Evolutionary neural network-based obstacle avoidance for a
planetary exploration rover

by

Yingying Ye

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial
fulfillment of the requirements for the degree of

Master of Applied Science

in

Ottawa-Carleton Institute for the Department of Mechanical and Aerospace
Engineering

Carleton University
Ottawa, Ontario

© 2015, Yingying Ye

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the Thesis

Evolutionary neural network-based obstacle avoidance for a planetary exploration
rover

Submitted by Yingying Ye
in partial fulfillment of the requirements for the degree of
Master of Applied Science

Dr. Alex Ellery, Supervisor

Department Chair

Carleton University

2015

Abstract

During space missions, a planetary exploration rover is subject to two main communication issues that are inherent to the distance and orbital difference between Earth and the target planet, the communication transmission delay and the limited transmission window. Besides, the computational resources are restricted onboard the vehicles for space missions. Based on these facts, the rover is desired to traverse on the target planet efficiently with fewer human commands and less computational needs, as well as with an autonomous decision-making scheme – allowing the rover to perform autonomous science.

The proposed autonomous path planning system presents a set of genetically evolved neural network controllers for local path planning of a mobile robot. Travelling environments are partitioned into three categories according to distribution patterns of targets and obstacles. Each evolved network is adopted to direct the rover travelling in one category of partitioned environments achieving a sequence of targets with obstacle avoidance. With a set of pre-learned networks, the rover would be adaptable to traverse in new environments of specific category. Input to the network is the range and bearing data measured from current position of the rover to surrounding obstacles and the approaching target. It outputs a turning angle ratio processed to be direction of the rover for next move. Genetic algorithm is used to obtain the evolved network by developing behavior strategies through evolutionary iterations.

Simulation results indicate that evolved neural controller can adapt to novel environments and generate satisfying path for the rover in a computationally economic manner.

Table of Contents

Abstract	iii
Table of Contents	iv
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Motivation	1
1.1.1 Current situations	1
1.1.2 Path planning overview	3
1.1.3 Classical pathfinding algorithms	4
1.1.4 Complexity of the algorithms	6
1.2 Scope	7
1.3 Organization	8
2 Background and Related Work	9
2.1 Artificial neural network	9
2.1.1 Overview	9
2.1.2 Neuron model	10
2.1.3 Network architecture	11

2.1.4	Network function	12
2.1.5	Learning algorithms	14
2.2	Genetic algorithm	14
2.2.1	Overview	14
2.2.2	Procedures of the algorithm	15
2.3	Related work	19
3	Proposed System Design	21
3.1	Map representation	21
3.2	Rover settings	22
3.3	System architecture	23
3.3.1	Overview of genetically evolved neural network	23
3.3.2	Neural network control	23
3.3.3	Evolving neural network by genetic algorithm	28
4	Simulations and Analysis	39
4.1	Unpartitioned maps	39
4.2	Partitioned maps	41
4.3	Contributions	46
5	Conclusion and Future work	48
5.1	Conclusion	48
5.2	Future work	50
	Appendices	51
A	Modifications to network output layer	51
B	Results for unpartitioned maps	51
B.1	Evolving	51
B.2	Validation	57

C	Results for partitioned maps	58
C.1	Partitioned category c_1	58
C.2	Partitioned category c_2	62
C.3	Partitioned category c_3	66
C.4	Validation	70
References		73

List of Figures

1.1	Finding shortest path using A* algorithm	5
1.2	APF net potential map	6
2.1	Example of a single neuron	10
2.2	Example structure of the neural network	11
2.3	A sigmoid function	13
2.4	A hyperbolic tangent function	13
2.5	Flow chart of genetic algorithm	18
3.1	Martian landscape <i>Courtesy NASA/JPL</i>	21
3.2	Example of terrain map	22
3.3	Laser sensors equipped around the rover	22
3.4	General steps of genetically evolving neural network	24
3.5	Input and output of the neural network	24
3.6	Rover's relating parameters at step s	25
3.7	Flow chart of neural network	26
3.8	Tree based structure of encoded chromosomes	29
3.9	Parameters used in fitness function	30
3.10	Examples of Case i	32
3.11	Examples of Case ii	32
3.12	Examples of Case iii	33

3.13	Examples of fitness bonus	34
3.14	Probability mass function	35
3.15	Cumulative distribution function	36
3.16	Overall genetic iterations	38
4.1	Performance of 100 individuals on map m_j	40
4.2	Performance of 100 individuals on map m_{j+1}	40
4.3	Generalization results on new/unpartitioned maps	41
4.4	Examples of three categories of partitioned maps	42
4.5	Performance of 100 individuals on map m_j under category c_i	43
4.6	Performance of 100 individuals on map m_{j+1} under category c_i	43
4.7	Generalization results on new/partitioned maps	44
4.8	Performance of specific expert on maps of other categories	45

List of Tables

4.1	Statistics from evolving process	43
4.2	Statistics from validation	46

Chapter 1

Introduction

1.1 Motivation

1.1.1 Current situations

The topics in wide-ranging exploration of Martian surface have received much attention and always been appealing to researchers worldwide. Increasingly coordinated missions provide a better comprehension of Mars' condition on several aspects: *i)* whether life ever arose on Mars; *ii)* the processes and history of climate on Mars; and *iii)* the evolution of the surface and interior of Mars [1]. Significant visions will be yielded concerning the search for evidence of organic carbon, taphonomy, and the habitability potential of Mars for sustained human presence.

Early in July 1997, *Sojourner*, as part of the Mars Pathfinder mission initiated by NASA, became the first landed robotic rover traversed on martian surface with a travelling distance of 100 meters; and started the close-up exploration of past and present life on Mars. Yet it was limited in the capability of autonomy. Landed in January 2004, NASA's twin Mars Exploration Rovers (MER) were assigned with the objective to search for and characterize a wide range of rocks and soils that hold clues to past water activity on Mars. They enjoyed greater autonomous capability in sensing and computing, and were designed to travel a longer distance compared to *Sojourner* [2]. *Spirit* had driven more than 7 km until it trapped in loose regolith and lost mobility in 2009 [3].

Opportunity has travelled more than 40 km and to date is in its exploration of Endeavour Crater's western rim. As part of the Mars Science Laboratory (MSL) mission, *Curiosity* landed in August 2012. It has traversed more than 8 km with the objective of assessing biological potential of the site by searching for organic and inorganic compounds in rocks and regolith fines [4].

There has been challenges in directing rover activities: First, direct teleoperation of a vehicle is possible under high-bandwidth, low-latency communications. However, the round-trip communication delay could take up to 42 minutes between Mars and Earth [5]; the availability for communication through Deep Space Network antennas is limited to several times during a Martian solar day (sol). These factors make direct teleoperation of the vehicle not practical. Second, compared to their commercial counterparts, the microprocessors qualified for space use have highly constrained computational power. Power constraints exist in the computer that supports the rover, and other factors exclusive to operating in the extreme physical environment, such as vacuum, radiation, and extreme temperatures. It limits the alternative of microprocessors for planetary rovers [6].

Rover compute element of *Curiosity* includes two identical on-board rover computers [5, 7]. They contain radiation-hardened memory for the rover to tolerate the extreme radiation. The memory of each computer includes 256 KB of EEPROM, 256 MB of DRAM, and 2 GB of solid state memory. The computers use RAD750 CPU operating at 200 MHz; whereas the average processor speed for commercial use – for example, a PC desktop – is 1.5 to 2.5 GHz.

Mission control centre must uplink a sequence of commands of vehicle operations to the rover at the beginning of each sol. These commands lay out the planned activities of the rover for a whole sol. The rover executes the commands and returns data to Earth at the end of the sol. The autonomy level of the rover determines the amount of activities accomplished each sol; which is reflected in all rover behaviors. In this work, we focus on the problem of autonomous path planning for the rover operating in unknown environment. This scenario was motivated by the rover exploring of realistic planetary surface; where rovers had to generate feasible path through rocks and access designated targets.

1.1.2 Path planning overview

A motion planning problem is to generate a continuous motion that connects a start configuration S and a target configuration T , while avoiding collision with obstacles. The motion is represented as a path in configuration space if the geometry for the robot and obstacles are illustrated in 2D or 3D space.

In path planning, a few main properties have been described based on the type of environment and algorithms engaged. It addresses if it is global or local planning (i.e., the information about the environment is known a-priori or not); static or dynamic (i.e., the environment contains moving obstacles/targets or not).

In static environment, it assumes that obstacles do not change their locations. The whole solution may be found before robot starts execution. However, replannings are normally required for partially observable or dynamic environments, and it needs more time for planning update [8].

Global path planning is map based. It requires complete knowledge of the environment, which means the robot will traverse with priori information of the environment. So the algorithm generates an appropriate path from start point to target point before robot motion starts. Robot is then released into the environment for execution. This process is considered to decide the best path for the robot travelling from start to target point. On the other hand, if path planning is local, the measuring information acquired is sensory based. It is known as real time obstacle avoidance and requires that the robot moving in environment where the information is commonly unknown a-priori. In such cases, robot needs to sense the surrounding environment before making decision for obstacle avoidance and generating path directing to the target. Algorithm adopted for planning is responding to the obstacle and the surrounding changes of the robot. While it keeps moving, the algorithm produces the path according to changes of surroundings that robot perceives during the traversal.

1.1.3 Classical pathfinding algorithms

One of the existing motion planning algorithms is grid-based search approach. It overlays a grid on configuration space, and a grid point is assumed to identify each configuration. At each grid point, the robot is able to move to adjacent grid points if the line between the points is contained within the free space – i.e., the set of configurations that avoids collision with obstacles. The set of actions is discretized, path from the start to the target is found by using search algorithms (such as A*).

A* is a heuristic search method that finds a path with least-cost from a given start node to one target node. A* builds up a tree of partial paths as it traverses the graph. The leaves of this tree are nodes remain to be explored and will be stored in a priority queue. A closed list contains the nodes that have already been evaluated. An open list contains the neighbours of nodes that have been evaluated. Evaluation is a heuristic estimate of the cost to reach a target node and the distance travelled from the start node. At each step, the best node with lowest cost value is popped out from the open list. Optimal path is achieved by each node on the optimal path keeping track of its predecessor until some node's predecessor being the start node.

Figure 1.1 depicts the expanding scheme of the A* algorithm to find the shortest path [9]. The empty circles represent nodes in the open set and the filled ones are in the closed set. Two lists update and provide the best node n at each step, optimal path has been found when n is the goal node.

Traditional grid-based methods generate paths with heading changes limiting to multiples of a base angle. Any-angle path planning approaches search for shorter paths and the heading changes of generated paths are not restricted to specific angles. Applications include a recent work at Stanford that developed a hybrid-A* algorithm which associates with the grid a continuous 3D state of the vehicle. The searching expands the grid by applying steering actions to continuous state of the grid [10].

Artificial potential field (APF) method is popular as a reactive local path planning algorithm. It considers the robot's configuration as a point in a potential field. The robot is moving under

Figure 1.1: Finding shortest path using A* algorithm

influence of artificial field of forces that produced by obstacles and target point around it. Obstacle generates repulsive field that compels the robot; while target generates attractive field that attracts the robot approaching this target [11]. At each time step, induced force of two fields is calculated by a potential function at the robot position; it moves according to the force – towards target while avoiding obstacles. Then a map of energy potential of local fields is created; as well as a gradient vector map. By following the steepest descent gradient between robot's current location and the target, the final path is achieved [12]. A net potential map of APF is illustrated, see Figure 1.2. It is the combination of attractor and repulser potential maps. Concave at the corner represents the attractor; while convex represents the obstacle known as the repulser [12].

Applications include the focus on developing new potential function to represent the attractions and repulsions [13, 14]; and the use of potential values to iteratively find the desired points in the workspace, then generates final path by connecting these waypoints [15]. Several applications use APF as global planner focuses on improving the algorithms in removing local minima which possibly happens to APF in local path planning [16]. However, global map is mostly unknown or incompletely known in pathfinding scenarios.

Figure 1.2: APF net potential map

1.1.4 Complexity of the algorithms

Due to computational resources limits, comparison of computational complexity between grid-based search algorithm is implemented. According to Sipser [17]: “the time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input”.

For grid-based search approaches like A*, time complexity is exponential given that the search space is a tree as the algorithm traverses the graph. In the worst case in which search space is unbounded, the number of nodes is expanding exponentially in the length of the shortest path d : $O(b^d)$, where b is the average number of successors at each state [18]. For a potential field algorithm that discretizes the workspace into bitmap, the time complexity of the algorithm is $O(a + b \log_b)$, where a is the number of points in the bitmap array, and b is the number of points in the augmented skeleton S extracted from the free space in bitmap [19].

Artificial neural network is known for its computational efficiency. The complexity for a neural network of n inputs, l hidden layers and h hidden neurons with activation function being hyperbolic tangent function is $2^{h(2h-1)/2} O((nl+n)^{n+2h})$ [20]. For a known network which the settings of network structure (i.e., n, l, h) are available; the overall complexity then equals $O(c)$, where c is a constant. Its omplexity is not subject to any changes while the network executes the computation.

1.2 Scope

The currently employed mobile rovers are autonomous machines; yet they are not adaptive. To date, biological evolution is the only force known to produce fully autonomous and adaptive machines. In the field of evolutionary robotics, evolutionary algorithms are exploited to improve the aspects of autonomous robot. These algorithms are considered a class of population-based metaheuristics. By the use of metaheuristics, this subfield of robotics is set apart from the mainstream of robotics research. Evolutionary algorithms are beneficial because it can improve both the architecture and parameters of the robot's control policy [21].

With computational resources limit of on-board rover, we propose the use of genetically evolved neural network controller for reactive local pathfinding given unknown environment, but implement the network operationally pre-learned. A reactive agent only plans the next step, and uses information that is immediately available such as local surroundings. Local path planning indicates that rover has no prior knowledge about the traversing map. Rover employs the already trained network as movement controller – it will operate in offline mode making appropriate decisions at each time step. Thus rover with pre-trained network is expected to intuitively react to immediate environment – i.e., in which direction to move for the next step with current surrounding. Training process provides *patterns* that relate the behavior (i.e., network response) to local environment. Given new environment, the rover would reach targets with a series of desired behaviors responding properly to its surroundings. This pathfinding problem employing neural controller has little computational cost as the complexity of neural network is comparatively low.

We assume the rover has adequate sensing and tracking capabilities as prerequisite: *i)* the visual target tracking system allows the rover to track the designated target at each move. Technical limitations that might happen to tracking system are not included, such as rover suffers sudden changes that occur to its tilt would cause target to leave the rover's field-of-view [22]; *ii)* the LiDAR laser sensors are employed on the rover to collect ranging data of distance measurements from rover's position to surrounding obstacles; *iii)* the rover is regarded as point-sized in the configuration space.

Development of proposed genetically evolved neural controller involves: 1) initializing individuals (rovers) to build up a population; 2) using neural network control to produce trajectory for each individual; 3) evaluating the ability of individual traversing efficiently according to fitness function; 4) preserving the best individual from genetic iterations.

In experimental simulations, we define terrain environment as bitmap with pixels representing objects. Environment settings refer to the arrangement and allocation of obstacles and targets. We explore the generalization capabilities of genetically evolved neural controller by adopting two strategies concerning the environment applied in neural network training. First, the network is evolved on multiple environments with little correlations in environmental settings. Then, we adjust the environment applied in evolving process by partitioning the problem space into several subspaces. Problem space refers to terrain environment with all the possible arrangement of obstacles and targets; subspaces are partitioned terrains which differ in the arrangement of target distributions. We provide three sets of partitioned terrain maps and assign three neural controllers to these terrains specifically. Each network (rover) is expected to respond organically to surrounding at each step given new environment under specific category; thus is able to generate satisfying path from start location to designated targets with obstacle avoidance.

1.3 Organization

The remainder of the thesis is organized as follows. Chapter 2 lists the background of proposed methodologies and related work in this field. Chapter 3 describes system setting and design of neural controller. Chapter 4 details the evolving and testing process of the network and analyzes proposed approach with simulation results. Chapter 5 draws conclusion and points out areas for future work.

Chapter 2

Background and Related Work

2.1 Artificial neural network

2.1.1 Overview

Artificial neural networks (ANNs) are influential brain-inspired computational models. It is based on models of human cognition by modelling a network of individual neurons to resemble the neurons in the human brain. They are consisted of neurons which represent computational elements. It use sequences of nonlinear computations to represent a nonlinear function. They are structured by consecutive layers in a graph topology. Another important element of neural network is synaptic weights which interconnect the layers. In most cases, the weights are assigned with numerical values with adaptability. It allows the network adjusting its behavior according to the situation [23]. ANN is favorable for its ability to process parameters that are complex or received from multiple sources. It is capable of handling large-scale problems because it classifies the objects into specific patterns associated with input data. Trained neural network is adaptable to new and unseen situations; and to identify new patterns. It shows good performances in situations that are too complex to achieve by traditional computational approaches.

Training process aims at identifying patterns and generalizing the network's reaction (i.e., output) to varying inputs. The network would be capable of adapting to novel and unseen patterns

with behaviors learnt. Specifically, it includes supervised learning and unsupervised learning. The former involves offline training of the network using classified input patterns. Unsupervised learning, on the other hand, uses minimum information to train the network in online mode (i.e., with no pre-classification of the input patterns) [24].

2.1.2 Neuron model

An ANN is a collection of neurons connected by synapses used for transmitting signals. Each neuron has activation energy and activation function [25]. Input neurons receive signals from the environment and output is known as a response to broadcast signals. The neuron's activation energy is the sum of the weighted outputs of neurons from the previous layer and a bias. The activation energy will be put through the activation function as in (2.1). The neuron example is presented in Figure 2.1.

$$e_p = \sum_{node=1}^n w_{pq} y_q + b_p, \quad (2.1)$$

$$y_p = K(e_p)$$

Figure 2.1: Example of a single neuron

where e_p is the activation energy of neuron p ; y_p is the output of the p^{th} neuron in the current layer, y_q is the output of the q^{th} neuron in the previous layer, b_p is the bias of the p^{th} neuron, K is

the activation function. The weight w_{pq} is conceptually the connecting strengths between neurons. A bias can be considered a value that shifts the activation function laterally.

A neural network is typically defined by three components: 1) interconnection pattern between the different layers of neurons; 2) learning process for updating the weights; 3) activation function that converts a neuron's weighted input to its output activation. Learning algorithm is used to tune the weights.

2.1.3 Network architecture

An example neural network system has three layers. The first layer has input neurons that send data through interconnections to hidden neurons of the second layer; then data are transferred to the third layer of output neurons. Figure 2.2 shows an example of the network structure. More complex systems will have more layers of neurons with increased input neurons and output neurons.

Figure 2.2: Example structure of the neural network

There are different types of neural networks. Mostly commonly used are feedforward neural network and recurrent neural network. The classification depends on how the activations of neurons interact with one another. Feedforward network is that connections between network units do

not form a directed cycle. Information in the feedforward network moves in the forward direction from input to the output. No cycles or loops will be occurred in the network. Recurrent network is that the connections between network units form a directed cycle. It creates an internal state of the network that allows it to display dynamic temporal behavior.

2.1.4 Network function

Mathematically, a neuron's network function $f(x)$ is defined as a composition of other functions $g(x)$. A widely used type of composition is the nonlinear weighted sum, where $f(x) = K(\sum_i w_i g_i(x))$, K is a predefined function known as the activation function. The activation function of a node defines the output of that node given an input or a set of inputs. It transforms the activation energy into the neuron's output. Neurons are the 'switches' that output a value of 1 when they are sufficiently activated (i.e., a threshold is reached) and a value of 0 when not activated. The most popular activation functions for neurons are the sigmoid function and the hyperbolic tangent function.

A sigmoid function has an easily calculated derivative, which is important in calculating weight updates of the network. It is very similar to input-output relationships of biological neurons. Below are the function (2.2) and a graph of sigmoid function (Figure 2.3).

$$K = \frac{1}{1 + e^{-\beta x}} \quad (2.2)$$

where β is the slope parameter.

Hyperbolic tangent function produces positive numbers between -1 and 1. It provides similar scaling to sigmoid activation function and often replaces the sigmoid function because of a greater numeric range. Below are the function (2.3) and a graph of hyperbolic tangent function (Figure 2.4).

Figure 2.3: A sigmoid function

$$K = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (2.3)$$

Figure 2.4: A hyperbolic tangent function

2.1.5 Learning algorithms

The neural network is trained through a series of example patterns of associated input and desired output values. Input to each hidden and output neuron is multiplied by its weight; the product is summed and is passed through a nonlinear transfer function to generate a result [26]. Practically it is to select a model that minimizes the cost criterion. There are numerous algorithms available for training neural networks. Training can be described as an application of statistical estimation. One of the well-adopted learning algorithms is backpropagation which computes the gradient of cost function according to network parameters and modifies the parameters in gradient-related direction [27]. It requires the designer providing direct indication of the expected values for network's output.

In conventional implementations, network topology needs to be predefined based on designer's experiences which may be biased. Evolutionary algorithms emphasize on a higher level of self-organization. It requires lower amount of supervision compared to traditional learning methods – only needs a general evaluation on the performance of an individual. In principle, there is no constraint on what elements can appear in the organizing process. Network architecture can become the objective to be organized [28]. Moreover, metaheuristic optimization strategy would result in solutions that depend on specific context of the problem rather than designer's bias [29]. Learning with evolutionary algorithm, best network topology would be evolved. Genetic algorithm is one of the evolutionary-based learning rules for neural network that replaces backpropagation.

2.2 Genetic algorithm

2.2.1 Overview

Genetic algorithms (GA) are a class of adaptive methods introduced by Holland and his colleagues. Modelled on genetics and Darwinian evolution, it involves large search spaces and is competent in solving search and optimization problems [30, 31]. The algorithm principally conducts a multi-

search that provides robust search in complicated spaces. Advantage of evolutionary computation is the possibility of adapting to varied environments in the same way that the living/evolving systems adapt to natural environment.

2.2.2 Procedures of the algorithm

The evolutionary process of GA is introduced as follows [32].

- i) Initialization: The algorithm begins by initializing a population with predefined size. Each individual in the population is a chromosome which represents one possible solution to the given problem. Encoding could be in either binary or float (real-value) format.
- ii) Fitness evaluation: It uses some predetermined criterion to evaluate the behavior of individuals. Evaluation is based on the design of fitness function. Fitness function is a particular type of objective function to summarize how close a given solution is to achieve the goals. Appropriate design and selection of fitness function will result in searching towards optimal solution. The idea is to delete the ' n ' worst solutions, and to produce ' n ' new ones from the best solutions after each genetic iteration. Each solution is 'marked' by the fitness function according to its validity. It indicates how close each solution is to meet the overall specification.
- iii) Parent selection: Selection is a crucial part of an evolutionary algorithm. It directs the algorithm towards fitter solutions. A principle of selection should favor fitter candidates over weaker candidates. Commonly used strategies are fitness-proportionate selection and rank selection.

The most popular fitness-proportionate selection is roulette wheel selection. Each individual in the population is assigned with a section of an imaginary roulette wheel. Sections are different in sizes and are proportional to individual's fitness; such that fittest candidate is allocated with the biggest slice of the wheel and weakest candidate with the smallest. The algorithm spins the wheel and selects the individual of the winning section. Wheel spins as

needed to select the full set of individuals as parents. Rank selection is similar to fitness-proportionate selection except that selection probability is proportional to relative fitness rather than absolute fitness.

There are occasions that good candidates can be lost – recombination or mutation may generate offspring that are weaker than parents. In most cases, GA will re-discover these lost improvements in subsequent generations but there is chance of ‘not getting back’. To combat this, a feature known as elitism is introduced. Elitism is to copy a small proportion of the fittest candidates unchanged to the next generation. Through elitism, candidate solutions preserved unchanged are eligible as parents in selection.

iv) Recombination/crossover: a genetic operator to differentiate the programming of chromosomes from one generation to the next. It combines two parent chromosomes to produce a new chromosome known as an offspring. The idea is that offspring may be better than both of the parents if it inherited good features from parents. It relies on a predefined recombination probability. Classical recombination/crossover methods include:

1) One point – a crossover operator that randomly selects a crossover point within a chromosome and interchanges the parent chromosomes at this point, to produce two offspring.
2) Two point – the operator randomly selects two crossover points in a chromosome and interchanges the parent chromosomes between these points to generate two new offspring.
3) Uniform – the crossover operator decides which parent contributes each of the gene values in forming the offspring according to a probability known as a mixing ratio. It allows parents to be mixed at the gene level rather than the segment level. These three methods normally apply to binary encoded chromosomes.
4) Arithmetic – the recombination operator linearly combine two parent chromosome vectors and produce two offspring according to following equations (2.4).

$$\begin{aligned}
O_1 &= a \times P_1 + (1 - a) \times P_2 \\
O_2 &= (1 - a) \times P_1 + a \times P_2
\end{aligned}
\tag{2.4}$$

where a is a random weighting factor. This method usually applies to real-coded genomes for recombination.

v) Mutation: a genetic operator to maintain genetic diversity of the population. It alters one or more gene values in a chromosome from its initial state. It relies on a predefined mutation probability which should be set low. If set too high, the search will turn into a primitive random search. Mutation allows the algorithm to avoid local minima by introducing randomness to the population and preventing individuals from becoming too similar to each other. Different mutation types are suitable for different types of chromosomes. Methods include:

1) Flip bit – it inverts the value of the chosen gene which can only be implemented on binary genes. 2) Boundary – it replaces the gene value with either the upper or lower bound of that gene. 3) Non-uniform – it increases the probability that the amount of mutation will be close to 0 as iteration continues. It keeps the population from stagnating in early stages of evolution and allows the algorithm to fine tune the solution in later stages. 4) Uniform – it replaces the value of chosen gene with a uniform random value selected between the predefined upper and lower bounds for that gene. 5) Gaussian – a mutation operator that will add a unit Gaussian distributed random value to the chosen gene. New gene value is clipped if it falls outside of predefined lower or upper bounds.

Population is updated after several iterations; fittest individual is considered the best solution. GA flowchart is shown in Figure 2.5. Loop continues until termination criterion (preset number of iterations) has been met.

Figure 2.5: Flow chart of genetic algorithm

2.3 Related work

Nolfi and Floreano addressed evolutionary robotics in [33] as: “a new approach that looks at robots as autonomous artificial organisms that develop their own skills in close interaction with the environment without human intervention.” Scholars brought up with a new class of robots that shared features with simple biological systems: simplicity, flexibility, robustness, modularity. Evolutionary robotics shares many of common features with other methods, such as behavior-based robotics and robot learning.

Most behavior-based systems are reactive; the robot uses information received from sensors to gradually correct its actions based on the changes in local environment. Behavior-based agents usually show more biological-appearing actions than their counterparts that involve intensive computations [34]. The organization of behavioral evolving system is considered the result of an adaptation procedure which a great quantity of evaluations of the interactions between the agent and the environment. Robot learning is that the agent (i.e., control system) is allowed to rely on its capability to generalize the obtained knowledge to new circumstances, after being trained using incomplete data. Typically, the control system of the agent is a neural network which learns to conduct a mapping between its input and output. Learning algorithms have imposed varied constraints on the type of structure that can be employed. With shared characteristics, artificial evolution may be considered a form of learning. Evolutionary robotics relies largely on self-organization with less constraints and is suitable to evolve the architecture of adaptive learning systems. In evolutionary robotics, neural network is widely chosen as control system for the evolving robot [33].

Prototypes of evolutionary robotics include attempts on evolving behaviors of the robot reacting to the environment. Neural networks were employed as controller in the work described. For instance, Lewis et al. have evolved a control system for a six-legged robot to move forward and backward [35]. A motor controller was trained to conduct a behavior of approaching and following light [36]. A Khepera robot was evolved to perform an obstacle avoidance task reported by several researchers [37, 38]. Evolution goes as each genome encodes a different neural network. The neural network generates activity for the robot to move while its performance was recorded.

A fitness function is used to evaluate the efficiency of each robot moving in the maze. Each robot was set to perform for a limited number of moves. During the life of each individual, there were no change on synaptic weights. Through generations, the best individual was developed with learned behaviors to move without collisions by properly processing sensory input and corresponded activation of the motors. It took a long time (approximately 40 minutes per generation with around 50 generations in total) to learn given the simple scenario that the robot implemented on.

Harvey et al. [39] evolved a system capable of approaching a visual target. Yamauchi et al. implemented an evolved neural network for solving a landmark recognition task by the use of a sonar [40]. Floreano [41] developed an evolved controller for a Khepera robot that implemented a homing navigation. Durr et al [43] has evolved a robot (with implicit genetic representations) that could complete the task of collecting reward tokens with changing locations while avoiding obstacles in a T-maze. Soltoggio et al. [44] implemented an evolved modulatory neural network for simulated bees to be adaptive in uncertain environment with varied rewards, to maximize the reward intake. They exploited evolutionary search for input features, neural network topologies and learning rules. Capi and Hideki [45] presented a genetically evolved robot using vision-based online learning – which adopted partitioned binary image data as network input – for autonomous navigation to avoid obstacle in a corridor; validation showed evolved neural controller generalized well given new environment. Wang et al. [46] evolved a behavior controller based on a spiking neural network for mobile robot avoiding obstacles.

More approaches have demonstrated the promising behavior-learning ability of neural controller given simple environment. Based on their developments, it is worth implementing the pre-learned neural controller in a more complex environment (rock distribution scenario) with an objective of achieving multiple targets with obstacle avoidance. The adaptability of a well-learned neural controller operating in multiple environment is left to be validated.

Chapter 3

Proposed System Design

3.1 Map representation

Terrain maps in simulation are represented by pixel-based bitmap images (BMP files). The rover is considered a particle represented as a point in pixel-domain. Maps imitate rock distribution on planetary surface as in Figure 3.2. The field is described as 10 m x 10 m with allocated obstacles and targets; relating to rock distribution of martian surface (Figure 3.1) as mentioned in [47]: statistical analysis denotes the rock abundance is of 6.9%. Maps are generated in reference to real environment and are simplified for simulations. Obstacles are distributed freely around targets; targets are selected by station personnel according to orbital images and are assumed to occur sequentially with short travelling distances – a line-of-sight distance of 2 - 6 m.

Figure 3.1: Martian landscape *Courtesy NASA/JPL*.

Figure 3.2: Example of terrain map

3.2 Rover settings

In this work, the simulated rover is equipped with 10 laser range sensors (LiDAR) for distance measurement of surrounding obstacles as shown in Figure 3.3. Each sensor has a field-of-view of 36 degree to fulfill the whole viewing range of 360 degree; which returns the measurement to closest obstacles in its viewing range. Rover acquires information (relative position measurement) of the *approaching* target through ob-board visual sensors at higher elevation. Realistic rover speed is 0.25 m/s.

Figure 3.3: Laser sensors equipped around the rover

3.3 System architecture

3.3.1 Overview of genetically evolved neural network

The process of genetically evolving neural network is as follows in Figure 3.4. First, a population with different chromosomes is created. Each chromosome defines the control system of the rover. In this work, the control system is a neural network whose topology and parameters would determine the behaviors of the rover. The network inputs are received from sensors of the rover, and the output is a heading for rover's next move. In a population, each individual (rover) represents a different neural network and generates specific responses interacting with local environment and finally generates a path on a terrain map. Fitness of each individual is affected by these behaviors, such as how often the rover collides with obstacles or how many targets it reaches. Initially, the rover has randomized network settings in the chromosome, thus resulting in random behaviors. Genetic operators select the fittest individuals to be recombined and mutated to produce a new generation of individuals. Evolutionary process repeats for many iterations until the behaviors of individuals have been stabilized. A sequence of maps are used during iterations for the rover traversing on to diversify the evolving environment. The fittest individual describes the best/evolved network – rover has developed stabled behavioral strategy (to generate satisfying path).

3.3.2 Neural network control

The motion control system consists of two subsystems: vision and traversing. Vision subsystem returns a series of real-time local range information received from equipped sensors. It includes the range data scaled from the rover's current location to its surrounding obstacles – only the closest ones appeared in the viewing range of each sensor will be counted; as well as the range and bearing data to the approaching target. Traversing subsystem computes the rover's heading of next move at each time step throughout the traversal. A feedforward neural network is used with sensory inputs and an output that directs rover's turning for next move, shown in Figure 3.5.

Figure 3.4: General steps of genetically evolving neural network

Figure 3.5: Input and output of the neural network

Input is described as a column vector – range data d_1 to d_T and bearing data θ_T from the rover's heading to the approaching target – (3.1). Output is a turning angle ratio r for rover's next heading.

$$input = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{10} \\ d_T \\ \theta_T \end{bmatrix} \quad (3.1)$$

Figure 3.6 illustrates the relating parameters of the rover at step s – from current location to surrounding obstacles and the approaching target.

Figure 3.6: Rover's relating parameters at step s

where, the dot in the middle indicates the rover's current location; the dots around indicate the surrounding obstacles; red dot is the approaching target; measured distances are $d_1 - d_{10}$ and d_T . Specifically, $d_1 - d_{10}$ are the ranges from the rover to local obstacles received by each laser sensor;

d_T represents the distance between rover and the next target. θ_T is bearing data indicating the target approaching angle.

Input-output interactions are in Figure 3.7 for a network with n hidden layers. Weights between each two layers are represented as a matrix – column numbers equals the number of neurons in former layer and row numbers equals the number of neurons in latter layer. Bias for neurons in each layer is listed in a column vector with dimension equals number of neurons in latter layer.

Figure 3.7: Flow chart of neural network

Given i as the number of neurons in layer a , k as the number of neurons in layer $(a + 1)$; we calculate the activation value of each neuron in the m^{th} ($m \in [1, n]$) hidden layer and list them in a column vector (A_{HL_m}).

$$W_{HL_m} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & \dots & w_{i1} \\ w_{12} & w_{22} & w_{32} & \dots & w_{i2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1k} & w_{2k} & w_{3k} & \dots & w_{ik} \end{bmatrix}, \quad B_{HL_m} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \quad (3.2)$$

$$\begin{aligned} \Theta_{HL_m} &= W_{HL_m} \times A_{HL_{m-1}} + B_{HL_m} \\ A_{HL_m} &= \tanh(\Theta_{HL_m}) \end{aligned} \quad (3.3)$$

Note: when $m = 1$ (1st hidden layer), $A_{HL_{m-1}}$ is the input vector.

Lastly, i as the number of neurons in the last hidden layer, k as the number of neuron in output layer ($k = 1$):

$$\begin{aligned} W_{out} &= \begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_i \end{bmatrix}, \\ B_{out} &= b_1 \end{aligned} \quad (3.4)$$

$$\begin{aligned} \Theta_{out} &= W_{out} \times A_{HL_n} + B_{out} \\ A_{out} &= \tanh\left(\frac{\Theta_{out}}{i}\right) \end{aligned} \quad (3.5)$$

Note: Modification made to output layer – averaging Θ_{out} , set w_i as 1, B_{out} as 0 – intends to smooth the path of each individual, according to experimental results (see Figure 1 in Appendices). A_{out} is actual output to be a turning ratio r .

Given turning ratio r as output, we calculate the angular displacement θ_{turn} (in radians), and a new heading θ_{s+1} (an absolute angle based on horizontal level) for rover's next step ($s + 1$) – with a constant velocity v and a fixed time step t .

$$\begin{aligned}\theta_{turn} &= 2\pi \cdot r \\ \theta_{s+1} &= \theta_s + \theta_{turn}\end{aligned}\tag{3.6}$$

Rover's next position can be calculated in rectangular coordinates as (x, y) :

$$\begin{aligned}x_{s+1} &= x_s + v \cdot \cos \theta_{s+1} \cdot t \\ y_{s+1} &= y_s + v \cdot \sin \theta_{s+1} \cdot t\end{aligned}\tag{3.7}$$

Rover continues traversal until one of the following conditions occur: *i)* Rover hits any of the obstacles – reflected by the real-time distance between rover and the nearest obstacle. If it is less than a pre-defined risky distance, then rover is considered in the danger region. *ii)* Energy consumption has reached a threshold, no further motions are allowed. In this work, the rover is assigned with a limited number of steps (1500) and should stop once steps are used up. *iii)* Rover has reached all the designated targets.

For processing feasibility of input data, we determine to use scalar reciprocal of input vector as actual values entering the network. Considering the scenario when there is no obstacle in the sight of any sensor, range data returned from corresponding sensors would be infinite; thus not practical for computations. Rather, by using the reciprocal, the farther the obstacles are to the sensors, the lower value it brings to the system. There would be less impact on the system input and the network becomes more sensitive interacting with closer obstacles.

3.3.3 Evolving neural network by genetic algorithm

A. Encoding

Each neural network is encoded into a chromosome. Different coding methodologies are applied to create chromosomes, such as binary coded strings and decimal coded strings. We choose

decimal coded string method because it is more flexible, needs less space and less memory than binary coding. An encoded chromosome is consisted of a sequence of network information as follows.

- i) The initial heading θ to start the traversal (based on the horizontal);
- ii) The number of hidden layers of the network;
- iii) The number of neurons of each hidden layer;
- iv) The weights for each layer;
- v) The biases for each layer.

Tree based structure is used to represent individuals in genetic algorithms. We assemble the population with 100 individuals, the initial population is generated randomly. Figure 3.8 depicts tree based structure of the variables encoded.

Figure 3.8: Tree based structure of encoded chromosomes

B. Fitness evaluation

The algorithm evaluates the performance of each individual according to feasibility of paths. In general, fitness function uses several measurements from generated path – such as displacement

from stop location to start location; displacement from stop location to the approaching target and the displacement between every two targets etc. – to evaluate the efficiency of the path. In this work, the fitness function is designed as follows.

$$fitness_a = \frac{i \times s}{t \times e} \quad (3.8)$$

$$fitness_b = b^n \quad (3.9)$$

$$fitness = fitness_a \times fitness_b \quad (3.10)$$

where, i is the displacement between every two targets according to the sequencing of targets; s is the displacement from start to stop location; t is the length of entire path; e is the displacement from stop location to the next target. b is a reward for individuals that reached any of the targets; n is the number of targets reached. In Figure 3.9, we depicts a possible path to explain the parameters appeared in fitness functions.

Figure 3.9: Parameters used in fitness function

Curvy line in blue indicates individual's actual path which length is t . Straight line in red is the cumulative sum i of displacements measured from previously reached target to the next one. In this example, it is counted from start location to the first target. If it passed the first target, displacement

from the first to the second target will be added to the sum. It is a reflection of how many target reached. Straight line in magenta indicates the displacement s from start to stop location where individual terminates travelling; which reflects the remoteness of the entire traversal. Straight line in green represents the displacement e from stop location to the next target, indicating whether individual is moving in preferable directions (i.e. towards or deviates from the target).

With assigned fitness values, well-performed individuals come to prominent as they have outperformed others in producing more feasible paths. In other words, they respond to local environment with more suited behaviors. We have demonstrated a series of examples of individuals' behaviors and presented the detailed interpretations in designing a fitness function; as well as analyzing the variables used.

First, we decompose $fitness_a$ into several parts, provide detailed examples and possibly occurred paths for analysis as below.

1. Product of i and s in (3.8) reflects the remoteness of traversal which indicates how far the individual has expanded from the start location. The more targets it achieves (larger i) and the farther it travels (larger s), the higher score it will get.

Examples in Figure 3.10 demonstrate the case for variable i (i.e., accumulated sum of displacement between sequenced targets). In the left, for Individual 1, i is the displacement from Start to Target 1. Individual 2 already passed Target 1, accumulated displacement is the sum of two sections (i.e., Start to Target 1 and Target 1 to Target 2). In the right, for Individual 1, i remains the displacement from Start to Target 1. Considering target sequencing, effective i for Individual 2 is the section of Start to Target 1, even though it came close to Target 2 at first.

2. Ratio of $(i \times s)$ over t : the actual path may consist series of redundant swirls or a number of unnecessary turnings. In such cases, individual could not be regarded favorable even though it travelled a long distance. We introduce t as denominator of $fitness_a$ and specify $(i \times s)$ to be in reverse proportion to t . This ratio is to adjust and to balance the effective distances and any unnecessary wanderings in the path. Accordingly, the result would be reasonably smaller for individual following a path long in distance yet with less targets reached.

Figure 3.10: Examples of Case i

Examples in Figure 3.11 demonstrate the above case. In the left, Individual 1 followed a path in a whirling direction. Individual 2 followed a smooth one as it directly headed to the target. In this scenario, i and s have no apparent difference, t for Individual 1 would be greater than for Individual 2. From the ratio, Individual 2 would have higher score. Similarly in the right figure, Individual 1 finished traversal with a smaller t , therefore outperformed Individual 2.

Figure 3.11: Examples of Case ii

3. Ratio of $(i \times s)$ over e : in some occasions, the number of targets achieved for different individuals are identical (i and s return similar values). Their fitnesses may be much varied concerning the direction they intend to follow thereafter. To avoid biasing the evaluation, we use variable e to differentiate if the direction of path is desired or not. Such as the case in which idling occurs and the route deviates from an expecting direction, such individuals will be degraded.

Examples in Figure 3.12 demonstrate the above situation. In the left, Individual 1 followed a path in an undesirable direction apart from the route to target. Individual 2 moved towards target and eventually stopped close-by. Thus e for Individual 1 is greater than Individual 2 with similar i and s ; the latter one receives higher fitness. In the right, Individual 1 finished traversal with smaller e , thus outperformed Individual 2.

Figure 3.12: Examples of Case iii

Second, $fitness_b$ is designed to differentiate the prominent and inconspicuous individuals by introducing exponential factors (3.9). Consequently, individuals become distinctive quickly if they are more adapted and react well to the surroundings. To expect reasonable results, b value could be set neither too high nor too low. If too high, other potential individuals may lose the opportunity to be selected for producing offsprings, and the variety of the population would be endangered. If too low, the expected individuals may be interfered by the existence of other undesirable ones, thus may be neglected in the selection process. In this work, b is manually set as 4 based on a set of ‘trial and error’ simulations.

In Figure 3.13, b is incorporated to separate individuals with similar results from $fitness_a$; and to differentiate situations such as: one individual reached none of the targets; the other reached one and stopped at a position close to passed target. In the left, no apparent differences for $fitness_a$ of the two individuals. Yet Individual 1 excelled the other as it had larger exponential for $fitness_b$. In the right, individual 1 outperformed individual 2 – individual 1 may have lower value for $fitness_a$, however it would overtake individual 2 by having a larger b than individual 2.

Figure 3.13: Examples of fitness bonus

C. Selection

The main principle of GA is survival of the fittest. Best genes survived and transferred to new generations. Individuals with prominent performances are assigned with higher probabilities. On the contrary, ill-performed individuals are assigned with lower probabilities. Two parent individuals are selected to produce offspring based on fitness values.

Instead of fitness proportional selection strategy (i.e., roulette wheel selection) commonly used in evolutionary algorithms, we adopt rank selection. Individuals are assigned with probabilities indicating their chances to be chosen as parents. It is the ranking of individuals (relative fitness) to be compared in this work, then higher ranked individuals are more likely to be chosen as parents. So we use rank selection rather than roulette wheel selection which depends on the absolute fitness of each individual. For better convergence, we adopt elite policy to preserve fitter individuals. Good genes are less likely to disappear throughout evolution, thus the population would continue improving.

Top 20% of the population are elite individuals that transferred directly to next generation. It enhances the possibility that offspring will be as good as their parents. To improve convergence, only elite individuals are considered parents according to simulations. They will be chosen to produce offsprings based on the selection probability which follows geometric distribution. The probability mass function (Figure 3.14) of a discrete random variable X is as follows.

$$Pr(X = k) = p \cdot (1 - p)^{(k-1)} \quad (3.11)$$

Figure 3.14: Probability mass function

The cumulative distribution function (Figure 3.15) of X is $F(x)$; which has a tendency towards 1, as x towards infinity.

$$\begin{aligned} F(x) &= \sum_{k=1}^n Pr(X = k) \\ &= 1 - (1 - p)^k \end{aligned} \quad (3.12)$$

$$\lim_{x \rightarrow +\infty} F(x) = 1 \quad (3.13)$$

where k indicates the ranking of individuals in the population ($k = \{1, 2, 3, \dots\}$). $Pr(X = k)$ represents probability of individual ranked k^{th} to be selected – p is set as 0.7. If too small, the difference between fitter and worse individuals becomes less obvious, fitter ones are less competent

Figure 3.15: Cumulative distribution function

to be chosen as parents. As a result, convergence would be slowed.

D. Recombination

Recombination operator takes genes extracted from parents and transferred them to offspring. Recombination probability decides the amount of genes transferred to offspring. We choose arithmetic recombination and offspring is considered a weighted arithmetic mean of parent chromosomes.

$$Child = a \times Parent_1 + (1 - a) \times Parent_2 \quad (3.14)$$

where a is the mixing ratio of recombining the two parents ($a \in [0, 1]$ – arbitrarily set as 0.5). The ratio a is 0.5, thus offspring could receive genes equally from both parents. It is more likely to inherit appropriate amount of genetic information to offsprings.

E. Mutation

Mutation operator introduces randomness to maintain diversity of the population through generations and to avoid local minima. Randomness is inserted to chromosomes in reproduction process. Probability for mutation (mutation rate) is the percentage of genes to be altered in a chromo-

some from its initial state. We moderately set it as 0.2 based on experiments in simulations; if set too large, genomes would be over-modified which increases the search effort in solution space. It is likely to prevent the population converging to satisfying solution as improved traits succeeded from previous generation may have a chance to be discarded, thus the search for solutions return to primitive. On the other hand, being too small may cause premature convergence. It would fall to local minima due to the lack of diversity. We choose to add unit random values to the chosen genes which obey Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$.

We have encoded neural network into chromosomes and initialized a population of 100 individuals. Each one directs a robot travelling on a simulated terrain map and generates a path. Performances of individuals are evaluated by fitness function and each individual is assigned with a fitness value. Reproduction procedure is as follows.

- i) New generation is consisted of four components:
 - a) the fittest individual: updated at every iteration to ensure it is the overall best;
 - b) the next top 20 individuals: the eligible candidates to be parents;
 - c) the next 69 individuals: generated by recombination and mutation operations;
 - d) the last 10 individuals: added to the population as randomness to increase diversity.
- ii) New population is established and will continue evolutionary cycles until the preset number of iterations achieved.

Figure 3.16 summarizes this evolutionary process.

Figure 3.16: Overall genetic iterations

Chapter 4

Simulations and Analysis

Individual rovers are released to traverse on terrain maps to develop behavioral strategies. During genetic iterations, individuals from the population traverse on a sequence of maps with varied distribution settings to diversify the evolving environment – with predefined number of iterations on each map. Two strategies are experimented concerning the property of applied maps throughout evolutions, as well as in the design of testing maps for validation: 1) we use maps with freely allocated targets; 2) we partition the maps into different categories; each category c_i ($i \in [1, 3]$) specifies a pattern of target distribution.

For both cases, a set of n maps is used in the iterations to get an evolved network. Individuals are initialized only once and run on each map m_j ($j \in [1, n]$) for 10 iterations then are provided with subsequent ones in the set, thus developing an evolved network takes $10n$ iterations. We validate the generalization ability of the resulted network later – to observe whether behaviors learned from training can be generalized given new maps.

Simulations are implemented in MATLAB with parallel computing to enhance execution speed.

4.1 Unpartitioned maps

We have included 15 unpartitioned maps to get an evolved network. Figure 2 shows an example of 100 individuals traversing on one map for 10 iterations (Generation 1 to 10); representing actual

paths of individuals – blue line indicates the best individual of current generation, green line as the second best. Individuals are expected to achieve targets sequentially. Through evolutionary iterations, some behaviors (network input-output pattern) are developed as individuals intend to avoid obstacles and have reached targets on current map. Population continues evolving on subsequent map (eg. Figure 3) that possesses little similarities with previous one. With the completion of 15 maps, we obtain the best individual.

Figure 4.1: Performance of 100 individuals on map m_j

Figure 4.2: Performance of 100 individuals on map m_{j+1}

Six testing maps are introduced which present little similarities in the way that targets and obstacles distribute. Figure 17 shows performance of acquired network on testing maps. Due to large scale of allocation changes in evolving process, behaviors learned on one map may be lost given

subsequent *unrelated* map; and start to build new patterns specifically for such unrelated environments. This strategy reduces the stability in developing the behaviors; behavioral system shall be built upon gradually altered environment. As a result, the generalization ability on unrelated testing maps is reduced; resulting in possible failure of the best individual finishing the task. Generalization capability is relatively low as behaviors developed through training may not be capable relating to *any* environment.

Figure 4.3: Generalization results on new/unpartitioned maps

4.2 Partitioned maps

To further develop adaptability of evolved network controller given unknown environment, we then experiment on partitioning maps applied to the network (during evolution) into three categories by specific target distribution patterns: diagonal based; left corner based and right corner based (as in Figure 4.4).

Each category includes 5 maps used in the evolution to acquire evolved network; and 6 testing maps for network validation. For each following map, adjustment of location shall be within the range – a radius of 0.5 m – for each target based on the first map in each category. Obstacles

are modified in shape and size and to be located within the range – a radius of 1.5 m – based on settings of the first map. Partitioning strategy emphasizes how relevant the different environments are, for the network to learn and validate its behaviors. One way is to fixate on target arrangement regarded as the base for adjustments of other maps.

Figure 23 shows an example of 100 individuals travelling on m_j map under category c_i for 10 iterations which represent actual paths of individuals – blue line indicates the best individual of current generation, green line as the second best. Through iterations, we notice gradually established behaviors; as individuals intend to avoid obstacles and to reach targets. Population continues evolving on subsequent map from category c_i (eg. Figure 24) that would exhibit some similarities with previous one. With the completion of evolving on 5 maps, we acquire the best individual as evolved network – an *expert* e ($e \in [1, 3]$) – expected to be adaptable on maps under category c_i . Execution on each map requires 1.5 - 2 min; it takes 7.5 - 10 min to get an evolved network.

Figure 4.4: Examples of three categories of partitioned maps

Table 4.1 reflects performance of individuals in evolving process. It includes the average elapse time required to get evolved network; the average iterations used when the first individual of the population appears to achieves all the targets; and the average iterations used for behaviors to be stabilized.

Similarly, generalization capability of resulted expert is validated on testing maps (Figure 4.7). Performances show that each network expert could generalize given new maps under specific category; unlike unpartitioned maps, behaviors developed on a sequence of relevant maps possess adaptability to new yet related environments. Each expert also performs on maps of the other two

Figure 4.5: Performance of 100 individuals on map m_j under category c_i

Figure 4.6: Performance of 100 individuals on map m_{j+1} under category c_i

Table 4.1: Statistics from evolving process

	Map category c_1	Map category c_2	Map category c_3
Avg. elapse time used on each map	1.5 min	1.8 min	2.0 min
Avg. iterations used for the 1st individual to achieve the goal	2	2	2
Avg. iterations used for behaviors become stabilized	5	6	5

categories; which indicates less satisfying results (Figure 4.8). From macroscopic perspective, reason could be that learned behaviors are preserved and individuals are evolved through slowly altered environment, according to principles observed in nature.

Figure 4.7: Generalization results on new/partitioned maps

Table 4.2 shows statics in validation including the average elapse time used by expert e_i to generate a satisfying path on maps under category c_i ; pass rate of expert e_i – whether it is able to generate a satisfying path given testing maps of category c_i ; and pass rate of expert e_i given other two types of maps.

Figure 4.8: Performance of specific expert on maps of other categories

Table 4.2: Statistics from validation

	$i = 1$	$i = 2$	$i = 3$
Avg. elapse time for expert e_i generating path on maps of category c_i	0.55 s	0.75 s	0.73 s
Pass rate of expert e_i given maps of category c_i	100%	100%	100%
Pass rate of expert e_i given maps of other categories	0%	17%	0%

4.3 Contributions

In this work, we have experimented on generalization capability of genetically evolved neural network; and provide a set of network experts to control mobile rover reactively traversing in partitioned environments with developed behavioral strategies.

Traditional pathfinding methods includes graph-based searching usually lead to complete solutions. However, it requires expensive computational efforts to thoroughly search for solutions in state space. Action-state could be limited and resulted trajectories inclines to possible sharp turnings, thus generating path with less feasibility and normally requires further smoothing. Reactive potential field method indicates good behaviors towards potentials to follow a path; potentials are applied to direct robot (by attracting or repelling according to specific functions) moving in trajectories to achieve target and avoid obstacles. However, it may depend on extra global planners or a heuristic approach to address local minima issues [12].

Proposed network is structured with input-output state patterns. Network input is convenient to acquire – sensory readings. Network outputs the heading for rover’s next step which enables it to be behaviourally flexible and to generate continuous and feasible path (free of strict turnings). During evolving, we notice the increase of network responsiveness – it has detected and identified new behavioral patterns.

We establish a partitioning strategy that emphasizes how related the environments are for generalization of a network – concerning both evolving and testing scenarios. As the ability to learn

behaviors is one of the main features of neural networks; behavioral strategy (sensor data as input and turning angle as output) of the rover responding to immediate environment has been developed through evolutionary processes. Results indicate that network evolved from partitioned environments presents satisfying performance on its generalization ability given new yet related maps. Rover with such evolved network is capable of organically responding to local environments and completes the objective of reaching targets with obstacle avoidance.

Evolved network is acquired through evolutionary iterations with the use of genetic algorithm which is a population-based approach. Population is established with each individual (a genome) encodes topology of a neural network. The idea is to evaluate behaviors of individuals and selects fitter ones to have genes transferred to next generations; over time the fittest one (network settings) could be preserved. We have designed a fitness function for evaluation, which effectively assesses the feasibility of each individual.

Chapter 5

Conclusion and Future work

5.1 Conclusion

In this work, we propose genetically evolved neural network controller for reactive local pathfinding of a rover given unknown environment; the network is operated with pre-learned behaviors. Rover employs the evolved network as movement controller. It operates in offline mode with appropriate decisions made for moving directions at each time step; intuitively reacting to surroundings. Training (evolving) is the process to develop behavior strategy that relates network response (network output) to local environment (network input). Given new environment, rover is expected to exhibit a series of behaviors (i.e., making proper reaction to surroundings at each step) travelling to targets.

We have developed the proposed neural controller through genetic evolutions using GA – which shapes the topology of evolved network (rather than hand-designed network architecture). It initializes a population that consists of individuals defined by different neural network. Each one generates specific responses interacting with local environment and generates a path on terrain map. A fitness function is designed to evaluate behaviors of individuals; genetic operators select the fittest individuals to be recombined and mutated to produce a new generation of individuals. Evolutionary process repeats for many iterations until the behaviors of individuals have been stabilized – the

rover has developed stabled behavioral strategy responding to its immediate environment. During genetic iterations, individuals traverse on a sequence of maps with varied distributions of target and obstacles to diversify the evolving environment (predefined number of iterations on each one). Fittest individual describes the best/evolved network; which is validated on testing maps.

Designed system is simulated in MATLAB, as well as the process of training and testing with paralleled structure. We explore the generalization capabilities of evolved neural controller by implementing two strategies on the environment applied in evolving process. First, the network is evolved on unpartitioned environments which are maps with little correlations concerning environmental settings (target and obstacle allocations). Throughout genetic iterations, best individual is preserved to be validated given new maps. This first set of experiments reveals that it is difficult to generalize on new/unknown maps (in validation) for the network evolved from a sequence of irrelevant maps. Possible and unpredictable failure could occur to the network when attempts relating to any given environments; due to large scale of changes on environmental settings.

We have adjusted the evolving environments for the network by partitioning the maps applied to the population (during evolution) into three categories based on specific target distribution patterns – to further develop adaptability of evolved neural controller given unknown environment. Changes on map settings follow specific criteria, thus maps under each category exhibit some relevance. Through evolutions, best network is evolved (an expert) to be adaptable in partitioned environments. Generalization capability of resulted expert has also been validated on testing maps. Results show that each neural controller could generalize on new maps; behaviors developed on partitioned maps can adapt to new yet relevant environments and generate satisfying path.

We provide three sets of partitioned terrain maps and have evolved three neural controllers specifically. Each network (rover) exhibits developed behaviors to respond organically to local environment given new maps with partition strategy. As the neural controller is operated in a pre-learned mode, it completes the pathfinding task with computationally economic solution given the relatively low complexity of neural computing.

5.2 Future work

In this work, we achieved three evolved networks for specific partitioned maps for demonstration. Each network is able to direct the rover traversing in partitioned environments with adaptability. Different partitions of maps are encouraged to be developed; thus we are able to get evolved networks for more categories of environments and enlarge the coverage of adaptable terrains. All the evolved networks are preserved in ‘expert pool’. There needs to be a supervision mechanism to select and to dispatch appropriate neural expert from the pool (which is stored in the rover system) to new terrain before traversal starts. Given a new terrain, ground personnel analyzes terrain image and specifies which group of partitioned maps it is likely (approximately) relating to; then chooses the corresponding expert for the rover autonomously completing objectives based on its behavioral strategies.

Moreover, current network output only controls the rover’s heading which travels at a constant velocity. Further development could incorporate velocity as another parameter for network output. Thus rover is able to adjust its speed according to its surroundings for effective movement control. For instance, it can move quicker when in less-crowded regions, while move slower when more obstacles are in the region.

Appendices

A Modifications to network output layer

Figure 1: *Left*: Path generated without modified output – more curves occur in the path. *Right*: Path generated with modified output – less curves occur in the path.

B Results for unpartitioned maps

B.1 Evolving

Figure 2: Performance of individuals from evolving on map m_1

Figure 3: Performance of individuals from evolving on map m_2

Figure 4: Performance of individuals from evolving on map m_3

Figure 5: Performance of individuals from evolving on map m_4

Figure 6: Performance of individuals from evolving on map m_5

Figure 7: Performance of individuals from evolving on map m_6

Figure 8: Performance of individuals from evolving on map m_7

Figure 9: Performance of individuals from evolving on map m_8

Figure 10: Performance of individuals from evolving on map m_9

Figure 11: Performance of individuals from evolving on map m_{10}

Figure 12: Performance of individuals from evolving on map m_{11}

Figure 13: Performance of individuals from evolving on map m_{12}

Figure 14: Performance of individuals from evolving on map m_{13}

Figure 15: Performance of individuals from evolving on map m_{14}

Figure 16: Performance of individuals from evolving on map m_{15}

B.2 Validation

Figure 17: Generalization results on new/unpartitioned maps

C Results for partitioned maps

C.1 Partitioned category c_1

Below are tables that record the fitness values from evolving process on five different maps, respectively.

Table 1: Fitness values from evolving process – category c_1 – map m_1

Generation	Best Fitness Value	Mean Fitness Value
1	3.4371e+03	6.0240e+01
2	3.6924e+03	9.8540e+02
3	3.7278e+03	1.5554e+03
4	3.7868e+03	1.8113e+03
5	3.7938e+03	2.0673e+03
6	3.8074e+03	2.0068e+03
7	3.8247e+03	2.0153e+03
8	3.8367e+03	1.9492e+03
9	3.8474e+03	2.1211e+03
10	3.8619e+03	2.0170e+03

Table 2: Fitness values from evolving process – category c_1 – map m_2

Generation	Best Fitness Value	Mean Fitness Value
1	2.1492e+03	2.9661e+01
2	2.9617e+03	4.1632e+02
3	3.4284e+03	1.5019e+03
4	3.6250e+03	1.6486e+03
5	3.6744e+03	1.6438e+03
6	3.7257e+03	1.6860e+03
7	3.7673e+03	1.7214e+03
8	3.7673e+03	1.7328e+03
9	3.7722e+03	1.6820e+03
10	3.7722e+03	1.6778e+03

Table 3: Fitness values from evolving process – category c_1 – map m_3

Generation	Best Fitness Value	Mean Fitness Value
1	1.3288 e+00	1.1993e+00
2	7.4257e+00	1.1833e+00
3	3.2548e+03	3.3873e+01
4	3.4209e+03	9.4906e+02
5	3.5747e+03	2.9889e+03
6	3.6937e+03	3.1553e+03
7	3.8362e+03	3.2711e+03
8	3.9625e+03	3.2499e+03
9	3.9901e+03	3.0127e+03
10	3.0601e+03	3.0295e+03

Table 4: Fitness values from evolving process – category c_1 – map m_4

Generation	Best Fitness Value	Mean Fitness Value
1	4.1411e+03	2.7694e+03
2	4.1461e+03	2.4833e+03
3	4.2488e+03	2.3431e+03
4	4.2488e+03	2.4291e+03
5	4.2489e+03	2.8991e+03
6	4.2496e+03	3.0964e+03
7	4.2496e+03	2.8908e+03
8	4.2487e+03	2.8264e+03
9	4.2495e+03	2.9075e+03
10	4.2495e+03	2.9552e+03

Table 5: Fitness values from evolving process – category c_1 – map m_5

Generation	Best Fitness Value	Mean Fitness Value
1	3.7738e+03	6.4471e+01
2	3.8897e+03	1.2162e+03
3	3.9091e+03	2.3956e+03
4	3.9454e+03	2.2572e+03
5	3.9454e+03	2.2719e+03
6	3.9454e+03	2.2842e+03
7	3.9454e+03	2.4429e+03
8	3.9508e+03	2.5892e+03
9	3.9508e+03	2.5401e+03
10	3.9513e+03	2.5935e+03

Figure 18: Performances of individuals from evolving on category c_1 – map m_1

Figure 19: Performances of individuals from evolving on category c_1 – map m_2

Figure 20: Performances of individuals from evolving on category c_1 – map m_3

Figure 21: Performances of individuals from evolving on category c_1 – map m_4

Figure 22: Performances of individuals from evolving on category c_1 – map m_5

C.2 Partitioned category c_2

Below are tables that record the fitness values from evolving process on five different maps, respectively.

Table 6: Fitness values from evolving process – category c_2 – map m_1

Generation	Best Fitness Value	Mean Fitness Value
1	4.7566e+02	1.0771e+01
2	2.1622e+03	3.8863e+01
3	3.0833e+03	7.6516e+01
4	3.1120e+03	1.7041e+02
5	3.3424e+03	1.0917e+03
6	3.3842e+03	1.6223e+03
7	3.4463e+03	1.8531e+03
8	3.4737e+03	1.8633e+03
9	3.5984e+03	1.9745e+03
10	3.6316e+03	1.8494e+03

Table 7: Fitness values from evolving process – category c_2 – map m_2

Generation	Best Fitness Value	Mean Fitness Value
1	1.9715e+02	1.0214e+01
2	1.6465e+03	4.7632e+01
3	2.8262e+03	2.3417e+02
4	2.8613e+03	1.3206e+03
5	2.9492e+03	2.2310e+03
6	3.0162e+03	2.6255e+03
7	3.0507e+03	2.6536e+03
8	3.0999e+03	2.5355e+03
9	3.1282e+03	2.5334e+03
10	3.1504e+03	2.4521e+03

Table 8: Fitness values from evolving process – category c_2 – map m_3

Generation	Best Fitness Value	Mean Fitness Value
1	3.7449e+03	2.2280e+03
2	3.7449e+03	2.6559e+03
3	3.7449e+03	2.6686e+03
4	3.7545e+03	2.5275e+03
5	3.7545e+03	2.5833e+03
6	3.7565e+03	2.5332e+03
7	3.7565e+03	2.5211e+03
8	3.7565e+03	2.5601e+03
9	3.7594e+03	2.5245e+03
10	3.7594e+03	2.6595e+03

Table 9: Fitness values from evolving process – category c_2 – map m_4

Generation	Best Fitness Value	Mean Fitness Value
1	3.6658e+03	2.4595e+03
2	3.6877e+03	2.4694e+03
3	3.6984e+03	2.6611e+03
4	3.6984e+03	2.6298e+03
5	3.6984e+03	2.4943e+03
6	3.7210e+03	2.5616e+03
7	3.7398e+03	2.5347e+03
8	3.7547e+03	2.6066e+03
9	3.7890e+03	2.5431e+03
10	3.7933e+03	2.5878e+03

Table 10: Fitness values from evolving process – category c_2 – map m_5

Generation	Best Fitness Value	Mean Fitness Value
1	4.2630e+03	2.6342e+03
2	4.2976e+03	2.6381e+03
3	4.3877e+03	2.6230e+03
4	4.3877e+03	2.6596e+03
5	4.3877e+03	2.7066e+03
6	4.4036e+03	2.6437e+03
7	4.4036e+03	2.7453e+03
8	4.4036e+03	2.7840e+03
9	4.4036e+03	2.8669e+03
10	4.4046e+03	2.7924e+03

Figure 23: Performances of individuals from evolving on category c_2 – map m_1

Figure 24: Performances of individuals from evolving on category c_2 – map m_2

Figure 25: Performances of individuals from evolving on category c_2 – map m_3

Figure 26: Performances of individuals from evolving on category c_2 – map m_4

Figure 27: Performances of individuals from evolving on category c_2 – map m_5

C.3 Partitioned category c_3

Below are tables that record the fitness values from evolving process on five different maps, respectively.

Table 11: Fitness values from evolving process – category c_3 – map m_1

Generation	Best Fitness Value	Mean Fitness Value
1	6.6188e+02	1.3190e+01
2	2.9760e+03	2.1331e+02
3	3.4082e+03	8.4718e+02
4	3.4082e+03	1.2309e+03
5	3.5543e+03	1.4917e+03
6	3.6407e+03	1.5906e+03
7	3.6407e+03	1.6073e+03
8	3.6610e+03	1.5873e+03
9	3.6752e+03	1.6123e+03
10	3.6752e+03	1.5906e+03

Table 12: Fitness values from evolving process – category c_3 – map m_2

Generation	Best Fitness Value	Mean Fitness Value
1	3.0705e+03	4.8198e+02
2	3.1893e+03	7.7763e+02
3	3.4857e+03	1.0995e+03
4	3.5549e+03	1.2386e+03
5	3.5554e+03	1.0854e+03
6	3.6529e+03	1.1037e+03
7	3.6951e+03	1.1164e+03
8	3.6951e+03	1.1094e+03
9	3.7769e+03	1.1069e+03
10	3.7769e+03	1.1015e+03

Table 13: Fitness values from evolving process – category c_3 – map m_3

Generation	Best Fitness Value	Mean Fitness Value
1	3.6033e+03	4.7248e+02
2	3.6678e+03	1.2955e+03
3	3.7551e+03	1.4270e+03
4	3.8199e+03	1.5927e+03
5	3.8415e+03	1.6939e+03
6	3.8722e+03	1.7309e+03
7	3.8771e+03	1.6669e+03
8	3.8925e+03	1.7024e+03
9	3.8925e+03	1.6893e+03
10	3.8925e+03	1.8516e+03

Table 14: Fitness values from evolving process – category c_3 – map m_4

Generation	Best Fitness Value	Mean Fitness Value
1	4.0452e+03	4.7912e+02
2	4.0452e+03	1.7289e+03
3	4.0666e+03	2.0191e+03
4	4.0783e+03	2.0398e+03
5	4.0870e+03	2.0478e+03
6	4.1101e+03	2.0418e+03
7	4.1101e+03	1.9214e+03
8	4.1101e+03	2.0371e+03
9	4.1101e+03	1.9677e+03
10	4.1101e+03	1.9310e+03

Table 15: Fitness values from evolving process – category c_3 – map m_5

Generation	Best Fitness Value	Mean Fitness Value
1	4.0750e+03	7.9965e+02
2	4.0750e+03	1.5698e+03
3	4.0822e+03	1.7343e+03
4	4.0822e+03	1.8887e+03
5	4.1165e+03	1.8751e+03
6	4.1257e+03	1.9977e+03
7	4.1374e+03	1.9859e+03
8	4.1374e+03	1.8674e+03
9	4.1374e+03	1.8986e+03
10	4.1374e+03	1.8774e+03

Figure 28: Performances of individuals from evolving on category c_3 – map m_1

Figure 29: Performances of individuals from evolving on category c_3 – map m_2

Figure 30: Performances of individuals from evolving on category c_3 – map m_3

Figure 31: Performances of individuals from evolving on category c_3 – map m_4

Figure 32: Performances of individuals from evolving on category c_3 – map m_5

C.4 Validation

Figure 33: Generalization results on new/partitioned maps of category c_1

Figure 34: Generalization results on new/partitioned maps of category c_2

Figure 35: Generalization results on new/partitioned maps of category c_3

Figure 36: Performance of expert e_1 on maps of other categories

Figure 37: Performance of expert e_2 on maps of other categories

Figure 38: Performance of expert e_3 on maps of other categories

References

- [1] B.G. Drake, S.J. Hoffman, and D.W. Beaty. Human exploration of Mars, Design Reference Architecture 5.0. In *2010 IEEE Aerospace Conference*, pages 1–24, March 2010.
- [2] M. Fleder, I.A. Nesnas, M. Pivtoraiko, A. Kelly, and R. Volpe. Autonomous rover traverse and precise arm placement on remotely designated targets. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2190–2197, May 2011.
- [3] Thibault P. Gouache, Nildeep Patel, Christopher Brunskill, and Gregory P. Scott. Soil simulant sourcing for the ExoMars rover testbed. *Planetary and Space Science*, 59(8):779–787, June 2011.
- [4] John P. Grotzinger, Joy Crisp, Ashwin R. Vasavada, and Robert C. Anderson. Mars Science Laboratory Mission and Science Investigation. *Space Sci Rev*, 170(1-4):5–56, September 2012.
- [5] M. Bajracharya, M.W. Maimone, and D. Helmick. Autonomy for Mars Rovers: Past, Present, and Future. *Computer*, 41(12):44–50, December 2008.
- [6] T.M. Howard, A Morfopoulos, J. Morrison, Y. Kuwata, C. Villalpando, L. Matthies, and M. McHenry. Enabling continuous planetary rover navigation through FPGA stereo and visual odometry. In *2012 IEEE Aerospace Conference*, pages 1–9, March 2012.
- [7] Mars Science Laboratory: Mission: Rover: Brains. Technical report, NASA/JPL, March 2009.

- [8] Rahul Kala, Anupam Shukla, and Ritu Tiwari. Dynamic Environment Robot Path Planning Using Hierarchical Evolutionary Algorithms. *Cybernetics and Systems*, 41(6):435–454, August 2010.
- [9] Subhrajit Bhattacharya. Illustration of A* search algorithm, April 2011.
- [10] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments. *The International Journal of Robotics Research*, 29(5):485–501, April 2010.
- [11] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, October 1992.
- [12] Cameron Frazier. *Re-Active Vector Equilibrium: A Novel Method of Autonomous Vehicle Navigation Using Artificial Potential Fields*. PhD thesis, University of Ottawa, January 2015.
- [13] S.S. Ge and Y.J. Cui. New potential functions for mobile robot path planning. *IEEE Transactions on Robotics and Automation*, 16(5):615–620, October 2000.
- [14] S. S. Ge and Y. J. Cui. Dynamic Motion Planning for Mobile Robots Using Potential Field Method. *Autonomous Robots*, 13(3):207–222, November 2002.
- [15] Hossein Adeli, M.H.N. Tabrizi, and Alborz Mazloomian. Path Planning for Mobile Robots using Iterative Artificial Potential Field Method. *IJCSI*, 8(4):28, July 2011.
- [16] F. Arambula Coso and M. A. Padilla Castaeda. Autonomous robot navigation using adaptive potential fields. *Mathematical and Computer Modelling*, 40(910):1141–1156, November 2004.
- [17] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, June 2012.
- [18] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

- [19] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):224–241, March 1992.
- [20] M. Bianchini and F. Scarselli. On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1553–1565, August 2014.
- [21] Josh C. Bongard. Evolutionary Robotics. *Commun. ACM*, 56(8):74–83, August 2013.
- [22] Won S. Kim, Issa A. Nesnas, Max Bajracharya, Richard Madison, and Adnan I. Ansar. Targeted driving using visual tracking on Mars: From research to flight. *J. Field Robotics*, 26:243–263, 2009.
- [23] Simon S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 2009.
- [24] S. B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.
- [25] Simon S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.
- [26] A. T. C. Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1995.
- [27] Randall J. Erb. Introduction to Backpropagation Neural Network Computation. *Pharm Res*, 10(2):165–170, February 1993.
- [28] Jrme Kodjabachian, Christophe Corne, and Jean-arcady Meyer. *Evolution of a Robust Obstacle-Avoidance Behavior in Khepera: A Comparison of Incremental and Direct Strategies*. 1998.

- [29] G. C. H. E. de Croon, L. M. O'Connor, C. Nicol, and D. Izzo. Evolutionary robotics approach to odor source localization. *Neurocomputing*, 121:481–497, December 2013.
- [30] K.H. Sedighi, K. Ashenayi, T.W. Manikas, R.L. Wainwright, and Heng-Ming Tai. Autonomous local path planning for a mobile robot using a genetic algorithm. In *Congress on Evolutionary Computation, 2004. CEC2004*, volume 2, pages 1338–1345 Vol.2, June 2004.
- [31] Ching-Chih Tsai, Hsu-Chih Huang, and Cheng-Kai Chan. Parallel Elite Genetic Algorithm and Its Application to Global Path Planning for Autonomous Robot Navigation. *IEEE Transactions on Industrial Electronics*, 58(10):4813–4821, October 2011.
- [32] R.J. Terrile, C. Adami, H. Aghazarian, and S.N. Chau. Evolutionary computation technologies for space systems. In *2005 IEEE Aerospace Conference*, pages 4284–4295, March 2005.
- [33] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press, 2000.
- [34] Rodney Brooks. Intelligence Without Representation. *Artificial Intelligence*, 47:139–159, 1991.
- [35] M.A. Lewis, A.H. Fagg, and A. Solidum. Genetic programming approach to the construction of a neural network for control of a walking robot. In , *1992 IEEE International Conference on Robotics and Automation, 1992. Proceedings*, pages 2618–2623 vol.3, May 1992.
- [36] M. Dorigo M Colombetti. Learning to Control an Autonomous Robot by Distributed Genetic Algorithms. pages 305–312, 1992.
- [37] Orazio Miglino, Henrik Hautop Lund, and Stefano Nolfi. Evolving Mobile Robots in Simulated and Real Environments. *Artificial Life*, 2(4):417–434, July 1995.
- [38] Stefano Nolfi, Stefano Nolfi, Dario Floreano, Dario Floreano, Orazio Miglino, Orazio Miglino, Francesco Mondada, and Francesco Mondada. *How To Evolve Autonomous Robots: Different Approaches In Evolutionary Robotics*. 1994.

- [39] Inman Harvey, Phil Husbands, and Dave Cliff. *Seeing the Light: Artificial Evolution, Real Vision*. 1994.
- [40] Randall Beer Brian Yamauchi. Integrating Reactive, Sequential, and Learning Behavior Using Dynamical Neural Networks. *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behaviour*, pages 382–391, 1994.
- [41] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(3):396–407, June 1996.
- [42] P. Durr, C. Mattiussi, and D. Floreano. Genetic Representation and Evolvability of Modular Neural Controllers. *IEEE Computational Intelligence Magazine*, 5(3):10–19, August 2010.
- [43] A. Soltoggio, P. Durr, C. Mattiussi, and D. Floreano. Evolving neuromodulatory topologies for reinforcement learning-like problems. In *IEEE Congress on Evolutionary Computation, 2007. CEC 2007*, pages 2471–2478, September 2007.
- [44] Toda Hideki Capi Genci. Evolution of Neural Controllers for Robot Navigation in Human Environments. *Journal of Computer Science*, 2010.
- [45] Zeng-Guang Hou Xiuqing Wang. A behavior controller based on spiking neural networks for mobile robots. *Neurocomputing*, 71(4):655–666, 2008.
- [46] M. P. Golombek, A. F. C. Haldemann, N. K. Forsberg-Taylor, E. N. DiMaggio, R. D. Schroeder, B. M. Jakosky, M. T. Mellon, and J. R. Matijevic. Rock size-frequency distributions on Mars and implications for Mars Exploration Rover landing safety and operations. *J.-Geophys.-Res.*, 108(E12):8086, December 2003.