitness value during the training phase

famous for applications involving maze-like environment. Consequently, on top of comparing the GEASM system with the other three reference systems, the performance of a mobile robot with simple WF behavior will be included in the following two sub-sections. This is done in order to see the practicality of implementing the proposed system over a simple rule-based algorithm. The rule for WF implemented in this thesis is shown in Algorithm 3.

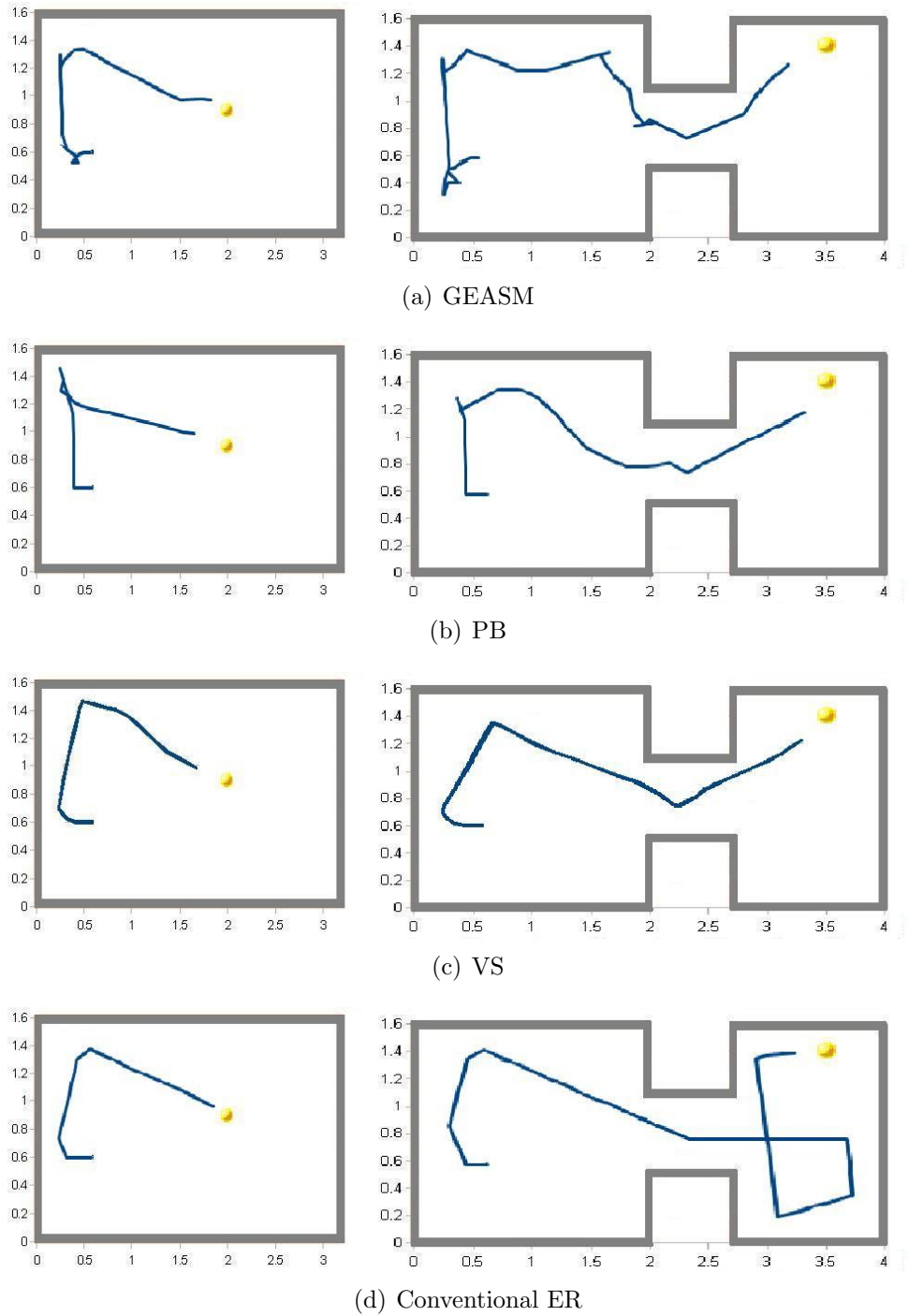


Figure 5.3: Search and exploration in a simple environment

Algorithm 3 Simple Wall Following - Left Hand Rule

```
1: Go straight to find a wall, then turn right
2: Keep following the left side of the wall
3: repeat
4:   if (Front Path is Clear) AND (Robot is at a sufficient distance with the wall)
      then
5:     Go Straight
6:   end if
7: else
8:   if (Front Path is Clear) AND (Robot is quite far from the wall) then
9:     Make a small turning to the left and move forward a little bit
10:  end if
11: else
12:  if (Front Path is Clear) AND (Robot is too near to the wall) then
13:    Make a small turning to the right and move forward a little bit
14:  end if
15: else
16:  if (Front Path is Clear) AND (No wall detected at the left side) then
17:    Turn 90 degree to the right and move forward
18:  end if
19: else
20:  if Robot sensed an Obstacles at the Front then
21:    Turn 90 degree to the right
22:  end if
23: until the termination criteria has met
```

The results shown in Figure 5.3 also indicate that in order to really gauge the performance of GEASM, the tests need to be redesigned to include other parameters or to use a more complex test environment. In the following sub-sections, the performance of all the five systems (i.e. GEASM, PB, VS, ER, and WF) will be tested to investigate their competency based on two specific parameters. The first parameter is the exploration coverage, which basically should provide an indication of how well a system covers a finite-sized mission environment. The second parameter is the efficiency in the execution of the search and exploration mission in various environments.

5.1.1 Performance on Exploration Coverage

Figure 5.4 and Figure 5.5 are the test environments that are used to investigate the exploration coverage of the five systems. The first environment is an extended and enlarged version of the training environment, creating 10 segmented pathways connected in a snake-like pattern. On the other hand, the second environment has been designed to investigate the ability of a robot to explore an environment with 10 isolated rooms. In this experiment, a target object is not placed in the environment, which effectively creates a need for the five systems to wander around, exploring its mission environment.

Each system is given 1000 decision cycles or time steps (TS) to explore as many paths or rooms as possible. Within the given duration, the route taken by a robot is recorded and the identified number of paths or rooms covered should indicate the overall coverage of mission environment. The number of TS taken to fully cover the respective paths or rooms is then determined. As an example, for a case where a robot keeps repeating a route through the same five paths or rooms within the allocated 1000 TS, the robot is said to have covered 50% of the mission environment. The number of TS taken by the robot to visit all five rooms the first time is counted as the time required to cover the visited area.

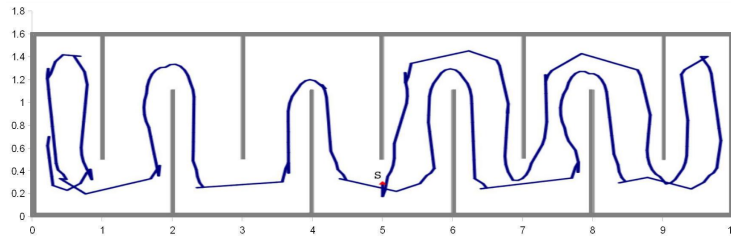
The experiments are executed with three different starting point ‘S’ (left-most, centre and rightmost of the field - refer Appendix A). At each starting position, the robot is placed facing different directions. For each case, the mission is repeated three times in order to assure consistency.

Table 5.1: Performance comparison of exploration coverage in two different environments

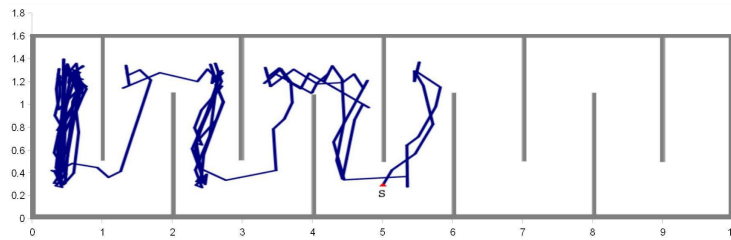
Environment	Exploration	GEASM	PB	VS	ER	WF
10 paths	Average coverage (%)	99.00%	46.67%	48.89%	55.56%	100%
	Average time (TS)	461	483	689	666	595
10 rooms	Average coverage (%)	100.00%	38.89%	53.33%	32.22%	100%
	Average time (TS)	679	489	763	450	674

Table 5.1 shows the average exploration coverage of the experiments in the two worlds. Note that the average time (TS) presented in the table has been rounded up into an integer value. Results in both cases presented in Table 5.1 show that the GEASM system can give better exploration coverage in less exploration time compared to PB, VS, and ER systems. Data from the experiment in the first environment shows that the GEASM system robot successfully covered 99% of the 10 paths within 461 TS. As for the PB, VS, and conventional ER methods, the robot only covered 46.67% of the 10 paths in 483 TS, 48.89% of the 10 paths in 689 TS and 55.56% of the 10 paths in 666 TS respectively. A similar pattern can be seen in the data from the experiment for the second environment. GEASM system made full coverage of the environment within 679 TS, while for PB, VS, and conventional ER approach the robot could only cover 38.89% of the 10 rooms in 489 TS, 53.33% of the 10 rooms in 762 TS and 32.22% of the 10 rooms in 449 TS, respectively.

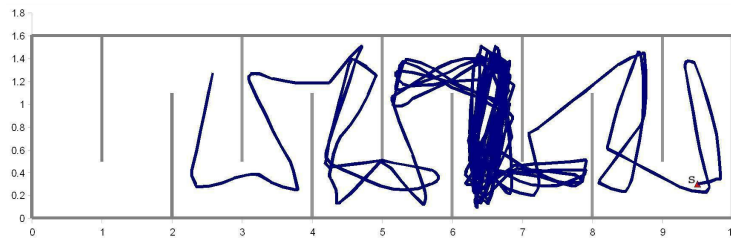
Since the walls for the two environments are connected together, it is guaranteed that a robot is able to visit all cells (i.e. rooms or paths) by keeping one hand in contact with one wall. Results in Table 5.1 has confirmed this theory with the WF system robot actually covered 100% of the two environments with relatively good coverage time (i.e. 595 TS for 10 paths, and 674 TS for 10 rooms).



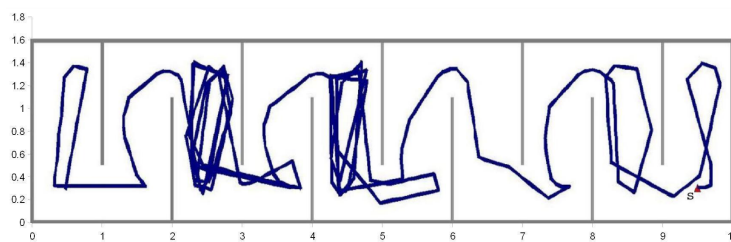
(a) GEASM takes 474 TS to complete the exploration



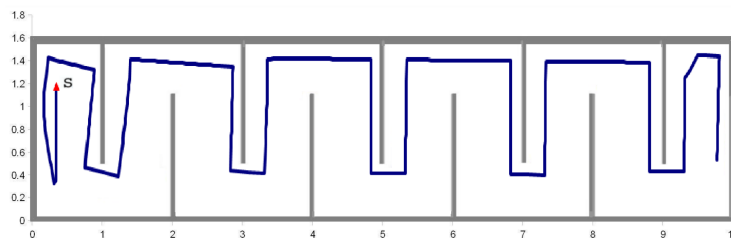
(b) PB covered 60% of the area in 570 TS. It was trapped at the left-most cell



(c) VS takes 988 TS to cover 80% of the 10 paths

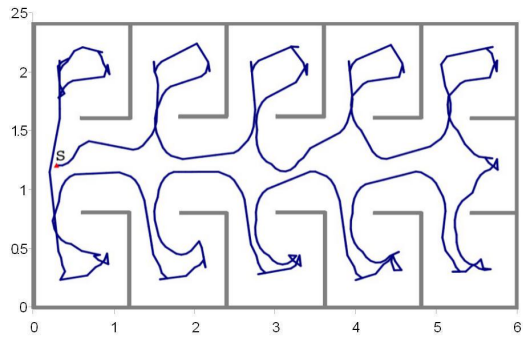


(d) ER takes 783 TS to cover 100% of the 10 paths

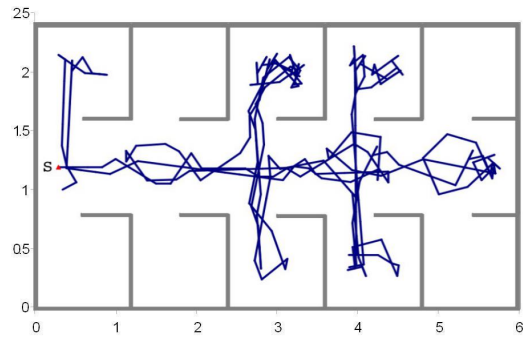


(e) WF takes 400 TS to cover 100% of the 10 paths

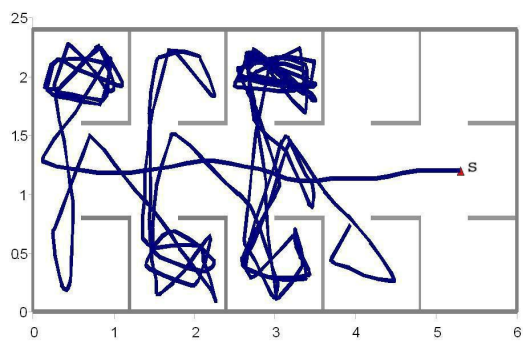
Figure 5.4: Best path traces for exploration in 10-paths environment



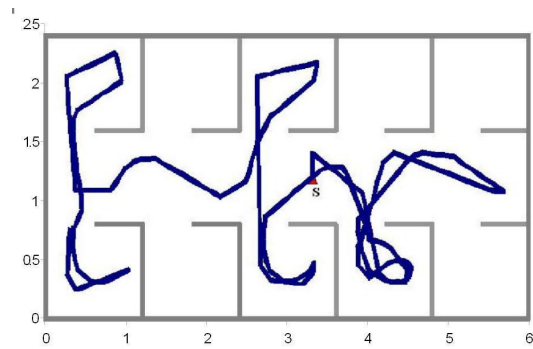
(a) GEASM takes 593 TS to complete the exploration



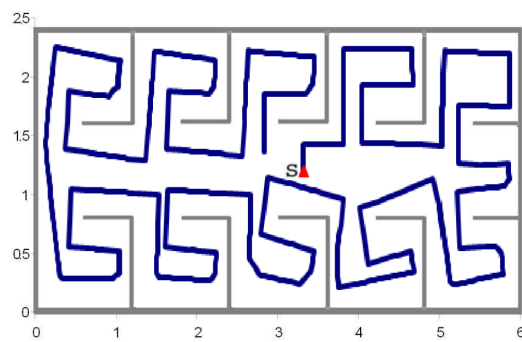
(b) PB covered only 5 rooms in 539 TS



(c) VS covered 7 rooms in 754 TS



(d) ER covered 5 rooms in 390 TS



(e) WF Takes 613 TS to complete the exploration

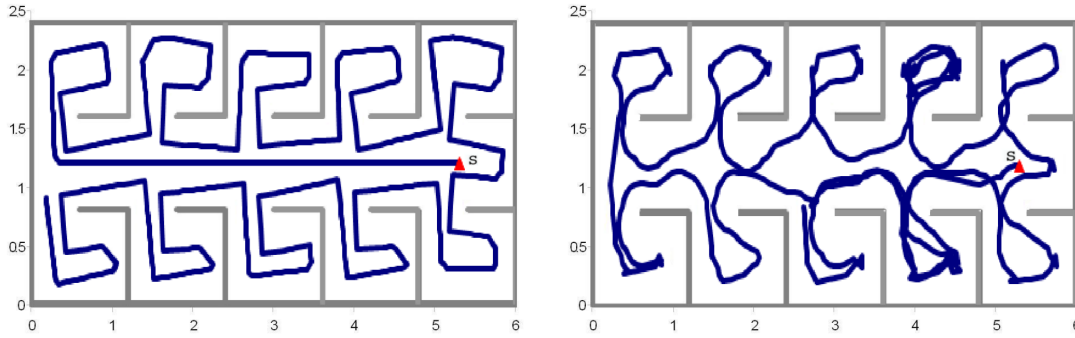
Figure 5.5: Best path traces for exploration in 10-rooms environment

Figure 5.4 and Figure 5.5 show the best path traces extracted from the recorded route taken by each of the five system platforms in the two test environments. The figures clearly show that using PB and VS methods, the robot is more likely to be trapped in local minima (can hardly move out from one cell to another). As for the conventional ER method, the results are not consistent especially when executing in an environment that is different from the training environment. Even though the best path traces in Figure 5.4 have shown that the conventional ER method is able to complete the exploration for all the ten paths, the average coverage of this environment is quite low. As mentioned earlier, the 10 path environment is an extended version of the training environment. Since during the training phase, the robot will be placed at the right-most position, and will wander around the world to find the target object which is placed at the left-most cell, the best solution of the trained system will control the robot to move from right to left. This might be one of the possible reasons why the results show a good coverage (i.e. 100% coverage) in the environment that is similar to the training environment (i.e. 10 paths) with the initial robot position at the right-most position. However, when the initial robot position is in the middle or at the left-most position of the environment, the coverage area is decreased, leading to a lower average.

In the environment which is completely different from the training environment (i.e. 10 rooms), the conventional ER control system has consistently produced a low percentage of rooms coverage. The best path traces for the three positions for ER (in 10 paths environment) are shown in Appendix B.

In order to examine the quality of the exploration made by each method, the worst path executed by all the five techniques (i.e. GEASM, PB, VS, ER, and WF) in 1000 TS are presented in Appendix C. A further analysis is then conducted between WF and GEASM on the same initial position where WF performed the worst. Figure 5.6 and Figure 5.7 show the comparison of the path traces taken by the two systems. From Figure 5.7 it can be seen that WF requires approximately double the number of TS compared to the GEASM system in the 10 paths environment.

This is because WF needs to visit all the nine paths twice, before it is able to complete its exploration with the final cell. On the other hand, it is observed that the GEASM system is able to explore its world whether the wall is at the right side or the left side of the robot. This is the main contribution of a better exploration coverage that has been achieved by the GEASM system in this environment. A similar effect could be obtained by adding more rules to the existing WF algorithm. However, it may take a few trials in determining the right rules and may require some parameter tuning before a successful exploration can be achieved (e.g. how much to turn, how much to go forward, what is the appropriate distance between robot and wall, etc.).



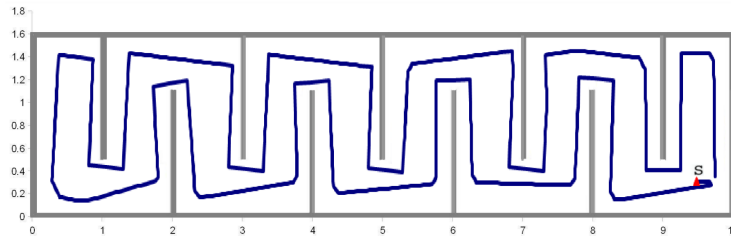
(a) WF takes 715 TS to complete the exploration

(b) GEASM takes 599 TS to complete the exploration

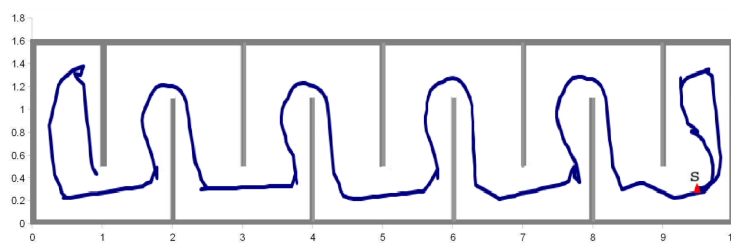
Figure 5.6: Comparing worst path traces of the WF system with GEASM in 10-rooms environment

5.1.2 Performance of Search and Exploration Mission

In this experiment, the systems that are being tested are put through some relatively more complex mission environments in order to identify specific distinction between their performance in a search and exploration mission. An execution time of 500 TS has been set for each mission. For each system, the experiment has been repeated ten times with different initial robot positions and random target object locations (refer Appendix D).



(a) WF takes 700 TS to complete the exploration



(b) GEASM takes 334 TS to complete the exploration

Figure 5.7: Comparing worst path traces of the WF system with GEASM in 10-paths environment

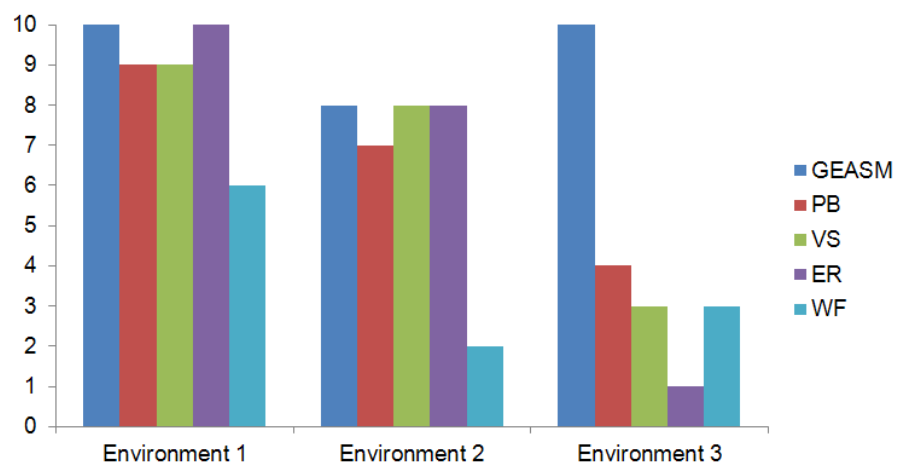


Figure 5.8: Number of successes for search and exploration missions in various environments

Table 5.2: Area ratio of testing environment over training environment

Environment	Height (mm)	Width (mm)	Area Ratio
Training	1600	3200	1.000
10 rooms	2400	6000	2.812
10 paths	1600	10000	3.125
Scattered (Environment 1)	3500	4000	2.734
Loop (Environment 2)	3500	4500	3.076
Maze (Environment 3)	6300	6000	7.383

Figure 5.8 shows the number of successful search and exploration missions performed by all the five systems in three different environments. From the result, it can be summarised that the average percentage of successful missions for GEASM, PB, VS, ER, and WF are 93%, 67%, 67%, 63%, and 37% respectively. From the experiment, it is observed that the efficiency of all five systems can be degraded by the structure of an environment. For example, while GEASM can consistently accomplish its missions in Environment 1 and Environment 3, it has failed to complete the mission in two different robot-target initial positions in Environment 2. As for PB, VS, and ER, they have performed badly in Environment 3. Even though results in the previous sub-section have shown that WF had performed excellently, this time the system has failed in most of the environments that have been set for the experiment.

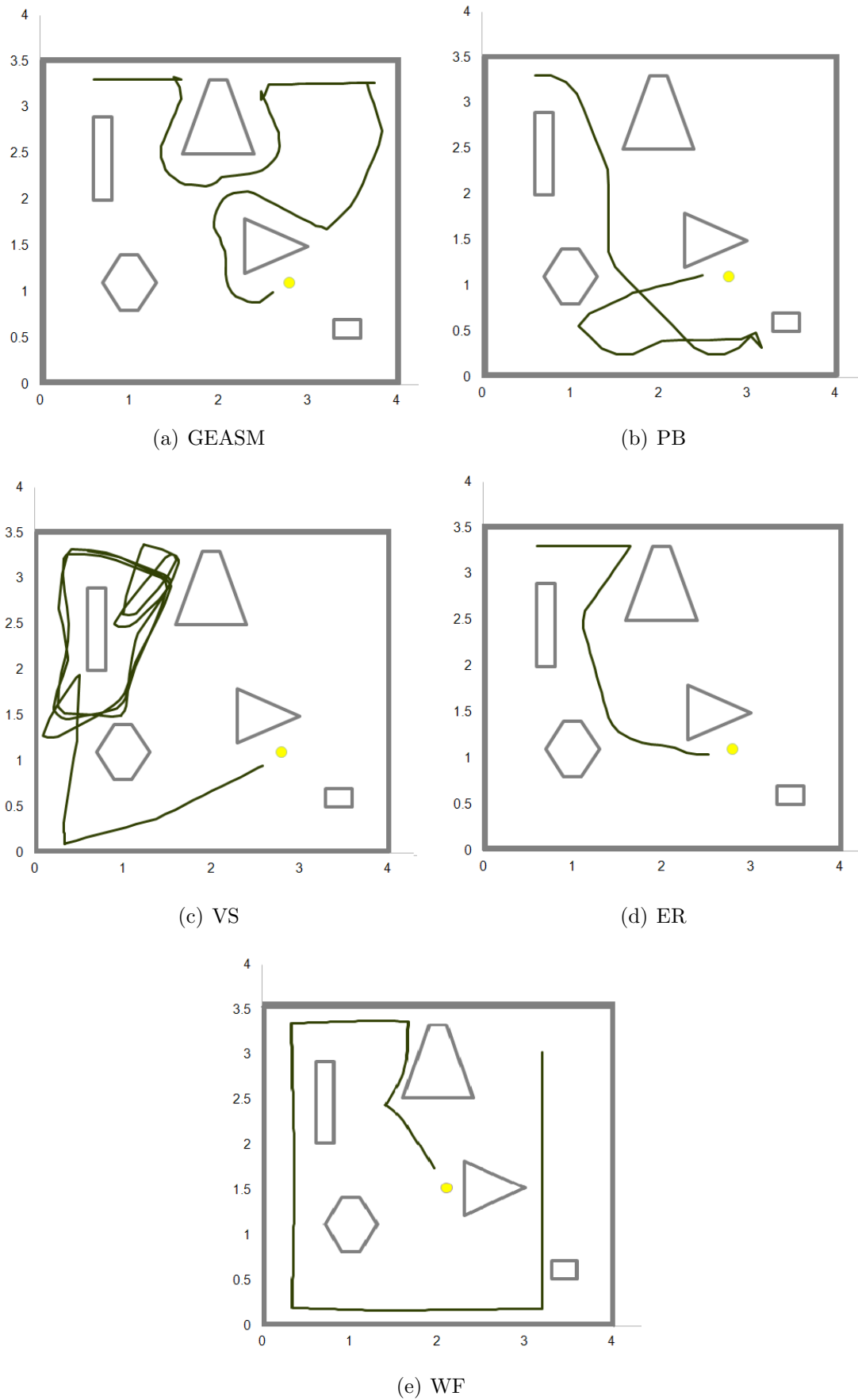
Figure 5.9, Figure 5.10 and Figure 5.11 are examples of the path traces that can be successfully executed by the five systems in the three environments. During the experiments with GEASM, PB, VS, and conventional ER, it became apparent that the GEASM system performed well in the exploration task. The GEASM approach has the ability to operate in a world which is completely different from the training environment. Table 5.2 shows the area ratio between the testing and the training environments that have been implemented so far. From the table, it can be seen that GEASM is capable of exploring an area up to about seven

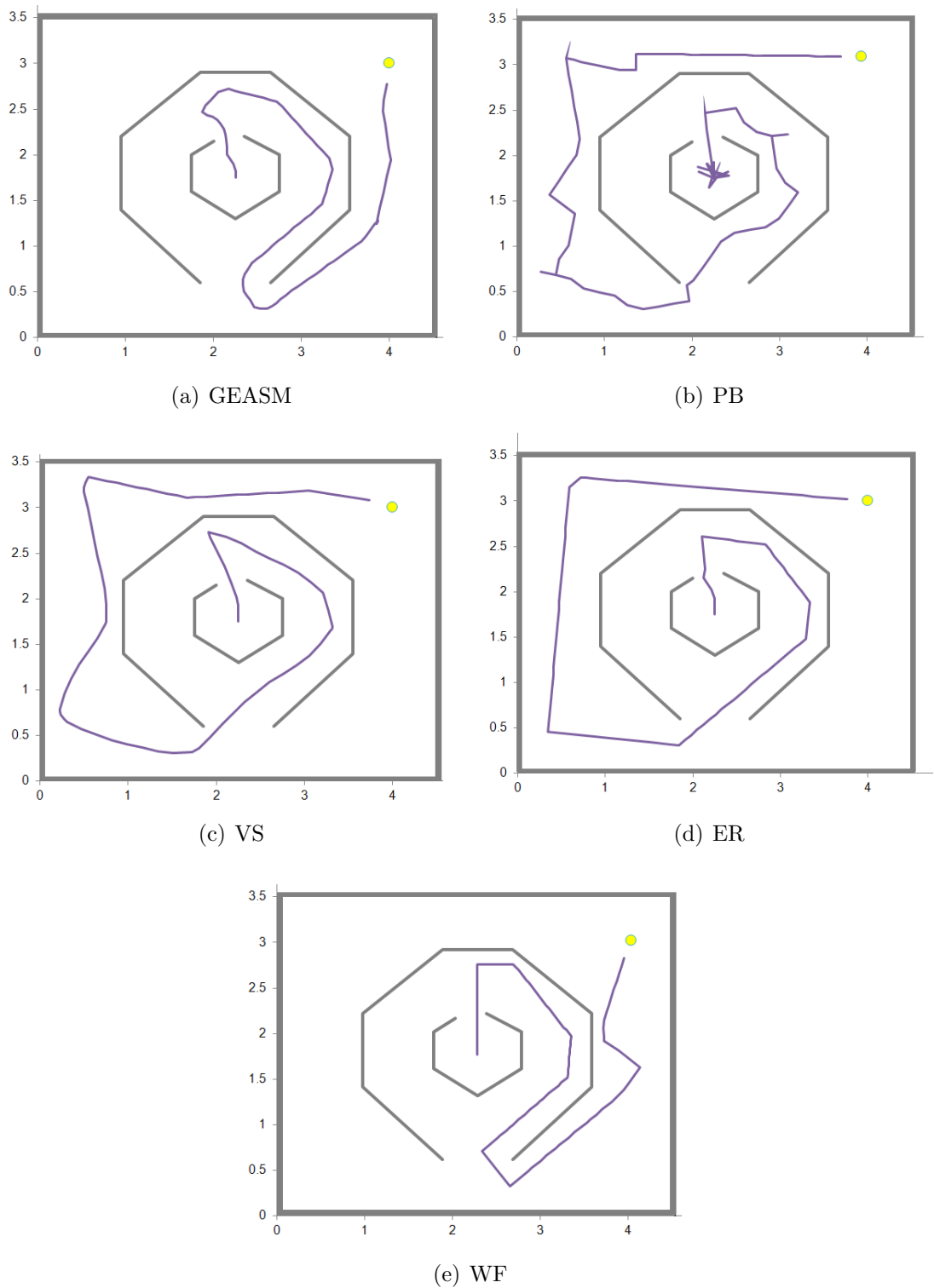
times larger than the area of the training environment. Moreover, the proposed techniques have been tested in various types of environments where the width and orientation of each cell are assorted.

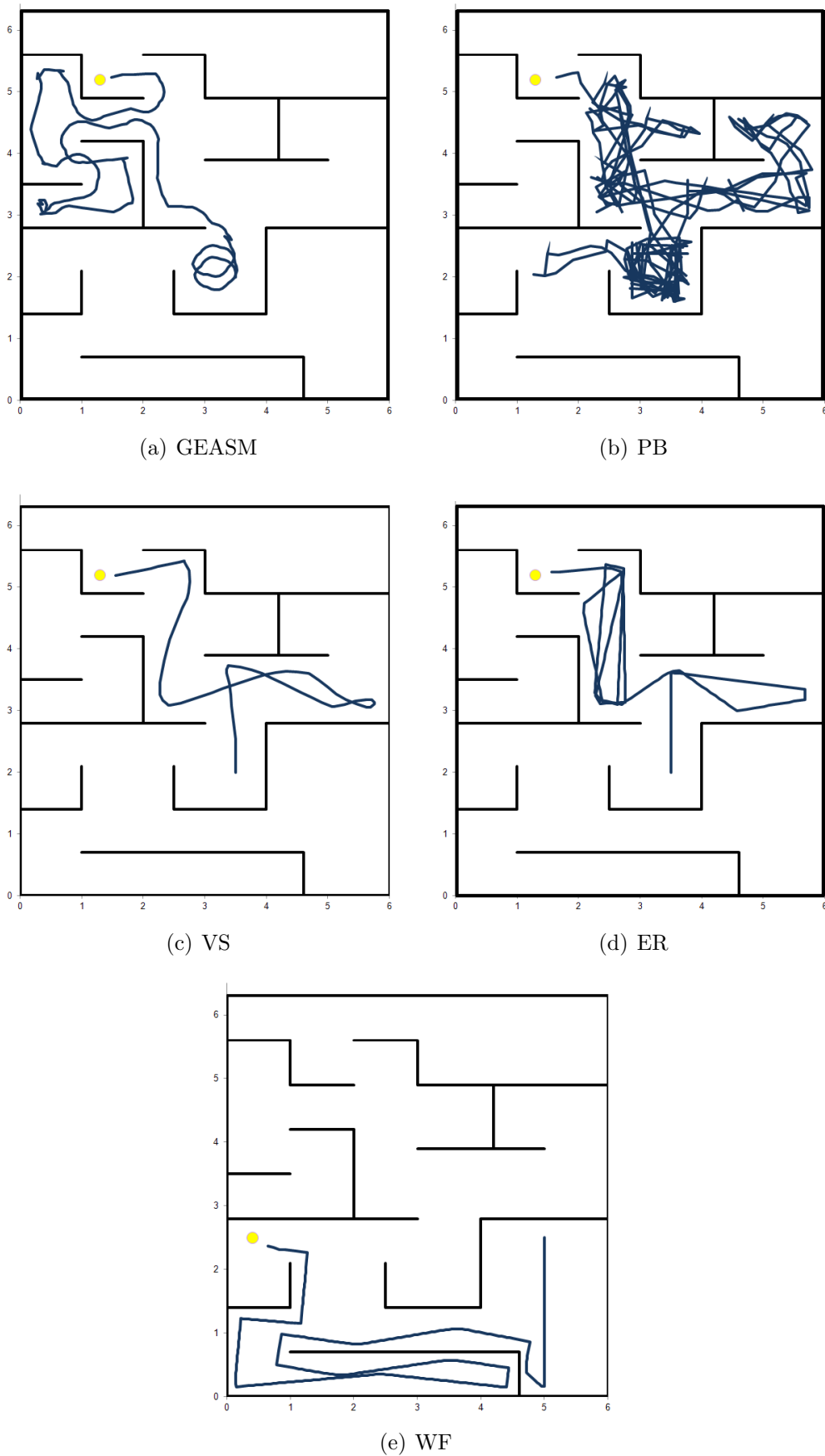
One of the factors contributing to the success of GEASM in this mission is the emergent behavior, which is often found in a behavior-based system. A behavior can *emerge* as a result of local interactions between the system components [148]. As illustrated in Figure 5.9(a), Figure 5.10(a) and Figure 5.11(a), it seems that the robot is performing the **WF** behavior. This could be due to the interactions between the system and its environment data (i.e. data from sensors). It can also occur as a result of the interactions of the three basic behaviors (i.e. **Avoid obstacle**, **Wander** and **Drive to Target**). Further information on emergent behavior can be found in [149, 2].

The result for GEASM in Environment 2 is then to be further investigated. Figure 5.12 shows the path traces of the two scenarios in Environment 2, where the GEASM is unable to complete the search and exploration mission. From the figures, it can be clearly observed that the main factor of the mission's failure is due to the **WF** behavior which emerged from the system's interaction. As stated in [150], WF is commonly implemented when a robot is exploring an unknown area. Therefore, the trained system actually presents a relevant outcome to its current situation. However, in general, WF works best when it is executed in a maze where the walls in the maze are connected together. However, if the wall is not 'simply connected' (e.g. the start and endpoints are in the centre of the structure), the chances for the robot to go around their ring (keep repeating moving in a loop) are higher. This reduces the probability of reaching the overall system objective. In this case, the robot may not be able to find the target object due to the inefficiency to explore its world.

However, it is important to note that the GEASM is not trained for WF. As long as during the exploration, a target object is spotted in the robot's view, the robot will release the **WF** behavior and execute a suitable behavior (e.g.

**Figure 5.9:** Search and exploration: Environment 1

**Figure 5.10:** Search and exploration: Environment 2

**Figure 5.11:** Search and exploration: Environment 3

Drive to Target behavior) to reach the target object safely. This explains why the GEASM is able to perform consistently in Environment 1, even though the wall is not connected together like the maze in Environment 3.

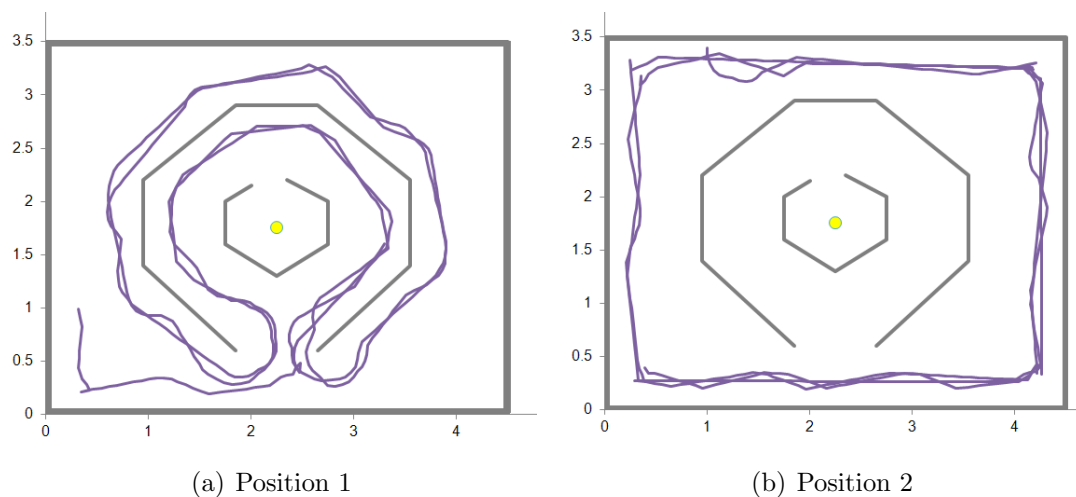


Figure 5.12: Execution of GEASM in Environment 2

Even though it is claimed that the success of the GEASM system in this mission is due to the **WF** behavior (i.e. the emergent behavior), it is very surprising that the WF system has failed in this mission in almost two thirds of the test environments. The movement of a robot with the WF system is then further investigated. From the observation, it is found that the unsuccessful mission for the WF system in Environment 1 and Environment 2 is due to the response made by the WF system when the robot has suddenly not been able to sense any walls on its left side. Based on Algorithm 3, in such a situation, the robot will move forward and make a small turning in order to find the wall again. While the algorithm works best in 10 paths and 10 rooms environments as presented in the previous sub-section, the WF has failed to deal with irregular shape and scattered obstacles in Environment 1 and Environment 2. Figure 5.13 shows an example of a scenario that could lead to the unsuccessful mission.

Theoretically, the WF system should be able to perform well in Environment 3, as the walls are fully connected like a maze. However, in most cases, it

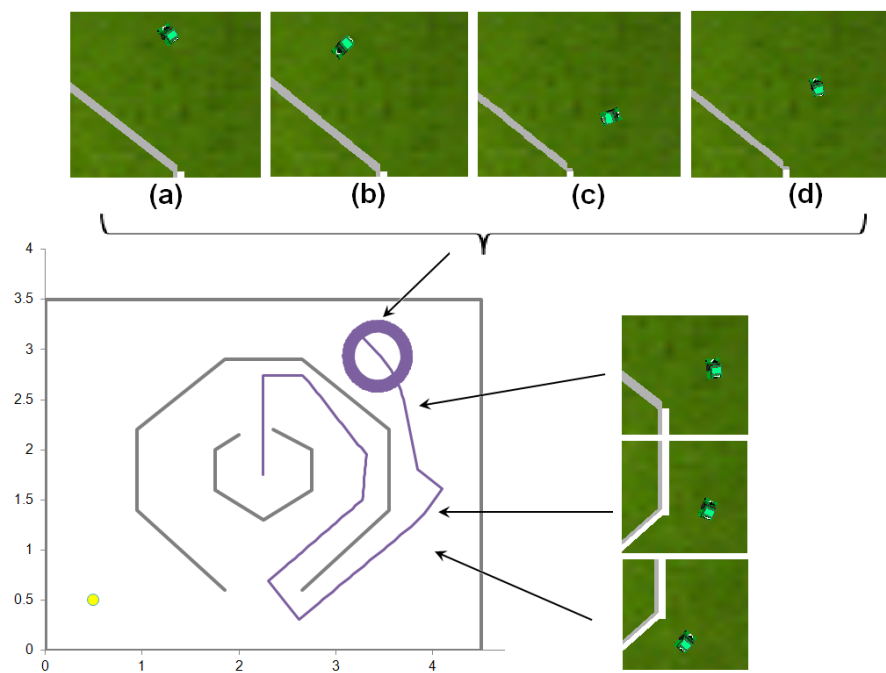


Figure 5.13: A scenario where the WF approach is unable to complete a mission. Sub-figures (a)–(d) show a sequence of movements when the robot is unable to detect a wall on its left side.

is discovered that the WF system requires a longer TS (i.e. more than 500 TS) to complete the mission, while in some specific cases it simply fails indefinitely (refer Appendix E). This is mainly because the WF system, being a rule-based system, stays true to its rule by keeping its path close to the walls, which consequently made its field-of-view ineffective for search purposes and therefore requiring more TS (or it simply fails). This also expectedly indicates that a WF system is only suitable for a maze environment with single-lane-sized corridors. It is also worthy to note here that the ‘simple’ WF rule does not necessarily mean that the implementation will be simple. Some implementations add multiple conditions to cater for various possible dynamics of an unknown environment, while others simply turn to AI for similar implementation [151, 152, 153, 154].

5.2 Target Tracking

This section investigates the efficiency of the proposed system to carry out the second task—the target tracking mission. Figure 5.14 shows the trend of fitness values during the training phase over 30 generations for the target tracking task. From Figure 5.14, it is clear that the best fitness value has converged to a good result of around 0.90.

The trained system has been tested in several scenarios. Figure 5.15 shows the path traces of a single robot, tracking a target object in a simple environment using four different systems (i.e. GEASM, PB, VS, and conventional ER). In this simple environment, the target tracking mission is being carried out without the existence of any obstacles. From the results it is observed that the four systems are able to execute the tracking mission, approximately with equal performance.

The four systems are further tested in a more difficult scenario. In this case, the target tracking mission is executed either in the environment consisting several obstacles, or the target robot made a sudden changes in its movement during the execution of the mission. For both scenarios the chances to miss the target

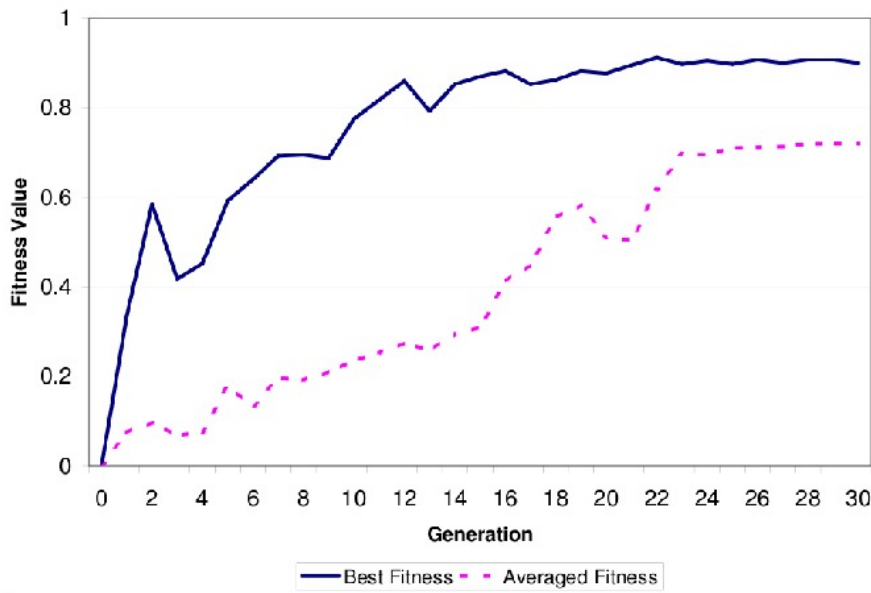


Figure 5.14: Fitness value during the training phase

are higher. From the results shown in Figure 5.16, Figure 5.17, Figure 5.18 and Figure 5.19, it can be observed that there are inconsistencies in the execution of the target tracking task for PB, VS and conventional ER systems. The performance for each of the systems in the environments will be further discussed in the following subsections.

5.2.1 Performance of Classical ASM Approaches

Figure 5.16, Figure 5.17, Figure 5.18 and Figure 5.19 shows that PB and VS systems performed rather poorly in more complex environments. The path traces have reflected that PB and VS methods failed to accomplish any tracking missions within those environments.

One of the major problems with the PB method is the loss of information and knowledge from the suppressed behaviors. The effect of this problem can be highlighted particularly in cases where the robot needs to consider suggestions from behaviors with different goals in order to achieve the overall system objective. As

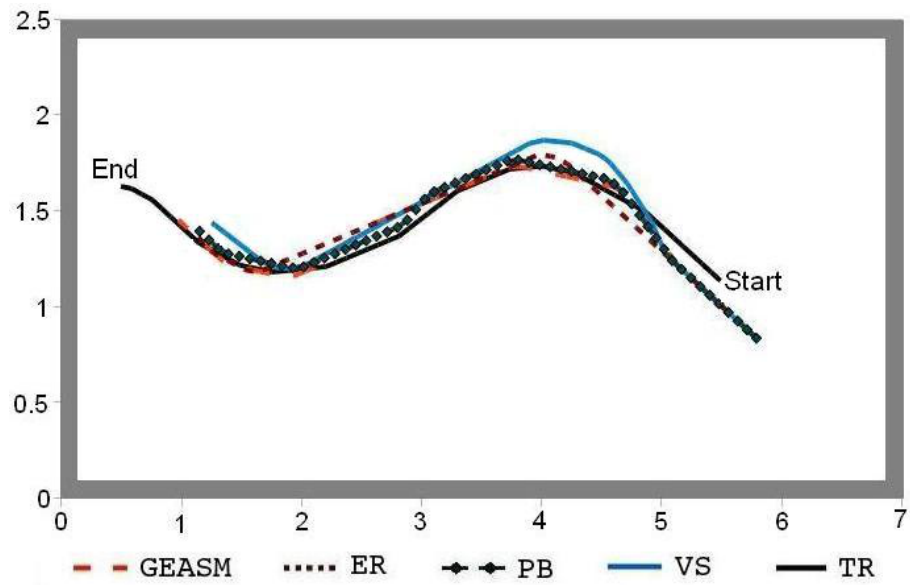


Figure 5.15: Path traces for GEASM, PB, VS, conventional ER: simple environment

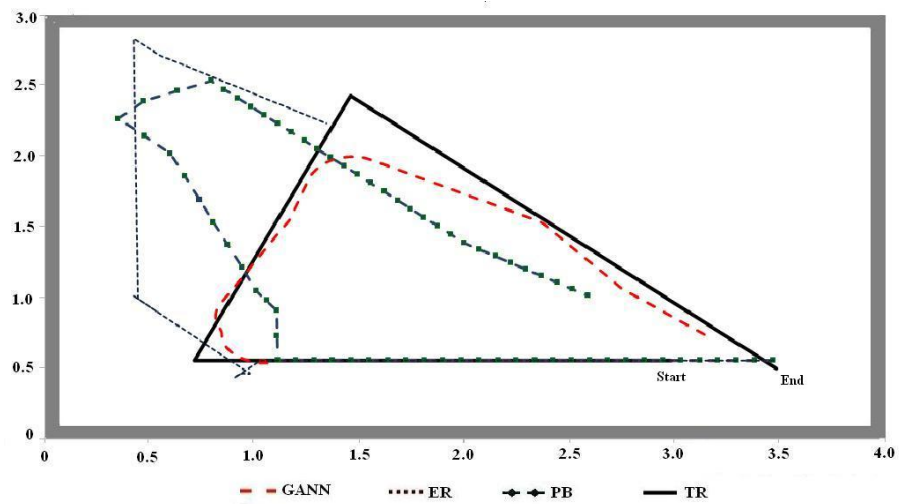


Figure 5.16: Path traces for GEASM, PB and conventional ER: Scenario 1

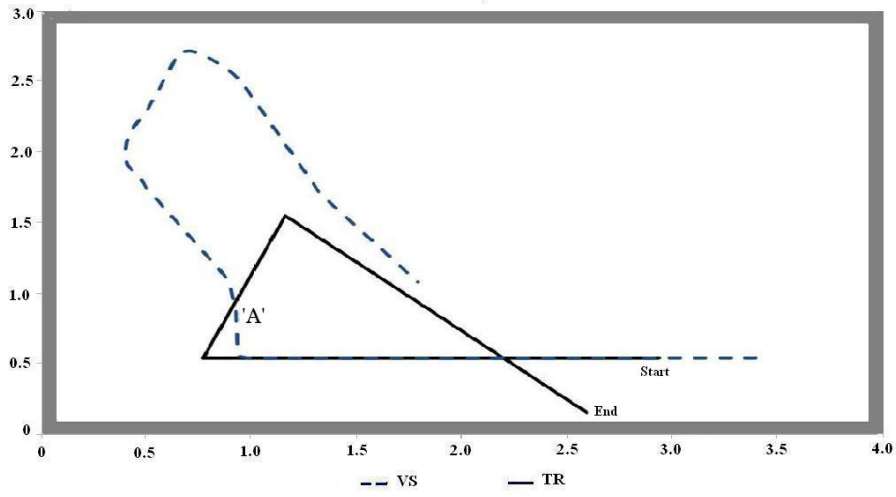


Figure 5.17: Path traces for VS: Scenario 1

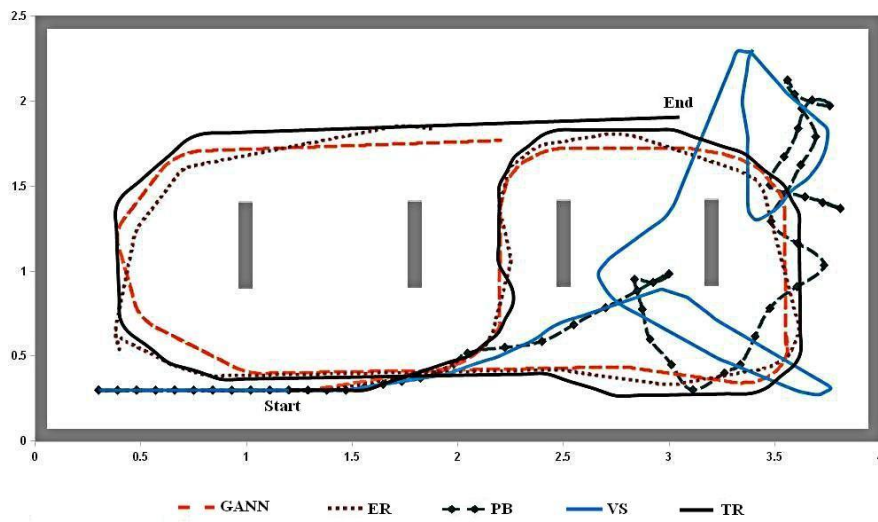


Figure 5.18: Path traces for GEASM, PB, VS and conventional ER: Scenario 2

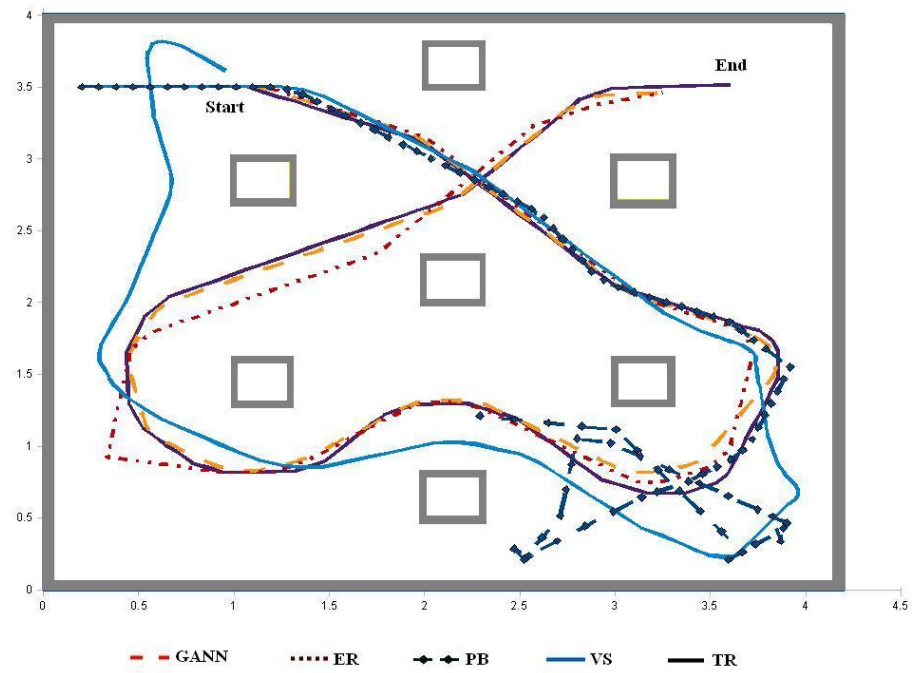


Figure 5.19: Path traces for GEASM, PB, VS and conventional ER: Scenario 3

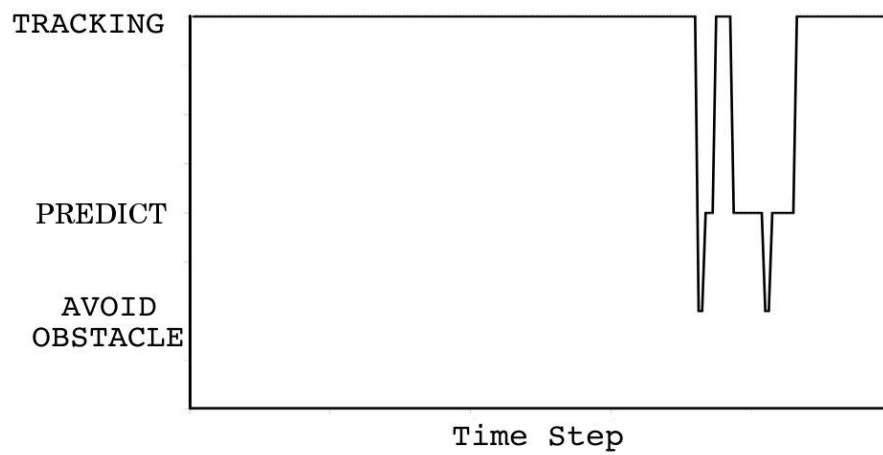


Figure 5.20: Active behavior during the execution of the PB system: Scenario 1

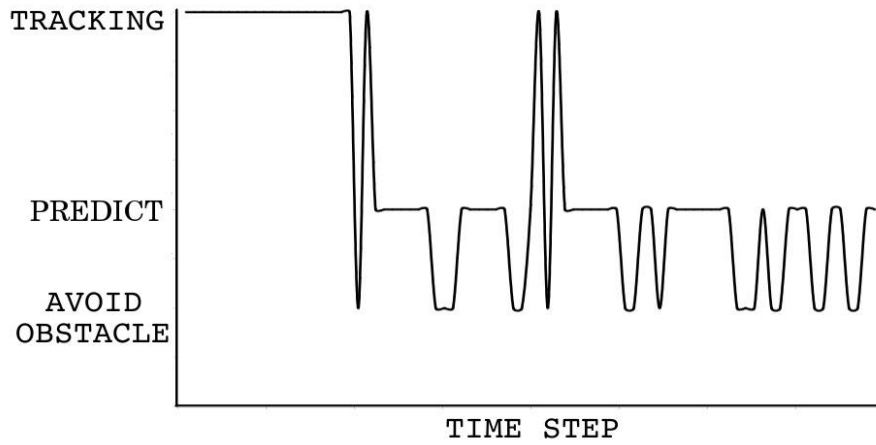


Figure 5.21: Active behavior during the execution of the PB system: Scenario 2

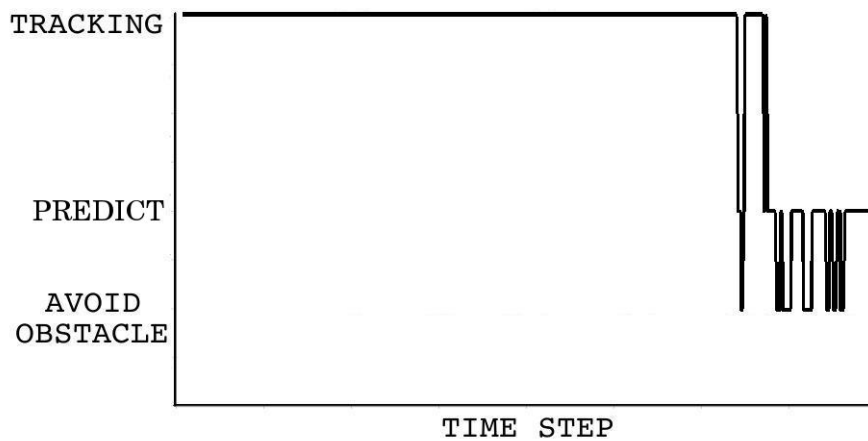


Figure 5.22: Active behavior during the execution of the PB system: Scenario 3

reflected in Figure 5.20, Figure 5.21 and Figure 5.22, once **Avoid Obstacle** behavior is activated (i.e. as it has the highest priority), the behavior will dominate the control system. The obstacle avoidance is being carried out without considering the other expected actions, especially the suggestion from **Track** behavior. Therefore, it can decrease the performance of target tracking and the chances for losing the target becomes higher. It is also observed, for a situation where the complexity of the environment has increased (e.g. due to the existence of more obstacles), the probability of failure to reach the main goal will be increased too. Note that the obstacles will be sensed when the robot is too close to walls or it is too near to the target robot. As revealed in Figure 5.16, Figure 5.18 and Figure 5.19, the PB technique has failed to complete the tracking task in the three environments. A similar result has been reported in [8]. In their experiments, the behavior arbitration technique is unable to accomplish any navigational task due to the activation of **Obstacle Avoidance** behavior. Once the behavior has taken control of the system, the suggested action from the other two behaviors (i.e. **Go to Target**, and **Route Follow**) have been neglected, causing the system to fail in getting to a target position.

Even though the previous argument concludes that the arbitration technique is not the solution for target tracking when it is executed in a complex environment, command fusion technique may not be the answer for this mission either. Principally, a command fusion technique such as the VS method, has the ability of coping with conflicting actions from the behavioral outputs. However, since both the robot and the target can dynamically move around in the environment, the chances for losing the target are very high. Therefore, it is very crucial to design a robot system that can maintain the overall system objective (i.e. keep locating the target object) during the execution of this mission. With the dynamic changes in the environment and the unpredictable next action of the target object, finding a fair representation of information for all contributing behaviors can be very difficult. Results in Figure 5.17, Figure 5.18 and Figure 5.19 have shown that the VS method has failed to complete the mission in the environment.

When the results for the VS method are further analysed, it can be observed that the robot has failed to continue locating the target object when the system is confronted with conflicting behaviors (particularly, the suggestions from **Track** behavior and **Avoid Obstacle** behavior). For an example, in Figure 5.17, when the robot is at position ‘A’, the camera image has detected the target object at the left side of robot’s camera view. At the same time, the left infra red sensor of the robot has detected an obstacle (because the robot is too near to the target object). Clearly, in such a scenario, the **Track** behavior will suggest the robot to move towards the target (i.e. turn left), while the **Avoid Obstacle** behavior will suggest the robot to move away from the obstacle (i.e. turn right). Using the VS method, these two distinct actions have led the robot to overtake the target object. This explains why the path for the target object has changed as the target object needs to stop from moving forward while the robot is overtaking it. Obviously, this is not the desired action for this mission. Even though, theoretically, this effect can be reduced by multiplying vectors of each behavior with an appropriate gain (refer to Equation 2.2), the inconsistency in the tracking performance of the VS system in the three environments have suggest that the gain value for each of the behaviors can vary during the execution of the mission. Clearly, in order to successfully implement a VS method for this mission, another mechanism is required in order to determine in which scenario the gain value should be changed and what gain value should be assigned to each behavior.

5.2.2 Performance of Evolutionary Neural Network Approaches

The proposed GEASM approach is expected to provide adequate means in dealing with multiple goals to reach the main objective. In order to analyse the relationship between the system output and the suggestion from each behavior, a Mean Squared Error (MSE) calculation has been applied. MSE can be used to quantify the amount by which the behavioral input will differ from the system output. For this application, an MSE of zero for a behavior will indicate full contribution from

that behavior action to the system output. The behavior with the smallest MSE will play the largest role in controlling the system. Figure 5.23, Figure 5.24 and Figure 5.25 show that on average the major function of the system output has been conducted mainly by the **Track** behavior. This indicates that in a typical condition, the GEASM has functioned as an arbitration ASM (e.g. PB method), where action has been selected from one behavior (i.e. **Track**) to control the robot movement. However, at certain intervals it can be seen that the system is compromising the behaviors' input especially from **Avoid obstacle** and **Track** to output an appropriate action for that current situation. As revealed in Figure 5.24 and Figure 5.25, the graph will fluctuate more frequently when the robot is executing the task in an environment that is surrounded by more obstacles. Clearly in this case, behavior integration is the key success for the completion of the task.

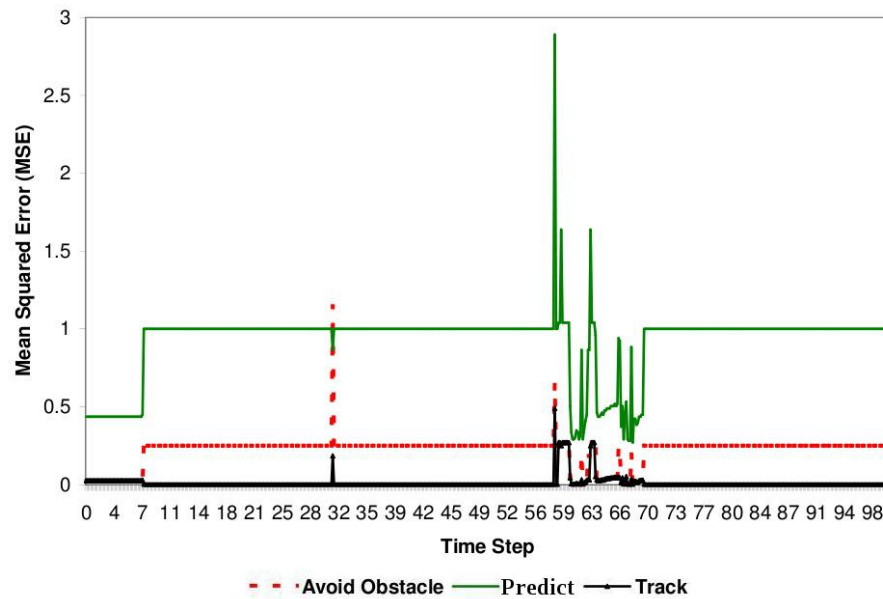


Figure 5.23: MSE between the output of behavioral module and the GEASM system: Scenario 1

Using conventional ER methods shows promising results when the object being tracked does not make any sudden changes in its movement pattern. This can be seen in the results shown in Figure 5.18 and Figure 5.19 where the object being tracked makes no sudden changes due to the obstacles in the environment. Observing the result for other environments, it is obvious that the method is unable

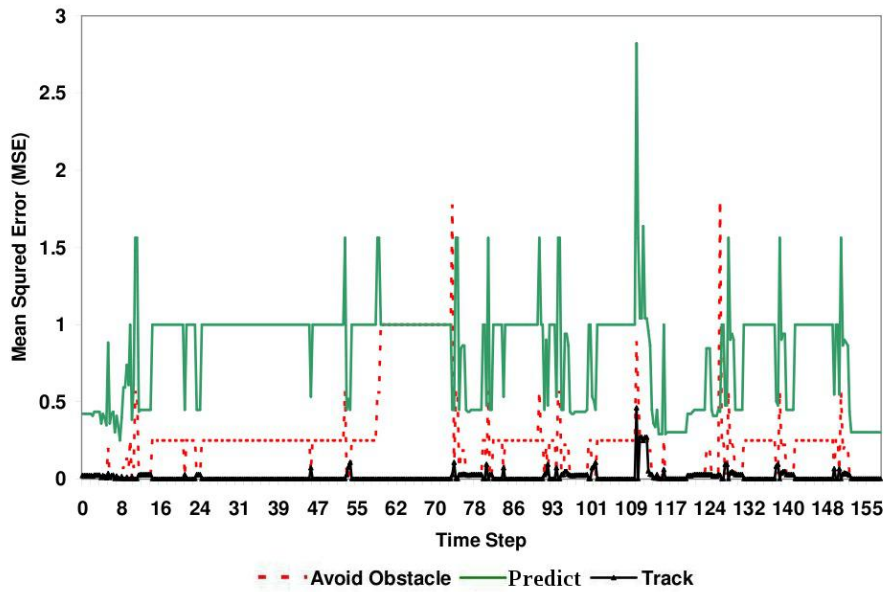


Figure 5.24: MSE between the output of behavioral module and the GEASM system: Scenario 2

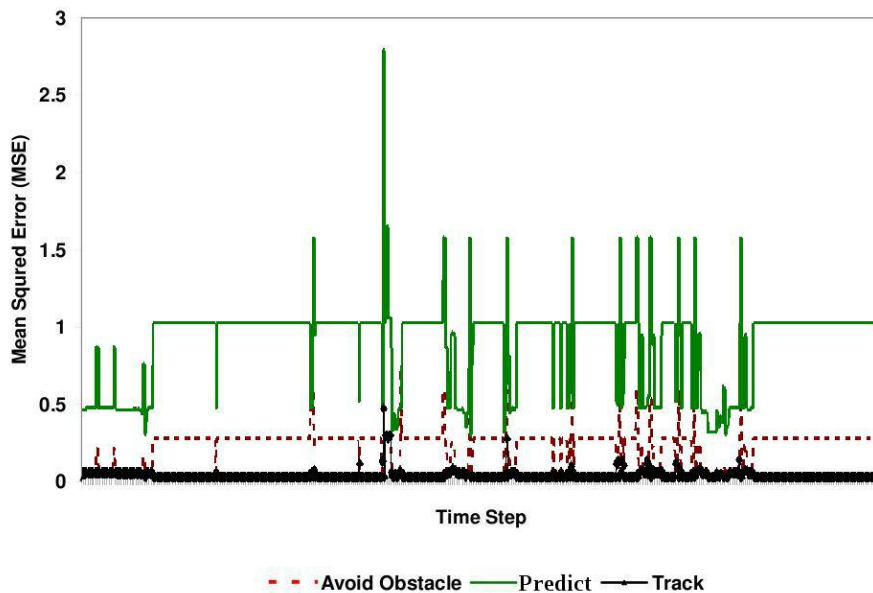


Figure 5.25: MSE between the output of behavioral module and the GEASM system: Scenario 3

to produce the same results once the object being tracked makes a sudden movement (i.e. the first corner of a triangle-shaped movement pattern). However, it does show some effort to retrace the object after a while. One of the reasons behind this scenario is that the evolved system has not dealt with such a situation during its training phase. This explains why the current research trend in ER inclines towards on-line training—this should help the system to continuously adapt to an unseen scenario while executing a given task.

5.3 Performance of the GEASM System under Image Noise

The results presented so far have been executed in noise-free simulated environments. However, in order to test the reliability of any system when executing in a real world, it is important to consider environment noise that may be integrated within the system's input signals.

Even though the nature of a noise is usually unknown and hard to calculate, the existence of a statistical model can be used to reflect the existence of the noise in a sensor model [132, 155]. There are several noise models that have been implemented for the EyeSIM simulator. Figure 5.26 shows the interface of error model that can be introduced to the EyeBOT sensor model.

Considering most of the input data is processed from the camera image, the proposed system has been developed to tolerate a certain degree of image noise without requiring any further training process. Figure 5.27 shows the maximum percentage (i.e. increment by 10%) of image noise that a GEASM system can tolerate in order to run its system successfully. In the experiment, the GEASM system is executed in three different environments for the search and exploration task (labelled as 1,2,3), and target tracking mission (labelled as 4,5,6). As reflected from the bar chart, the proposed system can consistently achieve the system's goal even if

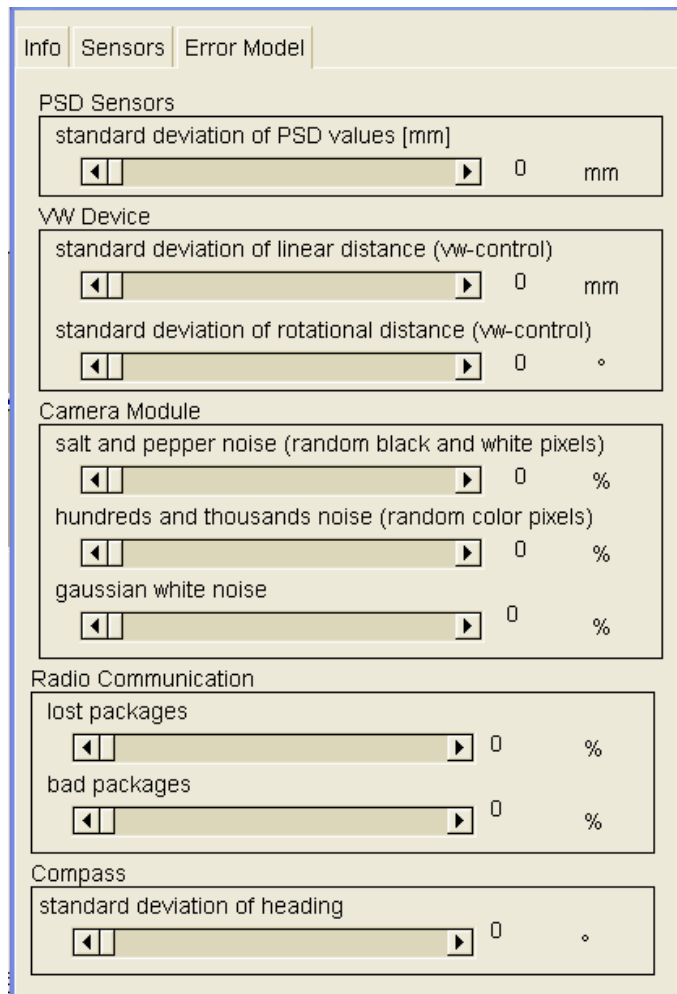
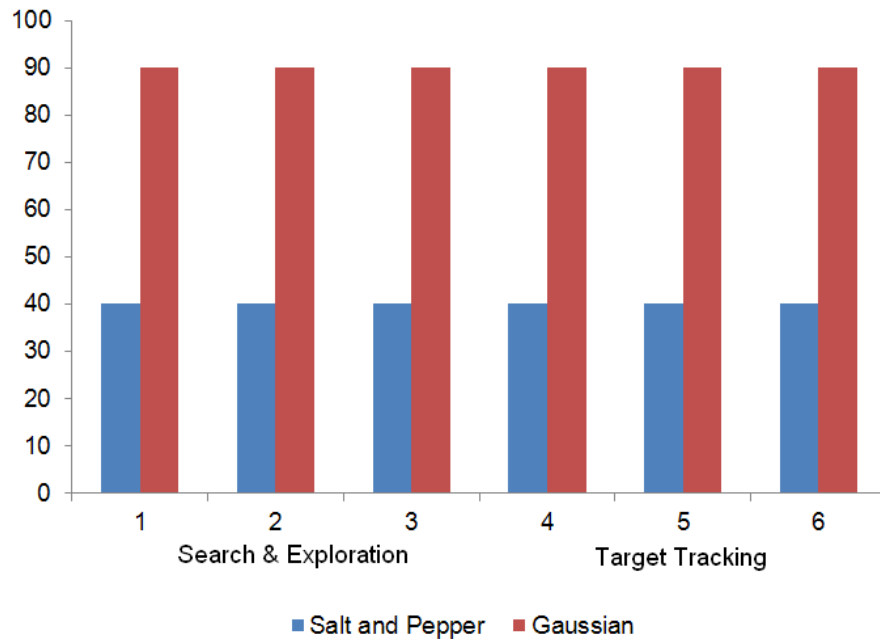


Figure 5.26: The interface of error model in the EyeSIM simulator

Table 5.3: Time steps to complete a search and exploration task under salt-and-pepper noise

Salt&Pepper Noise (%)	Environment 1	Environment 2	Environment 3
0%	100	125	309
25%	111	126	823
40%	305	211	1347

it is executed under the existence of 40% salt-and-pepper noise, and the existence of 90% Gaussian noise. Adding the noise to the system at greater than this level may lead the system to produce undesirable behavior such as continuous reversing from the current position, or moving in a loop. This is only logical since the process of abstracting environment data from such distorted input would be unreliable—which denies any sensible control decisions. Figure 5.28 shows the snapshot of a camera view in three different scenarios: noise free, salt-and-pepper noise, and Gaussian noise.

**Figure 5.27:** The maximum percentage of image noise that GEASM can tolerate

Result from the bar chart (Figure 5.27) suggests that the system is more sensitive to salt-and-pepper noise. This is because, the white pixel has been used

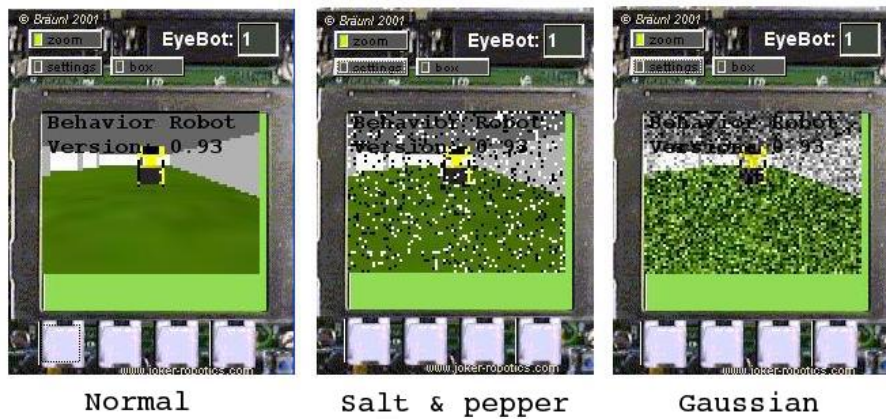
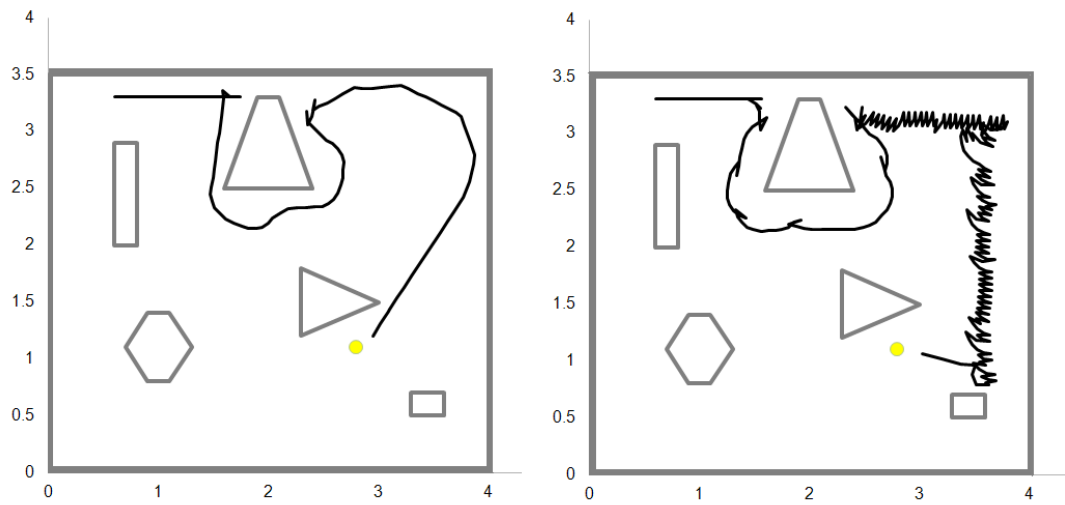


Figure 5.28: Robot's view without noise, with salt-and-pepper noise, and with Gaussian noise

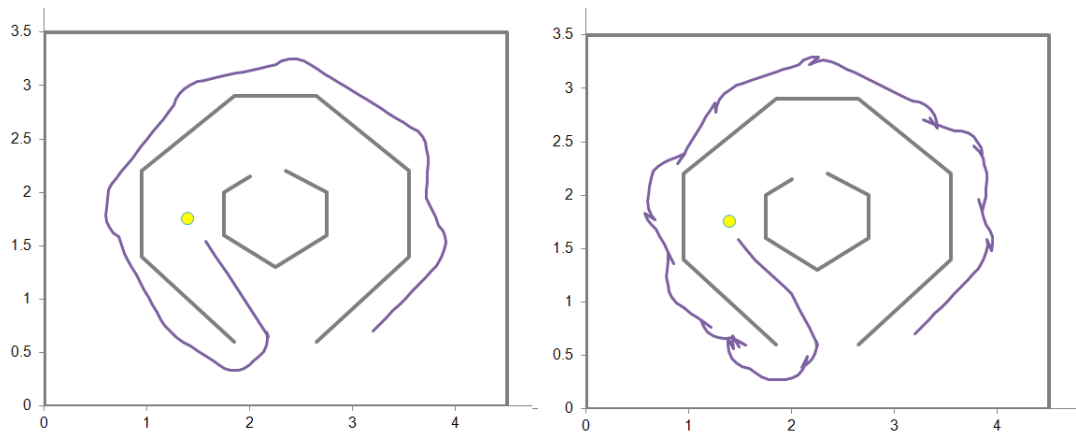
to represent the wall (i.e. obstacle), which is very crucial in the abstraction of environment obstacles. In the previous section, the results presented for search and exploration mission have demonstrated that the emergent behavior (i.e. **WF** behavior) have been in control, most of the time. Since the walls have been constructed using white pixels, it is expected to see some degradation in the exploration performance when the system is executed under this noise. As reflected from the result in Table 5.3, a larger number of TS is needed to complete the mission for a higher percentage of salt-and-pepper noise that is added to the system input. In addition, it is also observed that the smoothness of the path traces have been decreased, due to the existence of these additional black and white pixels (refer to Figure 5.29).

Unlike the search and exploration mission, the major function of the system output for the target tracking mission has been dominated mostly by the **Track** behavior. Since the target is tracked based on yellow pixels, it is observed that the tracking mission can still be achieved with a good performance under the presence of 40% salt-and-pepper noise (refer to Figure 5.30). However, as stated earlier, a higher percentage of the noise will lead to undesirable behavior produced by the system's output. Beyond this level, the noise will cause the system to keep avoiding the obstacles rather than following the target object.

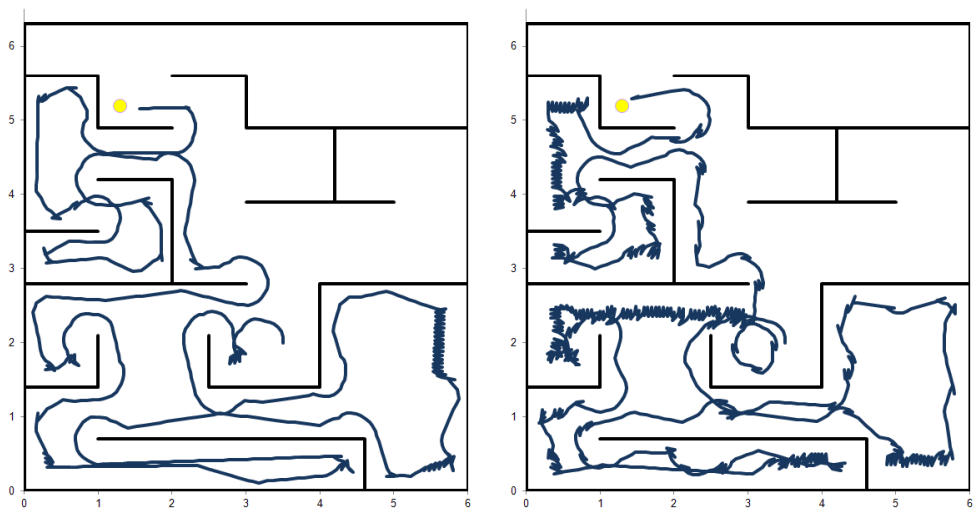
5.3. PERFORMANCE OF THE GEASM SYSTEM UNDER IMAGE NOISE 121



(a) Environment 1



(b) Environment 2



(c) Environment 3

Figure 5.29: Search and exploration: execution under salt-and-pepper noise. Path traces on the left side are at 25% salt-and-pepper noise. Path traces on the right side are at 40% salt-and-pepper noise.

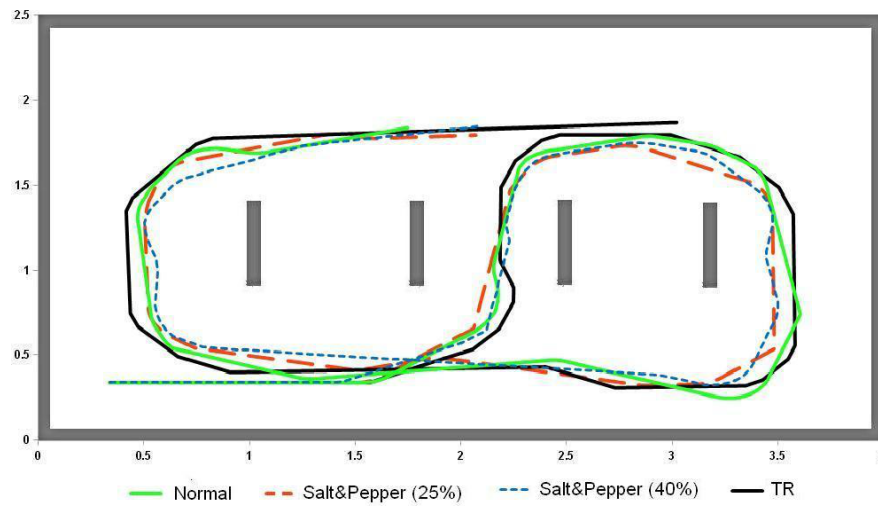


Figure 5.30: Target tracking of the GEASM system under 25%, and 40% of salt-and-pepper noise

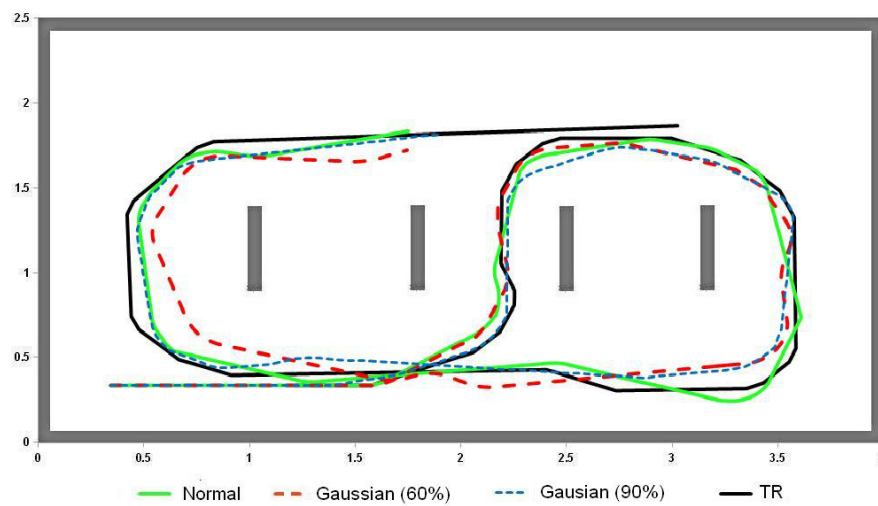
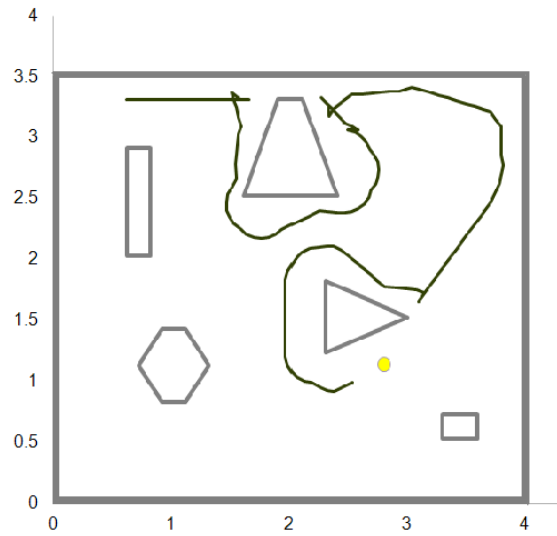
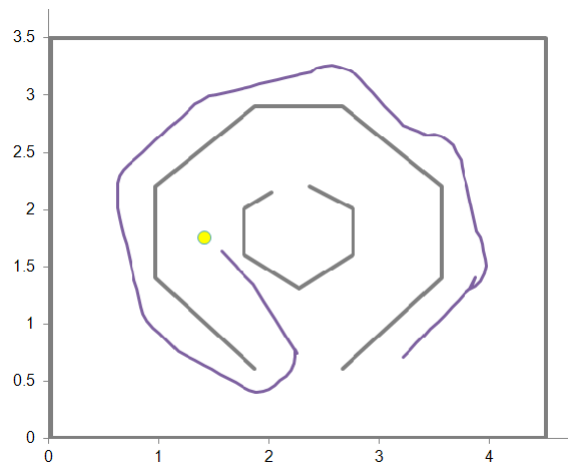


Figure 5.31: Target tracking of the GEASM system under 60%, and 90% of Gaussian noise

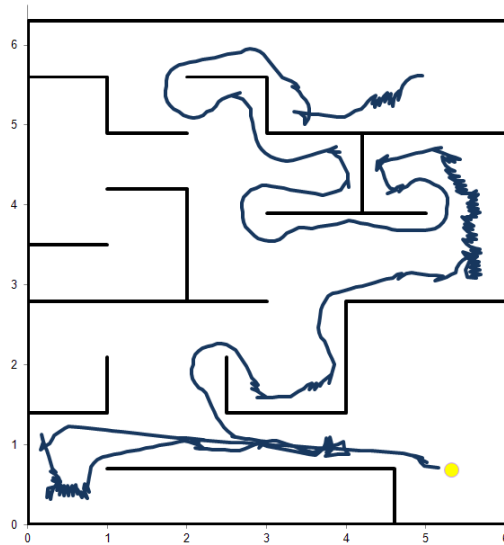
5.3. PERFORMANCE OF THE GEASM SYSTEM UNDER IMAGE NOISE¹²³



(a) Environment 1



(b) Environment 2



(c) Environment 3

Figure 5.32: Search and exploration: execution under 90% of Gaussian noise

Similar to the results presented in the bar chart, the path traces in Figure 5.31 and Figure 5.32 have demonstrated that the GEASM is less sensitive to Gaussian noise. Missions in both applications can be executed with acceptable performances even when 90% of the noise has been added.

5.4 Re-utilising the GEASM System

One of the main drawbacks in implementing an evolutionary neural network approach is the time required to evolve the ANN until a desired system is obtained. However, as presented in the previous sections, GEASM is able to tolerate dynamic structures within mission environments (that are different from its training environment), and is able to cope with up to a certain degree of image noises, without requiring further training phase. In this section, more scenarios on how a fully evolved GEASM system can be utilised for other robot missions will be discussed.

It has been mentioned in the previous chapter that the data going into the behavior modules and the ANN in GEASM are actually abstract information. One advantage for using this approach in extracting information from the sensors is that the trained system can be reused and applied for various criteria of target object or obstacles. In simple words, the ANN is trained to search for a target object—the fact that the yellow colour has been selected to represent a target object is transparent to the system. This way, if another colour is used to represent the object, or any other information abstraction is used for that matter, the system does not need to be re-trained. As shown in Figure 5.33, the path traces have reflected that changing the criteria of a target object has not degraded the overall performance of the system. In fact, a similar result may be observed if other image processing methods (e.g. feature detection) are applied.

It is important to note that the experiments conducted previously have required the robot system to make a suitable decision for a low level action. However, when a clear sequence can be outlined between a set of high level behavior and

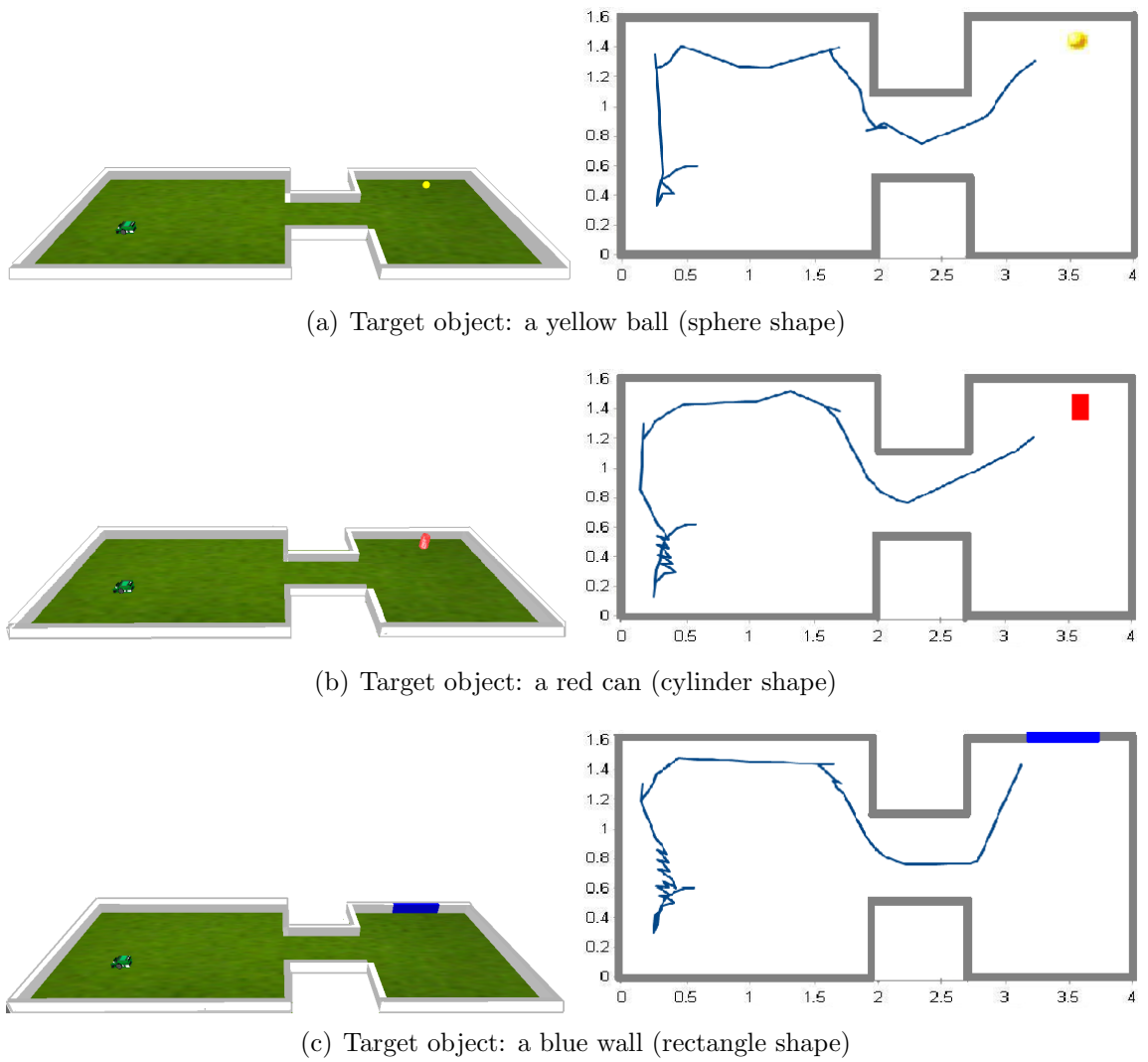


Figure 5.33: Search and exploration performance when changing the target object's criteria

its triggered signal, a Finite State Machine (FSM) approach can be implemented for a higher level robot control system. This section will explore the use of an FSM, utilizing a fully evolved GEASM system in implementing a higher level robot control application—*foraging*.

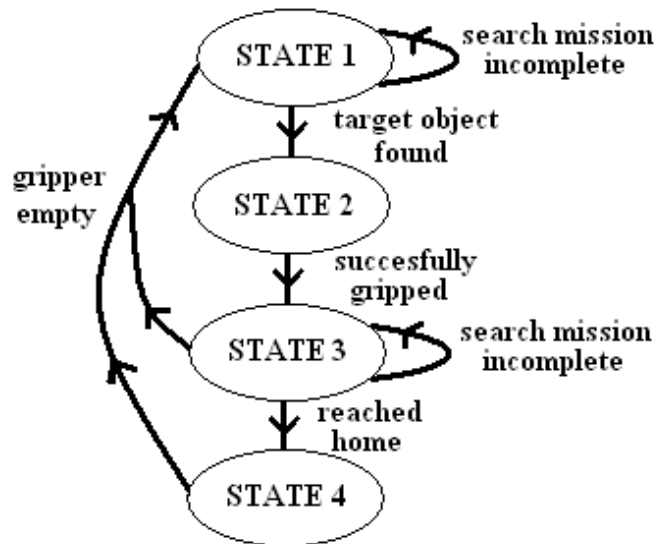


Figure 5.34: Implementing a foraging system using FSM

Figure 5.34 shows the FSM diagram of the overall system. As shown in the figure, the main system is made up of multiple sub-systems— each of which is capable of executing a part of the overall mission. Using FSM approach, at any given time, the system will be in a discrete state that is programmed to activate one of the sub-systems. The description of each sub-systems linked to each state can be further described as follows:

1. **STATE 1:** This is the state in which the main system is trying find a yellow target object. For this, a fully evolved GEASM system— trained for search and exploration in Section 5.1—will be activated.
2. **STATE 2:** is a sub-system developed using a rule-based method which will be activated to control the gripper to acquire the target object.

3. **STATE 3:** is a fully evolved GEASM system—trained for search and exploration in Section 5.1 which will be activated to find home (i.e. coloured with blue). It is important to note that the similar GEASM system has been implemented for this state without requiring a further training phase even though the shape and the colour of the target object in **STATE 3** (i.e. home) are different from **STATE 1** (i.e. yellow target object).
4. **STATE 4:** is a sub-system developed using a rule-based method which will be activated to release the object from the gripper, and move away from the home area.

From Figure 5.34 it can be seen that the foraging system starts with **STATE 1**, allowing the robot system to explore the environment searching for a yellow target object. As shown in the figure, the system will remain active at this state until the target object is directly in front of the robot. When this happens, the main system changes state to **STATE 2**, and the robot starts to acquire the target object by controlling its gripper. Once the robot has successfully gripped the target object, a transition from **STATE 2** to **STATE 3** will occur. In **STATE 3**, the robot starts another search and exploration mission, but this time, it looks for a pre-defined home location. Once the robot has successfully brought the target object to its home location, **STATE 4** will be invoked to release the target object and move away from the home area. Once the gripper is empty, the system will be put back into **STATE 1**. This is also true in cases where the robot accidentally dropped the target object in the middle of its way to home—the empty gripper will always trigger a transition from **STATE 3** to **STATE 1**.

Figure 5.35 is the simulation setup to test the proposed foraging system. Unfortunately, the sub-system for gripper control is not available and therefore cannot be implemented. Consequently, in order to allow this experiment to continue, this sub-system will always successfully complete its task—this is simulated by allowing the system to simply pass through **STATE 2** and **STATE 4**. It should be

noted that this does not, in any way, invalidate the experiment on the main system. Evidently, as shown in Figure 5.36, the foraging system has been found to be able to perform its task successfully.

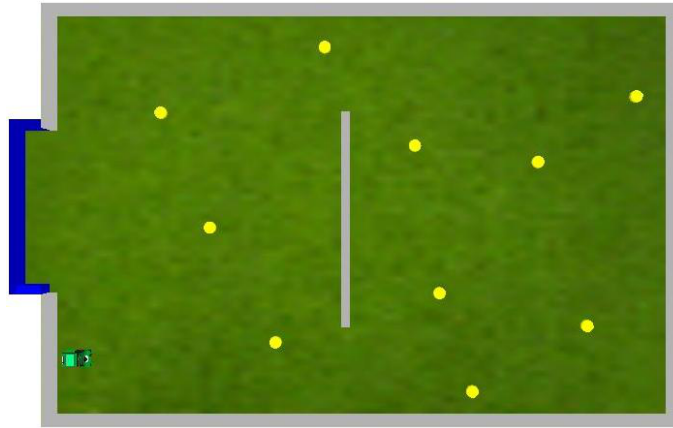


Figure 5.35: Initial environmental setup for a foraging application

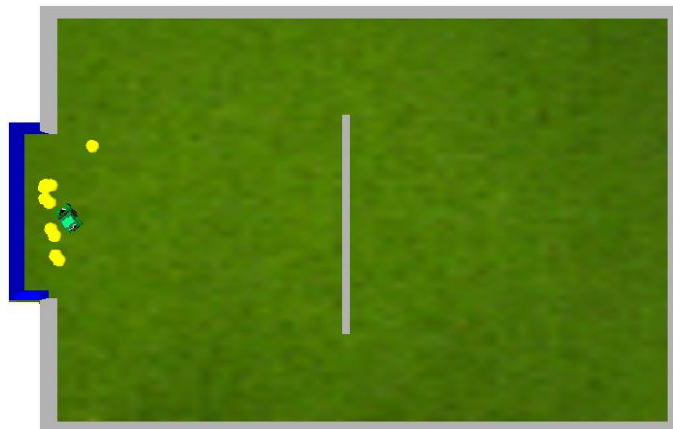


Figure 5.36: All target objects have been successfully placed at the home area

Similar to the foraging system, a self-recharging system can be simulated using the same approach. In this application, a robot will keep following a moving target robot until its battery level has reach a critically low level. At this point, the robot system should find a battery station to recharge its battery before it is able to execute the target tracking task again. As illustrated in Figure 5.37, it is assumed that only two states will be involved in the development of the system.

By default, the system will remain active in **STATE 1**, where the main system activates a fully evolved GEASM system trained for target tracking. However, when the battery level has reached a pre-defined minimum level, the main system changes state to **STATE 2**, and the robot starts to find a battery station. For this, a fully evolved GEASM system—trained for search and exploration—will be invoked. The system will remain at **STATE 2** until the battery has been fully recharged, and the system will be put back into **STATE 1**.

It is important to highlight that a new state can always be added to the current system. The requirement to add more states in the system may vary depending on the implementation of each robot application. For an example, in order to create a more complete implementation of the system, a state **STATE 3** that allows the robot to be attached perfectly at the battery recharging station can be added into the main system. In this case, a sub-system which implements a docking algorithm can be used and will be activated when the robot has found the charging station.

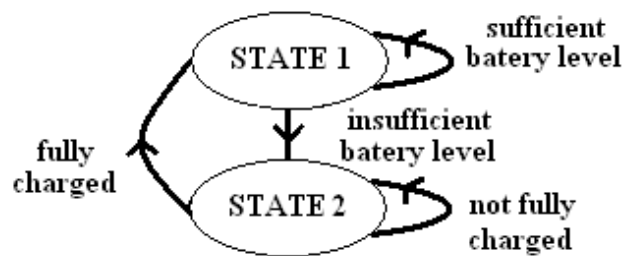


Figure 5.37: Implementing an autonomous battery recharging system using FSM

In short, similar to many other ER approaches, the proposed method is unable to eliminate the limitation of using an evolutionary algorithm (EA)—the time required for training a neural network until a desired robot control system has been achieved. However, as presented in this section, the evolved GEASM can be used for many other robots’ missions without any further training phase. This flexibility offered by the GEASM is worthwhile as the robot control system can be re-utilised in many ways.

5.5 Discussion

Whether it is an arbitration, or a command fusion, the ASM plays an important role in an implementation of a behavior-based system. Depending on the robot's current situation and task, the system might need to employ a different category of ASM during the execution of the system. However, there are no general rules that can be used as a guideline to determine what type of ASM is best employed at every decision cycle. This issue will become more complicated when the robot system is operating in an unknown, dynamic environment as the desired action will become more unpredictable. In this research, a GA is used to modify the weights of an ANN in order to generate a suitable ASM for the execution of a behavior-based system, which may vary based on a given task. The proposed technique is employed without the requirement of implementing any existing ASM. Once the ANN is trained using GA with a satisfactory fitness value, the genetically evolved ASM (i.e. GEASM) will be able to produce appropriate actions: either by selecting an action of one behavior (i.e. arbitration type), or by combining suggested actions from several or all behaviors (i.e. command fusion).

From the experiments that have been carried out in Section 5.1, and Section 5.2, it is observed that the GEASM method has the ability to perform the action selection mechanism as arbitration, or command fusion depending on its current situation. The system will change its type of ASM in order to achieve the overall system goal. As an example, for a search and exploration application, without the existence of a target object within the robot views, the trained system will produce a **WF** behavior. This behavior has emerged as a result of fusing the information from the three basic behaviors (i.e. **Avoid Obstacle**, **Wander**, **Drive to Target**) and its environment. However, in a situation where the target object is clearly within the robot's view (i.e. without any obstacles in front of it), the system will absolutely select **Drive to Target** behavior and release the **WF** behavior in order to reach system objective. Another example that reflects the ability of the

Table 5.4: Summary on percentage of success in the execution of robot missions

Mission	GEASM	PB	VS	ER
Search & exploration	93%	67%	67%	63%
Target tracking	100%	25%	25%	75%

proposed system to change the type of ASM during the execution of its task has been suggested through the MSE analysis in a target tracking application. The results have demonstrated that most of the time **Track** behavior is selected for controlling the robot movement. The integration of the basic behaviors is mostly executed in a situation where the robot is surrounded by obstacles.

In Equation 2.2, it can be seen that besides the coordination function \mathbf{C} (i.e. method of ASM), the gain g_i plays an important role in providing useful information to the coordination function. Whether the coordination function is an arbitration category, or it is a command fusion type, the gain will be used to set the relative importance of the behavioral components. Even though a suitable gain has been initially assigned to each of the behaviors, still the system may need to change the gain value during the execution of the robot's task. This may be necessary in order to ensure the overall system objective can be achieved or maintained. If this is the case, other mechanisms are required to determine what value for each gain should be assigned in each situation. Within GEASM, this mechanism will be handled by the ANN and GA. As shown in Figure 5.23, Figure 5.24 and Figure 5.25, the MSE plot shows that most of the time, the TRACK behavior will dominate (i.e. MSE equal to 0) the system's control. However, at certain intervals, it can be seen that there are requirements to consider actions suggested from the other behaviors as well. The inconsistency in the fluctuation of the MSE plot within these intervals suggest that the gain assigned to each of the behaviors has varied. Clearly, depending on the robot's current situation, the proposed system has been assigned with a suitable gain in order to achieve the overall system objective.

Table 5.4 shows the percentage of success in the execution of search and exploration, and target tracking missions presented in Section 5.1.2, and Section 5.2 respectively. As reflected in the table, GEASM is able to perform successfully in both missions with a good percentage number, far better than the other compared control systems. However, it is worthwhile to note that there are still possibilities that the compared systems are able to perform better in the test environments. As discussed previously, gain in Equation 2.2 is an important parameter toward the success of a behavior-based system. Therefore, in this case, a higher percentage of success may be observed for PB and VS methods if the systems have been assigned with an appropriate gain. Unfortunately, there is no straightforward method to determine the appropriate gain value for each robot task. A number of experiments need to be carried out before an optimal initial gain can be manually tuned. As mentioned in the earlier paragraph, the initial gain may no longer be appropriately applied to the control system due to some changes in the interaction between system's behaviors and its environment. Clearly, in this case, the requirement to have a different set of gains for the behavioral modules becomes higher for the system to be executed in different environment scenarios. In all experiments conducted for this project, a fixed gain has been assigned to each of the behaviors during the execution of the robot control systems. This might be the reason why PB, and VS are able to perform only in some of the test environments, presented in the thesis.

Since the approach of the proposed method is closely related to the work in evolutionary robotic fields, a comparison with conventional ER has been made. In evolutionary robotics research, the aim of this field is to develop a suitable robot control system automatically. Using this technique, the decomposition of behavioral modules, and the development of coordination mechanisms will be merged during the training phase via a self-organising process (refer Appendix F). Since the scope to train the robot system is wider than the proposed technique (i.e. training only for a coordination mechanism), conventional ER may require a longer training phase in order to get a better performance. As shown in the previous results, the performance of conventional ER experiments are lower with the respect to GEASM.

Physical damage is one of the main problems that has been highlighted by ER researchers when implementing GAs in robot control systems. The critical error suppressor introduced in the proposed control architecture has partially lifted this problem. The major role of the critical error module in this thesis is to ensure that the robot does not get into a serious error state due to the random value generated especially at the early stages of the training phase. In addition, this critical error module is used in the fitness evaluation. A generic fitness function has been defined in Equation 3.3, where the information manipulated from the critical error suppressor is used to measure f_{Err} component.

Even though the success of implementing EAs (e.g. GA) relies significantly on a good fitness function definition, no specific rule has been outlined on how to design it. Therefore, the proposed generic fitness function will indirectly reduce the problem of defining an appropriate fitness function. On top of that, as mentioned in the earlier chapter, the f_{Err} term has the ability to converge the genetic population towards a desirable solution, particularly in the early generations. The information from the critical error suppressor can distinguish the candidate solutions especially when most (if not all) of them failed to complete their tasks.

When using GA for any applications, the formulation of the fitness function is of critical importance. Many aspects of designing a fitness function with various examples, particularly in the robot control area, have been discussed in [71]. The surveyed research has compared the amount of prior information being used to successfully evolve a specific controller to perform a given task. For a more practical approach, fitness functions should be defined using data that is available in a real environment. Without relying on data that is only available from the simulator, the system will be more adaptive and more robust to a changing environment where further evolution can be carried out (if necessary) on a real robot platform.

In the implementation, the proposed fitness function has been created based on the availability of parameters on actual robots. For example, although

the distance-to-object parameter can actually be calculated in a simulation environment, it has not been used in the proposed fitness function. This is because the actual robot that was simulated would not know where the object is in a real mission, unless the robot is fitted with something like a global positioning system (GPS) device and the object's position is a known parameter. With a more realistic fitness function, the system should be able to adapt better to the real environment. In addition to that, the system can actually acquire training data whilst executing a mission that can be used to update the genetic population for ASM solutions. Even though this platform has been designed with the possibility of running an on-line training (if needed) to adapt with the environment changes, the results of the GEASM system show that further training is unnecessary. Based-on the results, it can be safely concluded that the GEASM system is very adaptive to its mission environment, and is able to produce appropriate output during the execution of its task.

5.6 Summary

The results for testing GEASM system and its respective benchmark systems has been presented. The trained system has successfully demonstrated its functionality as an action selection mechanism. Results have also shown that the proposed ASM has a high competency in integrating multiple control signals suggested by the behaviors in the system. In fact, the trained system has the ability of changing the type of action selection mechanism either to integrate the behaviors (command fusion) or to select one from several behaviors (arbitration) during the execution of its task. GEASM managed to switch the type of ASM being carried out based on its platform's current situations. Overall, the results presented in the two test applications executed in unknown environments have shown that the system was able to accomplish the missions efficiently. Moreover, the presented results have shown that GEASM is able to tolerate up to a certain degree of image noises.

Chapter 6

Summary and Future Work

This chapter concludes the thesis with an overall summary and some discussions on possible extensions of the presented work.

6.1 Thesis Summary

The action selection mechanism in a behavior-based system presents a very interesting problem for intelligent systems to solve. Producing a viable action that can generally satisfy primary mission objectives while preserving information gathered by system behaviors can be very tricky. However, a solution to such problems can usually be identified by searching for a familiar pattern in the problem at hand.

As with many other pattern recognition problems, using a neural network to identify an optimised action selection mechanism can be very effective—especially when no deterministic algorithm can be used. A neural network has the capability to identify a pattern even when the solution set is not clearly segmented (not linearly separable—e.g. the XOR problem). While selecting an action from multiple sources mostly tends to favour one (winner-take-all scenario), a neural network may be able to find a middle ground between the suggested actions.

The presented evolving neural network for the action selection mechanism in behavior-based systems utilises genetic algorithm as its learning mechanism. An issue with other commonly used learning algorithms based on the gradient-descent method (e.g. back-error propagation) is the possibility of converging towards local minima (a solution for only a small part of the bigger problem). Genetic algorithms can theoretically cover the whole solution space and avoid the local minima issue.

The most important component in ensuring the success of a genetic evolution is the determination of fitness value for the current weights in the neural network. This usually depends on an application but in this thesis, it has been generalised into a function of two distinct components: performance and error. The first component is an obvious choice, but performance indicators can sometimes be inaccurate, especially in simple tasks. This fact has influenced the addition of the second component. The error component can help in separating false results by looking at a result from another point of view. For example, the performance indicator for a mobile platform that simply circles around when executing a search behavior may be relatively high because it does not come across any obstacle. However, this clearly will not help in searching a large area and therefore, a suitable error definition for movements in a circle can help in producing a more balanced indicator for the system.

In general, based on simulation results, the proposed method has executed selected robot control missions with good performance. Analysis shows that the genetically evolved action selection mechanism (GEASM) can switch its coordination method from either the selection of one primary behavior or the integration of multiple behaviors with suitable gains, in order to produce appropriate robot action. In addition to that, the GEASM may have also produced capabilities that are only synthesizable by looking at the dynamic parameters in a real environment. Realistically, to implement such an action selection mechanism module using rule-based coding can be quite costly, if not impossible, especially considering all the uncertainties in an unknown environment.

6.2 Key Findings

Using genetic algorithms as the learning method has been proven to be quite successful, but it is clear that the process itself is rather time consuming. However, if a general purpose solution that requires no further training can be found, this constraint becomes non-relevant. In any case, a system that does not need any rules to be defined for it to operate in an unknown environment can surely be considered ‘intelligent’. This is consistent with the use of behavior-based systems to imitate real biological systems.

Controlling the population size during training is also an important factor in genetic algorithms. Specifying birth rate allows new solution candidates to be considered. This is important in providing a possibility of escaping the local minima trap. Meanwhile, specifying a death rate allows purging of solution candidates that do not perform at all, but this must be controlled so that most of the existing population can have enough chances to prove their capabilities. Including mutation capabilities among the newborns is another reason why the genetic algorithms can usually overcome the local minima trap. This gives the general population a chance in exploring new possible solutions.

Inserting critical error detection in the implementation of the behavior-based mobile robot model is made necessary by the fact that randomly created weights can produce damaging movement. For example, the neural network may produce an output like moving forward even when there is a wall in front. This is logical and is actually quite necessary from a learning point of view so that it can cover all possible solution space. This is one of the main reason why the genetic algorithm is said to be able to avoid the local minima issue. However, it is important to remember that this random input may result in an invalid or erratic movement.

The generic fitness function used in the presented system is only based on two components—performance and error. This is possible since the chosen applications are still relatively simple and utilise a minimum number of inputs. For a more

complex mission with more than one performance indicator, a suitable weighting policy needs to be defined for each performance. The contribution from the error component may need to be made dynamic and gradually changed while going through the learning process. This is simply because the error component may inhibit the output (when the contribution is too high) or may not help at all (when the contribution is too low) in separating the false positive results.

As stated in Equation 2.2, each behavior is usually assigned a gain value that represents the importance of that particular behavior towards the overall system response. A simple Mean Squared Error (MSE) analysis in a target tracking task (discussed in Chapter 5) following manual observations in search and exploration mission simulations has shown that in the proposed architecture, this value actually needs to be continuously changed based on the system environment. This confirms early assumptions that it is very difficult, if not impossible, to manually design such a system based on pre-determined rules and decision logic.

The proposed system is meant for a vision-based mobile robot platform. The method used to extract data from the captured image is based only on simple image processing—pixel counting. This results in a modest processing time for a relatively small field of view. With a more reliable vision-based platform, powered by a more powerful hardware, the system can be a powerful intelligent system for a more complex task.

It is also important to highlight the advantages of GEASM model which have been observed from the results presented in Chapter 5. The first advantage is the generalization capability that has been demonstrated by the proposed GEASM system during simulation. When implementing ANN for any application, it is important to observe the ability of the neural network to handle unseen data (i.e. generalization capability). One of the main prerequisite for a neural network to avoid poor generalization is to be able to provide sufficient data during its training phase. Alternatively, as proposed in GEASM, a genetic algorithm can be used to train the neural network engine without the need for any training data. Conducted

experiments have shown that GEASM is able to outperform the other systems that have been used as comparisons. Thus, it is safe to claim that GEASM has a good generalization capability as it is able to operate in various unknown environments even it is trained without any prior data.

It is also observed that GEASM system is able to exploit hand-designed behaviors in order to synthesize new elementary behaviors that are needed in current scenario. For example, in search and exploration task, three behavioral sets (i.e. Avoid obstacle, Wander, Drive to Target) have been defined to execute the mission. During the exploration of an environment, instead of executing one of the behavioral sets mentioned earlier or generating an arbitrary cumulative action, the proposed GEASM system actually performed a well-known wall following behavior to search a target object. It is important to note that a wall following is commonly implemented when a robot is exploring an unknown area, especially in a maze-like environment. From this observation, it is clear that GEASM has the capability to synthesize any behavior as needed.

Finally, it has been found that GEASM can be a more effective ASM when compared to the other two systems: PB (arbitration type) and VS (command fusion type). One of the strength in GEASM is its capability to switch the type of ASM either to select one behavior from many (i.e. arbitration), or to combine the suggestions from several behaviors (i.e. command fusion), depending on its current situation. As discussed and presented in Chapter 5, this feature is the key success for GEASM to successfully perform the two tasks.

6.3 Limitation and Future Work

When working with any simulation software, one question is surely unavoidable—will a developed system still work in a different simulation software with a different simulation model. Different mobile robot simulation software may have different data models representing the same hardware. To analyse this, some ground work has

actually been done to accommodate the possibility of implementing the same system on a different mobile robot simulation software. This could give some practical feedback, while expanding the range of possible mission scenarios for the system to train.

The presented work has only so far been tested on a simulator. Although the code can theoretically be directly transferred to a real robot implementation, there are some things that still need extra attention. For example, the movement model used in the simulator may not be accurate and therefore presents an error in the trained system. Another foreseen issue is the fact that actual robot hardware sometimes has a different executable binary compared to a simulation executable binary even when it has the same C code. This may not be critical, but it still presents a possible error. So, one of the things that needs to be done in the next stage is to actually implement this on real hardware and analyse the performance of the real system.

All the experiments presented here have been executed on a single robot system. However, the more interesting question in robotics research is how well a team of robots can work together on the same task and environment. One of the advantages offered by a multiple robot system is the flexibility and robustness through parallelism and redundancy. On the other hand, one problem that will surely need detailed attention would be the communication issue—how a robot can (or needs to) relay (or exchange) information with another robot in the team. The performance of a behavior-based system in a multiple-robot scenario is certainly something that can be further investigated.

Genetic algorithms have traditionally been using offline training due to the huge storage and computing power requirements. With a more powerful processing hardware and significant sized non-volatile flash memory, it would be nice to find out if this system can be evolved online—while executing ‘training’ missions. This would make the system even more robust and very similar to any naturally ‘intelligent’ biological system.

On a smaller scale, studies on the effect of genetic parameters used in the system (i.e. birth/death/mutation rate) would help in understanding the evolution process. Variations in the weight value of fitness calculation should also be an interesting case study.

References

- [1] R. C. Arkin, *Behavior-based Robotics*. The MIT Press, 1998.
- [2] M. Mataric, *The Robotics Primer*. Intelligent robotics and autonomous agents, MIT Press, 2007.
- [3] M. J. Mataric, “Behavior-based control: Examples from navigation, learning, and group behavior,” *Journal of Experimental and Theoretical Artificial Intelligence*, special issue on *Software Architectures for Physical Agents*, vol. 9, pp. 323–336, 1997.
- [4] L. Parker, “Alliance: an architecture for fault tolerant multirobot cooperation,” *Robotics and Automation, IEEE Transactions on*, vol. 14, pp. 220–240, Apr 1998.
- [5] S. Yamada and J. Saito, “Adaptive action selection without explicit communication for multirobot box-pushing,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 31, pp. 398–404, Aug 2001.
- [6] S. C. Gadanho, “Learning behavior-selection by emotions and cognition in a multi-goal robot task,” *Journal of Machine Learning Research*, vol. 4, pp. 385–412, 2003.
- [7] A. W. Stroupe and T. R. Balch, “Value-based action selection for observation with robot teams using probabilistic techniques,” *Robotics and Autonomous Systems*, vol. 50, no. 2-3, pp. 85–97, 2005.

- [8] R. Huq, G. Mann, and R. Gosine, "Behavior-modulation technique in mobile robotics using fuzzy discrete event system," *Robotics, IEEE Transactions on*, vol. 22, pp. 903–916, Oct. 2006.
- [9] P. Pirjanian, "Behavior coordination mechanisms - state-of-the-art," Tech. Rep. IRIS-99-375, Institute of Robotics and Intelligent Systems IRIS, University of Southern California, October 1999.
- [10] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, pp. 14–23, Mar 1986.
- [11] I. H. Suh, M. J. Kim, S. Lee, and B. J. Yi, "A novel dynamic priority-based action-selection-mechanism integrating a reinforcement learning," *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 3, pp. 2639–2646, April-1 May 2004.
- [12] G. Kim and W. Chung, "Navigation behavior selection using generalized stochastic petri nets for a service robot," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, pp. 494–503, July 2007.
- [13] A. M. Farahmand, M. N. Ahmadabadi, C. Lucas, and B. N. Araabi, "Interaction of culture-based learning and cooperative co-evolution and its application to automatic behavior-based system design," *IEEE Trans. Evolutionary Computation*, vol. 14, no. 1, pp. 23–57, 2010.
- [14] F. Montes-Gonzalez and C. M. Contreras, "The evolution of motivated and modulated robot selection," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.
- [15] E. Daglarli, H. Temeltas, and S. M. Yesiloglu, "Behavioral task processing for cognitive robots using artificial emotions.," *Neurocomputing*, vol. 72, no. 13–15, pp. 2835–2844, 2009.

- [16] G. Petrosino, D. Parisi, and S. Nolfi, “Selective attention enables action selection: evidence from evolutionary robotics experiments,” *Adaptive Behaviour*, vol. 21, no. 5, pp. 356–370, 2013.
- [17] R. Arkin, “Motor schema based navigation for a mobile robot: An approach to programming by behavior,” *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4, pp. 264–271, Mar 1987.
- [18] J. Rosenblatt, “Damn: A distributed architecture for mobile navigation,” *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 1, pp. 339 – 360, 1997.
- [19] J. Riecki and J. Roning, “Reactive task execution by combining action maps,” *International Conference on Integrated Robots and Systems*, pp. 224–230, 1997.
- [20] M. Scheutz and V. Andronache, “Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems,” *IEEE Transactions on Systems, Man, and Cybernetics, PartB*, vol. 34, no. 6, pp. 2377–2395, 2004.
- [21] M. Proetzsch, T. Luksch, and K. Berns, “Development of complex robotic systems using the behavior-based control architecture ib2c,” *Robotics and Autonomous Systems*, vol. 58, no. 1, pp. 46–67, 2010.
- [22] M. Knudson and K. Tumer, “Adaptive navigation for autonomous robots,” *Robot. Auton. Syst.*, vol. 59, pp. 410–420, June 2011.
- [23] L. T. Bui, Z. Michalewicz, E. Parkinson, and M. B. Abello, “Adaptation in dynamic environments: A case study in mission planning,” *IEEE Trans. Evolutionary Computation*, vol. 16, no. 2, pp. 190–209, 2012.
- [24] I. G. del Amo, D. A. Pelta, J. R. Gonzalez, and A. D. Masegosa, “An algorithm comparison for dynamic optimization problems,” *Applied Soft Computing*, vol. 12, no. 10, pp. 3176 – 3192, 2012.

- [25] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA: MIT Press, 2000.
- [26] M. J. Mataric and F. Michaud, "Behavior-based systems.," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 891–909, Springer, 2008.
- [27] M. J. Mataric, "Learning in behavior-based multi-robot systems: Policies, models, and other agents," *Cognitive Systems Research, special issue on Multi-Disciplinary Studies of Multi-Agent Learning*, vol. 2, pp. 81–93, 2001.
- [28] R. A. Brooks, "A robust layered control system for a mobile robot," tech. rep., Cambridge, MA, USA, 1985.
- [29] R. A. Brooks, "Intelligence without representation," *Artif. Intell.*, vol. 47, no. 1-3, pp. 139–159, 1991.
- [30] M. J. Mataric and F. Michaud, "Behavior-based systems," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 891–909, Springer, 2008.
- [31] A. W. Stroupe and T. Balch, "Value-based action selection for observation with robot teams using probabilistic techniques," *Robotics and Autonomous Systems*, vol. 50, no. 2-3, pp. 85–97, 2005. Multi-Robots in Dynamic Environments.
- [32] Y. Nojima, "Multi-objective behavior coordination based on sensory network for multiple mobile robots," in *Robotic Intelligence in Informationally Structured Space, 2009. RIIS '09. IEEE Workshop on*, pp. 66–72, march 2009.
- [33] A. Stroupe, T. Huntsberger, A. Okon, H. Aghazarian, and M. Robinson, "Behavior-based multi-robot collaboration for autonomous construction tasks," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 1495–1500, 2005.

- [34] O. Ahmad, J. Iqbal, and M. Khan, “Coordination without communication: Finite state automation in behavior based multi-robot systems,” in *Information and Emerging Technologies, 2007. ICIET 2007. International Conference on*, pp. 1–4, 6-7 2007.
- [35] R. A. Brooks, “Intelligence without representation,” *Artificial Intelligence*, vol. 47, pp. 139–160, 1991.
- [36] M. Mataric, “Designing emergent behaviors: From local interactions to collective intelligence, from animals to animats,” *Proceedings Second International Conference on Simulation of Adaptive Behaviors*, pp. 432–441, 1993.
- [37] J. A. Fernandez-Leon, G. G. Acosta, and M. A. Mayosky, “Behavioral control through evolutionary neurocontrollers for autonomous mobile robot navigation,” *Robot. Auton. Syst.*, vol. 57, no. 4, pp. 411–419, 2009.
- [38] S. A. Rahim, A. M. Yusof, and T. Bräunl, “Genetically evolved action selection mechanism in a behavior-based system for target tracking,” *Neurocomputing*, vol. 133, pp. 84–94, 2014.
- [39] J. Kosecká and R. Bajcsy, “Discrete event systems for autonomous mobile agents,” *Robotics and Autonomous Systems*, vol. 12, no. 3-4, pp. 187–198, 1994.
- [40] P. Maes, “The dynamics of action selection,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 991–997, 1989.
- [41] D. Hanon, G. Strugeon, and R. Mandiau, “A behaviour based decisional model using vote,” in *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, vol. 1, pp. 39–44, 28-30 2005.

- [42] F. M. Montes-Gonzalez and A. Marin-Hernandez, “Central action selection using sensor fusion,” *Mexican International Conference on Computer Science*, pp. 289–296, 2004.
- [43] J. Jaafar and E. McKenzie, “A fuzzy action selection method for virtual agent navigation in unknown virtual environments,” *Journal of Uncertain Systems*, vol. 2, no. 2, pp. 144–154, 2008.
- [44] A. Bonarini, G. Invernizzi, T. H. Labella, and M. Matteucci, “An architecture to coordinate fuzzy behaviors to control an autonomous robot,” *Fuzzy Sets and Systems*, vol. 134, no. 1, pp. 101–115, 2003.
- [45] P. Pirjanian and M. Mataric, “Multi-robot target acquisition using multiple objective behavior coordination,” in *In IEEE International Conference on Robotics and Automation*, 2000.
- [46] Y. Nojima, “Multi-objective behavior coordination based on sensory network for multiple mobile robots,” in *Robotic Intelligence in Informationally Structured Space, 2009. RIISS '09. IEEE Workshop on*, pp. 66–72, march 2009.
- [47] R. Arkin and D. MacKenzie, “Temporal coordination of perceptual algorithms for mobile robot navigation,” *Robotics and Automation, IEEE Transactions on*, vol. 10, pp. 276–286, Jun 1994.
- [48] J. A. Fernandez-Leon, G. G. Acosta, and M. A. Mayosky, “Behavioral control through evolutionary neurocontrollers for autonomous mobile robot navigation,” *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 411–419, 2009.
- [49] J. Rosenblatt, “Damn: A distributed architecture for mobile navigation - thesis summary,” in *Journal of Experimental and Theoretical Artificial Intelligence*, pp. 339–360, AAAI Press, 1995.
- [50] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *Int. J. Rob. Res.*, vol. 5, pp. 90–98, Apr. 1986.

- [51] A. Jayasiri, G. Mann, and R. Gosine, “Behavior coordination of mobile robotics using supervisory control of fuzzy discrete event systems,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, pp. 1224–1238, Oct 2011.
- [52] A. Jayasiri, G. Mann, and R. Gosine, “Behavior coordination of mobile robotics using supervisory control of fuzzy discrete event systems,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, no. 5, pp. 1224–1238, 2011.
- [53] O. Zapata, G. Acosta, and J. Jimenez, “Coordination scheme and behavior fusion based on fuzzy weighting factors for a mobile robotic navigation,” *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 13, pp. 18–26, Jan 2015.
- [54] T. Daji, W. Shaoping, and A. El Kamel, “Behavior modulation of autonomous mobile robotics using fuzzy control method,” in *Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on*, pp. 819–824, June 2014.
- [55] A. Ram, R. Arkin, G. Boone, and M. Pearce, “Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation,” *Adaptive Behavior*, vol. 2, pp. 277–304, 1994.
- [56] P. Maes and R. A. Brooks, “Learning to coordinate behaviors,” in *Proceedings of the eighth National conference on Artificial intelligence*, vol. 2 of *AAAI’90*, pp. 796–802, AAAI Press, 1990.
- [57] K. Jolly, R. S. Kumar, and R. Vijayakumar, “Intelligent task planning and action selection of a mobile robot in a multi-agent system through a fuzzy neural network approach,” *Engineering Applications of Artificial Intelligence*, vol. 23, no. 6, pp. 923 – 933, 2010.

- [58] H. Mehdi and O. Boubaker, “Impedance controller tuned by particle swarm optimization for robotic arms,” *International Journal of Advanced Robotic Systems*, vol. 8, no. 5, pp. 93–103, 2011.
- [59] R. Batllori, C. Laramee, W. Land, and J. Schaffer, “Evolving spiking neural networks for robot control,” *Procedia Computer Science*, vol. 6, no. 0, pp. 329 – 334, 2011.
- [60] N. Navarro-Guerrero, C. Weber, P. Schroeter, and S. Wermter, “Real-world reinforcement learning for autonomous humanoid robot docking,” *Robotics and Autonomous Systems*, vol. 60, no. 11, pp. 1400 – 1407, 2012.
- [61] H. Qu, K. Xing, and T. Alexander, “An improved genetic algorithm with co-evolutionary strategy for global path planning of multiple mobile robots,” *Neurocomputing*, vol. 120, pp. 509–517, 2013.
- [62] D. Maravall, J. de Lope, and J. A. Martin H., “Hybridizing evolutionary computation and reinforcement learning for the design of almost universal controllers for autonomous robots,” *Neurocomputing*, vol. 72, no. 4-6, pp. 887 – 894, 2009.
- [63] D. Ashlock and C. Lee, “Agent-case embeddings for the analysis of evolved systems,” *IEEE Trans. Evolutionary Computation*, vol. 17, no. 2, pp. 227–240, 2013.
- [64] N. Nedjah and L. de Macedo Mourelle, eds., *Evolvable Machines: Theory & Practice*. Springer, 2004.
- [65] M. J. Mataric and D. Cliff, “Challenges in evolving controllers for physical robots,” *Robotics and Autonomous Systems*, vol. 19, no. 1, pp. 67–83, 1996.
- [66] G. Capi and K. Doya, “Evolution of recurrent neural controllers using an extended parallel genetic algorithm,” *Robotics and Autonomous Systems*, vol. 52, no. 2-3, pp. 148 – 159, 2005.

- [67] A. L. Nelson and E. Grant, “Using direct competition to select for competent controllers in evolutionary robotics,” *Robotics and Autonomous Systems*, vol. 54, no. 10, pp. 840–857, 2006.
- [68] W.-P. Lee, J. Hallam, H. H. Lund, and E. U. K., “Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots,” in *In Proceedings of IEEE 4th International Conference on Evolutionary Computation*, pp. 495–499, IEEE Press, 1997.
- [69] S. Chernova and M. M. Veloso, “An evolutionary approach to gait learning for four-legged robots,” in *In Proceedings of IROS’04*, pp. 2562–2567, IEEE, September 2004.
- [70] J.-C. Zufferey, D. Floreano, M. van Leeuwen, and T. Merenda, “Evolving vision-based flying robots,” in *Biologically Motivated Computer Vision*, pp. 592–600, 2002.
- [71] G. J. B. Andrew L. Nelson and L. Doitsidis, “Fitness functions in evolutionary robotics: A survey and analysis,” *Robotic and Autonomous Systems*, vol. 57, pp. 345–370, April 2009.
- [72] Z. Miljkovic, M. Mitic, M. Lazarevic, and B. Babic, “Neural network reinforcement learning for visual control of robot manipulators,” *Expert Systems with Applications*, vol. 40, no. 5, pp. 1721 – 1736, 2013.
- [73] S. Aydin, “A fuzzy clustering neural networks for motion equations of synchro-drive robot,” *Expert Systems with Applications*, vol. 37, no. 12, pp. 7819 – 7824, 2010.
- [74] M. Ghatee and A. Mohades, “Motion planning in order to optimize the length and clearance applying a hopfield neural network,” *Expert Systems with Applications*, vol. 36, no. 3, Part 1, pp. 4688 – 4695, 2009.
- [75] H.-M. Yen, T.-H. S. Li, and Y.-C. Chang, “Adaptive neural network based tracking control for electrically driven flexible-joint robots without velocity

- measurements,” *Computers & Mathematics with Applications*, vol. 64, no. 5, pp. 1022 – 1032, 2012.
- [76] S. Soyguder, “Intelligent control based on wavelet decomposition and neural network for predicting of human trajectories with a novel vision-based robotic,” *Expert Systems with Applications*, vol. 38, no. 11, pp. 13994 – 14000, 2011.
- [77] C.-Z. Pan, X.-Z. Lai, S. X. Yang, and M. Wu, “An efficient neural network approach to tracking control of an autonomous surface vehicle with unknown dynamics,” *Expert Systems with Applications*, vol. 40, no. 5, pp. 1629 – 1635, 2013.
- [78] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.
- [79] B. L. Jasmin Velagic, Nedim Osmic, “Neural network controller for mobile robot motion control,” *International Journal of Intelligent Systems and Technologies*, vol. 3, pp. 127–132, Summer 2008.
- [80] D. Gachet, M. A. Salichs, L. Moreno, J. R. Pimentel, and D. Ingenier, “Learning emergent tasks for an autonomous mobile robot,” in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 94)*, pp. 290–297, 1994.
- [81] X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
- [82] K. Chellapilla and D. Fogel, “Evolving neural networks to play checkers without relying on expert knowledge,” *Neural Networks, IEEE Transactions on*, vol. 10, no. 6, pp. 1382–1391, 1999.
- [83] C.-F. Juang, “Combination of online clustering and Q-value based GA for reinforcement fuzzy system design,” *Fuzzy Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 289–302, 2005.

- [84] F. Gomez, J. Schmidhuber, and R. Miikkulainen, “Accelerated neural evolution through cooperatively coevolved synapses,” *J. Mach. Learn. Res.*, vol. 9, pp. 937–965, 2008.
- [85] D. Floreano and F. Mondada, “Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot,” in *Proceedings of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3: From Animals to Animats 3*, SAB94, (Cambridge, MA, USA), pp. 421–430, MIT Press, 1994.
- [86] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA, USA: MIT Press, 2000.
- [87] A. C. Nearchou, “Adaptive navigation of autonomous vehicles using evolutionary algorithms,” *Artificial Intelligence in Engineering*, vol. 13, no. 2, pp. 159 – 173, 1999.
- [88] K. Asami, H. Hagiwara, and M. Komori, “Visual navigation system based on evolutionary computation on fpga for patrol service robot,” in *Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on*, pp. 295–298, Oct 2012.
- [89] R. D. Beer, R. D. Quinn, H. J. Chiel, and R. E. Ritzmann, “Biologically inspired approaches to robotics: What can we learn from insects?,” *Commun. ACM*, vol. 40, pp. 30–38, Mar. 1997.
- [90] T. Li, Y.-T. Su, S.-W. Lai, and J.-J. Hu, “Walking motion generation, synthesis, and control for biped robot by using pgrl, lpi, and fuzzy logic,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, pp. 736–748, June 2011.

- [91] J.-G. Juang, “Fuzzy neural network approaches for robotic gait synthesis,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 30, pp. 594–601, Aug 2000.
- [92] Z. Liu, Y. Zhang, and Y. Wang, “A type-2 fuzzy switching control system for biped robots,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, pp. 1202–1213, Nov 2007.
- [93] I. sik Lim, O. Kwon, and J. H. Park, “Gait optimization of biped robots based on human motion analysis,” *Robotics and Autonomous Systems*, vol. 62, no. 2, pp. 229 – 240, 2014.
- [94] P. Vadakkepat, X. Peng, B. K. Quek, and T. H. Lee, “Evolution of fuzzy behaviors for multi-robotic system,” *Robotics and Autonomous Systems*, vol. 55, no. 2, pp. 146 – 161, 2007.
- [95] S.-W. Moon and S.-G. Kong, “Block-based neural networks,” *Neural Networks, IEEE Transactions on*, vol. 12, pp. 307–317, Mar 2001.
- [96] H. Zhou, G. Guo, and M. Liu, “Ga-aided elman neural network controller for behavior-based robot,” *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, vol. 2, pp. 9068–9072, 0-0 2006.
- [97] H. Hagaras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke, “Evolving spiking neural network controllers for autonomous robots,” *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, pp. 4620–4626, April-1 May 2004.
- [98] F. Mondada and D. Floreano, “Evolution of neural control structures: Some experiments on mobile robots,” *Robotics and Autonomous Systems*, vol. 16, pp. 183–195, 1995.
- [99] F. Hoffmann and J. C. S. Z. Montealegre, *Soft Computing and Industry: Recent Applications*, ch. Evolution of a Tactile Wall-Following Behavior in Real Time. Springer London, 2002.

- [100] P. Petrovič, “Evolving behavior coordination for mobile robots using distributed finite-state automata,” *Frontiers in Evolutionary Robotics*, vol. 2008, pp. 413–438, 2008.
- [101] S. Nolfi and S. Nolfi, “Using Emergent Modularity to Develop Control Systems for Mobile Robots,” in *Adaptive Behavior*, pp. 343–363, 1996.
- [102] J. Grefenstette, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. Taylor & Francis, 2013.
- [103] D. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, 1999.
- [104] A. Zalzala and P. Fleming, *Genetic Algorithms in Engineering Systems*. Institution of Electrical Engineers, 1997.
- [105] R. Rojas, *Neural Networks: A Systematic Introduction*. Springer, 1996.
- [106] S. Haykin, *Neural Networks and Learning Machines*. Pearson, 2009.
- [107] Kurt Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [108] S. Tzafestas, *Introduction to Mobile Robot Control*. Elsevier, 2014.
- [109] K. Swingler, *Applying neural networks - a practical guide*. Academic Press, 1996.
- [110] S. Karsoliya, “Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture,” *International Journal of Engineering Trends and Technology*, vol. 3, no. 1, pp. 714–717, 2012.
- [111] A. Weigend, “On overfitting and the effective number of hidden units,” in *Proceedings of the 1993 Connectionist Models Summer School* (M. C. Mozer,

- P. Smolensky, and A. S. Weigend, eds.), pp. 335–342, Lawrence Erlbaum Associates, 1994.
- [112] P. McCullagh and J. A. Nelder, *Generalized linear models (Second edition)*. London: Chapman & Hall, 1989.
- [113] S. Geman, E. Bienenstock, and R. Doursat, “Neural networks and the bias/variance dilemma,” *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.
- [114] T. Rajkumar and J. Bardina, “Training data requirement for a neural network to predict aerodynamic coefficients,” tech. rep., SAIC@NASA Ames Research Center, Moffett Field, California, USA, 2003.
- [115] C.-L. Chen, D. B. Kaber, and P. G. Dempsey, “Using feedforward neural networks and forward selection of input variables for an ergonomics data classification problem,” *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 14, no. 1, pp. 31–49, 2004.
- [116] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, “Data preprocessing for supervised learning,” *International Journal of Computer Science*, vol. 1, no. 2, 2006.
- [117] G. Hinton, “Supervised learning in multilayer neural networks,” in *The MIT Encyclopedia of the Cognitive Sciences* (R. Wilson and F. Keil, eds.), pp. 814–816, MIT Press, 1999.
- [118] P. Dayan, M. Sahani, and G. Deback, “Unsupervised learning,” in *In The MIT Encyclopedia of the Cognitive Sciences*, The MIT Press, 1999.
- [119] M. Mitchell, *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. The MIT Press, February 1998.
- [120] B. Müller and J. Reinhardt, *Neural Networks: An Introduction*. Physics of Neural Networks, Springer Berlin Heidelberg, 2013.

- [121] P. Tonupunuri and S. I. U. at Carbondale. Computer Science, *Evolutionary Based Path-finding for Mobile Agents in Sensor Networks*. Southern Illinois University at Carbondale, 2008.
- [122] Á. Jobbágy, *5th European Conference of the International Federation for Medical and Biological Engineering 14 - 18 September 2011, Budapest, Hungary*. IFMBE Proceedings, Springer Berlin Heidelberg, 2012.
- [123] S. Sumathi, T. Hamsapriya, and P. Surekha, *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*. Springer, 2008.
- [124] P. Arena, L. Fortuna, M. Frasca, and L. Patane, “Learning anticipation via spiking networks: Application to navigation control,” *Neural Networks, IEEE Transactions on*, vol. 20, no. 2, pp. 202–216, 2009.
- [125] C.-F. Juang and Y.-C. Chang, “Evolutionary-group-based particle-swarm-optimized fuzzy controller with application to mobile-robot navigation in unknown environments,” *Fuzzy Systems, IEEE Transactions on*, vol. 19, pp. 379–392, April 2011.
- [126] N. Deshpande, E. Grant, and T. Henderson, “Target localization and autonomous navigation using wireless sensor networks - a pseudogradient algorithm approach,” *Systems Journal, IEEE*, vol. 8, pp. 93–103, March 2014.
- [127] P. Fabiani, H.-H. Gonzalez-Baos, J.-C. Latombe, and D. Lin, “Tracking an unpredictable target among occluding obstacles under localization uncertainties,” *Robotics and Autonomous Systems*, vol. 38, no. 1, pp. 31 – 48, 2002.
- [128] O. Yilmaz, “Oscillatory synchronization model of attention to moving objects,” *Neural Networks*, vol. 29 - 30, pp. 20 – 36, 2012.
- [129] F.-B. Duh and C.-T. Lin, “Tracking a maneuvering target using neural fuzzy network,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 1, pp. 16–33, 2004.

- [130] T. Bräunl, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Springer, 2008.
- [131] T. Bräunl, A. Koestler, and A. Waggershauser, “Fault-tolerant robot programming through simulation with realistic sensor models,” *International Journal of Advanced Robotic Systems (ARS)*, vol. 3, pp. 99–106, June 2006.
- [132] A. Koestler and T. Bräunl, “Mobile robot simulation with realistic error models,” *International Conference on Autonomous Robots and Agents, ICARA*, pp. pp. 46–51, December 2004.
- [133] M. Asada and H. Kitano, *RoboCup-98: Robot Soccer World Cup II*. No. no. 1604 in *Lecture Notes in Artificial Intelligence*, Springer Berlin Heidelberg, 1999.
- [134] X. Peng, “Combine color and shape in real-time detection of texture-less objects,” *Computer Vision and Image Understanding*, vol. 135, pp. 31 – 48, 2015.
- [135] S. Duffner and J.-M. Odobez, “Leveraging colour segmentation for upper-body detection,” *Pattern Recognition*, vol. 47, no. 6, pp. 2222 – 2230, 2014.
- [136] K. Lee, C. Lee, S.-A. Kim, and Y.-H. Kim, “Fast object detection based on color histograms and local binary patterns,” in *TENCON 2012 - 2012 IEEE Region 10 Conference*, pp. 1–4, Nov 2012.
- [137] V. Toğan and A. T. Daloğlu, “An improved genetic algorithm with initial population strategy and self-adaptive member grouping,” *Comput. Struct.*, vol. 86, no. 11-12, pp. 1204–1218, 2008.
- [138] F. G. Lobo and C. F. Lima, “A review of adaptive population sizing schemes in genetic algorithms,” in *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, (New York, NY, USA), pp. 228–234, ACM, 2005.

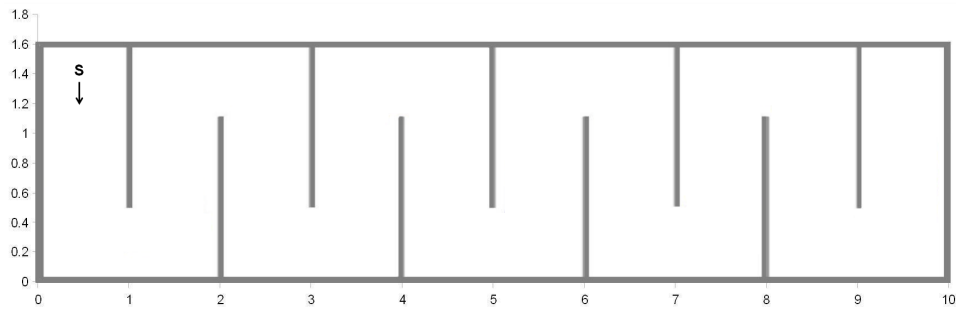
- [139] P. Kouchakpour, A. Zaknich, and T. Bräunl, “Dynamic population variation in genetic programming,” *Information Sciences*, vol. 179, no. 8, pp. 1078 – 1091, 2009.
- [140] S. E. Fahlman and C. Lebiere, “Advances in neural information processing systems 2,” ch. The Cascade-correlation Learning Architecture, pp. 524–532, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.
- [141] A. Blum, *Neural Networks in C++: An Object-oriented Framework for Building Connectionist Systems*. New York, NY, USA: John Wiley & Sons, Inc., 1992.
- [142] Y. Leung, Y. Gao, and Z. Xu, “Degree of population diversity - a perspective on premature convergence in genetic algorithms and its markov chain analysis,” *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1165–1176, 1997.
- [143] M. Rocha and J. Neves, “Preventing premature convergence to local optima in genetic algorithms via random offspring generation,” in *Multiple Approaches to Intelligent Systems* (I. Imam, Y. Kodratoff, A. El-Dessouki, and M. Ali, eds.), vol. 1611 of *Lecture Notes in Computer Science*, pp. 127–136, Springer Berlin Heidelberg, 1999.
- [144] S. Z. Ramadan, “Reducing premature convergence problem in genetic algorithm: Application on travel salesman problem,” *Computer and Information Science*, vol. 6, no. 1, pp. 47–57, 2013.
- [145] E. S. Nicoară, “Mechanisms to avoid the premature convergence of genetic algorithms,” *Petroleum–Gas University of Ploiești Bulletin, Math.–Info.–Phys. Series*, vol. 61, pp. 87–96, 2009.
- [146] R. C. Arkin, “Motor schema-based mobile robot navigation,” *International Journal of Robotics Research*, pp. 92–112, 1989.

- [147] J. Jones and D. Roth, *Robot programming: a practical guide to behavior-based robotics*. Tab Robotics, McGraw-Hill, 2004.
- [148] M. J. Mataric, *Interaction and Intelligent Behavior*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1994.
- [149] J. Johnston, *The Allure of Machinic Life: Cybernetics, Artificial Life, and the New AI*. The MIT Press, 2008.
- [150] F. Hoffmann, *Soft computing: methodologies and applications*. Advances in soft computing, Springer, 2005.
- [151] M. Mucientes, D. L. Moreno, A. Bugarín, and S. Barro, “Design of a fuzzy controller in mobile robotics using genetic algorithms,” *Appl. Soft Comput.*, vol. 7, pp. 540–546, Mar. 2007.
- [152] Y.-L. Chen, J. Cheng, C. Lin, X. Wu, Y. Ou, and Y. Xu, “Classification-based learning by particle swarm optimization for wall-following robot navigation,” *Neurocomputing*, vol. 113, no. 0, pp. 27 – 35, 2013.
- [153] C.-H. Hsu and C.-F. Juang, “Evolutionary robot wall-following control using type-2 fuzzy controller with species-de-activated continuous aco,” *Fuzzy Systems, IEEE Transactions on*, vol. 21, pp. 100–112, Feb 2013.
- [154] C.-F. Juang and C.-H. Hsu, “Reinforcement ant optimized fuzzy controller for mobile-robot wall-following control,” *Industrial Electronics, IEEE Transactions on*, vol. 56, pp. 3931–3940, Oct 2009.
- [155] L. Meeden, “Bridging the gap between robot simulations and reality with improved models of sensor noise,” in *Genetic Programming 1998: Proceedings of the Third Annual Conference* (J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, eds.), pp. 824–831, Morgan Kaufmann, 1998.

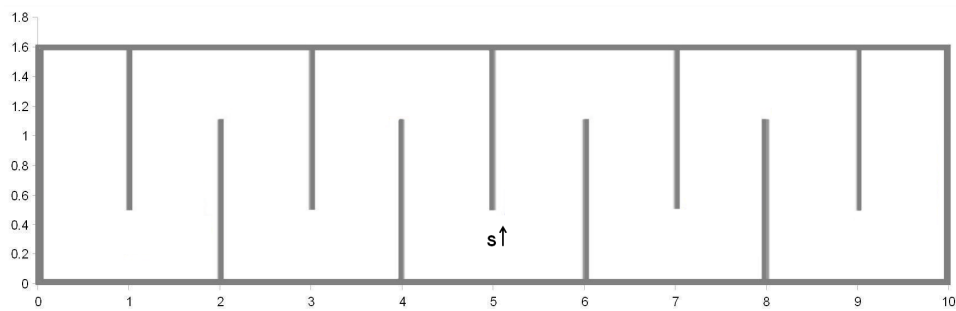
Appendices

Appendix A

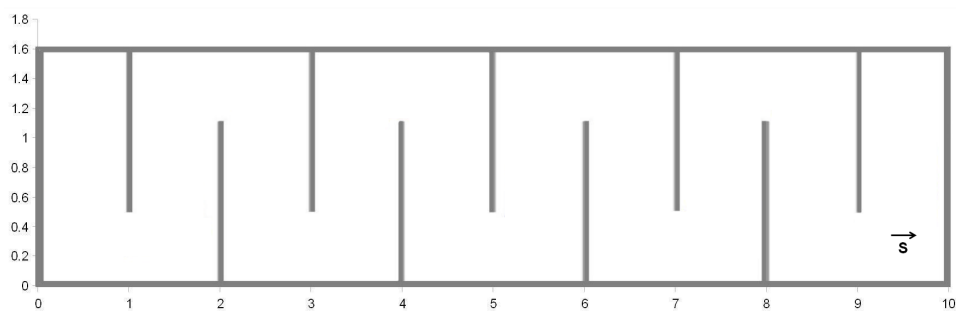
Initial Robot Position and Direction



(a) Robot at the most left of the 10-paths

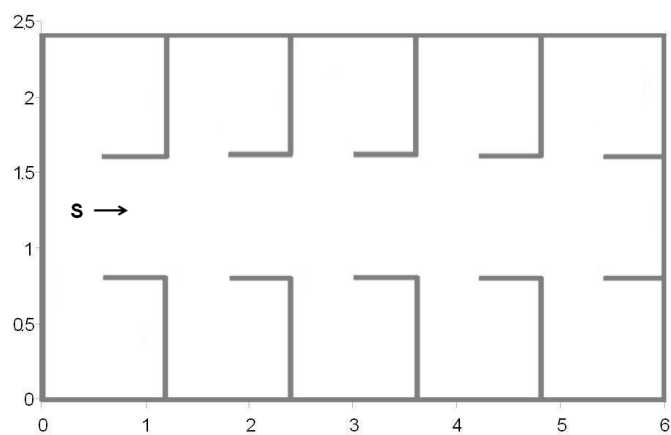


(b) Robot at the center of the 10-paths

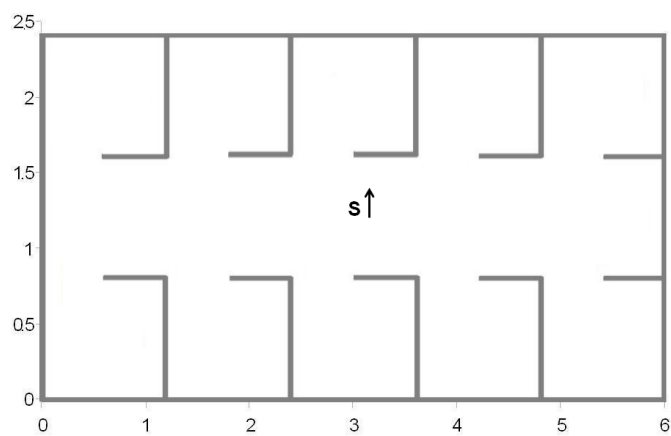


(c) Robot at the right of the 10-paths

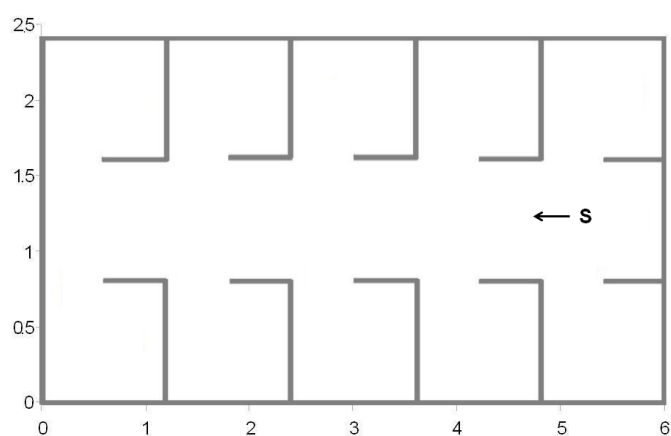
Figure A.1: Three initial positions in 10-paths environment



(a) Robot at the most left of the 10-rooms



(b) Robot at the center of the 10-rooms

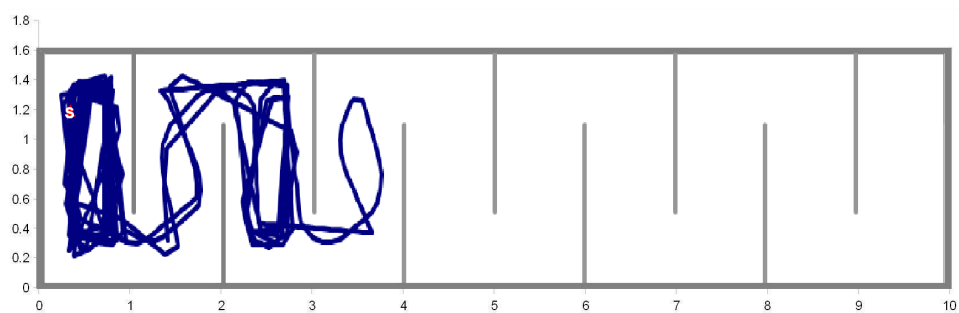


(c) Robot at the right of the 10-rooms

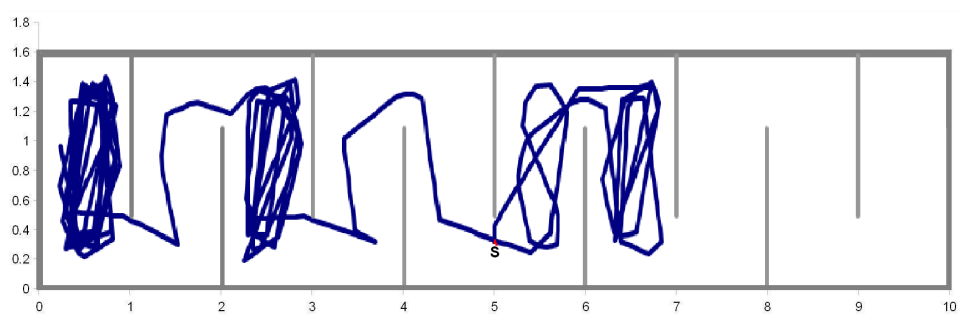
Figure A.2: Three initial positions in 10-rooms environment

Appendix B

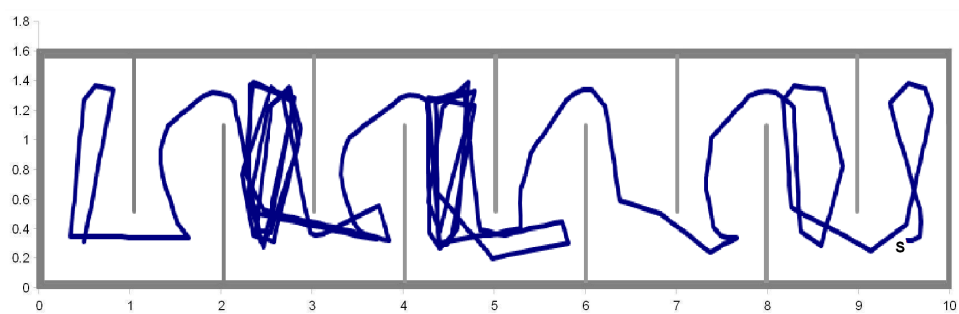
Best Path Traces for Conventional ER Approach



(a) Robot's initial position is at the most left of the 10-paths



(b) Robot's initial position is at the center of the 10-paths

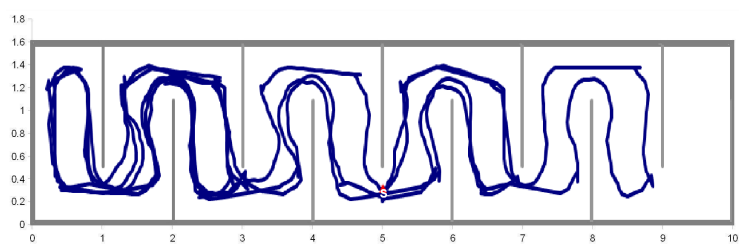


(c) Robot's initial position is at the most right of the 10-paths

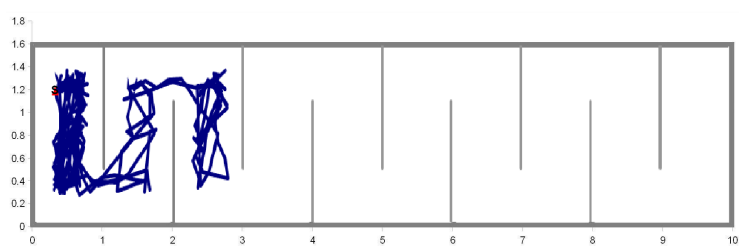
Figure B.1: Conventional ER approach in 10-paths environment

Appendix C

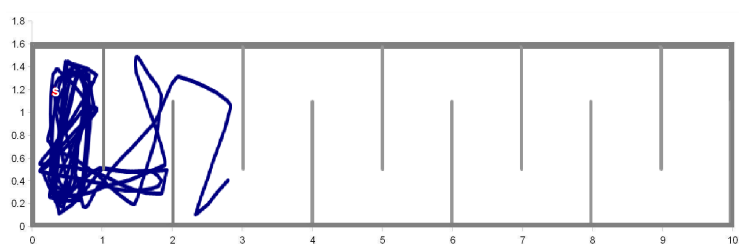
Worst Exploration Coverage



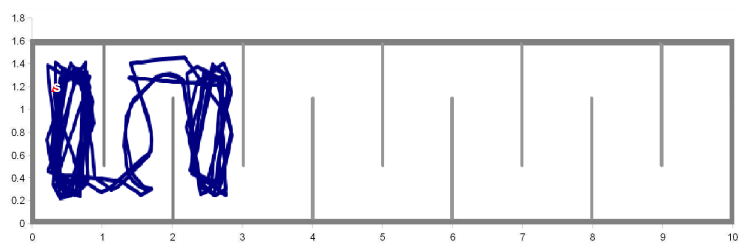
(a) GEASM



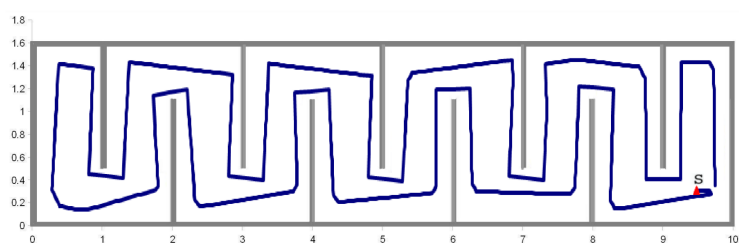
(b) PB



(c) VS



(d) Conventional ER



(e) WF

Figure C.1: Worst exploration coverage in 10-paths environment

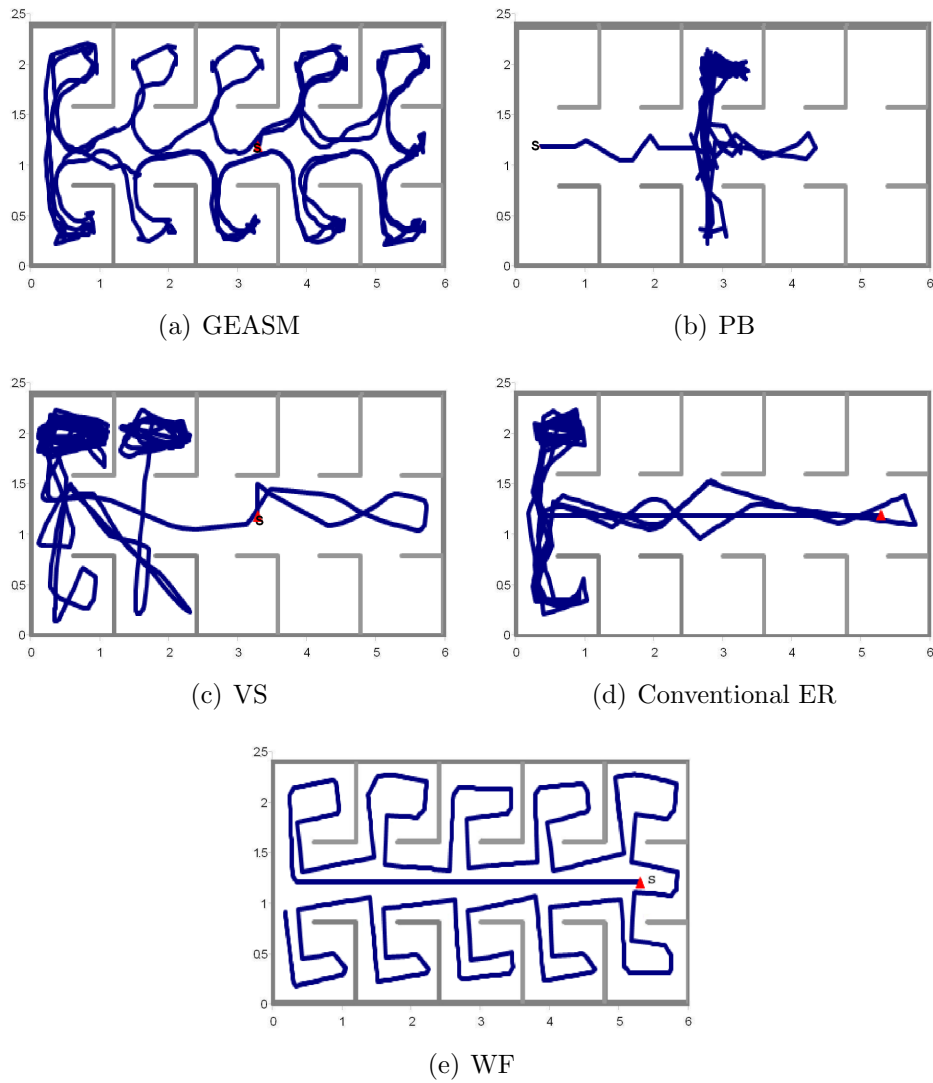
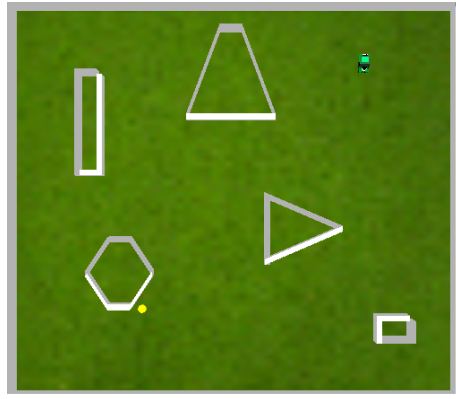


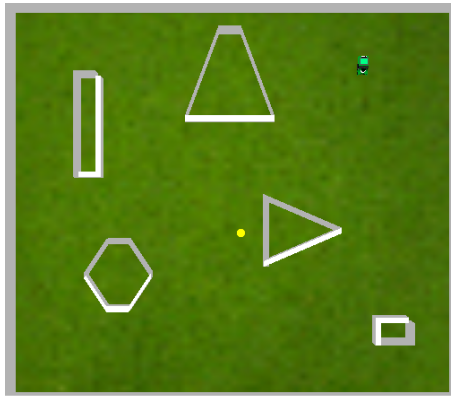
Figure C.2: Worst exploration coverage in 10-rooms environment

Appendix D

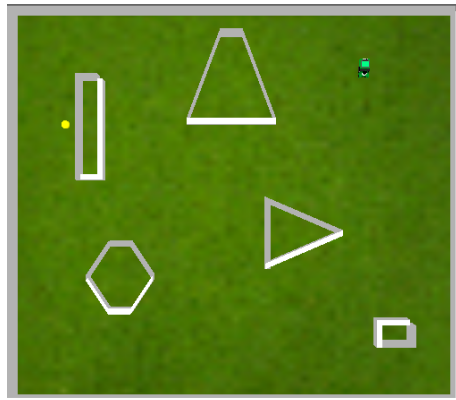
Environmental Setup for Search and Exploration Mission



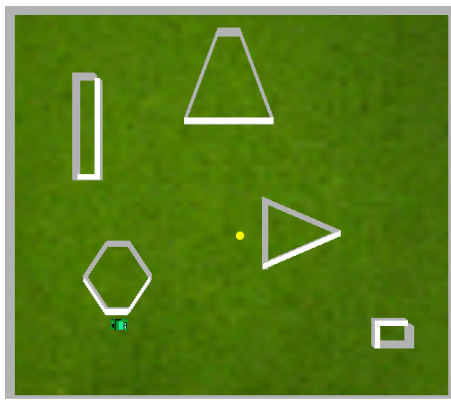
(a) Position 1



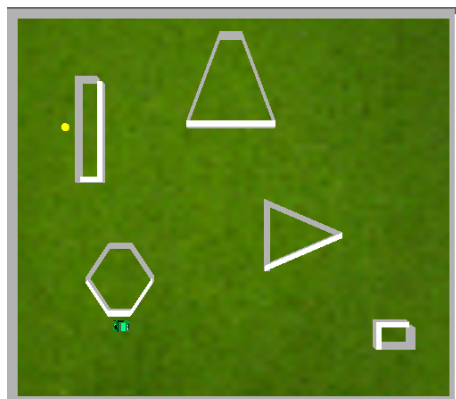
(b) Position 2



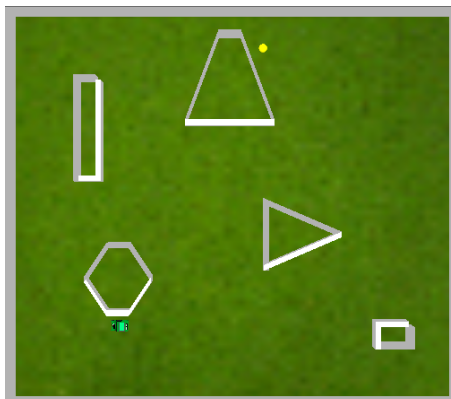
(c) Position 3



(d) Position 4



(e) Position 5



(f) Position 6

Figure D.1: Initial position: Environment 1

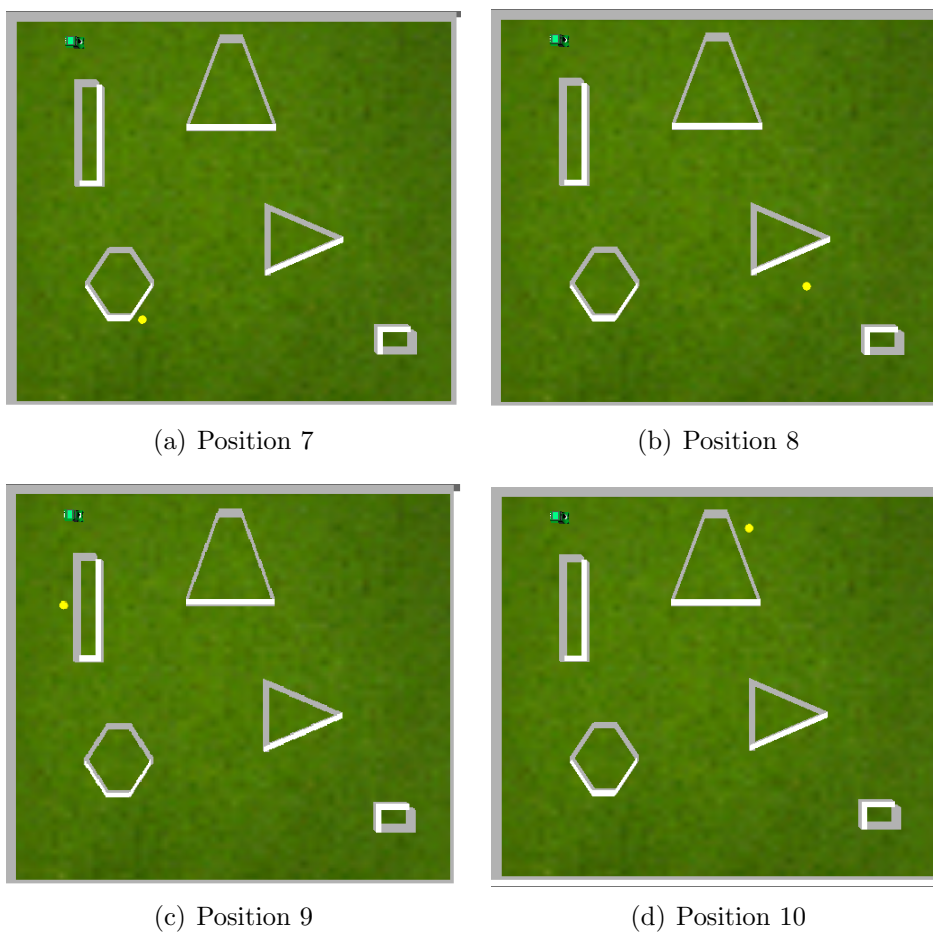


Figure D.2: Initial position: Environment 1 (... continued)

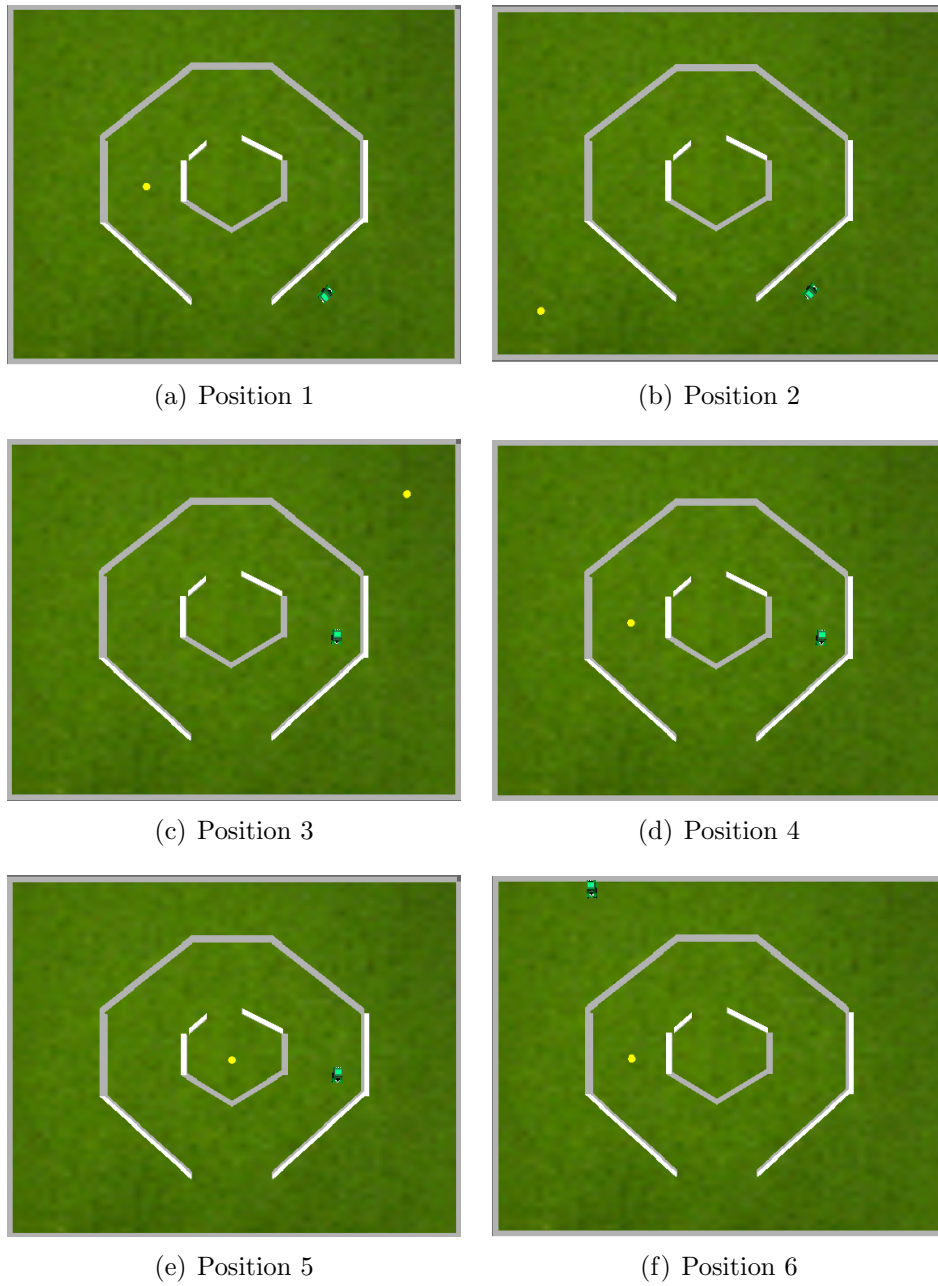


Figure D.3: Initial position: Environment 2

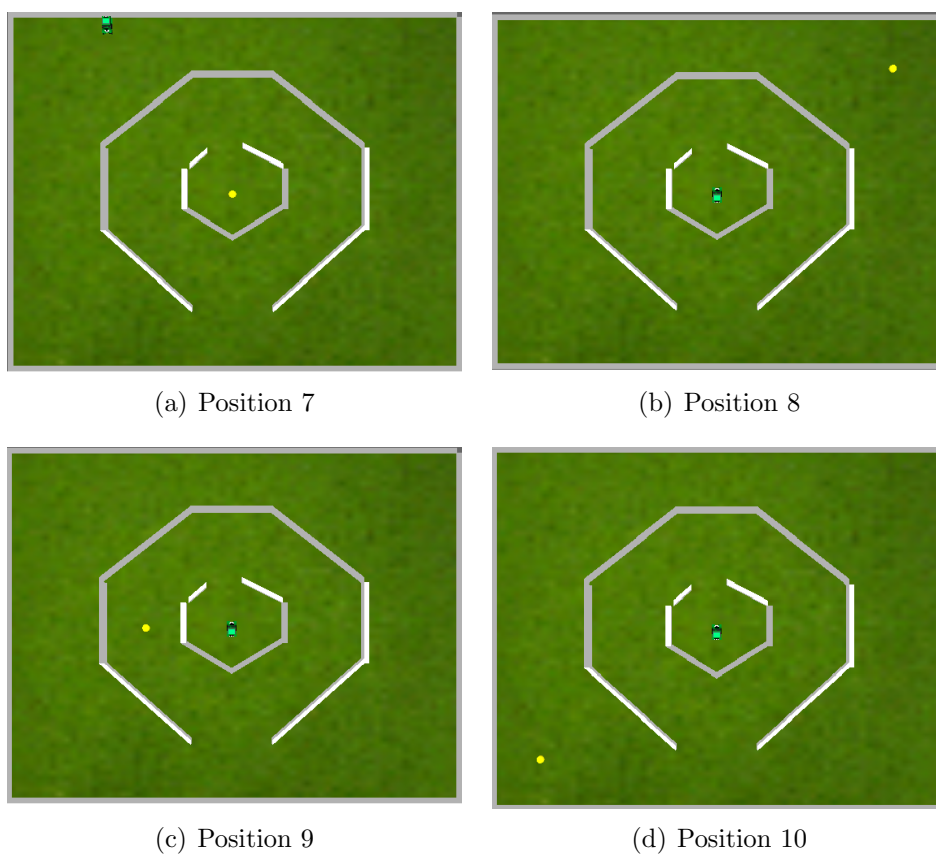


Figure D.4: Initial position: Environment 2 (... continued)

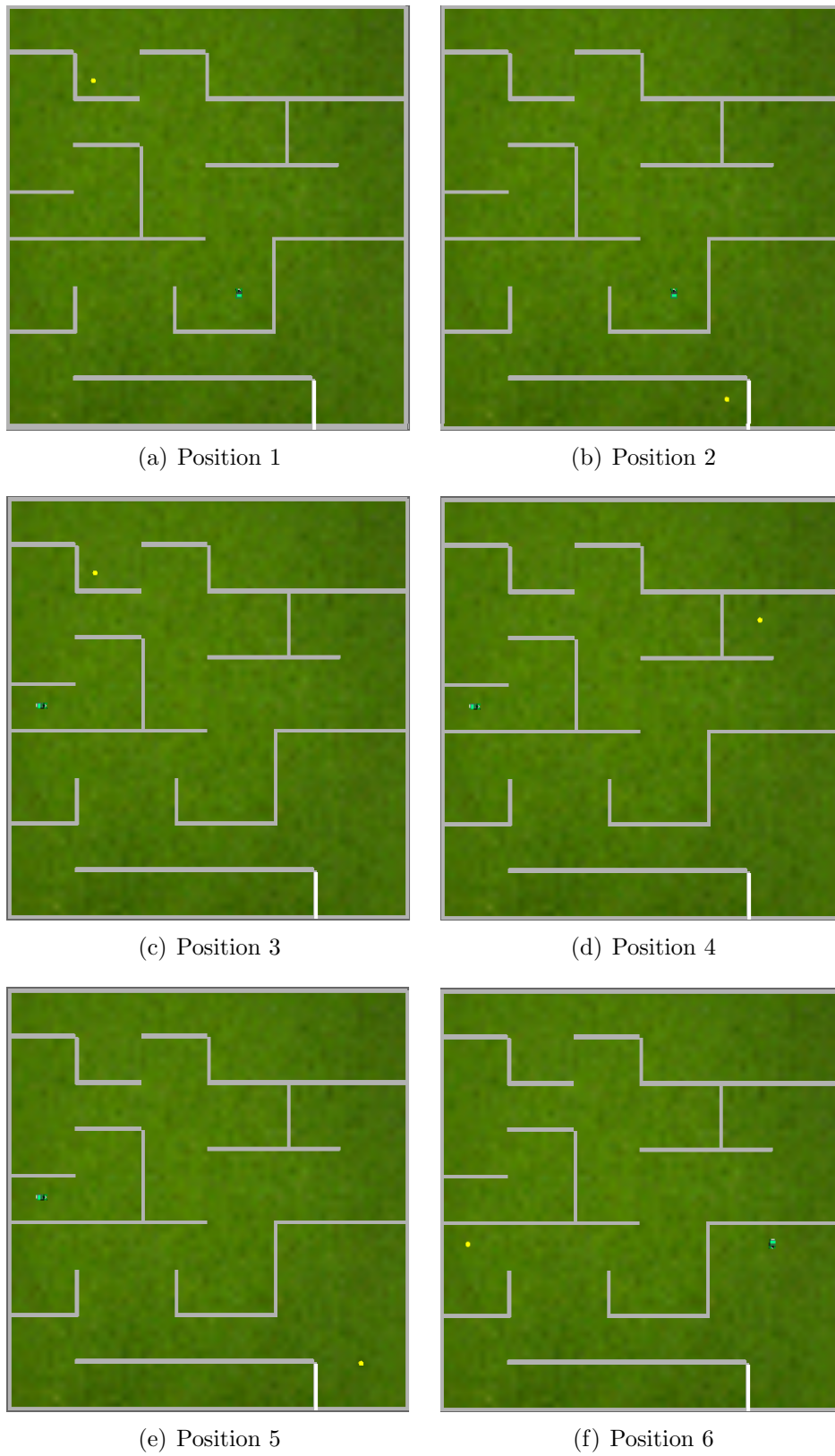


Figure D.5: Initial position: Environment 3

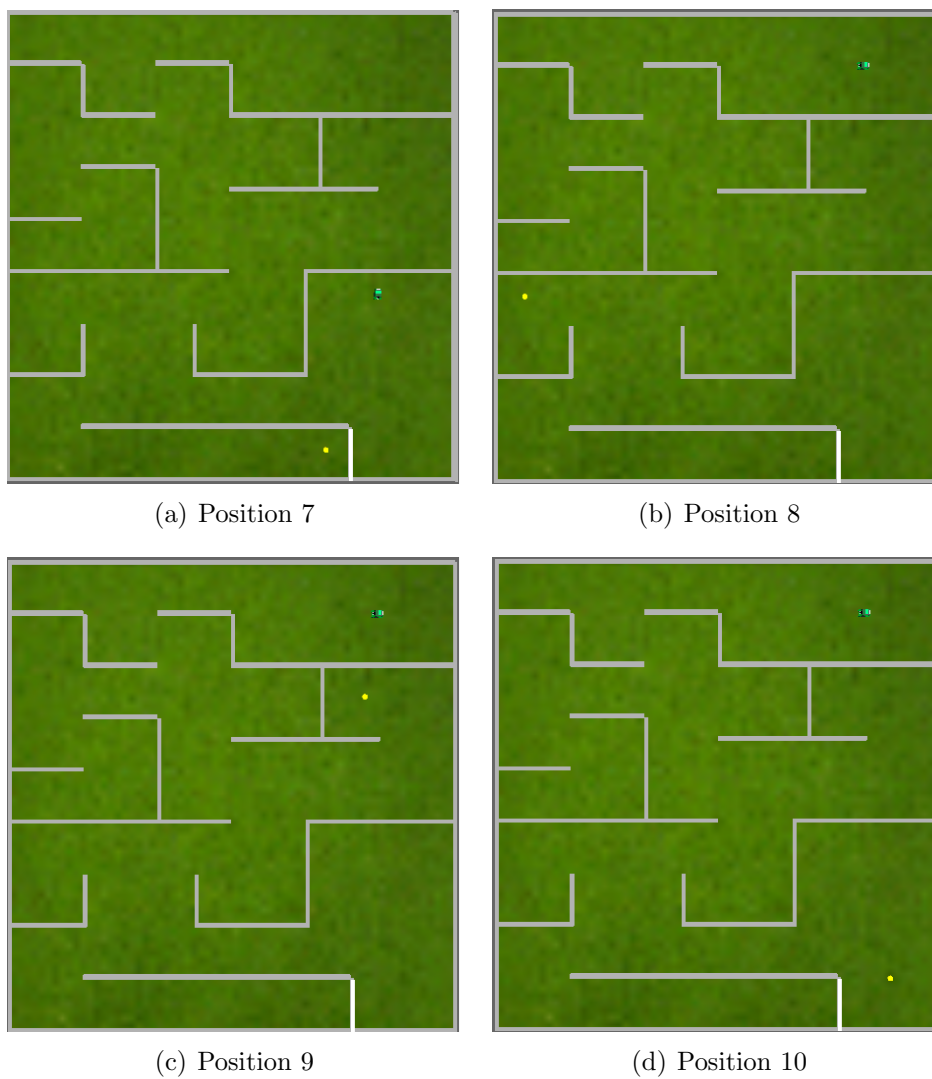
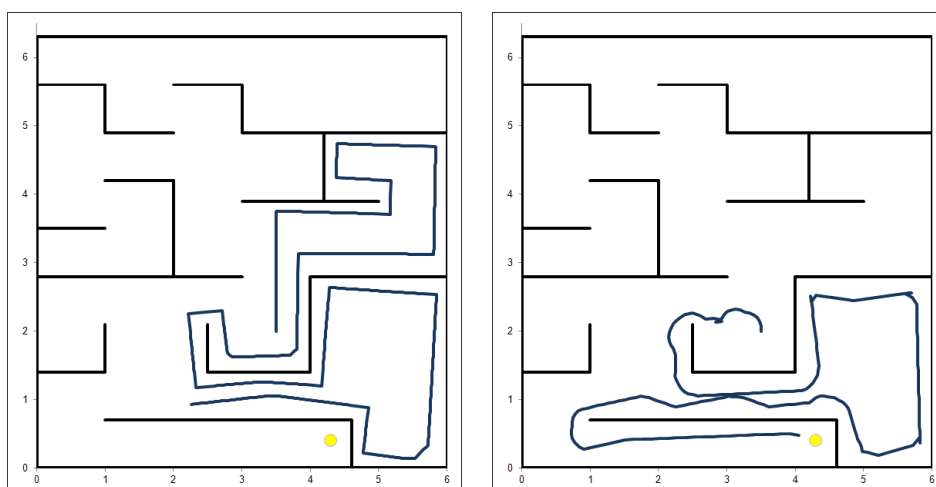


Figure D.6: Initial position: Environment 3 (... continued)

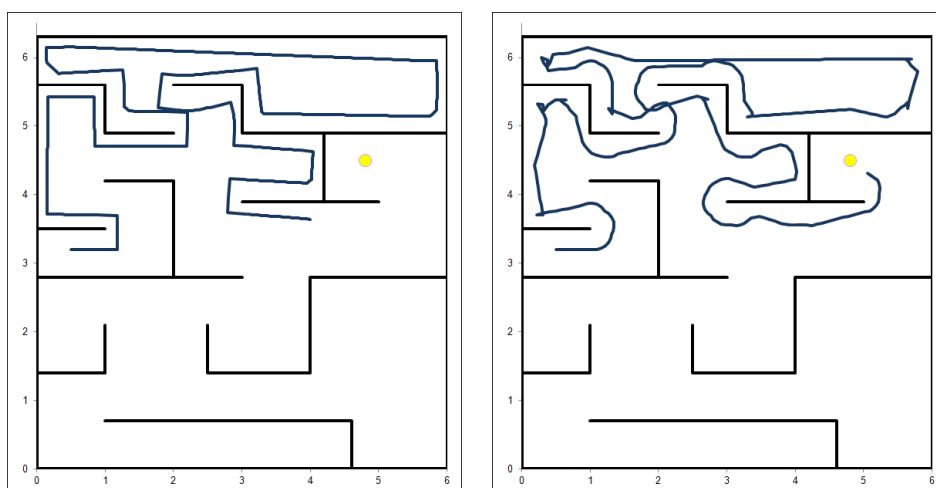
Appendix E

Wall Following Fails in Search and Exploration Mission



(a) WF unable to complete in 500TS

(b) GEASM has complete its mission in 245 TS



(c) WF unable to complete in 500TS

(d) GEASM has complete its mission in 411 TS



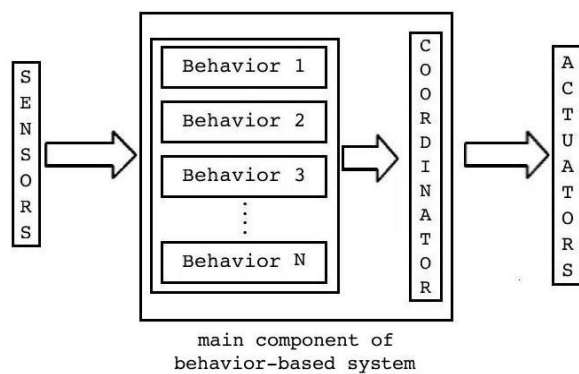
(e) WF unable to spot the target object during its exploration

(f) GEASM has complete its mission in 178 TS

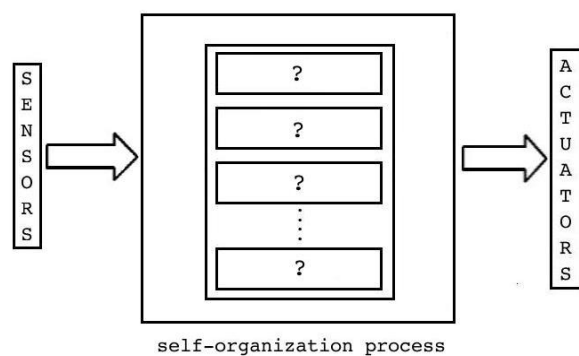
Figure E.1: Path traces of GEASM and WF in an environment where WF has failed.

Appendix F

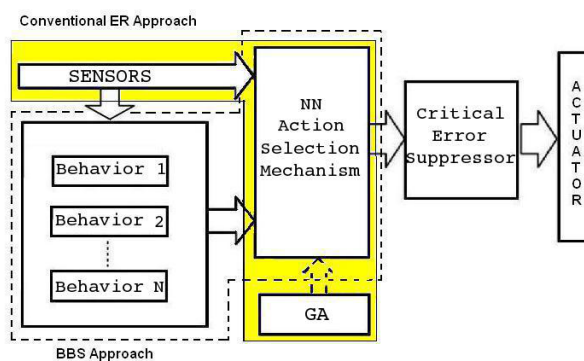
Comparison of Robot Control Architectures



(a) Behavior-based



(b) Evolutionary Robotic (ER), redrawn after Nolfi and Floreano [25]



(c) GEASM

Figure F.1: A comparison of three robot control architectures