THE UNIVERSITY of ADELAIDE

Evolution of High Level Motion Control for Autonomous Ground Vehicles

Mohd Faisal Ibrahim

A Thesis submitted for the degree of Doctor of Philosophy

School of Computer Science

The University of Adelaide

2015

Contents

List of Figures

List of Tables

Abstract

Autonomous robotic exploration is the task of building models of an environment. This task requires robots to rapidly plan, re-plan and execute their motion trajectories using sensory data that is provisional, uncertain and noisy. To navigate successfully under these conditions, robots require carefully designed motion controller software to guide the robot safely, quickly, reliably and efficiently to intermediate exploration objectives. Conventionally, the basic design of a motion controller is derived from first principles using simplified models of motion control and then refined by hand in response to observed performance. While this approach works in simpler applications, it becomes more challenging and less effective as applications become more complex and the number of variables to consider increases. Moreover, changes in robot configuration and environment can entail costly redesign of the controller. As such, we argue that this manual approach will become increasingly impractical as our exploration tasks become more ambitious. In this thesis, we address the development of motion control using techniques from Evolutionary Computation (EC). Our approach is to view the motion control design as a search problem, that can be subject to automation. In this work we present a novel framework for evolving the core component of motion control based on a form of EC called Grammatical Evolution (GE). GE systematically refines populations of potential programatic solutions for a given problem, until an effective solution is found. In our approach, we use GE to search automatically for the best motion control for a given set of exploration tasks. GE allows the user to constrain the search space for programs using Backus-Naur Form (BNF) grammar specifications. We use these grammars to define the search space for controllers for each exploration application. We conducted four experiments to evaluate our proposed approach. Each experiment demonstrates the framework in different exploration configurations and different requirements. All of our experiments evolved controller code for unmanned ground vehicles (UGV's). Our first experiment evolved numerical parameters for the control of small teams of UGV's. Our second experiment evolved control for a single UGV to optimise exploration performance and energy consumption. Our third experiment evolved both the structure and parameters of the core control function. Our fourth experiment evolved the input factor selection and numerical constants for well-established navigation approach in progressively

more realistic situations - culminating in deployment on real platforms. In each of our experiments we found that the automated search approach outperformed carefully designed handwritten control. Moreover the structure of the evolved equation helped to reveal the nature of the trade-offs inherent in the exploration task and what factors appear to be most relevant to informing effective control.

Thesis Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree. I give consent to this copy of my thesis when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

The author acknowledges that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

MOHD FAISAL IBRAHIM
Date: 8^{th} DECEMBER 2015

Acknowledgement

First of all, I would like to express my sincere gratitude and appreciation to my supervisors Prof. Zbigniew Michalewicz and Dr. Bradley Alexander for your countless guidance and supports during my Ph.D study. Both of you are great teachers teaching me a lot about research. I could not think about completing my study without their encouragements, supports and motivation.

Next, my appreciation to the members of Optlog group and my lab's colleagues for your generous thoughts and useful discussions that spark ideas most of the time.

My sincere thanks also to all academic and supporting staffs at the School of Computer Science, International Student Centre (ISC) and Adelaide Graduate Centre (AGC) at The University of Adelaide for helping me on administration stuffs.

I would also like to thank the Ministry of Education Malaysia and Universiti Kebangsaan Malaysia (UKM) for the full sponsorship during my Ph.D study. Not to forget to all my friends and relatives in Adelaide and Malaysia, many thanks for your warm supports and encouragements.

A special thank to my family for their support and sacrifices. To my father Hj. Ibrahim, my mother Hjh. Jahani, my father-in-law Dr. Hj. Baseri Huddin, my mother-in-law Hjh. Zaharah and all my siblings, words cannot describe my gratitude for your prayers and supports. To my beloved wife Aqilah who is always at my side, special appreciation for your endless sacrifices and understandings along the moments of pursuing our study. To my daughter Aliya, you always cheer up my life. Love you all.

Last but not least, I would also like to thank the members of my examination committee. Their comments have improved the final presentation of the thesis significantly.

Abbreviations

APF	Artificial Potential Field
BNF	Backus-Naur Form
CFG	Context-Free Grammar
CMA-ES	Covariance Matrix Adaptation Evolutionary Strategy
DTC	Decision-Theoretic based Motion Control
DWA	Dynamic Window Approach
EC	Evolutionary Computation
ER	Evolutionary Robotics
ES	Evolutionary Strategy
FLC	Fuzzy Logic Controller
GA	Genetic Algorithm
GE	Grammatical Evolution
GP	Genetic Programming
GS	Ground Station
GPS	Global Positioning System
HRL	Hierarchical Reinforcement Learning
IMU	Inertia Measurement Unit
LE	Layered Evolution
LIDAR	Light Detection and Ranging Sensor
MPC	Model Predictive Control
ND	Nearness Diagram
NN	Neural Network
POMDM	Partially Observable Markov Decision Model
PSDP	Policy Search Dynamic Programming
RL	Reinforcement Learning
SGHC	Simple Genetic Hill Climbing
SLAM	Simultaneous Localisation and Mapping
SVM	Support Vector Machine
UGV	Unmanned Ground Vehicle
VFH	Vector Field Histogram

1 Introduction

Autonomous robotic exploration is a complex problem. The task requires robots to rapidly plan, re-plan and execute their motion trajectories through a progression of states. In a robotic exploration task, motion control is a fundamental component to accomplishing the above processes efficiently and effectively. Despite strong advances in various aspects of robot-related technologies, the challenge of developing robust motion control for autonomous exploration tasks is still an open problem. In fact, tele-operated or semi-autonomous motion control that requires human intervention during operations is still preferred to fully autonomous motion control [16]. The cause of this lack of automation lies the complexity of the task of developing autonomous exploration and motion control [76]. Such control requires a high level of prior knowledge and substantial development time. One issue that yields such fact is due to the complexity of the development of a motion control algorithm [76], in which, high level of prior knowledge, expertise and development time are mostly required.

In this thesis, we present a method based on Evolutionary Computation (EC) to develop a motion control algorithm for robotic exploration tasks. Our method is designed to search for the well-refined form of motion control using the Darwinian principle of evolution whilst minimising the need of domain knowledge and human development time. One key contribution of our method is the flexibility of the evolutionary process that allows roboticists to select elements to be optimised in a motion controller. We exhibit various configurations of our method to adapt the evolution of motion control at various levels i.e. to tune constant parameters, to design a control structure or to select input variables used by a motion controller. Our experimental

results show that our method improves exploration performance significantly in real and simulated settings. The rest of this chapter presents our motivation, research goals, method overview and the outline of this thesis.

1.1 Overview

An autonomous robotic exploration task can be defined as the act of a single robot or a team of robots, moving through an unknown, unstructured and possibly hazardous environment, while building a map that can be used for subsequent navigation [130]. This generic task is applicable to a wide range of real-world robotic applications, such as surveillance [111], search and rescue [15], volcanic exploration [96], planetary exploration [67], wildfire detection [105] and home vacuum cleaning [57]. Initially, a robot has no idea of what its environment looks like. The robot performs a sequence of *sense-evaluate-act* steps to partially explore the environment. Firstly, the robot *senses* its surrounding using its sensors to acquire raw provisional environmental data. Such raw data are *evaluated* and transformed into useful information. A provisional map is built using this information. The robot also uses the same information to identify a location that needs to be explored and then selects the best trajectory to follow towards that identified location. Finally, the robot *acts* upon its planning by sending commands to its actuators to perform appropriate movements. The above processes are iterated until the unknown environment is fully explored by the robot or the program is terminated.

Motion control is an important element of the exploration strategy. It is responsible for executing major parts of the *evaluate* and *act* processes. Motion control can be defined as the strategy where the robot uses available information to make a decision about how to act, and then implements that action. Accordingly, the decision quality of a motion controller is a key determinant of its exploration performance. The precise design of a motion controller will depend heavily on the priorities that govern the exploration task. For example, in an exploration task where completion time is critical, one may need a motion controller that can drive robots at a higher average speed to

complete the task quickly. On another exploration task where robot-safety is more crucial, the motion control of a robot that chooses a safer trajectory rather than a faster trajectory is preferred.

As exploration tasks become more complex, so does motion control. Empirical study has shown that there appears to be no optimal exploration strategy for all application domains [82]. As such, motion control of an exploration task is unlikely to be a generic framework that applies to all types of robots and software. Instead, the design of motion control must be shaped by the context of its specific exploration domain. The effort of refining a motion controller to an exploration domain is a non-trivial task that, when performed by humans, requires substantial domain knowledge, expertise and development time.

1.2 Motivation

The line of research addressed in this thesis was motivated by the problem of developing a robust robotic exploration system. This requires the problem to be seen from various perspectives as follows:

Different robotic exploration tasks require different control settings - A robot control system must be designed such that it adapts to the intended exploration task. If, over time, the task requirements change then the control system may require refinement. Such changes could be in terms of the robot's environment, hardware or software. Such changes might include: the number of robots involved, the robots' body design, computational resources, sensory devices and signals processing algorithms. These variations affect the required behaviour of motion control. Conventionally, roboticists handle this situation. A common approach to this manual re-design task is to apply domain knowledge intuitively. This process involves understanding the new navigational structure, applying available knowledge about the new exploration requirements, designing control preferences by hand, testing and refining the modified motion control. This approach normally involves repeated trial-and-error and finding a good solution by this means can be

time consuming. This classical approach is prone to some fundamental design errors that scaled with the complexity of the robot system. In particular, there are three main issues that could arise as described in [54]: (i) unclear decomposition of controller's decision-making structure, (ii) conflict between direct interaction within robot system and indirect interaction via environment, and (iii) the number of interactions between sub-systems increases, thus robot system becomes more complex. As a consequence, the shortcomings of heuristic approach can lead to less-than-ideal outcomes [13].

Rich sensing capabilities, but poor knowledge interpretation - Modern robots use a number of sensors to extract information from their environments. The use of multiple sensors is suggested to improve the performance of an exploration task by providing more reliable data through redundancy. One key advantage over single sensor usage is the convergence of sensor information that can reduce sensor uncertainty [118]. However, the interpretation of the incoming sensory data into useful knowledge is an open problem. This vital data transformation is an essential process in order to design a robust motion controller. For example, a typical autonomous ground vehicle robot might be equipped with an odometry sensor, a proximity sensor, a global positioning system (GPS) and a battery-level sensor. The robot uses the information extracted from such sensors to understand its current state i.e. the current speed, the distance to obstacles, its exact current location and the power consumption rate, such that it can traverse effectively towards a given goal location. The lack of principled ways to incorporate such sensor signals into a motion control design causes the robot to be less efficient in its decision-making process. In this circumstance, the fundamental question is: "What are the hidden relationships between inputs that can be used to infer the best action for the robot when input states are approximated?". To answer this question, a potential motion controller must ensure that every robot action complies with various navigational constraints and the demands of exploration tasks. These can only be achieved by maximising the synergies between them.

Motion control design is a large search space problem - A set of different classes of robot control methodologies have been established for use in

various robotic domains [81]. Depending on the complexity and predictability of an environment, roboticists can choose a suitable control model to run a specific robotic task. Given that the objective of an exploration task has been initiated and the fundamental model of a motion control is chosen, we are still left with the problem of refining the motion control specification as follows:

1. the selection of factors or features as inputs to the reasoning and decision-making element of motion control.

2. the generation of the control model's internal structure to relate one factor to another.

3. the fine-tuning process of numerical constants representing the influence of each factor on the controller's actions.

Each of these specifications has design-spaces varying from just few options to hundreds or thousands of possibilities. Given such broad options, the motion control design specification problem for realistic exploration tasks entails a large, computationally intractable search space. The task to find near-optimal solutions from the available options is a demanding and time-consuming process. Due to the complexities described above, and the presence of sensor noise and environmental uncertainty, it is highly unlikely that a near-optimal motion control can be derived - perfectly formed - from first principles [57]. Noise, in particular is a key concern, in that on a single trial, it is possible for a poor controller to perform well and conversely, for a good controller to occasionally perform poorly. Given this level of noise, conventional gradient based search techniques are infeasible. In contrast, EC search is more robust in the face of such hostile search spaces through a systematic and iterative evaluation process. EC is capable in handling local minima problem due to the existence of noise by adaptively distributing candidate solutions over an entire search space. It also provides multiple trials for each candidate solution such that the effect of noise can be reduced.

1.3 Research Goal

The research goal of this thesis is related to the the following fundamental research question:

"How can we help roboticist to design a near-optimal specification of motion control's policy for an autonomous robotic exploration task automatically using techniques from EC given various levels of prior knowledge?"

In section 1.2, we outlined three challenges that make motion control design harder using heuristic or handwritten approaches. In order to overcome these problems, the search for some specifications of a motion controller must be derived automatically. In this thesis, our main goal is to develop an automatic approach using EC techniques. In this work, our primary EC search technique is Grammatical Evolution (GE) [103]. GE is an EC technique that can searches for good programming solutions automatically given an objective function and user-defined grammar specifications. GE is a variant Genetic Programming (GP) [73]. GP is a form of EC where search is conducted across a space of programs in a limited domain. The prime advantage of GE over other forms of EC is that it is able, through the user-defined grammar to limit search only to syntactically correct individuals. This yields advantages in terms of limiting the search space and being able to easily handle grammars where the space of valid programs is tightly constrained. In experiments that require the evolution of numerical constants of motion control, we also use another EC technique called Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [52] as benchmark for comparison. CMA-ES is chosen because it is a highly adaptive and informed search heuristic with a high rate of convergence for noisy numerical problems. This is achieved by sampling new candidate solutions via multi-variate normal distribution. In the case of the evolution of a robotic exploration system, the convergence rate is an important element to get the result faster since the task requires longer time of evaluation for one candidate solution.

To be more specific, we divide our main goal into three sub-goals as follows:

1. To develop a novel framework for evolving motion control of an exploration task using GE.

2. To apply the proposed GE approach to a custom motion controller and the well known Dynamic Window Approach (DWA) motion controller [42, 10] on a set of exploration task configurations.

3. To compare and analyse the exploration performance of GE-based motion control and handwritten motion control.

In this thesis, our EC techniques are designed to be flexible. Roboticists can select to automate the search process ranging from a more basic search for numerical constants to complex search for the inputs and structure of motion control. This flexibility enables our technique to accommodate different control specification problems according to our prior knowledge of the task. In practice, this flexibility offers roboticists the scope to blend their domain knowledge with automatic search to design controllers that leverage human expertise and automatic optimisation.

1.4 Method Overview

Artificial Intelligence (AI) techniques have proven to be more effective than handwritten approaches to adapt models to deal with uncertain situations and constantly changing environments [126] such as autonomous exploration tasks. One useful class of AI techniques is EC. EC techniques are defined as stochastic algorithms whose search methods model natural phenomena: genetic inheritance and Darwinian striving for survival [113]. EC techniques have been shown to be effective in challenging search problems characterised by noise and local minima where most conventional search techniques failed [3]. In this thesis, we utilise GE, a class of EC techniques, as a motion control

optimiser by producing series of potential motion control specifications by stochastic search until a near-optimal, or at least feasible solution is found.

Specifically, we explore the effectiveness of GE to automatically find good policy for a motion controller. In most control systems, policy is the primary determinant of the best action to take, given the current state of a robot. Policy can also be defined as the decision-making model that determines the quality of a motion controller [126]. We encode our policy representations as ordinary mathematical functions. Such encoding, also known as the white-box model normally is found in many motion controllers [9, 42, 90, 86, 133]. In contrast to black-box models such as Fuzzy Logic [124, 62, 56] or Neural Networks [69, 8, 2], ordinary mathematical functions have a distinct advantage i.e. the representation of a policy is clear and inspectable, and thus can be more easily be analysed. Moreover, unlike less transparent approaches, the inspectable solutions produced by GE can be compared across different applications to look for common features and patterns. Over time such patterns can inform future research. However, a shortcoming of using policy functions is the possibility of selecting the wrong structure. The scope of poor choices include: choosing incorrect factors, improper relationships between factors and poorly tuned weights for each factor. These mistakes are easily made in handwritten approaches. GE is good at configuring this problem by allowing users to explicitly define possible solutions in a grammar. This is in contrast to most other GP techniques where grammars for solutions are not as easily configurable. The grammars used by GE are compactly described using Backus-Naur Form (BNF).

In this thesis, we divide the problem of motion control optimisation into several autonomous exploration configurations. Each experiment, presented in the following chapters, is dedicated to a specific exploration configuration which varies in one or more of the following settings:

1. the number of robots involved in an exploration task.

2. the number of objectives or missions to accomplish in an exploration task.

3. the policy function representation.

Through the use of progressively richer grammars we show how each of these problems can be posed as an automatic search task. This thesis demonstrates the utility of framing the search for control as a search problem guided by an easily defined grammar relevant to the search domain.

1.5 Contributions

In the following, we present our contributions in this thesis:

1. A general framework of the evolution of motion control for an autonomous exploration task using grammar guided evolutionary search – through GE. We present how we set up our navigational frameworks to allow motion control to be expressed as a design problem. Vital components in the steps include encoding controller's specification in terms of search space domain, setting up learning environment, performance indicator and genetic operators.

2. Evolution of a motion controller for coordinated multi-robot exploration based on the proposed GE framework. The policy of a motion controller in a multi-robot system is evolved to minimise exploration duration while avoiding collision with obstacles and interaction between team members. We use GE to evolve numerical constants of the policy's function by considering trade-off between distance to moving objects, static obstacles and target locations. A comparison between the evolved policy and a handwritten policy is presented with simulation results.

3. Evolution of a single-robot motion controller with multiple exploration objectives based on the proposed GE framework. The exploration mission in this experiment requires the robot to minimise exploration time, as well as, minimise power consumption via robot movements.

Both requirements are explicitly taken as factors to be considered by the policy of the robot motion control in a form of multiplicative function. We extend the evolution with GE to evolve not only the constants, but also unary functions applied to each factor. We compare simulated exploration results from the best GE-derived policy, a CMA-ES-derived policy and a handwritten policy.

4. Evolution under minimal prior knowledge of the policy's structure. We broaden the application of GE to evolve the whole structure of a policy in case roboticists unable to determine the structure that governs a good policy. We propose a number of grammars that allow numerical constants, unary functions and the structure of a policy to evolve accordingly. We found that exploration performance can be improved by allowing more aspect of specifications of motion control to evolve significantly.

5. Evolution of the selection of factors to be used in a Dynamic Window Approach (DWA) motion control. This GE grammar is presented to help roboticists to select appropriate factors automatically from a set of candidate factors. In the case of DWA controls, there are various DWA schemes with different set of factors reported in literature, however, there is no general rule that specifies the selection of factors or even the number of factors needed. As such, we present this method to automate the process of searching the best factors using GE.

1.6 Thesis Outline

This thesis is organised into eight chapters. The remaining chapters of this thesis are:

1. *Chapter 2 Literature Review* - This chapter reviews the literature directly relevant to this work. The aim of this chapter is to describe and evaluate related methodologies used in the development of motion control.

Another aim is to examine corresponding EC techniques implemented in the design stage of motion control. We also discuss basic EC techniques used in this thesis namely GA, GE and CMA-ES.

2. *Chapter 3 Theoretical Frameworks and Descriptions of Experiments* - This chapter describes the theoretical frameworks and concepts implemented in this thesis, including exploration strategies, decision-theoretic based motion control and GE optimisation methods.

3. *Chapter 4 Autonomous Exploration with Multiple Robots* - The design problem of motion control in an exploration task with teams of robots is presented in this chapter. We present our first GE technique to design a coordinated control for a small team of unmanned ground vehicles (UGVs) mapping an unknown two-dimensional area.

4. *Chapter 5 Single Robot Multi-Objective Exploration* - We extend our GE technique to a single robot system with multi-requirements planetary exploration task. Besides GE, we also report another EC technique called CMA-ES as the performance benchmark of evolving numerical parameters of a motion controller.

5. *Chapter 6 Evolving the Structure of a Policy* - GE technique is further extended to evolve not only numerical parameters of a motion controller. Multiple data types such as arithmetic operators and mathematical functions are also evolved to find a near-optimal motion controller. We propose a number of BNF grammars to accommodate different levels of evolution based on available prior knowledge.

6. *Chapter 7 Evolving Input Factor Choice of a Policy* - This chapter presents the final experiment of evolution for a well-known reactive motion controller known as Dynamic Window Approach (DWA). We use GE to find appropriate factors as the input to the policy of a DWA controller as well as searching for the best policy structure.

7. *Chapter 8 Conclusions* - This final chapter provides conclusions of this

thesis and highlights future directions of this work.

1.6.1 Related Publications of the Described Contributions

Parts of the above experiments have been published in the following publications:

1. *Chapter 4* - M.F. Ibrahim and B. Alexander. Evolving a Path Planner for a Multi-Robot Exploration System Using Grammatical Evolution. In: *Proceedings of 7th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)* 2011. [59].

2. *Chapter 5* - M.F. Ibrahim and B. Alexander. Designing a Navigational Control System of an Autonomous Robot for Multi-Requirements Planetary Navigation using Evolutionary Algorithms Approaches. In: *Proceedings of 12th Australian Space Science Conference (ASSC)* 2012. [60].

3. *Chapter 6* - M.F. Ibrahim and B.J. Alexander. Evolving Decision-Making Functions in an Autonomous Robotic Exploration Strategy using Grammatical Evolution. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 2013. [61].

2 Literature Review

The aims of this chapter are:

1. to describe and evaluate different methodologies that have been used to develop exploration strategy and motion control for autonomous robotic exploration tasks.

2. to describe and evaluate different EC techniques that have been used to optimise motion control design.

3. to provide brief introduction to EC techniques used in this thesis namely GA, GE and CMA-ES.

In the previous chapter, we briefly examined the broad challenges relating to autonomous robotic exploration. We focused on some issues relating to robust motion control design and trade-offs required by the task of exploring in complex unknown environments. In this chapter we survey past and current works in the field of autonomous robotic exploration in general and explore decision-theoretic based motion control in particular.

Our survey shows that autonomous exploration problem is an active research area. As exploration applications become more complex and diverse, there is a growing range of exploration strategies being proposed to cater for different exploration requirements [130, 36, 20, 80, 14, 118, 6, 104, 19]. External factors such as environmental conditions and internal factors such as the robot's kinodynamics and sensory noise are known to add substantial complexity to the task of developing an effective exploration strategy [82]. As such, the structure

of exploration strategies must be customised to each exploration application. This includes the design of robots' motion control [71]. Given the diversity of applications and the noise and uncertainty inherent in exploration tasks, the construction of motion control is a significant design challenge. However, thus far, there has been limited research done on automated search for better motion controller design for autonomous exploration. This may be due to the complexity of such systems [69].

The remainder of this chapter is organised as follows. In the next section we briefly present a general framework for autonomous robotic exploration. In section 2.2, we present a set of selected works that shows some methods used to construct exploration strategies. Subsequently, we examine a number of corresponding motion control designs used in those exploration strategies. In so doing we also recapitulate the challenges posed by autonomous exploration tasks as observed in the literature. In section 2.3, we outline the latest works that use EC as the technique to optimise motion control design. Finally, we provide a brief introduction to GA, GE and CMA-ES for fundamental reference of this thesis in section 2.4.

2.1 Autonomous Robotic Exploration

Autonomous robotic exploration is a complex task. In this thesis, the term *exploration* drawn from the definition in [130] which states that exploration is the act of moving through an unknown environment while building a map that can be used for subsequent navigation. This particular area of robotics is applicable to a large range of robotic applications, including surveillance [111], rescue [15], volcanic exploration [96], planetary exploration [67], wildfire detection [105] and even vacuum cleaning [57]. Autonomous robotic exploration is challenging due to the need to perform several non-trivial tasks simultaneously. Figure 2.1 illustrates a general framework for autonomous robotic exploration [119]. At its core, the framework consists of three modules namely *mapping*, *localisation* and *motion control*.

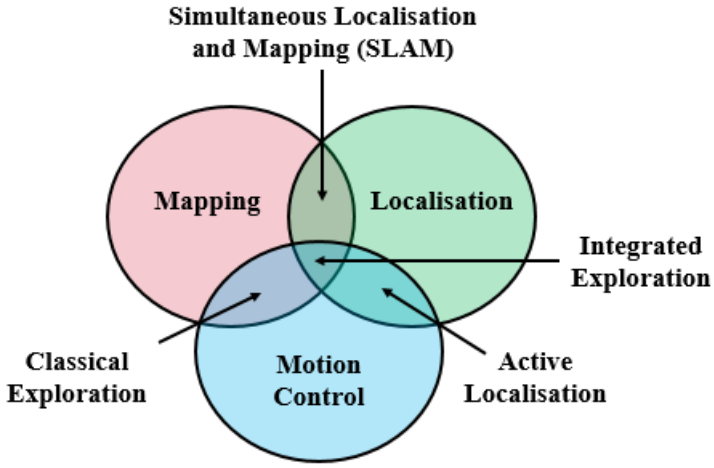


Figure 2.1: A general framework of autonomous robotic exploration [119]. The mapping module provides a map model to the robot system. The size of the map model increases dynamically while the robot explores. The localisation module tracks the current robot's pose estimation by using information gathered from the map model and sensory data. The motion control module plans paths and drives robots from one location to another to explore unknown environments. The overlapping areas in the figure indicate that two or more modules are run simultaneously to perform an exploration task.

Mapping is a process of building a model of an environment via the robots' sensors by interpreting acquired information into a specific representation. This representation is typically known as a map model. Well known examples of representations are grid maps [94] and topological maps [75]. As a part of autonomous robotic exploration activities, *mapping* provides a systematic way to capture, interpret and manipulate sensory data into useful environmental information that can be interpreted by robots to inform further actions. On the other hand, the *localisation* module resolves the problem of finding the position of robots whilst exploring. *Localisation* is of importance since it provides the basic but vital pose information for robots to track their current location. *Localisation* interpolates between a map model sourced from the *mapping module* and sensory data to precisely estimate robots' positions. Thus, *localisation* and *mapping* modules are tightly coupled which introduces an overlapping problem called *simultaneous localisation and mapping (SLAM).*

The third component, *motion control* module is the focus of this thesis. The *motion control* module is the key element that drives robots from one location to another. In order to perform effectively in an exploration task, *motion control* must intelligently decide on the next target location for the robots, guide them to follow a path or a direction and meet any additional exploration mission constraint such as collision avoidance or energy conservation. There are a few types of exploration set up. The first type is to combine *motion control* and *localisation* modules, producing an *active localisation* solution. Given a map, *active localisation* attempts to improve pose estimation by commanding robots to move to locations that reduce pose ambiguity [36, 80]. Another type is classical exploration merges the *mapping* and *motion control* modules. This set up focuses on guiding robots to explore environments optimally fast and build a map model simultaneously while assuming perfect pose estimation. With advancements in sensor technology, precise pose information can be achieved by installing positioning devices such as a highly accurate diffential global positioning system (GPS) for outdoor robots or invisible barcodes for indoor robots [57].

Finally, integrating all three modules produces an *integrated exploration*. In these integrated approaches, the *mapping, localisation* and *motion control* modules are executed simultaneously. Robots acquire environmental information through their sensors, build a map, localise themselves, plan trajectories and move themselves to a new location. The whole process iterates continuously. Exploration is typically considered complete when robots succeed in building a map model for a whole bounded unknown space within a specified time frame.

Within this exploration framework, this thesis focuses on the aspect of *motion control* for two-dimensional unknown planar space exploration. This thesis discusses *motion control* for autonomous mobile ground vehicles under various exploration set ups. Next, to add context to our work we explore methods that have been presented in literature to develop exploration strategy and *motion control* for autonomous robotic exploration.

2.2 State-of-the-Art of Autonomous Robotic Exploration System

In the domain of unknown environment exploration, systems vary according to features of the system and environment in which motion control is embedded. Hence, it introduces some challenges to design motion control for a specific exploration strategy. In this section, we divide the discussion into two parts. The first part presents state-of-the-art of exploration strategies reported in literature. Subsequently, state-of-the-art of motion control is discussed in the second part.

2.2.1 State-of-the-Art of Exploration Strategies

In the past two decades, there are vast approaches of exploration strategy being proposed to accommodate various exploration missions. We divide the works on exploration strategies into two distinct categories: single-robot exploration and multi-robot exploration. Both categories are similar in that they share some common strategy frameworks, but are different due to the focus of research problems. Single-robot exploration research topics often deal with fundamental exploration components such as the strategy to find an immediate target location [130, 36, 20, 80, 43, 6, 19]. Meanwhile, multi-robot exploration research topics focus more on the coordination strategy of robot teams [131, 132, 82, 14, 118, 104, 67]. Either single-robot or multi-robot exploration is used, the motion control design relies heavily on the selection of exploration strategy.

Single-robot exploration - One of the most widely used exploration approaches is the Frontier-based exploration strategy [130, 80, 6]. Yamauchi (1997) [130] introduced a frontier concept to guide robots to explore autonomously in unknown environments. Frontiers are defined as regions on the boundary between free space and unexplored space. The exploration process has two-stages. The first stage is to assign a frontier as the target

location. This is known as task or goal assignment. The latter stage is to move the robot from its current location to the assigned target location. This task is called motion control. In this approach, frontier points are generated to produce a set of possible next locations. The fundamental concept of this approach is that when a robot moves to a frontier point, it can see a previously unexplored area and maximise information-gain. Within a generated set of frontiers, the robot is instructed to choose the nearest accessible unvisited frontier with an obstacle-free path. Then, the robot moves to the frontier. Once the robot reached the frontier, it will perform a sensor sweep and update the map. Then, the process of generating new frontier and moving to the best frontier is repeated until the whole environment is explored. One main feature of Frontier-based exploration is that it is not restricted to specific geometries of an environment. This generality feature makes the approach suitable for diverse environments. In many applications [130, 131, 132, 80, 6, 14, 118], the system is made independent from the environment's geometrical condition by applying a spatial map representation such as occupancy grid map to identify free, occupied or unknown spaces. With such representation as the base information, frontiers can be determined from a list of grid cells that satisfy the definition of a frontier.

From a motion control point of view, a reactive obstacle avoidance controller is commonly used to drive robots towards a selected frontier. In an ideal static environment with perfect localisation, the controller can follow the generated path as close as possible to get to the frontier as fast as possible. However, in the real world, the robot has to deal with a few issues [63]. First, the localisation accuracy. When exploring a large environment, the robot tends to accumulate more errors over time on its real location. As a consequence, the robot might not be able to reach the given frontier due to erroneous positioning. Second, in the presence of sensor noise, the built map sometimes lacks accuracy. Because of these problems, the controller must set a certain distance threshold of the robot position from any obstacle to avoid unpredicted collision. To maximise navigational efficiency, the robot should learn from experience to avoid too small or large distance threshold to avoid both collisions on one hand and unnecessary navigational restrictions on the other. Both issues can be considered as problems arising from the uncertainty of state signals. Third, in

a dynamic environment, the controller must be designed to recognise and avoid moving obstacles as early as possible. This can help the controller to avoid collision by changing its direction of movement and at the same time finding the quickest way to go back to the defined path. Alternatively, the robot can dynamically generate a new path towards a frontier location after detours at a certain distance. Nevertheless, the selection of the right navigational decision to be made is difficult without refinement of strategy through experience [71].

Frontier-based exploration techniques have evolved over time. Makarenko et.al. (2002) [80] suggests an integrated exploration approach based on the Frontier strategy to improve pose uncertainty. In this work, candidate frontiers are ranked by assessing the following three metrics i.e. information gain, navigation cost and localisation quality. An additional localisability metric is introduced to estimate the robot's pose uncertainty at a given position. By aggregating these three metrics, an exploration that balances area coverage, map accuracy and speed of exploration can be achieved. In Makarenko's work the utility of information gain is calculated based on the average entropy of a frontier. Cost of navigation is calculated using a numerical potential-field algorithm. The utility of localisability is calculated by converting the state-covariance-matrix of a robot position, obtained from an EKF-SLAM algorithm [27], into scalar utility using Shannon's information function. Conceptually, the localisability value approximates the level of uncertainty of a robot's position at a certain location by measuring distances to pre-defined feature locations. Finally, a total utility function is used to aggregate all three utility metrics using a weighted sum equation. The frontier with the highest aggregated utility is preferred as the target location.

Basilico and Amigoni (2011) [6] extends the Frontier-based strategy to deal with multiple-criteria exploration missions. In the presence of multiple criteria, trading off between criteria is shown to be difficult. Such example can be seen in an exploration that needs a balance between time-constraint criteria and power-constraint criteria, while at the same time, considering acceleration and communication range constraints. One dominant approach to optimise the trade-off is to use multi-criteria decision making (MCDM) tools [46]. In this decision-theoretic approach, a utility function is created to evaluate frontiers'

quality. The function is responsible to weigh each criteria dynamically in order to rank every frontier accordingly. However, explicitly weighing up all criteria with precision remains demanding.

An exploration strategy can also be developed using Feature-based method. Feder et.al. (1999) [36] proposed an instance of Feature-based exploration, aiming at developing a navigational strategy for autonomous underwater vehicles in unstructured environments. This work constructs a hybrid approach by combining a stochastic mapping algorithm with an adaptive sensing metric to guide the exploration task. Stochastic mapping uses a single state vector to represent estimated locations of the robot and features with associated error defining uncertainty in these location estimations. Adaptive sensing is used to evaluate a set of possible sensing actions when robots move to a specific feature location. Similar to Makarenko et.al. (2002) [80], a utility function calculating all entropy measurements, is used as a metric to choose the best next robot's action by trading off between uncertainty in feature locations and uncertainty in the vehicle estimated position. This type of exploration also falls under the class of active localisation exploration.

Another exploration strategy is Graph-based exploration. Choset and Burdick (2000) [20] develops the Hierarchical Generalised Voronoi Graph (HGVG) method for reactive sensor-based exploration. One prominent advantage of HGVG is that it can reduce search problems in arbitrary dimensioned spaces into a one-dimensional problem. However, the deficiency of the HGVG is that it is designed to assign tasks based on local information only gathered from line-of-sight sensor. In Garrido et.al. (2009) [43], potential target locations of unexplored areas are extracted from a potential map constructed by a hybrid algorithm called Voronoi Fast Marching (VFM). In this algorithm, an Extended Voronoi Transform (EVT) is implemented to mimic the repulsive force from obstacles, while the fast marching method is used to find the best path to the shortest and safest point from the EVT point of view. In order to select a target location, EVT is used to construct two matrices: W and VT. The W matrix is calculated to represent the distance of each cell to the nearest obstacles meanwhile the VT matrix is constructed to represent the density of a cell being visited. By performing weighted sum operation on both the W

and VT matrices, a final WV matrix is obtained. A cell with the highest value in WV matrix is chosen as the target location.

Cepeda et.al. (2012) [19] proposes a Behaviour-based exploration strategy that eliminates the decoupling of goal-assignment and motion control. The proposed exploration strategy is purely reactive in which sensing activity is mapped directly to action activity. Four basic behaviours are selected in the strategy including obstacle-avoidance behaviour, past-avoidance behaviour, locate-open-area behaviour and disperse behaviour. The emergent exploration behaviour is triggered using a Finite State Automata (FSA) to combine all basic behaviours using a weighted summation or subsumption [11] operation according to the current state. The output of the emergent behaviour goes directly to robot actions. The real advantage over other exploration strategies is its simplicity. However, setting up the emergent behaviour becomes less easy as the number of basic behaviours increase. In fact, even for a small number of behaviours, the control design needs to deal with the arrangement of behaviour sequences, behaviour priorities and switching points between behaviours. Finding the right settings for these requires experience, thus in some ways this system can be categorised as an expert system.

Multi-robot exploration - Multi-robot exploration is introduced to take advantages of team-based perspective. One advantage of using multiple robots to explore an unknown area is that the exploration task can be completed faster than a single robot exploration. It can also increase map accuracy by merging overlapping sensory data from multiple robots. However, it is argued that the existence of a number of robots can reduce the exploration performance if the type of collaboration between robots is not defined properly [82, 67]. The problems that may arise include task redundancy and deadlock situation due to each robot tries to avoid collision with each other [118]. Thus, multi-robot exploration poses new challenges in team coordination. Behaviours of team robots must be coordinated effectively to ensure each robot contributes significantly to the exploration task.

Yamauchi (1999) [131, 132] extended the Frontier-based approach for multi-robot exploration systems. To achieve good coordination, Yamauchi

(1999) developed an approach that is cooperative and decentralised. The robots cooperate by global map sharing. Each time a robot sweeps its sensor, it updates a local grid map. This local map is combined with the robot's global grid map. Then, the global map is broadcast to other robots, assuming communication between robots is available at all time. For all robots, an individual robot that receives global maps from other robots will integrate those external global maps with its own maps via log odds representation [93] to resolve any conflict between maps. In simple words, the probability of a cell being occupied is added between maps and the sum is stored as the probability of the corresponding cell in a merged global map. In order to find the target location of each robot, the Frontier-based strategy and motion control from single-robot exploration [130] is extended. In this setting, each robot plans its own navigational action without considering other robots' plan. By adopting this control strategy, the system acts in a, mostly, decentralised way. The advantage of this approach is that each robot is independent and in the event of an individual robot failure, the effect on other robots is minimal. However, from the perspective of exploration optimality, the risk of task-duplication is high. This can happen when more than one robot selects the same frontier as there is no mechanism for the robot to share information regarding other robots' target location. The reactive navigational control used in this approach encounters several limitations. This is due to the fact that the dynamic obstacles, in this case the robot's team members, are incorporated as static obstacles within a grid map. Hence, the robot cannot differentiate between real static obstacle and dynamic obstacle when making decision on the robot's next action. Consequently, this limitation can lead to a situation in which one robot blocks another robot or multiple robots block each other. When this happens, the selected frontiers might be marked as inaccessible, thus the corresponding unexplored regions are not visited. Secondly, robots represented as static obstacles can disturb the best frontier's selection due to the change in cost calculation to each frontier. In turn, robots can tend to opt for longer route towards a selected frontier.

In contrast to the above approach, Burgard et.al. (2005) [14, 118] designed a centralised coordination system for multi-robot exploration. This centralisation is designed to avoid having several robots move to the same

frontier as their target point. A decision-theoretic approach is used for robot-to-frontier assignment. An appropriate frontier for each individual robot is assigned by managing the trade-off between information gain at the frontier and the cost for a robot to reach a point. A utility value is allocated for each frontier to measure information gain of an unexplored area. Once a frontier is chosen as the target location of a robot, the utility values of the frontier and its surrounding frontiers are reduced so that the probability of the same frontier being chosen by other robots is minimised. The cost of the robot-to-frontier path is calculated as the optimal path from the current robot location to the frontier location in a static environment. A graph search algorithm is used to compute this cost. From a motion control perspective, the lack of integrating dynamic elements in the cost calculation can reduce the exploration performance because the robot has no idea on the direction of movement of other robots. As with other conventional motion control, the controller will rely solely on reactive collision avoidance behaviour in which it deals with dynamic obstacles only when the objects are in the robot's sensor range. As a result, the problem of blockage between robots is still possible.

Putting the problem of coordination in a broader context, Mataric et.al. (2003) [82] describes different coordination levels of robotic exploration systems. Four exploration approaches are evaluated. Two essential aspects of coordination are considered: commitment level and coordination level. Each approach falls into a category determined by these aspects. Commitment level is divided into two types namely: a fully committed strategy and a fully opportunistic strategy. For a fully committed strategy, the robot must complete its current task before looking for a new one (in this case, the task is target location). Meanwhile, a fully opportunistic strategy enables the robot to consider a new task and drop the ongoing task at any time. Coordination level is also grouped into two types named as individualistic strategy and mutual-exclusion strategy. An individualistic strategy allows the robot to perform task allocation individually like the algorithms proposed in [131, 132]. Whereas, a mutual-exclusion strategy dictates that only one robot can be assigned to a task as in [14, 118]. For motion control, all four combinations of these categories use the same reactive control that directly controls the robot's motors. The robot's motors are controlled by a weighted average function

of two signal states: target distance intensity and an obstacle avoidance algorithm. From their experimental results, the article suggests that the performance of exploration strategy changes according to the levels of noise in the system. Overall, they found that the exploration that incorporates fully opportunistic and mutual-exclusion strategies outperforms other strategies within acceptable noise level. However, there is no analysis on the effect of the variation of motion control to these exploration strategies.

Pal et.al. (2011) [104] explores another means of team coordination by the mean of frontier regions clustering. Unexplored areas are divided into several regions based on the number of robots using the K-means technique. Each robot is assigned a region to explore according to the Hungarian method [74] that generates robot-region pairs based on estimated path distance from the robot position to the centroid point of a region. Finally, each robot is assigned the lowest cost frontier cell within the region as the target location. This approach has been shown to minimise travelling time due to the fact that it is able to reduce the infeasible frontier list by 50%.

For task-specific context, Kisdi and Tatnall (2011) [67] developed a task-assignment strategy for planetary exploration based on honeybee search for searching cave entrances on Mars. The honeybee search imitates the way natural bees search for new hives. Caves are important elements to be searched on Mars because liquid water is highly likely to be found in these areas. Due to warm temperature and low radiation, deep water reservoirs could still be maintained in the present day by the geothermal gradients and magmatic heat. Cave characteristics on Mars are not well understood. Due to this fact, the honeybee search exploration strategy is set up to allow multiple robots to visit the same potential caves. Their results show that the number of caves found increases by using honeybee search with appropriate number of robots. However, this approach is limited due to its configurations that rely heavily on specific environmental models.

From the above works, we have shown that different configurations of exploration strategies produce different challenges and have different limitations. In summary, the challenges in terms of decision making complexity

can be categorised as follows:

1. state signal uncertainty due to localisation errors and sensor noise.

2. trade-off between multi-signal in a decision-making function.

3. trade-off between multi-objective in exploration missions.

4. coordination in multi-robot exploration.

To meet these challenges, the selection of an appropriate motion control structure for an exploration strategy is vital to achieve good exploration performance. The best choice of control is highly dependent on the task and the environment. One control approach may fit a particular exploration context very well but be unfit for others. Therefore, attempts must be made to find the best controller for each exploration context. Next, we describe some state-of-the-art of approaches that have been reported in literature to design motion control to fit an exploration context.

2.2.2 State-of-the-Art of Motion Control

In this part, common types of motion control used in exploration tasks are presented. We formulate the problem of motion control design as a decision problem, in which the paradigm of decision-theoretic control is discussed. Finally, we discuss a few optimisation approaches that have been used to improve motion control performance.

Type of motion control - The complexity of exploration requirements clearly has a direct relation to the complexity of motion control design. Different controllers are suitable for different exploration strategies, the nature of environments, and the physical robot design. From the perspective of decision-making model, there are two major classes of motion control in exploration strategies: reactive/deliberative control and behaviour-based

control [81]. Both classes share some common features i.e. purely react to input to produce output and apply bottom-up design approach. In contrast, they are differ in that reactive/deliberative control typically applies single-layer decision-making model, while behaviour-based control adopts multi-layer decision-making model.

Reactive/deliberative control uses either reactive control or deliberative control or a combination of both controllers. Deliberative control [58] acquires sensory inputs, builds internal knowledge (often global information) and performs an in-depth decision process to reason about action choice. Meanwhile, reactive control [90, 95] imitates the 'stimulus-response' of behaviours found in biological settings by using local information to decide an action in approximately real-time. In systems where deliberative and reactive control are combined [134], the deliberative part is used at the policy level to decide the target location of a robot, while the reactive part executes the robot motion according to its policy's decision. In general, reactive/deliberative control pass all input signals into a sole decision-making model, and the model decides the best action.

The second class of control type is behaviour-based control [124, 37, 110]. Behaviour-based control is designed from a set of simple behaviours such as avoid-obstacle and locate-safe-area. It is a bottom-up approach in which complex behaviours such as exploration are built on top of simple behaviours. In this way, a simple behaviour can be designed separately. Each behaviour has its own decision-making model that decides an action based on the selected number of inputs. Then, complex behaviour is achieved via another independent decision-making model that determines a final action by consolidating a set of proposed actions from simple behaviour modules.

In our works, we focus on the reactive and deliberative control type. In particular, we examine the decision-making module of this control to improve exploration performance.

Decision-theoretic based motion control - Whether the motion control is reactive/deliberative or behaviour-based type, we observe that there

are at least some parts in these controllers that apply the concept of decision-theoretic approach [90, 95, 69, 134, 124, 37, 110, 9, 122, 123, 42, 10, 68, 66, 5, 7]. Decision-theoretic based control is implemented using a cycle called: sense-evaluate-act [126]. This is equivalent to the formal definition of decision-theoretic approach described in Mintzberg et.al. (1976) [91] that consists of three major phases: identification-development-selection. Identification is a phase of using existing information as the input to a decision problem. For motion control, this can be the act of gathering data from sensors about the environment. The development phase provides a routine to search for possible solutions or generate a new one. This can be implied in motion control as the process of filtering only feasible actuator output to be considered for the next action. Finally, the selection phase is where the actual choice between alternatives is made. In a motion control context, this is equivalent to the process of selecting the final actuator output to be sent to the hardware. One of great challenges in designing decision-theoretic based control is to formulate a decision function that is responsible for evaluating the quality of an action given a set of state signals [63]. In some literature, it is called a reward function [63] or a value function [83]. Because it is the decision function that ultimately dictates robot behaviour, the optimisation of this function is of prime importance.

Conventionally designed motion control - Conventional controllers are designed to guide robot movements. One earlier method is based on the Artificial Potential Field (APF) approach [66, 5, 134]. The robot motion is guided by a field of artificial forces called potential field. The potential field is generated on an occupancy grid map of a robot based on the combination of attractive force, F_a and repulsive force, F_r representing the distance to goal location and obstacle locations, respectively. In an APF, the decision-making model is simply a summation of F_a and F_r. The steering direction of a robot is determined based on the gradient of forces between the current robot cell and the next possible robot cell. Another motion control approach is the Vector Field Histogram (VFH) -based algorithms [9, 122, 123]. Unlike APF that defines the steering direction using a cell-based algorithm, VFH uses sectors of a histogram grid map to find the steering direction. Each sector, consists of a number of grid cells, contains information of obstacles in terms of polar

obstacle density function. In VFH algorithms, a decision-making model is used to select the best sector with low polar obstacle density to determine the safe direction of robot's movement. The decision is simply calculated based on the angular distance between safe sectors to the target location. Another motion control class that applies the decision-theoretic based approach is the popular Dynamic Window Approach (DWA) -based algorithm [42, 10, 68]. DWA-based algoritms sample the velocity space (output) according to robot dynamics. Each feasible sample is given a score based on a trajectory cost defined as a weighted average function. In Fox et.al. (1997) [42], the trajectory cost is constructed to trade-off between the following states: target heading, nearest obstacle distance and current velocity. Four numerical parameters are required to be tuned. In addition, Brock and Khatib (1999) [10] converts DWA control to a global navigation function by incorporating a wave-propagation -based navigation field parameter. Kiss et.al. (2012) [68] improves DWA evaluation function by using Model Predictive Control (MPC) to eliminate weight tuning. However, MPC still requires a number of model coefficients to describe a response effectively. A decision-theoretic control based on situated activity paradigms is proposed in Nearness Diagram (ND) -based algorithms [89, 90, 32]. ND algorithms apply multi-stage decision models to identify the state of obstacles and the state of target location. Each possible situation will fall into one of the several defined scenarios. Then, in each scenario, the robot's actuator output is calculated using scenario's unique velocities calculation. Beside these other decision-theoretic based controls can be found in literature. Examples include distributed behaviour controls [110] and velocity obstacles control [7].

All of the above examples of motion control are based on handwritten design. Often, optimisation of the parameters of the controller is not explicitly described. This is due to the fact that most of these controllers have simple structure, hence, a trial-and-error approach can be carried out quite quickly. However, applying such an approach to a more complex navigational system is burdensome. For example, if we include new state signals to the system, the controller's decision-making model becomes harder to derive using a handwritten approach. The additional factors of environmental uncertainty and sensor noise make this problem even worse [60]. The challenge for

most decision-theoretic based motion control is to develop a decision-making function that can predict nearly accurately the consequences of performing an action and determine its suitability for the current situation.

As the complexity of the task of configuring motion control increases rapidly with the complexity of exploration tasks, the requirement for additional learning capability for such problems becomes clear. Roboticists have the difficulty to persuade their knowledge to get the best control design for the robot to use. As such, machine learning approaches are used to automate some part of control design. We discuss two main classes of optimisation technique from machine learning in the following.

Reinforcement learning optimisation - One optimisation approach is to let a controller learn from unlabelled instances. This approach is called unsupervised learning. In unsupervised learning. when input data is available, appropriate output is learned through a reward function as the performance index. One prominent example of unsupervised learning approaches is reinforcement learning [120]. Kollar and Roy (2006) [70] discuss the impact of optimising motion control settings on map quality produced as the result of an exploration. A learning approach to optimise motion control using reinforcement learning is proposed. Classically, robot kinodynamics are not considered when calculating the cost of moving from a robot location to a potential target location. Thus, the robot tends to choose a target location that theoretically has the lowest cost - absent of any consideration of kinodynamics. However, the real actions that the robot must perform may be harder than as expected such as sharp turns, hence, the target location becomes costly or even infeasible. As such, motion control must be designed to compensate for this restriction by adding kinodynamics parameters in the robot's decision model. This work proposed a reinforcement learning approach to find the best decision model that can improve the inference process that maps input signals data to action selection. In this work, Policy Search Dynamic Programming (PSDP) is used to train multi-step policies, while Support Vector Machines (SVM) are applied to train iterated one-step policies. Doroodgar and Nejat (2010) and Liu et.al. (2012) [29, 78] use a Hierarchical Reinforcement Learning (HRL) approach to develop control architecture for semi-autonomous rescue

robots. The controller is responsible for deciding a task to execute and selects whether the task should be carried out by a human or robot. Both scene exploration and victim identification tasks are learned using a HRL approach based on MAXQ methodology. In general the biggest barrier to the use of reinforcement learning is related to the reward function identification. In most cases, the reward that needs to be given to each learned action of the robot has to be manually defined.

Evolutionary Computation optimisation - In contrast to reinforcement learning approaches, EC, a class of algorithms that imitates the process of natural selection to find solutions for optimisation problems, is a promising tool for searching good motion control configuration for an exploration strategy. In principle, EC techniques are defined as stochastic algorithms whose search methods model the natural phenomena: genetic inheritance and Darwinian striving for survival [113]. As such, we argue that the solution to motion control optimisation is a product of systematic meta-heuristic search process. EC has some advantages over reinforcement learning approaches [128]. One of the advantages is that EC is good at handling partial observable problems in which states uncertainty is fairly high since it is not manipulating relationship of subsequent state-action pairs. Rather, it find solutions via aggregated evaluation mechanism. Other advantages include EC crafts solution's representation simpler and more flexible. In addition, EC works well to handle problems with large or continuous action spaces. Those advantages make EC becomes promising to optimise robot motion control in various configurations. In the following section, we thoroughly describe some works that have been done using EC techniques to improve motion control.

2.3 Evolutionary Computation based Motion Control

EC techniques have been successfully applied in various robotic navigation problems from a simple navigation task such as wall-following to a complex

navigation like planetary reconnaissance. In this section, we thoroughly go through some prominent works in the literature that best exemplify the effectiveness of EC techniques to optimise navigation performance via the evolution of several types of control structure including evolution done on decision-theoretic based controls.

One of early works that implements EC techniques on robot motion control is Schultz (1991) [114]. A specialised software, SAMUEL was developed to evolve rules of a rule-base control to solve sequential decision tasks of navigating in simulated dense mine fields. A Genetic Algorithm (GA), a common EC technique, is used to evolve a set of numerical values represent the strength of each rule. The strength measures the expected influence of each rule on decision process to choose an action. A number of customised GA operators, namely specialize, generalize, creep and merge, are introduced besides the standard GA operators (crossover and mutation) to accommodate the complexity of the problem of rule modification. The fitness of a solution is defined as the pay-off of a robot reaching its rendezvous point. Comparison of results exhibits that the success rate of navigation increases from 8% by using hand-design approach to 96% by using the EC technique.

GA is also used by Ram et.al. (1994) [107] to evolve a schema-based behavioural control for robot's navigations in obstacles-dense environments. To demonstrate the capability of EC paradigm to find a good controller setting, GA is used to tune a set of numerical weights to balance the level of influence of five schema parameters. Those schema parameters are obstacle gain, goal gain, obstacle sphere-of-influence, noise gain and noise persistence. Modification of the standard GA is done by changing the GA's genome representation in which an array of floating point numbers is used rather than binary numbers for faster convergence of the genes. As such, GA operators i.e. crossover and mutation are also modified to cater for the different genome representation. By simulating a single goal navigation, the performance of a candidate controller is measured by using a function of weighted penalties consisting of number of collisions, number of steps and distance travelled. The function also enables the designer to select the navigation style of a robot by manually changing the weights of penalties, in the context of which GA will tune controller's

parameters accordingly to suit with the selected style.

Evolution of behaviour-based control has another variant in which a layered paradigm is implemented. Fernandez-Leon et.al. (2009) [37] describes this Layered Evolution (LE) to learn several basic behavioural modules separately. In this work, a GA is used for each evolution of a single behavioural module. Each GA is set up differently according to a specific type of behaviour. This includes setting up genomes and a fitness function. Two basic behaviours are evolved: phototaxis and wall-following. The idea of using LE is to ensure the complexity of evolution would not grow exponentially when dealing with complex behaviours. Instead, complex behaviours can be constructed intelligently via concatenation of evolved basic behaviours. This work demonstrated that a light-source-as-goal-location navigation behaviour can be designed by combining phototaxis behaviour with wall-following behaviour. Their experimental results show that the layered evolution controller outperformed the monolithic evolution controller. However, a bottleneck of this approach is the subsumption architecture that is responsible for behaviour aggregation being defined manually.

Knudson and Tumer (2011) [69] describe a neuro-evolutionary based reactive controller for a single robot navigation. A neural network (NN) controller is used to guide the robot to navigate through various obstacle rich environments using a decision-theoretic approach. In this approach, a NN controller acts as the decision maker. The state and action spaces are designed to help the controller make a selection from a set of possible actions. The selection is done by assigning a numerical value for each potential action that represents the action quality. An e-greedy evolutionary algorithm, a variant of GA is implemented to tune the NN-controller's numerical parameters namely weights within the network. In such configuration, two input states, the nearest obstacle distance and destination heading are considered while controlling the robot heading. The experimental results show that the neuro-evolutionary based control outperforms rule-based control on all tested environments.

All of the above approaches tune parameters to motion control. In contrast, Xiao et.al. (1997) [129] developed a novel Evolutionary Planner/Navigator

(EP/N) tool to generate an adaptive path planning system. EP/N is designed to accommodate various path planning problems with minimal changes in GA's configurations. This adaptability is achieved via two steps: 1) maintain chromosome structure consisting of feasible or infeasible paths, and 2) define eight specialised operators with pre-defined probability of each operator to be chosen based on the condition of chromosome. Although EP/N gives more freedom for the designer to apply the tool on various path planning problems, defining probability values is already a non-trivial task that requires complex tuning process before the evolution process can be performed.

Another work that deals with path planning problems is Dozier et.al. (1998) [30]. This work implements an Artificial Potential Field (APF) method to navigate a micro-rover named AGIE-3. Unlike standard hand-coded APF methods where the next best positions are selected from a set of pre-defined candidate next positions, this work uses a Simple Genetic Hill Climbing (SGHC) evolutionary technique to dynamically generate a number of candidate next positions. With a relatively small population, SGHC-based APF is able to select a better next position thus improves the robot navigation performance in terms of the length of path, number of steps and change of direction. In constrast to [129], this work is limited in that the SGHC is designed with fixed structure, thus make it hard for the technique to be modified for other variants of path planning.

In a more complex multi-robot application, Vadakkepat et.al. [124] investigates evolution of Fuzzy Logic based controllers (FLCs) in a multi-level behavioural-based robot-soccer system. In this system, robot tasks are decomposed into four levels of behaviours where the top two levels manage team strategy and coordination. Meanwhile, the bottom two levels handle individual behaviours and primitive actions that perform the real robot movement. A GA is applied to optimise certain aspects of FLCs at different levels. For the top level behaviours, the GA is used to tune the rule-base part of FLCs with the objective to improve strategic decision of team robots, whereas, for the bottom level behaviours, a GA is implemented to tune membership functions parameters of FLCs to enhance the smoothness and accuracy of robot actions. Various configurations of GA's chromosomes are designed manually

to accommodate different settings of FLC tuning. Their results show that the evolution has improved team's game strategy against a hand-coded rival team's strategy and at the same time trained more skillful behaviours.

In the context of exploration, Sotzing and Congdon (2005) [116] describes an application of GA to the multi-robot Frontier-based exploration task. The objective is to improve robots coordination in order to optimise the process of mapping unknown environments. To meet this objective, they focus on the task-assignment component by evolving parameters involved for bidding process. Those parameters determine how a trade-off between the following two costs is done, to bid for a frontier cell. The parameters are 1) estimated cost of getting to the frontier cell, 2) estimated information gained by mapping the frontier cell. Other than that, the GA is also used to encode several threshold values to be tuned i.e. minimum distance between frontiers and maximum time elapsed before entering then next bidding round. Their results show that the GA-tuned parameters outperform hand-coded parameters in all explorations through significant reductions in time steps taken. It should be noted that learning processed used was computationally intensive with 167 hours used to evolve 20 individuals for 100 generations. This search converged well in these experiments. However as the number of parameters to be tuned increases this search task will become harder. This challenge makes evolving an exploration task harder than other robotic tasks.

Another variation of EC technique is presented in Hsu and Juang (2013) [56]. A robot control with a wall-following task was optimised using an elegant evolutionary technique called as species-differential-evolution-activated continuous ant colony optimisation (SDE-CACO). In this work, SDE-CACO is applied to optimise two robot's Fuzzy controllers of Interval Type-2 (IT2FC) [51]. Both controllers control robot's orientation and speed, respectively. SDE-CACO is responsible for learning robot behaviour to follow walls at a pre-defined distance accurately. An online approach to evolve rule generation of IT2FC using SDE-CACO is proposed. The controller's fitness is evaluated based on average moving speed and average distance to wall. While the application of SDE-CACO improves IT2FC performance, direct use of this evolutionary technique for other domains is infeasible due to the major changes

in SDE-CACO configuration required.

In the abovementioned works, we can observe several similar characteristics of the use of EC to robotic control. EC techniques, dominated by GA, are primarily deployed to evolve parameters of motion control. The exception being the application of GA to the path planning problem in [129, 30] where numerical parameters are two dimensional position coordinates. In most cases, a substantial customisation of the search framework was required for each problem, which limits the generalisability of the techniques used.

Another prominent EC technique that has attracted the attention of researchers in evolutionary robotics is Genetic Programming (GP). One advantage of GP over GA is its capability to find a small computer program to solve a given problem. One pioneering work that applied GP to robotic problems is by Koza and Rice (1992) [72]. The authors introduce GP to automatically generate a functional program to perform the robotic task of moving a box. GP is used to learn the structure of the program to effectively control a simulated robot to push a box to the wall. The fitness function to evaluate each candidate solution is determined by taking the aggregated distance between the end position of the box and the nearest wall from four evaluation runs. The preparation steps to design the learning approach is compared with Q-learning, a reinforcement learning approach, compared to which it showed a significant reduction in the number of steps required from 13 to 5. Thus, GP in general gives a simpler solution than the reinforcement learning approach.

Reynolds (1994) [108] applied GP to evolve the controller of a corridor following behaviour robot. The evolution process is done to find the best program to directly map a noisy sensor information to a noisy steering direction. The fitness is calculated based on the number of steps taken to move along a corridor without any collision. The results show that in such a noisy environment, the surviving solution is a compact and robust controller structure rather than a brittle and opportunistic controller.

Nordin and Banzhaf (1997) [100] demonstrates the first attempt to use GP

to evolve a robot's control in real-time, attached to a miniature mobile robot platform Khepera. In this work, GP is applied to form a world model in the form of a function. Then, the controller uses this model as the basis for decisions of motor actions to avoid static obstacles. The model is constructed as a symbolic regression function that associates each sensory-motor state with its fitness. In each decision cycle, the motor speeds is selected from the state with the highest fitness. The world model function is incrementally evolved by analysing the fitness of sensory-motor states from the previous actions taken. Evolution has been extended one step further by evolving not only numerical parameters, but also some of simple arithmetic and logic operations. As the evolution is executed in real world, one disadvantage is that the robot keeps colliding with obstacles during the first few generations as the learning curve of the robot's control is incremental.

Clifton and Fang (2007) [22] address the issue of learning time as an important challenge that needs attention when developing an EC technique for robotic exploration tasks. In this work, GP is implemented to directly control a robot performing exploration in an unknown environment. Although the exploration was quite conceptual and simplified, it has shown that the GP is able to find a program that can appropriately choose among 4-direction moves to explore unknown areas at each decision cycle. To investigate learning time efficiency, time pressure has been put into the fitness function i.e. the original fitness is divided by the measured time the robot completes the task. Surprisingly, the exploration performance does not increase markedly. This result suggests that within that framework attempting to go faster lead to other problems.

Despite these successful applications of GP technique to various navigation problems, formulation of description of real-world problems into appropriate GP search-space representations is still a non-trivial task especially for a complex navigation task such as robotic autonomous exploration. In EC terms, a real-world problem must be represented in the form of a numerical string called as genotype. The genotype is then converted into phenotype that represents a syntactically correct program code. In GP, there is little distinction between genotype and phenotype. As such, for a specific navigation problem, the genotype must be carefully designed to satisfy both

the requirements of the problem space module and search space module. The task of achieving balance between both with a single grammar is difficult.

An alternative to GP, called Grammatical Evolution (GE) [103] resolves the above problem by having a clear separation line between genotype and phenotype (search space and problem space). A great advantage of this separation is that the robot designers do not have to worry about the internal structure of the evolutionary search space that carries out the work of heuristic search of potential solutions. Instead, designers can focus on preparing the best design for problem space module in a form of a generative computer grammar i.e. Backus-Naur Form (BNF). In turn, GE will handle the processes of converting the problem space into search space using its unique generative production rules and performing heuristic search using its pre-defined search engine. While there are many successful applications in various disciplines have applied GE to solve their problems, the number of works of GE on robotic problems are quite small. The first work that applies GE to work on a robotic problem is done by O'Neill and Ryan (1999) to control a Khepera robot's dance improvisation. The authors established a behaviour grammar to generate a behaviour-based control architecture. The objective is to create a basic but effective behaviour to move away from the robot's starting position.

GE applications to robotic tasks is extended by Mingo and Aler (2008) [88] to find a navigational control of a simulated Khepera robot. In this work, GE is assisted by a reinforcement learning approach to enable the robot to learn during it lifetime (simulation). A grammar is developed to learn the if-then rule sets to guide the robot exploring a maze map without collision.

Burbidge et.al. (2009) [12] presents the GE-based evolution of a robot's reactive control. In this work, a grammar that constructs a register machine program is developed. The program is used to map 16 sensor values to 2 wheel motor values in order to perform the moving-to-light-source task while avoiding obstacles. A fitness function that compromises between a reward of being nearer to the light source and a penalty of being close to an obstacle is set. Results have shown that the evolved controller is better than random controllers in terms of incurring less collisions, high probability to reach the

light and generalise well in new environments.

Burbidge and Wilson (2014) [13] studies the application of GE based on a vector-valued function estimation in which a robot control architecture is built on. A novel generative vector grammar for a vector-valued function estimation is proposed to generate register machine programs. The empirical result on a simulated Khepera robot control shows that the proposed vector grammar outperforms a classical list-valued grammar in terms of learning performance. This evidence proves that the selection of grammar structure in GE is essential to the overall performance of evolution.

Our earlier works on the application of GE to robotic autonomous explorations are presented in [59, 60, 61]. These works will be presented in details in later chapters of this thesis. In brief, we have implemented GE on various settings of motion control to meet different requirements of exploration strategies including single-robot systems and a multi-robot system.

2.4 Introduction to EC Techniques - GA, GE and CMA-ES

In this section, we briefly describe an introduction to EC techniques that are used in this thesis namely GA, GE and CMA-ES. This section serves as a basic reading for readers who are not familiar with EC techniques to understand the fundamental concept behind each of them. Note that through out this thesis, some parts of EC techniques explained here might be modified to cater for the domain under study which will be further discussed in chapter 3.

2.4.1 Genetic Algorithms

Genetic Algorithms (GA) are a class of EC technique that imitates the process of biological evolution to solve a problem with a rough function.

GA implements a stochastic search method in that it selects and evaluates a number of candidate solutions called chromosomes iteratively until a near-optimal global solution is found. In order to perform such search, GA chooses a number of candidate solutions accordingly to form a pool of parent chromosomes. The parents are forced to mating by the means of exchanging some genetic materials to produce new candidate solutions called offspring. Then, each offspring is evaluated to know how good is the solution to solve the problem in hand. Eventually, the current population of candidate solutions is replaced with a new population consisting of new offspring. The above processes are repeated for few generations of chromosomes' population until a satisfied solution is found.

The depicted processes simulates the Darwinian evolution seen in biological settings. This theory of evolution proposed by Charles Darwin states that reproduction is the ability of an organism to adapt continuously to its changing environment [23]. The concept of natural selection is a powerful paradigm of biological evolution to produce the best organism in that a good organism who by nature chooses a good partner will have higher chances to produce better child. If such natural selection continues by hundreds or thousands of generations, it is likely possibly the latest generation will be few times better than the early generations. In the real-world, the evaluation of an organism might not be measured explicitly. Instead, it is measured from the capability of organisms to survive. The survivability determines the quality of organisms to predict, take action and get through challenges they anticipated during their life. Thus, organisms can be ranked by the quality of their survivability.

In this section, we briefly introduce the canonical GA technique as presented in [44] as the main reference of GA used by researchers worldwide. In general, the core process of finding optimal solution for a problem has two steps [113] as shown in figure 2.2.

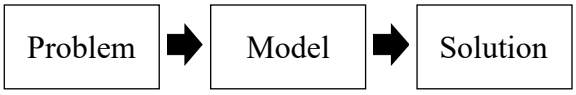


Figure 2.2: Steps for finding optimal solution of a problem. 1) Formulate a problem with a model, and 2) Find solution based on the developed model.

GA implements the same process. First, GA requires users to formulate a problem with an exact model. One advantage of GA is that it allows a problem to be encoded with a representative model that could precisely describe the form of candidate solutions. This is in contrast to conventional search methods such as gradient-based algorithms that require a complex problem to be simplified with an approximate model which can be trapped with an over-fitting problem. In GA, the step to convert a problem into a model is known as encoding. In [44], the model is developed using binary representation which some literatures called it as the binary GA. Precisely, the possible form of candidate solutions in the problem space or also known as phenotype is converted into its equivalent binary number representation called genotype. GA uses genotype in its search space to perform genetic operators. Note that GA works with a finite search space. Hence, an appropriate genotype design is required to ensure GA can search for the best solution within the specific range of possible solutions.

To illustrate the conversion from phenotype to genotype, consider a problem to solve a sampling function or a sinc function as in equation 2.1.

$$z = |\frac{sin(r)}{r}| \tag{2.1}$$

where

$$r = sqrt(x^2 + y^2) + 1.0E - 8 \tag{2.2}$$

In this problem, there are two parameters that define the result of the function: x and y. The objective is to find a pair value of x and y that produces the maximum value of z. For simplicity, we restrict x and y as integers. Assume that the acceptable range of x and y is between -10 to 10. Thus, the possible number of solutions is 21x21 = 441. Given such problem specification, the required conversion is straightforward such that the integer value of x and y must be represented with their equivalent binary signed numbers. For example, if $x = 5$ then its equivalent representation is $x = 00101$. Meanwhile, for a negative integer, its two's complement binary number can be used. In this case we can say the integer value of x and y is the phenotype and the binary

number of x and y is the genotype. Before genotypes can undergo genetic operations, they must be concatenated together as a string of binary numbers called chromosome. In this case, the genotype of x and y can be concatenated together as in figure 2.3. Here, the first five bits or also known as genes represent the value of $x = 4$ and the latter five genes represent the value of $y = 9$.

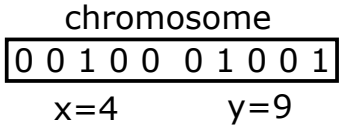


Figure 2.3: An example of a chromosome represents parameters x and y of equation 2.2. The first five bits or genes represent $x = 4$ and the last five genes represent $y = 9$.

Once a model is established, the next process is to use the model to find the best solution. Algorithm 1 defines the process taken by GA to use a specific model to find a near-optimal solution. Briefly, the process is started by GA generates a population of random chromosomes (genotypes). The number of generation gen_{size} and population size pop_{size} are pre-defined by users. Each chromosome, using its corresponding phenotype, is evaluated with an evaluation function defined by users. This evaluation is used by GA to rank the performance of each chromosome. The fitness value, represents the relative performance of one chromosome to another is acquired by using a fitness function. Typically, a fitness function has a direct relationship with an evaluation function. Then, genetic operators are performed on the evaluated chromosomes to produce offspring. The genetic operators used in this GA are selection, crossover and mutation. Note that all genetic operators manipulate genotypes in their operation. The current population of parents is replaced with a new population of offspring. After that, the generation of new population undergoes the above process again. This process is repeated for gen_{size} set by users. The process terminates after executing gen_{size} generations or until a satisfactory result is achieved. A chromosome with the best fitness value is selected as the best solution. Next, the details on fitness calculation, selection, crossover and mutation are explained.

Algorithm 1: General description of GA process.

1 Initialise the number of generation gen_{size}.
2 A population of chromosomes with the size of pop_{size}, encoding a set of candidate solutions are randomly generated.
3 Each chromosome is evaluated with a user-defined evaluation function.
4 Fitness value for each chromosome is calculated based on a fitness function.
5 *Selection* operator is performed to select parent chromosomes.
6 *Crossover* operator is performed to produce new offspring.
7 *Mutation* operator is performed to change a small amount of genes of the new offspring.
8 The current population is replaced with the new offspring.
9 Repeat from step 3 for gen_{size} time or until the performance criterion is satisfied.

Fitness calculation - Fitness value for every chromosome is used by GA as an indicator to select arbitrary chromosomes into a pool of parents. Fitness value for each chromosome can be calculated once the performance of all chromosomes has been acquired using a fitness function. For example of the sample problem above, assume pop_{size} is set to 4. Let's say four chromosomes are randomly generated by GA and their corresponding performance are acquired using equations 2.1 and 2.2 as the evaluation functions. Table 2.1 shows the fitness value f_i of chromosome i that can be calculated by using fitness equation 2.3.

$$f_i = \frac{z_i}{\sum_{j=1}^{4} z_j} \tag{2.3}$$

Table 2.1: An example of four chromosomes, $i = 1, 2, 3, 4$ and their fitness value calculated using equation 2.3.

i	Genotype	Phenotype	z_i	Fitness f_i
1	11101 11100	x=-3,y=-4	0.000	0.000
2	00010 11111	x=2,y=-1	0.352	0.273
3	00001 00000	x=1,y=0	0.841	0.650
4	00101 11011	x=5,y=-5	0.100	0.077
			$\sum$ =1.293	$\sum$ =1.000

Selection - At each generation, a subset of the current population of chromosomes is selected as parents. Chromosomes are selected based on their fitness value. A fitter chromosome, indicating a better solution, has better chance to be selected. There are many selection methods available in literature. The original method used in [44] uses Roulette-wheel method. With Roulette-wheel method, the probability of a chromosome to be selected as a parent is proportionate to the fitness value of the corresponding chromosome. Algorithm 2 illustrates Roulette-wheel method.

Algorithm 2: Roulette-wheel method.

1 Calculate the accumulated fitness f'_i of every chromosome i.
2 Set the size of a parents' pool $prnts_{size}$.
3 **for** $m = 1$ *until* $m = prnts_{size}$ **do**
4 Generate a random number r_m between 0.0 to 1.0.
5 Select chromosome j $(j = 1, 2, ..., i)$ if $f'_j \geq r_m$ and $f'_{j-1} < r_m$.
6 Insert chromosome j into the parents' pool.
7 $m++$.

Note that, the accumulated fitness of chromosome i can be calculated by summing up its fitness value f_i with previous chromosomes' fitness $f_{i-1}, f_{i-2}, ..., f_1$ as in equation 2.4.

$$f'_i = f_i + \sum_{j=1}^{i-1} f_j \tag{2.4}$$

In an application where each pair of parents can produce a pair of offspring, $prnts_{size}$ always equals to pop_{size}. One chromosome can be selected few times into the parents' pool. Figure 2.4 shows an example of a parents' pool created from a population of chromosomes defined in table 2.2.

Crossover - Crossover operators exchange arrays of bits or genes within some parents to produce offspring. In most cases, two parents are involved, but this needs not be the case [113]. Conceptually, crossover between parents is performed with an assumption that if two or more good parents exchange their genes, the produced offspring may be better than their parents. As with

Table 2.2: An example of four chromosomes with their accumulated fitness calculated using equation 2.4.

i	Genotype	Fitness f_i	Accumulated fitness f'_i
1	11101 11100	0.000	0.000
2	00010 11111	0.273	0.273
3	00001 00000	0.650	0.923
4	00101 11011	0.077	1.000

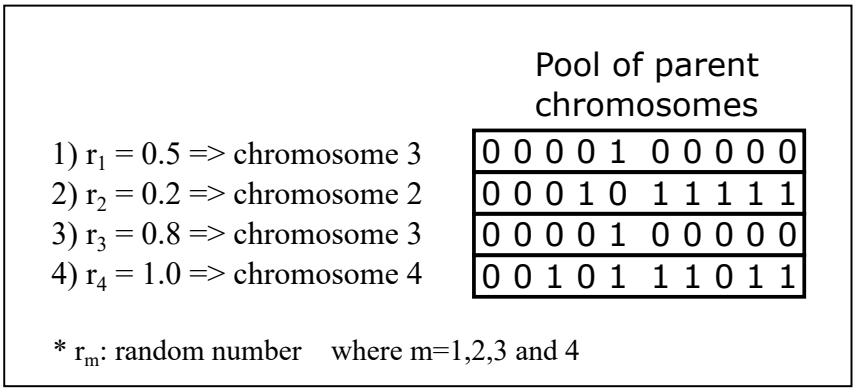


Figure 2.4: An example of a pool of parent chromosomes created using a population of chromosomes in table 2.2.

selection methods, there are variants of crossover approaches. One common approach is the one-point crossover. With the one-point crossover, there are two parameters involved. The first parameter is the probability of crossover ρ_{xo}. This parameter is set by users, normally between 0.9 to 1.0. ρ_{xo} controls whether a parent pair will perform crossover operation or not. If a random number generated for each parent pair is less than ρ_{xo}, crossover will take place for that particular parent pair. The second parameter is the location of crossover loc_{xo}. loc_{xo} is randomly generated in between the first gene and the last gene of a chromosome. This location tells GA the crossover point where one parent will exchange an array of its genes with its partner. Figure 2.5 shows an example of one-point crossover operation applied to chromosomes in the parents' pool of figure 2.4. Note that loc_{xo} of the first parent pair is between gene 6 and gene 7. Thus, parents A and B maintain their first 6 genes and exchange their last 4 genes with each other to produce new offspring AB' and AB''. For the second parent pair, loc_{xo} is randomly generated in between gene 4 and gene 5, thus parents C and D exchange the last 6 genes between

each other to produce offspring CD' and CD''.

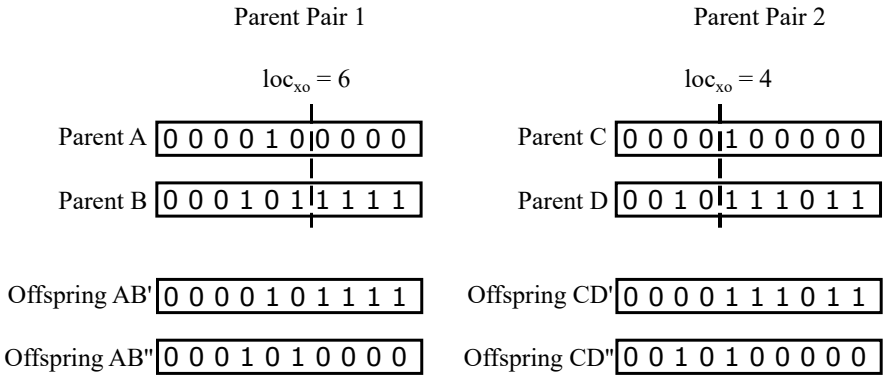


Figure 2.5: An example of one-point crossover operation.

Mutation - The final genetic operator used in every generation is mutation. Mutation is a small change in an offspring. This change is required to avoid local minima in our search and to maintain genetic diversity. A standard mutation method is known as point mutation. With point mutation, each bit of a chromosome is conditionally flipped according to the probability of mutation, ρ_{mut} provided by users. Common sense suggests that ρ_{mut} must be very small between 0.001 to 0.1, since if ρ_{mut} is high, the search process will become a random walk and difficult for the solution to converge. This is in-line with the theoretical foundation of GA that is based on the schema theorem [55]. Mutation with low ρ_{mut} tends to increase the probability of schema to survive in the next generation [97]. Figure 2.6 shows an example of offspring produced in figure 2.5 being mutated using point mutation operation. Genes in red color are being mutated.

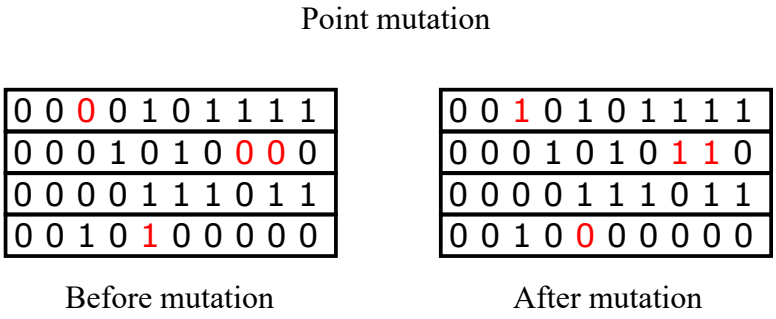


Figure 2.6: An example of point mutation operation. Genes in red color are mutated. The gene with value 0 is flipped to value 1 and vice-versa.

2.4.2 Grammatical Evolution

There is another class of EC techniques called Genetic Programming (GP). GP is a technique that is used to find a computer program or a fragment of a program using evolutionary search. Grammatical Evolution (GE) is a robust variant of GP. GP performs genetic operations on the actual program or phenotype, whereas GE does the operations using a variable-length binary number chromosome or genotype. With such implementation, GE is not restricted to any specific programming language like GP and it can be applied to different languages. From the perspective of GA, one major hindrance that makes the process of solving a problem using EC not effective is because of the ill-defined encoding structure. For a problem that requires multiple data types to be encoded into one single binary number representation, the process of finding the right structure for mapping phenotype to genotype might be sometimes very difficult. Furthermore, for a unique chromosome, user might also have to define a specific method for each genetic operator. In this case, GE is a powerful technique that assists users to simplify the process of mapping phenotype to genotype and vice-versa. The basic idea of GE is to separate a search space and a problem space as independent modules. Thus, the search process can be executed with an evolutionary search technique without having to heavily rely on the changes of a problem to be solved. Meanwhile, users can focus on the design of problem space without having to modify or introduce new genetic operators and finding a way to convert phenotype to genotype. Figure 2.7 shows steps taken by GE to find a near-optimal solution of a problem using evolutionary search.

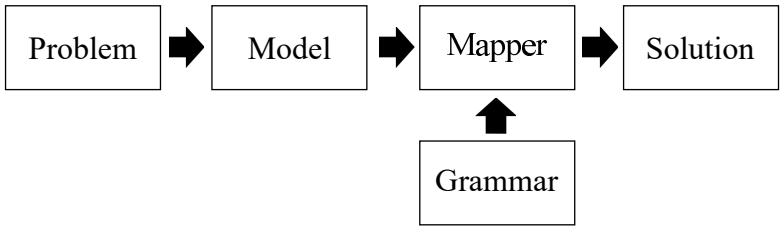


Figure 2.7: GE steps for finding optimal solution of a problem. GE simplifies the conversion from a problem to a model. GE uses a grammar and a mapper to find a solution from the model.

The first step to convert a problem to a model is simplified by GE in that chromosomes will not represent directly the actual candidate solutions of a given problem. Instead, a chromosome stores an array of integer numbers that is used as rule selector for GE mapper to extract rules from a user-defined grammar to execute the model-to-solution step. This representation is universal and mostly unchanged regardless of any problem to be solved. Note that, GE still uses binary number representation for genetic operator processes which can be gathered by using a decimal-to-binary number conversion to convert an array of integer numbers to an array of binary numbers. Thus, a standard binary GA with selection, crossover and mutation operators can be used as the GE search engine.

The unique process of GE is materialised by the introduction of a grammar and a mapper to perform the model-to-solution step. A grammar is a way provided by GE for users to define the possible choices of each parameter being evaluated for a given problem. GE employs a user-supplied BNF grammar as the standard language-specification [103]. BNF is a common notation technique for defining a Context-Free Grammar (CFG) [4, 49]. In the field of computer science, BNF is widely used to define the syntax of a language. GE manipulates BNF grammars to remove the complication of designing each chromosome's structure. Figure 2.8 shows an example of a BNF grammar that defines the possible choices of parameters x and y for the problem of equations 2.1 and 2.2.

```
1 <expr> ::= x=<sign><num>; y=<sign><num>;
2 <num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
3 <sign> ::= 0 | -
```

Figure 2.8: The BNF grammar for parameters x and y of equations 2.1 and 2.2.

In brief, each line in the grammar is a production rule. In this example, we assume the evaluation of equations 2.1 and 2.2 is done using C++ programming language. Thus, the rule on line 1 represents statements to define parameters x and y with some values. The rules on line 2 and line 3 become the supporting statements to feed elements that need to be replaced with some values. Note that in this case, elements in the grammar with a name enclosed by the pair <>, i.e. <expr>, <sign> and <num>, also known as non-terminal elements

are the elements that must be replaced with a value that can be chosen from their corresponding elements on the right hand side of symbol ::=, also known as terminal elements. For example, <num> can be replaced by an integer ranging from 0 to 10, while <sign> can be replaced by a negative sign or value 0 indicating a positive sign. Such replacement process is determined by an array of integer values in a chromosome. The function of GE mapper is to ensure this process can be done in order to convert information from genotype to phenotype. Here, phenotype represents a syntactically correct program code in a specific programming language. For every chromosome that represents a candidate solution, GE mapper takes an integer number from a chromosome and a production rule from a grammar. Then, GE mapper uses a mapping function as in equation 2.5 to determine the final value of a non-terminal element. GE mapper will repeat the process until all non-terminal elements are replaced with terminal elements.

$$\begin{aligned} choice \quad = \quad & [\text{integer}] \\ & MOD \\ & [\text{number of choices in a production rule}] \end{aligned} \tag{2.5}$$

To illustrate a single mapping process, consider a simple example as in figure 2.9. Assume that an integer being extracted from a chromosome is 40 and rule in line 2 of figure 2.8 is chosen. By applying equation 2.5, the resultant choice is choice= 40 MOD 11 = 7. Thus, the final value of a non-terminal <num> is chosen from the 7^{th} element of the array of integers in the rule in that <num> is replaced by integer 6.

Algorithm 3 describes the general GE algorithm to perform an evolutionary search process. Comprehensive explanation on GE components with respect to the domain under study is discussed in section 3.3 of chapter 3.

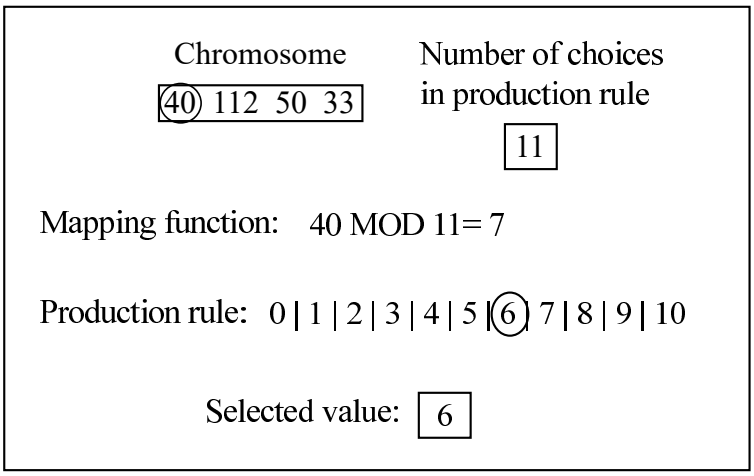


Figure 2.9: An example of single mapping in GE using equation 2.5 and rule in line 2 of figure 2.8. Integer 6 has been chosen to replace non-terminal <num>.

Algorithm 3: General description of GE process.

1 Initialise the number of generation gen_{size}.
2 A BNF grammar is acquired describing the production rules for parameters to be evolved.
3 A population of chromosomes encoding binary strings are randomly generated.
4 **for** *each chromosome* **do**
5 Data from the chromosome is mapped with production rules of the BNF grammar to produce phenotype.
6 Performance of the phenotype is evaluated using a user-defined evaluation function.
7 Fitness value of all chromosomes is calculated using a fitness function.
8 *Selection* operator is performed to select parent chromosomes.
9 *Crossover* operator is performed to produce new offspring.
10 *Mutation* operator is performed to change a small amount of genes of the new offspring.
11 The current population is replaced with the new offspring.
12 Repeat from step 4 for gen_{size} time or until the performance criterion is satisfied.

2.4.3 Covariance Matrix Adaptation Evolutionary Strategy

CMA-ES is an instance of another class of EC called Evolutionary Strategy (ES). ES is designed to deal with numerical optimisation problems in

continuous domain. ES differs from GA and GE in that ES performs genetic operations using real numbers or phenotypes rather than encoded representation. ES has only one genetic operator namely mutation. Typically, an ES technique implements a mutation method called normally distributed mutation. This mutation method enables each numerical value in a candidate solution to be updated with a random value determined by a uni-variate normal distribution function with mean m equals to zero and user-defined standard deviation σ. Normal distribution mimics the natural biological mutation where small changes happen more frequently than larger ones [97].

However, CMA-ES differs from other ES techniques in that CMA-ES samples new candidate solutions or offspring using a multi-variate normal distribution function [52]. With this function, m and σ are both dynamic where these parameters are constantly updated to reflect the most promising distribution according to the fitness values sampled in the previous generation [60]. The basic equation to sample a new offspring (consists of a set of numerical values) is shown in equation 2.6 [52].

$$\mathbf{x}_k \sim N(\mathbf{m}, \sigma^2\mathbf{C}) \tag{2.6}$$

where $\mathbf{x}_k$ is a vector of numerical values of an offspring k and $N(\mathbf{m}, \sigma^2\mathbf{C})$ is the multi-variate normal distribution with mean vector $\mathbf{m}$, overall standard deviation σ and covariance matrix $\mathbf{C}$. The key idea to use $\mathbf{C}$ is to align the shape of search distribution to the contour of the evaluation function f. From experiments it has shown that the adaptation of $\mathbf{C}$ can improve the iteration needed to achieve a near-optimal solution. Figure 2.10 displays an example of the convergence of candidate solutions towards a global optimum by adapting $\mathbf{C}$. Note that how $\mathbf{C}$ adapts as an ellipsoid to align perpendicularly with the contour of fitness landscape. Once aligned, the convergence towards the desired global solution location is fast. Algorithm 4 describe the CMA-ES evolutionary search process to solve a numerical optimisation problem. A detailed description of CMA-ES and the method to update $\mathbf{m}$, σ and $\mathbf{C}$ can be found in [52].

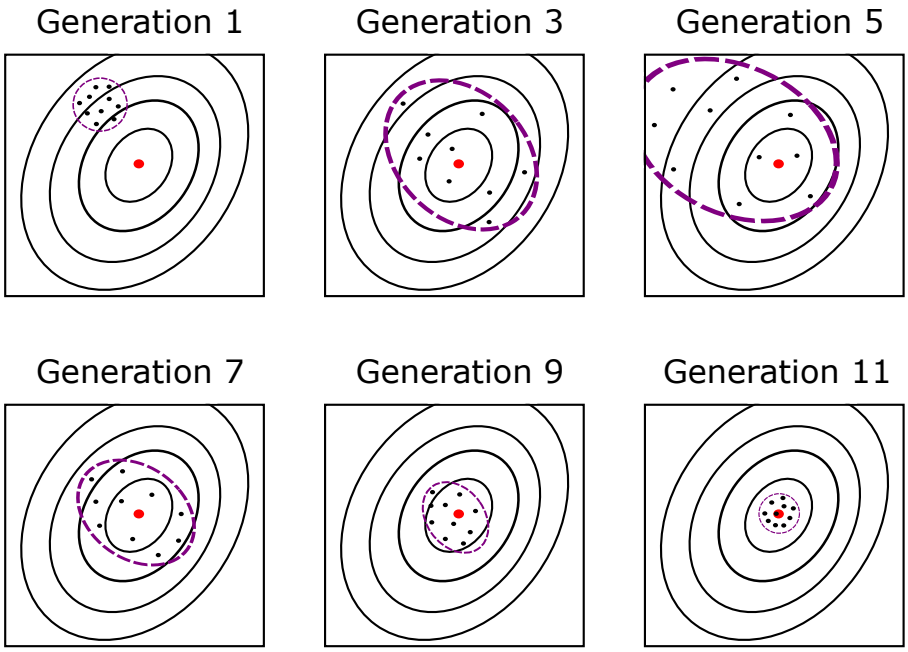


Figure 2.10: An example of CMA-ES candidate solutions distribution on certain generations. Note that the thin lines represent the contours of equal fitness. The dashed line represents the distribution boundaries determined by covariance matrix $\mathbf{C}$. The black dots are the location of candidate solutions and the red dot is the location of the best solution.

Algorithm 4: General description of CMA-ES evolutionary search process.

1 Initialise the number of generation gen_{size}.
2 Initialise population size $\lambda \geq 2$.
3 Set covariance matrix to identity matrix $\mathbf{C} = \mathbf{I}$.
4 Users define initial mean vector $\mathbf{m}$ and standard deviation σ.
5 An initial population with randomly generated chromosomes is generated.
6 Fitness value of each chromosome is calculated.
7 **for** $g = 0$ *until* $g = gen_{size}$ **do**
8 New offspring are created using multi-variate normal distribution function as in equation 2.6.
9 Fitness value of the new offspring is calculated.
10 $\mathbf{m}$ is updated.
11 σ is updated.
12 $\mathbf{C}$ is updated.

2.5 Conclusion

As a conclusion of this chapter, we highlight the major limitations of the existing works found in literature. Decision-theoretic controls are commonly applied in robotic exploration systems. In these systems, a decision function is the main element that determines the action of a robot given a set of state signals. From the literature review, we found that current decision functions are either developed from first principles for an idealised model or developed and refined by hand intuitively. As the complexity of navigation problems increase, so does the need for better learning capability for decision-theoretic motion control. Classical learning techniques such as reinforcement learning have limitations in identifying the reward function for such complex navigation problem. One class of promising learning approach is EC. Early works of EC on robotic applications mostly focus on evolving numerical parameters of robot controllers. Much of this early work evolves task-specific behaviour. While EC techniques have been successfully applied to various robotic navigation tasks, there is little existing work to optimise motion control for autonomous exploration. Autonomous exploration represents a robotic problem in the domain where theories describing the design of optimum control are incomplete or insufficient, and the evaluation of designs is time-consuming and the search space is difficult. Though works to date is promising the generalisation of EC search for control of autonomous exploration is still an open question.

3 Theoretical Frameworks and Descriptions of Experiments

The aim of this chapter is two-fold:

1. to describe theoretical frameworks used in this thesis related to the evolution of motion control for autonomous exploration.

2. to provide brief descriptions of experiments that have been conducted in this thesis.

In this chapter, we first describe a set of theoretical frameworks to develop our evolution-based optimisation technique to improve motion control for robotic autonomous exploration. We also present related concepts, descriptions and algorithms. Throughout this thesis, we consider a Frontier-based exploration strategy as the main exploration framework. Then, we adopt a decision-theoretic based controller for motion control of our exploration strategy. Our main concern is to evolve decision models applied by motion control in various configurations and exploration environments. We describe the model of decision theory applied in this context in section 3.2.2. We also introduce the EC framework we use, namely GE, as the main evolutionary algorithm for motion control design in section 3.3. In the second part of this chapter, we present a brief overview of each of the four experimental studies conducted for this thesis. Each of these studies is described in more detail in later chapters.

3.1 Frontier-based Exploration Strategies

Frontier-based exploration strategies refer to a class of strategies that uses frontier cells in an occupancy grid map as mid-range targets (beacons) for robots to explore undiscovered space. The fundamental concept behind this strategy is based on the following statement: *"To gain the most new information about the world, move to the boundary between open space and uncharted territory"* [130, 131, 132]. As such, frontiers are described as regions on the boundary between free space regions and unexplored space regions that a robot can expect to receive new information once it moves to these regions. Autonomous exploration to map an unknown area is achieved by commanding robots to move to frontiers sequentially until no accessible unvisited frontier exists.

The Frontier-based exploration strategy enjoys significant popularity for several reasons. First, the strategy allows robots to perform exploration in arbitrary environments. Unlike some other strategies which assume a specific geometric structure of the environment, the Frontier-based strategy encapsulates the complexity of environments into generic spatial descriptions using an occupancy grid map model [93]. This feature enables the strategy to generalise to multi-dimensional models. The second advantage of the Frontier-based strategy is the flexibility to engage any relevant robot's motion controller in its framework. The decoupling of goal-assignment and motion control tasks helps roboticist to select or design a controller that best suits an exploration task. For example, a pure reactive collision-avoidance controller that only considers the trade-off between the distance to goal and surrounding obstacles can still be used. In contrast, a more complex controller combining reactive and deliberative approaches which consider the robot kinematics and limitations can also be implemented to increase navigational performance. Figure 3.1 shows the algorithm for the Frontier-based exploration strategy.

To recap, the Frontier-based strategy can be divided into two major parts: a goal-assignment module and a motion control module. The algorithm starts by acquiring data from sensor hardware. Two forms of sensory information

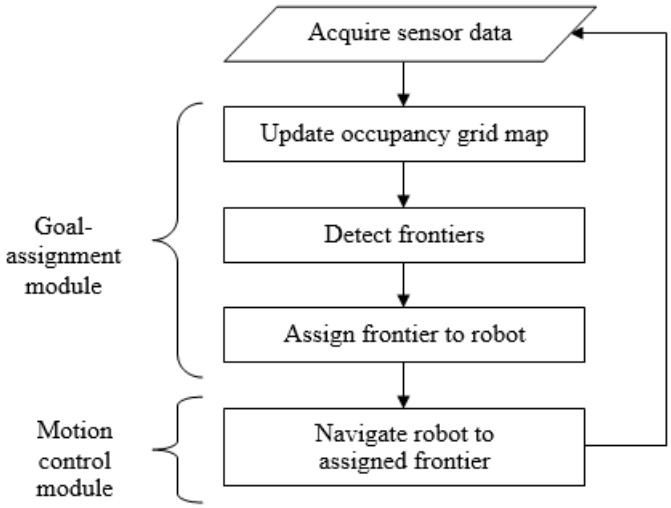


Figure 3.1: Algorithm for Frontier-based exploration strategy. The algorithm starts by acquiring data from sensor hardware. Three steps in the goal-assignment module are then executed: i) robots update its occupancy grid map from currently acquired data, ii) robots identify potential frontier points in the grid map, and iii) robots select the best frontier using its goal-assignment technique. Finally, motion control module is executed to drive the robot to move from its current location to the targeted frontier location. Once robots completes the cycle of steps above, it will repeat them again until the environment is completely explored.

are needed: range reading and pose reading. Proximity sensors such as sonars and laser-range finders are typically used to get range readings that approximate the distance of robots to surrounding obstacles. Pose-estimation sensors such as odometry and global positioning system (GPS), often combined with localisation techniques, can be used to estimate robot's pose. In the goal-assignment module, both forms of data are used to build an occupancy grid map that serves as the robot's model of its environment. An occupancy grid map can be represented as a Cartesian coordinate system containing grid cells in a two-dimensional environment. Every cell maintains the probability that the corresponding cell is occupied. A cell containing 1.0 represents a definitely occupied cell, a cell containing 0.0 is a definitely free cell and a cell with 0.5 is a completely unknown cell. Frontiers are then detected in the occupancy grid map by storing the location of free cells that are adjacent to unknown cells [131]. Subsequently, these frontiers are used as potential target

locations for each available robot to choose. Selected frontiers can be assigned to each robot appropriately by using a standard goal-assignment technique which will be explained in further detail in section 3.1.3. Meanwhile, in its motion control module, the robot activates its motion controller to navigate towards the assigned frontier safely and quickly. Once the robot completes the cycle of steps above, it will perform sensor sweep from its new location, collect new sensory data and repeat the rest of above steps again until the environment is completely explored. There is also a variant in this step, in which a new frontier is assigned using new sensory data prior to reaching the currently assigned frontier. Previous experimental results have shown that this approach can improve exploration performance significantly [82]. Comparison made in [82] shows that robots can explore faster with an exploration strategy that deploys mutual exclusion and opportunistic features. The mutual exclusion feature allows only one robot to be assigned to one frontier. The opportunistic feature defines that the current frontier can be dropped as a target location even though the robot did not reach it yet to allow a 'better' frontier to be selected as a new target. In contrast, other exploration strategies used for comparison apply either one of the features or totally in contrast of both features.

3.1.1 Updating Occupancy Grid Maps

An occupancy grid map is defined as a multi-dimensional random field that maintains stochastic estimates of the occupancy state of the cells in a spatial lattice [35]. State estimation is constructed incrementally by using incoming range sensors' measurements using probabilistic sensor models. Range sensors like laser-range finders and sonars are noisy in nature. In addition, the sensors are also sensitive to the angle of an object surface relative to the sensor and the reflective properties of the surface (absorption and dispersion) [121]. As such, a probabilistic sensor model approach is one of the best approaches to cope with both problems.

Our occupancy grid map follows the procedure of Bayesian estimation to update the occupancy probability of grid cells [35]. To generate an occupancy grid map, the multi-dimensional space of an environment is formulated into an array of discrete cells. This array is also known as the occupancy field, $O(\mathbf{c})$, where $\mathbf{c} = \{c_1, c_2, ..., c_n\}$ is a set of continuous spatial coordinates with n cells. Each cell, c_i, where $i = 1, 2, ..., n$ keeps a probabilistic estimate value of its occupancy state, $P[occ(c_i)]$. Figure 3.2 shows the algorithm to update an occupancy field, $O(\mathbf{c})$.

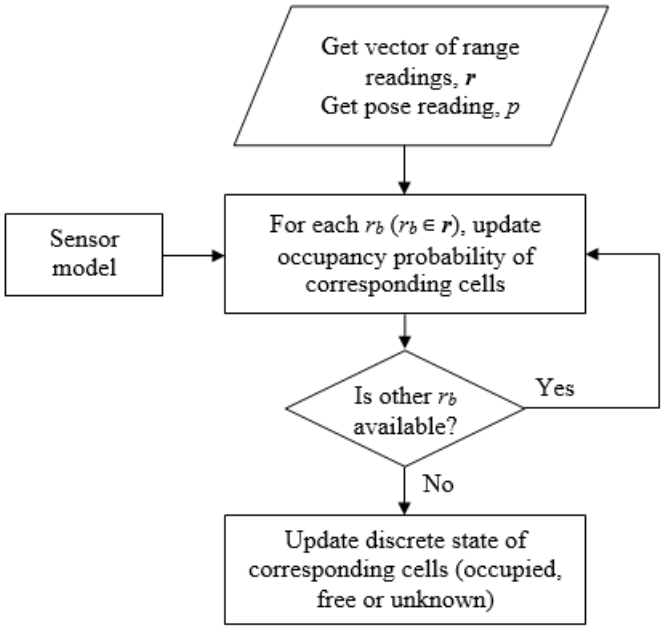


Figure 3.2: Algorithm for occupancy grid update [35]. Sensor data are first acquired. For each range beam r_b, the occupancy probability of corresponding cells along line-of-sight of r_b is updated using Bayes formula correlated with a sensor model. This step is iterated until all r_b are processed. Finally, the discrete state of those corresponding cells are updated to be occupied, free or unknown.

Firstly, a vector of range readings, $\mathbf{r}$ and robot pose reading, p are acquired as the main inputs. p is represented in the form of grid cell position, c_s. $\mathbf{r}$ contains a set of range beams, r_b. Each r_b may represent range measurement from a particular projection angle. r_b measures the estimated distance between c_s and the nearest obstacle position, c_{no} along line-of-sight of the range beam.

After the data acquisition, occupancy probability of corresponding cells, $P[occ(c_i)]$ is updated using the Bayes formula. A sensor model is used to correlate sensor readings with sensor noise. For a simple explanation, consider a one-dimensional sensor model. In an ideal case without noise, the sensor model can be represented as a unit function as in figure 3.3(a) characterised by r and σ. Due to range reading certainty, we can assume that only one cell, c_r with the distance of r from a sensor pose has the probability, $P[r|occ(c_r)]$ of being occupied. On the other hand, a Gaussian probability density function can be used to model a noisy sensor. Figure 3.3(b) shows a Gaussian sensor profile. In this model, cell c_r has the highest probability of being occupied. Meanwhile, its neighbouring cells have lower occupancy probability determined by the Gaussian's width. For a two-dimensional model, the same sensor model profile can be extended by taking into account the sensor range, r and beam's angle, θ to define c_r.

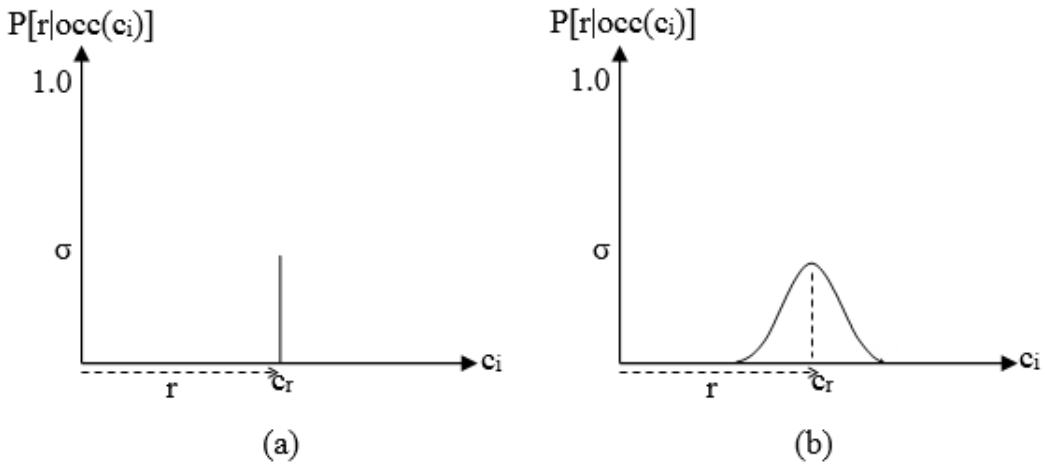


Figure 3.3: One-dimensional sensor model profile [35]. (a) Ideal case: only one cell c_r with the distance of r from a sensor pose has the probability $P[r|occ(c_r)]$ of being occupied b) Gaussian-noise case: cell c_r has the highest probability of being occupied, while its neighbouring cells have lower occupancy probability determined by the Gaussian width.

By combining a sensor model and sensor readings, the occupancy probability of a cell, $P[occ(c_i)]$ given the latest measurement r_{t+1} can be updated with the Bayesian equation in 3.1:

$$\frac{P[r_{t+1}|occ(c_i)]P[occ(c_i)|r_t]}{P[r_{t+1}|occ(c_i)]P[occ(c_i)|r_t] + (1 - P[r_{t+1}|occ(c_i)])(1 - P[occ(c_i)|r_t])} \tag{3.1}$$

where $P[r_{t+1}|occ(c_i)]$ is obtained from sensor model $P[r|c_r]$ using Kolmogoroff's theorem [35] and $P[occ(c_i)|r_t]$ is the current occupancy probability of cell c_i.

Finally, for an exploration task, an occupancy grid map in the discrete form is needed to tell explicitly whether a cell is occupied, free or unknown. To do so, a decision rule can be applied to assign specific state of a cell. In this thesis, we adopt a technique from [130] that uses prior probability to determine the discrete state. Initially, all cells in $O(\mathbf{c})$ are set with a prior probability value, p_{prior} that indicate an unknown state. Here, we set p_{prior} to 0.5. Once occupancy probabilities are updated, the decision rule is applied to determine the discrete state of corresponding cells, $S[c_i]$:

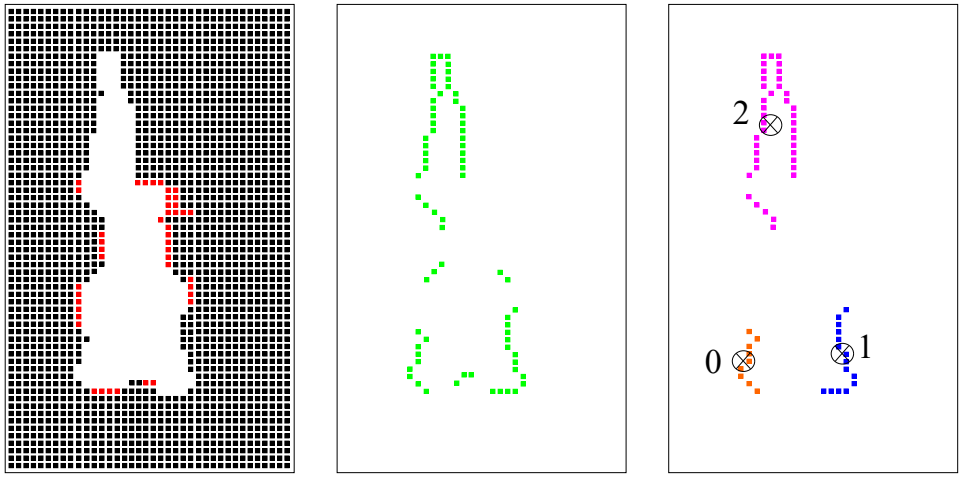
$$S[c_i] = \begin{cases} \text{free} & \text{if } P[occ(c_i)] < (p_{prior} - \delta) \\ \text{unknown} & \text{if } (p_{prior} - \delta) \leq P[occ(c_i)] \leq (p_{prior} + \delta) \\ \text{occupied} & \text{if } P[occ(c_i)] > (p_{prior} + \delta) \end{cases} \tag{3.2}$$

where δ is the range of unknown cell probability.

3.1.2 Detecting Frontiers

As mentioned earlier in section 3.1, frontier cells are subset of free cells that are adjacent to unknown cells. Potential target locations to explore unknown areas can be selected by robots from this subset. To illustrate the process of detecting frontier cells in a two-dimensional occupancy grid, consider an example from [132] as shown in figure 3.4. Figure 3.4(a) shows an occupancy grid map of a hallway. White area indicates free cells, red dot points indicate occupied cells and thin black dot points represent unknown cells. Figure 3.4(b)

shows extracted frontier cells in the grid that lie between the boundary of free space and unknown space. In many applications further filtering is needed to consider only the feasible frontiers. We use the clustering algorithm from [132] which groups contiguous frontiers as one frontier region. Any region that has size bigger than the robot size is considered as feasible. Then, the centroid frontier of the feasible region is calculated. Figure 3.4(c) illustrates the clustering of frontiers, where crosshairs denote region centroids. In this specific instance, all regions are feasible since the size of the regions is bigger than the size of the robot.



(a) (b) (c)

Figure 3.4: Example of frontiers detection process [132]: (a) A given occupancy grid map consists of free cells (white area), occupied cells (red dot points) and unknown cells (black dot area). (b) Frontier cells in the corresponding map are indicated as green dot points. (c) Frontier cells are clustered into three regions with crosshairs denote region centroids.

3.1.3 Assigning Frontiers to Robots

The task of assigning frontiers to robots is called the goal-assignment problem. The objective of this task is to assign the best possible target location for each available robot automatically, given a set of potential frontiers. By assigning appropriate target locations, the exploration performance is expected to increase. There are few steps to be taken to perform the goal-assignment task.

In this thesis, we deal with both single-robot and multi-robot goal-assignment tasks. Thus, we illustrate two variants of goal-assignment to accommodate both robotic systems.

Single-robot goal-assignment - General steps of goal-assignment for a single-robot system are as follows. Once potential accessible unvisited frontiers have been detected, each of them will be evaluated in terms of accessibility from the current position of a robot. This evaluation is necessary to estimate the effort needed by the robot to reach a frontier. In this thesis, we deploy a simple cost path evaluation to measure the relative distance between frontiers and the robot. A path planner is used to find the estimated cost of an obstacle-free path from the robot's current pose to the cell containing the frontier. The robot attempts to navigate to the nearest accessible, unvisited frontier according to the cost path value [130]. Various types of cost path calculation can be applied. We choose Dijkstra's algorithm [26] applied to a generalised Voronoi graph [20] to calculate the cost path in our experiments. Dijkstra's algorithm is chosen because it can perform a complete cost exploration on a cost map rather than approximate exploration like A* algorithm [53]. Even though such exploration is slower for single destination search, it is more effective in the case of calculating cost path for multiple frontier destinations as in our case. By comparing cost path of all frontiers, the frontier with the minimum cost path is assigned as the target location of the robot.

Multi-robot goal-assignment - For a multi-robot exploration system, the goal-assignment task is more complex than for a single-robot system. This is due to the coordination issue in which a robot team must be coordinated accordingly to avoid two or more robots approaching the same frontier at a time. Where communication and computational resources permit the centralised coordination paradigm has advantages over decentralised coordination. One of the advantages is that the centralised coordination can efficiently distribute robots to different unexplored areas to avoid task redundancy [118]. A number of centralised coordination techniques have been proposed by researchers in past literature including the Bidding strategy [82], Maximum-Profit strategy [115], Market Economy approach [136] and the Cost-Utility approach [14, 118].

In this thesis, we apply the Cost-Utility approach [118] with minor variations. The Cost-Utility approach can fully utilise information available in a centralised coordination paradigm. Not only it can avoid one frontier from becoming a target of more than one robot, it also can avoid nearby frontiers of a targeted frontier to be chosen by another robot. Thus, it provide good spread of robots over an exploration area. We outline this approach next. In brief, each candidate frontier is assigned with a value calculated from its corresponding cost and utility values. The next goal location is chosen based on the highest net cost-utility value.

Cost - For each robot R, the cost path for reaching each frontier F is calculated. Consider a two-dimensional cost map $M(\mathbf{c})$ of an occupancy grid map $O(\mathbf{c})$. $M(\mathbf{c})$ holds the cost values of every grid cell, $V[c_i]$. Initially, the cost of grid cells, $V[c_i]$ that contain the robot position are set to zero, meanwhile, the cost of all other cells within the map's boundaries are is set to infinity.

$$V[c_i] \leftarrow \begin{cases} 0 & \text{if } c_i \text{ is robot pose} \\ \infty & \text{otherwise} \end{cases} \tag{3.3}$$

Subsequently, all free cells are iteratively updated until convergence is achieved. The cost of a free cell is updated by looking at the cost value of all neighbour cells, $V[c_n]$. The final cost of a free cell is set according to the value of the lowest cost neighbour, plus the distance to this neighbour. Eventually, the cost of each frontier F from robot R, $V[c_F|c_R]$ can be extracted directly from the cost map, $M(\mathbf{c})$. The two-dimensional cost map element, $V[c_i]$ of a free cell can be updated using the following equation 3.4:

$$V[c_i] \leftarrow \min \left\{ V[c_n] + \sqrt{\triangle x^2 + \triangle y^2} \mid \triangle x, \triangle y \in -1, 0, 1 \right\} \tag{3.4}$$

where $\triangle x$ and $\triangle y$ are distance to neighbour cells, c_n in x-axis and y-axis, respectively.

Utility - The utility of frontiers is calculated to predict how much new information can be gathered when moving to a specific frontier. A precise prediction of actual utility depends on the particular structure of the exploration environment and sensor coverage. However, as the structure of the unknown exploration environment is arbitrary, the utility of a frontier is calculated by considering only the sensor coverage element. Theoretically, when a robot has reached a frontier, the utility of the frontier and its surrounding frontiers within sensor range is reduced. Consequently, a frontier with a lower utility value is unlikely to be chosen by other robot as target location. An algorithm as described in [118] to calculate the utility of a frontier U_F is applied. Initially, the probability $p_{vis}(F, F')$ that the proximity sensor covers frontier F' if the robot reaches frontier F is dynamically set using equation 3.5.

$$p_{vis}(F, F') = \begin{cases} 1.0 - \frac{\|F-F'\|}{max\_range} & \text{if } \| F - F' \| < max\_range \\ 0.0 & \text{otherwise} \end{cases} \tag{3.5}$$

where max_range is the maximum range of proximity sensor.

The U_F of all frontiers is initialised to 1.0. Whenever a frontier is assigned to a robot, the U_F of all frontiers are updated accordingly. Given frontiers $F_1, ..., F_{m-1}$ have been assigned to the robots $1, ..., m-1$, the utility of any frontier cell m, U_{F_m} can be updated using equation 3.6:

$$U(F_m|F_1, ..., F_{m-1}) = U_{F_m} - \sum_{i=1}^{m-1} p_{vis}(F_m, F_i) \tag{3.6}$$

In the above equation, U_{F_m} will only be reduced if the location of F_m is within the coverage of sensor range at the assigned frontier locations. Otherwise, the utility remains unchanged to increase the probability of F_m to be chosen as the target location of an unassigned robot.

Decision-theoretic goal-assignment - To define the best possible target location of a robot, we need to account for both the cost $V[c_F|c_R]$ and the utility U_F. This trade-off is necessary to ensure each robot moves to a frontier that has minimal cost but has optimal information gain in a coordinated manner. An algorithm to assign frontiers to robots using the Cost-Utility approach is shown in algorithm 5:

Algorithm 5: Goal-assignment using Cost-Utility approach.

1 For each robot R, compute the cost, $V[c_F|c_R]$ for reaching each frontier F.
2 Set the utility U_F of all frontiers to 1.0.
3 **while** *there is at least one unassigned robot* **do**
4 Determine a robot R and a frontier F pair which satisfy:
5 $(R, F) = argmax_{(R',F')}(U_{F'} - \beta V[c'_F|c'_R])$
6 where $\beta \geq 0$ dictates the relative importance of utility versus cost.
7 Reduce the utility of each frontier conditionally using equation 3.6.

3.2 Navigating Robots to Frontiers using Motion Control

The main objective of the motion control module of a Frontier-based exploration strategy is to navigate each available robot optimally from its current pose to its target pose (an assigned frontier) while avoiding specific constraints. Navigation is achieved through a sequence of actions where the control cycle is determined by the control frequency. In each cycle, a decision to select an appropriate action is performed based on the current state of the robot.

In such cases where the state is approximate, developing a motion controller for an exploration strategy is challenging due to the fact that the environment is partially and incrementally observable through out the exploration horizon. This section describes a decision-theoretic based motion controller (DTC) to deal with such an environment. A DTC has several advantages:

1. it provides a formal framework for controllers working under stochastic and partially observable environments. This framework utilises the concept of Partially Observable Markov Decision Model (POMDM) [17] in which the decision of action is processed by using current state only and the state is approximate from observation via robot's sensors.

2. provides a formal method to develop a policy (decision model) to map states to actions. This method is important to cater for non-trivial trade-off process in high dimensional state spaces.

Our main focus in this thesis is to find the best policy of a DTC given specific exploration requirements. In the following sub-sections, we first describe the generalised framework of a DTC. Within this framework, we focus on information fusion and policy. Then, we provide theoretical explanation on decision theory in the context of motion control optimisation before proceeding to the next section describing evolution of the policy of a DTC.

3.2.1 General Framework of DTC

A DTC must be designed to conform with exploration requirements. Sometimes, the requirements change drastically. For example, an existing DTC of an exploration strategy might need to integrate a power management module to save battery usage in a power-limited exploration mission. In such a case, the controller must be allowed to acquire new data such as battery level to balance trade-off between changes in robot's direction and power usage. In another example, an existing DTC might need extreme alteration with changes in robot's sensors and body, which changes the robot's kinematics in turn. These situations require some parts of the existing controller to be tuned significantly. Hence, a DTC has an advantage in which it has the flexibility to adapt its policy to choose correct actions given new exploration requirements. The general framework of a DTC for each control cycle is illustrated in figure 3.5.

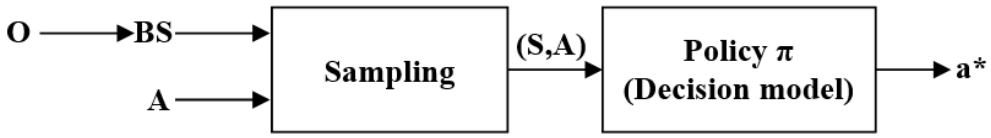


Figure 3.5: General framework of a DTC. O is a finite set of sensor observations. O is used to generate a finite set of robot belief states BS. A is a finite set of possible action that can be performed by the robot. (S, A) is a set of navigational options A sampled at their corresponding sampled states S. π is the policy that determines the best navigational option a^* to choose from (S, A).

Formally, a DTC receives three inputs. O is a finite set of observations where partial environmental information is received at the time of sensory data acquisition. The observation includes the robot and its target poses. A finite set of states, BS represents belief states of an environment built upon observation O. Meanwhile, A is a finite set of possible actions that can be performed by the robot. This framework takes BS and A as inputs. The first stage is to sample the corresponding state $s \in S$ of each possible action $a \in A$ to produce a set of navigational options (S, A) by using the inputs. Given these navigational options, the next stage is to find the best navigational option by using a policy (decision model) π. The output of π is the best action, a^* such that after performing a^*, the exploration progress is expected to increase significantly. Using this framework, the overall exploration performance is determined by a sequence of actions selected by π at each iteration.

3.2.2 Decision Theory in DTC

In further detail, consider (S, A) in figure 3.5 with sets of size J. (S, A) is a list of J navigational options where each option contains a possible action characterised by its corresponding states. Assume $(s, a_j, s_j^{'}) \in (S, A)$ is a navigational option, where s is the current state of the robot, $a_j, j = \{1, .., J\}$ is the action of option j and $s_j^{'}$ is the predicted state after performing action a_j. The core problem of a DTC is to find π such that the optimal option can be selected by scoring a_j given s and $s_j^{'}$. Given that A is finite, then the best

action $a^* \in A$ is defined as:

$$a^* = \arg\max_{a_j \to A} \left\{ \pi(s, a_j, s_j^{'}) \right\} \tag{3.7}$$

Note that π in the above formula acts as a reward function that considers a_j with the maximum score as the best action. Note that, if a cost function is preferred, then the best action selection can be simply changed to the minimum score of a_j.

The formulation of equation 3.7 has π as its core component. Building π for a complex system with non-linear response of S is a non-trivial process due to several factors:

1. Large state spaces. There are two reasons for having large S space. First, high dimensionality of S due to various types of observation. Each dimension of S known as state signal ss perceives particular information in regards to robot's environment. Second, the range value of each ss. Each dimension of S can hold different range of values to cover all possible observations that may occur in the system. As such, consider discrete S space with N state signal where each $ss_n, n = \{1, .., N\}$ has range of A_{ss_n}. Hence, the size of S space can be calculated as:

$$\text{Size of } S = \prod_{n=1}^{N} A_{ss_n} \tag{3.8}$$

2. Environmental factors. π is difficult to express as a valid function due to the difficulty to find relationship between all ss in S. The exact form taken by π depends on many factors including: the nature of an environment; the robot hardware and sensors; computer processing resources; the algorithms used to extract state signals; and the way in which navigational options are specified [61]. In addition, the effect of uncertainty due to sensor noise makes the process of building π harder.

In this thesis, we examine π for various exploration configurations. We evolve π using EC technique such that π has more freedom to adapt to its context. To achieve that, we allow the evolution of π to occur at several levels including evolving constants, unary function of ss and structure of π function. As the complexity of exploration requirements grows and the space of S increases, the ability to evolve π is a significant step to eliminate bias arising from predetermined structure.

3.3 Using Grammatical Evolution to Optimise Motion Control

EC search methods can be implemented as optimisers by producing a set of solutions iteratively until an optimal, or at least feasible solution is found [97]. To date, efforts to integrate EC techniques to the field of robotics are well established in the form of the sub-field of Evolutionary Robotics (ER). ER aims at fulfil the requirements of creating autonomous and adaptive robots so that the robots can perform their tasks constructively and cope with dynamic environments efficiently. This can be done by adopting the concept of natural biological evolution to optimise aspects of an autonomous robot including its motion control. Algorithm 6 [99] presents a general algorithm for evolving a robot controller.

Algorithm 6 has been widely applied to various type of robot controllers including Neural Network controllers [69, 39], Fuzzy Logic controllers [28, 56, 124] and reactive controllers [107, 100]. Each evolutionary set up is different in terms of its configuration according to the application of the controller to a specific robotic task. In the particular case of evolving π of a DTC as mentioned in sub-section 3.2.2, the success of evolution depends on many factors:

1. *Representation of chromosomes* - One vital task when applying an EC technique is to encode a real-world problem into a manipulable representation known as chromosome. Chromosome enables an EC

Algorithm 6: General description of algorithm for evolving a robot controller.

1. A population of chromosomes, encoding a set of candidate controllers are randomly generated.
2. Each robot, provided with a candidate controller, is allowed to act in an environment (physical or simulated) to accomplish a task.
3. The performance of each candidate controller is evaluated automatically after it has accomplished a given task.
4. A number of good controllers are selected to reproduce by creating copies of their genotypes with some modification using genetic operators (e.g. crossover, mutation).
5. This process is repeated from step 2 for a number of generations or until the performance criterion is satisfied.

technique to search for good solutions within search space boundaries. Constraining a chromosome to produce only feasible solutions can be beneficial to evolution performance. However, producing an effective representation of a chromosome is non-trivial for evolving the π function of a DTC. This is due to the fact that π take on a wide variety of forms of mathematical function. Such freedom of form π gives roboticists the flexibility to change π structure according to the ultimate needs of an exploration strategy. Because of this, limiting the search space of a π may hinder EC from finding a good solution that remains outside the search space. On the other hand, expanding the search space may reduce the speed of convergence towards a good solution. In order to design a good representation of chromosomes of π, one needs to answer the following questions first: How many state signals are received by π? What is the range of values of each state signal? Is the relationship between each state signal well-defined? The last question is important because it helps the designer to determine the level of evolution needed by a π. The clearer the relationships, the fewer parts of π are required to be evolved. Unfortunately, such relationships are typically vague a-priori especially when new state signals are introduced. Our aim is to come up with a method that can reduce the amount of evolution needed by a π such that given any initial condition of state signals relationship, π can accurately estimate the right relationship between each state signal.

2. *Learning environment* - Once potential solutions are generated, they need to be evaluated in order to know how good they are. In the case of exploration tasks, the robot must be tested to explore initially unknown environments within a certain period of time. One of the challenges in designing the learning environment is to minimise the time taken to evaluate an individual. An autonomous exploration task is typically a long running compared to other robotic tasks. Evolution with an EC technique can become infeasible if the time taken to evaluate each solution is too long. Therefore, it is important to find a way to reduce learning time without losing vital learning information needed to measure solution performance. Our approach in this thesis is to implement simulation-based learning environments in which learning time can be reduced by speeding up the simulation several times faster than the real-time. We carefully design the simulation by taking into account the physics of the robot model to mimic the real-world environment as close as possible.

3. *Performance indicator* - A quantitative measurement of a potential solution is needed to know how *fit* the solution is compared to others. In EC field, this measurement is known as fitness. The fitness, calculated with a fitness function, measures the quality of a solution after performing its task. Specifically, for an exploration task, an essential fitness function is designed to estimate the area that can be explored by the robot in the learning environment. The real value or relative value of explored area can be used to form a single fitness value. Where there are multiple objectives - for example, safety, area-explored and battery conservation - a way needs to be found to feed these into an evaluation of fitness. The two main options are to 1) combine the multiple factors mathematically into a single value and 2) to keep the values separate and rank individuals according to pareto dominance. Option 1 is usually simpler and applies well when the goals are not in conflict. Option 2 is most applicable when there is a trade-off between goals. In this work, we use option 1 i.e. we combine metrics into a single evaluative function. This is possible in the case where one factor dominates another factor. In our case, the most dominant factor is the

exploration area (embedded with collision status penalty), in which for all cases, a candidate solution with higher exploration coverage is always better than a candidate solution with lower exploration coverage.

4. *Genetic operators* - Genetic operators are responsible for producing better solutions as the EC progresses. For GA, the common operators are crossover and mutation. Both operators change some segments in selected chromosomes accordingly. Crossover performs changes by swapping corresponding segments of two relatively good solutions. Then, the mutation operator is used to flip some segment values according to a probability supplied by the user. While common operators are adequate to solve many GA-based EC problems, in certain cases, user-defined operators are sometimes necessary to focus the search process. These user-defined operators are strongly related to the representation of chromosomes. Either common operators or user-defined operators or combination of both have advantages. Our EC design prefers to use common genetic operators that have loose-coupling with chromosome representations such that it can accommodate various configurations of π evolution without requiring user-defined operators.

Based on above factors, we propose GE, an EC technique first described in [103] to evolve the π function of a DTC. GE has significant advantages for the optimisation of a DTC. GE has genetic properties as follows:-

1. *Chromosome* - A chromosome in GE is represented as a variable-length binary string. This single generic representation of chromosome tallies with our aim to minimise the design effort for the selection of appropriate chromosome representation. This is the key feature of GE in which it separates search space and solution space using a Backus-Naur Form (BNF) grammar. GE is able to make the representation of chromosome fixed since the chromosome does not directly encode the candidate solution. Instead, the chromosome acts as a selector string that maps a BNF grammar to a syntactically correct solution. In other word, GE avoids the requirement to modify the representation of chromosomes in the event of Problem-specification changing.

2. *Genetic operators* - Again, the representation of chromosome influences the selection of genetic operators. As such, in GE, genetic operators can remain static as only one representation of chromosome is used for various exploration configurations.

Algorithm 7 specifies steps to evolve the π function of a DTC using GE.

Algorithm 7: Algorithm for evolving π of a DTC using GE.

1 A BNF grammar is developed describing the production rules for π.
2 A population of chromosomes, encoding binary strings to map a BNF grammar to a candidate π, are randomly generated.
3 Each robot, provided with a candidate π, is allowed to execute simulated exploration tasks within a specific period of time.
4 Performance of each candidate π is evaluated automatically after each task execution based on exploration properties.
5 A set of good π are selected to reproduce by creating copies of their genotypes with some modification using standard genetic operators (i.e. crossover and mutation).
6 Steps above are repeated for a number of generations or until the performance criterion is satisfied.

Graphically, the framework of GE for π evolution is illustrated in figure 3.6. Mainly, the framework consists of 4 components: i) Search engine, ii) Language-specification, iii) Problem-specification, and iv) GE mapper. Search engine is the component that performs search process within a bounded search space. A population of chromosomes is generated in this search engine. Throughout this thesis, we adopt GA as the search engine. The search engine outputs chromosomes (genotypes) to the GE mapper. The second input to the GE mapper is a BNF grammar describing the set of valid forms for π. GE uses the values in the chromosome to traverse the provided grammar to build a syntactically correct π function. It should be noted that, during an evolutionary run the BNF grammar is fixed. It is the chromosomes in the population that provide for variations in the form of π. This π is also known as phenotype in GE terms. The GE mapper then provides phenotypes to the Problem-specification component. Problem-specification is the component where users can specify the execution of the task to be solved. In our case the Problem-specification as an autonomous exploration

task. Once a π is supplied, the robot will be ordered to perform autonomous exploration tasks within a specific time-frame. At the end of each execution, the Problem-specification component evaluates the performance of the supplied π and sends the evaluation result in the form of a fitness value to the GE mapper. Afterward, GE mapper passes the fitness value to the Search engine. The search engine performs reproduction of chromosomes using genetic operators in each generation. The process above iterates over generations until either a solution meets the required criteria or a pre-determined number of generations or evaluations is performed. In the following, each of these components specialised to our works on evolving π of a DTC will be discussed in detail. For further reading on the GE algorithm, readers can refer to work by [103].

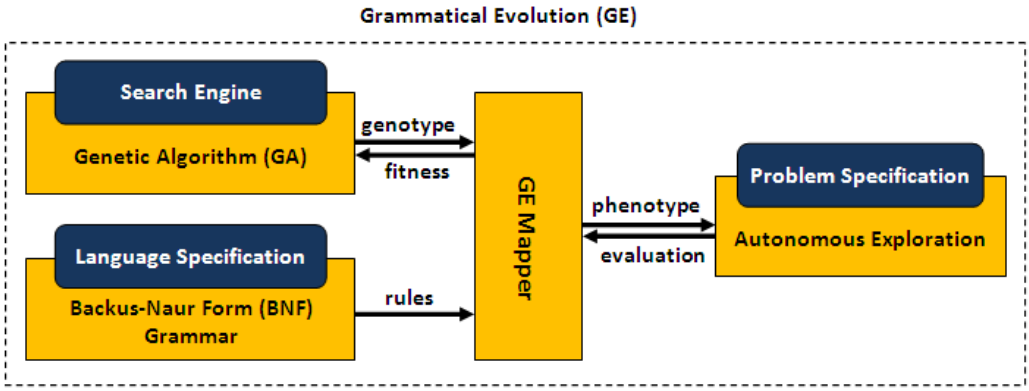


Figure 3.6: The framework of GE consists of 4 main components. *Search engine* GA is used to perform the search process within a bounded search space. It provides a set of genotypes to the *GE mapper*, and later receives the corresponding fitness of each genotype. A *Language-specification* component supplies a BNF grammar to the *GE mapper*. The *GE mapper* uses production rules in the grammar to convert genotypes into phenotypes. A phenotype represents a valid form of π. The *GE mapper* then provides a phenotype to the *Problem-specification* component such that a candidate π can be integrated into the robot system. The *Problem specification* executes and evaluates the fitness of the robot that performs autonomous exploration tasks within a specific time-frame. Finally, the fitness is sent to *Search engine* via *GE mapper* such that the evolution can be continued.

3.3.1 Genetic Algorithms as the Search Engine

In this sub-section, we describe the configuration of a GA implemented specifically to evolve the π function of a DTC. The GA used in this thesis is a variation of the standard GA as described in [44]. Our GA uses variable-length chromosomes rather than fixed-length chromosomes. Readers can refer to [44] for a detailed explanation on the standard GA we use. Since the chromosome acts as the selector that maps a BNF grammar to a syntactically correct solution, the number of genes (binary numbers) needed to perform the mapping process is not known beforehand. Because of this, chromosomes of a variety of lengths are provided in the initial population. Chromosomes that are to short to encode a valid individual can either be allocated minimum fitness or we can configure the encoding process to wrap-around to the start of the chromosome.

In this thesis, we use different methods for selection and crossover operators from the standard GA [44] discussed in chapter 2. The detail of these methods is explained next.

Selection - The one we adopt in this work is tournament-based selection [45]. Essentially, tournament selection is chosen over fitness proportional selection because it maintains stronger selection pressure over the entire search process [87]. We choose a moderate tournament size because we had no a-priori model predicting the best tournament size for this application. Even though there is no analytical study being done in this thesis regarding the tournament size, it will be useful for future investigation on determining the number of generations needed to evolve this robotic problem. In addition, tournament selection has the advantage of being robust in the presence of a noisy fitness function [87]. This is very important in the case of evolving the π function of a DTC for an exploration task since the task itself is noisy in nature. Tournament selection requires several *tournaments* to be accomplished among k chromosomes. The number k, defined by the users, determines the tournament size. In this work, we set k to 2 which produces a binary tournament selection. In each *tournament*, k chromosomes are chosen randomly from the current population

to form a subset group. From this subset group, the best chromosome with the highest fitness among them is selected as a parent and is stored in a mating pool with size $mpool_{size}$. *Tournament* is repeated until the mating pool is filled.

Crossover - In this thesis, we utilise *effective* crossover [98]. With the standard one-point crossover, there is probability that offspring will produce the same solution as their parents. This is because there is no mechanism to identify whether an array of genes of parents being exchanged has an effect to the phenotype to genotype mapping process. However, *effective* crossover can eliminate this problem by controlling the range of crossover point to be selected within the selective part of genes only. These genes are the bits that are used to select production rules during the phenotype to genotype mapping. In other word, effective crossover performs gene exchange on the effective bits of chromosomes only. The *effective* bits are determined whenever a chromosome is used to map a BNF grammar to produce a solution, GE will store the number of genes being utilised to select appropriate production rules. This number of genes defines the *effective* bits. By selecting exchange points within the *effective* bits, the crossover operation remains relevant to the mapping process used by GE mapper. Algorithm 8 defines the process for using *effective* crossover. Figure 3.7 illustrates graphically algorithm 8 with two instance parents.

Mutation - For mutation, we apply the standard point mutation method as explained in section 2.4.1. Algorithm 9 explains the point mutation of a chromosome used in this thesis.

Algorithm 8: Effective crossover operation.

Get two chromosomes (parents) from a mating pool, A and B.
Get total size of both parents, N_A and N_B, respectively.
Get effective size of both parents, E_A and E_B, respectively.
Get a crossover point, p_A from parent A:
if $E_A > N_A$ **then**
    p_A = random position between 0 and N_A.
else
    p_A = random position between 0 and E_A.
Get a crossover point, p_B from parent B:
if $E_B > N_B$ **then**
    p_B = random position between 0 and N_B.
else
    p_B = random position between 0 and E_B.
Copy corresponding bit array according to the generated crossover points:
    x_1 = bit array of parent A from position 0 to p_A.
    x_2 = bit array of parent A from position $p_A + 1$ to N_A.
    y_1 = bit array of parent B from position 0 to p_B.
    y_2 = bit array of parent B from position $p_B + 1$ to N_B.
Generate new chromosomes (offspring), C and D:
    C = concatenation of x_1 and y_2.
    D = concatenation of y_1 and x_2.
Repeat the above processes until there are no more parents in the mating pool.

Algorithm 9: Point mutation operation.

Data: probability of mutation, ρ_{mut}.
Initialise current bit position, $i = 0$.
Get number of bits, B in a given chromosome.
while $i < B$ **do**
    Get a random number, r between 0-1.
    if $r < \rho_{mut}$ **then**
        Flip bit value at position i.
    else
        Maintain bit value at position i.
    i++.

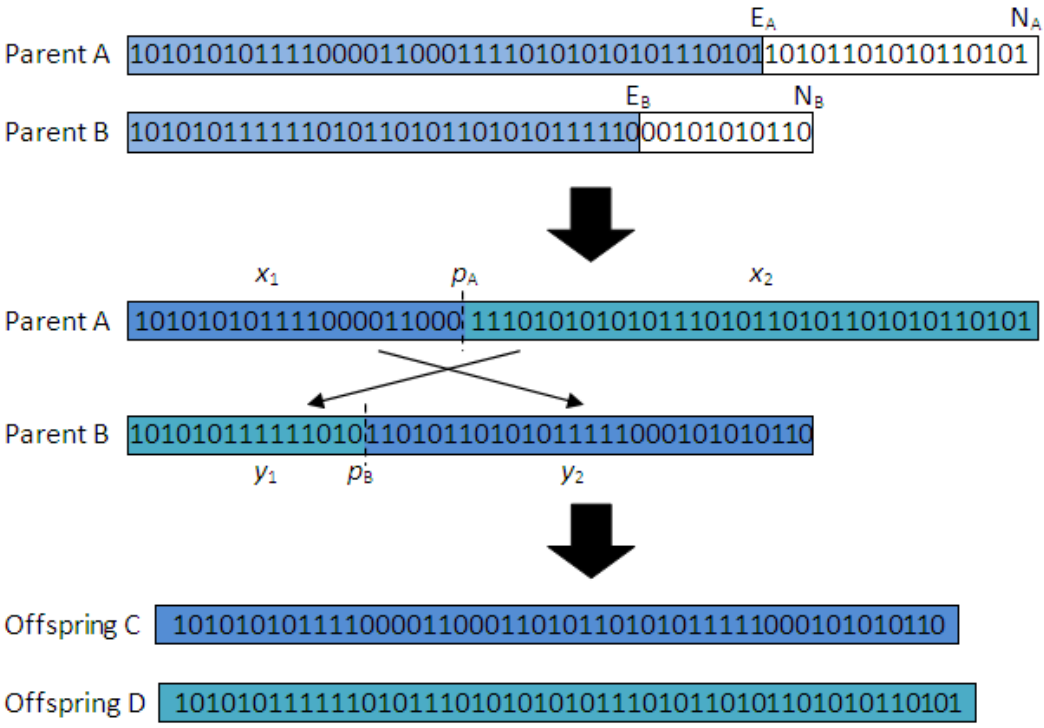


Figure 3.7: An example of the *effective* crossover operation. Parent A and B have the total size of N_A and N_B bits, respectively. The first E_A and E_B bits are used by parent A and B to map a genotype to a phenotype (π), respectively. Crossover points p_A and p_B of parent A and B are randomly determined between bit 0 to bit E_A and E_B, respectively. x_1 and y_1 represent bit arrays of parent A and B from position 0 to p_A and p_B respectively. While x_2 and y_2 represents the rest. The crossover operation is performed such that two offspring C and D are generated. Offspring C concatenates bit array x_1 and y_2, while offspring D concatenates bit array y_1 and x_2.

3.3.2 Formulating the Language-specification

A Language-specification defines the possible choices of solution. We employ a user-supplied BNF grammar as the standard Language-specification [103]. Conceptually, BNF grammars consist of four main elements: {T,N,P,S}. T is a set of terminals. A terminal is a variable that can appear in the final computer program. A terminal can be written in any form such as integer, string or arithmetic operator that is syntactically correct for a specified computer program. N is a set of non-terminals. A non-terminal is a symbol that can be

expanded into one or more non-terminals and terminals. It does not appear in the final computer program. A non-terminal is represented with a name enclosed by the pair < >, e.g. <symbol>. P is a set of production rules. A production rule converts a non-terminal to a terminal or another non-terminal. The generic form of a rule can be written as follows:

$$<symbol> ::= choice_1 \mid choice_2 \mid ... \mid choice_M \tag{3.9}$$

In equation 3.9 above, <*symbol*> on the left hand side (LHS) of the equation is always a non-terminal item. ::= indicates that <*symbol*> must be replaced by an expression from a set of choices ($choice_1$, ...) on the right hand side (RHS). A $choice_M$ can be a terminal or a non-terminal. M defines the number of choices delimited by the '|' symbol.

The final element S represents a start symbol. S is a member of N. It is the designated entry point to execute production rules in a BNF grammar.

For better understanding, consider a simple example of a BNF grammar in figure 3.8 that is used to produce a mathematical expression:

(R1)	<expr> ::=	<var><op><var>	(C0)
		| <expr><op><var>	(C1)
		| <var>	(C2)
(R2)	<var> ::=	x | 1 | 2 | 3 | 4	
(R3)	<op> ::=	+ | - | / | *	

Figure 3.8: An example of production rules in a BNF grammar to generate mathematical expressions. It consists of three production rules P = {R1, R2, R3}, three non-terminals N = {expr, var, op}, nine terminals T = {+,-,/,*,x,1,2,3,4} and a sole start symbol S = <expr>.

There are three production rules, three non-terminals and nine terminals in the grammar with <expr> as the start symbol. The BNF grammar elements of the above example can be described as:

P = {R1, R2, R3}
N = {expr, var, op}
T = {+,-,/,*,x,1,2,3,4}
S = <expr>

There are three main rules in the grammar. Each rule describes the possible production of a non-terminal, respectively. Rule 1 (R1) provides three choices for non-terminal <expr>. For choice 1 (C1), <expr> is expanded into other non-terminals namely <var> and <op>. <var> and <op> are described by rule 2 and rule 3, respectively. Rule 2 (R2) substitutes <var> with one of terminals described on the RHS of the rule. In this case, there are five choices to be selected from a set of options: x, 1, 2, 3 or 4. Meanwhile, Rule 3 (R3) replaces <op> with one of four basic mathematical operators i.e. +, -, / or *. An example of the final form of a possible mathematical equation produced by C1 is x + 4. Choice 2 (C2) and choice 3 (C3) of rule 1 can generate other mathematical equations with the same process as above.

In this thesis, we design a BNF grammar such that a set of valid π for a DTC can be generated. In particular, we develop a BNF grammar to construct a π written in a C++ language-based program. We carefully design the grammar for each specific exploration problem and show these in detail in the remaining chapters describing the experiments we have conducted.

3.3.3 Formulating the Problem-specification

The function of the Problem-specification component is three-fold: i) to evaluate a given phenotype using a user-defined evaluation method, ii) to convert the evaluation result in the form of Search engine's fitness value using an objective function, and iii) to return a fitness value to the GE mapper. Each problem domain has a unique evaluation method and objective function. Thus, both evaluation method and objective function must be designed accordingly by users.

In the domain of an autonomous exploration tasks, the evaluation method must be designed such that information regarding the exploration performance can be acquired as accurately as possible. A robot, provided with a candidate π, must be permitted to perform exploration in unknown environments within a specific time-frame. An objective function must be formulated to reward a π that allows fast exploration and penalise behaviours that slow down the exploration such as collision. In addition, the objective function must also consider other exploration requirements such as power consumption, robot distribution or communication strength if specified by users.

In this thesis, the evaluation method is executed in simulated environments due to several factors described earlier in section 3.3. Simulated maps defining areas to be explored by robots are generated by users. Robots are required to explore the area by which the navigation is controlled solely by the given DTC. At the end of the simulation, exploration performance is evaluated and the fitness value is calculated. An algorithm to evaluate a candidate π performing autonomous exploration task is outlined in Algorithm 10. Note that, this algorithm will have minor variations according to specific exploration task settings. We explain these variations in the remaining chapters describing our experiments.

3.3.4 GE Mapper

The GE mapper interacts with all GE components to receive and send appropriate input and output data, respectively. First, the GE mapper takes a genotype from the Search engine and a BNF grammar consisting of production rules from the Language-specification. The GE mapper generates a phenotype (candidate solution) by mapping the genotype with production rules. The phenotype is then provided to the Problem-specification to perform the evaluation process of the phenotype. In return, the GE mapper receives a fitness value representing the evaluation performance. The GE mapper sends the fitness value to the Search engine for further evolutionary processing. In this sub-section, we explain the mapping process to produce a phenotype given

Algorithm 10: Performance evaluation of a candidate π using simulated exploration settings.

Data: a phenotype represents a π and an objective function, obj_{func}.

```
Embed π into the C++ code of our exploration strategy.
Re-compile the code.
Initialise number of environments to be explored, M.
Initialise simulation time-frame for each environment, t_M.
Initialise simulation termination condition, C.
m = 1.
while m ≤ M do
    Set robot(s) initial position in environment m.
    while C not met do
        Perform exploration task on m and build map within t_M.
        Evaluate exploration performance periodically.
    Store the performance of the exploration on environment m: P_m.
Aggregate all P_m to obtain the final exploration performance, P_π.
Calculate the corresponding fitness value f_π of P_π using obj_func.
Return f_π to GE mapper.
```

a genotype and a BNF grammar.

As mentioned previously, the genotype of our GE is in the form of binary string. It acts as the selector of choices in production rules of a BNF grammar. The first step to generate a phenotype is to convert the binary string genotype into a decimal string representation. Following a method from [103], the binary string genotype is divided into a sequence of 8 bit binary numbers called codons. Each codon is then converted into a decimal number called a codon integer by using standard binary-to-decimal conversion. Figure 3.9 shows an example of a binary string genotype to a decimal string genotype conversion.

Once a set of codon integers is gathered, it can be used to select appropriate choices in production rules. All selections are performed using a mapping function as in equation 3.10.

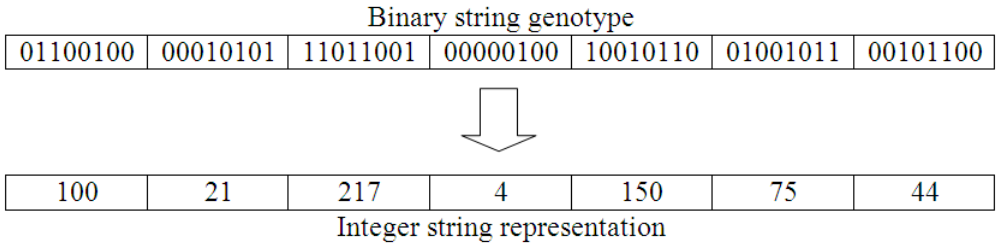


Figure 3.9: An example of conversion from a 56-bits binary string genotype to a 7 decimal string genotype.

$$choice = \frac{\text{[codon integer]}}{MOD} \quad \text{[number of choices in a production rule]} \tag{3.10}$$

For example, consider a codon integer 8 and the production rule R1 from figure 3.8. Rule R1 has three choices to be selected: $C0, C1, C2$. Applying equation 3.10, 8 MOD 3 = 2. Hence, choice 2 (C2) is selected, thus, <expr> is substituted with non-terminal <var>.

The selection of productions is an iterative process. The process starts by using the first codon integer to select a choice from a start symbol (S) production rule. Next, the second codon integer is used to perform selection of another production rule resulting from the first selection. Then, a subsequent codon integer is applied to make another selection of a production rule determined from the previous selection. The selection process is repeated until all relevant non-terminals are converted to terminals. In the case where codon integers are insufficient to perform the selection process e.g. a genotype with a small size of binary string, a wrapping technique as explained in [103] that re-uses existing codons can be implemented. This wrapping technique can be used to prevent GE from producing an invalid solution.

To illustrate the whole selection process using GE mapper, consider example from figure 3.10. This time, we show how a mathematical expression can be produced by mapping the genotype of figure 3.9 with production rules from figure 3.8.

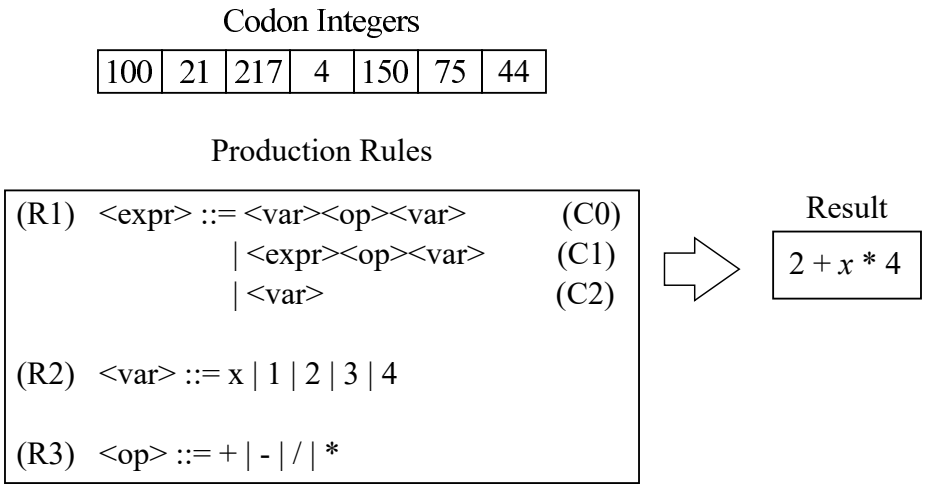


Figure 3.10: An example of mapping codon integers with production rules into a valid mathematical expression. Each codon integer is used to substitute a non-terminal of a production rule accordingly. This substitution process is performed in sequence. The final result of the substitution is a set of terminals representing a valid mathematical function.

First, GE mapper reads the first codon integer 100. This codon integer will be used to select a choice from the start symbol <expr> of production rule R1. Applying equation 3.10, <expr>: 100 MOD 3 = 1, thus C1 is chosen to substitute <expr>. The current form of the result is:

$$< expr >< op >< var > \tag{3.11}$$

Note that, we always start the substitution process from the leftmost non-terminals. In the current form, codon integer 21 is read by GE mapper to determine the value of <expr> in equation 3.11. Thus, <expr>: 21 MOD 3 = 0 and C0 is selected. The result now is expanded as:

$$< var >< op >< var >< op >< var > \tag{3.12}$$

At this step, we then use the third codon integer 217 to select a substitution for the leftmost <var>. There are 5 choices for <var> non-terminal as described

in production rule R2. Thus, 217 MOD 5 = 2, which replaces <var> with the integer value 2. The current result now contains one terminal item:

$$2 < op >< var >< op >< var > \tag{3.13}$$

Subsequently, codon integer 4 is read to substitute the leftmost <op> where 4 choices are given as in production rule R3. 4 MOD 4 = 0, thus <op> = +. This updates the result to:

$$2+ < var >< op >< var > \tag{3.14}$$

The remaining non-terminals are continually substituted by using the remaining set of codon integers. Next codon integer 150 determines <var> as x in which 150 MOD 5 = 0. <op> is replaced by the operator $*$ by using codon integer 75 and the last <var> is substituted as 4 as the result of performing 44 MOD 4 = 4. Consequently, the final result is a valid mathematical equation:

$$2 + x * 4 \tag{3.15}$$

In this example, all codon integers are used for mapping genotypes using production rules. Thus, the effective size, E of the genotype is equal to the total size N of the genotype. In the case where N is longer than E, then the remaining codon integers are not used and can be ignored. In contrast, if N is shorter than E, then the same codon integers may be re-used to produce a valid result using wrapping.

3.4 Descriptions of the Experimental Set Up

In this thesis, we have conducted four groups of experiments, each with various exploration strategy configurations, in order to evaluate our proposed approach based on the theoretical frameworks explained in section 3.1 to section 3.3. In the following four chapters, we dedicate each chapter to each group of experiments:

1. *Chapter 4* - We consider the problem of motion control of exploration with teams of robots. We present our approach to design a coordinated path planner for a small team of unmanned ground vehicles (UGVs) mapping an unknown two-dimensional area.

2. *Chapter 5* - We extend our approach to a single robot system with multi-requirement exploration. We use a variation of evolutionary technique namely Covariance Matrix Adaptation Evolutionary Strategies (CMA-ES) as a comparison of optimisation performance.

3. *Chapter 6* - We demonstrate that our approach using GE can be expanded to evolve not only parameters of a motion controller but also the entire structure of π of the controller. We propose a number of BNF grammars to accommodate different level of evolution needed.

4. *Chapter 7* - Finally, we evaluate our approach on exploration that uses one of the most popular types of reactive controllers: Dynamic Window Approach (DWA). We further extended our GE capability to choose factors (state signals) that should be considered by π of DWA to perform good navigation. This experiment is conducted on a commercial robot platform called Turtlebot [41].

3.4.1 Software

In order to perform above-mentioned experiments, we use a number of open-source software packages. Following is the list of open-source software being used:

1. Player/Stage Robot Simulator [106].

2. Robot Operating System (ROS) [40].

3. GALib version 2.4.7 [92].

4. libGE version 0.26 [102].

All experiments are conducted under Ubuntu 10.04 and 12.04 operating system environments.

4 Autonomous Exploration with Multiple Robots

In this chapter, we consider the problem of multi-robot autonomous exploration. Multi-robot exploration has particular benefits over single-robot exploration as briefly described in section 2.2.1. Among the benefits are faster exploration, better fault-tolerance and sensor uncertainty compensation [14]. Multi-robot exploration is also becoming progressively less expensive, making multi-robot exploration a more accessible option. However, developing motion control for a multi-robot system presents extra challenges in terms of coordination, collision avoidance and sharing of information. In this work, we adopt a coordinated Frontier-based exploration strategy as in [14] to develop the coordinated goal-assignment module of our exploration system. Hence, we show that the performance of the coordinated exploration strategy can be enhanced by optimising its motion control that takes into account constraints involving distances between robots and static hazards (obstacles) in an unknown environment. We assume robots can communicate and exchange information with the system's ground station at all time. In the case of the communication interruption, the impact to exploration performance is minimised due to the fact that the ground station is only responsible to update goal-assignment planning, meanwhile motion control is done independently by each robot. We demonstrate that GE can be used to evolve the parameters of π for the above motion control. This system is implemented on a team of simulated wheel-driven mobile robots also called Unmanned Ground Vehicles (UGVs) performing autonomous exploration of two-dimensional unknown environments. The experimental results suggest that GE-based design of motion control outperforms handwritten design motion control significantly.

4.1 Objectives

The main objectives of this experiment are:

1. to evolve a π for motion control of a coordinated Frontier-based multi-robot exploration strategy using GE.

2. to introduce a BNF grammar for tuning numerical constants of a polynomial function representing π.

3. to compare exploration performance of the evolved controller with a carefully designed handwritten controller.

4.2 The Exploration Framework

This section describes the overall architecture of the coordinated Frontier-based exploration strategy. In the later sections, we demonstrate the application of GE to tune motion control of this coordinated exploration strategy. The configuration of a team of UGVs used in this work is shown in figure 4.1. Note that our set up is partially centralised with motion control is made independently by each UGV but map fusion and goal-assignment are done globally.

Physically, this system consists of a ground station (GS) and a number of UGVs. The system permits communication to take place between the GS and UGVs. Bi-directional data transfer between GS and UGVs is allowed. It is assumed that each UGV has accurate localisation information for itself and has access to a composite map fused from all UGVs via the GS. Each UGV provides proximity data from its sensor reading to the GS. The GS reads these data and translates them into a global occupancy grid map. Then, the GS distributes the global map to every UGV such that global information can be shared among all UGVs. Another data type that is shared between the GS and

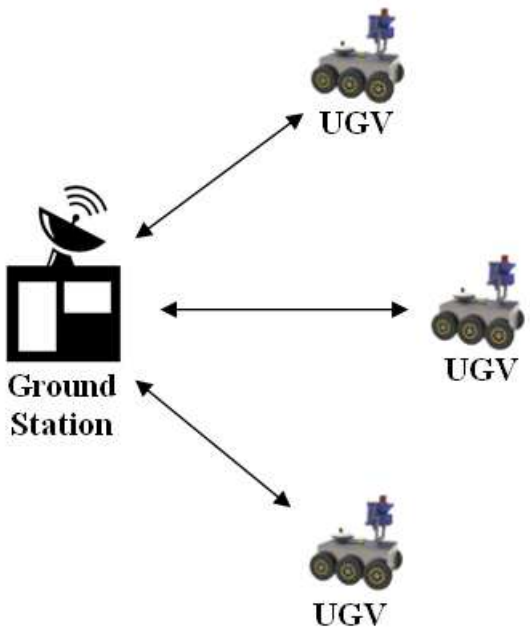


Figure 4.1: A multi-UGV system with one ground station (GS). Information can be transferred in a bi-directional way between the GS and UGVs. Goal-assignment for all UGVs is processed in the GS, while motion control is handled independently by each UGV.

the UGVs is location coordinates. Each UGV receives two types of location information in the global frame from the GS i.e. i) the current pose of all UGVs, and ii) the immediate goal pose for the corresponding UGV.

Motion control of each UGV is made decentralised. Every UGV is responsible for planning and executing its movement individually by using shared information received from the GS. This independent control structure allows each robot to make its decision faster and is communication-failure proof. Once global information is received, the motion control of each UGV generates appropriate state signals, makes navigational decision using a specific π, and executes the navigational decision accordingly. In the case of temporary communication failure between the GS and a UGV, the ground station will generate a global map based on information gathered from connected UGVs only. Consequently, it will only update the immediate goal pose of the connected UGVs. For the unconnected UGV, it still can navigate towards its current goal since motion control is done locally by the UGV. Once the

UGV has reached the goal location, the system can have two options, either wait for the communication to remedy to assign a new goal or the UGV can define its new goal independently. Note that, in this experiment, we did not perform any testing in the condition of communication-failure.

4.2.1 UGV Model

In our implementation, we assume all UGVs that we use have the same configuration and dimensions. The UGV is based on a hardware prototype developed at the University of Adelaide that was built purposely for a competition named Multi-Autonomous Ground-robotic International Challenge (MAGIC) 2010 [38]. The UGV has a measurement of 0.7 by 0.9 metres with six wheels and skid-steering [18]. This non-holonomic robot is equipped with a range of sensors. A 2D Light Detection and Ranging Sensors (LIDAR), a laser-range finder, is installed to generate distance and range information about the surrounding environment. The model used is Leuze Rod4-08 Plus [34] with a maximum 190° of detection angle. A differential-type Global Positioning System (GPS) is embedded into the UGV system to provide localisation information for outdoor environments. The GPS unit chosen was a Novatel OEMV-2 model which has the measurement accuracy up to $0.2m$ RMS error when set up with RT-20 firmware [101]. An Inertial Measurement Unit (IMU) and wheel encoders are also installed to handle localisation in indoor environments. For the IMU, a Microstrain 3DM-GX3-25 [85] model is used. Figure 4.2 shows the prototype UGV.

The kinematics of the UGV are described as follows. Given a steering direction, a two-stage point-and-shoot motion model is used to move the UGV in the desired direction as shown in figure 4.3(a). First, the UGV skid-steers at an angular velocity, v_a to face the desired steering direction (Stage 1). Second, the UGV moves for a short distance forward in that direction at a translational velocity, v_t (Stage 2). Note that the UGV cannot drive backwards under this regime. In this experiment, fixed velocities are applied for all UGVs, a pragmatic decision to help reduce positioning error for the evolutionary

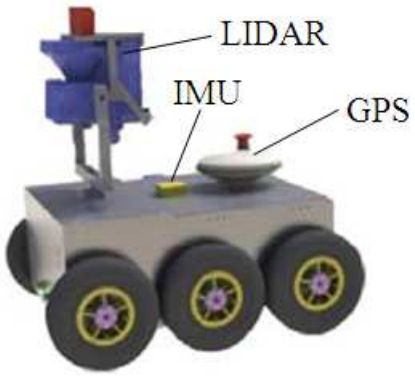


Figure 4.2: A prototype UGV developed at the University of Adelaide for Multi-Autonomous Ground-robotic International Challenge (MAGIC) 2010. The UGV is equipped with three sensors: a Laser-Range Finder (LIDAR), a differential-type Global Positioning System (GPS) and an Inertia Measurement Unit (IMU).

process. Velocities of the robot are set as $v_t = 1m/s$ and $v_a = 40deg/s$. Later experiments uses varied velocities to cater for other applications.

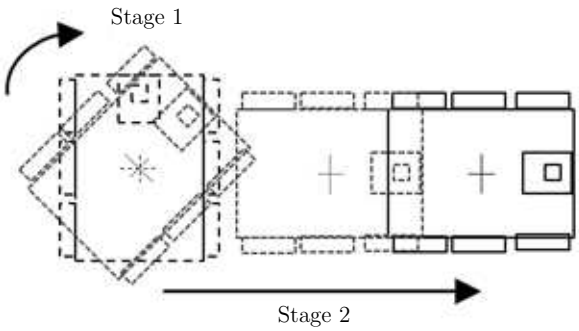
The kinematic model of the UGV is based on the configuration transition equations [77]. Equation 4.1 describes the kinematic model for the UGV in two-dimensional environments:

$$\dot{x} = rv_t cos\theta \qquad \dot{y} = rv_t sin\theta \qquad \dot{\theta} = \frac{r}{L}v_a \tag{4.1}$$

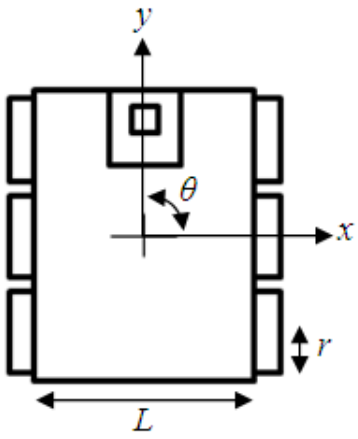
where $(\dot{x}, \dot{y})$ is the change of the UGV's pose in two-dimensional Cartesian coordinates and $\dot{\theta}$ is change of its angle bearing. L is the distance between two wheels, meanwhile r is the radius of the UGV wheels.

4.2.2 Exploration Strategy

The framework of exploration strategy adopted in this experiment adheres to the framework of the Frontier-based strategy explained in section 3.1. Figure



(a)



(b)

Figure 4.3: (a) Motion model of the UGV: rotation (stage 1) followed by translation (stage 2). First, the UGV skid-steers at an angular velocity, v_a to face the desired steering direction (stage 1). Second, the UGV moves for a short distance forward in that direction at a translational velocity, v_t (stage 2). (b) Kinematic model of UGV [59, 60].

4.4 shows the building blocks of the framework and data fusion from hardware to software and vice-versa.

The process starts with a request from the GS to acquire sensory data from all UGVs. The sensory data are proximity data and each UGV's current pose. Each UGV generates proximity data from its LIDAR and its pose by triangulation of the GPS device, IMU and wheel encoders. In our simulation, we simplify positioning measurement by assuming the position data received by the GS is accurate within a small acceptable error due to the installation of

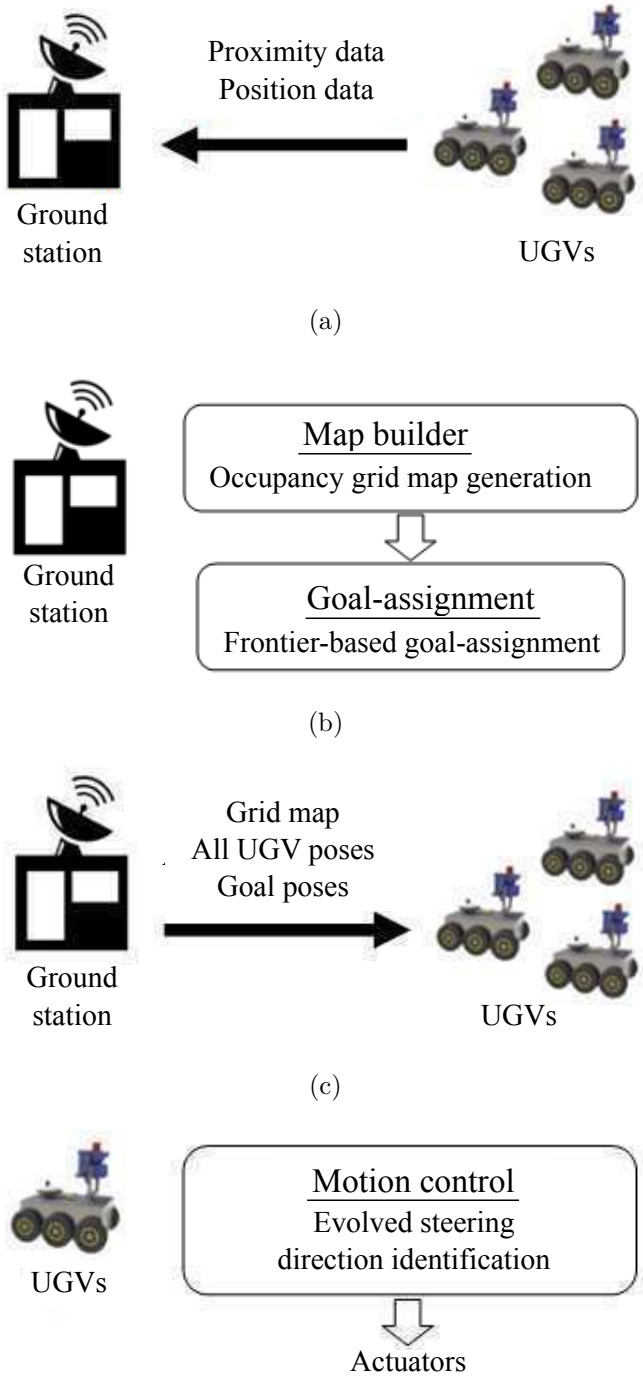


Figure 4.4: The framework of our exploration strategy: (a) The ground station requests sensory data from the UGVs. (b) Goal-assignment is executed by the ground station. (c) The round station sends map and goal pose to UGVs (d) Motion control is performed by each UGV.

high-end differential GPS Novatel OEMV-2. The acceptable error in this case is set within 0.1m which is equal to the resolution of occupancy grid map being used in this experiment. We assume localisation is done by the UGV before sending its approximated position to the GS. After gathering UGV data, the GS activates the map builder in which an occupancy grid map, $O(\mathbf{c})$, where $\mathbf{c}$ is an array of grid cells, is updated based on current available data. $O(\mathbf{c})$ is a global map representation that concatenates sensory data from all UGVs and transforms them into global frame coordinates. Subsequently, the GS triggers the goal-assignment module to perform allocation of immediate goal poses for each UGV using the Frontier-based algorithm with simultaneous Utility-and-Cost calculations [14]. The process of updating a grid map and assigning frontiers was discussed in the previous chapter in sub-sections 3.1.1 and 3.1.3, respectively. After frontier assignment, the GS distributes $O(\mathbf{c})$, the pose of each UGV and its corresponding goal pose to all available UGVs. The GS acquires sensory data and builds the map at frequency f_{GS}, while goal-assignment is updated at frequency f_{GA}, in which $f_{GA} \leq f_{GS}$. We set $f_{GS} = f_{GA} = 1$ Hz such that the map can be updated in real-time while the latest frontiers can be chosen as target locations at a high rate. From our computational resources, the GS is able to execute both processes at a rate higher than 1 Hz. Sensory data synchronisation is explicitly done by the underlying program in the GS that validates the data from each UGV through the embedded time-stamp.

Now, the process is continued by the UGVs. Each UGV is responsible for controlling it movement by executing Decision-Theoretic Motion Control (DTC) at a pre-defined control frequency, f_C, where $f_C \leq f_{GA} \leq f_{GS}$. Here, we set $f_C = 0.1$ Hz. In every f_C, the DTC of a UGV is required to select a good steering direction. A π is used to make the decision based on a set of current states and a set of available options. The UGV sends the steering direction value to the low level control that later translates it into significant wheel actuator movement.

4.2.3 Motion Control Structure

The aim of the DTC of each UGV_i in this experiment is to select an optimised steering direction, $\theta_{i,d}$ in every f_C. In regard to the exploration mission, the DTC must ensure that by navigating through a sequence of $\theta_{i,d}$, UGV_i will not collide with any obstacle, minimise the occurrence of entrapment and maintain a safe-distance among other UGVs while moving towards goal locations.

The core component of the DTC is its π. Recall the steps taken to build π from the previous chapter of section 3.2.1. To build π, two sets of auxiliary components are needed i.e. a set of current states and a set of actions. The current states are sampled for each possible action, producing a set of navigational options. Using the produced set, the best navigational option is determined for π according to a scoring function (refer equation 3.7). In particular, three state signals are defined in this configuration representing the current states, hence, producing a three dimensional state space. The defined state signals are static hazard, dynamic hazard and goal strength. The static hazard measures the safety of a UGV from collision with static obstacles. The dynamic hazard measures the risk of a UGV hitting another UGV. The goal strength measures the progress made by a UGV towards its goal location [59]. The action set is the set of candidate steering directions, hence the navigational options are a set of candidate steering directions with their corresponding samples of state signals.

Figure 4.5 shows the structure of the DTC being used in this experiment. We explain the process of generating relevant components in this structure in turn.

Candidate Steering Directions

Consider figure 4.6. Given a current pose of UGV_i, $p_{c,i} = (x_i, y_i, \alpha_i)$, where (x_i, y_i) is a global coordinate of the center point of UGV_i in a two-dimensional map and α_i is the current heading of UGV_i, a set of candidate steering

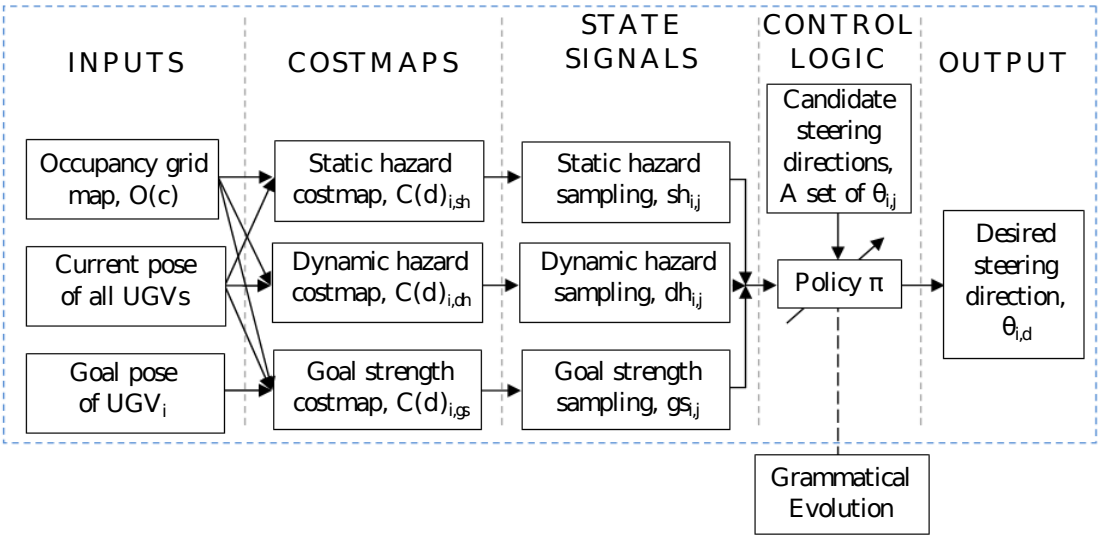


Figure 4.5: The motion control structure of a UGV_i. The DTC receives three types of input from the main exploration system: a global occupancy grid map, all robots' poses and UGV_i's goal pose. The DTC builds a set of costmaps from these inputs. The costmaps are used in the sampling process of state signals for each possible steering direction $\theta_{i,j}$, which produces a set of navigational options. Finally, π evaluates all the navigational options and chooses the best steering direction $\theta_{i,d}$. This process occurs within f_C.

.

directions $\theta_{i,j}$ can be generated by using equation 4.2.

$$\theta_{i,j} = \alpha_i + \beta * j \tag{4.2}$$

where β is the angular resolution and $j = 0, 1, ..., N-1$ where $N = 360/\beta$ in degree unit or $N = 2\pi/\beta$ in radian unit, determines the number of candidates. In this experiment, we set $\beta = 5°$ or $0.087266rad$ which results in $N = 72$ candidate steering directions. Since f_C is set at 0.1Hz, the selection of β with 5° is considered as high angular resolution that is enough to differentiate the direction of movement of the UGV in one control cycle. In every f_C, the policy π will select the best steering direction, $\theta_{i,d}$ from the list of 72 candidate directions and perform point-and-shoot movement towards the selected direction.

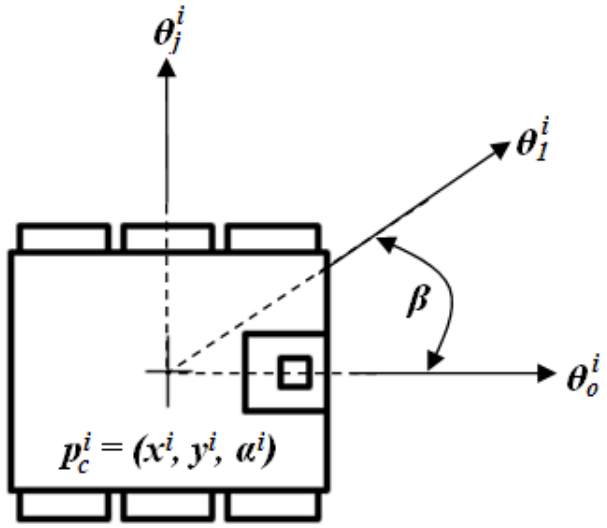


Figure 4.6: UGV state schema in the global frame. $p_{c,i} = (x_i, y_i, \alpha_i)$ is a global coordinate of the center point of UGV_i in a two-dimensional map. $\theta_{i,j}$ is a candidate steering direction.

State Signals

Each state signal highlights a specific feature of a UGV's surrounding environment. The idea behind the state signal generation is that each signal views its surroundings from a particular perspective. Broadly, state signals are built upon a global occupancy grid map supplied by the GS. However, extracting a highlighted feature directly from the grid map is difficult without enhancing the particular information being observed. Therefore, a costmap is first generated to enhance features and filter out unnecessary information. Next, the costmap is used to sample the corresponding state signal at each navigational option. In the following, we explain the steps taken to extract each state signal.

Static hazard state signal extraction - The static hazard $sh_{i,j}$ of $\theta_{i,j}$ of a UGV_i gives approximate information on the degree of distance to static obstacles for the UGV to safely traverse along $\theta_{i,j}$. A costmap $C(\mathbf{d})_{i,sh}$ featuring static obstacles on $O(\mathbf{c})$ is developed by establishing a grid distance-field that provides each free cell in $O(\mathbf{c})$ with a cost-distance d to the nearest fixed

obstacle. Figure 4.7 shows an example of a $C(\mathbf{d})_{i,sh}$ interpreted from an $O(\mathbf{c})$. In this example, black-coloured areas in $C(\mathbf{d})_{i,sh}$ represent obstacle cells or unknown cells. We consider those cells as fully unsafe to navigate. The cost-distance of those cells, $d_o, d_u \in \mathbf{d}$ is zero. In contrast, grey-coloured areas denote free cells with a positive valued cost-distance $d_f \in \mathbf{d}$ where d_f is determined by the distance to the nearest fixed obstacle. The brighter area has further distance from obstacles and vice-versa. Meanwhile, white-coloured areas are assumed as a group of free cells that are considered as fully safe to navigate since these cells have reasonably great distance from the nearest obstacle. The cost-distance of those free cells is set to ∞.

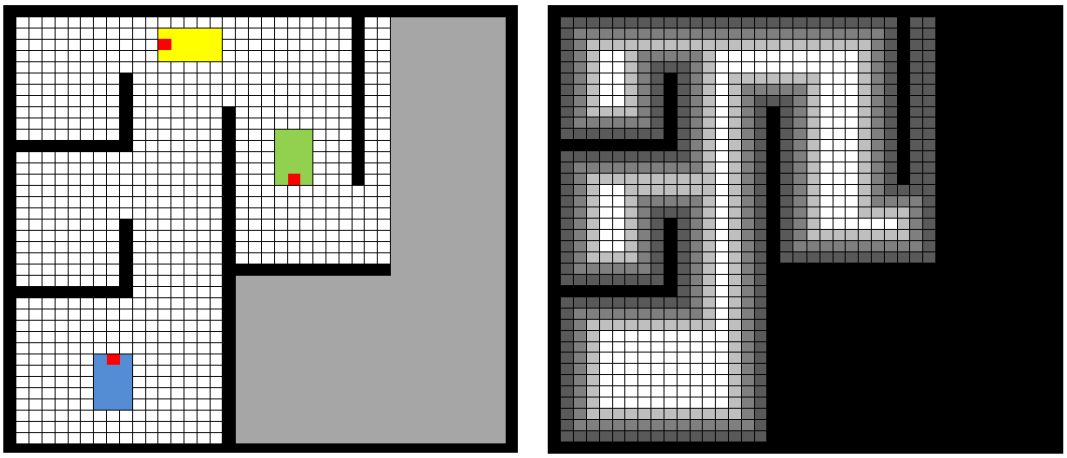


Figure 4.7: An example of static hazard costmap generation (a) an occupancy grid map (b) a corresponding static hazard costmap

A $C(\mathbf{d})_{i,sh}$ is built using a d_1 distance transform algorithm in [5]. In essence, all occupied and unknown cells in $O(\mathbf{c})$ are initialised to zero distance indicating that they are unsafe cells, while free cells are initialised to ∞ distance. Then, the value of appropriate free cells is updated iteratively according to their position from the nearest unsafe cell. Algorithm 11 shows the algorithm we use to generate $C(\mathbf{d})_{i,sh}$ given a $O(\mathbf{c})$.

Next, given $C(\mathbf{d})_{i,sh}$ and the current pose $p_{i,c}$ of a UGV_i, we can sample $sh_{i,j}$ at each $\theta_{i,j}$. We calculate $sh_{i,j}$ such that it can estimate the risk of UGV_i hitting a static obstacle through its movement to the goal using the

Algorithm 11: Algorithm to generate a static hazard costmap $C(\mathbf{d})_{i,sh}$ of UGV_i.

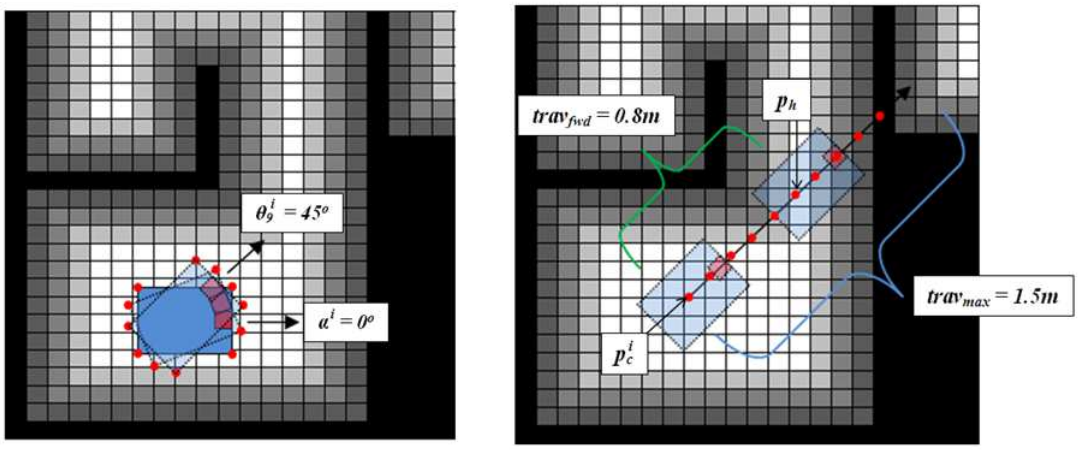
```
1  Initialise C(d)_{i,sh} field of O(c) size.
2  Initialise the cost-distance of all free cells, d_f ∈ d as ∞ (i.e. a large
   number, M).
3  Initialise the cost-distance of all occupied cells, d_o ∈ d and unknown
   cells, d_u ∈ d as 0.
4  Set a list, L_curr with the index of those occupied and unknown cells.
5  Set d_curr = 0.
6  Set d_max = D where D is a positive value bigger than the radius of
   UGV_i.
7  while d_curr < d_max do
8      Initialise another list, L_next to the empty list.
9      for every cell, k in L_curr do
10         for every free cell neighbour of k, l do
11             Get the current cost-distance of cell l, d_l ∈ d.
12             if d_l = M then
13                 Update d_l = d_curr.
14                 Insert l into L_next.
15     Update d_curr ++.
16     Update L_current = L_next.
```

point-and-shoot movement described in section 4.2.1. This algorithm for any UGV_i approximates the possibility of collision with static obstacles by checking the cost-distance of relevant points of the UGV's body to the nearest obstacle using $C(\mathbf{d})_{i,sh}$. When the UGV_i performs stage 1 movement (turn), it will check whether the corners of the UGV body (in the case of a rectangular body, the number of corners are 4) hypothetically hit an obstacle along the generated arcs. While the UGV_i performs stage 2 movement (the traverse along $\theta_{i,j}$), it will use the centre point of the UGV body to check for any possible collision along the traversal. In addition, UGV_i also performs checking that it is possible to make a safe turn after the traversal to avoid entrapment. This is done by checking the location along $\theta_{i,j}$ that is bigger than the radius of UGV_i. The final value of $sh_{i,j}$ is formulated using equation 4.3 by taking into account the results of above checking processes. $sh_{i,j}$ has a value normalised to the range 0.0 to 1.0, from the lowest hazard to the highest hazard, respectively. Algorithm 12 calculates $sh_{i,j}$ of $\theta_{i,j}$ based on the point-and-shoot movement. Equations

4.3 and 4.4 can be used to calculate $sh_{i,j}$ given that $trav_{max}$ is the maximum checkable traverse distance, $trav_{safe}$ is the net maximum safe-traverse distance, $trav_{fwd}$ is the maximum safe-traverse distance in forward movement and $trav_{bwd}$ is the maximum safe-traverse distance in backward movement.

$$sh_{i,j} = \frac{trav_{max} - trav_{safe}}{trav_{max}} \qquad sh_{i,j} \in [0,1] \tag{4.3}$$

$$trav_{safe} = trav_{fwd} - trav_{bwd} \tag{4.4}$$



(a) (b)

Figure 4.8: An example of $sh_{i,j}$ calculation (a) Evaluation of stage 1 hypothetical moves, (b) Evaluation of stage 2 hypothetical moves.

To illustrate the calculation of $sh_{i,j}$ using Algorithm 12, consider an example in figure 4.8. In this figure, UGV_i is surrounded by obstacles. Assume $p_{i,c}$ is $(0, 0, 0^o)$ and the candidate steering direction under consideration is $\theta_{i,9} = 45^o$ in global frame. The maximum allowable traverse distance, $trav_{max}$ is set to 1.5m. User-defined safe-distance thresholds th_{s1} and th_{s2} are defined as 0.1m and 0.4m, respectively. We can manipulate th_{s1} and th_{s2} to urge the UGV to stay away from obstacles at a specific distance. To calculate $sh_{i,9}$, first we check for possible collision on stage 1 movement from 0^o direction to 45^o direction as in figure 4.8(a). The cost-distance of the red dots representing hypothetical corners of the UGV during turning are checked. Sampling these

Algorithm 12: Algorithm to sample static hazard state signal, $sh_{i,j}$ of $\theta_{i,j}j$ for UGV_i.

Check for possible collision from performing stage 1 movement (turn) at the current pose of UGV_i, $p_{i,c} = (x_i, y_i, \alpha_i)$. This is done by checking the cost-distance of the UGV corner points, $\mathbf{d}_{cn} \in \mathbf{d}$ at every hypothetical turning angle, ϕ_h from α_i to $\theta_{i,j}$, where ϕ_h increment or decrement towards $\theta_{i,j}$ is determined by an angle interval set by the user.

for $\phi_h = \alpha_i$ *to* $\phi_h = \theta_{i,j}$ **do**

  Check $\mathbf{d}_{cn}$ values from $C(\mathbf{d})_{i,sh}$ field at an instant ϕ_h.

  if *any* $d \in \boldsymbol{d}_{cn}$ *is equal to* 0 *or less than a user-defined safe-distance threshold,* th_{s1} **then**

    Terminate and return $sh_{i,j} = 1.0$ immediately. //indicates collision

Check possible collision of performing stage 2 movement (traverse along $\theta_{i,j}$). This is done by checking the cost-distance of the UGV centre point, $d_{ct} \in \mathbf{d}$ at every hypothetical position, p_h from $p_{i,c}$ to p_{max} along $\theta_{i,j}$. p_{max} is determined by the maximum allowable traverse distance, $trav_{max}$ set by the user.

Initialise the maximum safe-traverse distance, $trav_{fwd}$ to 0.

for $p_h = p_{i,c}$ *to* $p_h = p_{max}$ **do**

  Check d_{ct} values from $C(\mathbf{d})_{i,sh}$ field at the current p_h.

  if $d_{ct} = 0$ *or* d_{ct} *is less than a user-defined safe-distance threshold,* th_{s2} **then**

    Goto A.

  Update $trav_{fwd}$ by calculating the Euclidean distance between $p_{i,c}$ and the current p_h.

A: Check for possible collision of stage 2 virtual backward movement to avoid entrapment. The UGV has to ensure that there is a point along the ray where it can safely turn after a traverse in $\theta_{i,j}$ direction. This is done by performing backward ray traversal from the last point of p_h back to $p_{i,c}$ to determine the distance of the first point in backward traversal, $trav_{bwd}$, that has a value larger than the radius of the UGV, r_i.

Initialise $trav_{bwd}$ to 0.

for $p_b = p_h$ *to* $p_b = p_{i,c}$ **do**

  Check d_{ct} values from $C(\mathbf{d})_{i,sh}$ field at an instant p_b.

  if $d_{ct} > r_i$ **then**

    Goto B.

  Update $trav_{bwd}$ by calculating the Euclidean distance between p_h and the current p_b.

B: Sample the final static hazard value of $\theta_{i,j}$ using equation 4.3 and 4.4.

values from $C(\mathbf{d})_{i,sh}$, we found that all values are above th_{s1} indicating no possible collision. Next, we check for possible collision on stage 2 movement as in figure 4.8(b). Again, the cost-distance of the red dots, now representing the hypothetical centres of the UGV during traversal at $\theta_{i,9}$, are being checked. We found that the last safe point p_h on the ray (where a sequence of red dots before the last safe point has cost-distance value above th_{s1}) has the distance 0.8m from $p_{i,c}$, thus the maximum safe-traverse distance, $trav_{fwd} = 0.8$m. Then, we check the backward movement at $\theta_{i,9}$ from p_h point. We found that the first point in backward traversal that has a cost-distance value larger than r_i is at the p_h point. Thus, $trav_{bwd} = 0.0$m. Finally, we can calculate $sh_{i,9}$ of $\theta_{i,9}$ as follows:

$$sh_{i,9} = \frac{1.5 - (0.8 - 0.0)}{1.5} = 0.467 \tag{4.5}$$

Dynamic hazard state signal extraction - The dynamic hazard state signal $dh_{i,j}$ of $\theta_{i,j}$ of UGV_i measures the risk of UGV_i hitting another UGV [59]. Using information from the dynamic hazard state signal, a UGV can avoid navigating near other UGV team members and maintain safe distance between each other. Hence, it can minimise the possibility of collision among UGVs which could shut down the exploration mission catastrophically. A candidate $dh_{i,j}$ is also measured in the same way as the corresponding static hazard but it features only the dynamic obstacles of other UGV team members. A dynamic hazard costmap, $C(\mathbf{d})_{i,dh}$ is first built. $C(\mathbf{d})_{i,dh}$ measures the cost-distance of each free cell in $O(\mathbf{c})$ to the nearest position of UGV team members. Consequently, $dh_{i,j}$ is sampled for each $\theta_{i,j}$ by extracting cost-distance information from $C(\mathbf{d})_{i,dh}$. Algorithm 13 defines an algorithm to build $C(\mathbf{d})_{i,dh}$ for a UGV_i, while figure 4.9 shows an example of a $C(\mathbf{d})_{i,dh}$ for a corresponding $O(\mathbf{c})$.

Sampling $dh_{i,j}$ is straightforward. Algorithm 14 describes the process of sampling $dh_{i,j}$ for a given $\theta_{i,j}$ of UGV_i. Briefly, for each $\theta_{i,j}$, the algorithm will check on how far UGV_i can traverse along $\theta_{i,j}$ direction before it hits other team member, if any. Given that $trav_{max}$ is the maximum checkable traverse distance and $trav_{safe}$ is the maximum safe-traverse distance, the dynamic

Algorithm 13: Algorithm to generate a dynamic hazard costmap $C(\mathbf{d})_{i,dh}$ of UGV_i.

```
Initialise C(d)_{i,dh} field of O(c) size.
Initialise the cost-distance of all free cells, d_f ∈ d to ∞ (i.e. a large number, M).
Initialise the cost-distance of all occupied cells, d_o ∈ d and unknown cells, d_u ∈ d to 0.
for each UGV team member, m where m ≠ i do
    Set the cost-distance of cells corresponding to the position of m, d_m ∈ d to 0.
    Set a list, L_curr with the index of cells corresponded to the position of m.
    Set d_curr = 0.
    Set d_max to the diameter of m.
    while d_curr < d_max do
        Update d_curr ++.
        Initialise another list, L_next to the empty list.
        for every cell, k in L_curr do
            for every neighbour of k, l do
                Get the current cost-distance of cell l, d_l ∈ d.
                if d_l > d_curr then
                    Update d_l = d_curr.
                    Insert l into L_next.
        Update L_current = L_next.
```

hazard value of $\theta_{i,j}$ can be calculated as in equation 4.6.

$$dh_{i,j} = \frac{trav_{max} - trav_{safe}}{trav_{max}} \qquad dh_{i,j} \in [0,1] \tag{4.6}$$

Goal strength state signal extraction - The third state signal is goal strength. A goal strength state signal measures the effect of UGV_i moving in $\theta_{i,j}$ direction in terms of the progress made towards the current goal location of UGV_i. It is used to attract UGV_i towards its goal location. The goal signal $gs_{i,j}$ of a candidate steering direction $\theta_{i,j}$ of UGV_i determines the relative closeness of a UGV to its goal location when moving in $\theta_{i,j}$. $gs_{i,j}$ is higher if the path in a desired $\theta_{i,j}$ is closer to UGV_i's goal location [59]. Like the previous two state

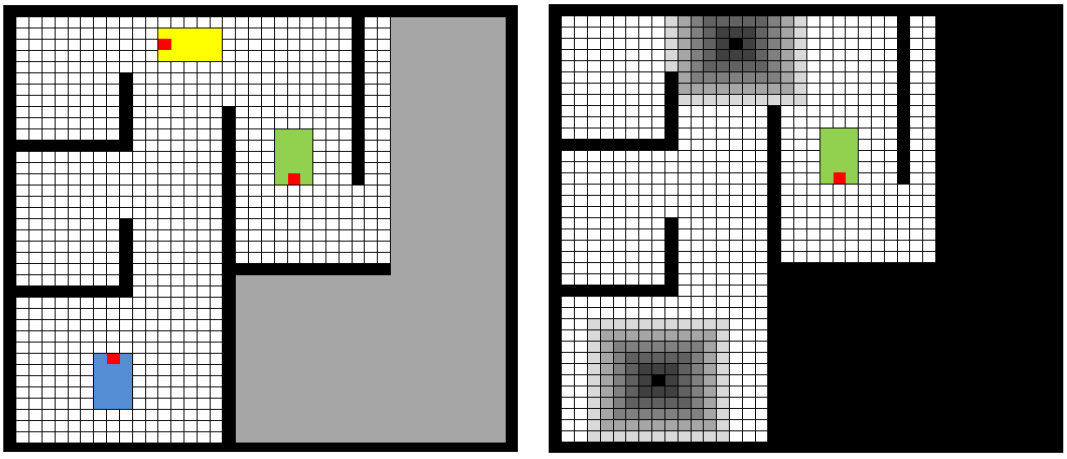


Figure 4.9: An example of dynamic hazard costmap generation (a) an occupancy grid map (b) a corresponding dynamic hazard costmap $C(\mathbf{d})_{i,dh}$ of UGV_i (in this case, the green-coloured UGV).

Algorithm 14: Algorithm to sample the dynamic hazard state signal $dh_{i,j}$ of $\theta_{i,j}$ of UGV_i.

1 Check for possible collision with other UGVs when traversing along $\theta_{i,j}$. This is done by checking the cost-distance of UGV_i's centre point, $d_{ct} \in \mathbf{d}$ at every hypothetical position, p_h from the initial position of UGV_i, $p_{i,c}$ to p_{max} along $\theta_{i,j}$. p_{max} is determined by the maximum checkable traverse distance, $trav_{max}$ set by the user.
2 Initialise the maximum safe-traverse distance, $trav_{safe}$ to 0.
3 Initialise a safe-distance threshold, th_{dh} as the diameter of UGV_i.
4 **for** $p_h = p_{i,c}$ *to* $p_h = p_{max}$ **do**
5 Check d_{ct} value at an instant p_h acquired from $C(\mathbf{d})_{i,dh}$ field.
6 **if** $d_{ct} < th_{dh}$ **then**
7 Goto A.
8 Update $trav_{safe}$ by calculating the Euclidean distance between $p_{i,c}$ and p_h.
9 A:
10 Sample the final dynamic hazard value of $\theta_{i,j}$ using equation 4.6.

signals, $gs_{i,j}$ also needs a costmap $C(\mathbf{d})_{i,gs}$ to guide the state signal sampling task. We adopt and modify the algorithm of Frontier cost calculation used in [118] to establish $C(\mathbf{d})_{i,gs}$. Algorithm 15 defines this calculation. At the start, the cost-distance of UGV_i's goal cell is initialised to 0 and other cells to ∞.

Then, the cost-distance of free cells is refined by the minimum cost-distance among all neighbour cells plus the Euclidean distance to the neighbour cell. Figure 4.10 shows an example of $C(\mathbf{d})_{i,gs}$ built for UGV_i given its goal location. Note that, dark-blue-coloured regions, which are relatively near to the goal location, have lower cost-distance rather than grey-coloured regions.

Algorithm 15: Algorithm to generate a goal strength costmap $C(\mathbf{d})_{i,gs}$ of UGV_i.

```
1  Initialise C(d)_{i,gs} field of O(c) size.
2  Initialise the cost-distance of UGV_i's goal cell, d_{i,g} ∈ d to zero.
3  Initialise the cost-distance of all other cells, d ∈ d to ∞ (i.e. a large
   number, M).
4  Set a list, L_curr with the index of UGV_i's goal cell.
5  while L_curr is not empty do
6      Initialise another list, L_next to the empty list.
7      for every cell, k in L_curr do
8          Get the current cost-distance of cell k, d_k ∈ d.
9          for every free cell neighbour of k, l do
10             Get the current cost-distance of cell l, d_l ∈ d.
11             Calculate the Euclidean distance between k and l, d_Euc.
12             if d_l > d_k then
13                 Update d_l = d_k + d_Euc.
14             Insert l into L_next.
15     Update L_curr = L_next.
```

Using information from $C(\mathbf{d})_{i,gs}$, UGV_i samples the $gs_{i,j}$ of each $\theta_{i,j}$ by measuring the minimum cost-distance along the $\theta_{i,j}$ path. $gs_{i,j}$ is calculated by manipulating the minimum cost-distance d_g and the cost-distance of UGV_i's current position $d_{i,c}$ to produce a relative distance measurement indicating the 'closeness' of UGV_i from its goal location if it decides to take the direction of $\theta_{i,j}$. Algorithm 16 calculates $gs_{i,j}$ for $\theta_{i,j}$. Equation 4.7 calculates $gs_{i,j}$. Note that, unit conversion is needed for $trav_{max}$ from real distance to cost-distance unit in this equation.

$$gs_{i,j} = \frac{(d_c^i - d_g)}{trav_{max}} \qquad gs_{i,j} \in [0,1] \tag{4.7}$$

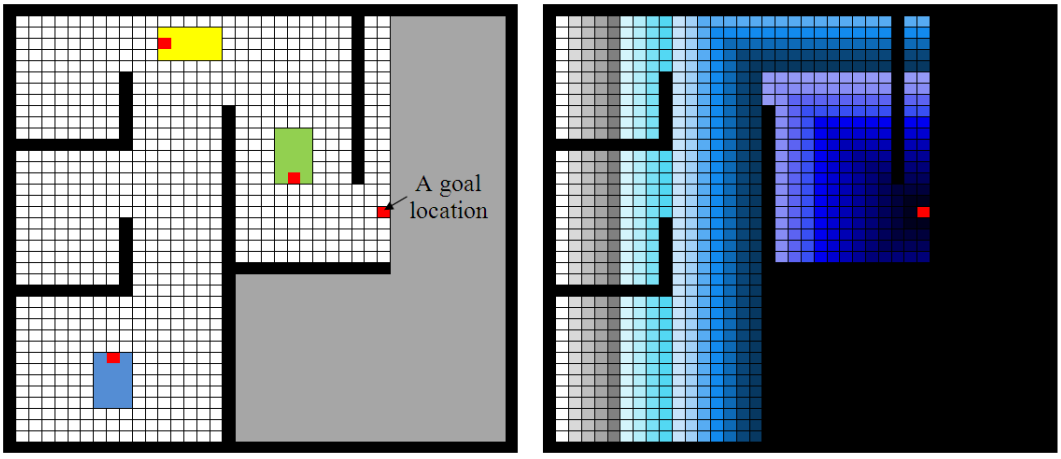


Figure 4.10: An example of goal strength costmap generation (a) an occupancy grid map (b) a corresponding goal strength costmap $C(\mathbf{d})_{i,gs}$ of UGV_i.

Algorithm 16: Algorithm to sample a goal strength state signal, $gs_{i,j}$ of $\theta_{i,j}$ for UGV_i

Get the minimum cost-distance along traversal ray in the direction of $\theta_{i,j}$. This is done by checking the distance of UGV_i's centre point, $d_{ct} \in \mathbf{d}$ at every UGV_i hypothetical position p_h from UGV_i initial position $p_{i,c}$ to p_{max} along $\theta_{i,j}$. p_{max} is determined by the maximum checkable traverse distance, $trav_{max}$ set by the user.
Initialise the minimum cost-distance to goal cell, $d_g \in \mathbf{d}$ as ∞ (i.e. a large number, M).
for $p_h = p_{i,c}$ *to* $p_h = p_{max}$ **do**
    Check d_{ct} value at an instant p_h acquired from $C(\mathbf{d})_{i,gs}$.
    if p_h *is not a free cell* **then**
        Goto A.
    if $d_g > d_{ct}$ **then**
        $d_g = d_{ct}$
A:
Get the cost-distance $d_{i,c}$ of $p_{i,c}$ from $C(\mathbf{d})_{i,gs}$.
Sample the final goal strength value of $\theta_{i,j}$ using equation 4.7.

The Policy

In the control structure of figure 4.5, π is the final component that functions as the decision maker to select $\theta_{i,d}$ from a list of $\theta_{i,j}$ in every f_C cycle. Literally,

π takes a list of $\theta_{i,j}$ and the corresponding state signals for each $\theta_{i,j}$ as inputs. π manipulates those inputs to produce a ranking list of $\theta_{i,j}$ using a scoring function, $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})$. For the purposes of ranking, moving in $\theta_{i,j}$ direction is said to be better than staying at the current position of UGV_i if the score for $\theta_{i,j}$ produced by $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})$ is higher than 0.

To illustrate the process of producing a ranking list and selecting the final $\theta_{i,d}$ of UGV_i, consider figure 4.11. Assume all state signals have been sampled for each $\theta_{i,j}$ using the algorithms mentioned previously. Here, we assume $i = 1$. We can represent a set of a particular sampled state signal for each θ_j by a histogram. Three histograms are created, each for sh_j, dh_j and gs_j, respectively. Each histogram maps the sampled state signal values versus its corresponding θ_j. Let's say, $f(\theta_j|sh_j, dh_j, gs_j)$ is a simple summation function, where the score of each θ_j is calculated by simply adding all corresponding state signal values of θ_j. Thus, a ranking histogram can be generated as the result of the above summation process. Finally, from this ranking histogram, π selects θ_j with the highest scoring value as θ_d. In this case, $\theta_d = \theta_{12}$.

For every UGV_i, having three type of state signals, $sh_{i,j}$, $dh_{i,j}$ and $gs_{i,j}$, the challenge now is to design $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})$ such that the ranking of $\theta_{i,j}$ is optimised. In order to do that, we use GE to tune some parts of the scoring function. The first thing to consider is to determine the form of $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})$. The baseline of π design is based on the classical control approach in which hand-coded design is implemented. Typically, using hand-coded approach, a weighted sum function is considered:

$$f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{handcoded} = a_1(sh_{i,j}) + a_2(dh_{i,j}) + a_3(gs_{i,j}) \tag{4.8}$$

where a_1, a_2, a_3 are numerical constants determine the influence of each state signal towards the final result. A weighted sum function is used in many policies because of several advantages. First, the function is the simplest form of multi-criteria decision-making model. Second, it is unbiased to any input as long as all input data are generated using the same units. In our

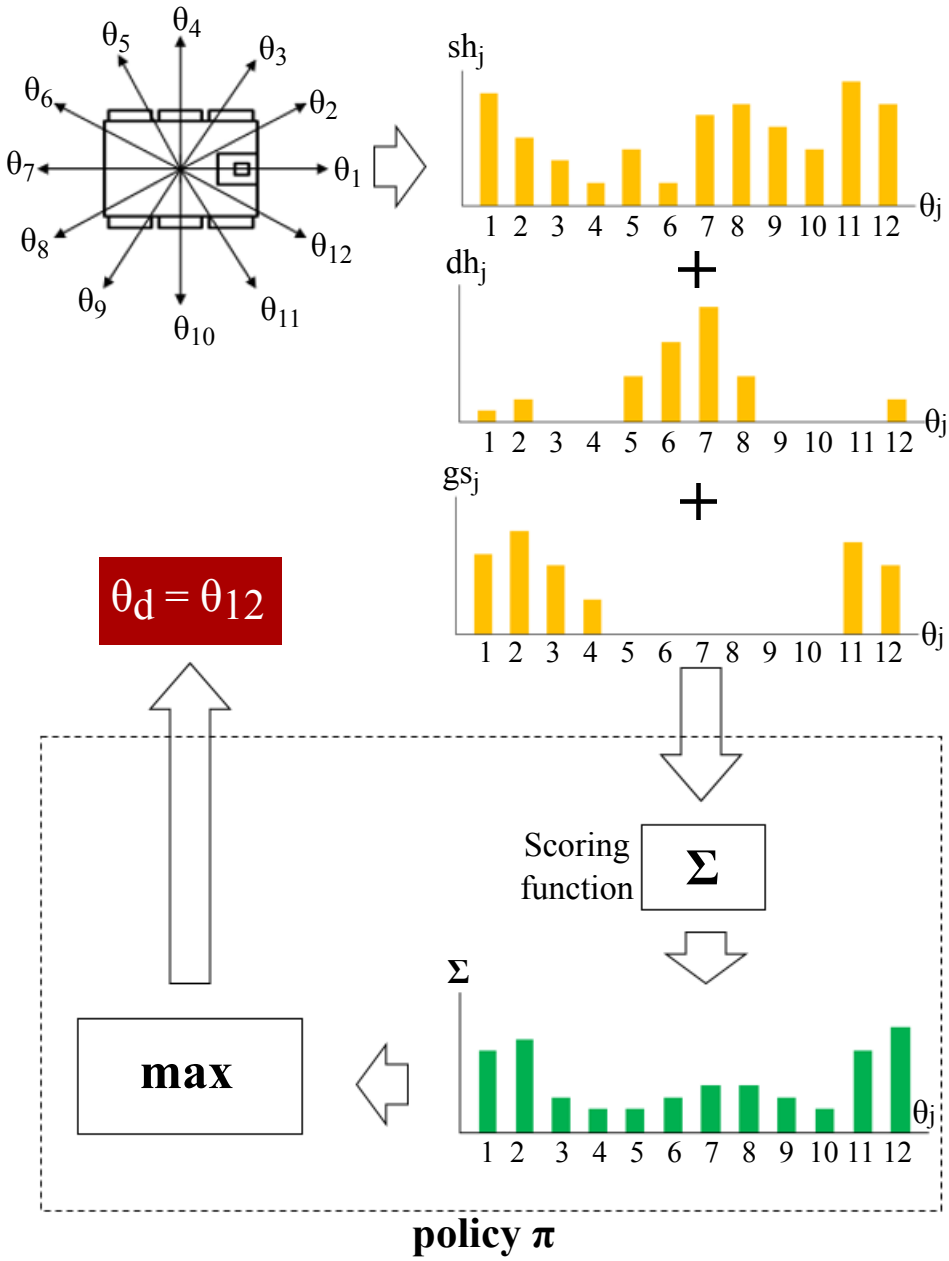


Figure 4.11: An example of the selection process of a UGV's steering direction θ_d: i) A set of candidate steering directions θ_j is identified, ii) Three state signal histograms for sh_j, dh_j and gs_j are calculated corresponding to every θ_j, iii) In π, an (overly) simple scoring function, summing up all corresponding state signal values of θ_j is used to get the score of each θ_j, iv) Finally, π selects θ_j with the highest scoring value as the result. In this case, $\theta_d = \theta_{12}$.

case, three state signal $sh_{i,j}$, $dh_{i,j}$ and $gs_{i,j}$ are all normalised within the same unit of 0.0 to 1.0. Third, the result from the weighted sum function is a

single parameterised value in which the value can unambiguously represent the quality of a candidate solution with a set of state signals. However, the weighted sum function only supports a linear relationship between state signals. It is not clear a-priori that linear functions are expressive enough to encode a good control function. Therefore, we extend the form of $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})$ to a polynomial function as shown in equation 4.9 to diversify the search for state signals relationship.

$$f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{evolved} = a_1(sh_{i,j})^{p_1} + a_2(dh_{i,j})^{p_2} + a_3(gs_{i,j})^{p_3} + b \quad (4.9)$$

The influence of each state signal is determined by the factors $\mathbf{a}$, b and powers $\mathbf{p}$, where $\mathbf{a} = \{a_1, a_2, a_3\}$ and $\mathbf{p} = \{p_1, p_2, p_3\}$. State signals are assigned with factor constants $\mathbf{a}$ and b to provide independence of scale and generality in case new state signals are needed to fulfil new exploration criteria. Power constants $\mathbf{p}$ are used to give relevant state signals domination over other signals if required [59]. Values of $\mathbf{a}$, b and $\mathbf{p}$ are evolved automatically using GE.

4.3 The Evolutionary Process

To evolve a scoring function, we re-define equation 4.9 to equation 4.10. The substraction operators are fixed as we know that state signals $sh_{i,j}$ and $dh_{i,j}$ are repulsive forces.

$$f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{evolved} = a_3(gs_{i,j})^{p_3} - a_2(dh_{i,j})^{p_2} - a_1(sh_{i,j})^{p_1} + b \quad (4.10)$$

We design the evolutionary process using GE as described in section 3.3. To conduct the evolutionary process of a set of parameters in $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{evolved}$, we define a BNF grammar that can syntactically produce correct expressions in the form of equation 4.10. We encode the

grammar such that all elements in **p** are restricted to the choice $\{1, 2, 4\}$ that allows terms to be either, linear, quadratic or quartic. The allowable range of **a** is set from 0.0 to 9.9, meanwhile b is restricted from -5.9 to +5.9 [59]. Both values have an increment of 0.1. Although all three sets of parameters (**p**, **a** and b) are restricted heuristically, such ranges already allow a large number of possible non-linear scoring functions that could have better performance rather than just a linear scoring function. With such restrictions, we can narrow the search space of those parameters by several orders of magnitude. However, the search space still contains hundreds of billions of possibilities which makes Brute-Force search impractical and costly to implement. In the future, a study on determining the best way to restrict the range of a parameter would assist the designer to have better understanding on the possible space of a scoring function.

In our implementation, the scoring function is written in the C++ programming language. Figure 4.12 shows a snippet of C++ source code describing the scoring function. The function called *scoringFunction* takes in three parameters representing $gs_{i,j}$, $dh_{i,j}$ and $sh_{i,j}$. The C++ function returns the score received by a candidate $\theta_{i,j}$ calling the function. The variables $p1$, $p2$, $p3$ in line (4) and the $a1$, $a2$, $a3$, b in line (5) define the constants for $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{evolved}$. the values of $p1$, $p2$, $p3$, $a1$, $a2$, $a3$ and b are assigned in the calls to the functions PA and B on lines (8) and (9). Those functions, as a part of the whole program, map each parameter with an appropriate value input by the user. For example in this case, $p1 = 1$, $a1 = 3.1$, $p2 = 4$, $a2 = 8.4$, $p3 = 2$, $a3 = 4.7$ and $b = -0.6$. Thus, the constants on lines (8) and (9) are the target of our evolutionary search. Finally, the scoring value is calculated by using an equation in line (11). This equation instantiates $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{evolved}$.

The BNF grammar must be structured such that it enables statements in lines (8) and (9) of figure 4.12 to be seeded by restricted values corresponding to parameters of the scoring function. Taking the above code and restricted values into an account, we have developed the BNF grammar shown in figure 4.13 to find the best solution for $f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{evolved}$.

```
1  double scoringFunction(double gs, double dh, double sh)
2  {
3          //Declare paramaters
4          int    p1, p2, p3;
5          double a1, a2, a3, b;
6
7          //Establish parameter values
8          PA(p1,a1,1,3.1); PA(p2,a2,4,8.4);
9          PA(p3,a3,2,4.7); B(b,-0.6);
10
11         //Calculate result
12         double score = a3*(gs^p3) - a2*(dh^p2)
13                      - a1*(sh^p1) + b;
14
15         return score;
16 }
```

Figure 4.12: Snippet of the scoring function source code.

```
1  <expr> ::= PA(p1,a1,<pa_vals>); PA(p2,a2,<pa_vals>);
2             PA(p3,a3,<pa_vals>); B(b,<b_val>);
3
4  <pa_vals> ::= 1,<big_num> | 2,<big_num> | 4,<big_num>
5  <b_val>   ::= <sign><low_num>
6
7  <big_num> ::= <num10>.<num10>
8  <low_num> ::= <num6>.<num10>
9
10 <num10> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
11 <num6>  ::= 0 | 1 | 2 | 3 | 4 | 5
12
13 <sign> ::= 0 | -
```

Figure 4.13: The BNF grammar used to map genotype into a valid scoring function as in line (8) of figure 4.12.

The first line describes a non-terminal <expr> that is structured in the form of the statements in lines (8) and (9) of figure 4.12. <expr> is the start symbol for the production rules of our grammar. <expr> consists of terminal items and another two non-terminals: <pa_vals> and <b_val>. Line (4) outlines <pa_vals> as a pair of p and a values. Three options of p value are given: $1, 2, 4$, while its corresponding a value is determined by non-terminal <big_num>. <big_num> can be decided by selecting decimal numbers from non-terminal

<num10> to incorporate a final value ranging from 0.0 to 9.9. On the other hand, <b_val> determines the value of constant b. Concatenating <sign> and <low_num> non-terminals gives b with a value between -5.9 to 5.9. Given a genotype, the GE mapper maps the above grammar using production rules to build a phenotype (candidate scoring function). The mapping process is described in section 3.3.2 in chapter 3. Note that, the method we use to concatenate numbers is called the digit concatenation technique [25]. Throughout this thesis, we generate the grammar for the evolution of numerical constants based on this method where each single constant is built by concatenating a set of single digits. Other approaches available in literature are ephemeral random constants [73], constant perturbation [117], numerical terminal mutation [1] and linear scaling [64]. Analysis done by [25] shows that the digit concatenation technique outperforms other techniques due to its ability to regularly generate new constants during the evolutionary process and having consistent evolutionary steps towards fitness targets. Furthermore, we choose this technique because a specific range of constant values can be easily set in the BNF grammar based on the fundamental building blocks of digit numbers as in lines (10) and (11) of figure 4.13. Thus, it will not restrict the designer to experiment with various ranges and types (e.g. floating point or integer) of numbers.

Next, the candidate solution must be evaluated with an objective function. This function is designed such that it rewards fast exploration but penalises collisions. Equation 4.11 outlines the objective function used in this experiment.

$$f_{obj} = \frac{\text{cell\_explored}}{1 + (\text{number\_of\_collision} * 0.5)} \tag{4.11}$$

Every time a prospective scoring function is generated, the following evaluation steps are taken:

1. the C++ source code of the scoring function is re-compiled.

2. the exploration strategy with the embedded scoring function is run in a simulated indoor environment. We use a 15m x 15m simulated area as in figure 4.14 to evaluate the performance of the candidate solution. This area is characterised with doorways, narrow paths and small rooms. The UGVs start close together near the top. For each run, simulation time is set to 60 seconds in real-time. We developed the simulator using the Stage platform [125]. We set up the simulation at double-speed to improve evaluation time.

3. After each run, the performance of the prospective scoring function was evaluated using equation 4.11. We have found that, due to non-determinism in the simulation, this objective function was noisy, with the same individual performing quite differently in different runs. To account for this we evaluated each candidate solution twice and choose the minimum fitness of both runs.

4. the fitness is sent to the GE search engine to inform the production of the next generation.

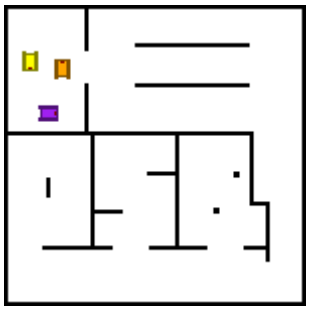


Figure 4.14: Map 1: 15m x 15m indoor-like area. This area is used in evolutionary runs for evaluation purpose.

At the end of a fixed number of generations we retrieve a candidate solution with the highest objective function as the best scoring function. Table 4.1 shows the set up of the evolutionary process for this experiment. One main criterion to set up the number of generations and population size is to estimate the duration needed to complete one evolutionary run. Since simulating a number of exploration tasks is a time-consuming process, an evolutionary run can only be effective if the result can be found within an acceptable time.

Thus, trade-off between run duration and evolution performance is crucial. In this experiment, we made several attempts to run evolution with various pairs of number of generations and population size. The one we choose in the table provided a good balance between run duration and evolution performance. Note that we use Roulette wheel selection to work in these experiments but in later experiments we opted for tournament selection since it is better at handling noisy functions.

Table 4.1: Evolutionary process set up for multi-robot autonomous exploration.

Item	Value
Search engine	Binary string GA
Number of generations	40
Population size	60
Selection type	Roulette wheel
Crossover type	Effective crossover
Mutation type	Point mutation
Probability of crossover	0.95
Probability of mutation	0.01
Elitism	Yes
Simulation time per run	60 secs

4.4 Experimental Results

4.4.1 The Derivation of Handwritten Scoring Function

For performance comparison, a scoring function developed based on handwritten approach is provided. A common approach to handle the design process using this approach is to have repeated trials using several pairs of candidate solutions. Algorithm 17 shows steps taken to derive one good scoring function.

In step (1), one form of scoring function that commonly used in various motion control designs is based on the weighted sum form [42]. Thus in this experiment we choose a linear equation type of the weighted sum function as in equation

4.8. Besides this, explicit knowledge about the state signals used in the scoring function is applied. For example, $sh_{i,j}$ and $dh_{i,j}$ are known as repulsive forces while $gs_{i,j}$ is known as an attractive force. Thus, positive weight is only given to $gs_{i,j}$, while negative weights are assigned to $sh_{i,j}$ and $dh_{i,j}$ as in equation 4.12.

$$f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{handwritten} = a_3(gs_{i,j}) - a_2(dh_{i,j}) - a_1(sh_{i,j}) \qquad (4.12)$$

In step (3), numerical constants typically represent the relative importance of one state signal to another. A state signal that has higher priority is given higher weighting and vice-versa. In this experiment, the effectiveness of the UGVs to navigate safely to their target locations is mainly determined by the sensitivity of the scoring function to $gs_{i,j}$, $dh_{i,j}$ and $sh_{i,j}$. As a rule of thumb, since the safety has higher priority than the navigation pace, $sh_{i,j}$ and $dh_{i,j}$ should have higher weighting than $gs_{i,j}$. After several iteration of the above steps, one good handwritten scoring function is found as in equation 4.14 below.

4.4.2 The Results

The evolutionary process was run for about 45 hours on an Intel i5 quad-core processor with 4GB RAM. 2400 candidate scoring functions were evaluated. The following scoring function as in equation 4.13 was found to be the best result. The result shows that a team of UGVs are able to explore the given

Algorithm 17: Algorithm to derive a handwritten scoring function

1. The designer chooses the structure of scoring function.
2. The designer manually tune numerical constants available in the scoring function.
3. Perform exploration tasks with the candidate scoring function determined in step 1 and 2.
4. If exploration result satisfies, then stop. If not, repeat step 2 with other numerical values, then step 3 and 4.

map completely more quickly and more safely than the handwritten equation shown in equation 4.14.

$$f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{evolved} = 1.4(gs_{i,j})^2 - 4.9(dh_{i,j})^4 - 7.5(sh_{i,j})^4 + 2.0 \quad (4.13)$$

$$f(\theta_{i,j}|sh_{i,j}, dh_{i,j}, gs_{i,j})_{handwritten} = 3.0(gs_{i,j}) - 9.9(dh_{i,j}) - 8.9(sh_{i,j}) \quad (4.14)$$

In order to measure improvement made by the evolutionary process, we compare the evolved scoring function $f_{evolved}$ as in equation 4.13 with a carefully designed handwritten scoring function $f_{handwritten}$. Both scoring functions were validated by setting up UGVs to explore a different environment as depicted in figure 4.15. Each scoring function is given 2 minutes to explore the map. 30 simulation runs were performed for each scoring function in this test.

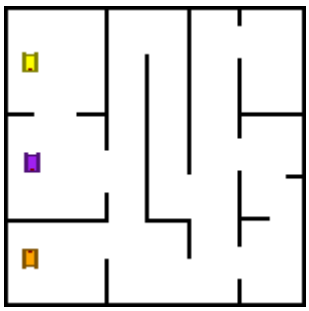


Figure 4.15: Map 2: 15m x 15m indoor-like area. This area is used to test compare the exploration performance between evolved control and handwritten control.

The primary performance indicator is the number of cells explored. To get an overall performance of total explored area, two box plots illustrated in figure 4.16 are presented for each scoring function, respectively. From the box plots, it is clear that the overall performance of $f_{evolved}$ improves significantly over $f_{handwritten}$. By taking into account all results from 60 simulation runs cumulatively, we found that the median explored area acquired by $f_{evolved}$ increases by about 22.2% from $f_{handwritten}$. The maximum and the minimum explored area, each accounts for 9.3% and 50.9% increment, respectively. A significant improvement is confirmed by a statistical T-test analysis where ($p < 0.05$) is obtained (see later for more detailed discussion on the statistical test). In terms of exploration capability, we can say that $f_{evolved}$ is more stable

than $f_{handwritten}$ because the difference between the maximum explored area and the minimum explored area is reduced from $11,779$ to $9,365$ accounted for about 20.5% improvement. In other word, the probability of achieving higher amount of explored area is greater by using $f_{evolved}$ rather than $f_{handwritten}$.

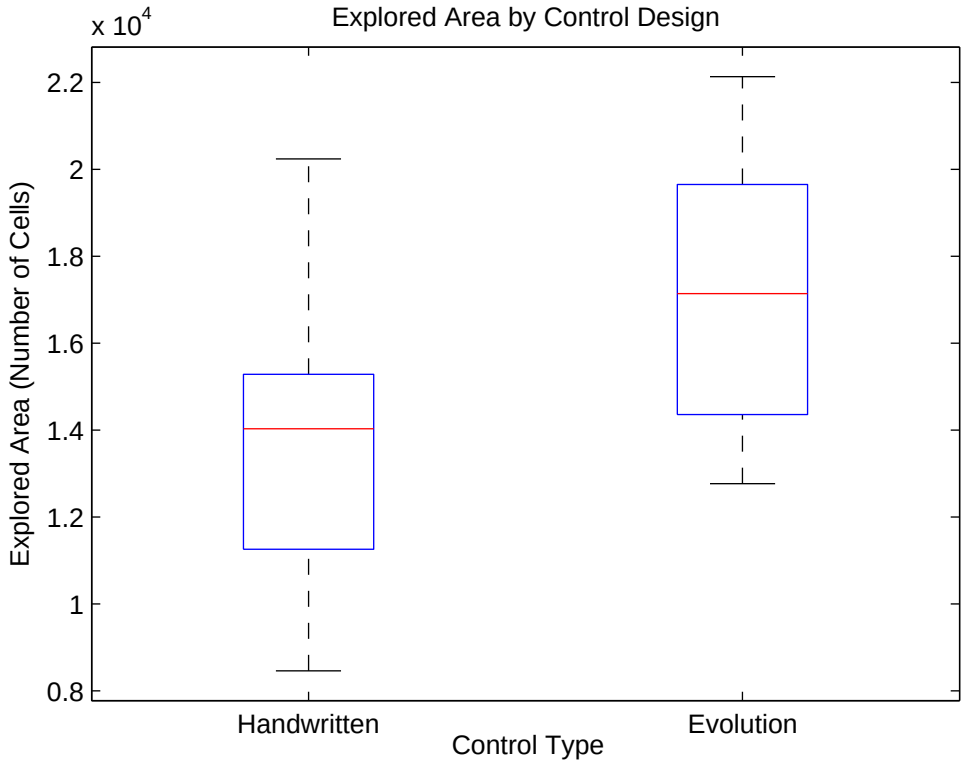


Figure 4.16: Boxplots of explored area between handwritten motion control and evolved motion control.

Table 4.2: Performance indicator for the handwritten motion control and the evolved motion control.

Performance Index	**Control type**	
	Handwritten	**Evolution**
Maximum explored area (cells)	20240	22131
Median explored area (cells)	14030	17139
Minimum explored area (cells)	8461	12766
Total number of entrapments	44	29

The second performance indicator is the number of entrapments that occurred during the exploration task. Consistent improvement by $f_{evolved}$ over

$f_{handwritten}$ can be observed from the simulation runs. In short, by using $f_{evolved}$, UGVs are less likely to get trapped (not moving) due to coordination issues especially in narrow areas. In this test, we can observe that trap situations are not totally eliminated for both scoring functions. However, $f_{evolved}$ has shown its capability for reducing the entrapment problem substantially with good parameter settings. We suspect that the entrapment problem can be overcome with longer evolution perhaps combined with an additional state signal focusing on predicting trap situations. Summary statistics performance indicators for our validation experiments can be found in table 4.3.

An independent-samples T-test was conducted to compare explored area using the handwritten control and the evolved control. There was a significant difference in the scores for $f_{handwritten}$ (Mean=13620, Standard deviation (SD)=2919) and $f_{evolved}$ (Mean=17122, SD=2713) conditions; p-value of this test is 0.000011. These results suggest that the evolution of controller really does have an effect on exploration performance based on the p-value. Specifically, our results suggest that when motion control is evolved, exploration performance increases. Table 4.3 shows the main statistical data between the handwritten motion control and the evolved motion control.

Table 4.3: Main statistical data comparison between the handwritten motion control and the evolved motion control.

Performance Index	**Mean**	**Median**	**Standard Deviation**
Handwritten control area explored	13620 cells	14030 cells	2919 cells
Evolved control area explored	17122 cells	17139 cells	2713 cells
Handwritten control entrapment	1.467	1	0.937
Evolved control entrapment	0.967	1	0.7649

4.5 Discussion

Some important remarks regarding the process of evolving π are discussed in the following. Figure 4.17 shows evolution run performance in terms of fitness values over generations. The best solution obtained in every generation can be observed from the maximum fitness value plotted in the figure (green line). Even though, the number of generations and the size of population were set relatively low (40 generations and 60 individuals), it is encouraging that a well performing individual has evolved after only 31 generations. In the case of evolving a controller for an exploration strategy, it is important to design an evolution set up that can search well using a small number of candidate solutions relative to the number of possible solutions. This is because, with a time-intensive evaluative step each candidate solution, we need to see substantial progress with a relatively small number of evaluations. We posit that part of this fast search performance is due to our carefully designed BNF grammar.

From another perspective, our decision to extend the structure of the scoring function from linear form to polynomial form is also crucial. As can be seen in the result section, we found that the best scoring function of π is in a non-linear form. Indirectly, the result reveals that the relationship between state signals, in our case, might not be optimally expressed in a linear fashion. Thus, by extending the scoring function into a non-linear form, we make it possible for roboticists to present a new state signal as needed without worrying about how the new state signal will be integrated into the whole control structure. The main reason behind this is that we left the problem of finding the best configuration of a controller to the evolutionary process.

4.6 Conclusion

In this experiment, our main intention was to utilise our simulation results as a proof-of-concept that shows GE's capability for finding improved solutions

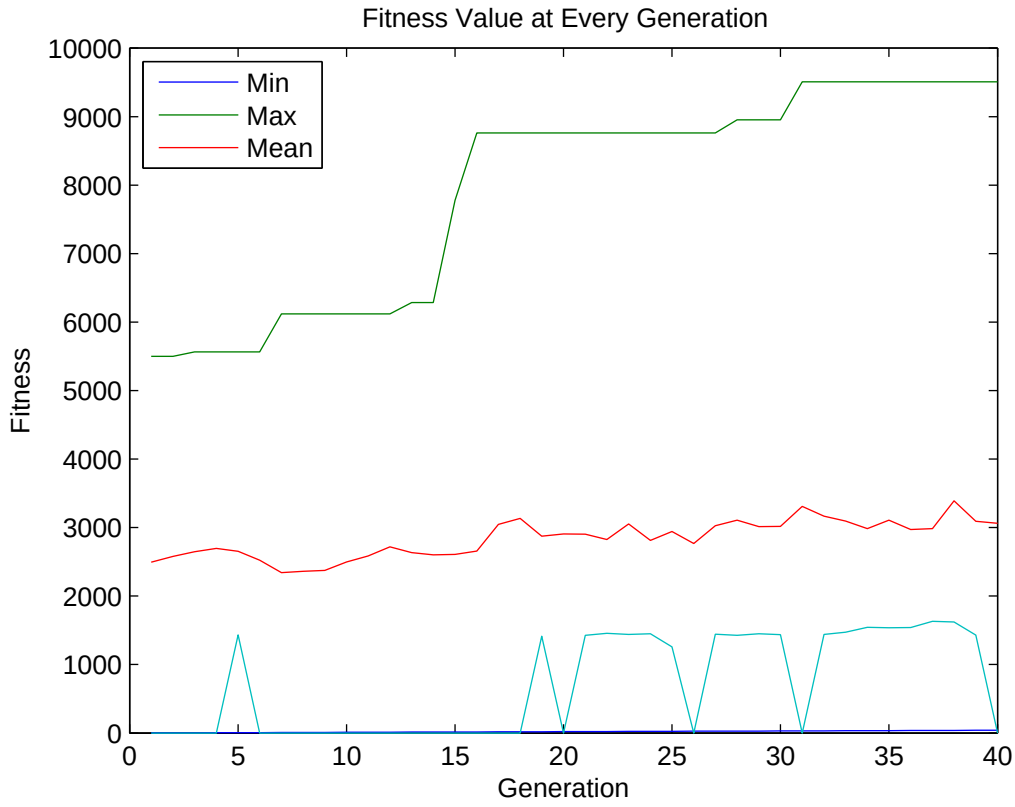


Figure 4.17: Evolution run performance in terms of fitness values at each generation.

for a scoring function of π for coordinated multi-robot exploration. We found that the evolved π can perform comparably with manually-designed handwritten π even on short evolutionary runs. By using the best evolved π, UGVs managed to exhibit natural behaviour by negotiating their motion when they are close to each other without collision while pursuing their own target location. The results also revealed that the exploration performance improved by the selection of a non-linear scoring function.

5 Single Robot Multi-Objective Exploration

In this chapter, we evolve motion control for a single robot platform with multi-objective exploration. The motivation of this experiment is to apply a GE-based optimisation technique to another exploration configuration that requires the robot to satisfy multiple objectives. The objectives may conflict with each other. Thus, a well configured exploration system including motion control must be designed to balance these objectives. In this experiment, we consider a case where the exploration mission requires the robot to optimise navigational safety, effectiveness (navigation speed) and power consumption. Such a mission is important and widely applicable to power-constrained exploration. Thus, finding a motion plan that can satisfy these three conflicting objectives simultaneously is challenging. As such, we use GE to find the best relationship between these objectives to increase exploration performance. GE is used for evolving motion control that receives three inputs corresponding to each exploration objective. We then compare the evolution performance of GE with a well-known numerical evolution technique named as CMA-ES [52] and with a handwritten design. The results demonstrate that exploration performance of the GE-based controller is on a par with the CMA-ES-based controller, and outperforms the handwritten controller significantly.

5.1 Objectives

The main objectives of this experiment are:

1. to demonstrate the applicability of GE to evolve motion control structure for single robot multi-objective autonomous exploration.

2. to compare exploration performance of controllers designed using GE and CMA-ES, as well as a handwritten controller.

5.2 The Exploration Framework

The experiment described in this chapter utilises the robotic exploration framework discussed in chapter 4 with some modifications to fulfill the requirements of the new exploration configuration. We retain the following components of the framework:

1. Robot kinematics follow a wheeled skid-steer configuration of a UGV as described in section 4.2.1.

2. The framework adopts the Frontier-based exploration strategy as depicted in section 4.2.2 and 3.1.

Major changes are made to the Decision-Theoretic Motion Control (DTC) of the exploration framework as follows:

1. State signals - Three state signals (input) are considered i.e. power consumption, static hazard and goal strength.

2. Action signal - Short-range target location is used as the action signal (output), replacing steering direction.

3. Policy - A new scoring function structure is designed to correlate with the changes of state signals and action signal. Hence, our main aim is to evolve the scoring function of this exploration configuration.

5.2.1 Motion Control Structure

There are significant changes being made to the DTC structure. First, the action state of the DTC is determined as a short-range target location p_d. In each controller cycle f_C, p_d is selected from a set of candidate target points surrounding the robot. The best p_d is chosen to be at a distance that the robot can reach in a few seconds, usually in the order of a metre depending on the terrain [60]. This selection mechanism is the aim of our evolutionary process that will be elaborated in section 5.3. A low level controller is responsible to convert the selection result into a set of velocity values. Second, a power consumption state signal is introduced such that the DTC can estimate the power needed to move the UGV from its current pose to a candidate pose of p_d. The power consumption state signal is estimated by utilising a power model of a skid steered wheeled ground vehicle as described in [21].

For an overview of the DTC, consider figure 5.1. The overall process flow of the DTC consists of three main tasks. These are: i) the generation of candidate short-range target locations, ii) the establishment of state signals, and iii) scoring the candidates.

Generating Candidate Short-Range Target Locations

Assume the current position of the robot is given by $p_c = (x_c, y_c, \alpha_c)$ where (x_c, y_c) is the coordinate of the centre of the robot and α_c is the current heading of the robot in the global frame. A number of candidate target locations are set up as a circle of points at a pre-determined distance D from (x_c, y_c). Each point is separated by a pre-determined angular resolution β. The position of each candidate short-range target location, $p_j = (x_j, y_j, \alpha_j)$, where $j = 0, 1, ..., N$ and N is the number of candidate points, can be calculated with equations 5.1 to 5.3.

$$\theta_j = \alpha_c + \beta * j \tag{5.1}$$

$$x_j = D * cos(\alpha_j) \tag{5.2}$$

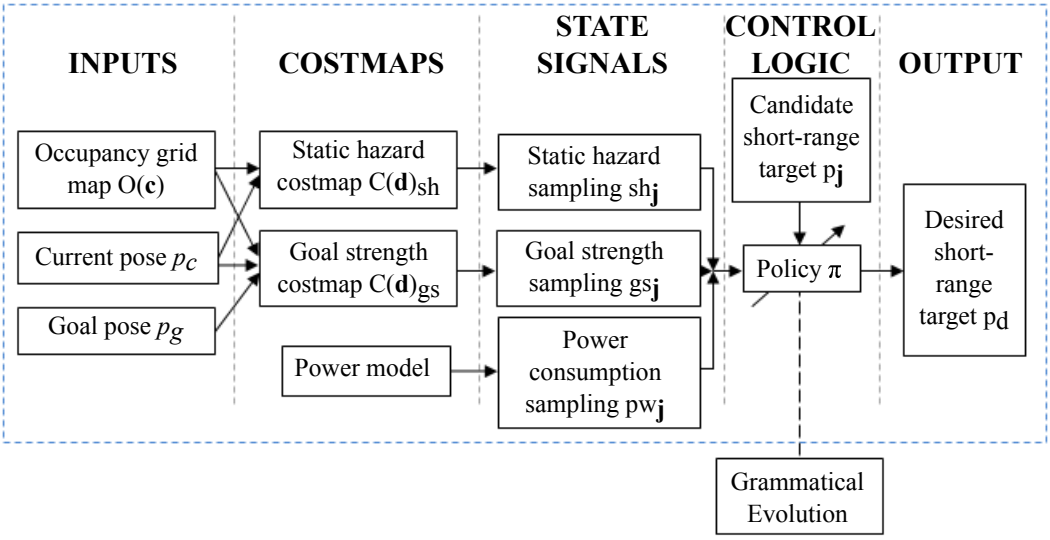


Figure 5.1: The motion control structure of a UGV in single robot multi-objective exploration set up. The DTC receives three types of input from the exploration framework: a global occupancy grid map $O(\mathbf{c})$, robot's current pose p_c and its goal pose p_g. The DTC builds a set of costmaps from these inputs. The costmaps are used in the sampling process of state signals on each possible short-range target $p_{\mathbf{j}}$, thus producing a set of navigational options. Finally, π evaluates all the navigational options and chooses the short-range target p_d. This process occurs within the timeframe of the controller cycle:f_C. GE is used to optimise π.

$$y_j = D * sin(\alpha_j) \tag{5.3}$$

In this experiment, we set $\beta = 5°$ and $D = 1$ metre. Our choice of β represents a reasonable trade-off between fine-grained navigational choice and computational processing time that yields 72 candidate points at each f_C. For an illustration, figure 5.2 shows an example of 16 potential p_js as the best target location p_d given the current pose of the robot is at $p_c = (0, 0, 0)$.

Establishing State Signals

A total of three state signals are generated for each candidate point p_j. Two existing state signals from the previous chapter are used in this experiment: a

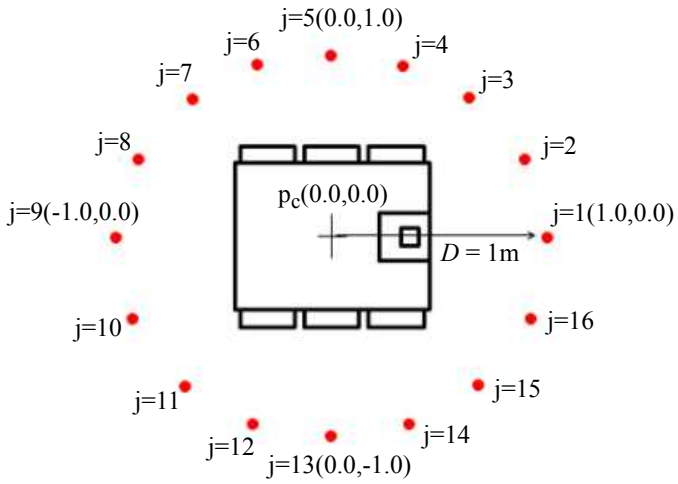


Figure 5.2: An example of 16 candidate short-range target locations surrounding the robot. Given the current pose of the robot is $p_c = (0, 0, 0)$, the next possible pose of the robot can be chosen from these locations.

static hazard state signal sh_j and a goal strength state signal gs_j. In addition, a new signal is introduced: power consumption state signal pw_j. The process to establish each state signal is explained below.

Static hazard state signal - The static hazard state signal sh_j measures the safety of a UGV from collision with static obstacles. We improve the calculation of sh_j by adopting the Separating Axis Theorem (SAT) technique [33] as the collision detection algorithm. The SAT technique improves collision detection accuracy by testing the intersection between the line of a UGV's edge and an obstacle. This mechanism provides better collision detection than checking only the UGV's corner points. Algorithm 18 computes sh_j for a candidate p_j.

Goal strength state signal - To measure the possible progress made by the UGV towards a long-range goal location by moving through a p_j, goal strength state signal gs_j is calculated. Given a goal strength costmap $C(\mathbf{d})_{gs}$ is available, gs_j of a p_j can be calculated as shown in Algorithm 19.

Power consumption state signal - The power needed to move the UGV from

Algorithm 18: Algorithm to sample sh_j of a p_j.

1 Apply the Separating Axis Theorem technique to check for possible collision during turning from α_c to α_j.

$$sh_j = \begin{cases} 1 & \textit{if collide} \\ 0 & \textit{otherwise} \end{cases}$$

2 If no collision in step 1, then check collision for moving forward towards (x_j, y_j).

$$sh_j = \begin{cases} 1 & \textit{if collide} \\ 0 & \textit{otherwise} \end{cases}$$

3 If no collision in step 1 and 2, then calculate sh_j considering the cost-distance $\mathbf{d}_j$ of (x_j, y_j) in $C(\mathbf{d})_{sh}$ field.

$$sh_j = [\frac{1}{mn - mx}\mathbf{d}_j - (\frac{mx}{mn - mx})] \in [0, 1] \tag{5.4}$$

where, mn is the lower bound value in $C(\mathbf{d})_{sh}$ before collision happens and mx is the upper bound value in $C(\mathbf{d})_{sh}$ before the robot moves to safe area. sh_j has range between 0.0 to 1.0. Higher sh_j value means that p_j has higher hazard and vice-versa.

Algorithm 19: Algorithm to sample gs_j of a p_j.

1 Get cost-distance $\mathbf{d}_j$ of a candidate point cell (x_j, y_j) from $C(\mathbf{d})_{gs}$.
2 Get cost-distance $\mathbf{d}_c$ of a current UGV pose cell (x_c, y_c) from $C(\mathbf{d})_{gs}$.
3 Calculate raw goal strength value, gs_{raw}.

$$gs_{raw} = \frac{\mathbf{d}_c - \mathbf{d}_j}{D} \in [-1, 1]$$

4 Normalise gs_{raw} to get gs_j.

$$gs_j = 0.5 * gs_{raw} + 0.5 \tag{5.5}$$

p_c to a p_j can be estimated from the power modelling of a skid-steer vehicle [21]. Exact power consumption is difficult to predict due to several factors such as skidding effect and terrain surface. In such cases, we simplified our estimation by assuming the UGV has turning energy consumption proportional to the angle of turn and operates on one terrain surface only. Generally, the

power consumption of moving from p_c to p_j is expressed as the sum of power needed to turn the UGV from α_c to α_j and forward movement from (x_c, y_c) to (x_j, y_j). For the power consumption state signal pw_j calculation of a p_j, we set up pw_j as a numerical value representing power consumption relative to the maximum estimated power pw_k, where k is the candidate point with the longest movement (both in turning and forward movements). Algorithm 20 denotes the pw_j calculation of each candidate j.

Algorithm 20: Algorithm to sample pw_j of a p_j.

1 Calculate the estimated power used to skid-steer the UGV from direction α_c to direction α_j.

$$p\_turn_j = MR * |\alpha_c - \alpha_j|$$

where MR is the moment of resistance.

2 Calculate an estimated power consumed by the UGV to move forward from (x_c, y_c) to (x_j, y_j).

$$p\_fwd_j = mass * a_t + FR * v_t$$

where $mass$ is the UGV's mass in kg, a_t is the UGV's acceleration, v_t is the UGV's translational velocity and FR is the longitudinal resistance.

3 Finally, calculate pw_j as follows:-

$$pw_j = \frac{p\_turn_j + p\_fwd_j}{pw_k} \tag{5.6}$$

where pw_k is the maximum estimated power consumption combining the maximum p_turn_k and p_fwd_k. pw_j has range between 0.0 to 1.0. The higher pw_j value the higher power needed to move to p_j and vice-versa.

The Policy

As the control structure receives significant changes, the importance of revising π of a DTC is inevitable. A revised π is required to ensure the exploration performance exhibited by the UGV is not degraded and is able to cope with the current exploration mission. As with the previous experiment, we evolve the scoring function of π to guide motion control. This time, the scoring function

evaluates the quality of a p_j by considering the three above-mentioned state signals: gs_j, sh_j and pw_j. The scoring function, $f(p_j|sh_j, pw_j, gs_j)$ returns a positive value for a good p_j. A return value of less or equal to 0.0 for a candidate point p_j suggests that the point is unattractive to the UGV. In this experiment, we design the structure of the scoring function with the format of equation 5.7. This structure was selected after a number of manual trials to form a good scoring function including a standard weighted sum function.

$$f(p_j|sh_j, pw_j, gs_j) = (1 - g(sh_j)) * (1 - h(pw_j)) * m(gs_j) \tag{5.7}$$

$g(sh_j)$ is a quadratic function relating to sh_j as follows:-

$$g(sh_j) = a_1 * (sh_j^{b_1}) \tag{5.8}$$

$h(pw_j)$ and $m(gs_j)$ are logistic functions relating to pw_j and gs_j, respectively:-

$$h(pw_j) = \frac{1.0}{1.0 + \exp(-a_2 * (pw_j - b_2))} \tag{5.9}$$

$$m(gs_j) = \frac{1.0}{1.0 + \exp(-a_3 * (gs_j - b_3))} \tag{5.10}$$

By considering equations 5.7 to 5.10, there are six numerical parameters that need to be determined by search. These parameters are a_1, b_1, a_2, b_2, a_3 and b_3. Next, we describe the evolution of these parameters using GE and CMA-ES to get the best tuned scoring function.

5.3 The Evolutionary Process

In this section, we describe in detail the set up of the evolutionary process performed in the following experiment. We present an objective function uniquely used to evaluate multi-objective exploration combining total explored area, power consumption and collision occurrence as objectives to consider. Then, we discuss the set up needed to evolve π using both GE and CMA-ES.

5.3.1 Objective Function

The evaluative process of a candidate solution follows the same rules from the previous experiment. Briefly, three steps: embedding, simulation and collation are taken to return a fitness value to a calling evolutionary search engine as depicted in figure 5.3.

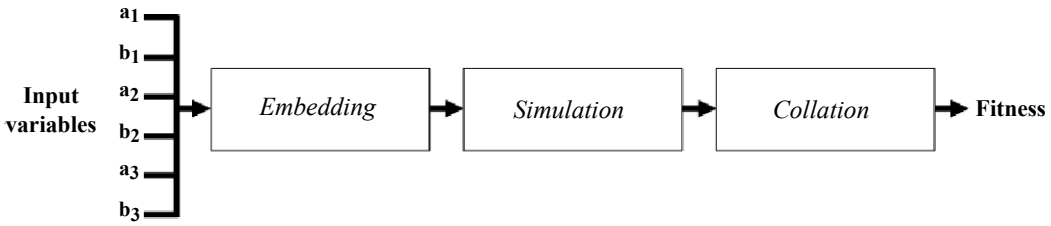


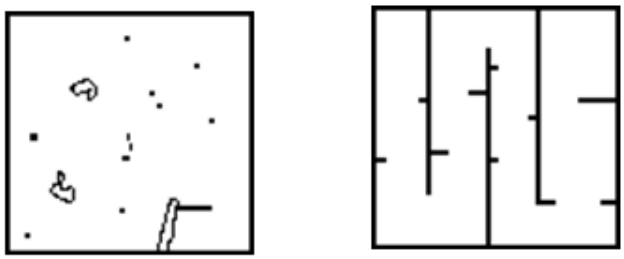
Figure 5.3: Outline of the evaluative process of a candidate π. The first step is to embed a candidate π into UGV motion control (*Embedding*). Then, simulation of exploration is performed (*Simulation*). Finally, a fitness value is calculated using a given objective function (*Collation*).

1. *Embedding* - A candidate of six numerical parameters yielded by the evolutionary search engine is embedded into the structure of motion control by substituting into the source code of the scoring function of π. The source code is re-compiled.

2. *Simulation* - Simulation is executed to model the performance of the UGV with a candidate π. For better environmental coverage, the UGV is tested on the two maps featured in figure 5.4. The first map emulates a planetary environment with scattered obstacles of different shapes and sizes. The second map represents a more artificial maze-like environment with a highly constrained path for exploration. The simulation runs at an improved ten-times speed up in Stage simulator.

3. *Collation* - For each simulation run, equation 5.11 is used as the objective function to calculate the fitness value of a candidate solution:

$$f_{obj} = \left(A^2 + \frac{1.0}{1.0 + P}\right) * \left(\frac{1.0}{1.0 + C}\right) \tag{5.11}$$



(a) (b)

Figure 5.4: The first (a) and second (b) learning environments for simulation.

where A is the total area explored in terms of the number of cells explored. Two penalties are added. P is the total power consumed in the range between zero and ∞; and C is the number of times the robot collides with obstacles. This equation is designed so that individuals (candidate solutions) that explore a large proportion of the map with relatively low power requirements and no collisions are rewarded with a high fitness value. Note that, the P term is very small compared to A term. This means that area explored takes precedence and power consumption is important when all other things are equal. Average fitness for each individual is calculated from several simulation runs.

5.3.2 Grammatical Evolution

The framework of GE previously discussed in chapter 3 is our reference point. In this experiment, we use GE once again solely for evolving numerical constants i.e. six numerical parameters in the scoring function. Firstly, these parameters are restricted to allowable values as follows: $a_1 \in [0.00, 9.99]$; $b_1 \in [0.00, 9.99]$; $a_2 \in [0.00, 29.99]$; $b_2 \in [0.00, 0.99]$; $a_3 \in [0.00, 29.99]$; $b_3 \in [0.00, 0.99]$; where within each range we have a discrete step size of 0.01 for candidate values. These ranges give a total search space of nearly 10^{15} different possibilities. Again, as mentioned in section 4.3, such a restriction is set to give wide diversification of linear and non-linear functions as candidate solutions. Note that, we use high resolution of step size at 0.01 to show the

capability of GE to handle a large search space. Thus, it is expected to produce more accurate solution at the end of the evolution.

Table 5.1 shows the set up of the evolutionary process for this experiment. As in the previous experiment, most of the parameters are manually determined by taking into consideration estimated total duration taken to perform a number of simulated explorations. As mentioned earlier in chapter 3, we use Tournament selection in this set up as it is robust in the presence of noisy fitness function [87].

Table 5.1: Evolutionary process set up for single robot multi-objective exploration.

Item	Value
Search engine	Binary string GA
Number of generations	100
Population size	30
Selection type	Tournament
Crossover type	Effective crossover
Mutation type	Point mutation
Probability of crossover	0.95
Probability of mutation	0.01
Elitism	Yes
Simulation time per run	60 secs real-time (6-10 secs after speeding up)

To allow GE to search within allowable values of parameters to be evolved, an appropriate BNF grammar is required. A BNF grammar as in figure 5.5 was developed to meet this purpose. In conjunction, the C++ source code for scoring function program as shown in figure 5.6 is used.

The BNF grammar establishes lines (7) and (8) of the scoring function code (figure 5.6) with appropriate numerical values. This is done by using the genotype provided by the GA search engine to translate a set of non-terminals in figure 5.5 into valid terminals. A complete mapping process from non-terminals to terminals is defined in section 3.3. Note that, <*pa_sh*> in line (4) of figure 5.5 refers to the parameters of the $g(sh_j)$ quadratic function. Meanwhile, <*pa_pw*> and <*pa_gs*> in line (5) and (6) represent the parameters of the $h(pw_j)$ and $m(gs_j)$ logistic functions, respectively.

```
<expr> ::= PA(a1,b1,<pa_sh>); PA(a2,b2,<pa_pw>);
           PA(a3,b3,<pa_gs>);

<pa_sh> ::= <num_type3>,<num_type3>
<pa_pw> ::= <num_type1>,<num_type2>
<pa_gs> ::= <num_type1>,<num_type2>

<num_type1> ::= <num3><num10>.<num10><num10>
<num_type2> ::= 0.<num10><num10>
<num_type3> ::= <num10>.<num10><num10>

<num10> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<num3>  ::= 0 | 1 | 2
```

Figure 5.5: The BNF grammar used to map genotype into a valid scoring function.

```
double scoringFunction(double gs, double sh, double pw)
{
  //Declare paramaters
  double a1, a2, a3, b1, b2, b3;

  //Establish parameter values
  PA(a1,b1,3.15,4.11); PA(a2,b2,29.80,0.92);
  PA(a3,b3,15.77,0.75);

  //Calculate result
  double score =
  (1-g(sh,a1,b1))*(1-h(pw,a2,b2))*(m(gs,a3,b3));

        return score;
}
```

Figure 5.6: Snippet of the scoring function source code.

5.3.3 Covariance Matrix Adaptation Evolutionary Strategy

We compare our GE-based approach for evolving motion control with a CMA-ES-based approach. We choose CMA-ES as the benchmark because the algorithm has shown good performance when dealing with numerical optimisation of noisy or ill-conditioned problems. As such, we can use results from CMA-ES as an indicator to help validate the performance of our GE search. It should be noted however that CMAES is restricted to

search in numerical domains, in contrast, GE can be extended to any domain representable by a BNF grammar.

In this experiment, we set CMA-ES to evolve the values of the six parameters in the scoring function. We initialised these parameters $(a_1, b_1, a_2, b_2, a_3, b_3)$ by harvesting the values from the hand-coded scoring function as the starting mean vector $(1.25, 1.00, 15.00, 0.50, 6.00, 0.80)$ and a Gaussian distribution that is uniform in all directions with a standard deviation of 1.0. Again, the population size was set to 30 running for 100 generations with the best 50% of the population being retained on each run [60]. Note that, the handwritten π was carefully designed prior to the evolutionary runs by a repetitive trial-and-error approach to get a reasonable solution.

5.4 Experimental Results

In this section, we present the experimental results of the following scoring functions:

1. The handwritten scoring function of π (HW-SF).

2. The GE-based scoring function of π (GE-SF).

3. The CMA-ES-based scoring function of π (CMAES-SF).

The first scoring function (HW-SF) represents the conventional approach of handwritten design as a basis for comparison. The derivation of HW-SF follows procedures depicted in section 4.4.1. HW-SF in this case is considered as a reasonable solution because no collision occur at all exploration runs when the UGV uses this scoring function. The latter two scoring functions (GE-SF and CMAES-SF) are gathered from evolution using GE and CMA-ES, respectively. All of our experiments were run on an IBM HS22 server with a 3.47GHz eight core Intel Xeon X5677 CPU and 48GB of memory. Note that, we used this

machine for the purpose of running several experiments in parallel and to increase simulation speed [1]. Equations 5.12 to 5.14 present the best scoring function found for each design approach:

HW-SF:

$$\begin{aligned} g(sh_j) &= 1.25 * (sh_j^1) \quad \in [0,1] \\ h(pw_j) &= \frac{1}{1+\exp(-6.0*(pw_j-0.8))} \quad \in [0,1] \\ m(gs_j) &= \frac{1}{1+\exp(-15.0*(gs_j-0.5))} \quad \in [0,1] \\ f(p_j|sh_j,pw_j,gs_j) &= (1-g(sh_j))*(1-h(pw_j))*m(gs_j) \end{aligned} \tag{5.12}$$

GE-SF:

$$\begin{aligned} g(sh_j) &= 3.15 * (sh_j^{4.11}) \quad \in [0,1] \\ h(pw_j) &= \frac{1}{1+\exp(-29.8*(pw_j-0.92))} \quad \in [0,1] \\ m(gs_j) &= \frac{1}{1+\exp(-15.77*(gs_j-0.75))} \quad \in [0,1] \\ f(p_j|sh_j,pw_j,gs_j) &= (1-g(sh_j))*(1-h(pw_j))*m(gs_j) \end{aligned} \tag{5.13}$$

CMAES-SF:

$$\begin{aligned} g(sh_j) &= 2.41329 * (sh_j^{2.33206}) \quad \in [0,1] \\ h(pw_j) &= \frac{1}{1+\exp(-10.53519*(pw_j-1.21823))} \quad \in [0,1] \\ m(gs_j) &= \frac{1}{1+\exp(-15.98139*(gs_j-1.38061))} \quad \in [0,1] \\ f(p_j|sh_j,pw_j,gs_j) &= (1-g(sh_j))*(1-h(pw_j))*m(gs_j) \end{aligned} \tag{5.14}$$

Our experimental results measure the UGV's exploration performance on a different environment featuring similar features to the learning maps. To

[1] Though it should be noted that each individual simulation still ran on a single core.

this end, we compare the evolved scoring functions of π (GE-SF, CMAES-SF and HW-SF) on the new environment shown in figure 5.7(a) to validate the performance of the UGV in meeting exploration objectives. Summary statistics were collected from 30 separate runs of each scoring function (up to eight of which were run simultaneously). Figure 5.7(b), 5.7(c) and 5.7(d) examples of the exploration area covered by the UGV within the 800 seconds time-limit using HW-SF, GE-SF and CMAES-SF settings, respectively.

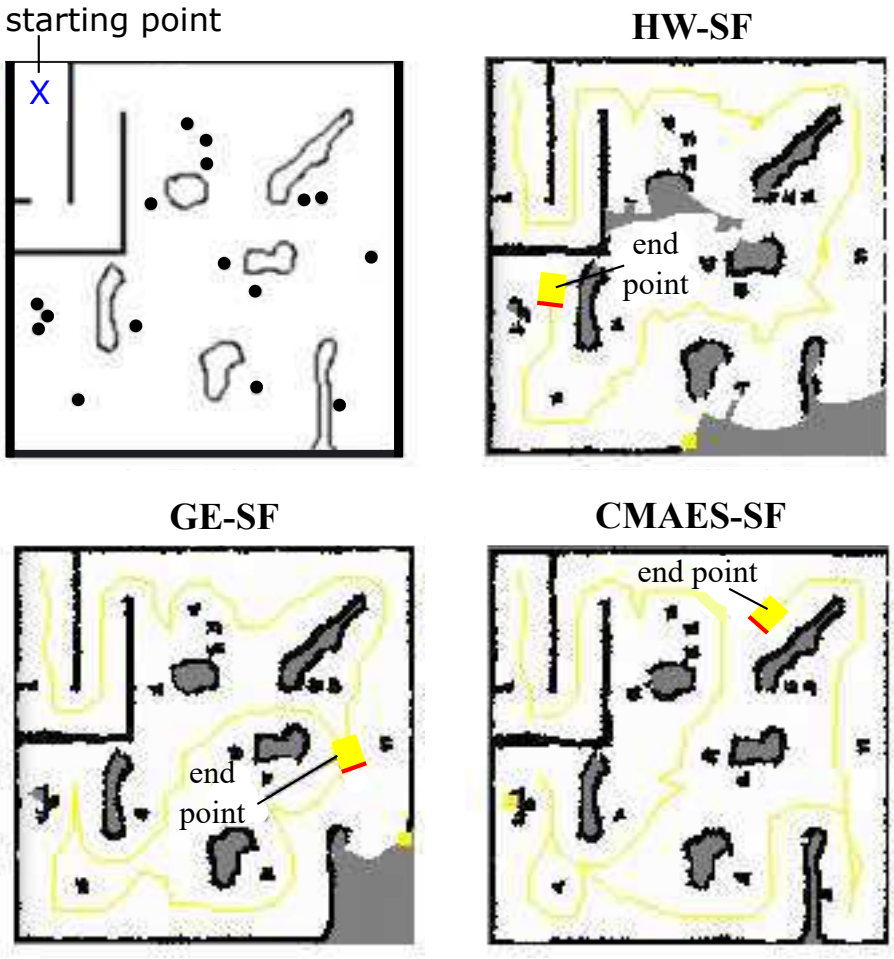


Figure 5.7: (a) 15m x 15m validation map (b) Exploration area HW-SF (c) Exploration area GE-SF (d) Exploration area CMAES-SF.

We compared scoring functions in terms of (i) exploration coverage, and (ii) average power consumption. Note that, no collision occurred in any of these simulations showing that all scoring functions are working reliably to stop the UGV from moving to close to obstacles.

Exploration Coverage

Figure 5.8 shows box plots presenting exploration coverage of the scoring functions plotted from all runs on the the environment shown in figure 5.7. Exploration coverage is measured in meter-squared unit (m^2). These box plots visualise the range and distribution of scoring functions under study. The median exploration coverage of HW-SF is 186 m^2. GE-SF has a higher median than the HW-SF at 213 m^2 of coverage, while CMAES-SF shows the highest median of 214 m^2. In brief, we can observe that the exploration coverage improved by about 14.5% and 15.0% respectively for GE-SF and CMAES-SF.

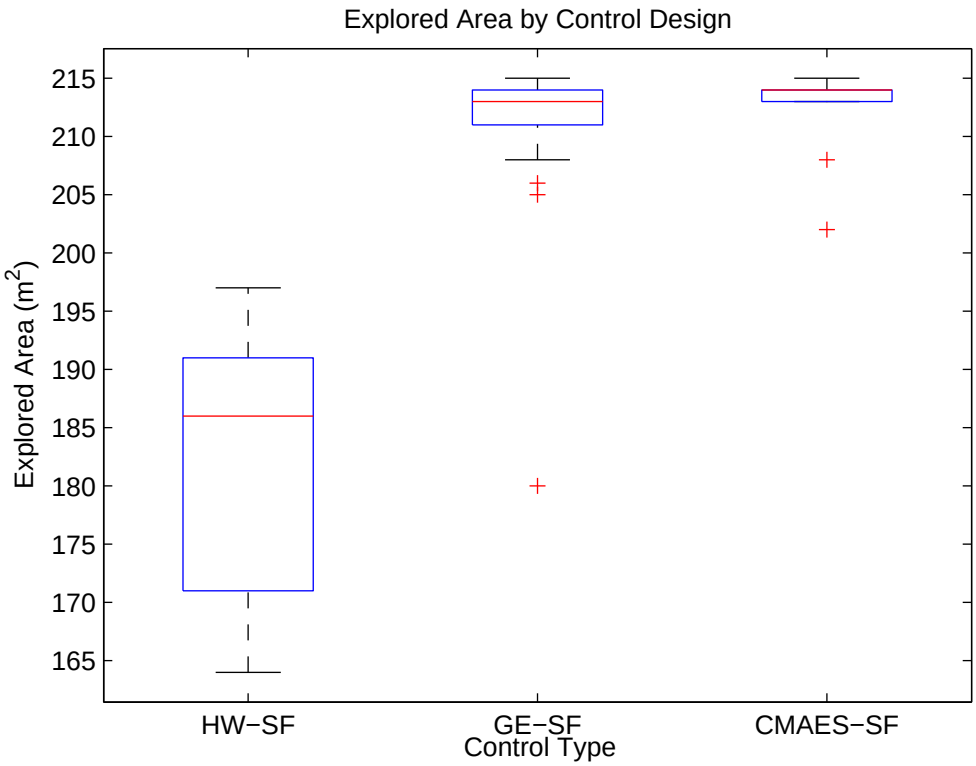


Figure 5.8: Box plots presenting exploration coverage of scoring functions.

The box plots also show that there is a clear performance difference between HW-SF and evolved scoring functions (GE-SF and CMAES-SF). This is supported by the lower quartile data from box plots of GE-SF and CMAES-SF

in which it suggest at least 75% of runs from GE-SF and CMAES-SF are better than the best possible performance of HW-SF (top whisker).

In terms of box plots distribution, we can see that the box plot of HW-SF is comparatively tall with standard deviation (SD) of $10.34m^2$, while the box plots of GE-SF and CMAES-SF are comparatively short ($SD = 2.47m^2$ and $SD = 1.23m^2$, respectively). The distributions demonstrate that HW-SF holds wider variance of exploration performance compared to GE-SF and CMAES-SF. The readings indicate that the HW-SF performance is less consistent than GE-SF and CMAES-SF.

By using the same box plots, we can also compare the performance of GE-SF with the performance of CMAES-SF. The box plots of these evolved scoring functions are skewed to the high end which suggests the exploration coverage is likely to be high. However, we observe that CMAES-SF performance is more consistent than that of GE-SF. One factor in favour of CMAES-SF is that CMA-ES can evolve numerical parameters more precisely. However, inspection of equations 5.13 and 5.14 shows that corresponding constants are different enough that better fine-tuning is unlikely to be the only factor. Another factor in favour of CMAES-SF is that it starts with a population centred around a good handwritten solution. In any case the results indicate that GE can evolve numerical parameters that produce results that almost at par with CMA-ES. While CMA-ES is mainly developed to evolve numerical parameters, GE is more generic in evolving various data types that gives GE the ability to improve the robot navigation by finding a new structure for π as will be shown in chapter 6 and 7.

To test for significance we performed a One-way Analysis of Variance (ANOVA) test to analyse the differences between three scoring function means. Table 5.2 and 5.3 tabulate exploration area data and the ANOVA test result, respectively. From table 5.3, there is a statistically significant effect of evolution of π's scoring function at the $p<0.05$ level for the three conditions $[F(2, 87) = 112.79,\ p = 6.89E - 25]$. Post hoc multiple comparison using the Tukey-Kramer criterion shows that the mean score for HW-SF($Mean = 182.77$) are significantly different from GE-SF($Mean =$

209.10) and CMAES-SF($Mean = 213.07$).

Table 5.2: Data of area explored (m^2) in 30 runs and its mean ($\bar{\mathbf{X}}$) for each tested π of the UGV performing exploration task.

Run	HW-SF	GE-SF	CMAES-SF
1	171	214	213
2	191	214	214
3	189	212	213
4	189	180	214
5	196	180	213
6	195	214	202
7	170	214	214
8	171	180	208
9	183	208	214
10	169	214	214
11	176	205	214
12	184	215	214
13	190	212	214
14	169	213	214
15	196	212	214
16	181	214	214
17	197	212	215
18	192	206	215
19	188	211	214
20	193	214	214
21	169	214	214
22	171	213	214
23	167	214	214
24	164	211	214
25	184	211	208
26	189	214	214
27	188	212	214
28	188	213	214
29	181	213	214
30	192	214	215
$\bar{\mathbf{X}}$	182.77	209.10	213.07

Table 5.3: One-way ANOVA and post hoc multiple comparison results to analyse the variation arises within or among three scoring functions for performance on exploration area.: HW-SF, GE-SF and CMAES-SF using samples from Table 5.2.

Control Type (A)	Mean of A	Standard Deviation of A	Control Type (B)	Mean Diff. (A-B)	95% Confidence Interval	
					Lower Bound	Upper Bound
HW-SF	182.77	10.34	GE-SF	-26.33	-31.56	-21.10
GE-SF	209.10	2.47	CMAES-SF	-3.97	-9.20	1.26
CMAES-SF	213.20	1.23	HW-SF	-30.30	-35.53	-25.07

Power Consumption

Another main exploration goal is to minimise power consumption during exploration. To achieve this, we have designed the evolution of the scoring function to take into account the effect of power usage by modelling the power consumption associated with UGV motion. Here, we highlight the effectiveness of evolutionary techniques in minimising UGV power consumption by measuring the average size of new area in m^2 that can be explored with 1 Joule (J) of energy consumed. We use map of figure 5.7(a) as the test-bed environment. This power efficiency calculation is measured in $\frac{m^2}{J}$. Figure 5.9 shows the box plots of power efficiency vs the scoring functions.

By observing the median of each box plot, HW-SF has the highest median with 0.037 $\frac{m^2}{J}$. CMAES-SF comes next with 0.036 $\frac{m^2}{J}$. Meanwhile, GE-SF produces the least median with 0.032 $\frac{m^2}{J}$. One possible reason CMAES-SF and HW-SF have the median lower than HW-SF is because to fully explore the environment, the UGV might need to take paths with some areas are already explored before. However, the standard deviation of CMAES-SF ($SD = 4.27E-4$) and GE-SF ($SD = 2.13E-3$) are better than HW-SF ($SD = 3.08E-3$). This indicates that the power efficiency of both evolved scoring functions are more consistent and predictable than HW-SF.

Overall we see a significant improvement in exploration performance from both

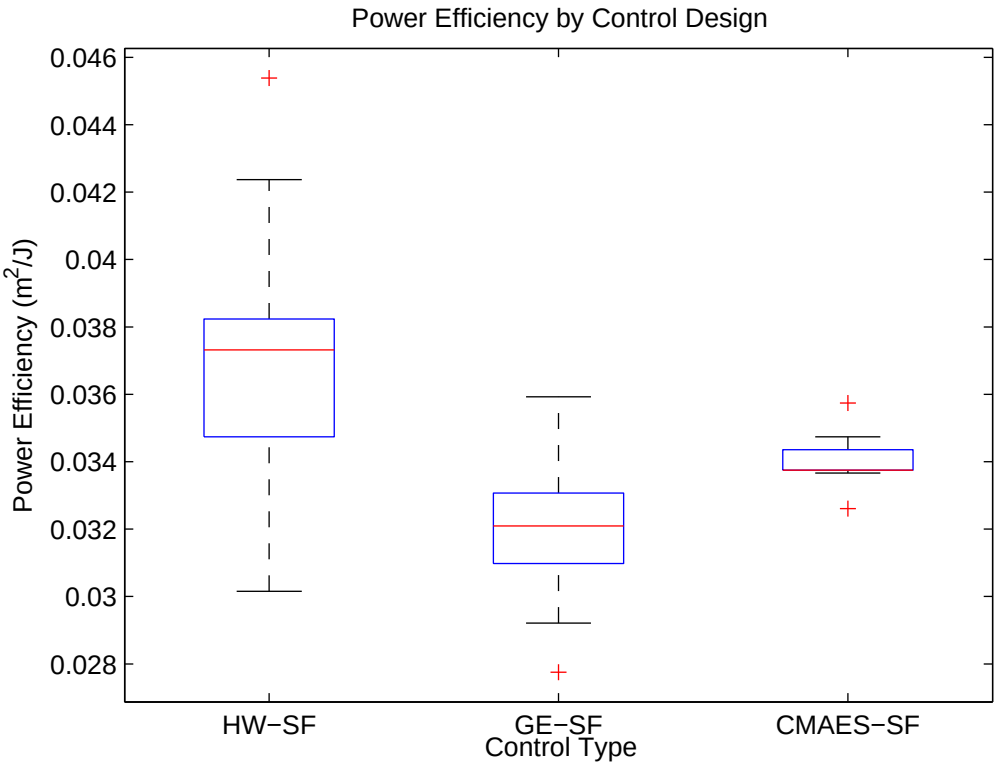


Figure 5.9: Box plots presenting power efficiency of scoring functions.

GE-SF and CMAES-SF. In addition, while we see that they are slightly less efficient for each new area explored they show more consistent behaviour. The weaker results for energy consumption for the evolved functions could be due, in part, to the lower priority for this objective in the context of our single objective function [2]. Nonetheless, even with a single weighted objective function our framework establishes a trade-off between objectives and we explore this next in our discussion section.

Even though the improvement seems very minimal, the benefit is real. This is because GE and CMA-ES techniques have shown their capability to trade-off between exploration coverage and power efficiency. This is done through concatenation of multiple exploration objectives in a single objective function as in equation 5.11. In this case, since power consumption is the secondary

[2] In future research we could explore a more equal treatment of this goal by using a multi-objective meta-heuristic [24, 135].

objective over exploration coverage, the trade-off between both objectives seems practical.

5.5 Discussion

We have examined motion control policies with various scoring function configurations. Our empirical data shows that the configuration of the π function is a key component that influences exploration performance. As part of good design practice, we want to perform further analysis to understand why a particular π has better behaviour than others. The explicit structure of the scoring function of π provided by our approach makes analysis of the controller's behaviour straightforward. It should be noted that in this section, we illustrate, by an example, the decision logic behind each examined π.

Figure 5.10 shows contour maps of scoring pattern of candidate points along bounded signal values of static hazard state signal (sh) and goal strength state signal (gs) whilst power consumption state signal (pw) is made static. Darker coloured area denotes points with higher rank, whilst lighter coloured area denotes lower ranked points. In this case, the highest ranked points are in the bottom right area of each contour map.

By observing the contour pattern, we can see that, overall, all scoring functions of π give higher rank to points with higher gs and lower sh. This pattern is significant because it drives the UGV towards goal locations whilst keeping the UGV at a safe distance from obstacles. However, each scoring function differs in the level of trade-off between gs and sh. While GE-SF and CMAES-SF are more sensitive to the gs value, HW-SF pays more attention to the value of sh. This behavior leads the UGV with GE-SF and CMAES-SF to more aggressively pursue goals than HW-SF. As can be observed from the contours, gs-axis of GE-SF and CMAES-SF have smoother vertical segments than the gs-axis of HW-SF. These segments clearly develop different gs rankings. From the safety side, GE-SF and CMAES-SF maintain collision-free movement by putting a lower threshold on sh. The horizontal lines at 0.8 and 0.7 are the

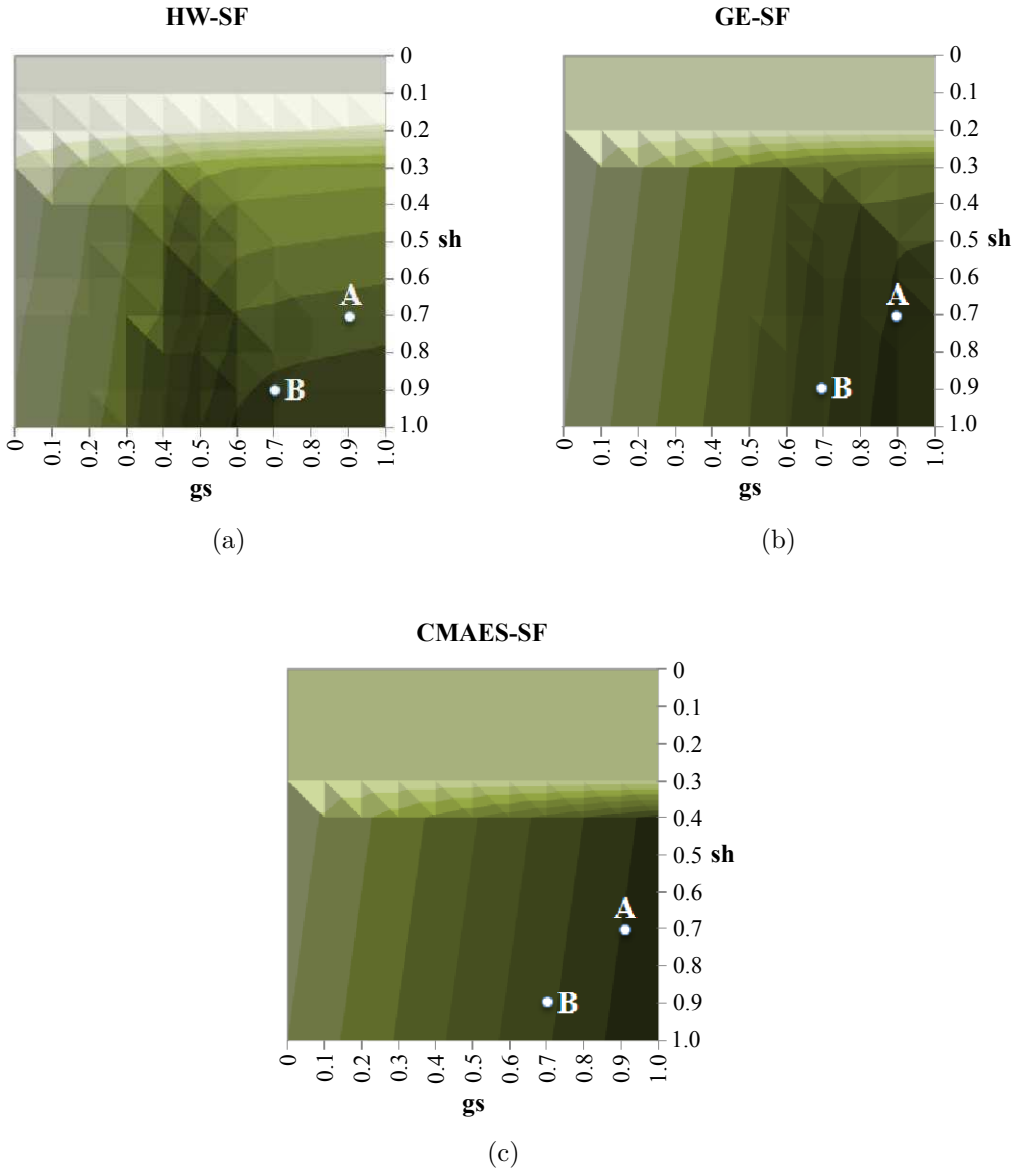


Figure 5.10: Contour maps of two-dimensional state signals: static hazard (sh) vs. goal strength (gs) for (a) HW-SF (b) GE-SF, and (c) CMAES-SF. Note that the contour maps are drawn assuming a power consumption signal (pw) is static at one value (in this case pw = 0.1).

threshold for sh for GE-SF and CMAES-SF, respectively, while HW-SF draws the threshold of sh at 0.9 (note that a higher the value of sh, the nearer the point to an obstacle). These thresholds indicate that any point that has an sh value beyond the threshold value has the lowest rank. In this case, HW-SF weights its ranking excessively in favour of static hazard to the detriment of

exploration [60].

To envisage the effect of scoring function configuration, consider two candidate points, A and B in figure 5.10. Point A is considered nearer to goal location than point B where the gs value for point A and B is 0.9 and 0.7, respectively. In contrast, point A has less distance to the nearest obstacle than point B with the sh value for point A and B is 0.3 and 0.1, respectively. Both points are at safe location in respect to the UGV size. By applying the assigned π, we can see that from the contours, GE-SF and CMAES-SF decide to choose point A rather than point B. This makes the UGV nearer to the goal location if it performs the action of moving to point A. In contrast, HW-SF ranks point B higher than point A. As such, the UGV moves to a safer location but is less attracted to the goal location. Such differences in π configuration will cause the UGV to make different decisions at critical stages of exploration.

5.6 Conclusion

In this chapter, we have analysed several policies used to perform single-robot multi-objective exploration tasks using EC techniques and a conventional hand-designed approach. GE and CMA-ES have been shown to be able to improve the tuning of scoring functions, making the UGV explore faster with consistently low power consumption. Empirically, the experimental results show that the EC-based scoring functions improve exploration coverage about 14% to 15%. The ANOVA test on exploration coverage performance also reported that there are statistically significant differences between all control designs. These results confirm that evolved policies indicate superior performance in comparison to a carefully designed handwritten π. Although CMA-ES exhibits slightly superior performance than GE in this experiment, it is limited to numerical parameters evolution only. In the following chapter, we discuss how GE can be used to evolve not only numerical parameters, but the whole structure of a controller's π. Such evolution can search for a better solution structure to substantially improve the performance of an exploration task.

6 Evolving the Structure of a Policy

Optimising the π function of a motion controller can be tedious work if prior knowledge of the best structure is minimal. In some cases, deriving the structure of π based on first principles is excessively difficult due to the complex nature of a robot system. Hence, a lack of a-priori knowledge to fully understand the robot system usually impedes the process of finding good motion control. In the previous experiments, we have used EC techniques to solve the problem of searching for numerical parameters of a π given a pre-existing form for π. In this chapter, we deal with cases where a pre-defined π structure is not available. We demonstrate the flexibility of GE in catering for this problem by developing a π structure through measurement of exploration performance. A number of GE's grammars are proposed to run the evolution process to contend with various knowledge restrictions. The results suggest that the exploration performance of a robot can be increased by enabling more parts of a π to be evolved.

6.1 Objectives

The aim of this chapter is two-fold:

1. to present the state-of-the-art design for a set of GE grammars for evolving π taking advantage of a-priori knowledge.

2. to compare the exploration performance of π's evolved with the above-mentioned GE's grammars.

6.2 Formulating a Policy under Different Levels of Knowledge Availability

As the complexity of an exploration mission grows and the number of robot's internal states increases, a-priori knowledge of how to explore in this arbitrary environment becomes harder to obtain. In our case, the effectiveness of searching for only one part of a π such as numerical constants alone may become inadequate to sustain good exploration performance. With this in mind, an interesting question is the degree to which the search process for π's structure can be automated and what benefit can be derived from such automation? [61]. Ultimately, it might be desirable to allow the entire structure of a π to be generated automatically with minimum human intervention and thus becoming less dependent on a-priori knowledge. By loosening our dependency on a-priori knowledge we can eliminate bias arising from a pre-determined structure, thus giving π more freedom to adapt to its context.

On the other hand, the evolution of the entire structure of π presents two major technical challenges. First, the evolutionary search space can be extremely large and complex with many local minima. Fortunately, many EC search techniques are well suited to such challenging search spaces. Second, diversity in the structure of π can lead, in conventional GP, to a need to carefully design a grammar that balances searchability and the needs of the problem at hand. Fortunately GE avoids this issue through an indirect mapping from a variable length binary string representation to produce a syntactically correct π structure by using a user-defined BNF grammar. The *separation* of a problem space and a search space allows complex problems to be formulated comfortably in a grammar, without the necessity to invoke the representation in the search space.

In the following section, we present three alternative generative grammars that are designed purposely to evolve a scoring function for π of a DTC at three different levels. The first grammar is used to evolve numerical constants in the scoring function given a known structure. The second grammar is implemented to evolve numerical constants and sub-functions of each state signal in the scoring function. Finally, the third grammar is employed to evolve numerical constants, sub-functions and arithmetic operations between state signals in the scoring function. This third grammar can be considered as the evolution of entire scoring function structure. The availability of a-priori knowledge is assumed in a descending order from the first grammar to the third grammar.

For this experiment, we maintain the application domain of chapter 5. This means the state signals under study are similar i.e. power consumption, static hazard and goal strength. We examine, through experiments, whether the exploration performance can be increased by finding a new structure to correlate between these state signals in the scoring function.

6.3 Setting Up the Grammars

One way to deal with cases where knowledge is limited is to develop a π by using black-box approaches. In black-box approaches, a π is represented as a model in which its internal structure (usually opaque and complex) can be empirically determined by applying input-output data pairs. Such model examples include Fuzzy Logic based models or Neural Network based models [56, 62, 69, 8, 127]. While applying such models looks likely, a-priori, to be productive, they do not solve the problem of finding an explicit model for a DTC. Thus, we remain conservative by using an ordinary mathematical expression as the basis of a π. We choose this white-box model because the internal structure of the model is crystal-clear, thus it can be easily interpreted and further analysed.

Figure 6.1 shows a generative model of a π to be evolved. This model is a subset of the motion control structure depicted in figure 5.1. For each f_C, a set of candidate solution $\boldsymbol{p_j}$ is determined (refer to section 5.2.1 for the

detailed process of establishing candidate target locations). The π then receives three sets of sampled state signals: static hazard ($\boldsymbol{sh_j}$), power consumption ($\boldsymbol{pw_j}$) and goal strength ($\boldsymbol{gs_j}$) (refer to section 5.2.1 for the detailed sampling process). The output from the policy is a short-distance target location (p_d). A scoring function, $f(p_j|sh_j, pw_j, gs_j)$ is the subject under study where its structure is evolved using GE to find the best input-output relationship. The π implements a simple maximum function (MAX) to choose the best output. Note that, $f(p_j|sh_j, pw_j, gs_j)$ returns a real value with a greater score indicating a more desirable output. Score below or equal to 0.0 indicates that it is better for the robot to stay at the current position rather than moving towards the proposed target location.

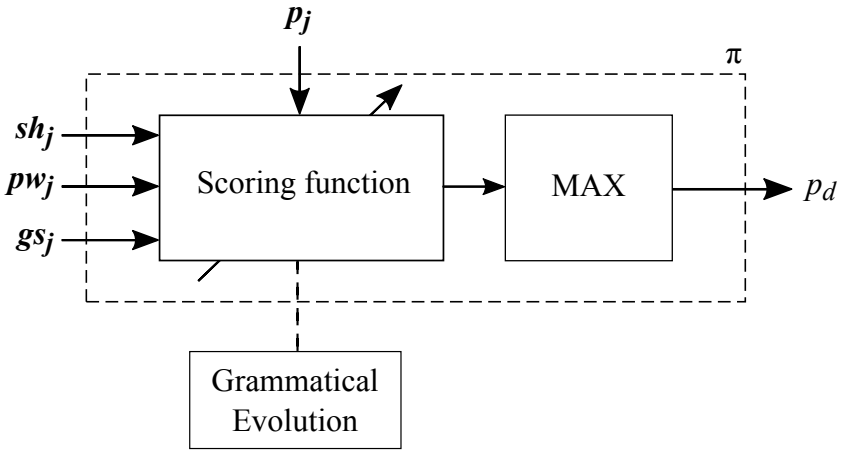


Figure 6.1: A generative π model for the DTC under study in this chapter. The π receives a set of candidate solutions $\boldsymbol{p_j}$ and three sets of sampled state signals: static hazard ($\boldsymbol{sh_j}$), power consumption ($\boldsymbol{pw_j}$) and goal strength ($\boldsymbol{gs_j}$). A scoring function is used to evaluate $\boldsymbol{p_j}$ according to their corresponding state signals sampling. MAX operation is applied to get the best solution (p_d). The scoring function is evolved using GE to get the best input-output relationship.

The general model of our scoring function for each candidate target location p_j, where $j = 1, 2..., N$ and N is the number of candidates, is as follows:

$$f(p_j|sh_j, pw_j, gs_j) = c(g(sh_j), h(pw_j), m(gs_j)) \tag{6.1}$$

Equation 6.1 indicates that $f(p_j|sh_j, pw_j, gs_j)$ is built from the aggregation of a set of unary function of each state signal, $g(sh_j)$, $h(pw_j)$ and $m(gs_j)$. A unary function of each sampled signal can be in form of a mathematical function such as a linear function or a polynomial function.

Conceptually, there are two stages of $f(p_j|sh_j, pw_j, gs_j)$ generation. First, the influence of each sampled state signal to $f(p_j|sh_j, pw_j, gs_j)$ can be manipulated independently by applying arbitrary unary functions. Second, the aggregation of all sampled state signals can be determined after each signal proposes its own function. By conceptualising $f(p_j|sh_j, pw_j, gs_j)$ as above, the level of autonomy allowed in the generation of the function can be set depending on the availability of prior knowledge. For example, in the previous experiments of chapter 4 and 5, we assume available a-priori knowledge is adequate to determine i) the unary function of each sampled state signal (where a linear, polynomial or logistic function is manually chosen for each signal), and ii) the aggregation method of all sampled state signals (weighted sum or multiplicative aggregation). Thus, only numerical parameters of every unary function is automatically tuned by our evolutionary mechanism. In the next section, we describe our evolutionary mechanism that can be extended to automatically find some parts of $f(p_j|sh_j, pw_j, gs_j)$ that were set manually before.

While other search methods might find this task complicated, GE is easily adapted to such problems. The primary advantage of GE to allow users to freely define grammars suited to their application, without the risk of generating large numbers of syntactically incorrect candidate solutions. This eliminates part of the complexity of setting up the evolutionary process when new condition is encountered. For the above problem, we define three novel grammars to construct $f(p_j|sh_j, pw_j, gs_j)$. We use the three levels of grammars as defined in the following three sub-sections to evolve the π function.

6.3.1 Grammar 1: Evolving Constants

For the purpose of comparing approaches, we revisit a grammar of the type that was previously used to evolve numerical constants of $f(p_j|sh_j, pw_j, gs_j)$. We generate the grammar for the evolution of numerical constants based on the digit concatenation technique [25]. The fundamental building block of the digit concatenation technique is the creation of production rules in a grammar that specifies a set of single digit as terminals. For example, one can define a production rule of a grammar that contains digit zero through digit nine as in figure 6.2. The non-terminal <digit10> can be used several times in other production rules to produce a constant using digits concatenation. For example, if the constant to be produced is in range from 0.00 to 9.99, then a production rule shown in figure 6.3 can be added to the grammar.

```
<digit10> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Figure 6.2: A production rule presents digit zero through digit nine as terminals.

```
<constant> ::= <digit10>.<digit10><digit10>
```

Figure 6.3: A production rule to produce a constant ranging between 0.00 to 9.99.

For the process of evolution of constants in π, two elements in $f(p_j|sh_j, pw_j, gs_j)$ are set manually to fixed values. First, we set the unary function of each sampled state signal to a linear function as shown in equation 6.2, where x is a state signal term, and a_1 and b_1 are constants.

$$a_1 * x + b_1 \in [0, 1] \tag{6.2}$$

Second, the outer structure of $f(p_j|sh_j, pw_j, gs_j)$ is fixed with multiplication operators such that:

$$f(p_j|sh_j, pw_j, gs_j) = g(sh_j) * h(pw_j) * m(gs_j) \tag{6.3}$$

Figure 6.4 shows a grammar that was developed to evolve the numerical constants of unary function of three abovementioned sampled state signals.

```
1  <expr> ::= (1 - Linear(sh,<var_lin>)) *
2             (1 - Linear(pw,<var_lin>)) *
3             (Linear(gs,<var_lin>))
4
5  <var_lin> ::= <num_A>,<num_B>
6
7  <num_A> ::= <digit10>.<digit10><digit10>
8  <num_B> ::= <sign><digit2>.<digit10><digit10>
9
10 <digit2> ::= 0 | 1
11 <digit10> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
12
13 <sign> ::= | -
```

Figure 6.4: Grammar 1: This grammar is used to evolve mumerical constants in $f(p_j|sh_j, pw_j, gs_j)$.

In the above grammar, the non-terminal <expr> is the starting production of $f(p_j|sh_j, pw_j, gs_j)$. <expr> contains the terms for $g(sh_j)$ (line 1), $h(pw_j)$ (line 2) and $m(gs_j)$ (line 3). Note that, $g(sh_j)$ and $h(pw_j)$ are set as inverse linear functions due to the fact that both state signals are repulsive forces. The non-terminal <var_lin> (line 5) provides two sets of numerical constants where <num_A> and <num_B> represent the constant a_1 and b_1 of a linear function as in equation 6.2, respectively. The non-terminal <num_A> (line 7) produces a floating point number by concatenating digits using non-terminal <digit10>. The produced number has range between 0.00 to 9.99. Meanwhile, non-terminal <num_B> (line 8) concatenates output from non-terminal <digit10>, <digit2> and <sign>. Note that, <sign> (line 13) is responsible for determining if <num_B> is a positive or negative term. The number produced by non-terminal <num_B> has a value between -1.99 to 1.99.

6.3.2 Grammar 2: Evolving Functions

In certain problems, tuning the constants alone does not produce a good solution because the model structure is sub-optimal. For example, a non-linear problem represented by a linear solution will often create a biased estimation model. Therefore, if there is not enough a-priori knowledge available to determine a model structure, then such structure becomes an appropriate target for search. In this case, GE is capable to determine such characteristic automatically by evolving functions used to develop a model. In this second grammar, we propose an evolution of the state signals unary functions as well as their corresponding constants simultaneously. We extend the grammar in section 6.3.1 so that each sampled state signal can be embedded in a linear or non-linear function to form the three terms of the model. For simplicity, we provide three function types to be selected in the grammar: linear function, polynomial function or logistic function. The linear function is set as in equation 6.2. Equation 6.4 and 6.5 describe the polynomial and logistic functions, respectively. a_2, b_2, a_3 and b_3 are the functions' corresponding constants, while x is a state signal corresponding to one of sh_j, pw_j or gs_j.

$$a_2 * (x^{b_2}) \in [0, 1] \tag{6.4}$$

$$\frac{1.0}{1.0 + \exp(-a_3 * (x - b_3))} \in [0, 1] \tag{6.5}$$

Figure 6.5 depicts a grammar used to evolve unary function of each sampled state signal and its numerical constants.

The start symbol <expr> of the grammar is set with each state signal's function chooser: <pw_func>, <sh_func> and <gs_func>. Each function chooser is given three options to be selected as the unary function of its corresponding state signal: Linear(...) for linear function, Polynomial(...) for polynomial function, and Logistic(...) for logistic function (line 4 to line 14). After selecting the appropriate unary functions, numerical constants of the selected functions are determined (line 16 to line 18). The process of generating

```
1  <expr> ::= (1 - <pw_func>) * (1 - <sh_func>)
2             * <gs_func>;
3
4  <pw_func> ::= Linear(pw,<var_lin>) |
5                Polynomial(pw,<var_pol>) |
6                Logistic(pw,<var_log>)
7
8  <sh_func> ::= Linear(sh,<var_lin>) |
9                Polynomial(sh,<var_pol>) |
10               Logistic(sh,<var_log>)
11
12 <gs_func> ::= Linear(gs,<var_lin>) |
13               Polynomial(gs,<var_pol>) |
14               Logistic(gs,<var_log>)
15
16 <var_lin> ::= <num_A>,<num_B>
17 <var_pol> ::= <num_A>,<num_A>
18 <var_log> ::= <num_C>,<num_D>
19
20 <num_A> ::= <digit10>.<digit10><digit10>
21 <num_B> ::= <sign><digit2>.<digit10><digit10>
22 <num_C> ::= <digit3><digit10>.<digit0><digit10>
23 <num_D> ::= 0.<digit10><digit10>
24
25 <digit2> ::= 0 | 1
26 <digit3> ::= 0 | 1 | 2
27 <digit10> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
28
29 <sign> ::= |
```

Figure 6.5: Grammar 2: This grammar is used to evolve unary functions and numerical constants in $f(p_j|sh_j, pw_j, gs_j)$.

those constants follows procedures explained in section 6.3.1. An example of output that may be produced from this grammar given an individual genotype is shown in figure 6.6.

6.3.3 Grammar 3: Evolving Structure

The third grammar in this section is the most comprehensive grammar among three presented grammars. As mentioned earlier in this chapter,

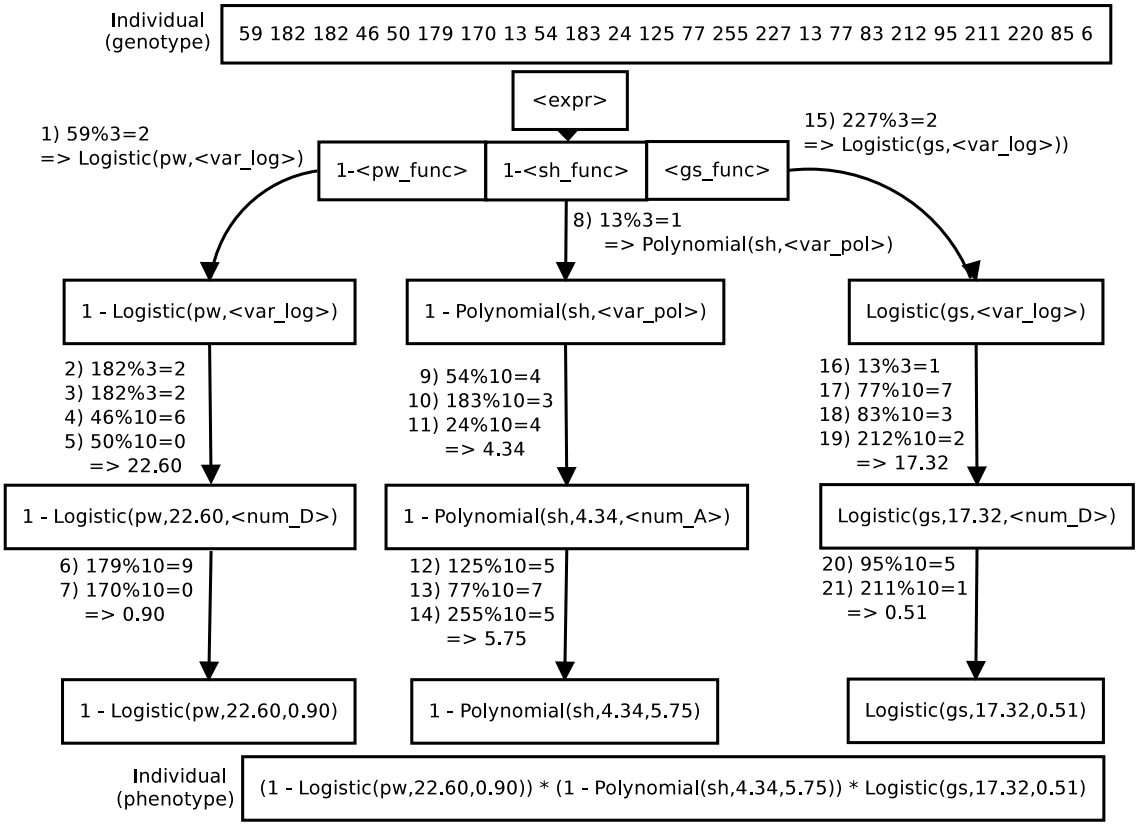


Figure 6.6: Example of output produced using grammar in figure 6.5.

the ultimate objective to allow the entire structure of π to be generated automatically. The third grammar is designed to achieve this objective, or at most, uses very minimal a-priori knowledge. While the two grammars above are restricted to change the structure of $f(p_j|sh_j, pw_j, gs_j)$, we design the third grammar to lift the restriction by allowing arithmetic operators between state signals to be defined automatically, as well as, the order of those signals in $f(p_j|sh_j, pw_j, gs_j)$.

Evolution of multiple types of terms requires the grammar to be designed carefully to constrain search to the space of reasonable individuals. Although there is no general rule to apply, we propose the following steps as in figure 6.7 as a good design heuristic for our grammar.

To better understand the steps, we present the third grammar as shown in figure 6.8. First, the start symbol of the grammar, <expr> is formulated

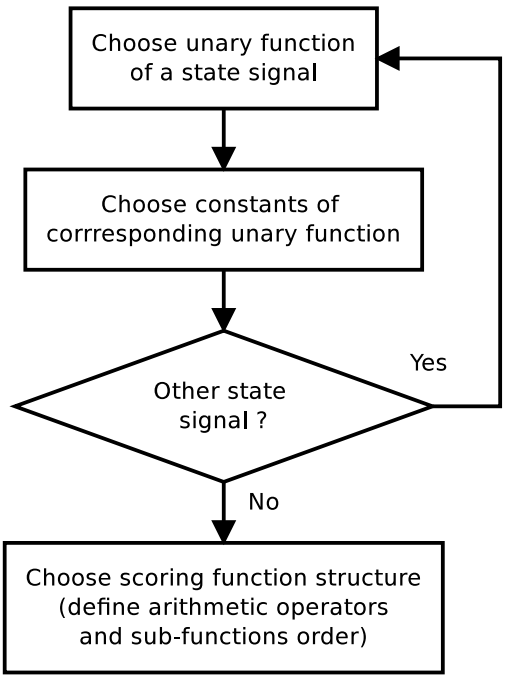


Figure 6.7: Grammar design heuristic for evolution of the $f(p_j|sh_j, pw_j, gs_j)$ structure.

to produce four C++ typed statements. Each statement represents a mathematical function. The first three statements (line 1) describe the unary functions: $g(sh_j)$, $h(pw_j)$ and $m(gs_j)$, respectively. Meanwhile, statement in line 2 builds the outer part of $f(p_j|sh_j, pw_j, gs_j)$ aggregating all the unary functions. By formulating our start symbol in this way, we have explicitly driven the grammar to define a unary function and its corresponding constants for each sampled state signal in turn. In this grammar, we set each state signal to opt for either a normal unary function or its negation. The choice of unary function is defined by selecting linear, polynomial or logistic functions. The rest of the calculation follows the same process as in section 6.3.1 and 6.3.2. After all unary functions are defined, the grammar then determines the outer structure of $f(p_j|sh_j, pw_j, gs_j)$ by selection of:

1. order of unary functions in $f(p_j|sh_j, pw_j, gs_j)$ (line 34 to 40).

2. arithmetic operators for aggregation between unary functions $(+, -, *, /)$ via the non-terminal <op> (line 42).

```
1  <expr> ::= <sh_struc>; <pw_struc>; <gs_struc>;
2             <score_struc>;
3
4  <sh_struc> ::= shf = <sh_func> | shf = 1 - <sh_func>
5  <pw_struc> ::= pwf = <pw_func> | pwf = 1 - <pw_func>
6  <gs_struc> ::= gsf = <gs_func> | gsf = 1 - <gs_func>
7
8  <sh_func> ::= Linear(sh,<var_lin>) |
9                Polynomial(sh,<var_pol>) |
10               Logistic(sh,<var_log>)
11
12 <pw_func> ::= Linear(pw,<var_lin>) |
13               Polynomial(pw,<var_pol>) |
14               Logistic(pw,<var_log>)
15
16 <gs_func> ::= Linear(gs,<var_lin>) |
17               Polynomial(gs,<var_pol>) |
18               Logistic(gs,<var_log>)
19
20 <var_lin> ::= <num_A>,<num_B>
21 <var_pol> ::= <num_A>,<num_A>
22 <var_log> ::= <num_C>,<num_D>
23
24 <num_A> ::= <digit10>.<digit10><digit10>
25 <num_B> ::= <sign><digit2>.<digit10><digit10>
26 <num_C> ::= <digit3><digit10>.<digit0><digit10>
27 <num_D> ::= 0.<digit10><digit10>
28
29 <digit2> ::= 0 | 1
30 <digit3> ::= 0 | 1 | 2
31 <digit10> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
32 <sign> ::= | -
33
34 <score_struc> ::= score = pwf <op> shf <op> gsf |
35                   score = (gsf <op> shf) <op> pwf |
36                   score = (gsf <op> pwf) <op> shf |
37                   score = (shf <op> gsf) <op> pwf |
38                   score = (shf <op> pwf) <op> gsf |
39                   score = (pwf <op> gsf) <op> shf |
40                   score = (pwf <op> shf) <op> gsf
41
42 <op>::= + | - | * | /
```

Figure 6.8: Grammar 3: This grammar is used to evolve the structure of $f(p_j|sh_j, pw_j, gs_j)$.

Note that we include explicit options for different orderings and associativity of terms. These ensure that each state signal unary function is included once in any solution but gives maximum flexibility in how it contributes. With this general method, GE has a very much larger search space than the solution in sections 6.3.1 and 6.3.2. As a consequence, a candidate $f(p_j|sh_j, pw_j, gs_j)$ has large structure diversity. Figure 6.9 shows an example of genotype to phenotype mapping using this grammar.

6.4 Experiments

In this section, we discuss experiments conducted to evolve π using the proposed grammars. This section is divided into two sub-sections as follows:

1. Experimental set up.

2. Experimental results.

6.4.1 Experimental Set Up

We ran three groups of experiments to evaluate the performance of π evolution by using three different BNF grammars. We named the groups: CONST_EVO, FUNC_EVO and STRUC_EVO. The first experiment group (CONST_EVO) applies the BNF grammar of figure 6.4 to evolve numerical constants of unary functions in the $f(p_j|sh_j, pw_j, gs_j)$ of a π. The second experiment group (FUNC_EVO) uses the BNF grammar shown in figure 6.5 to evolve both the unary function of each sampled state signal and its corresponding numerical constants. Finally, the third experiment group (STRUC_EVO) is designed to evolve the entire structure of $f(p_j|sh_j, pw_j, gs_j)$ including arithmetic operators, unary functions and numerical constants by using the grammar depicted in figure 6.8. Figure 6.10 summarises the experimental set up.

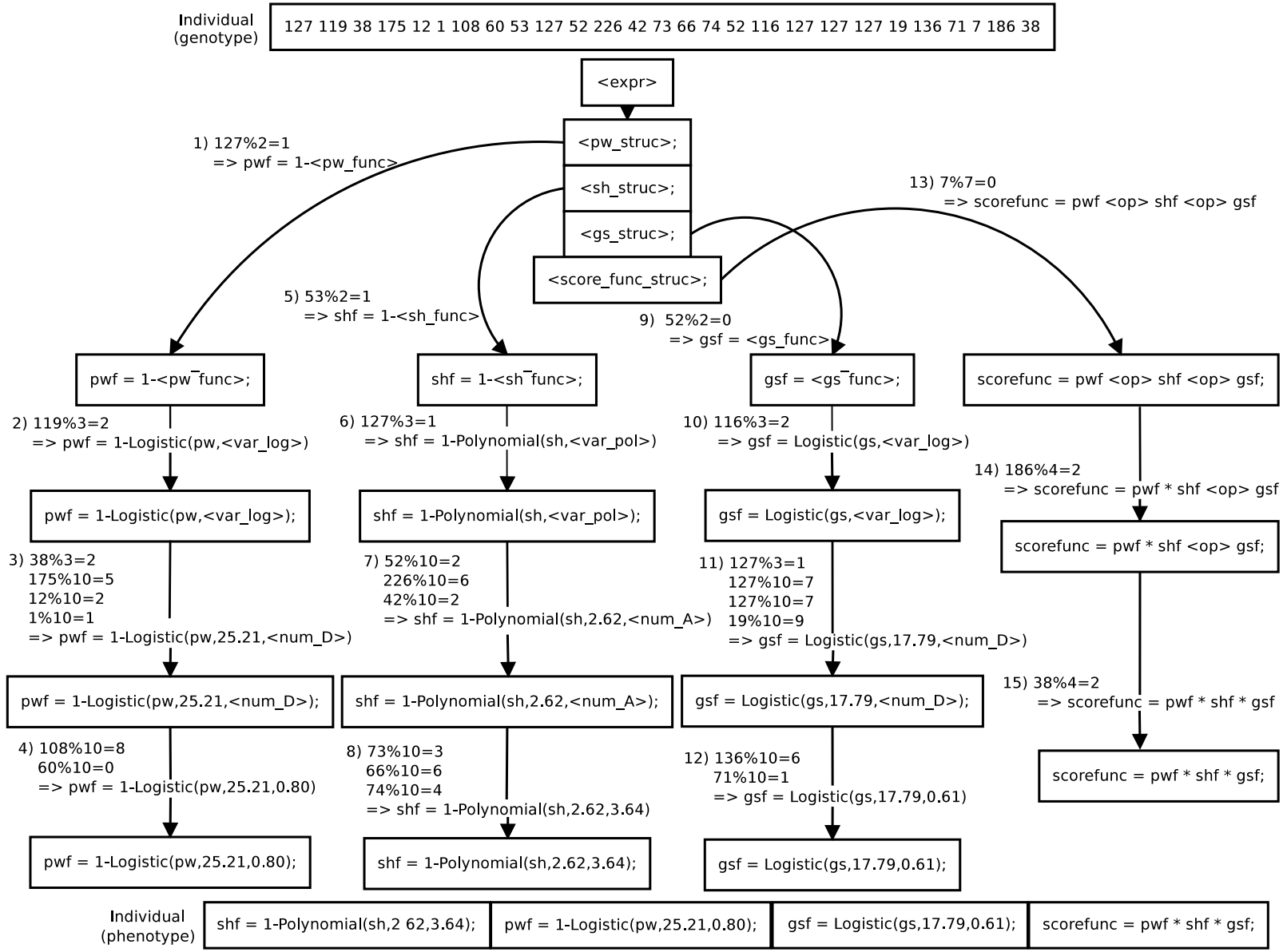


Figure 6.9: Example of output produced using grammar in figure 6.8.

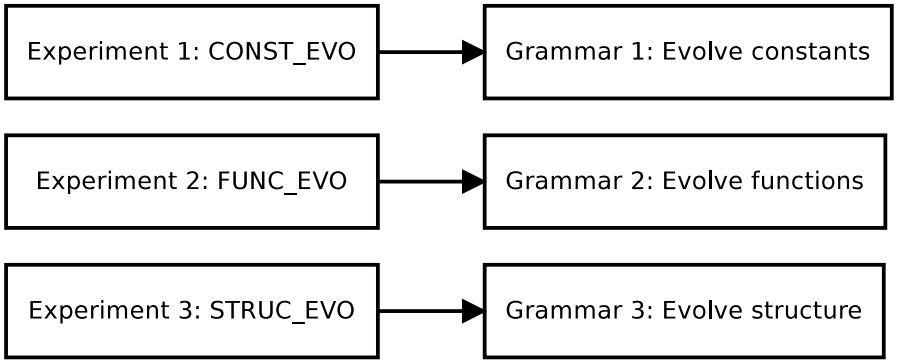


Figure 6.10: Experimental set up. CONST_EVO experiments are conducted using the first grammar to evolve numerical constants in $f(p_j|sh_j, pw_j, gs_j)$. FUNC_EVO experiments use the second grammar to evolve both unary functions and numerical constants. STRUC_EVO experiments apply the third grammar to evolve the whole structure of $f(p_j|sh_j, pw_j, gs_j)$.

Evolution - For a fair comparison, all experiment groups are configured with the same GE settings except for the number of generations. With the same configuration, any individual produced by the search engine in GE can be evaluated with the same evaluative function. Meanwhile, a different number of generations is used because each grammar is divergent in terms of search space size. As such, we want to give ample time for the evolutionary process to converge according to the size of each search space. In this case, grammar 1 in figure 6.4 has the smallest search space, while grammar 3 in figure 6.8 has the largest search space. Table 6.1 summarises the configuration of GE for all experiments. Note that the selection of these parameters follows the same rule used in previous chapters where we aim for a good balance between total run duration and evolution performance.

Every individual produced by GE is scored by embedding the evolved scoring function in a high-fidelity simulation platform. The evaluative process for each individual follows *embedding-simulation-collation* cycle as depicted in figure 5.3 previously. In *embedding*, each particular individual is concatenated into a C++ source code as a scoring function, which then is re-compiled. In *simulation*, a simulated exploration task is performed on a maze map emulates an office-like environment shown in figure 6.11 within the maximum time-frame of 180 seconds. Simulations were run on the *Stage* platform [125]. A simulation

Table 6.1: GE configurations for three evolution groups: CONST_EVO, FUNC_EVO and STRUC_EVO.

Item	CONST_EVO	FUNC_EVO	STRUC_EVO
Number of generations	100	200	300
Population size	100		
Search engine	Genetic algorithm		
Selection type	Tournament		
Crossover type	Effective crossover		
Mutation type	Point mutation		
Probability of crossover	0.95		
Probability of mutation	0.05		
Elitism	Yes		
Simulation time per run	180 secs real-time (4-10 secs after speeding up)		

run is terminated upon the first occurrence of the following conditions:

1. run-time exceeds the maximum time-frame.

2. collision occurs.

3. the robot reaches a stall position.

4. robot power consumption exceeds the power usage threshold.

5. the robot explores the map completely.

In *collation*, an individual is scored based on exploration performance gathered from *simulation*, taking into account the total area being explored (*area*), total power consumption (*power*) and collision status (*coll*). An aggregated performance index concatenating *area*, *power* and *coll* is formed as a fitness function (evaluative function) as shown in equation 6.6.

$$fitness = \left(area + \frac{1}{1 + power} \right) * \left(\frac{1}{1 + coll} \right) \tag{6.6}$$

Hardware - Experiments were conducted on an IBM HS22 server with an

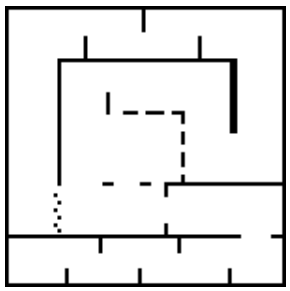


Figure 6.11: A maze map emulates an office-like environment used by GE to evaluate exploration performance of a candidate solution.

3.47GHz eight core Intel Xeon X5677 CPU and 48GB of memory. Each evolutionary experiment was repeated 15 times. The server allows multiple experiments to be run in parallel. The longest evolutionary runs would consume 2 days of CPU time.

6.4.2 Experimental Results

We report the results of the conducted experiments in two phases:

1. Learning phase - the output produced in each evolution group and statistical analysis of the performance of each group are discussed.

2. Application phase - comparison of exploration performance between each evolution group tested on other environments is presented.

Learning Phase

Experimental results for the above evolution runs are considered in this section. All 45 experiment runs (15 runs for each group) were completed successfully. Each run produces the best $f(p_j|sh_j, pw_j, gs_j)$ out of C evaluated candidates, where C =*number of generations*∗*population size*. In this case, $C = 10,000$, $20,000$ and $30,000$ respectively for the CONST_EVO, FUNC_EVO and STRUC_EVO experiments. Figures 6.12, 6.13 and 6.14 show the evolution

performance of the best fitness produced at all generations for all three experiment groups.

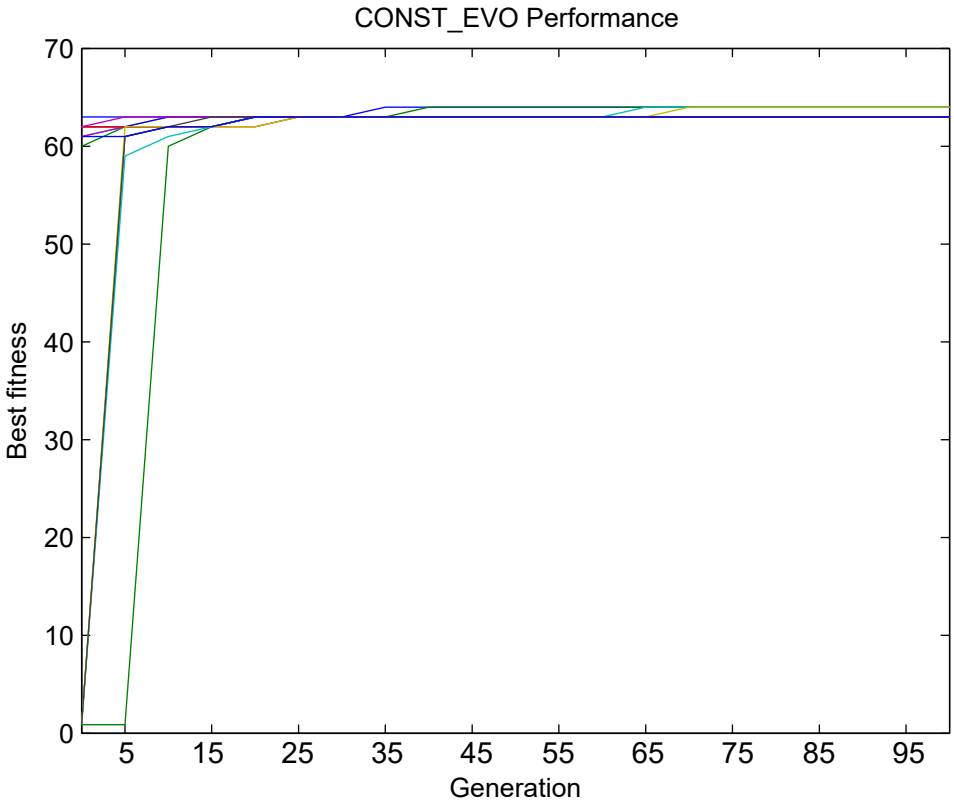


Figure 6.12: CONST_EVO evolution run performance - the best fitness produced at all generations for 15 runs.

Since a GE-based search engine with an elitism function is applied, the best $f(p_j|sh_j, pw_j, gs_j)$ on each run will be the best individual in the last generation. This data on the best individual can be used to compare the evolution performance of CONST_EVO, FUNC_EVO and STRUC_EVO side by side. Figure 6.15 depicts the fitness of the best $f(p_j|sh_j, pw_j, gs_j)$ produced by all experiment groups represented in box plots.

From the boxplots, we observe that CONST_EVO produces the lowest median fitness at 63. In contrast, the performance of FUNC_EVO and STRUC_EVO increases significantly with median fitness at 188 and 186, respectively. The bottleneck in the performance of $f(p_j|sh_j, pw_j, gs_j)$ for CONST_EVO appears to be, as we shall see later, linear unary function used for each state

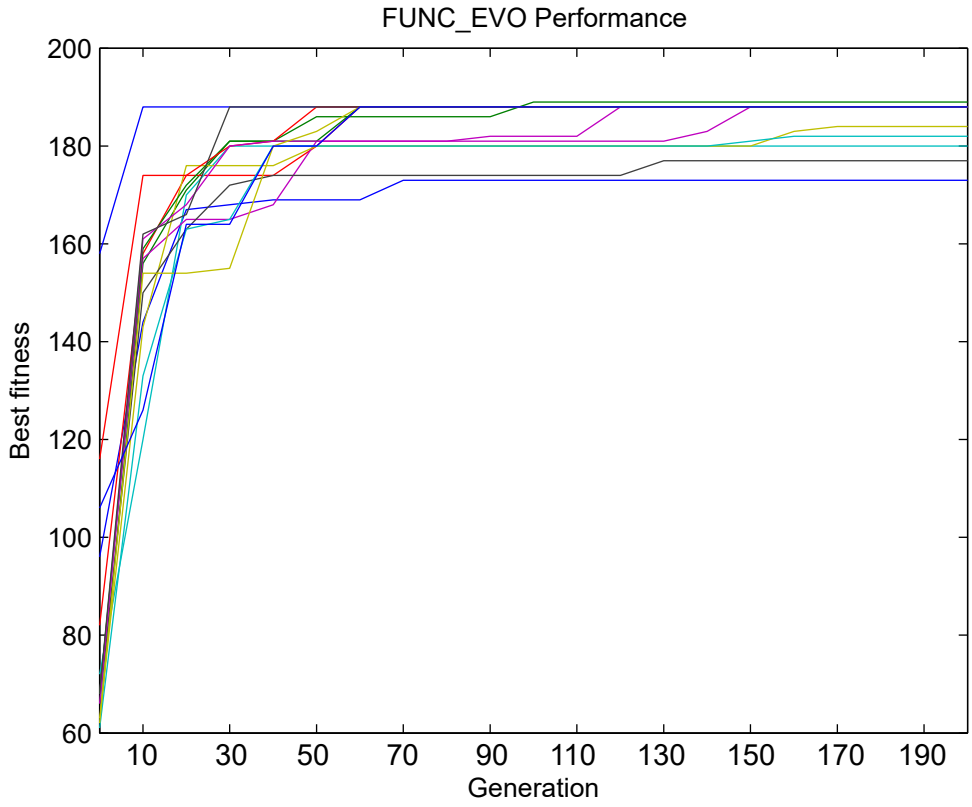


Figure 6.13: FUNC_EVO evolution run performance - the best fitness produced at all generations for 15 runs.

signal. This outcome suggests that the problem at hand should not be handled with a linear scoring function. GE, however, manages to break out of this performance bottleneck by adapting appropriate grammars in FUNC_EVO and STRUC_EVO, respectively. The importance of automating the search for the function's structure appears to be reinforced by these findings. Given an appropriate set of options, GE can best search for the design of $f(p_j|sh_j, pw_j, gs_j)$ by simultaneously adapting numerical constants, unary functions and arithmetic operators of the function.

Back to the boxplots, another observation can be seen from the performance range between the maximum and the minimum fitness of each experiment group. The output from CONST_EVO experiments are consistent with a standard deviation of 0.414 - which is very small compared to the magnitudes of the fitness values. One possible reason of this consistency is that the low fitness

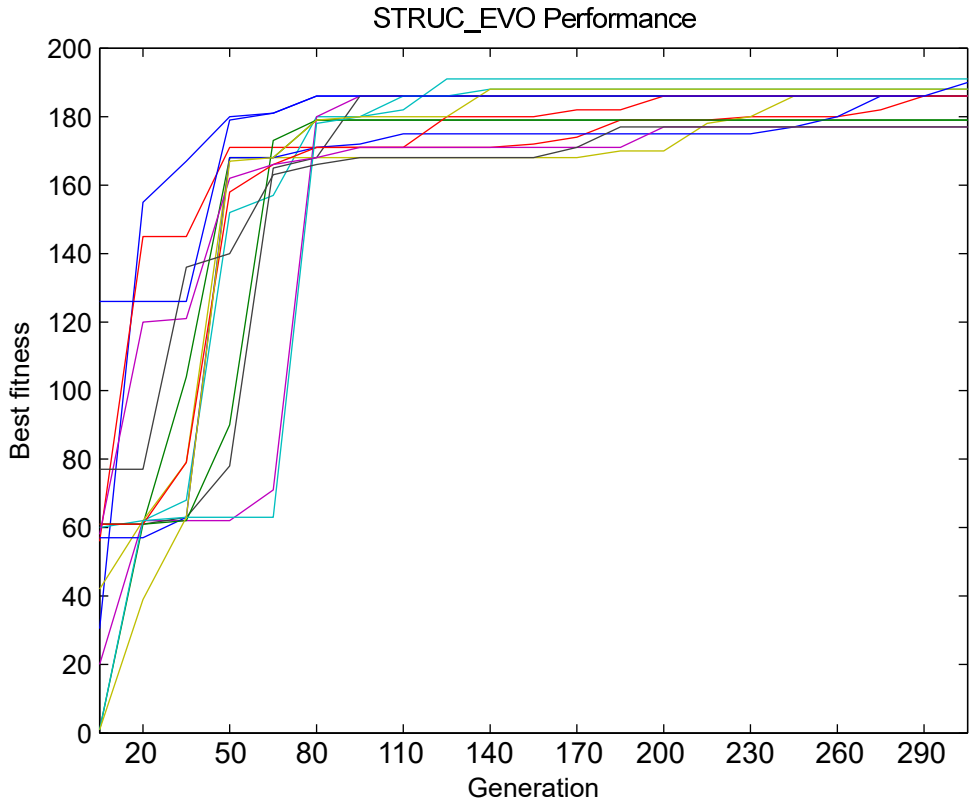


Figure 6.14: STRUC_EVO evolution run performance - the best fitness produced at all generations for 15 runs.

score stems from a relatively low explored area in which most of the linear model scoring function candidates make the robot move through the same area in cyclic pattern. In contrast, FUNC_EVO and STRUC_EVO produce broader range of fitness than CONST_EVO with a standard deviation of 4.926 and 4.580, respectively. The findings indicate that while the robot can explore new area reliably in FUNC_EVO and STRUC_EVO, the accumulated paths the robot takes in each experiment might be different due to noise factors. However, the level of variation is not extreme as, in each case, the motion control manages to drive the robot safely while navigating.

Another observation from the boxplots suggests that the performance of FUNC_EVO and STRUC_EVO is highly comparable. However, STRUC_EVO has advantages over FUNC_ EVO in that the complete structure of $f(p_j|sh_j, pw_j, gs_j)$ is determined automatically. In contrast, FUNC_EVO

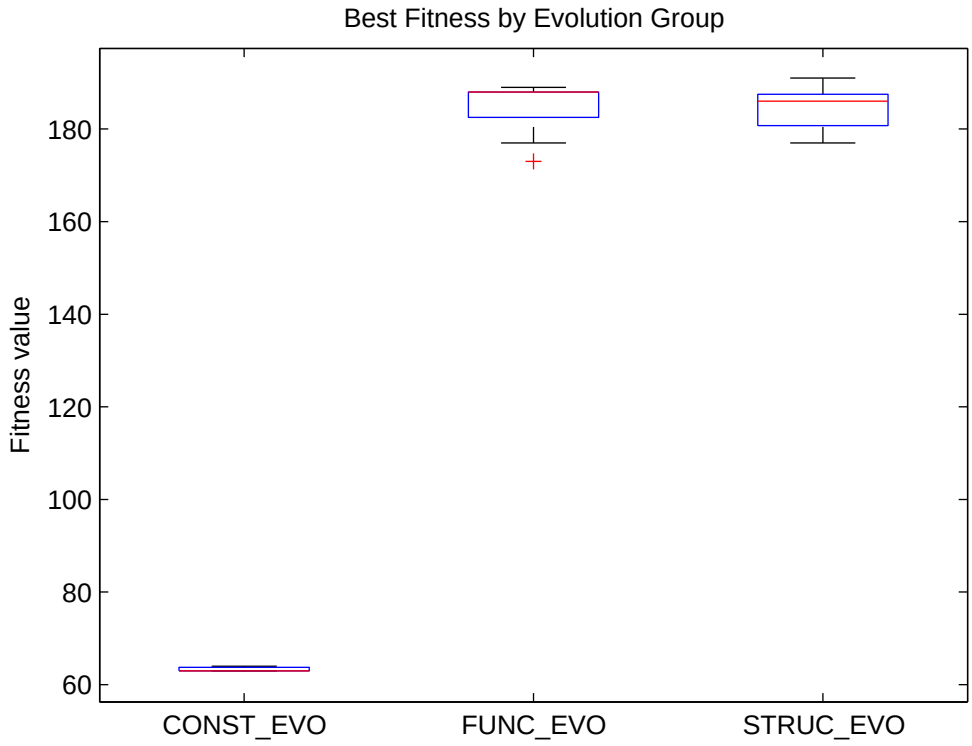


Figure 6.15: Boxplots of the best fitness value produced from every run for (a) CONST_EVO (b) FUNC_EVO, and (c) STRUC_EVO experiments.

requires roboticists to manually define the aggregation method for the state signals of the unary functions. Equations 6.7 to 6.9 present the best found $f(p_j|sh_j, pw_j, gs_j)$ from the three groups of experiments.

CONST_EVO:

$$\begin{aligned} g(sh_j) &= 1 - Linear(sh_j, 4.93, -1.93) \quad \in [0,1] \\ h(pw_j) &= 1 - Linear(pw_j, 0.90, -0.02) \quad \in [0,1] \\ m(gs_j) &= Linear(gs_j, 0.54, 0.11) \quad \in [0,1] \\ f(p_j|sh_j, pw_j, gs_j) &= g(sh_j) * h(pw_j) * m(gs_j) \end{aligned} \tag{6.7}$$

FUNC_EVO:

$$\begin{aligned} h(pw_j)) &= 1 - Polynomial(pw_j, 0.96, 9.62) \quad \in [0,1] \\ g(sh_j) &= 1 - Polynomial(sh_j, 7.6, 5.61) \quad \in [0,1] \\ m(gs_j) &= Logistic(gs_j, 26.9, 0.62) \quad \in [0,1] \\ f(p_j|sh_j, pw_j, gs_j) &= g(sh_j) * h(pw_j) * m(gs_j) \end{aligned} \tag{6.8}$$

STRUC_EVO:

$$\begin{aligned} h(pw_j) &= Logistic(pw_j, 16.27, 0.06) \quad \in [0,1] \\ g(sh_j) &= Polynomial(sh_j, 2.92, 4.67) \quad \in [0,1] \\ m(gs_j) &= Logistic(gs_j, 20.75, 0.55) \quad \in [0,1] \\ f(p_j|sh_j, pw_j, gs_j) &= (1 - g(sh_j))/h(pw_j) * m(gs_j) \end{aligned} \tag{6.9}$$

To find out whether the difference between the $f(p_j|sh_j, pw_j, gs_j)$ are statistically significance, T-tests between every two groups were performed. This is done by comparing the best fitness from the previous experimental runs of each group. Table 6.2 tabulates data used for the statistical calculation, while table 6.3 shows the results of the T-tests. The results show that the p-values from the CONST_EVO/FUNC_EVO test and CONST_EVO/STRUC_EVO test are respectively $9.75E-37$ and $9.38E-38$, below the required 5% confidence level. This means the results from FUNC_EVO and STRUC_EVO are significantly different from CONST_EVO. Meanwhile, FUNC_EVO/STRUC_EVO test shows the p-value of 0.82, suggesting that FUNC_EVO and STRUC_EVO are comparable, conforming the result from the box plot.

Application Phase

For validation, the results found in the learning phase were further analysed on other environments that have never been explored by the robot. Another two

Table 6.2: Data of area explored (m^2) in 15 runs and its mean ($\bar{\mathbf{X}}$) for each tested π of the robot performing exploration task.

Run	CONST_EVO	FUNC_EVO	STRUC_EVO
1	64.0004	188	186
2	64.0005	180	179
3	63.0004	188	186
4	63.0005	189	191
5	63.0005	173	188
6	63.0006	184	186
7	63.0003	177	186
8	63.0003	188	186
9	63.0006	188	179
10	63.0006	188	186
11	64.0003	182	188
12	63.0006	188	177
13	63.0004	188	188
14	63.0006	188	177
15	63.0006	188	190
X	63.267	185.133	184.733

Table 6.3: T-test results to analyse the variation arises among three evolution groups: CONST_EVO, FUNC_EVO and STRUC_EVO using samples from Table 6.2.

Group	p-value
CONST_EVO/FUNC_EVO	9.75E-37
CONST_EVO/STRUC_EVO	9.38E-38
FUNC_EVO/STRUC_EVO	0.82

simulated office-like environments are created to validate the evolved π. Both environments are shown in figure 6.16. In this application phase, we allow the robot to explore the environments with a particular π with extremely large allowance for running time. Scoring functions of equation 6.7, 6.8 and 6.9 represent the π produced by CONST_EVO, FUNC_EVO and STRUC_EVO, respectively.

Each policy are given 30 trial runs. The average performance of the robot on both environments in terms of time taken to complete exploration and its corresponding power consumption is summarised in Table 6.4. The table

shows that the robot with FUNC_EVO and STRUC_EVO policies manages to explore safely and completely on both environments. In contrast, the robot with CONST_EVO policy only manages to wander around area near its start pose. The behaviour shown by these three policies is consistent with the behaviour shown in the learning phase. The second comparison is the accumulated power consumption. From the table, FUNC_ EVO uses less power than CONST_ EVO and STRUC_ EVO as this is proportionate to the total path length taken by the robot. Overall, we can conclude that the scoring function of FUNC_EVO and STRUC_EVO outperforms CONST_EVO demonstrating that the performance of the π is significantly consistent across learning phase and application phase.

Figure 6.16: Application maps showing footprints of the robot using three different scoring function: red line for STRUC_EVO, green line for FUNC_EVO, and blue line for CONST_EVO (a) Robot pose on environment 1 after 240 seconds run-time (b) Robot pose on environment 2 after 234 seconds run-time.

Table 6.4: Average performance. Performance parameters: A-explored area, T-exploration time, P-consumed power and TP-total path length [map 1/map 2].

f	A(m^2)	T(sec)	P(kJ)	TP(m)
CONST_EVO	51/145	∞/∞	4.02/5.47	64.02/94.52
FUNC_EVO	225/225	217/236	3.52/3.77	58.72/62.38
STRUC_EVO	225/225	240/234	3.83/3.97	62.74/65.55

6.5 Discussion

In this section, we examine the results produced by the evolutionary runs of CONST_EVO, FUNC_EVO and STRUC_EVO to understand the effect of π structure to exploration performance. We divide this discussion into two parts. The first part is dedicated to describing the response curves of of each term (defined by a unary function) in the scoring function of π suggested by the evolutionary process in all 45 experimental runs of the learning phase. The latter part is allocated to apprehend the scoring quality of the whole π graphically by constructing ranking surface contours. We illustrate the effect of ranking surface on target location selection.

Patterns of Unary Functions

We studied the relationship between exploration performance and the pattern of state signals' unary functions. In the learning phase, each experimental run is designed to find the best $f(p_j|sh_j, pw_j, gs_j)$ by using GE to fine-tune some parts of the function. $f(p_j|sh_j, pw_j, gs_j)$ is composed of a set of either pre-defined or evolved state signals' unary functions. Thus, the unary function has significant impact on exploration performance, yet the answer to the question: "Which patterns of unary functions constitutes a good or bad scoring function?" is currently unclear. By compiling all state signals' unary functions gathered from a set of the best $f(p_j|sh_j, pw_j, gs_j)$ produced by all 45 evolutionary runs (15 runs each for the CONST_EVO, FUNC_EVO and STRUC_EVO experiments), we are able to differentiate some good and poorly performing patterns.

We briefly reintroduce the descriptions of each state signal to allow for an easier visualisation of the impact of unary functions. Please note that all state signals are normalised to a value between 0.0 and 1.0. For pw_j, the higher the value, the higher the power needed to move robot from its current position to a target position and vice-versa. For sh_j, a higher value means that a target location is nearer to a static obstacle and vice-versa. Finally, gs_j with a value equal to 0.5

or higher indicates that the robot makes a positive progress towards a target position. However, if a gs_j value is less than 0.5, it shows that the robot will move further away from a target location. Figure 6.17 summarises the state signals description graphically.

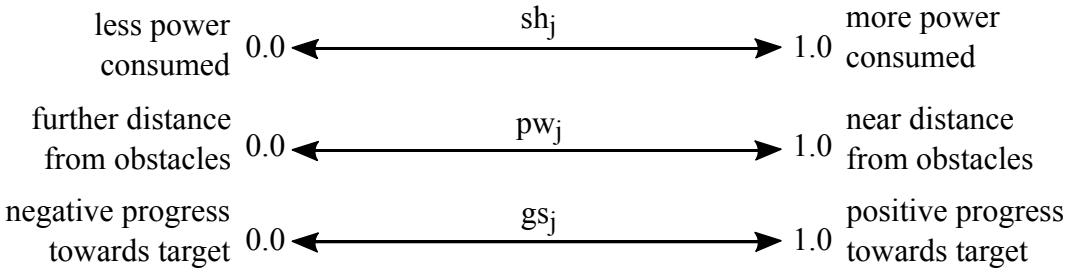


Figure 6.17: A description of state signals. A sampled state signal has a value between 0.0 to 1.0.

CONST_EVO - Firstly, we observe the pattern of unary functions produced by CONST_ EVO experiment runs. Figure 6.18 shows three graphs representing unary functions for each state signal pw_j, sh_j and gs_j, respectively. The top graph plots the pattern of pw unary functions, $h(pw_j)$ extracted from the best scoring functions from 15 runs. The middle graph shows the pattern of corresponding sh unary functions, $g(sh_j)$. Meanwhile, the bottom graph displays the pattern of corresponding gs unary functions, $m(gs_j)$. Note that, $h(pw_j)$ and $g(sh_j)$ are in inverse linear function form, while $m(gs_j)$ is in linear function form. Red lines in all graphs indicate that the corresponding scoring functions have performance below the median. In contrast, the blue line scoring functions have performance above median. This color code also holds for the following three figures 6.18, 6.19 and 6.20.

The overall pattern of unary function graphs tells that a better score is given to a candidate target position that has a higher gs value and lower pw and sh values. We can say that the solutions from CONST_EVO only partially meet the requirements of exploration missions. This can be explained by looking at each state signal's unary function graph. The requirement to explore safely is achieved throughout all runs. This is supported by the $g(sh_j)$ unary functions pattern showing that any sh_j value bigger than 0.8 makes a contribution of zero to the scoring function. The rule can be translated into a distance measurement by saying that if the path towards a target has an obstacles less

than 0.5 meter of clearance (where half of the robot width is 0.35 meter), then the target's score will be zero. However, the exploration capability for CONST_EVO controllers is low due to the unreliable influence of the $m(gs_j)$ pattern in giving a good score to a higher gs_j value. From the 15 best $m(gs_j)$ functions, only one solution gives positive linear score proportional to the gs_j value. In contrast, other solutions trigger a plateau score of 1.0 to almost all gs_j with value bigger than 0.5. This means the CONST_EVO scoring function is unlikely to differentiate well between candidate targets based on gs_j. For power consumption requirement, it is observed that there is no single pattern that can be observed to separate the good from the bad.

FUNC_EVO - The unary functions from FUNC_EVO experiments are not constrained to linear form and show more adaptive patterns than the CONST_EVO experiments. Figure 6.19 shows the pattern of unary functions from FUNC_EVO experiments. All solutions that have exploration performance above median value exhibit a consistent pattern distribution across all state signals (observe blue lines in the graphs). FUNC_EVO maintains safe exploration by setting up 0.8 as the minimum threshold for sh_j to have zero score. The safety measurement is smoothed by polynomial functions that give a higher score when the sh_j value is even marginally 0.8. In terms of exploration capability, FUNC_EVO improves progress towards a target position by giving a high positive score to higher gs_j value and zero score to gs_j with negative progress towards the target. All 15 solutions suggest a logistic function as the best unary function to represent $m(gs_j)$. For the power consumption pattern, the graph shows that solutions with performance above median (in blue lines) do not consider any pw_j value below a threshold around 0.65 as degrading the score of a candidate target location. This group of unary functions suggests that pw_j will only have effect on the overall score when the pw_j value is really high. This pattern makes the robot less likely to choose a movement with large turning angle and longer target distance.

STRUC_EVO - Finally, the unary function graphs of STRUC_EVO are presented in figure 6.20. Note that, the grammar of STRUC_EVO allows the unary function of a state signal to evolve with either a positive or negative sign and be combined with evolved operators. The generic pattern induced

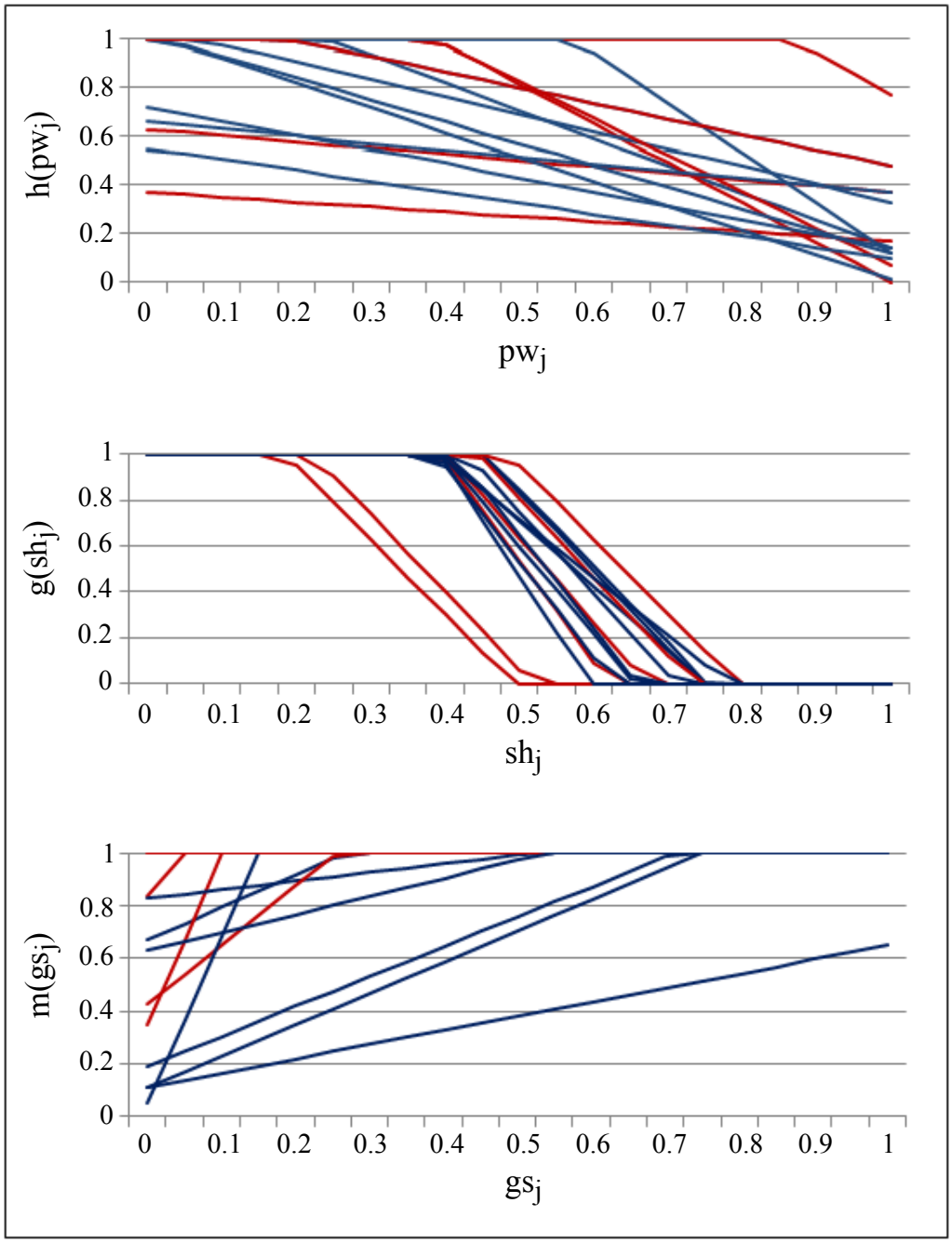


Figure 6.18: CONST_EVO: Unary functions of state signals of the best scoring function from 15 evolutionary runs.

from STRUC_EVO solutions is similar to the pattern of FUNC_EVO solutions especially for solutions with performance above median. This pattern can be used as a general guide to confirm that a good scoring function can be created if unary function of each state signal is developed along the pattern shown in

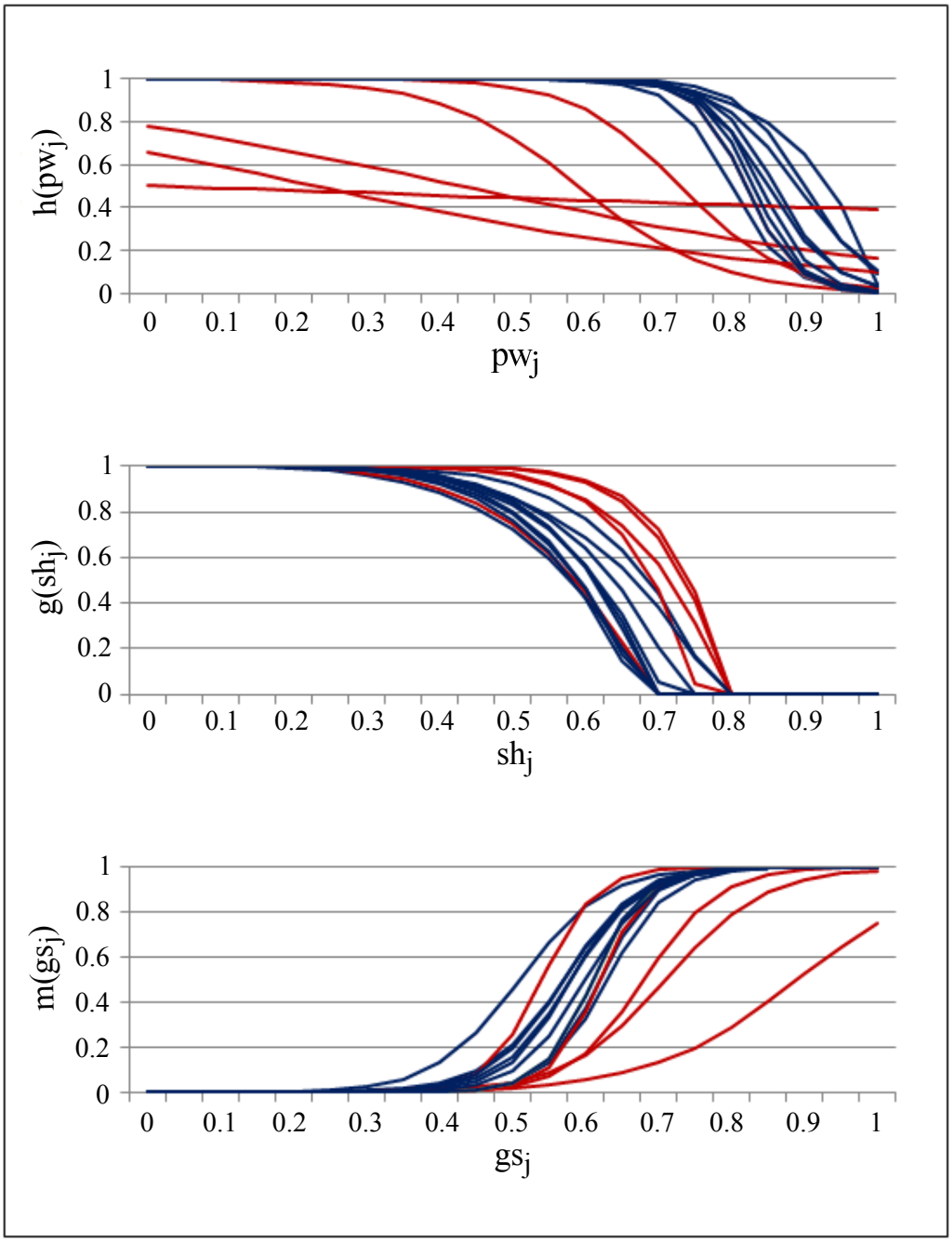


Figure 6.19: FUNC_EVO: Unary functions of state signals of the best scoring function from 15 evolutionary runs.

the graph.

Given that the structure of the scoring function can be evolved in this experiment group, several types of unary functions have been selected by

the best scoring functions in our 15 runs. For $h(pw_j)$, 4 solutions choose a linear function, 3 solutions use polynomial functions and 8 solutions implement logistic functions. For $g(sh_j)$, all solutions choose polynomial functions, while for $m(gs_j)$, all solutions apply logistic functions. In summary, STRUC_EVO suggests that a good scoring function can be generated by applying logistic function to $m(gs_j)$ and a polynomial functions to $g(sh_j)$. Meanwhile, $h(pw_j)$ can implement any of the three functions since none of them is dominant. It should be noted that the exact form of these functions is going to be dependent on the evaluative function we use during evolution. For example, if we were to put a higher premium on lower power consumption we would expect to see a higher penalty applied to individuals with higher power consumption and, perhaps, a smaller variety of functions for pw_j.

Ranking Surfaces of Scoring Functions

To further investigate state signals relationship in a π, we sample the best scoring function of CONST_EVO, FUNC_EVO and STRUC_EVO as ranking surface contours. The dimension of the scoring functions is reduced to a 2-dimensional surface by keeping pw_j constant. In this case, we can analyse the relationship between gs_j and sh_j and their influence to the navigational choice. In general, signal-to-signal relationship analysis can be performed by assigning other state signals with fixed values. Figure 6.21 shows the contours of the best scoring function of CONST_EVO, FUNC_EVO and STRUC_EVO at $pw_j = 0.1$. From the figure, the highest score is at a *peak* indicating a point with the highest gs_j and the lowest sh_j as the best point to navigate. Meanwhile, the lowest score with zero value is in *valley* (region in black) where those points are unlikely to be chosen as the navigational point. In comparison, we can see that STRUC_EVO and FUNC_EVO are able to be more aggressive in exploration than CONST_EVO. To illustrate, target location choice **A** shown in each contour figures has a high-pay-off($gs_j = 0.9$) and high hazard ($sh_j = 0.6$) point, but still a safe navigational choice. In CONST_EVO, choice **A** is poorly ranked - it will almost never be chosen. Meanwhile, in STRUC_EVO and FUNC_EVO, choice **A** is highly ranked.

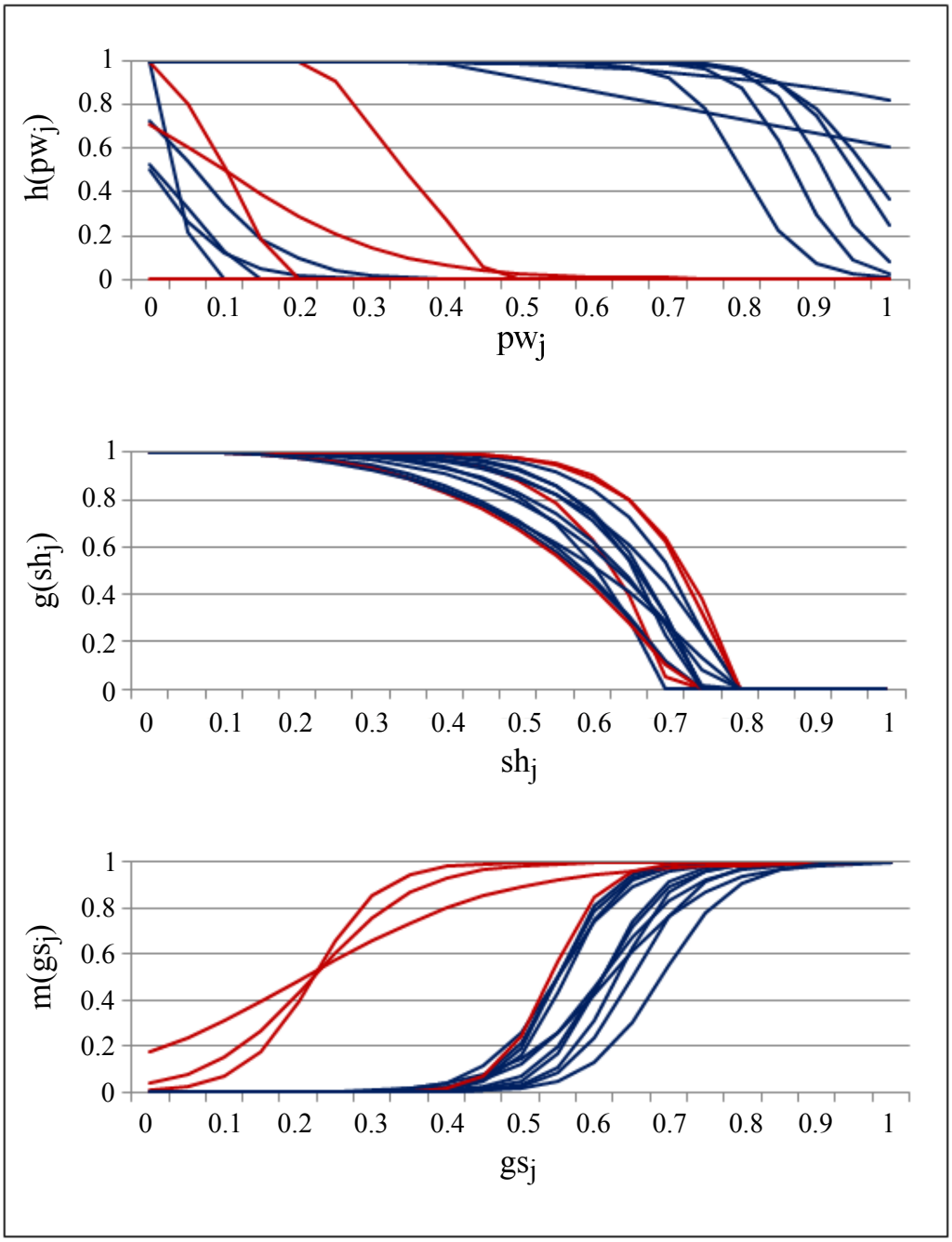


Figure 6.20: STRUC_EVO: Unary functions of state signals of the best scoring function from 15 evolutionary runs.

Conversely, choice **B** which is a low-pay-off($gs_j = 0.5$) and low-hazard($sh_j = 0.4$) point is relatively attractive in CONST_EVO but ranked very poorly in STRUCT_EVO and FUNC_EVO. Therefore, we can infer that the ranking surfaces of STRUC_EVO and FUNC_EVO yield better navigational decision

making compared to CONST_EVO. The most persistent features of the ranking surfaces of STRUC_EVO and FUNC_EVO are the sharply defined low plain for sh_j above 0.8, the low plateau for gs_j below 0.5 and the gentle rise for gs_j values above 0.5. This echelon structure, and its boundaries, seem to characterise what is required of a good scoring function in our particular set up and tallies with the foregoing analysis done on state signal unary functions.

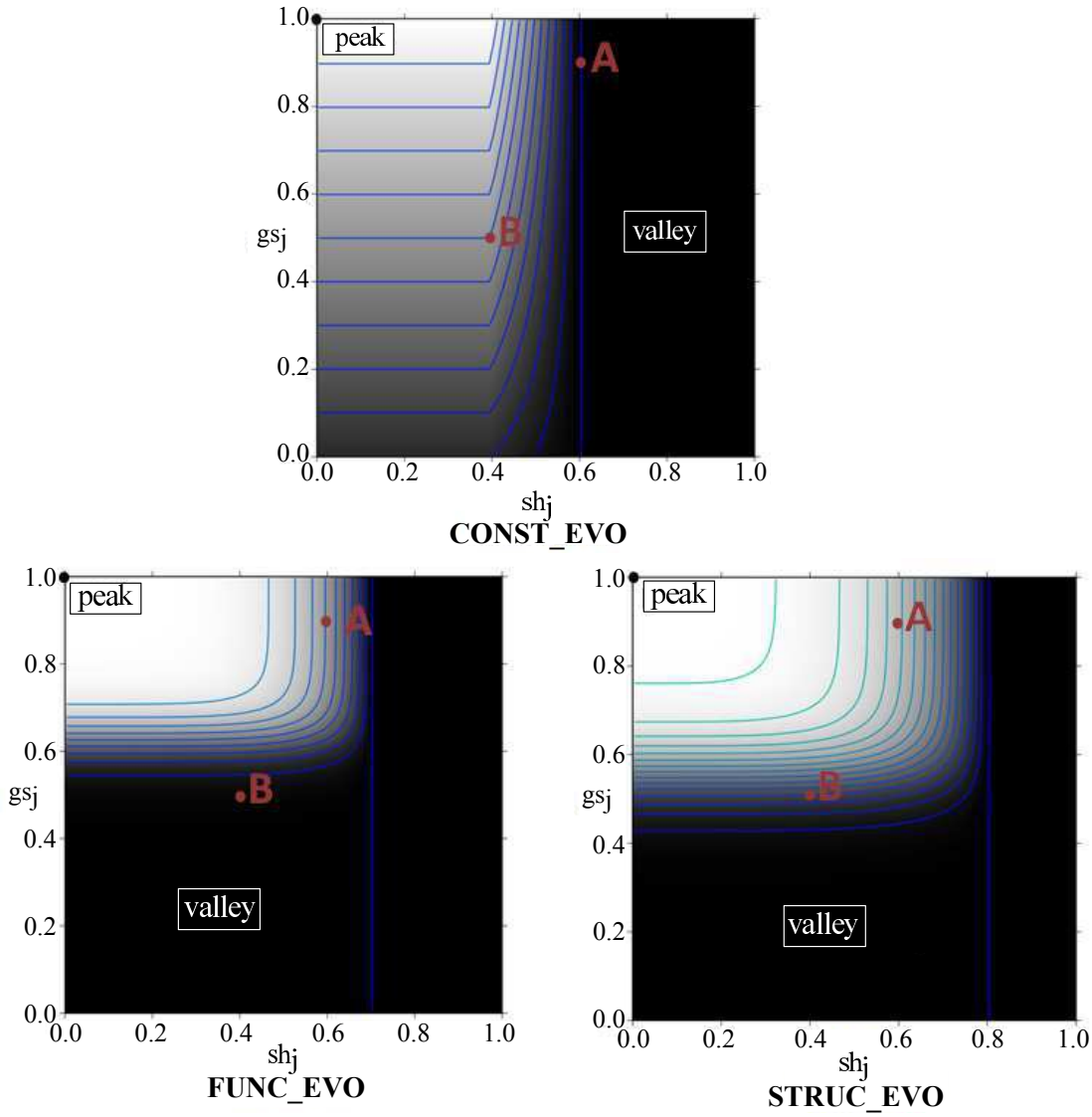


Figure 6.21: Scoring function ranking surfaces. The surfaces show the contours of gs_j vs. sh_j for the best scoring function of CONST_EVO, FUNC_EVO and STRUC_EVO at $pw_j = 0.1$. Point **A** and **B** are sampled points to compare decision made by each scoring function.

6.6 Conclusion

To achieve optimal robotic exploration performance, a mechanism to design the structure of a controller's policy automatically is highly beneficial. Our approach of using GE to evolve the scoring function of a π at various levels shows that the exploration performance can be improved significantly by searching a space of scoring functions that is hard to explore thoroughly with conventional manual design. The flexibility of GE's grammars to accommodate different levels of prior knowledge to support π design is also useful especially if little knowledge about an exploration system is available beforehand. Our experimental results have shown that GE is able to search for good function structures in a very large search space. Observing the results of multiple evolutionary runs also yields some patterns that can guide designers in understanding the intrinsic properties behind a good scoring function given the robot's set up and environment.

7 Evolving Input Factor Choice of a Policy

In this chapter we investigate the design of a well-known navigation scheme, the Dynamic Window Approach (DWA) [42], as motion control for an indoor robot known as Turtlebot. This control mechanism differs from the previous chapters in that it incorporates the kinematic model of the robot to decide the steering command required. The DWA takes into account the limitations of maximum velocity and acceleration of the robot. Thus, its π requires different implementation in terms of its decision-making model. We integrate the DWA as motion control for our existing Frontier-based exploration system. There are a few DWA variations [42, 10, 112, 40] where each DWA uses a different set of input factors for its scoring function of π. Here, we use the term: input factors to refer to the state signals used in previous chapters. However, there is no general rule that specifies the number or selection of factors needed for any DWA. Depending on the DWA implementation, these factors can differ from one application to another. The objective of this chapter is to evolve π for a DWA controller such that appropriate input factors can be selected automatically from a set of candidate factors. We present a new BNF grammar for GE to perform automatic selection of factors for π of a DWA controller, in addition to the task of structure evolution from the earlier chapters. The robot control with a number of evolved π is extensively tested under several variations of conditions: i) perfect localisation ii) use of SLAM, and iii) a real-robot platform. Experimental results from both simulations and real robot platforms are presented.

7.1 Objectives

The main objectives of this experiment are:

1. to present a method for the automatic selection of input factors for π of DWA-based motion control using GE.

2. to compare exploration performance of evolved DWA controllers and a handwritten DWA controller on simulated and real hardware-based autonomous exploration tasks.

7.2 Related Work on DWA Policies

An early work on the use of a DWA algorithm as a local planner suggests the following scoring function in its π [42]:

$$\sigma[\alpha \cdot heading + \beta \cdot obs\_dist + \gamma \cdot vel] \tag{7.1}$$

There are three factors in this weighted-sum based π function. *heading* measures the alignment of the robot with the target direction. It is calculated as $180 - \theta$, where θ is the angle of a target point relative to the robot's heading direction. *obs_dist* measures the distance to the closest obstacle that intersects with the curvature. *vel* measures the progress of the robot on the corresponding trajectory. It is simply a projection on the translational velocity v_t. Meanwhile, σ, α, β and γ are tunable constants.

Another scoring function for a local DWA algorithm is proposed by [10] as follows:

$$\alpha \cdot heading2 + \beta \cdot vel2 + \gamma \cdot goal\_reg \tag{7.2}$$

*heading*2 measures the alignment of the robot with a target direction, calculated as $1 - |\theta|/\pi$, where θ is the angle of the target point relative to the robot's heading direction. *vel*2 measures the progress of the robot on the

corresponding trajectory calculated as:-

$$vel2 = \begin{cases} \frac{\|v_t\|}{v_{max}} & \text{if robot is far from goal} \\ 1 - \frac{\|v_t\|}{v_{max}} & \text{if robot is close to goal} \end{cases}$$

goal_reg is a binary function that gives the value of 1 if robot passes through the goal region in forward simulation, 0 otherwise. α, β and γ are constants.

Furthermore, [10] extends the scoring function to suit a global DWA algorithm as shown in equation 7.3:

$$\alpha \cdot nf1 + \beta \cdot vel2 + \gamma \cdot goal\_reg + \sigma \cdot \Delta nf1 \tag{7.3}$$

where $nf1$ measures the distance to a global goal. Its potential field is computed using a wave-propagation technique. It labels cells in the occupancy grid with the L^1 distance to the goal, taking into account obstructions by obstacles. The result is a local-minima-free potential function with a unique minimum at the goal. $\Delta nf1$ estimates the rate of progress made to reduce the value of $nf1$. α, β, γ and σ are constants.

The DWA algorithm was further improved by [112] proposing a crashing probability factor to create robot's awareness to obstacles near trajectory. The scoring function is designed as follows:

$$\alpha \cdot goal\_dist + \beta \cdot heading + \gamma \cdot obs\_dist + \sigma \cdot vel + \eta \cdot safety \tag{7.4}$$

The proximity of the destination point is measured by *goal_dist*. The value of zero is given if the goal point is on the trajectory. *safety* measures the probability that robot does not crash with any obstacle if it applies a velocity command. This factor is calculated as a function of an obstacle probability density function. α, β, γ, σ and η are constants.

The last scoring function discussed in this section refers to the algorithm implemented in the library of ROS Electric [40]. According to the

documentation, the scoring function of the DWA algorithm is computed as:

$$\alpha \cdot path\_dist + \beta \cdot goal\_dist + \gamma \cdot obs\_dist \tag{7.5}$$

where *path_dist* measures the distance to suggested path from the endpoint of a trajectory, while α, β and γ are constants of each factors, respectively.

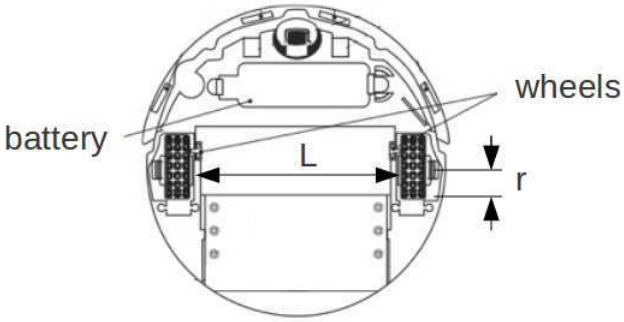
7.3 Turtlebot - A Mobile Indoor Robot

In this experiment we use an open-source mobile robot development kit known as Turtlebot 2 [41]. Figure 7.1 shows the Turtlebot 2 platform. We adapt this new robot platform to demonstrate the capability of GE to be applied across different robot designs. Turtlebot 2 is a non-holonomic differential-drive robot. The motion of the robot is controlled via two separately driven wheels. It has only two controllable degrees of freedom: translational velocity and rotational velocity.

Figure 7.1: Turtlebot 2 platform [41].

Figure 7.2(a) shows the base of Turtlebot 2. Turtlebot 2 has two main wheels placed on both sides of the robot body. A motor, attached to each wheel, can be controlled independently. The direction of movement can be determined by varying the relative rate of rotation of both motors.

(a)



(b)

Figure 7.2: Turtlebot 2 base [109].

We use the kinematic model of the robot's motion as described in [77] to simulate robot movements. Consider two parameters L and r in figure 7.2(b), in which L is the distance between the two robot's wheels and r is the wheel's radius. We assume both wheels have the same diameter. If the action space of the robot is defined as $u = (v_{a_r}, v_{a_l})$, where v_{a_r} and v_{a_l} are angular wheel velocities (e.g. in degrees per second unit) for the right and left wheels, respectively, then the configuration transition equations can be developed as follows:

$$\dot{x} = \frac{r}{2}(v_{a_r} + v_{a_l})cos\theta \qquad \dot{y} = \frac{r}{2}(v_{a_r} + v_{a_l})sin\theta \qquad \dot{\theta} = \frac{r}{L}(v_{a_r} - v_{a_l}) \qquad (7.6)$$

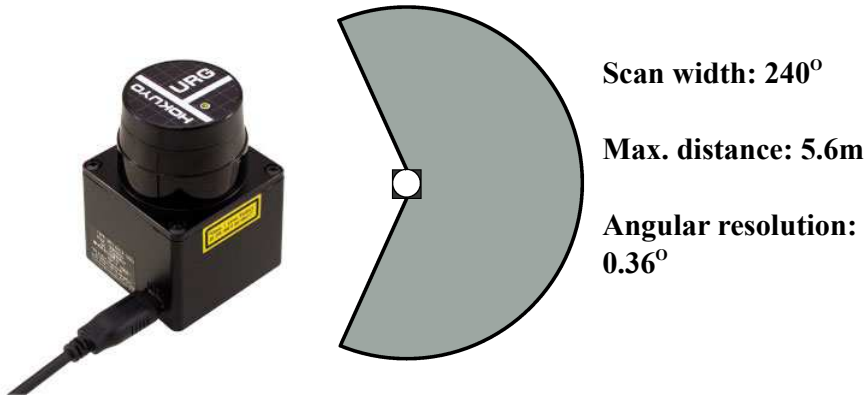
where (x, y, θ) denotes the robot's configuration. However, for software

development purposes, we do not determine angular wheel velocities separately. Instead, we transform the action space as $u = (v_t, v_a)$, where $v_t = (v_{a_r} + v_{a_l})/2$ is known as translational velocity and $v_a = v_{a_r} - v_{a_l}$ represents rotational velocity. Consequently, the transition equations of equation 7.6 is re-defined as:

$$\dot{x} = rv_t cos\theta \qquad \dot{y} = rv_t sin\theta \qquad \dot{\theta} = \frac{r}{L} v_a \tag{7.7}$$

From the sensory side, each wheel is equipped with a wheel encoder to allow the use of dead-reckoning. Dead reckoning calculates raw position estimation by counting wheel revolutions. Even though dead reckoning can approximate of how far a robot has travelled, the reading may yield an inaccurate estimation. This is due to the possibility of wheel slippage that creates accumulated error in the measurement of distance travelled. As such, the estimate, also known as odometry, will be further adjusted by a software-based algorithm to produce a better position estimation.

The standard Turtlebot 2 package comes with Microsoft's Kinect 360 sensor [84] that collects Red-Green-Blue-Depth (RGBD) images and then turns them into imitated laser scan readings. Due to the small angular field-of-view (FOV) of the Kinect (57^o horizontally), we replace the sensor with Hokuyo URG-04LX-UG01 [79], a LIDAR sensor. This Hokuyo sensor provides true laser scan reading with FOV of 240^o and maximum ray distance of 5.6 metres. The measurement accuracy of the scanner is up to 3% error. Figure 7.3 shows an image of the physical sensor and its detectable area. Meanwhile, figure 7.4 shows the modified version of Turtlebot 2 configured with the Hokuyo LIDAR sensor that we use in our experiments.



(a) (b)

Figure 7.3: Hokuyo URG-04LX-UG01 laser scanner (a) Physical image (b) Detectable area.

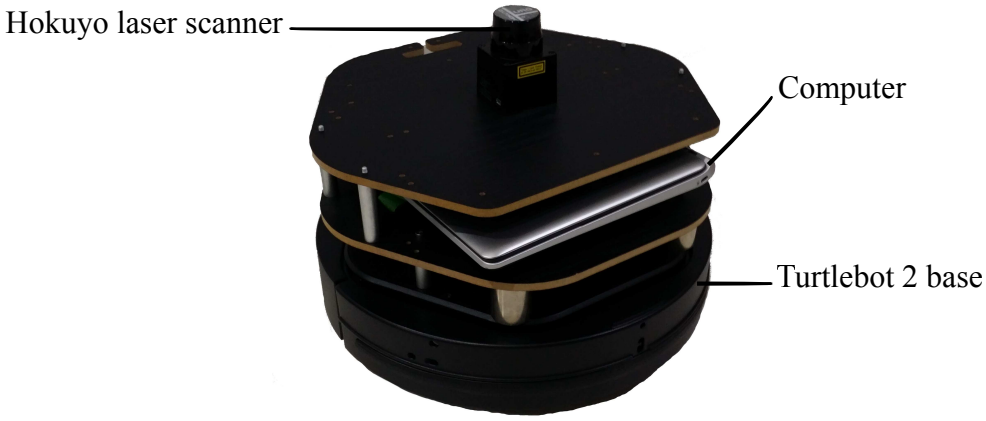


Figure 7.4: Turtlebot 2 equipped with a Hokuyo LIDAR sensor. We use this robot's configuration in all our experiments.

7.4 The Exploration Framework on ROS

From the software perspective, we build our autonomous exploration framework under Robot Operating System (ROS) [40]. Several fundamental libraries have been provided for the Turtlebot software packages such as turtlebot startup, velocity commands and odometry readings. The ROS system also provides a number of algorithms off-the-shelf such as mapping packages, path planner packages and local base controller packages that can

be used by roboticists to perform specific robot tasks. We use such libraries to develop our ROS-based robotic exploration system as indicated in figure 7.5.

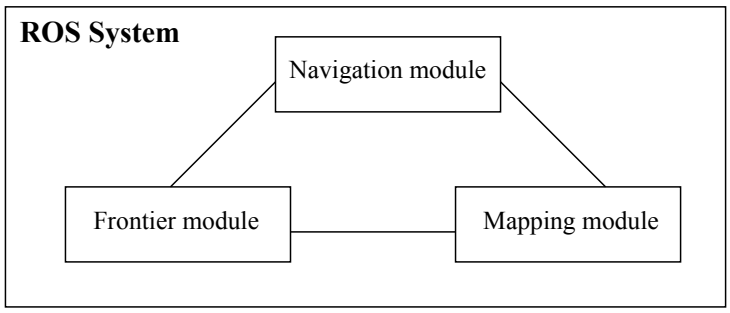


Figure 7.5: ROS-based robotic exploration system consists of three core components: mapping, frontier and navigation modules.

The core components of the exploration system are divided into three major modules: the mapping module, the frontier module and the navigation module. Each of these modules is explained in turn.

7.4.1 Mapping Module

Turtlebot is primarily designed for indoor operation which, in most cases, precludes the use of GPS positioning. Instead, the robot is equipped with a LIDAR as proximity sensor and wheel encoders as motion sensor, to estimate its current position and to, simultaneously, build a map. The algorithmic process that fuses input data in the presence of sensory noise to estimate pose and map information is known as a Simultaneous Localisation and Mapping (SLAM).

In this system, we use the Gmapping SLAM algorithm [47, 48] to provide estimated robot poses and grid maps at the system level. Gmapping is a particle-filter based algorithm that provides accurate localisation by using multiple particles to approximate a probability distribution of poses and maps. Each time Gmapping receives new sensory data, localisation or pose estimation is done by proposing a number of possible poses in terms of particles using the robot's motion model. For each proposed position, Gmapping computes its

corresponding map. Then, it calculates the importance weight of each particle by performing scan matching between the particle's corresponding map and the latest proximity data. A particle with the highest weight is chosen as the current estimated position of a robot, as well as its corresponding map. The process is then iterated whenever new sensory data are received. Two data types are available from Gmapping to be distributed among modules: the robot's current pose and a global grid map. Figure 7.6 shows the block diagram of Gmapping's interface indicating inputs and outputs to the SLAM process algorithm.

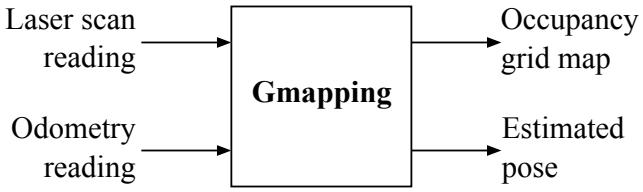


Figure 7.6: Block diagram of the mapping module.

7.4.2 Frontier Module

Once we have an estimation of robot pose and a grid map from Gmapping, the frontier module performs the goal-assignment task. This module provides global-goal locations that drive a robot towards unexplored areas. The algorithm we use in this module is derived from [130]. Refer to section 3.1 of this thesis for details of the algorithm. Figure 7.7 shows the block diagram of the frontier module.

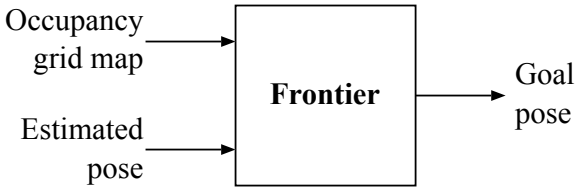


Figure 7.7: Block diagram of the frontier module.

7.4.3 Navigation Module

From the point of view of this research, the central component of this exploration system is the navigation module. We select the DWA to perform the task of navigation module. Unlike in the earlier chapters where the output of the navigation module is either a steering direction or a short-range target location, DWA provides a steering command in terms of velocity vector (v_t, v_a) to the robot's low level controller. This command will be translated into wheel motor rotation by the low level controller to achieve a desired speed. This task is crucial because steering commands have the final say in determining the motion of a robot – bounded with a maximum possible velocity and a maximum possible acceleration – and have a direct influence on exploration performance.

In order for the DWA scheme to work properly, the navigation module contains several sub-modules to provide processed input to our navigation scheme. Figure 7.8 presents the block diagram of the navigation module with its associated sub-modules.

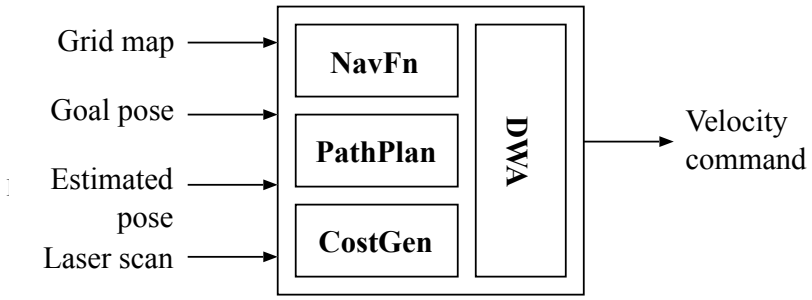


Figure 7.8: Block diagram of the navigation module.

These sub-modules are explained as follows:-

1. Potential-field navigation function (navFn) - This sub-module generates a global potential field of a given map such that the cost of moving to an assigned goal from any point on the map can be estimated. The DWA algorithm uses the generated potential field to estimate the distance to the goal point from the end-point of a trajectory of a candidate steering command.

2. Path planner (pathPlan) - This sub-module takes the current robot pose and the goal pose to calculate the shortest path between those two points. The path planner computes this path using Dijkstra's algorithm [26]. The DWA algorithm uses the generated path as a guide to maintain the robot on the shortest track towards the goal.

3. Cost map generator (costGen) - This sub-module provides information about obstacles of an occupancy grid map. A global costmap is generated such that each cell in the occupancy grid map has a cost value indicating the distance to the nearest obstacle. The cost value of an obstacle cell is set to the maximum of 254, while other cells are set to zero. From each obstacle cell, cost values are propagated outward to its adjacent cells until a user-specified inflation radius is reached. The cost value of each cell is updated by using the exponential function of equation 7.8 [40] below:

$$\text{cost\_value} = exp(-1.0 * w * (dist - ins\_rad)) * (ins\_obs - 1) \qquad (7.8)$$

where w is a scaling factor, $dist$ represents the real distance value measured in metres of a cell from obstacle, ins_rad describes the inscribed radius of a robot and ins_obs is an inscribed inflated obstacle cost. In this case w is set to 10 to give better differentiation between safe and unsafe cells, ins_rad is set to 0.16 in that it corresponds to the physical radius of the UGV and ins_obs is by default set to 254, the maximum cost value of an obstacle cell. The DWA algorithm uses the global costmap to measure the level of safety of a candidate trajectory by identifying the highest cost value along the trajectory's points.

Figure 7.9 shows the example of an obstacle cost map with planned path to a goal location.

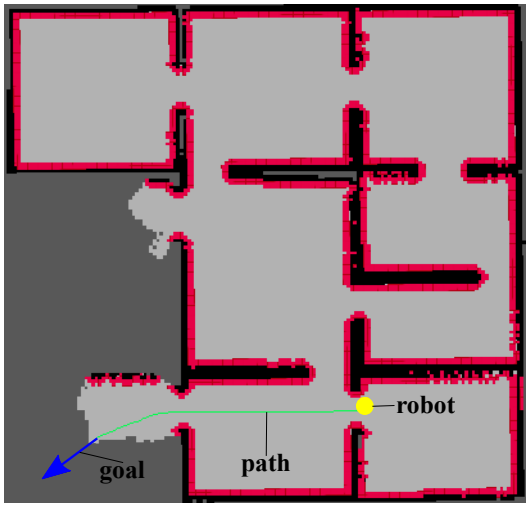


Figure 7.9: Example of an obstacle cost map with planned path to a goal location generated by DWA. Gray color areas are cells with cost value of zero. Red color areas represent cells with some positive cost values indicating unsafe locations. Green line is the planned path as a guidance for the robot to move towards the goal.

7.5 The Dynamic Window Approach

As stated earlier, our main objective is to evolve the π of a DWA algorithm. In order to do that, we need to understand the way the decision model in DWA works. To select an appropriate steering command, a DWA algorithm uses provided information such as potential-field, suggested path and costmap to optimise robot motions to navigate to an immediate goal pose. The performance of DWA relies on how such information is integrated into navigational decision-making by its π. In this section, we elaborate on the details of DWA so that the process of selecting a steering command can be understood clearly.

DWA was introduced by [42] as a reactive local obstacle avoidance scheme that considers the limitations of robot's velocities and acceleration when selecting a steering command. DWA was implemented later for global planning by

[10]. This is done through a combination of a real-time collision avoidance scheme and a local-minima driven free navigation function. Since then, several modifications to DWA algorithms have been implemented by modifying its π [68, 112] to improve its navigational performance.

In turn, DWA proposes a set of circular trajectories, also known as curvatures, for selection of the next best steering command in every short time control cycle, f_c. Each curvature is defined by a pair (v, ω) of translational velocity, v and rotational velocity, ω. For simplicity, it is assumed that the velocities of each curvature are constant in each f_c. Figure 7.10 shows an example of possible curvatures to be selected as a steering command for a differential-drive robot.

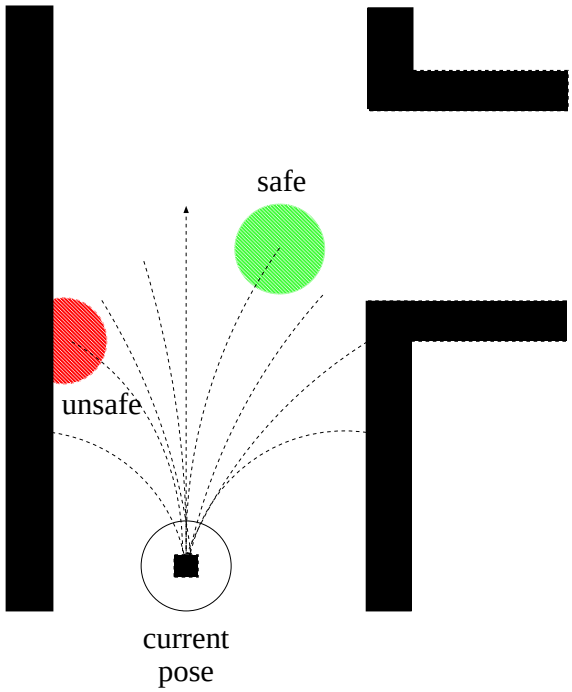


Figure 7.10: A set of candidate circular trajectories.

In general, there are two main processes in DWA: i) identify a set of valid curvatures, and ii) select the best curvature.

Identify valid curvatures - A DWA algorithm is designed such that in every f_c, it can suggest a set of valid curvatures to its π, according to the robot dynamics. Velocity and acceleration limits can be directly included in calculation when

curvature sampling is being done in velocity space. However, searching in velocity space can be time-consuming since the number of velocity pairs is large in a two-dimensional velocity space. As such, the range of velocity pairs are downsized to be in the neighbourhood of the robot's current velocities. Thus, the maximum and the minimum possible value for translational velocity and rotational velocity are varied and capped according to the robot's current velocities. Figure 7.11 shows an example of a dynamic window created around a current velocity pair that samples a set of next velocity pairs (v, ω) inside the window's boundaries as a uniform distribution.

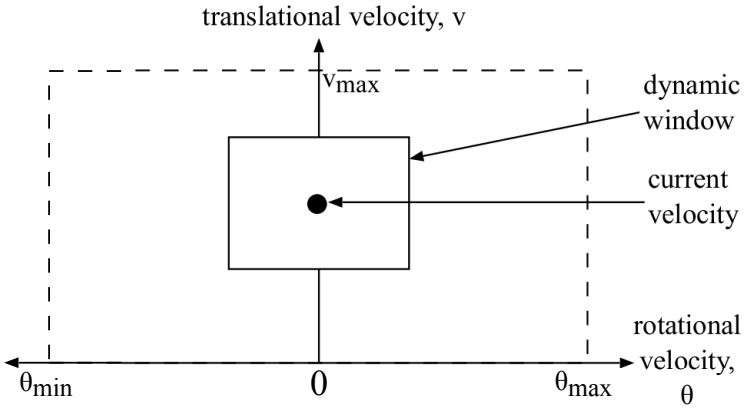


Figure 7.11: An example of velocity pairs distribution.

Select the optimal curvature - The best steering command is selected based on the curvature that maximises the scoring function of the DWA's π. For each candidate curvature, a forward simulation of its trajectory is performed. Results from the forward simulation are i) the estimated end-pose of the robot when applying the curvature's associated velocities in f_c period, and ii) a sequence of points sampled at intervals along the trajectory. The next step is to integrate the resultant trajectory with the auxiliary information described in section 7.4.3 regarding the current state of an underlying global map. Typically, the DWA's π is designed to trade off between several factors such as proximity to goal location, proximity to obstacles, proximity to a suggested path, speed and heading alignment. Such factors can be gathered by projecting the resultant trajectory onto one or more maps. For example, to calculate proximity to obstacles, the trajectory's points can be projected to an obstacle costmap generated by the costGen sub-module and select the

highest cost value among those points. Finally, the DWA's π uses a scoring function to concatenate all related factors to produce an associated score for the candidate trajectory.

In this work the generation of the scoring function of DWA's π is our main concern. With a large variety of existing DWA-based algorithms, there are also a large number of candidate factors to include in the scoring function. Unfortunately, there are no underlying guidelines on which input factors are best to include and vice-versa. In some cases, some combinations of input factors might only carry redundant information that might distort the scoring function. As such, selecting appropriate input factors for the scoring function is an important task that can improve the performance of a DWA algorithm in its intended environment, which in our case is for exploration missions. In the next subsection, we present some scoring functions with their associated input factors that have been used in the literature. Some of these input factors will become the basis of our new BNF grammar used in the evolution of a highly effective DWA π.

7.6 Setting the BNF Grammar for Input Factor Selection

A subset of input factors described in section 7.2 is selected as candidate factors for an evolved scoring function. The selected candidate factors are:

1. *path_dist*
2. *goal_dist*
3. *obst_cost*
4. *vel*
5. *rel_vel*
6. *head*
7. *rel_head*
8. *goal_reg*

The main aim of the evolutionary runs in this chapter is to automatically select and combine the appropriate factors that can produce the best scoring function for exploration tasks. Note that, in addition, our search process simultaneously finds other components of the scoring function. Besides input factors, other components to be evolved include: arithmetic operation between factors and numerical constants. In this experiment, the unary function of each factor is in the form of a linear function only to avoid the search space becoming unmanageably large. [1]

Figure 7.12 below shows a production rule template that can be added into a BNF grammar to evolve factors in a scoring function:

```
1 <expr>::= <factor_name> <op> <expr> | <factor_name>
2
3 <factor_name>::= <factor_1> | <factor_2> | ...
4                | <factor_N>
```

Figure 7.12: A production rules template for evolution of input factors.

The first production rule in line 1 functions as the start symbol of the grammar, where GE initiates the process of genotype-to-phenotype mapping. The non-terminal <expr> is designed to add a factor into a scoring function. Two options are given, both options have equal chance to be selected based on a particular codon integer provided. With the first option, a factor is selected (<factor_name>), a single arithmetic operation is defined (<*op*>) and the mapping process is iterated to add another factor (<expr>). The latter option only performs the task of a factor selection (<factor_name>) and then the mapping process stops. Another production rule in line 3 serves as a complement to the first production rule. This rule is responsible for selection of a particular factor from a list of possible factors determined by the user. <factor_1>, <factor_2> ...<factor_N>, where N is the number of possible input factors, are a generic expression that can represent a particular factor in a syntactically correct form of the specific programming language used to express a π. Both production rules can be integrated with other production rules to generate a working BNF grammar as will be shown in the next section.

[1] The use of linear functions is also consistent with the linear functions used in the literature surveyed above.

7.7 Experiments

This section describes experiments being conducted to evolve DWA's policies for exploration tasks. Generally, we divide the experiments into two phases:

1. learning phase, and
2. application phase.

In the learning phase, we evolve DWA's policies under three evolution schemes:

1. Evolution of numerical constants in a π (CONST_EVO).
2. Evolution of factors in a π with random initial population (FACRAN_EVO).
3. Evolution of factors in a π with partially pre-determined initial population (FACDET_EVO).

Subsequently, we validate a set of high-ranking evolved DWA policies in the application phase under various exploration conditions:

1. Simulated complex environments with perfect localisation.
2. Simulated complex environments with SLAM implementation.
3. Real-world environments.

7.7.1 Learning Phase

Description of Evolution Schemes

The first evolution scheme of CONST_EVO requires only the numerical constants of a DWA's π to be evolved. This scheme is used as the baseline of evolution performance. The π is set with a weighted sum DWA cost function as implemented in ROS packages [40] (hereafter, the cost function is known as

the scoring function). The structure of the scoring function is described as in equation 7.9 below:

$$scorefunc = w1 * path\_dist + w2 * goal\_dist + w3 * obst\_cost \tag{7.9}$$

In the equation, there are three factors under consideration: *path_dist*, *goal_dist* and *obst_cost*. Each factor is multiplied with a numerical constant labelled as *w1*, *w2* and *w3*, respectively. The final cost value is calculated as a weighted sum of all factors. Given a list of possible trajectories, the best solution is associated to a trajectory with the least positive cost value. In the CONST_EVO scheme, focus is given to the optimisation of the value of those three numerical constants i.e. *w1*, *w2* and *w3*, to get the best trade-off between factors.

The second scheme (FACRAN_EVO) and the third scheme (FACDET_EVO) are the core subjects under study in this chapter. Both schemes select appropriate factors of a DWA's π automatically, define suitable arithmetic operations between factors and set numerical constants accordingly. The only difference between the two schemes is that the former scheme (FACRAN_EVO) is set with a random initial population to start the evolution progress, while the latter scheme (FACDET_EVO) is added with a partially pre-determined initial population. FACRAN_EVO and FACDET_EVO schemes use a BNF grammar as depicted in figure 7.13, concatenating the production rules shown in section 7.6. This grammar is intended for the development of a scoring function with minimal prior knowledge.

Since the evolution time is very crucial in most of our experiments, finding the best solution within small number of generations is of great importance. As such, for FACDET_EVO, we hypothesize that the evolution progress can be improved in terms of the convergence rate if the initial population is filled with a set of pre-evolved individuals. Those pre-evolved individuals are evaluated with a simple navigational problem that requires shorter time to evolve than a solution to our goal problem. In our case, the pre-evolved individuals are gathered from the evolution of a navigation to a single target location in a known environment. Obviously, this navigation task is much simpler and less

time-consuming than exploring in an unknown environment.

```
1  <expr>    ::= <wfactor> <op> <expr>
2              |   <wfactor>;
3
4  <wfactor>::= <num_A> * path_dist
5              |   <num_A> * goal_dist
6              |   <num_B> * obst_cost
7              |   <num_A> * vel
8              |   <num_A> * rel_vel
9              |   <num_A> * head
10             |   <num_A> * rel_head
11             |   <num_A> * goal_reg
12
13 <num_A>   ::= <number10><number10>.<number10>
14             | 100.0
15
16 <num_B>   ::= 0.<number10><number10>
17             |   <number10>.<number10>
18             |   10.0
19
20 <number10>::= 0|1|2|3|4|5|6|7|8|9
21
22 <op>   ::= +|-|*
```

Figure 7.13: A complete BNF grammar for FACRAN_EVO and FACDET_EVO schemes, enabling the evolution of factors to be included in a DWA's π.

Configuration of Evolution Schemes

Common evolutionary set up - In this experiment, the common set up for all of the above evolution schemes is shown in table 7.1. Most items in the configuration are similar to the previous experiments in this thesis. The only difference is that the number of generations and the population size are fixed for all schemes. Our intention is to observe the convergence rate of each evolutionary scheme under the same configuration even though the search space size of each scheme differs in their underlying grammar.

Table 7.1: GE configuration for all runs.

Item	Value
Number of generations	60
Population size	30
Search engine	GA
Selection type	Tournament
Crossover type	Effective crossover
Mutation type	Point mutation
Probability of crossover	0.95
Probability of mutation	0.05
Elitism	Yes
Average time of fitness calculation per individual	120 seconds

The evolutionary process - We implement the *embedding-simulation-collation* cycle from the previous chapters to evaluate each candidate solution (individual) produced by GE in all evolution schemes. In *embedding*, each individual is translated into a working C++ source code of a DWA's π. We modified the standard package of the DWA controller in ROS [40] such that the scoring function of the DWA's π is separated as one source file. This is done to ease the process of re-compiling the source code every time a new individual is supplied by GE. Figure 7.14 shows the flow of the *embedding* part.

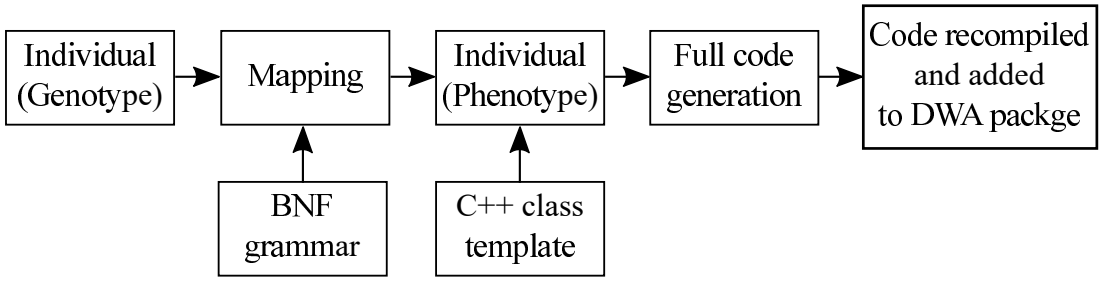
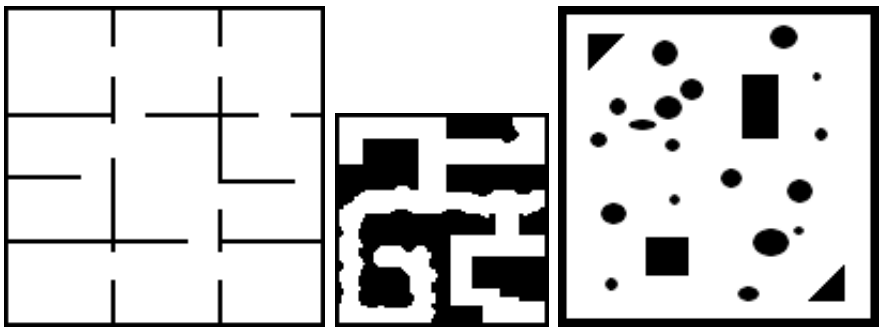


Figure 7.14: *Embedding* - An individual, represented as an array of codon integers, known as genotype, is mapped into a phenotype form by using a BNF grammar as the mapping agent. The phenotype represents a single scoring function equation in the form of C++ code compatibility. The phenotype is then combined with a template class file to form workable C++ code. The generated code is then re-compiled before being passed into ROS's DWA package.

In *simulation*, a robot provided with a particular DWA's π as compiled in *embedding*, is required to explore an initially unknown area (from the robot's

point of view). The robot uses the DWA's π to plan its movement sequentially, while generating an incremental grid map of the explored area within an allowable time frame. The more the robot explores, the higher the fitness given. Three different simulations are run per π. All simulations differ in terms of the feature map used. Each simulation contains an environment with simple features. In particular, simulation 1 uses an office-like map as in figure 7.15(a). This map presents narrow doors and a simple office-room layout. In simulation 2, the feature map of figure 7.15(b) is used. In this map, the robot is required to move on a trail of ant-nest-like paths to test its ability to navigate in narrow corridors. Finally, we use map of figure 7.15(c) to execute simulation 3. This map features an obstacle-dense environment such that the robot can be tested in exploring unstructured environments. The intention of running multiple simulations with simple maps is to train the robot in complex navigation skills by exercising a variety of simple navigation skills.



(a) (b) (c)

Figure 7.15: A set of learning maps.

Note that in the *simulation* stage of evaluation, we replace the Gmapping SLAM module with a perfect localisation. We avoid the use SLAM since it is very computationally intensive which slows down simulation. In addition, we want to avoid false drops in fitness due to map construction errors that might be produced by the SLAM module. However, we expect there will be no major problem in transferring the DWA's π into a SLAM-based system as long as the SLAM module can handle localisation task reasonably well. This issue will be further discussed in the application phase section.

Meanwhile, in the *collation* stage of evaluation, robot performance on each simulation run is calculated as per equation 7.10.

$$P_{sim} = \left[1 - \left(\frac{area_{total} - area_{explored}}{area_{total}}\right)\right] * weight \tag{7.10}$$

where $area_{total}$ is the ground-truth area of a reference map and $area_{explored}$ is the area explored by the robot measured at the end of each simulation. A $weight$ factor is added to the equation to penalise the robot if collision occurs during the simulation. Since collisions can occur only once in every simulation, $weight$ has a value of 0.5 if robot collides, or 1.0 otherwise. [2]

Finally, the overall fitness of an individual is gathered from the aggregation of P_{sim} from each simulation. The overall fitness of a candidate solution is calculated as in equation 7.11.

$$fitness = \sum_{x=1}^{3} P_{sim(x)} \tag{7.11}$$

where x is the simulation run number.

Note that, P_{sim} in equation 7.10 produces a normalised value between 0.0 to 1.0. We avoid direct use the value of $area_{explored}$ to represent the exploration performance because it will create bias towards maps with higher free area. Normalising $area_{explored}$ by subtracting from $area_{total}$ will fairly distribute P_{sim} when aggregating all simulation performance into one scalar fitness value.

7.7.2 Application Phase

This section refers to the application of evolved controllers on various exploration conditions for investigating its application generality. One of the main concerns in the evolutionary robotics research is the reliability of results

[2] We only consider one collision per simulation as we assume once the robot collides, it cannot move and the exploration task is stopped automatically.

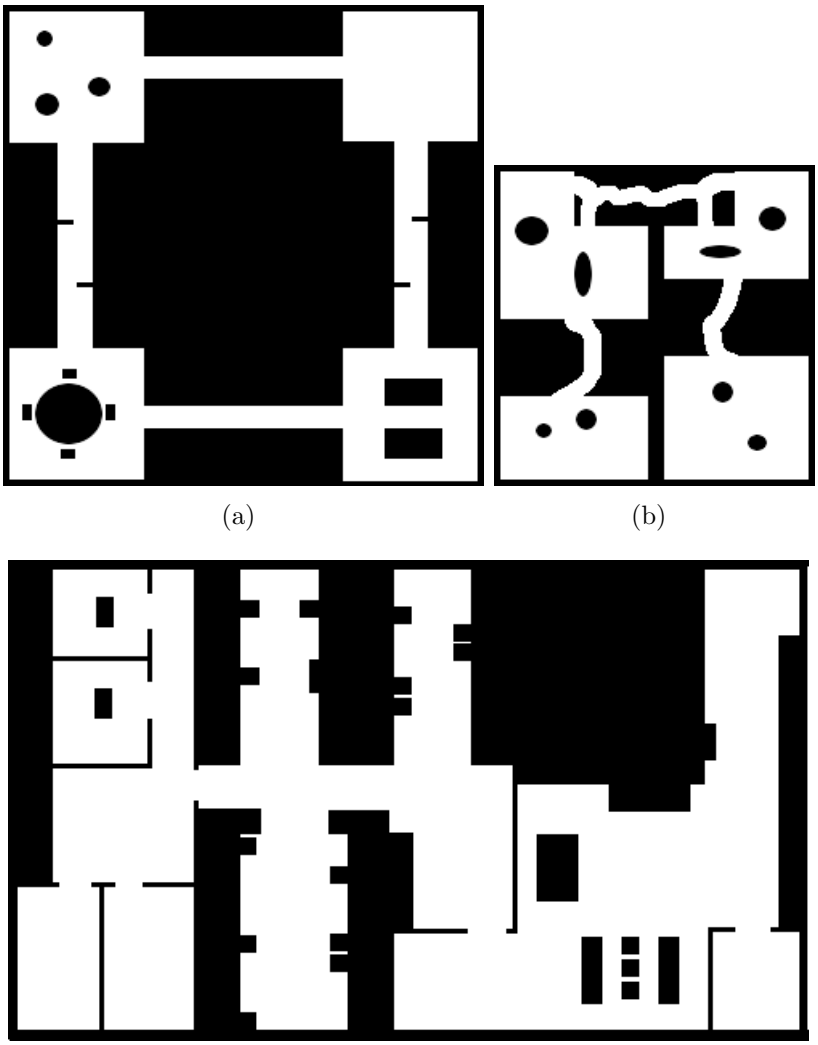
found in the learning phase to be applied to other environments. Therefore, we investigate the transition of the controllers in stages as follows:

1. Map transition.
2. Localisation transition.
3. Platform transition.

Map Transition

In the learning phase, we used three simple feature maps as shown in figure 7.15 to train the robot to explore optimally. Our intention is that, the robot learns a specific navigational behavior from each feature map, for example, moving in narrow corridors, from the second map. By combining all learned behaviors (by the mean of fitness function), the robot should be able to work in a complex environment that combines multiple map features. As such, the first transition to be examined is the ability of the evolved controllers to perform exploration in complex environments that may consist of two or more types of simple features.

To test this ability, for each controller, we run three separated exploration tasks on three simulated complex maps shown in figure 7.16. Initially, the robot knows nothing about the environment. For each task, the robot is given an ample time to explore the environment. The allocated maximum time is determined by the means of time taken from 30 runs for the robot to explore with tele-operated control. At the end of each run, the exploration result is collected and analysed based on the following performance indicators: i) collision status, ii) completion status, and iii) exploration coverage. For each tested controller, 30 replicate runs are conducted to obtain a statistical summary of the controller's performance.



(a) (b)

(c)

Figure 7.16: A set of complex maps for map transition observation.

Localisation Transition

In the learning phase, we configure the robot with a perfect localisation setting for the reasons mentioned before. However, in real world applications, this assumption of perfect localisation is rarely true due to the existence of sensor noise and, sometimes, systematic inaccuracies. To measure how well our learned individuals cope with the challenge, we apply the SLAM module, described in section 7.4.1 to manage imperfect localisation and mapping. To conduct the experiment of this section, we set the robot's odometry with the

maximum of 5% linear positioning error and 10% angular positioning error. After that, we repeat experiments in the map transition section with this new configuration.

Platform Transition

Finally, the ultimate objective of the application phase is to transfer the resultant controllers from simulated environments to real world environments. In this platform transition section, we replace the simulated platform with the Turtlebot platform discussed in section 7.3. After the platform migration is done, we run an exploration task for the real robot to explore a given environment. Bear in mind, no information about the environment is known initially by the robot. Once again, all three best controllers of each evolution scheme and the handwritten controller are used for comparison purpose.

7.8 Results and Discussion

7.8.1 Learning Phase Results

In this section, we report the evolutionary performance of each evolution scheme mentioned in section 7.7.1. We observe the convergence rate of each scheme by analysing the highest fitness achieved at each generation. The maximum fitness possible to achieve is 3.000, an aggregation of three $P_{sim(x)}$. Meanwhile, the minimum fitness value is 0.000. Note that, we set the baseline fitness as the median fitness of 20 replicate runs of the DWA's π with the default configuration (DWAD) from [40]. Thus, we can say that any controller with higher fitness than the baseline fitness has better exploration performance than DWAD. In this experiment, we replicate 20 runs for each evolution scheme, a sample large enough for statistical validity given our effect sizes. Figures 7.17 to 7.19 plot the learning results of all evolution schemes in terms of the highest fitness achieved at each generation for every replicate run. Together with

each graph, the corresponding standard deviation of fitness at each generation between the replicate runs is also presented.

Some commonalities appear in all evolution schemes. Obviously, the highest fitness trends upward across generations. The rate of fitness growth for CONST_EVO, FACRAN_EVO and FACDET_EVO is high in the first-half generations and slows down in the second-half generations. This shows that GE is able to converge in searching for good solutions under various sizes of search space and initial condition. The reported best end-of-run fitness in each replicate run shows a consistent output above the value of 2.9. The standard deviation of fitness at the final generation of all evolution schemes indicates a value below 0.04, which indicates reasonable consistency across runs. The result portrays the ability of GE to converge the fitness near the maximum possible fitness (value of 3.0 in this case) within a given small number of generations. Note that, CONST_EVO converges the fastest among all schemes – a likely effect of its relatively small search space.

In contrast, there is a number of interesting points that uniquely characterise a particular evolution scheme. Firstly, the highest fitness at the initial generation. We observe that CONST_EVO starts at a higher value with a median value of the best individuals from the runs at 2.8982, in contrast the corresponding values for FACRAN_EVO and FACRAN_DET are 2.4658 and 2.8278 respectively. This is, likely, due to the fact that the search space of CONST_EVO is smaller than the other two schemes. At the opposite end, FACRAN_EVO produces the most varied result among the three schemes. It can be observed from the fitness standard deviation of 0.22475 at the initial generation. Moreover, some replicate runs of FACRAN_EVO set off with the fitness value near the baseline of 1.8. This situation is perhaps predictable given the size of the search space for FACRAN_EVO. When observing the statistics for the first generation, FACDET_EVO behaves much better than FACRAN_EVO. The fitness standard deviation reduces to 0.1995, or an 11% improvement. We also can see that the median fitness of the best FACDET_EVO individuals at generation 1, with the value of 2.8278, which is higher that of FACRAN_EVO of 2.4658. This fact confirms our prediction that a partially pre-determined population can form a good start of an evolution.

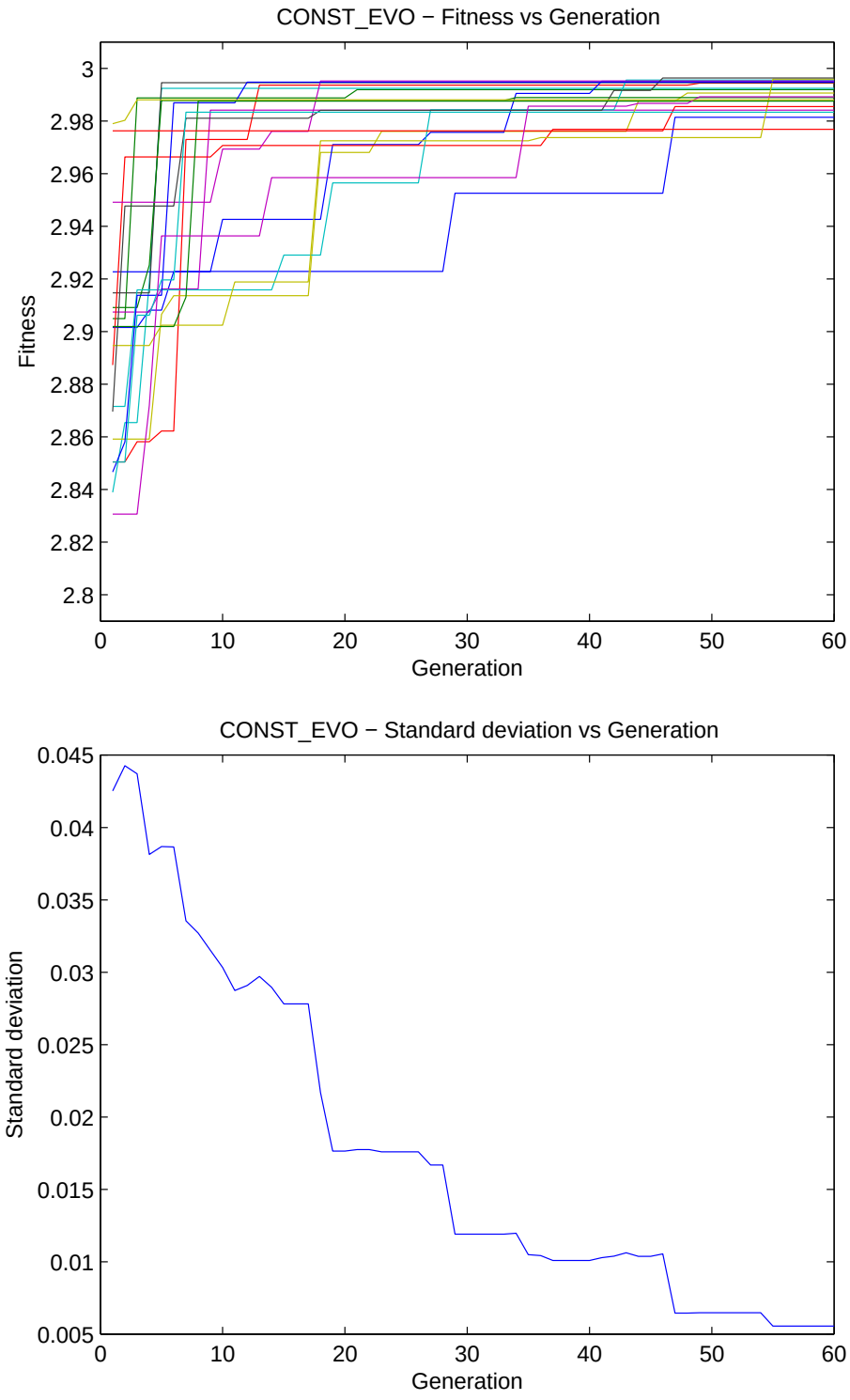


Figure 7.17: CONST_EVO scheme: The top graph plots the highest fitness at each generation for 20 replicate runs. The bottom graph shows the standard deviation of the highest fitness at each generation between the replicate runs.

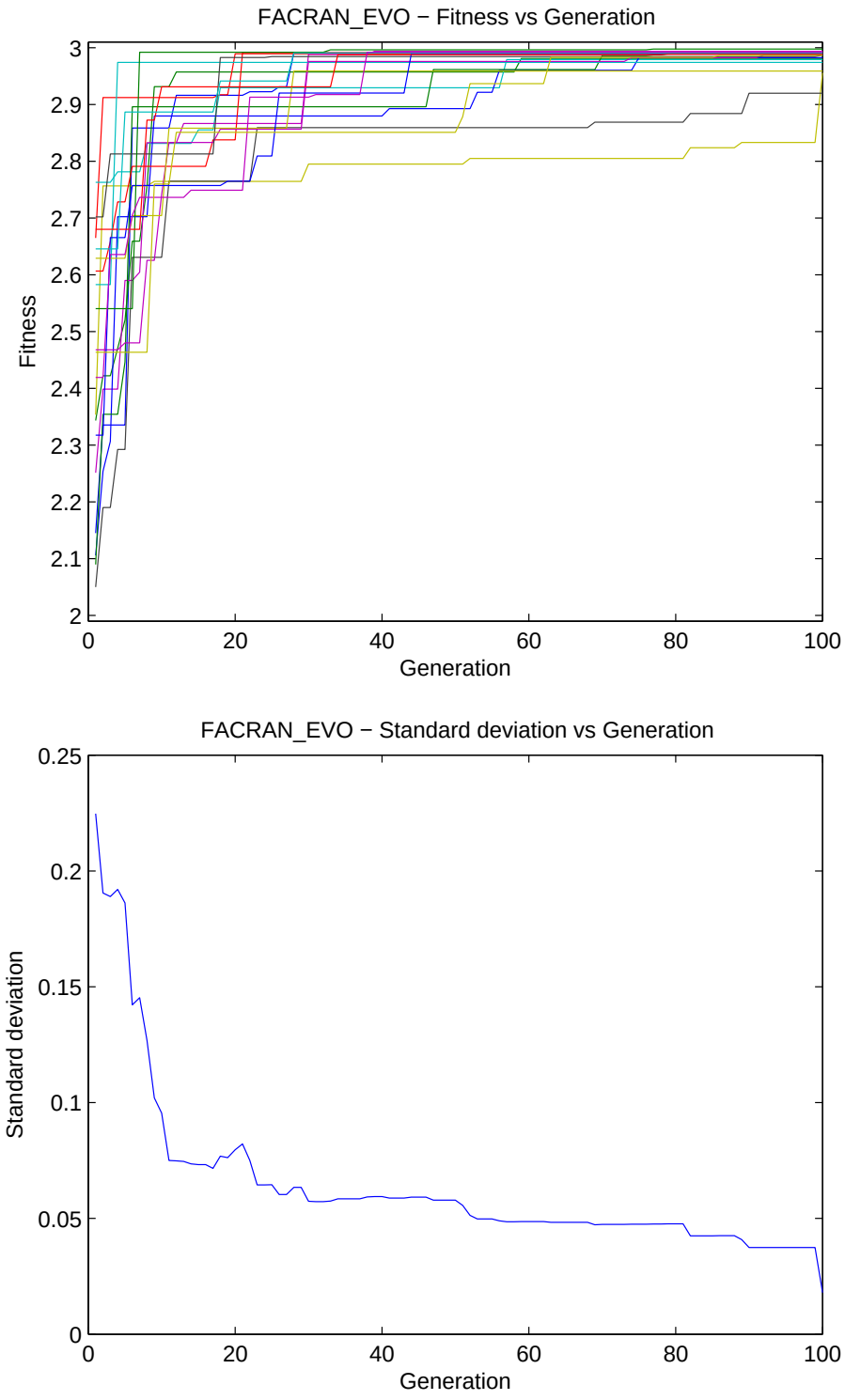


Figure 7.18: FACRAN_EVO scheme: The top graph plots the highest fitness at each generation for 20 replicate runs. The bottom graph shows the standard deviation of the highest fitness at each generation between the replicate runs.

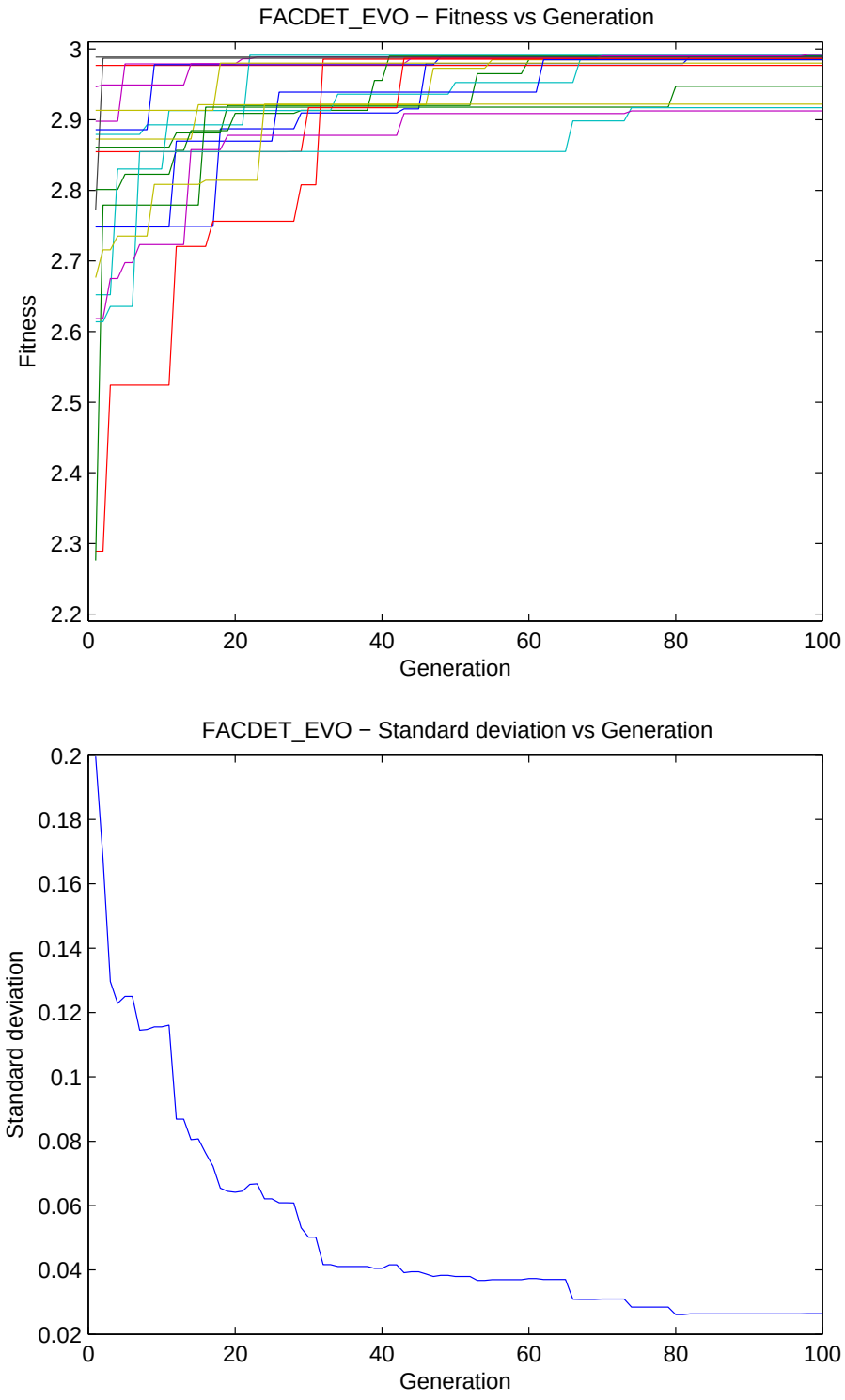


Figure 7.19: FACDET_EVO scheme: The top graph plots the highest fitness at each generation for 20 replicate runs. The bottom graph shows the standard deviation of the highest fitness at each generation between the replicate runs.

A second characteristic to measure is the convergence to the steady state. Since simulation time is crucial for the evolution of motion control for the exploration task, finding a set up that can produce a significant result within a shorter time is very important. One indicator to measure the rate evolution's progress towards a steady state condition is to observe convergence. To measure the steady state condition, we use the median of the highest fitness at each generation for all evolution schemes depicted in figures 7.20 to 7.22 as the benchmark. We set the steady state value to 2.94, or a 2% error of the maximum fitness. For the CONST_EVO scheme, the steady state value is achieved at generation 6 (the medians of the runs). Meanwhile, this value is gained at generation 30 and 39 for the FACRAN_EVO and FACDET_EVO schemes, respectively. While the steady state condition may vary according to the user design, the statistical data suggest that the convergence rate of an evolution run is related to the size of the search space.

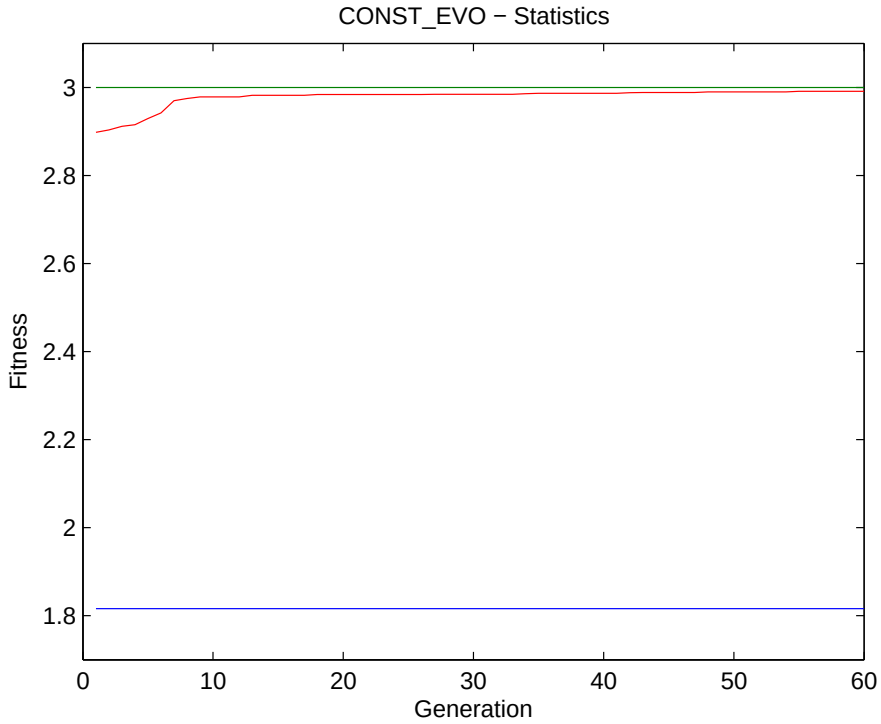


Figure 7.20: Median of the highest fitness at each generation aggregated from 20 replicate runs for CONST_EVO scheme.

Last but not least, comparing the graphs in figures 7.20, 7.21 and 7.22, the final results from the FACRAN_EVO and FACDET_EVO are slightly outperformed

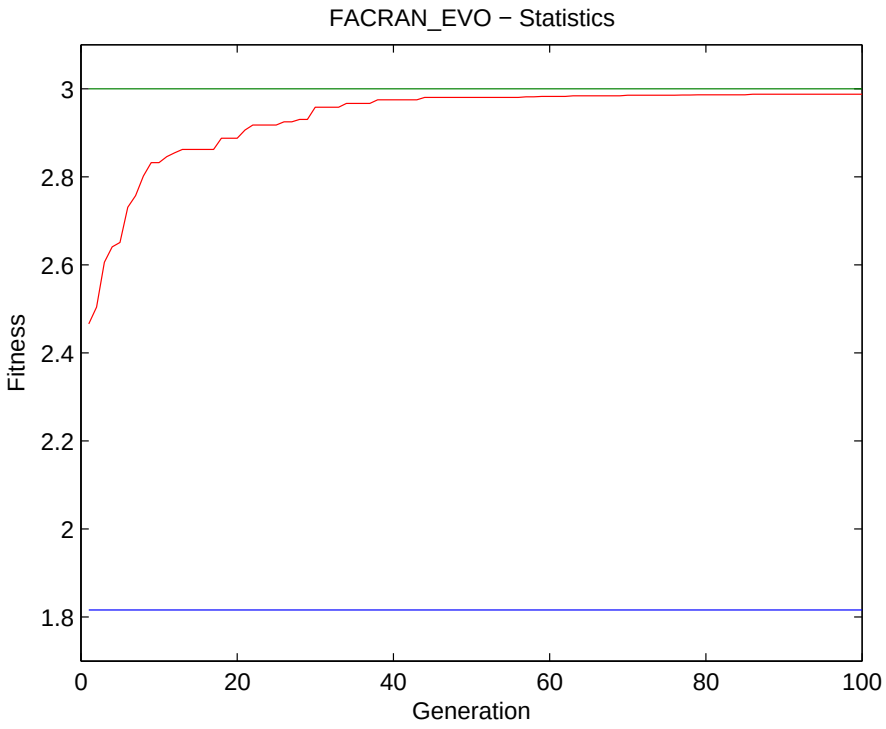


Figure 7.21: Median of the highest fitness at each generation aggregated from 20 replicate runs for FACRAN_EVO scheme.

by the CONST_EVO. This is not the result we initially expected. One possible reason can be analysed by looking at resultant scoring functions suggested by both FACRAN_EVO and FACDET_EVO. One main point to remark is "What input factors are selected by the evolution schemes?". Table 7.2 and 7.3 aggregate all factors selected on each replicate run of FACRAN_EVO and FACDET_EVO, respectively.

Note that, on both tables of FACRAN_EVO and FACDET_EVO, the most selected factors are *goal_dist* and *obst_cost*. In the third place is *path_dist*. The results indicate that most of the time, the selected factors in both schemes are almost the same as the default factors available in CONST_EVO. The results suggest that although a number of factors are given equal opportunity to appear in a scoring function, all evolution schemes are consistent in choosing the default factors as the most influential factors to the performance of the robot's exploration task. Even though our initial hypothesis suggest that additional factors may improve the exploration performance, the final results

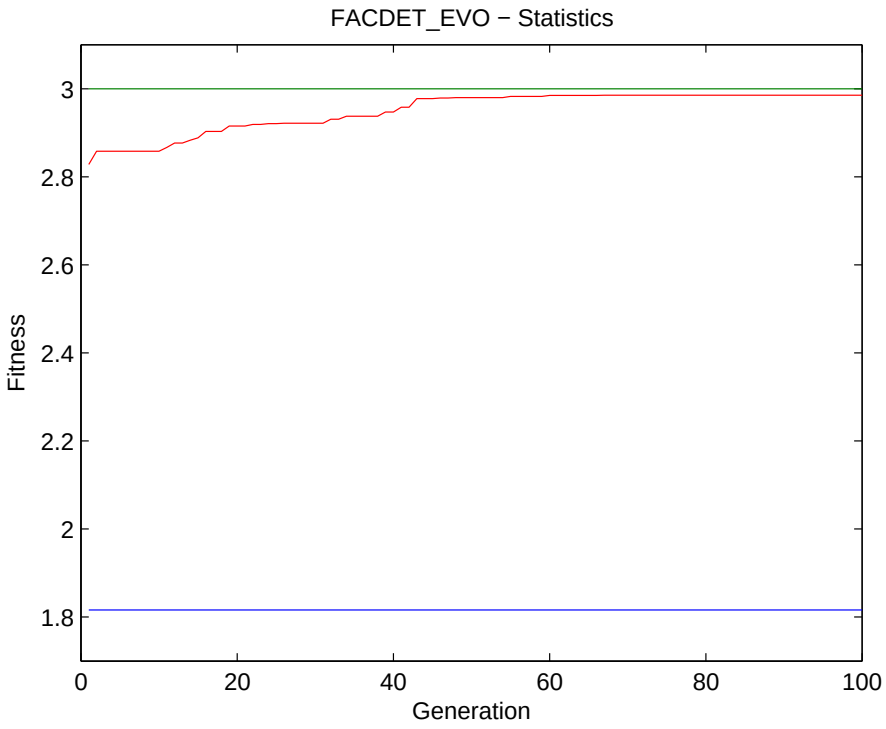


Figure 7.22: Median of the highest fitness at each generation aggregated from 20 replicate runs for FACDET_EVO scheme.

have shown otherwise. The results appear to indicate why the final fitness of FACRAN_EVO and FACDET_EVO are similar to CONST_EVO.

In summary, all evolutionary runs under all three schemes yielded reasonable results with the best of all runs exhibiting very similar exploration performance. All three schemes exhibited reasonable convergence – vital when simulation overheads dictate small populations and moderate numbers of generations. Finally all three exhibited convergence in the selection of input factors though the exact input factors used among some good individuals and their associated constants varied somewhat.

Table 7.2: FACRAN_EVO: Table of selected input factors on each on the replicate runs.

Run	path_dist	goal_dist	obst_cost	vel	rel_vel	head	rel_head	goal_reg
1		X	X					X
2		X	X					
3		X	X					
4		X	X					
5		X	X					
6	X	X				X		
7		X	X					
8	X	X	X					
9		X	X					
10	X	X	X					
11		X	X					
12		X	X		X			
13	X	X						
14		X	X		X			
15	X	X						
16		X	X					
17		X	X					
18		X	X	X				X
19		X	X					
20		X	X					X
Total	5	20	17	1	2	1	0	3

7.8.2 Application Phase Results

In this section, we report the results of applying selected DWA policies obtained from the learning phase on various exploration conditions as described in section 7.7.2. One DWA's π function is selected from each evolution scheme for this experiment. The DWA's π functions are selected based on an individual that has the nearest fitness to the median fitness of each evolution scheme. Besides, we also include the default configuration of DWA's π (DWAD) for benchmarking. The scoring functions of the chosen DWA policies are shown in table 7.4. We compare exploration performance between these DWA policies.

Table 7.3: FACDET_EVO: Table of selected input factors on each of the replicate runs.

Run	path_dist	goal_dist	obst_cost	vel	rel_vel	head	rel_head	goal_reg
1		X	X					
2	X	X	X					
3	X	X						
4		X	X					
5		X	X				X	
6		X	X					
7		X	X					
8		X	X					
9		X	X					
10	X	X	X	X				
11	X	X	X	X				X
12		X	X					
13		X	X					
14		X	X					
15		X	X					X
16	X	X	X				X	
17	X	X	X					
18	X	X	X		X		X	
19	X	X	X		X			
20	X	X						
Total	9	20	18	2	2	0	3	2

Condition 1: Map Transition

For the first transition, we compare exploration performance of above DWA policies when the robot is transferred to environments with multiple features. Three different simulated complex environments shown in figure 7.16 are generated. For each π, 30 replicate runs are conducted to obtain statistical data for each policy performance. Three performance indicators are used to indicate exploration effectiveness of each tested π on each environment. The performance indicators are i) collision status, ii) completion status, and iii) exploration coverage.

Table 7.4: Table of scoring functions of selected DWA policies. DWAD represents the default DWA's π with a pre-defined scoring function. Meanwhile, CONST_EVO, FACRAN_EVO and FACDET_EVO represent DWA policies with an evolved scoring function from each evolution scheme.

Scheme	DWA scoring function
DWAD	$32.0 * path\_dist + 24.0 * goal\_dist + 0.01 * obst\_cost$
CONST_EVO	$18.4 * path\_dist + 46.6 * goal\_dist + 0.26 * obst\_cost$
FACRAN_EVO	$82.5 * goal\_dist + 1.37 * obst\_cost + 47.6 * rel\_vel$
FACDET_EVO	$89.6 * goal\_dist + 0.58 * obst\_cost$

Collision status

Collision status indicates whether the robot collides with obstacles or not during exploration. Figure 7.23 shows the accumulated collision count of the robot running on all three environments with four different DWA policies. From the graph, the robot with DWAD π collides in 35 out of 90 exploration runs. This number suggests that the probability of the robot colliding in these environments is as high as 38.9%. Referring to the DWAD's scoring function in table 7.4, we can see that the weight given to *obst_cost* is relatively low. This has caused the robot to move into the danger zone with low penalty in the scoring function. Meanwhile, CONST_EVO reports 3 collisions out of 90 runs or about 3.3%. The reduction in the number of collision is clearly contributed from the higher weight given to *obst_cost*. On the other hand, the robots with FACDET_EVO and FACRAN_EVO policies record zero collision during all exploration runs. This fact indicates that the safety measurement in these DWA policies work really well to avoid the robot navigating through the danger zone. In the scoring function of both DWA policies, we can observe that the weight of *obst_cost* has been increased significantly to alert the robot to nearby obstacles. In this case, we can say that all evolved DWA policies are better than DWAD's π in terms of safety performance.

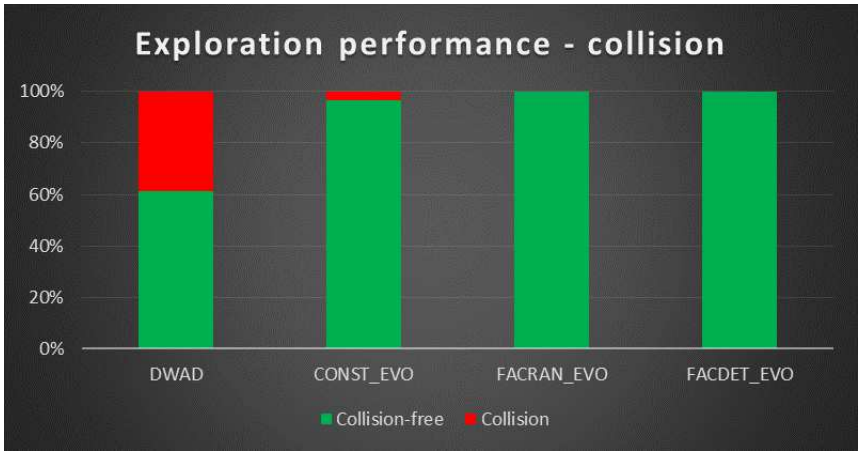


Figure 7.23: Map transition experiment: Accumulated collision status from all exploration runs of all environments.

Completion status

Completion status measures the capability of a DWA's π function to drive the robot to explore an environment completely within a given time-limit. In this experiment, an exploration is considered complete if the robot is able to explore at least 95% of the environment within 310 seconds for map of figures 7.16(a) and 7.16(c), and within 250 seconds for map of figure 7.16(b). Figure 7.24 indicates the completion status of all DWA policies. Comparing all four policies, DWAD shows the least completion rate with only 26 out of 90 runs able to complete exploration. This number indicates that the probability of DWAD completing exploration without collision and within our time-limit is only about 29%. From the balance of 64 incomplete exploration runs, collision occurs in 35 of these runs. With evolution, the completion rate is able to increase significantly. FACDET_EVO reports the highest rate of complete exploration with 49 out of 90 runs or about 54% completion. CONST_EVO is able to complete exploration in 42 out of 90 runs or about 47% completion. Finally, FACRAN_EVO reports 44% completion indicating 40 out of 90 runs have complete exploration. The completions made by all evolved DWA π is almost double that of DWAD's π. The remaining incomplete runs of all evolved DWA π due to run out of time have no collision occuring during exploration. All evolved DWA π outperform DWAD's π in terms of exploration

completion.

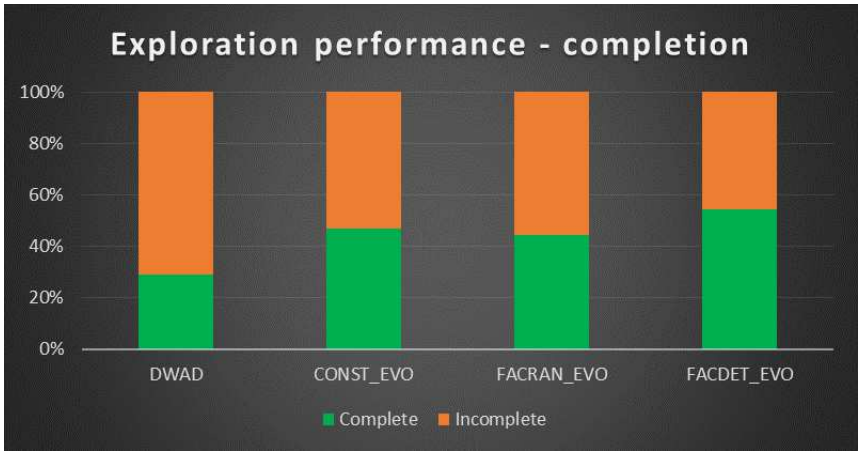


Figure 7.24: Map transition experiment: Accumulated completion status from all exploration runs of all environments.

Exploration coverage

The third performance indicator, exploration coverage, measures total explored area at the end of an exploration run. This performance indicator can be represented graphically with boxplots as shown in figures 7.25 to 7.27. Each figure represents results from ground-truth of figure 7.16.

First, figure 7.25 indicates the boxplot of exploration area achieved by all DWA policies when tested on the ground-truth environment of figure 7.16(a). All policies have the median of explored area of about $350m^2$. However, CONST_EVO, FACRAN_EVO and FACDET_EVO show better consistency as their boxplots are comparatively short compared to DWAD with a standard deviation below $12m^2$. DWAD has a taller boxplot, thus has a higher standard deviation of about $101m^2$ which confirms its relative inconsistency. Note that, this environment has the least complex features compared to the other two test environments, as such the median performance of evolved DWA policies is not very different from DWAD.

Moving to the second ground-truth environment of figure 7.16(b), the boxplot in figure 7.26 shows the performance of all DWA policies in terms of exploration

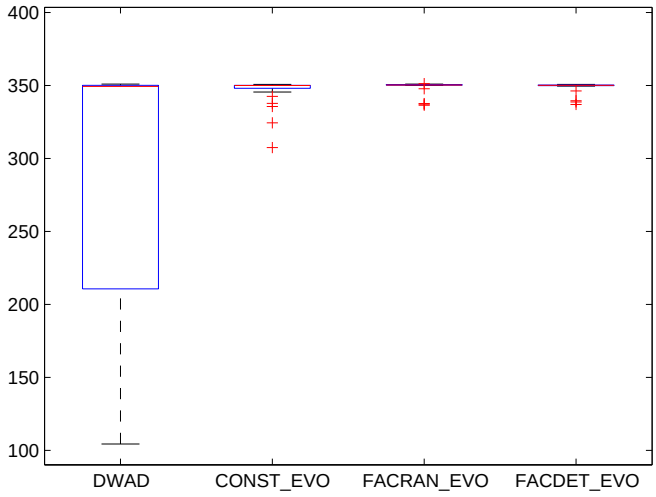


Figure 7.25: Map transition experiment: Boxplot of explored area (m^2) at the end of exploration runs on the ground-truth environment of figure 7.16(a) for all DWA policies.

coverage. DWAD's performance is the most varied with the median explored area of $129m^2$ - only half of the total area. CONST_EVO, FACRAN_EVO and FACDET_EVO report higher median explored area at $191m^2$, $230m^2$ and $247m^2$, respectively. The median explored area of all evolved DWA policies are above 75% of the total area. In terms of exploration consistency, the tallest boxplot is produced by DWAD. Meanwhile, CONST_EVO, FACRAN_EVO and FACDET_EVO have similar boxplot sizes that are smaller than DWAD. This indicates that evolved DWA policies are more consistent than DWAD in environments of different complexity.

Figure 7.27 presents the boxplot of the explored area by all DWA policies running on the ground-truth environment of figure 7.16(c). Again, it is obvious that DWAD remains the least performed controller. The median explored area of DWAD stood at only $124m^2$. The box plot of DWAD is comparatively tall with 50% of the runs producing an explored area between $188m^2$ (75th percentile) and $77m^2$ (25th percentile). FACDET_EVO reports the highest median explored area at $219m^2$. The box plot of FACDET_EVO is comparatively low compared to others with the distribution of the explored area of 50% runs varying in a range of only $20m^2$ (the upper quartile at

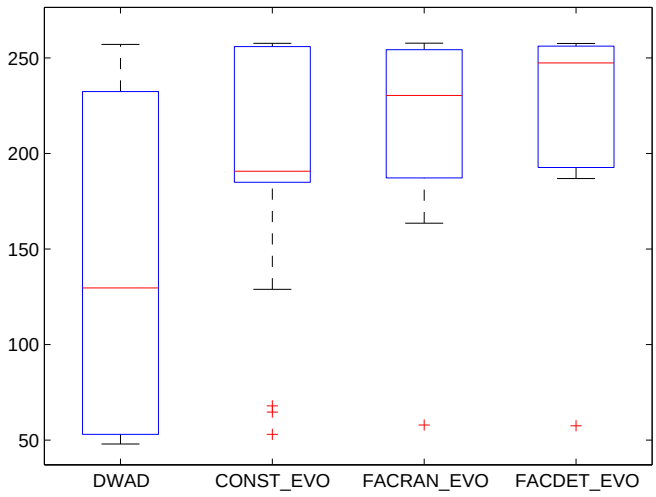


Figure 7.26: Map transition experiment: Boxplot of explored area (m^2) at the end of exploration runs on a ground-truth environment of figure 7.16(b) for all DWA policies.

$230m^2$ and the lower quartile at $210m^2$). This indicates that FACDET_EVO is the most stable controller when exploring this environment with a higher probability to finish with a higher explored area. The performance of CONST_EVO and FACRAN_EVO policies are also promising with the median explored area at $211m^2$ and $202m^2$, respectively.

Map transition overall performance

From the results of map transition experiment, all evolved DWA policies outperform the default DWA policy in terms of its capability to improve exploration performance i.e. collision avoidance, probability of exploration completion and the average explored area. It shows that the DWA policies gathered from the learning phase are able to be transferred from a simple environment to a complex environment. The overall performance in the three experiments of all DWA policies is consistent. To compare the effect of evolution on exploration performance, a T-test analysis between every two policies was conducted for all involved DWA policies. Table 7.5 shows the test

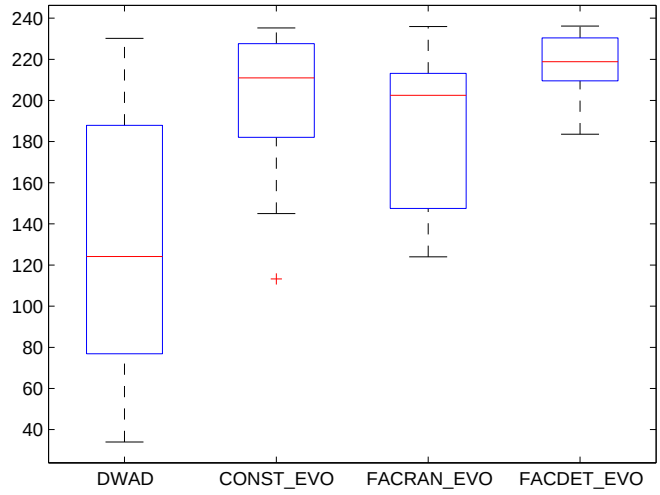


Figure 7.27: Map transition experiment: Boxplot of explored area (m^2) at the end of exploration runs on a ground-truth environment of figure 7.16(c) for all DWA policies.

results on samples from the robot exploration coverage on the environment of figure 7.16(a) to 7.16(c). From the table, there is a significant difference in the exploration performance between DWAD and all evolved DWA policies with p-value below the significant level, $p< 0.05$. Meanwhile, there is no significant difference in most of the results among all evolved DWA policies indicating the consistency of the evolved solutions in this experiment. There is also a significant difference between FACDET_EVO and the others particularly on the more complex maps which may indicate an advantage of being able to chose factors and in starting search with sensible defaults.

Table 7.5: Map transition experiment: T-test results to analyse the variation of the exploration coverage performance arises among all DWA policies.

Policy 1	**Policy 2**	***p*-value**		
		Map A	**Map B**	**Map C**
DWAD	CONST_EVO	**0.0032**	**0.0021**	**9.1438E-7**
DWAD	FACRAN_EVO	**0.0019**	**6.5483E-5**	**4.7122E-5**
DWAD	FACDET_EVO	**0.0019**	**2.9517E-6**	**1.2536E-10**
CONST_EVO	FACRAN_EVO	0.1392	0.3122	0.1779
CONST_EVO	FACDET_EVO	0.1602	**0.0411**	**0.0114**
FACRAN_EVO	FACDET_EVO	0.8526	0.2162	**8.9829E-5**

Condition 2: Localisation Transition

Our second transition is to apply a localisation algorithm to cater pose uncertainty arising from odometry errors. We apply the Gmapping SLAM algorithm in our mapping module to help cope with this uncertainty, so that an accurate map can still be generated. As with the map transition, we compare the performance of DWAD, CONST_EVO, FACRAN_EVO and FACDET_EVO on the environments of figure 7.16. The number of replicate runs per controller per environment is 30. Three performance indicators i.e. collision, completion and exploration coverage are measured.

Collision status

Figure 7.28 reveals the accumulated collision data collected from all runs when the SLAM algorithm is applied. From the graph, DWAD shows the worst performance with 54.4% collision probability or equal to 49 collisions in 90 runs. The collision probability is reduced drastically in the evolved controllers. CONST_EVO reports 8.8% collision probability or equal to 8 collisions per 90 runs. This is followed by FACDET_EVO with 2.2% collision probability, indicating 2 collisions in 90 runs. Meanwhile, the best performing controller is FACRAN_EVO with zero collisions. Comparing this result with the result from the map transition, it can be seen that the performance of the robot in terms of collision probability has been decreased by 15.5% for DWAD, 5.5% for CONST_EVO and 2.2% for FACDET_EVO, while FACRAN_EVO shows no performance reduction. This indicator suggests that DWAD is the most affected controller when the localisation transition is done. Meanwhile, evolved controllers can maintain their performance with small error tolerance.

Completion status

Figure 7.29 maps the completion status for all DWA controllers. We maintain the indicator for a complete exploration in that, the robot must explore

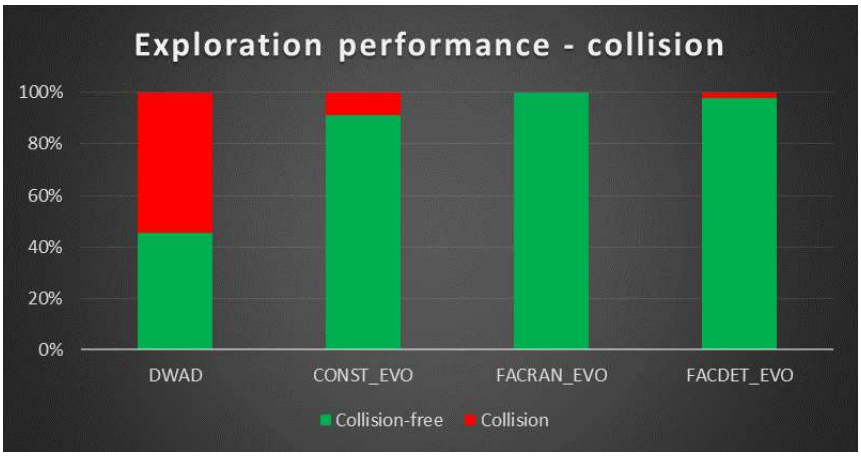


Figure 7.28: Localisation transition experiment: Accumulated collision status from all exploration runs when imperfect odometry and the Gmapping SLAM are used in the robot system.

at least 95% of the environments within specified time-frame. DWAD shows the worst performance with only 1.1% completion probability. All evolved DWA controllers exhibit higher performance in terms of completion probability than DWAD. In an ascending order, CONST_EVO reports 5% completion probability, FACRAN_EVO has 7% completion probability and FACDET_EVO indicates 9% completion probability. Comparing with the map transition results, all DWA controllers show a downward trend in completion probability with the decrement of 22% for DWAD, 42% for CONST_EVO, 37% for FACRAN_EVO and 45% for FACDET_EVO. This reduction of the completion performance is explained further in the exploration coverage section.

Exploration coverage

Figures 7.30 to 7.32 show boxplots of explored area for all exploration runs under the localisation transition. For the first environment, DWAD reports the worst performance in that the median of explored area is at $244m^2$ with a relatively tall boxplot. Both CONST_EVO and FACDET_EVO have the median of explored area above $300m^2$ with the exact value at $308m^2$ and $326m^2$, respectively. Meanwhile, FACRAN_EVO gains the median of explored area at

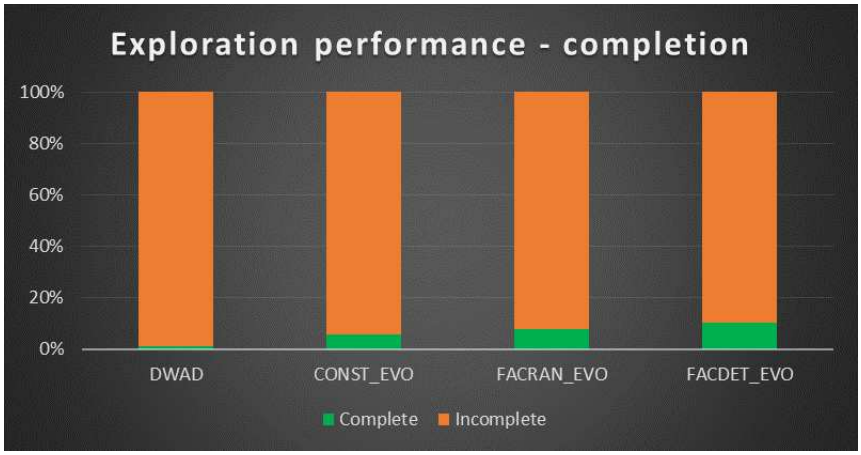


Figure 7.29: Localisation transition experiment: Accumulated completion status from all exploration runs when imperfect odometry and the Gmapping SLAM are used in the robot system.

$267m^2$. All evolved controllers have comparatively short boxplots indicating better consistency than DWAD.

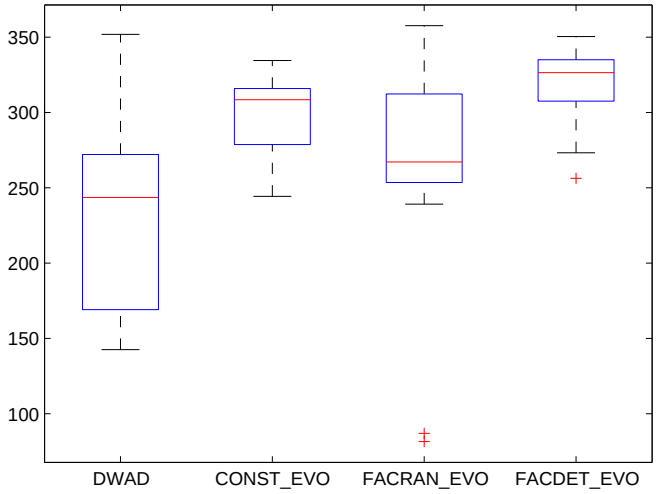


Figure 7.30: Localisation transition experiment: Boxplot of explored area (m^2) at the end of exploration runs on a ground-truth environment of figure 7.16(a) for all DWA policies with SLAM integration.

For the second environment, the medians of explored area of all evolved controllers are better than DWAD in that they are above half of the total area. CONST_EVO reports the median at $172m^2$. Meanwhile, FACRAN_EVO and

FACDET_EVO have the median at $182m^2$ and $181m^2$, respectively. DWAD shows the lowest median at $111m^2$. In brief, FACDET_EVO has the best consistency with the shortest boxplot with 50% of the runs producing an explored area between $188m^2$ (75th percentile) and $172m^2$ (25th percentile). In contrast, DWAD has the worst consistency with the highest boxplot with 50% of the runs produces the explored area between $132m^2$ (75th percentile) and $51m^2$ (25th percentile).

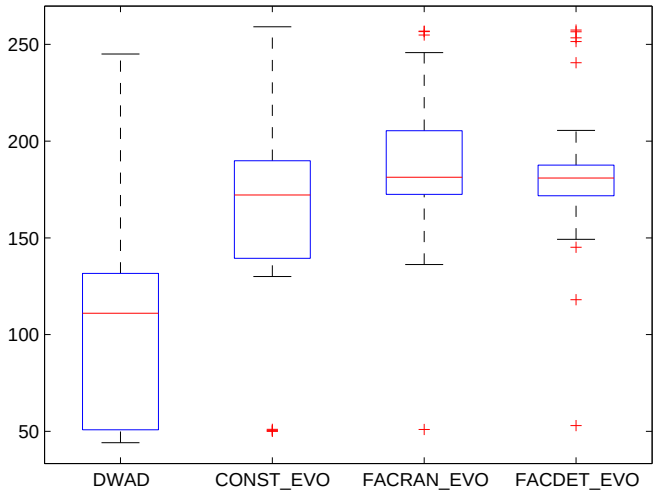


Figure 7.31: Localisation transition experiment: Boxplot of explored area (m^2) at the end of exploration runs on a ground-truth environment of figure 7.16(b) for all DWA policies with SLAM integration.

Figure 7.32 presents the boxplot of the explored area by all DWA policies running on the ground-truth environment of figure 7.16(c). It can be seen that DWAD remains as the worst performed controller. The median explored area of DWAD stood at only $90m^2$. The box plot of DWAD is comparatively tall with 50% of the runs produces the explored area between $155m^2$ (75th percentile) and $65m^2$ (25th percentile). FACDET_EVO reports the highest median explored area at $197m^2$. The box plot of FACDET_EVO is comparatively short compared to others with the distribution of the explored area of 50% runs varies in the range of only $36m^2$ (the upper quartile at $210m^2$ and the lower quartile at $174m^2$). This shows that FACDET_EVO is the most stable controller when exploring this environment with higher probability to end up with higher explored area. The performance of CONST_EVO and

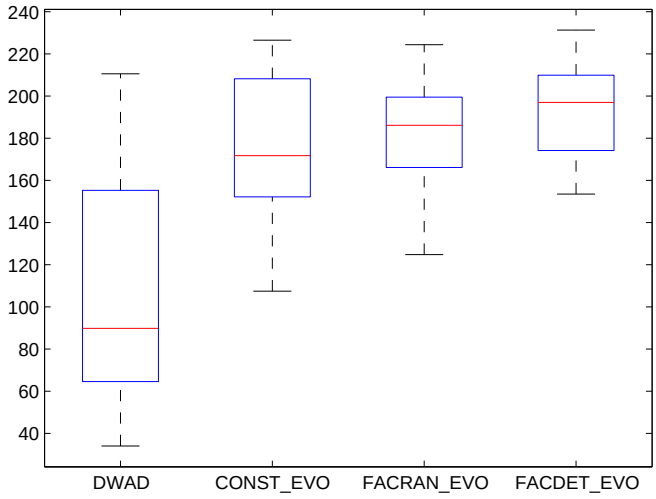


Figure 7.32: Localisation transition experiment: Boxplot of explored area (m^2) at the end of exploration runs on a ground-truth environment of figure 7.16(c) for all DWA policies with SLAM integration.

FACRAN_EVO policies are also promising with the median explored area at $171m^2$ and $186m^2$, respectively.

Even though the completion rate for the robot to explore more than 95% of the environment within the given time-frame has decreased from the results of the map transition experiment, in terms of area explored, the evolved controllers are shown to be less badly affected compared to DWAD. This is supported by the changes to median exploration coverage from all three maps. DWAD reports the worst change to median exploration coverage with the decrements for about 23.9%. FACRAN_EVO has a change of about 17.5%. Meanwhile, FACDET_EVO and CONST_*EVO* have a change of median exploration coverage at respectively 14.5% and 13.6% only. This suggests that evolved controllers are more prone than DWAD to be affected by noise coming from inaccurate localisation.

Localisation transition overall performance

All evolved DWA policies are better than the default DWA policy when performing under the condition of pose uncertainty and the SLAM implementation. We also found that our evolved DWA policies are better at sustaining exploration performance than DWAD when imperfect localisation is introduced to the system. To further clarify the significant difference of the exploration coverage performance, we performed T-test analysis between every two policy as shown in table 7.6. From the table, the results suggest that the evolution on the DWA policy really does have an effect on the exploration performance in that the p-values between DWAD and all evolved policies are below the significant level, $p < 0.05$.

Table 7.6: Localisation transition experiment: T-test results to analyse the variation of the exploration coverage performance arises among all DWA policies.

Policy 1	**Policy 2**	***p*-value**		
		Map A	**Map B**	**Map C**
DWAD	CONST_EVO	**1.8763E-05**	**4.2655E-04**	**2.5720E-07**
DWAD	FACRAN_EVO	**0.0333**	**5.8083E-07**	**3.7606E-09**
DWAD	FACDET_EVO	**2.1213E-08**	**1.0506E-06**	**3.0015E-11**
CONST_EVO	FACRAN_EVO	0.0532	0.1846	0.3988
CONST_EVO	FACDET_EVO	**5.8076E-04**	0.2381	**0.0164**
FACRAN_EVO	FACDET_EVO	**2.9526E-04**	0.8554	**0.0461**

Condition 3: Platform Transition

The final experiment in the application phase is to test the DWA policy functions on the real-robot platform. To obtain exploration performance for all policies, the following experimental procedures were conducted were conducted:

1. The robot is required to explore the environment shown in figure 7.33. It is assumed that the robot has no map and other information about the environment at the beginning of a run.

2. In this experiment, the objective is to acquire the time taken for the robot to explore the environment. To achieve this, we manually determine seven fixed frontier locations that need to be reached by the robot in sequence to explore the whole map. We set the frontier manually so that we can validly compare the time taken for each DWA policy to navigate to the same frontier locations. The frontier locations and the start pose of the robot are shown in figure 7.34. We also observe for any collision occurrence during exploration. In this case, the robot is considered to be colliding when the bumper sensor touches any object.

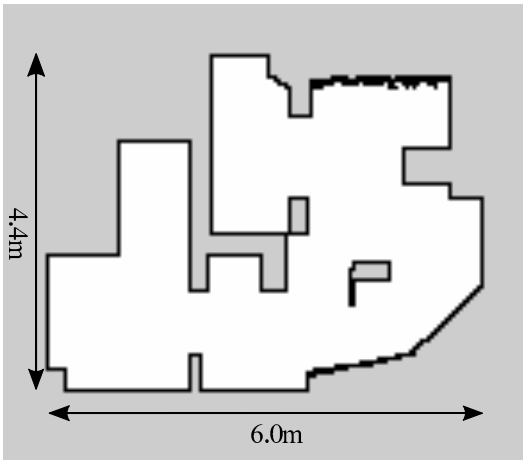
3. For each run, the time taken for the robot to navigate through the first frontier until the last frontier is acquired. The exploration task is stopped once a collision has occured.

4. For each controller, 10 runs are conducted to get the statistical data.

By conducting the above procedures, we are able to record the performance of all DWA policies in terms of exploration time and the ability of the robot to complete exploration without collision. First, we discuss the completion and collision status. Figure 7.35 shows the accumulated collision status of all DWA policies when performing exploration runs. It is understood that in this experiment, all DWA policies are able to drive the robot towards the current frontier if no collision occurs. Thus, if the robot does not collide, an exploration run is considered complete. Looking back to the graph of figure 7.35, it can be seen that for DWAD 80% or 8 out of 10 runs ended with collision. There are only two runs with complete exploration in which the time taken to complete exploration for both runs is, respectively, 77 seconds and 100 seconds. As will be shown shortly, both of these recorded times are long than the average of evolved DWA policies - we shall discuss these timings shortly. On the other hand, all evolved DWA policies are able to complete exploration in most runs. CONST_EVO and FACRAN_EVO report respectively two and one runs with collision, whilst FACDET_EVO records no collisions at all runs.

In terms of exploration time, we only compare the results of complete exploration runs for evolved DWA policies since DWAD was not able to

(a)



(b)

Figure 7.33: Platform transition experiment: A 6.0m x 4.4m testing environment to be explored by the robot. (a) Snapshots of the environment at various angles. (b) The ground-truth map of the environment.

complete most runs. Table 7.7 shows the average exploration time and its standard deviation for all evolved DWA policies. From the table, we

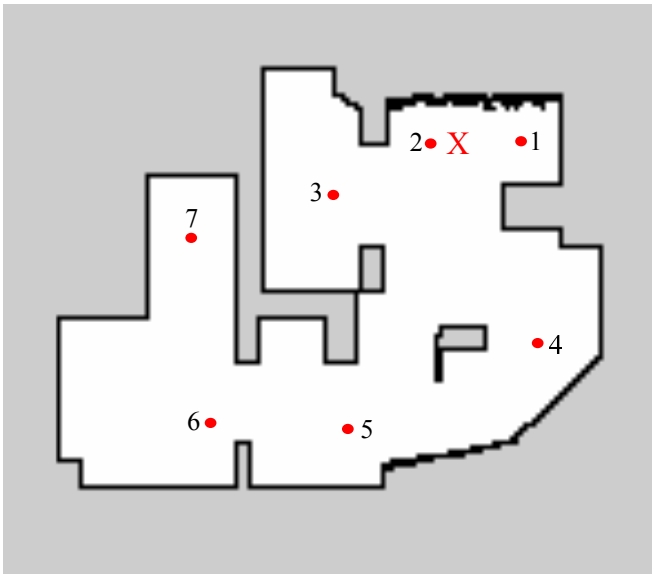


Figure 7.34: Platform transition experiment: Seven frontier points plotted on the ground-truth map of the testing environment. The frontier points are marked as red dots, while the number on each frontier shows the navigational sequence. The start pose of the robot is marked as **X**.

can see that the average exploration time for all evolved DWA policies are similar at about 75 seconds. This confirms that the exploration performance achieved in the learning phase and exhibited in the real platform is consistent. CONST_EVO has the minimum standard deviation of exploration time with 3.94 seconds, meanwhile FACRAN_EVO has the maximum value with 8.78 seconds.

Table 7.7: Platform transition experiment: Average exploration time and its standard deviation for all evolved DWA policies.

Policy	**Exploration time (sec)**	
	Mean	Standard deviation
CONST_EVO	75.88	3.94
FACRAN_EVO	75.11	8.78
FACDET_EVO	76.70	5.68

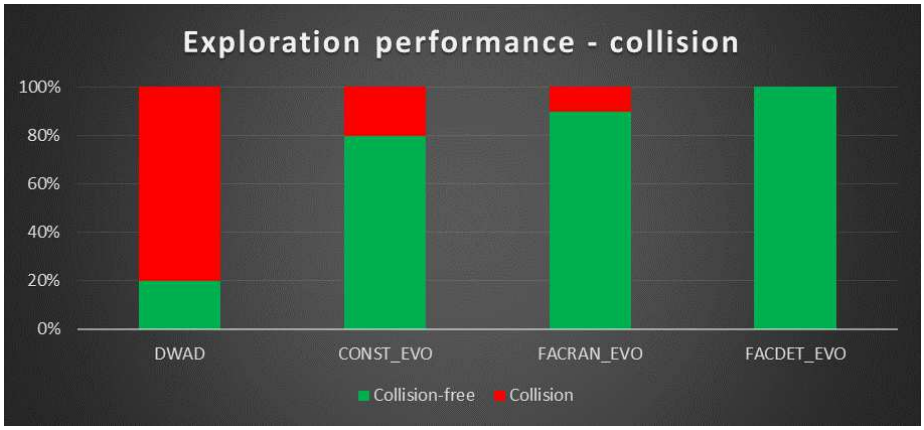


Figure 7.35: Platform transition experiment: Accumulated collision status from all exploration runs when the DWA policies are used in the real-robot system.

Platform transition overall performance

Overall, the transition of the resultant DWA policies from the simulated environment to the real-world environment was successful. The behavior of the DWA policies shown in the real platform is consistent as in the simulated platform. In brief, we can conclude that all evolved DWA policies implemented on the real-robot platform (integrated with Gmapping SLAM) outperform DWAD while exploring the sampled environment of figure 7.33.

7.9 Conclusion

In summary, the objectives of this chapter to evolve the DWA policy function by allowing the automatic selection of input factors for π and, to compare the exploration performance between the default π function and all evolved π functions are achieved. We have shown that the GE's grammar specification structure is flexible in that a production rule template for the selection of input factors can be added into an existing BNF grammar to simultaneously evolve input factors, arithmetic operations and numerical constants in the policy of a scoring function. Using such grammars can facilitate the roboticists

in designing a motion controller without depending too much on a-priori knowledge of the system. Finally, we have shown that the resultant controller suggested by the evolutionary process can be used productively in both in simulated environments and real-platform environments.

8 Conclusions

This thesis has addressed the effectiveness of applying Grammatical Evolution (GE) to the task of evolving motion control for autonomous robotic exploration. In this work we frame motion control design as a search space with a large number of possible solutions. With this perspective, an efficient way to traverse in the search space in order to find good motion control settings given a specific robotic exploration task must be designed. We have proposed a number of BNF grammars for GE to evolve different parts of the policy of motion control. Our experimental results have shown that the GE search process configured with our grammars is able to converge to a good solution in these various configurations. The solutions provided by our GE design in all case studies outperform corresponding conventional handwritten solutions in terms of robotic exploration performance.

In this last chapter, we summarise our overall findings in section 8.1. This chapter closes with future work and possible improvements related to this work in section 8.2.

8.1 Summary of Findings

In this thesis, the main research goal as stated previously in chapter 1 is to develop an automatic approach to design highly effective motion control for an autonomous robotic exploration task. We achieved this goal by developing an automatic approach using GE, particularly in the development of various BNF grammars that can cater various exploration specifications. The limitations

of conventional hand-derived approach (first principle with refinement) for handling this design problem in a complex working environment means that there is scope for exploring search-based approaches to design. Some common challenges in designing a good controller are: i) sensor and computational resource limitations which mean we have to deal with incomplete and abstract representations of the environment, ii) sensory data is provisional and noisy which requires filtering mechanisms to get accurate observations of the robot's environment, ii) motion control design is not just dependent on sensory data, but also on other robot's modules that run simultaneously such as its SLAM algorithm and goal-assignment algorithm, iv) changes in exploration requirements and robot's configurations require adaptation of the controller to these changes. These challenges have led the roboticists to carefully design motion control that can take into account a number of control variables – representing the current robot state – into the robot's decision-making model. However, crafting such models perfectly is not possible when the structure between state signals and their relationship to desired responses for navigational goals are unclear. In this context, our GE-based design approach can be used to assist the roboticists to design motion control in at least three ways: i) finding the relationship between state signals and discovering useful trade-offs via the evolved decision-making model, ii) defining the structure of well-adapted mathematical expressions representing the decision-making model of motion control, and iii) selecting appropriate state signals and ignoring inappropriate or redundant state signals in order for the decision-making model to correctly observe the robot's environment.

In chapter 2 we explored previous approaches to the problem of controller design. Many works in the literature have framed this process as an optimisation problem. One line of research uses machine learning approaches such as Neural Network (NN) and Reinforcement Learning (RL) [70, 29, 78]. However, these approaches are not as effective at handling partially observable problems. A NN model has an opaque structure and not inspectable. RL is not as efficient as EC in learning because state uncertainty is high since the learning process is done separately for every data pair. In contrast, EC techniques handle this problem better using an aggregated evaluation mechanism. Comparing within the field of EC, global numerical optimisers

such as CMA-ES have shown their superiority for evolving numerical constants of motion control but not for the whole structure of the controller. GE has the advantage that it can frame the problem space using a simple and flexible grammar representation, whilst still preserving access to effective search of complex search spaces which is characteristic of EC.

Based on our survey, we have proposed a novel framework for evolving motion control of an exploration task using GE in chapter 3. Specifically, we formulate the way to encode the motion control problem into the evolutionary search domain using a grammar specification. This specification, represented in the form of a BNF grammar, determines which part of the policy of motion control needs to be evolved. In particular, we have designed a set of generative grammars that can be used to evolve numerical constants, unary functions, arithmetic operations and input factor selection of a policy. In all of our experiments we refine our simulation framework to the problem domain and performance constraints of each experiment. Similarly, we defined evaluative functions suited to the factors of importance to each experiment.

In chapter 4, we applied GE to the discovery of settings for motion control in a coordinated multi-robot exploration setting. In this setting, motion control must be designed to complete exploration faster by allowing multiple robots to work on the task together. In these experiments, we used GE to find an effective motion controller that is able to trade-off between three state signals: distance-to-static-obstacle, distance-to-other-robot and distance-to-target-location. This experiment was restricted to the discovery of appropriate weights (numerical constants) for all state signals. Our finding reveals that the evolved motion controller can perform better than a carefully manually-designed handwritten motion controller even on short evolutionary runs.

We then further extended the application of GE to the numerical constant evolution for another exploration setting in chapter 5. In this chapter, the exploration task is formulated as multi-objective exploration in which requires the robot to maximise exploration coverage and safety performance and at the same time to minimise the robot's power consumption. This task is executed

by single robot. Comparison of motion controls used in this experiment is done between a handwritten approach and two evolutionary approaches: GE and CMA-ES. The evolved motion controls indicate superior performance to the handwritten motion control showing that the GE is able to be used in various exploration settings.

In some cases where a pre-defined structure of motion control is not well defined, the process of finding the structure must be carried out. In chapter 6, we propose a number of GE grammars that can accommodate different levels of a-priori knowledge to support the design of policy structure. We demonstrate the flexibility of GE in catering for this problem by allowing the whole structure to be evolved including numerical constants, unary function of each state signal and arithmetic operators between state signal functions. Using the same exploration setting in the previous chapter, we found that the exploration performance can be increased by finding a new policy structure of motion control. Our experimental results have shown that GE is able to search for good policy structures in a large search space. We also found that there are some patterns in the state signal functions that can produce a good motion controller which can be further studied to understand the intrinsic properties that make up a good design.

Finally in chapter 7 we implement the GE algorithm to a new robot platform that uses the Dynamic Window Approach (DWA) as motion control. In the literature, we found that many works had proposed different sets of input factors to DWA policy. However, there is no guideline for choosing the factors that are appropriate for particular autonomous exploration settings. As such, we develop an automatic mechanism to choose these factors by using GE. On top of our existing GE's grammars, we add new specification to the grammar such that the input factors, arithmetic operation and numerical constants in the policy of motion control can be evolved simultaneously. Our extensive experiments both in simulated and real environments have shown that a policy with appropriate input factors and with well-defined policy structure performs better than a fixed policy structure in terms of exploration performance. This is achieved by all our evolved policies of DWA motion control.

8.2 Future Work

In this section, we highlight future work that can be investigated to further improve the application of a GE approach to autonomous exploration.

Online evolution - Throughout the thesis we have conducted the evolution of policy functions using a simulation approach. This approach is categorised as offline evolution. In contrast, online evolution can be used to evolve motion control for autonomous exploration exploration while that exploration is taking place. With online evolution, the robot's controller can be evolved and improved while the robot undertakes the exploration task. A candidate solution can be evaluated to align with real-environment performance. This is very useful when the kinematic model of the robot is is difficult to derive with accuracy with simulation. In addition, the errors due to platform transition can be reduced. The benefits of online evolution have been reported in the literature for simple robotic tasks [50, 31, 65]. However, the online evolution approach also raises new challenges for autonomous exploration. Since exploration is normally a long-run task, evaluating candidate solutions in a real-time environment requires a massive time-frame for the evolutionary process to converge. Furthermore, it is difficult to evaluate each candidate solution with the same exploration setting (e.g. similar starting point) at the beginning of the evaluation process. Thus, a further research to investigate the best evolutionary configuration for online evolution for autonomous exploration needs to be carried out.

Evolution of high-dimensional state-spaces - In this thesis, we have shown the capability of GE to solve the motion control problem on various sizes of search space. The number of state signals used by the policy of motion control has direct impact on the size of the search space. In our experiments, we found that GE is able to come up with a good solution for the case of three state signals. Even though the number of state signals is not high, we have shown that the search space is large due to the setting in which we allow several parts of a policy to be evolved including numerical constants, the unary function of state signals and arithmetic operations defining the structure of the policy. For

future work, we suspect that GE can handle larger search spaces which can determine the ability of GE to evolve a high-dimensional state signals policy. Since the trend of robot systems is to become more complex over time, the need to cater for such systems is necessary.

Heterogeneous multi-robot system motion control - The growth of multi-robot systems with diverse robot types is an interesting area. For example, a co-operative unmanned ground vehicles and unmanned air vehicles (UGV/UAV) platform for wildfire detection and fighting has been developed by [105]. In this case, each type of robot may have its own motion control. To achieved optimal trajectory coordination, information flow among robots, taking into account constraints posed by each robot type must be fully utilised by the policy of motion control. As such, GE can be used to evolve multiple policies concurrently to find the best controller for each robot. Alternatively, GE can also be used to evolve a hierarchical exploration system where there are multiple policies or decision-making functions available in each exploration component i.e. goal-allocation, SLAM and motion control.

As the capabilities and complexities of multi-robot exploration systems grow so does the challenge of designing good motion control. It makes sense that we should harness computing resources to aid in this design process and make the most of the potential of robotic platforms.

Bibliography

[1] P. Angeline and J. Peter. *Chapter 5, Advances in Genetic Programming 2*, chapter Two Self-Adaptive Crossover Operators for Genetic Programming, pages 89–110. MIT Press, 1996.

[2] A. Q. Arif, D. G. Nedev, and E. Haasdijk. Controlling Evaluation Duration in Online, Onboard Evolutionary Robotics. In *Proc. IEEE Conference on Evolving and Adaptive Intelligent System (EAIS)*, 2013.

[3] T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, 1997.

[4] J. W. Backus. The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. In *Proc. International Conference on Information Processing UNESCO*, pages 125–132, 1959.

[5] 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, 1992.

[6] N. Basilico and F. Amigoni. Defining Effective Exploration Strategies for Search and Rescue Applications with Multi-Criteria Decision Making. In *Proc. IEEE International Conference on Robotics and Automation*, pages 4260–4265, 2011.

[7] R. Bis, H. Peng, and G. Ulsoy. Velocity Occupancy Space : Robot

Navigation and Moving Obstacle Avoidance with Sensor Uncertainty. In *Proc. ASME 2009 Dynamic Systems and Control Conference*, 2009.

[8] J. C. Bongard. Spontaneous Evolution of Structural Modularity in Robot Neural Network Controllers. In *Proc. 13th Annual Conference on Genetic and Evolutionary Computation*, pages 251–258, 2011.

[9] J. Borenstein and Y. Koren. The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

[10] O. Brock and O. Khatib. High-Speed Navigation Using the Global Dynamic Window Approach. In *Proc. IEEE International Conference on Robotics and Automation*, pages 341–346, 1999.

[11] R. Brook. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robot. Autom.*, 2:14–23, 1986.

[12] R. Burbidge, J. H. Walker, and M. S. Wilson. Grammatical Evolution of a Robot Controller. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 357–362, 2009.

[13] R. Burbidge and M. S. Wilson. Vector-Valued Function Estimation by Grammatical Evolution for Autonomous Robot Control. *Information Sciences*, 258:182–199, 2014.

[14] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider. Coordinated Multi-Robot Exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005.

[15] D. Calisi, A. Farinelli, L. Iocchi, and D. Nardi. Multi-Objective Exploration and Search for Autonomous Rescue Robots. *Journal of Field Robotics*, 24(8/9):763–777, 2007.

[16] J. Carlson and R. R. Murphy. How UGVs Physically Fail in the Field. *IEEE Transactions on Robotics*, 21(3):423–437, 2005.

[17] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under Uncertainty : Discrete Bayesian Models for Mobile-Robot Navigation. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 963 – 972, 1996.

[18] B. Cazzolato, S. Grainger, C. Madden, M. Baulis, A. Cundy, N. Gaskin, P. Hardy, P. Huynh, S. Komandurelaiyavalli, K. Pilch, A. Skeketee, B. Quast, and S. Wong. Magic 2010 Prelimenary Report.

[19] J. S. Cepeda, L. Chaimowicz, R. Soto, J. L. Gordillo, E. Alanis-Reyes, and L. C. Carrillo-Arce. A Behavior-based Strategy for Single and Multi-Robot Autonomous Exploration. *Sensors*, 12(9):12772–12797, 2012.

[20] H. Choset and J. Burdick. Sensor-Based Exploration : The Hierarchical Generalized Voronoi. *The International Journal of Robotics Research*, 19(2):96–125, 2000.

[21] O. Chuy, E.G. Collins, and C. Ordonez. Power Modeling of a Skid Steered Wheeled Robotic Ground Vehicle. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 4118–4123, 2009.

[22] M. Clifton and G. Fang. Genetic Programming in Robot Exploration. In *Proc. IEEE International Conference on Mechatronics and Automation*, pages 451–456, 2007.

[23] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle of Life.* John Murray, Albemarle Street London, 1859.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and

Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, 2002.

[25] I. Dempsey, M. O'Neill, and A. Brabazon. Constant Creation in Grammatical Evolution. *Int. Journal on Innovative Computing and Applications*, 1(1):23–38, 2007.

[26] E.W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[27] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csobra. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE Trans. on Robotics and Automation*, 17(3):229–241, 2001.

[28] L. Doitsidis, N. C. Tsourveloudis, and S. Piperidis. Evolution of Fuzzy Controllers for Robotic Vehicles: The Role of Fitness Function Selection. *Journal of Intelligent and Robotic Systems*, 56(4):469–484, 2009.

[29] B. Doroodgar and G. Nejat. A Hierarchical Reinforcement Learning Based Control Architecture for Semi-Autonomous Rescue Robots in Cluttered Environments. In *Proc. 6th IEEE Conference on Automation Science and Engineering*, pages 948 – 953, 2010.

[30] G. Dozier, A. Homaifar, S. Bryson, and L. Moore. Artificial Potential Field Based Robot Navigation, Dynamic Constrained Optimization and Simple Genetic Hill-Climbing. In *Proc. IEEE International Conference on Evolutionary Computation Proceedings Computational Intelligence*, pages 189–194, 1998.

[31] S. C. Duong, H. Kinjo, E. Uezato, and T. Yamamoto. Online Adaptive Control via Evolutionary Algorithm. *IEEJ Transactions on Electrical and Electronic Engineering*, 6(S1):S57–S64, 2011.

[32] J. W. Durham and F. Bullo. Smooth Nearness-Diagram Navigation

Global Planner Planner Dynamics. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[33] D. Eberly. Intersection of Convex Objects: The Method of Separating Axes. geometrictools.com/Documentation/MethodOfSeparatingAxes.pdf, Jun 1998.

[34] Leuze Electronic. Leuze Electronics. http://www.leuze.com/, April 2015.

[35] A. Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 22(6):46–57, 1989.

[36] H. J. S. Feder, J. J. Leonard, and C. M. Smith. Adaptive Mobile Robot Navigation and Mapping. *The International Journal of Robotics Research*, 18(7):650–668, 1999.

[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.*, 57(4):411–419, 2009.

[38] A. Finn, A. Jacoff, M. D. Rose, B. Kania, J. Overholt, U. Silva, and J. Bornstein. Evaluating Autonomous Ground-Robots. *Journal of Field Robotics*, 29(5):689–706, 2012.

[39] D. Flores and J. Cervantes. Rank Based Evolution of Real Parameters on Noisy Fitness Functions: Evolving a Robot Neurocontroller. In *Proc.. 10th Mexican International Conference on Artificial Intelligence*, pages 72–76, 2011.

[40] Open Source Robotics Foundation. Robot Operating System (ROS). ros.org, October 2014.

[41] Open Source Robotics Foundation. Turtlebot 2. http://www.turtlebot.com, May 2015.

[42] D. Fox, W. Burgard, and S. Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[43] S. Garrido, L. Moreno, and D. Blanco. Exploration and Mapping Using the VFM Motion Planner. *IEEE Transactions on Instrumentation and Measurement*, 58(8):2880–2892, 2009.

[44] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[45] D. E. Goldberg and K. Deb. Analysis of Selection A Comparative Used in Genetic Algorithms Schemes. In *Foundations of Genetic Algorithms*, 1991.

[46] M. Grabisch and C. Labreuche. A Decade of Application of the Choquet and Sugeno Integrals in Multi-Criteria Decision Aid. *4OR A Quarterly Journal of Operations Research*, 6(1):1–44, 2008.

[47] G. Grisetti, C. Stachniss, and W. Burgard. Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[48] G. Grisetti, C. Stachniss, and W. Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[49] D. Grune and C. J. H. Jacobs. *Parsing Techniques: A Practical Guide*. US:Springer, 1999.

[50] E. Haasdijk, A. Q. Arif, and A. E. Eiben. Racing to Improve On-line, On-board Evolutionary Robotics. In *Proc. 13th Conference on Genetic and Evolutionary Computation (GECCO)*, 2011.

[51] H. Hagras. A Hierarchical Type-2 Fuzzy Logic Control Architecture for Autonomous Mobile Robots. *IEEE Trans. on Fuzzy System*, 12(4):524–539, 2004.

[52] N. Hansen. *The CMA Evolution Strategy : A Comparing Review*, chapter Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms., page 17691776. 2006.

[53] P.E. Hart, N.J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[54] I. Harvey. Artificial Evolution and Real Robots. *Artificial Life and Robotics*, 1(1):35–38, 1997.

[55] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[56] C. H. Hsu and C. F. Juang. Evolutionary Robot Wall-Following Control Using Type-2 Fuzzy Controller With Species-DE-Activated Continuous ACO. *IEEE Transactions on Fuzzy Systems*, 21(1):100–112, 2013.

[57] J. Huh, W. S. Chung, S. Y. Nam, and W. K. Chung. Mobile Robot Exploration in Indoor Environment Using Topological Structure with Invisible Barcodes. *ETRI Journal*, 29(2):189–200, 2007.

[58] T. Hussain, D. Montana, and G. Vidaver. Evolution-Based Deliberative Planning for Cooperating Unmanned Ground Vehicles in a Dynamic Environment. In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pages 1185–1196, 2004.

[59] M. F. Ibrahim and B. Alexander. Evolving a Path Planner for a Multi-Robot Exploration System Using Grammatical Evolution. In *Proc. 7th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 590–595, 2011.

[60] M. F. Ibrahim and B. Alexander. Designing a Navigational Control System of an Autonomous Robot for Multi-Requirements Planetary Navigation using Evolutionary Algorithms Approaches. In *Proc. 12th Australian Space Science Conference (ASSC)*, pages 223–234, 2012.

[61] M. F. Ibrahim and B. J. Alexander. Evolving Decision-Making Functions in an Autonomous Robotic Exploration Strategy Using Grammatical Evolution. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4340–4346, 2013.

[62] C. F. Juang and Y. C. Chang. Evolutionary-Group-Based Particle-Swarm- Optimized Fuzzy Controller With Application to Mobile-Robot Navigation in Unknown Environments. *IEEE Transactions on Fuzzy Systems*, 19(2):379–392, 2011.

[63] M. Julia, A. Gil, and O. Reinoso. A Comparison of Path Planning Strategies for Autonomous Exploration and Mapping of Unknown Environments. *Autonomous Robots*, 33(4):427–444, 2012.

[64] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *Proc. 6th European Conference on Genetic Programming*, pages 70–82, 2003.

[65] D. Keymeulen, M. Iwata, Y. Kuniyoshi, and T. Higuchi. Online Evolution for a Self-adapting Robotic Navigation System using Evolvable Hardware. *Artificial life*, 4(4):359–93, 1998.

[66] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Proc. IEEE International Conference on Robotics and Automation*, pages 500–505, 1985.

[67] A. Kisdi and A. R. L. Tatnall. Future Robotic Exploration Using Honeybee Search Strategy: Example Search for Caves on Mars. *Acta Astronautica*, 68:1790–1799, 2011.

[68] D. Kiss and G. Tevesz. Advanced Dynamic Window based Navigation Approach using Model Predictive Control. In *Proc. International Conference on Methods & Models in Automation & Robotics*, pages 148–153, 2012.

[69] M. Knudson and K. Tumer. Adaptive Navigation for Autonomous Robots. *Robotics and Autonomous Systems*, 59(6):410–420, 2011.

[70] T. Kollar and N. Roy. Using Reinforcement Learning to Improve Exploration Trajectories for Error Minimization. In *IEEE International Conference on Robotics and Automation*, pages 3338–3343, 2006.

[71] T. Kollar and N. Roy. Trajectory Optimization using Reinforcement Learning for Map Exploration. *The International Journal of Robotics Research*, 27(2):175–196, 2008.

[72] J. R. Koza and J. P. Rice. Automatic Programming of Robots using Genetic Programming. In *Proc. 10th National Conference on Artificial Intelligence*, pages 194–201, 1992.

[73] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[74] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1):83–97, 1955.

[75] B. Kuipers and T. Levitt. Navigation and Mapping in Large-Scale Space. *Artificial Intelligence Magazine*, 9(2):25–43, 1988.

[76] J.P. Laumond. *Robot Motion Planning and Control*. Springer Berlin Heidelberg, 1998.

[77] S. M. LaValle. *Planning Algorithms*, chapter Decision-Theoretic Planning, pages 433–710. Cambridge University Press, 2006.

[78] Y. Liu, G. Nejat, and B. Doroodgar. Learning Based Semi-Autonomous Control for Robots in Urban Search and Rescue. In *Proc. IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6, 2012.

[79] Hokuyo Automatic Co. Ltd. Hokuyo Laser-Range Finder URG-04LX-UG01. https://www.hokuyo-aut.jp, January 2015.

[80] A. A. Makarenko, S. B. Williams, F. Bourgault, and H. F. Durrant-whyte. An Experiment in Integrated Exploration. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 534 – 539, 2002.

[81] M. J. Mataric. *Encyclopedia of Cognitive Science*, chapter Situated Robotics. Nature Publishers Group, 2002.

[82] M. J. Mataric, G. S. Sukhatme, and E. H. Ostergaard. Multi-Robot Task Allocation in Uncertain Environments. *Autonomous Robots*, 14:255–263, 2003.

[83] L. Matignon, L. Jeanpierre, and A. Mouaddib. Distributed Value Functions for Multi-Robot Exploration. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 1544 – 1550, 2012.

[84] Microsoft. Microsoft Kinect 360. http://www.xbox.com, October 2014.

[85] LORD MicroStrain. Microstrain 3DM-GX3-25. http://www.microstrain.com/inertial/3DM-GX3-25, April 2014.

[86] M. Milford and G. Wyeth. Hybrid Robot Control and SLAM for Persistent Navigation and Mapping. *Robotics and Autonomous Systems*, 58(9):1096–1104, 2010.

[87] B. L. Miller and D. E. Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, 9:193–212, 1995.

[88] J.M. Mingo and R. Aler. The Role Of The Lamarck Hypothesis In The Grammatical Evolution Guided By Reinforcement. *IEEE Latin America Transactions*, 6(6):500–504, 2008.

[89] J. Minguez and L. Montano. Nearness Diagram Navigation (ND): A New Real Time Collision Avoidance Approach. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2094–2100, 2000.

[90] J. Minguez and L. Montano. Nearness Diagram (ND) Navigation : Collision Avoidance in Troublesome Scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, 2004.

[91] H. Mintzberg, D. Raisinghani, and A. Theoret. The Structure of Unstructured Decision Processes. *Administrative Sciences Quarterly*, 21:246–275, 1976.

[92] MIT. GAlib: A C++ Library of Genetic Algorithm. http://lancet.mit.edu/ga, September 2014.

[93] H. Moravec. Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine*, 9(2):61–74, 1988.

[94] H. Moravec and A. Elfes. High Resolution Maps from Wide Angle Sonar. In *Proc. IEEE International Conference on Robotics and Automation*, pages 116–121, 1985.

[95] M. Mujahad, D. Fischer, B. Mertsching, and H. Jaddu. Closest Gap Based (CG) Reactive Obstacle Avoidance Navigation for Highly Cluttered Environments. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1805–1812, 2010.

[96] G. Muscato, F. Bonaccorso, L. Cantelli, D. Longo, and C. D. Melita. Volcanic Environments Robots for Exploration and Measurement. *IEEE Robotics & Automation Magazine*, 19(1):40–49, 2012.

[97] M. Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems*. Essex: Pearson Education Limited, 2005.

[98] M. Nicolau and D. Slattery. *libGE - Grammatical Evolution Library*, 2006.

[99] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence and Technology of Self-Organizing Machines*, chapter The Role of Self-Organization for the Synthesis and the Understanding of Behavioral Systems, pages 1–18. The MIT Press, 2001.

[100] P. Nordin and W. Banzhaf. Real Time Control of a Khepera Robot using Genetic Programming. *Control and Cybernatics*, 26(3):533–561, 1997.

[101] Novatel. OEM-V2 Compact Dual Frequency GPS Datasheet. http://www.novatel.com/assets/Documents/Papers/OEMV2.pdf, April 2014.

[102] University of Limerick. libGE: A C++ Library of Grammatical Evolution. http://bds.ul.ie/libGE, September 2014.

[103] M. O'Neill and C. Ryan. Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[104] A. Pal, R. Tiwari, and A. Shukla. *Multi Robot Exploration Using a Modified A* Algorithm*, chapter Intelligent Information and Database Systems, pages 506–516. Springer Berlin Heidelberg, 2011.

[105] C. Phan and H. H.T. Liu. A Cooperative UAV/UGV Platform for Wildfire Detection and Fighting. In *Proc. 7th International Conference on System Simulation and Scientific Computing*, pages 494–498, 2008.

[106] PlayerStage. Player/Stage Robot Simulator. http://playerstage.sourceforge.net, October 2014.

[107] A. Ram, R. Arkin, G. Boone, and M. Pearce. Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation. *Adaptive Behavior*, 2(3):277–304, 1994.

[108] C. W. Reynolds. Evolution of Corridor Following Behavior in a Noisy World. In *Proc. 3rd International Conference on Simulation of Adaptive Behavior*, 1994.

[109] Yujin Robot. Kobuki. http://kobuki.yujinrobot.com/home-en, May 2015.

[110] J. K. Rosenblatt. DAMN: A Distributed Architecture for Mobile Navigation. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):339–360, 1997.

[111] P. E. Rybski, S. A. Stoeter, M. D. Erickson, M. Gini, D. F. Hougen, and N. Papanikolopoulos. A Team of Robotic Agents for Surveillance. In *Proc. 4th International Conference on Autonomous Agents*, pages 9–16, 2000.

[112] P. Saranrittichai, N. Niparnan, and A. Sudsang. Robust Local Obstacle Avoidance for Mobile Robot based on Dynamic Window Approach. In *Proc. 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 2013.

[113] M. Schoenauer and Z Michalewicz. Evolutionary Computation. *Control and Cybernetics*, 26:307–338, 1997.

[114] A. C. Schultz. Using a Genetic Algorithm to Learn Strategies for Collision Avoidance and Local Navigation. In *Proc. 7th International*

Symposium on Unmanned Untethered Submersible Technology, pages 213–225, 1991.

[115] A. Solanas and M. A. Garcia. Coordinated Multi-Robot Exploration Through Unsupervised Clustering of Unknown Space. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[116] C.C. Sotzing and C.B. Congdon. GENCEM: A Genetic Algorithms Approach to Coordinated Exploration and Mapping with Multiple Autonomous Robots. In *Proc. IEEE Congress on Evolutionary Computation*, pages 2317–2324, 2005.

[117] G. Spencer. *Chapter 15, Advances in Genetic Programming*, chapter Automatic generation of programs for crawling and walking, pages 335–353. MIT Press, 1994.

[118] C. Stachniss. *Robotic Mapping and Exploration*, chapter Coordinated Multi-Robot Exploration, pages 43–71. Springer-Verlag Berlin Heidelberg, 2009.

[119] C. Stachniss. *Robotic Mapping and Exploration*, chapter Introduction, pages 3–6. Springer-Verlag Berlin Heidelberg, 2009.

[120] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1st edition, 1998.

[121] S. Thrun. *Exploring Artificial Intelligence in the New Millennium*, chapter Robotic Mapping: A Survey, pages 1–29. 2002.

[122] I. Ulrich and J. Borenstein. VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1572 – 1577, 1998.

[123] I. Ulrich and J. Borenstein. VFH *: Local Obstacle Avoidance with

Look-Ahead Verification. In *Proc. IEEE International Conference on Robotics & Automation*, pages 2505–2511, 2000.

[124] P. Vadakkepat, X. Peng, B. K. Quek, and T. H. Lee. Evolution of Fuzzy Behaviors for Multi-Robotic System. *Robotics and Autonomous Systems*, 55(2):146–161, 2007.

[125] R. Vaughan. Massively Multi-Robot Simulation in Stage. *Swarm Intelligence*, vol. 2:189–208, 2008.

[126] S.M. Veres, L. Molnar, N.K. Lincoln, and C.P. Morice. Autonomous Vehicle Control Systems a Review of Decision Making. *Journal of Systems and Control Engineering*, pages 155–195, 2011.

[127] K. Watanabe, K. Izumi, J. Maki, and K. Fujimoto. A Fuzzy Behavior-Based Control for Mobile Robots Using Adaptive Fusion Units. *Journal of Intelligent and Robotic Systems*, 42(1):27–49, 2005.

[128] S. Whiteson. *Reinforcement Learning: State of the Art*, chapter Evolutionary Computation for Reinforcement Learning, page 325358. Springer, 2012.

[129] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski. Adaptive Evolutionary Planner/Navigator for Mobile Robots. *IEEE Transactions on Evolutionary Computation*, 1(1):1–23, 1997.

[130] B. Yamauchi. A Frontier-Based Approach for Autonomous Exploration. In *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 146–151, 1997.

[131] B. Yamauchi. Frontier-Based Exploration using Multiple Robots. In *Proc. 2nd international conference on Autonomous agents (AGENTS98)*, pages 47–53, 1998.

[132] B. Yamauchi. Decentralized Coordination for Multirobot Exploration. *Robotics and Autonomous Systems*, 29:111–118, 1999.

[133] T. Yasuda and K. Ohkura. Multi-Robot Cooperation Based on Continuous Reinforcement Learning with Two State Space Representations. In *Proc. IEEE Conference on Systems, Man and Cybernetics*, pages 4470–4475, 2013.

[134] Chang Y.C. and Yamamoto Y. On-line Path Planning Strategy Integrated with Collision and Dead-Lock Avoidance Schemes for Wheeled Mobile Robot in Indoor Environments. *Industrial Robot: An International Journal*, 35(5):421–434, 2008.

[135] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In *Proc. Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN2001)*, pages 1–21, 2001.

[136] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-Robot Exploration Controlled by a Market Economy. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2002.