Hybrid Approaches for Mobile Robot Navigation

by

Yang Wang

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the award of

Doctor of Philosophy

of

Loughborough University

October 2007

ABSTRACT

The work described in this thesis contributes to the efficient solution of mobile robot navigation problems. A series of new evolutionary approaches is presented.

Two novel evolutionary planners have been developed that reduce the computational overhead in generating plans of mobile robot movements. In comparison with the best-performing evolutionary scheme reported in the literature, the first of the planners significantly reduces the plan calculation time in static environments. The second planner was able to generate avoidance strategies in response to unexpected events arising from the presence of moving obstacles.

To overcome limitations in responsiveness and the unrealistic assumptions regarding *a priori* knowledge that are inherent in planner-based navigation systems, subsequent work concentrated on hybrid approaches. These included a reactive component to identify rapidly and autonomously environmental features that were represented by a small number of critical waypoints. Not only is memory usage dramatically reduced by such a simplified representation, but also the calculation time to determine new plans is significantly reduced. Further significant enhancements of this work were firstly, dynamic avoidance to limit the likelihood of potential collisions with moving obstacles and secondly, exploration to identify statistically the dynamic characteristics of the environment. Finally, by retaining more extensive environmental knowledge gained during previous navigation activities, the capability of the hybrid navigation system was enhanced to allow planning to be performed for any start point and goal point.

ACKNOWLEDGEMENTS

First and foremost, I am greatly indebted to Dr David Mulvaney, my research supervisor, for his excellent supervision, invaluable advice, constructive suggestions and helpful discussion throughout this research project.

Secondly, I owe a great deal of thanks to Dr Ian Sillitoe of the Scottish Association for Marine Science, who acted as joint supervisor, for his expert advice and motivation, an infinite supply of patience, support and immense help throughout this research project.

Also, I would like to express my gratitude to my colleagues for their kind help in my project and life. Furthermore, many thanks are due to the Department of Electronic and Electrical Engineering at Loughborough University for constructing a friendly research environment.

Finally, special thanks to my wife and my family.

PUBLICATIONS

1. Y. Wang, D. J. Mulvaney, and I. P. W. Sillitoe, "Robot navigation by genetic algorithms," *Electronics and Systems Division Conference*, Department of Electronic and Electrical Engineering, Loughborough University, UK, January 2005, pp. 9–10.

2. D. J. Mulvaney, I. P. W. Sillitoe, E. Swere, Y. Wang, and Z. H. Zhu, "Real-time machine learning in embedded software and hardware platforms," *Workshop on Automatic Learning and Real-Time*, Siegen, Germany, September 2005, pp. 65–78.

3. D. J. Mulvaney, Y. Wang, and I. P. W. Sillitoe, "Waypoint-based mobile robot navigation," *6th IEEE World Congress on Intelligent Control and Automation*, Dalian, China, June 2006, pp. 9063–9067.

4. Y. Wang, D. J. Mulvaney, and I. P. W. Sillitoe, "Genetic-based mobile robot path planning using vertex heuristics," *2006 IEEE Conf. Cybernetics and Intelligent Systems*, Bangkok, Thailand, June 2006, pp. 463–468.

5. D. J. Mulvaney, I. P. W. Sillitoe, E. Swere, Y. Wang, and Z. H. Zhu, "Real-time machine learning in embedded software and hardware platforms," *International Journal of Intelligent Systems Technologies and Applications*, vol. 2, no. 2/3, pp. 187–204, 2007.

6. Y. Wang, I. P. W. Sillitoe, and D. J. Mulvaney, "Mobile robot path planning in dynamic environments," *IEEE International Conference on Robotics and Automation*, Rome, Italy, April 2007, pp. 71–76.

7. Y. Wang, D. J. Mulvaney, I. P. W. Sillitoe, and E. Swere, "Robot navigation by waypoints," submitted to *Robotica*.

8. Y. Wang, D. J. Mulvaney, and I. P. W. Sillitoe, "Genetic-based mobile robot planning in dynamic environments," in preparation for submission to the IEEE Transactions on Evolutionary Computation.

9. Y. Wang, D. J. Mulvaney, and I. P. W. Sillitoe, "The use of waypoints for robot navigation in dynamic environments," in preparation for submission to the International Journal of Robotics Research.

10. Y. Wang, D. J. Mulvaney, and I. P. W. Sillitoe, "A generalised waypoint navigation system for mobile robots," in preparation for submission to the IEEE Transactions on Systems, Man and Cybernetics, part B: cybernetics.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

This chapter provides the background and context of the research work reported in this thesis. The chapter begins with a brief history of the mobile robots, followed by an introduction to the problem under investigation, namely robot navigation and the principal approach of genetic algorithms adopted in the research project. The objectives of the research project are presented, the contributions to knowledge are summarised and the thesis structure is outlined.

1.1 Mobile robots

The popular conception of robots is that of machines with a human appearance, behaviours and emotions. This image has been fostered in the media from the first performance of Karel Capek's play, R.U.R (Rossum's Universal Robots), to the modern movie series Star Wars. The practical reality is more mundane. The majority of robots are in use in the manufacturing industry, either repeatedly performing definable tasks or working in environments that are dangerous, perhaps toxic or intemperate. In contrast, relatively few specialist robots have been developed for research purposes, such as for operation in deep seas or in outer space. Recent developments include: micro-robots or nano-robots that can be injected into the human body to assist diagnosis and return detailed pathological data (Guo, Sawamoto and Pan 2005); domestic robots performing household chores such as cleaning or

weeding (Lee 1998) and robot pets (Fujita and Kitano 1998) and football-playing robots (Asada *et al.* 1999) entertaining their human masters.

The definition of a robot can be very general, 'any device which replaces human labour' (Soska 1985), or quite specific, 'a robot is a pre-programmable, multi-functional, manipulator designed to move material, parts, tools, or specialised devices through variable programmed motions for the performance of a variety of tasks' (Jablonski and Posey 1985). Robots themselves have been classified in various ways; for example, robots can be differentiated in terms of the type of control, compatibility level, configuration or moving ability (Critchlow 1985). As this thesis focuses on the ability of a robot to be mobile, the distinction between fixed and mobile robots is the most relevant here. Most industrial robots have their base fixed in physical location and consequently their workspace is constrained to be the maximum extension of their linkages. To overcome this problem, two approaches have been taken, namely flexible manufacturing cells and mobilising the robots. In the former approach, the change to the effective workspace volume that results from modifications to the robot's configuration is limited, at least in comparison with that achievable by mobile robots. Various locomotion mechanisms have been designed to mobilise robots, including wheels, tracks, legs and motor thrusters. These have enabled the development of serpentine robots, climbing robots, underwater robots, free-flying robots, and self-reconfigurable robots. A detailed explanation of locomotion mechanisms is presented in (Bekey 2005; Siegwart and Nourbakhsh 2004), along with discussions of their biological counterparts. Although nature did not evolve any living species with fully-rotating actively-powered joints, wheels are the most common method for locomotion in human-designed systems. Much recent research concerning robot mobility has focused on self-reconfiguring (morphing) robots that can change their mode of locomotion according to either internal intention or the external terrain. For example, such a robot could change from a rolling machine into a legged robot when a well is detected.

1.2 Robot navigation

A mobile robot can be teleoperated, pre-programmed for repetitive tasks or navigate autonomously. The work in this thesis considers only autonomous navigation. Task descriptions for autonomous robots often need to only specify what the operator wants done rather than how it is to be done. Achieving autonomous navigation requires the successful application of many artificial intelligence attributes, including sensing, actuation, planning and problem solving algorithms, as well as the specification of a suitable embedded platform, including real-time software and hardware architectures.

In any given environment, a mobile robot is expected to move between two or more specified locations in order to accomplish an assigned task. The following four questions reflect the functions that a navigation system must perform (Levitt and Lawton 1990; Murphy 2000).

- *Where am I going?* The robot should be clear about where it needs to go, and the destination is usually determined and assigned by a human operator or a machine-based mission planner. Some tasks may require that a set of sub-goals is followed along the route to the final destination. The answer to this question is assumed to be known in most robot scenarios.
- *What is the best way to get there?* The robot needs a plan to reach the destination efficiently and with consideration of optimisation criteria. Although navigation is more than just path planning, this vital area has received considerable research attention.
- *Where have I been?* Map building helps the robot identify where it has been, allowing it to incrementally gain knowledge of previously unknown parts of its environment. Even if the robot is operating in areas of its environment previously visited, future performance may not only be improved by refining stored information, but any changes since the last visit can be re-mapped. The representation of environmental knowledge should be in a form which aids retrieval and augmentation.

- *Where am I?* This is the localisation problem. The robot should have knowledge of where it is now, so that its next step can be accurately determined. Although global localisation provides a unique identification of position, localisation is often relative to a local landmark, such as the corner of a street. If localisation is inexact, the robot may fail to recognise when it has returned to a point already mapped and consequently build an inaccurate map with duplicated entries. Localisation errors are generally cumulative, in that the further the robot travels the greater the error will be.

Leonard and Durrant-Whyte (1991) added the question "How should I get there?" that emphasises there may be a number of alternative routes that need to be considered. This is closely related to the path planning problem, an area that is given a particular emphasis in this thesis.

1.2.1 Deliberative, reactive and hybrid systems

Traditionally, two types of control methods have been adopted for robot navigation, namely deliberative planning and reactive behaviour (Arkin 1998; Kortenkamp, Bonasso and Murphy 1998; Lyons 1992; Mali 2002; Muñoz-Salinas *et al.* 2005; Murphy 2000; Orebäck and Christensen 2003; Stoytchev and Arkin 2004; Urdiales *et al.* 2003b). A solution that incorporates both deliberative and reactive components is termed a hybrid navigation system (for example Aguirre and González 2003; Arkin 1998; Kortenkamp, Bonasso and Murphy 1998; Lyons 1992; Mali 2002; Muñoz-Salinas *et al.* 2005; Murphy 2000; Orebäck and Christensen 2003).

Deliberative systems rely on an accurate world model to generate a plan for a given navigation task. The movement is directed by the decision, which is made in a hierarchical architecture involving functional decomposition, world modelling, and path planning (Arkin 1998; Murphy 2000). The use of perception in deliberative systems has been restricted to finding a means to ensure the accuracy of the global representation of the environment and no feedback to the planner exists from the robot action that results from the implementation of the plan (Arkin 1998; Mali 2002). The approaches in this category are also called model-based approaches, as the world

model is essential for deliberative reasoning. Deliberative navigation methods generally assume that the environment in which the robot moves, as well as its start and goal points, are known and the obstacles are either static or move in pre-defined manners. With such knowledge, the navigation task is to plan paths without collision based on complete knowledge of the working environment (Nearchou 1998; Xiao *et al.* 1997). Such global planning normally results in optimal or near-optimal movements of the robot when moving between specified pairs of locations. In dynamically changing environments, deliberative methods often need to re-plan movements and, as these calculations can be very time consuming, deliberative methods are often unsuitable for real-time navigation. Also, the assumption that the environment is completely known is improbable in practical applications. Consequently, deliberative approaches to planning a collision-free path are often criticised for not being able to deal with uncertainties in practical environments (Ryu and Yang 1999).

In reactive approaches, the robot performs an action according to the pattern of perceived sensor information and the direction of the goal. Consequently, the design of this type of control generally forms a tight linkage between stimulus and response to achieve real-time performance (Kortenkamp, Bonasso and Murphy 1998; Mali 2002; Urdiales *et al.* 2003b). The approaches of this type are also known as sensor-based approaches. As reactive methods make few, if any, assumptions regarding the arrangement of obstacles in its environment, they are often more robust in dealing with dynamic environments and are more tolerant to uncertainties in sensor measurements and the errors that accumulate during actuator movement sequences (Muñoz-Salinas *et al.* 2005; Ryu and Yang 1999; Urdiales *et al.* 2003b). The reactive approaches are often capable of autonomously exploring new regions in the environment and, as there is no plan to modify or repair, they are generally able to respond rapidly to any changes that may occur in the operating environment. However, without a global view of the environment, movements under reactive control are unlikely to be optimal and, as there is no memory of the locations at which previous decisions have been taken, localisation is not normally feasible, nor is escape possible from certain obstacle configurations.

Hybrid architectures make use of reactive motor behaviours that are activated according to a higher deliberative cognitive process (Aguirre and González 2003; Mali 2002; Murphy 2000; Orebäck and Christensen 2003). The sensory information about the environment may be shared between the two layers: a suitable behaviour can be generated based on this stimulus, while cognitive functions in the deliberative layer integrate the observation into a world model. A plan may be subsequently made with up-to-date knowledge of the environment to guide the robot in accomplishing the navigation task. An important issue arises from the investigation of hybrid systems, namely, what is the appropriate way to interface the deliberative and reactive systems in order to maximise navigation performance (Arkin 1998; Lyons 1992)? The hybrid systems presented in this thesis attempt to deliver a suitable solution to this problem.

1.2.2 Topological and metric navigation

Two navigation techniques, namely *topological navigation* and *metric navigation* (also sometimes known as *qualitative navigation* and *quantitative navigation* respectively) have emerged as distinct and popular strategies for representing robot environments (Murphy 2000; Ryu and Yang 1998; Thrun and Bucken 1996; Urdiales *et al.* 2003a).

Topological navigation is often viewed as a human-like way of navigating. If a visitor at a reception desk asks, 'Where is Scott?' directives such as, 'Pass through the door, go up the stairs, turn left at the second floor and enter the second room on the right at the end' may be given to guide the visitor to Scott. Such an approach relies on the human ability to identify and navigate based on landmarks or features, such as 'door', 'stairs', 'floor', 'room', and 'end'. Analogously, topological navigation uses distinctive landmarks (which can be natural or artificial) and their interconnections, to describe environments (topologically represented) and plan paths consisting of a sequence of identifiable landmarks. However, there are significant drawbacks: processing overheads are often substantial in realistic implementations, feature

extraction may require model-based vision processing and localisation can become difficult or even impossible if landmarks cannot be identified.

Metric navigation requires the existence of a map of the environment in which the robot's environment is defined by a single, global coordinate system. The paths generated by metric techniques can usually be decomposed into a set of path segments consisting of sub-goals with fixed locations or global coordinates. A data structure called *configuration space* (or *c-space* for short), is used in metric approaches to specify the position and orientation of the robot and the obstacles (that are represented in their own configuration space termed *c-obstacle*) (Choset *et al.* 2005; Latombe 1991; Lumelsky 2005). *C-space* reduces the robot's physical dimensions to a single point, so that the path planning problem can be simplified to moving a point through a scattered set of obstacles. The objects (except for the robot) in *c-space* are normally approximated by polygons in order to reduce planning complexity and memory usage.

The advantages and disadvantages of each type of navigation have been widely recognised (Aguirre and González 2003; Muñoz-Salinas *et al.* 2005; Murphy 2000; Ryu and Yang 1998; Urdiales *et al.* 2003b). Due to the compact characteristic of the topological representation, it scales better than metric maps for larger environments. Also, topological maps are more tolerant to errors in metric information, but the ability to distinguish landmarks has proved difficult to solve in many practical situations, particularly when more than one landmark with the same or similar features is present. Additionally, global optimal navigation can be generated based on the metric representation, whereas it is unlikely to be able to generate an optimal solution (in terms of metric criteria) using the approaches relying on topological representation alone. Many recent navigation schemes rely on a hybrid representation (for example Aguirre and González 2003; Jia, Zhou and Chen 2004; Muñoz-Salinas *et al.* 2005; Poncela *et al.* 2002; Ryu and Yang 1998; Urdiales *et al.* 2003a and 2003b) that integrates the metric information with the topological representations to overcome the disadvantages of the individual navigation techniques.

1.3 Genetic algorithms

Genetic algorithms (GAs) generally refer to a family of computational models inspired from biological evolution, specifically those that follow the principle of 'survival of the fittest' firstly laid down by Charles Darwin (Goldberg 1989 and 2002; Holland 1975; Mitchell 1996; Nolfi and Floreano 2000; Osyczka 2002; Wang, Tan and Chew 2006; Watanabe and Hashem 2004). GAs are generally used as adaptive heuristic search algorithms, mimicking the natural evolutionary process and maintaining a population of candidate solutions or 'chromosomes' that are evolved over a series of generations. Competitive selection favours fitter chromosomes, pairs of which are chosen for mating to produce the next generation; the expectation being that the resulting offspring will also be fitter individuals, biasing the search towards regions in which fitter chromosomes have already been discovered. To avoid the loss of population diversity, and so reduce the possibility of terminating the search at a local optimum, mutation of the offspring often occurs with only a small probability.

Due to the implicit parallelism of GAs (Goldberg 1989 and 2002; Holland 1975; Mitchell 1996), they search a larger space with a relative small number of manipulations carried out on a set of artificial chromosomes. Results in the literature demonstrate that not only do GAs provide an alternative approach to solving problems, but outperform other methods for many real-world search-related problems (De Jong 1992; Osyczka 2002; Watanabe and Hashem 2004). With the simple and general form, GAs can operate on each kind of parameter space (such as, discrete, continues, or combinatorial spaces) to fulfil single or multiple optimisation criteria with no requirement of gradient information regarding the search space and any other internal knowledge (Beasley, Bull and Martin 1993; Wang, Tan and Chew 2006). The parallel implementation can be easily achieved with the concept of population, resulting in faster execution compared to conventional approaches (Cantu-Paz 2000; Watanabe and Hashem 2004). Previously intractable real-world problems can be solved with little need to perform deep analysis of the application itself (Rothlauf 2002). GAs are well suited to problems in which noise exists (Sano and Kita 2002; Watanabe and Hashem 2004) and GAs are sufficiently flexible to allow users to

modify the genetic operators (Carrano *et al.* 2006; Passone, Chung and Nassehi 2006) or even invent new operators (Vannoy and Xiao 2004) that effectively introduce domain knowledge to improve the performance for specific problems. The merits of GAs can be summarised as follows: large application domain, simple mechanisms, applicable to many parameter spaces, no gradient information or internal knowledge required, suitable for single and multiple objective optimisation, easy parallelisation, suitable for difficult problems, robustness and flexibility. Such advantages are central to the requirements of the work in this thesis, but GAs also have a number of drawbacks, including computational complexity (Watanabe and Hashem 2004), appropriate control parameters are difficult to choose (Goldberg 2002; Mitchell 1996), and no guarantee that the global optimal will be found (Chen, Lee and Park 2005; De Jong 1992; Rudolph 1994). Advances in computer hardware have to some extent alleviated the computational disadvantage, for example, Minami, Gao and Mae (2007) developed a GA for catching fish in real time. Moreover, calculation time can be reduced by parallel implementations (Cantu-Paz 2000; Watanabe and Hashem 2004). There remains no solid theoretical guidance for choosing appropriate GA parameters for a specific application and their determination is largely based on trail and error. It is difficult to predict or accurately govern the evolutionary progress to a sufficient extent that it can be guaranteed that the desired solutions can be found within a certain time limit. Other researchers have attempted to direct the search by incorporating domain knowledge as heuristics (for example Elshamli, Abdullah and Areibi 2004; Smierzchalski and Michalewicz 2000 and 2006; Zheng, Ding and Zhou 2003; Zheng *et al.* 2005). GAs cannot guarantee that, with a finite population, the global optimal solution can be determined every time and sub-optimal solutions are often accepted as a necessary consequence of the finite computational resources available in practice (Chen, Lee and Park 2005; Rudolph 1994).

1.3.1 A brief history and application examples

In the 1950s and 1960s, evolutionary systems were studied with the aim of establishing the mechanism as an optimisation tool for engineering problems. John Holland (1975) proposed GAs as a method for designing robust adaptive systems. His GA introduced most of the features of a modern GAs, such as a population and the

genetic operations of crossover, inversion, and mutation. The ongoing advances in computational power have also helped to establish GAs in many new areas of application.

GAs have been applied in a wide variety of fields. Examples in the field of engineering (Abraham, Jain and Goldberg 2005; Chambers 2000) include optimisation tasks such as circuitry routing, job-shop scheduling and automatic programming to generate complex programs from programming elements. The applications of GAs in machine learning include evolving sensors for robots, determining optimal weights for neural networks, and generating rules for classifiers systems. There are also successful examples of the application of GAs to evolve social behaviours, cooperation and communication in multi-agent systems.

1.3.2 Features of genetic algorithms and variants of the canonical form

There is no broadly accepted definition of a genetic algorithm that distinguishes it from other evolutionary computation methods (Mitchell 1996; Osyczka 2002; Watanabe and Hashem 2004). However, the salient features are those of a population of chromosomes, selection based on fitness and the application of genetic operators, and these are common across all variations of the canonical genetic algorithm, see Figure 1.1. Chromosomes are generally formed of fixed-length from a binary encoding of the problem being tackled (Abraham, Jain and Goldberg 2005; Goldberg 1989 and 2002; Holland 1975; Osyczka 2002; Rothlauf 2002; Wang, Tan and Chew 2006; Watanabe and Hashem 2004), although other encoding mechanisms, such as real values (Herrera and Lozano 2000; Hrstka and Kucerova 2004; Montana and Davis 1989; Suzuki, Sawai and Piaseczny 2006) and character sets (Kitano 1990; Rothlauf 2002) have been successively applied in specific applications. In addition, variable-length chromosomes (Goldberg, Korb and Deb 1989; Hutt and Warwick 2007; Kim and De Weck 2005), where the length of the chromosomes is adapted during the evolutionary process, can be well suited to problems where the length of the optimal search path can vary greatly from one application of the GA to the next. A good example of adaptive encoding is found in the messy GA (Goldberg, Korb and Deb 1989), a form that was specifically developed to improve the performance of

GAs by constructing longer fitter chromosomes from combinations of smaller elite seeds. In GAs, the selection to bias individuals towards promising regions of the search space is a sensitive parameter: too high a selection pressure results in rapid convergence towards sub-optimal solutions, but if it is too low, the evolutionary process is likely to be rather slow.

```
procedure canonical genetic algorithm
  begin
    generate initial population randomly
    evaluate the fitness for each individual
    while optimisation criteria not met
      select parents
      apply crossover, mutation operators to parents to produce offspring
      evaluate the fitness for offspring
      form a generation
    end while
  end
end procedure
```

Figure 1.1 The structure of canonical genetic algorithm.

GA selection schemes can be classified into two categories, namely proportionate and ordinal-based selection (Ahn and Ramakrishna 2002; Goldberg 2002). Proportionate selection chooses individuals according to their relative fitness, examples being the roulette wheel (De Jong 1975), stochastic remainder (Booker 1982; Brindle 1981) and stochastic universal selection (Baker 1987; Grefenstette and Baker 1989). Ordinal-based selection ranks individuals in the population, examples are tournament (Brindle 1981), truncation (Mühlenbein and Schlierkamp-Voosen 1993) and ranking selection (Baker 1985).

The most commonly-used genetic operators are crossover and mutation, but a number of other operators often feature, particularly inversion, delete and swap (Goldberg, 2002; Mitchell 1996). In applying GAs, a range of parameters need to be set, and these include population size, the number of generations, and the application rates of the applied operators. There are no general quantitative rules for choosing these parameters, suitable values depend largely on the nature of the problem under investigation and values are normally determined experimentally (De Jong 1975; Mitchell 1996). Altering the parameter values will likely affect the performance of the GA in terms of the rate of convergence and the quality of the solution produced.

There is a number of different ways of replacing the population with generated offspring. In generational GAs, the next generation is normally formed of individuals resulting from genetic operations on the old population, which is entirely discarded (Goldberg 1989 and 2002; Holland 1975). Elitist replacement strategies (Dumitrescu *et al.* 2000; Reed, Minsker and Goldberg 2001; Rudolph 1994) clone a (typically small) number of best fit solutions directly into the next generation without any genetic alteration and the remainder of the new population is filled with offspring modified by operators. In the tournament replacement scheme (Smith 2007), the members to be inserted into the new population are determined by tournament between the individuals in the current population and the offspring generated. The crowding approach (De Jong 1975) and its variants (Affenzeller and Wagner 2004; Mahfoud 1995a; Mengsheol and Goldberg 1999; Sareni and Krahenbuhl 1998) have this replacement strategy. Another replacement approach, less common due to ineffectiveness, is the random replacement scheme (Ballester and Carter 2003), in which only those individuals randomly selected from the current population will be replaced by offspring.

The following are examples of variants on the canonical GA form.

- *Modifying one or more GA features, the genetic representation or the structure* Examples are messy GAs (Goldberg, Korb and Deb 1989), mentioned above, that permit variable-length chromosomes and steady-state GA (Syswerda 1991; Whitley 1989; Whitley and Kauth 1988) in which only a small fraction of population is involved in the genetic reproduction for the next generation.
- *Adapting parameters during the evolutionary process* An example is the work of Jerald *et al.* (2005) who proposed a GA that adapts the probabilities of application of its genetic operators.
- *Implementing GAs in parallel* An additional operator, namely migration, is commonly found in parallel GAs to define the degree of interaction between separate sub-population streams (Conceicao Antonio 2006; Srinivasa, Venugopal and Patnaik 2007).

- *Niched GAs* Niching techniques attempt to maintain population diversity, allowing separate promising regions to be investigated simultaneously and reducing the risk of concentrating the search in areas that may lead to only a locally optimal solution. Crowding (De Jong 1975) and sharing (Goldberg and Rechardson 1987) techniques are common examples.

GAs have also been combined with other optimisation algorithms in order to improve the overall search performance, such as in the simulated annealing genetic algorithm (Wang, Z.G. *et al.* 2005; Yildirim, Erkan and Ozturk 2006). However, such hybrid algorithms may behave very differently from GAs.

The basic idea about GAs was given in this section and the next chapter extends this brief introduction by discussing a number of important aspects of GAs based on the recent developments found in the literature.

1.4 Research aim and objectives

The aim of the research was to design an autonomous navigation system for a mobile robot that has no *a priori* knowledge of the environment. Once the robot has had the opportunity to move through the environment, either as a consequence of navigation or exploration activities, it should be capable of obtaining and storing information regarding the environment for future use of planning by genetic algorithms.

The aim was achieved in a sequence of logical stages that can be formulated as the following objectives.

1. To reduce the time taken to generate plans for navigating through environments that contain known static obstacles. It is likely that this will require the development of a suitable and novel method for representing the obstacles.
2. To extend the planner for static environments in such a way that the navigation technique can also deal with dynamic obstacles whose paths may not be known. This is likely to require the incorporation of motion parameter into planning process.

3. To develop a means of automatically gathering information of the environment as the robot moves among the static obstacles. It is likely that a hybrid solution will be required, in which a reactive navigator will guide the initial movements and the information gained then communicated to a high-level planner. As more is learned of the environment, so the planner will become better placed to plan future movements.
4. To enhance the operation of the reactive part of the hybrid navigation solution so that it is able to avoid moving obstacles with minimal disruption to the overall navigation plan.
5. To implement a generalised version of the hybrid navigation system that is able to provide navigation from any start point to any goal point in the environment. This will require that additional information is gathered from the obstacles and modelled in a suitable form that is not overly extravagant in terms of memory usage or planning time.

1.5 Contributions to knowledge

The contributions of the research project reported in this thesis are as follows.

1. A new genetic-based planner for stationary environments was developed that included the novel aspect of constraining the search space to only a set of vertices. The planner was found to significantly reduce planning time compared with earlier evolutionary planners, yet generated a similar quality of path.
2. In an extension to the vertex planner, the adaptive modification of the planned route was permitted in order to allow it to be changed in response to observed moving obstacles. The speed parameters of both the robot and the moving obstacles were encoded as part of the genetic planning process, allowing the selection of an appropriate robot speed for each path segment.
3. The novel waypoint-based hybrid navigation system further simplified the representation of the environment, reduced the memory storage requirement for the environmental knowledge and considerably shortened the time to deliver a suitable path between the detected waypoints.

4. Two contributions can be drawn from the research work on the waypoint-based hybrid navigation system for dynamic environments. Firstly, a set of algorithms was developed to enable the robot to avoid potential collisions with the moving obstacles sensed. Secondly, a new method of statistical exploration was devised to identify the dynamic patterns that potentially characterise a dynamic environment.
5. The generalised version of the waypoint navigation system enabled the formulation of planned paths between any pair of locations using waypoints determined during the previous tasks. This approach required the development of a novel method of describing the environmental knowledge elicited. Although significantly more knowledge of the environment now needed to be stored, its memory requirement was kept to a minimum by a piece-wise polynomial representation.

1.6 Structure of the thesis

The next chapter provides a deeper introduction to GAs based on a survey of recent developments of GAs related to the algorithms proposed in the thesis. The three principal mobile robot navigation approaches, namely planner-based, reactive and hybrid systems are all relevant to the current research and a review of work found in the literature in each of these areas is discussed in chapter 3. Two specific planning systems described by other authors are discussed in detail as they are closely related to the current work.

Chapters 4 to 8 all describe new work performed by the author. A new genetic-based planner that uses the vertices of obstacles to create paths through static obstacles is described in chapter 4. Chapter 5 extends the vertex planner of chapter 4, so that it is able to perform navigation in environments that contain dynamic obstacles as well as static obstacles. Chapter 6 describes a novel waypoint navigation system that is able to gather information autonomously about the environment for use in the generation of future plans. Chapter 7 presents a hybrid navigation system which augments the navigation ability of the waypoint navigation system presented in chapter 6, so that it

can be applied to dynamic environments. A generalised waypoint navigation system is described in chapter 8 and this navigation system enables path planning for future navigation tasks that can start at any point and end at any point. The thesis is concluded in chapter 9, in which the research reported in the thesis is summarised, achievements are outlined, shortcomings are discussed, and future work is proposed.

Chapter 2

GENETIC ALGORITHMS REVIEW

The previous chapter presented an overview of the research project, introduced mobile robots, described the general nature of navigation problems and gave a brief introduction to genetic algorithms (GAs). This chapter reviews the recent developments in GAs with particular emphasis on the aspects relevant to the algorithms proposed in this thesis and justification of the choice of GA structure adopted in this thesis. For the earlier work on GAs, good reviews can be found in Bäck, Hammel and Schwefel (1997), Chaiyaratana and Zalzala (1997) and De Jong and Spears (1993). A survey of the application of GAs in the robot navigation is given in the next chapter.

The GA literature is extensive. All recent work cannot be covered in a thesis: many publications of high quality have had to be omitted, and emphasis is given to these papers of upmost relevance to that investigated in this thesis. This chapter is arranged as follows: steady-state GAs are firstly introduced, followed by genetic representation, the selection schemes, genetic operators, and deterministic crowding is introduced in the context of premature convergence and population diversity.

2.1 Steady-state genetic algorithms

As the steady-state GA is the main structure of the GAs developed in this thesis, this section compares its operation to the generational GA and explains the choice made.

An implementation termed Genitor (Whitley 1989; Whitley and Kauth 1988) was probably the first realisation of the form of GA that later became known as 'steady-state' (Syswerda 1991). The majority of GAs described in the literature are generational, meaning that each new generation produces an offspring population that entirely replaces the previous population. In contrast, a small number of individuals (typically one or two) in the steady-state GA are involved in the genetic modification at each iteration, with replacement strategy being such that only the worst individuals in the population are replaced by offspring. In addition, ranking methods are often used for selection in steady-state GAs to identify individuals for mating, rather than using the fitness values themselves. A number of earlier works (De Jong and Sarma 1992; Goldberg and Deb 1991; Syswerda 1991; Vavak and Fogarty 1996; Whitley 1989) has investigated steady-state GAs in their comparison with generational approaches. The effects on performance that arise due to the first two major differences (that is the number of individuals involved in genetic operation and replacement strategy) are discussed in this section and those that arise due to the difference in selection strategy are considered (in conjunction of other common selection schemes) in section 2.3.

As steady-state GAs modify only a small number of individuals at each iteration, comparisons of the frequency of alteration of individuals is difficult to make with generational GAs. The effect of 'birth and death rate' and 'life span' of the individuals in two different models has recently been examined by Jones and Soule (2006), in their application to a problem where the fitness landscape consists of a broad, low peak, and a narrow, high peak. A variable representation strategy was adopted for the generational and steady-state GAs, rather than using individuals of equal length as in standard GAs. The experimental results showed that the steady-state GA is able to smoothly converge to the higher peak once present in the

population, but the shift from the lower, broader peak to the higher, narrower peak occurs suddenly. This has been attributed by the authors to the different roles of genetic robustness (defined as a measure of the average fitness change of a chromosome following a genetic operation) in directing the evolutionary process. An individual is more genetically robust if it is located in a flatter peak, but less robust when on a narrow peak as it is more likely to be moved away. In steady-state GAs, elite individuals evolved from the previous generation are naturally inherited into successive generations and so fitness improvements are accumulated monotonically. In contrast, such smooth convergence is rarely observed in the generational models due to the entire population being replaced by their offspring, resulting in sudden shifts in locations. Bullinaria (2004) compared generational with steady-state GAs that were used to optimise the aspects of a neural network system, namely initial weight distributions, gradient descent learning rates, and regularisation parameters, in order to improve the performance of the neural network system. The simulation results indicated the performance of evolutionary strategies (generational or steady-state) is largely dependent on the specific problem being addressed and consequently evolutionary strategy should be tailored to fit that problem in order to achieve the best possible performance. Elitist selection (Dumitrescu *et al.* 2000; Reed, Minsker and Goldberg 2001) GAs were compared with steady-state GA by Shi *et al.* (2004), who concluded that the steady-state GA is simple and effective and performs well in low-dimensional environments, and is especially adapt at on-line optimisation, whereas the elitist selection GA was better in high-dimensional environments and in off-line optimisation. The relatively good on-line performance achieved by the steady-state GA was attributed to the replacement strategy used, and the better search capability of the elitist selection GA to the relatively larger number of schemata processed. Fewer schemata were explored by the steady-state GA, as only one new individual was examined at each generation. An effect of the steady-state replacement scheme is to make individuals more similar with each passing generation, resulting the loss of diversity, however, even the worst member still has a chance, though small, to be selected for genetic operation, alleviating the loss of allele. Rogers and Prügel-Bennett (1999b) performed a comparison of the dynamics of steady-state and generational GAs using a statistical mechanics approach. As the ranking selection and

least fit replacement makes it difficult to directly compare to the generational GA, the comparison was made by isolating the genetic operation and adopting the Boltzmann roulette wheel selection and random deletion for the two evolutionary models. The analysis results indicate that loss of population variance of the steady-state GA was twice as rapid as that of the generational GA. An additional experiment was conducted on the steady-state GA to investigate the effect of rescaling of selection pressure while considering mutation. It is found that, with weak selection, the steady-state GA was able to regenerate the same dynamics as that of the generational GA with only half computational effort in terms of function evaluations. The analysis of the steady-state GA based on the experimental results, is complicated by the effect of the application on performance. The theoretical approaches used to evaluate the pros and cons of steady-state GAs as compared with generational GAs have adopted the decomposition (divide and conquer) strategy, isolating each part or set of parts and evaluating them alone. However, the interaction between the selection, genetic modification and replacement makes the theoretical analysis complex and non-linear, and no investigation has been carried out to date. Despite the theoretical investigations, steady-state GAs has increasingly employed in many fields often with some modifications of its original version, such as Li and Kou (2005), Miconi and Channon (2006), Raghuwanshi and Kakde (2006), Sasaki *et al.* (2006) and Shi, Cui and Zhang (2004).

As only a single reproduction was designed to be carried out at each replacement in steady-state GAs used in this thesis, the evolutionary process can be interrupted at any time in order to extract the current best solution. Figure 2.1 illustrates a possible arrangement for a sensor-based planning system, where every generational operation is arranged in the sampling interval, so that the planning continues while the robot moves.

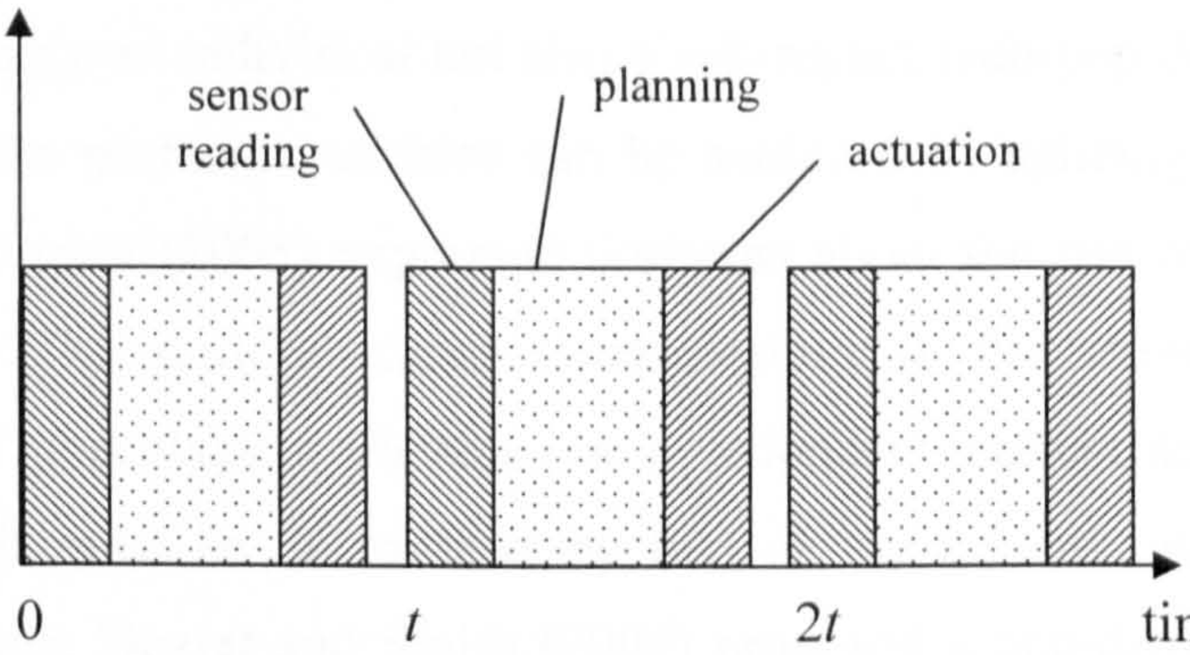


Figure 2.1. Planning arrangement for a sensor-based system. Note that t is the sampling interval.

2.2 Genetic representation

The genetic representation stores the genotypic information used to determine the phenotypic attributes (such as eye colour, hair colour and shape). A chromosome consists of a number of alleles whose value is one drawn from the set of possible values. For example, the value can be 0 or 1 for a binary allele whose cardinality is 2. A specific phenotypic property is determined by a gene containing one or more alleles. The representation is important in constructing an efficient GA (Bäck, Fogel and Michalewicz 1997; Rothlauf 2002). Binary representation is simple and commonly used, but many other encoding methods (such as integer, floating point, tree, and so on) have emerged in the literature. A number of publications (Bäck, Fogel and Michalewicz 1997; Larrañaga *et al*. 1999; Mitchell 1996) have summarised the earlier representation schemes, but research in the effects of choosing particular genetic representations has not been as active as in other aspects of GAs (such as genetic operators or selection strategies). This section presents the recent developments on the most common representations.

Binary representations are the most widely used representations as they are simple to formulate and can be manipulated directly by digital computers. A splicing/decomposable binary representation was developed by Leung, Sun and Xu (2002), Liang, Leung and Lee (2006), Liang, Leung and Xu (2007) and Xu *et al*.

(2003). A cell decomposition strategy was used to construct a binary string which represents not only an individual but also a sub-region (sub-population) and a higher resolution for the problem variables can be achieved by splicing one with another. Fonseca and Correia (2005) expressed concerns about the use of redundant binary representation. The non-redundant representation is the identity map between genotypes and phenotypes, whereas, in a redundant representation, at least one phenotypic trait must be determined by two or more genes, which is known as polygene. Dengiz, Dozier and Smith (2004) proposed a non-deterministic decoding technique for binary representation that maps an individual, not to the same point every time as the conventional decoding approaches, but to a Gaussian neighbourhood around it. Zhao and Long (2005) combined standard binary coding with gray coding (Whitley, Rana and Heckendorn 1997) into a new binary representation, as there is often lack of *a priori* knowledge about which representation is suitable for a given real-world problem. The gray coding was used to eliminate *Hamming cliffs* (Rowe *et al.* 2004; Whitley, Rana and Heckendorn 1997) corresponding to an adjacent locations in numeric space whose genetic representation are bit complementary. For example, binary strings, 0111 and 1000, represent 7 and 8 in its numeric space, but a significant change in the genotype is required for a minor change to the decimal equivalent.

Floating point representations (Bäck, Fogel and Michalewicz 1997) operate directly in continuous space rather than on the discrete set used by binary representations. Gaing and Huang (2004) presented a mixed integer representation containing continuous and discrete control variables for non-convex optimal power flow problems. Examples adopting the floating point representation are Abbas and Bayoumi (2006), Elshamli, Abdullah and Areibi (2004), Zheng, Ding and Zhou (2003) and Zheng *et al.* (2005). Recent literature contains few contributions that describe any significant modification of the conventional floating point representations. Pereira *et al.* (2002) presented a genetic vehicle representation consisting of several routes, each of them composed by an order list represented by integer.

Tree representations are commonly used where mapping between the phenotypes and genotypes can be realised by a tree-like graph. Chang, Hou and Su (2006) developed a binary tree structure to represent RLC (resistor, inductor and capacitor) circuits, in which the labelled terminal nodes consists of the three types of electrical components and the non-terminal nodes represent either series or parallel connections. Tree structures have also been used to represent the language sentences (for example Lim and Cho 2005).

Efforts have also been made to combine different representation schemes into hybrid representations with the intention of gaining efficiency and flexibility. Schnier and Yao (2000) described a hybrid strategy to create an initial population in which half of the individuals used a Cartesian representation and half a pseudo-polar representation. Aguilar-Ruiz, Giraldez and Riquelme (2007) proposed a method of hybrid coding for decision rule learning problems, termed nature coding, that combined binary and floating point representations to encode discrete and continuous attributes respectively.

Although attempts have recently been made to improve the efficiency of genetic representation, there is still an absence of solid guidance for choosing a suitable representation for a specific class of problems.

2.3 Selection schemes

The role of selection is to favour the fitter individuals over the less fit ones and determine which individuals should participate in mating. Consequently, it is critical to choose an appropriate selection mechanism to achieve a fast convergence but with a desired quality of solution. A number of earlier publications (Blickle and Thiele 1997; Goldberg and Deb 1991; Wiese and Goodwin 1998) provided comprehensive analyses on selection algorithms. This section attempts to summarise recent contributions available in the literature with regard to selection mechanisms, before a brief introduction is given on common selection methods.

As introduced in section 1.3.2, the most common selection schemes used in proportionate selection are roulette wheel, stochastic remainder and stochastic universal selection (Ahn and Ramakrishna 2002; Goldberg 2002; Mitchell 1996). The roulette wheel method (De Jong 1975) randomly selects the parents by spinning a wheel whose slot size is tailored to be proportional to the fitness of individuals. Stochastic remainder selection (Booker 1982; Brindle 1981) works as follows. The expected number of offspring for each individual is found; its integer part is used to determine the number of samples for the individual and the fractional component is used as the probability of whether this particular individual produces an additional offspring. Instead of the single pointer used in the roulette wheel method, stochastic universal selection (Baker 1987; Grefenstette and Baker 1989) employs N pointers equally separated where positions are determined by a single number randomly generated in the range [0, 1/N], generally resulting in a more diverse set of individuals. Such proportionate-based selection tends to result in a rapid convergence to promising subspaces during the initial stages of evolution, but the much lower selective differential provides little incentive to prefer one individual over another in the later stages of evolution (De Jong 1992). To preserve a constant selective pressure doing evolution, a number of ordinal-based approaches, namely tournament (Brindle 1981), truncation (Mühlenbein and Schlierkamp-Voosen 1993) and ranking selection (Baker 1985), have emerged to give independence to the raw values of fitness of individuals.

Ranking selection (Baker 1985) chooses individuals based on their rank allocated after sorting according to fitness, so that the selective pressure is independent of the fitness distribution of the population. In the tournament selection scheme (Brindle 1981), the winner is selected for reproduction from a number (specified by the tournament size) of individuals randomly chosen. The tournament size is directly related to the selective pressure, since the expected number of times the fittest individuals would be selected is equal to the tournament size (Sokolov, Whitley and da Motta Salles Barreto 2007). In truncation selection with a threshold (Mühlenbein and Schlierkamp-Voosen 1993), a fraction of fittest individuals are selected with

same probability for genetic reproduction. The specified threshold then controls the selective pressure.

A number of recent publications have concentrated on the loss of diversity that results from conventional selections approaches. Hutter (2002) introduced a new selection algorithm, termed the fitness uniform selection scheme (FUSS). The majority of popular selection schemes (proportionate, truncation, ranking, and tournament) propagate the genetic material of fitter individuals and inhibit the spread of poor quality genes in the population. However, Hutter (2002) argued that it is not necessary for the entire population to converge to a small sub-region of the search space, as, in most situations, a single individual of maximal fitness is sufficient, and the population exhibits low diversity. FUSS was proposed to overcome these drawbacks. In FUSS, a fitness value is randomly chosen uniformly from the interval between the maximum and minimum fitness values, then, the selection algorithm selects the individual with the shortest distance to the chosen fitness value. A copy of the individual selected is inserted in the population after genetic modification, increasing the space available for the increased size of population. There is no convergence of the population, as the selection pressure is specific to a local fitness level (although a new fitness level may yield). The probability that a specific individual is chosen is proportional to the distance to the nearest fitness level, and, as the selection is local, a slow evolutionary process may be expected. Furthermore, a population with increasing size will not only require extra memory but will take more time to process. Appropriate selective pressure is needed in order to meet the requirements, with high selective pressure being desired if quick convergence is required with a less accurate solution, but a fine quality solution being chosen if more detailed exploration is needed. Consequently, suitable selection should balance exploration (into new and undiscorved areas) and exploitation (in the immediate regions around solutions so gained) according to the specific requirements of a given problem.

A dynamic selection scheme was proposed by Agrawal *et al.* (2005) based on proportionate selection, where the criteria for choosing parents varies during

evolution and depends on the number of generations in a run and the diversity of the current generation. Although an improvement in the population diversity was obtained, slower convergence was also observed. Affenzeller and Wagner (2005) proposed a mechanism termed offspring selection, so called as the selection is performed after genetic reproduction. When the offspring are generated, they are compared to their parents and are accepted as a member of next generation, if and only if they are fitter than their parents. This selection approach is similar to 'pre-selection' scheme of Cavicchio (1970), see section 2.5.

The standard tournament selection (also called random tournament selection as the individuals in the tournament are chosen at random) is biased and can result in loss of diversity in that it is likely that some individuals may not be sampled at all, yet others may be sampled too frequently (Poli 2005). Further, the larger the tournament, the more rapidly loss of population diversity occurs. To overcome this drawback, an unbiased tournament selection strategy (Sokolov and Whitley 2005; Sokolov, Whitley and da Motta Salles Barreto 2007) was proposed. In place of uniform sampling, the unbiased tournament selection technique uses a set of permutations of set size equal to the tournament size during tournament construction, thereby guaranteeing that every member in the population is selected for a number of times equal to the tournament size. One drawback recognised by the authors is that the unbiased tournament selection reduces the degree of parallelism compared with standard tournament selection. However, the experiments presented indicated better results were obtained by the unbiased tournament selection than conventional tournament selection when applied to generational GAs. Also, it appeared that the tournaments of smaller size benefited more from unbiased selection technique. The authors also remarked that this variant of tournament selection may not be suitable for steady-state implementation as it performs recombination one-at-a-time. Although tournament size needs to be determined before evolution commences (it being tightly related to the desired selective pressure), it is difficult to gain *a priori* knowledge of the appropriate selective pressure required by many real-world problems.

The ranking selection used in the steady-state GA implementations ensures that a constant selective differential is maintained between the best and worst individuals during the evolution process (assuming the population size is fixed). Scaling may be applied to the ranks of the individuals before selection in order to produce appropriate selection pressure: for example, the quadratic ranking technique (De Jong 1992; Watanabe and Hashem 2004) scales the ranks of the individuals by $1/\sqrt{r}$, where r is an individual's rank, with the intention of increasing the probability of selecting fitter individuals. The combination of rank scaling and roulette wheel selection (Goldberg 2002) can prevent the loss of diversity to some degree (Sokolov, Whitley and da Motta Salles Barreto 2007).

2.4 Genetic operators

A wide range of different genetic operators have been applied across a range of applications. This section does not attempt to cover all the existing operators, but rather concentrates on those most relevant to the work presented in this thesis. In canonical GAs, significant emphasis was given to the crossover operator as it attempts to combine useful building blocks to generate better solutions. On the other hand, the mutation operator applied with a small probability perturbs the genetic structure to promote diversity. The inversion operator proposed by Holland (1975) is used to reorder the positions of alleles in order to increase the probability of linkage of fitter schemata and reduce the disruptive effect due to the one-point crossover operator (Goldberg 1989 and 2002; Mitchell 1996). A good review of the crossover and mutation operators developed in earlier works was provided in (Larrañaga *et al.* 1999) for travelling salesman problems. Spears (1997) reviewed the earlier development of recombination (crossover) operators. Since the development of Holland's schema theory (Holland 1975), there has been considerable discussion regarding crossover and mutation operators, mainly from the two aspects, namely 'disruption' (Goldberg 1989 and 2002; Holland 1975) and 'construction' (Spears 1992; Spears and De Jong 1998). This discussion continues with the establishment of "no free lunch" theorem (Ho and Pepyne 2002; Koehler 2007; Koppen 2004; Schumacher, Vose and Whitley 2001; Wolpert and Macready 1997).

Genetic operators can be broadly classified into *sexual* (where genetic material from two individuals are combined), or *asexual* (where only the genetic material of one individual is operated upon). Recent research implies a trend that genetic operators employed in many GAs are created or modified from their prototypes by incorporating domain knowledge to enhance effectiveness and efficiency.

The simplest sexual operator, one-point crossover operator (Goldberg 1989 and 2002; Holland 1975; Mitchell 1996), operates on a pair of equal-length chromosomes chosen by the selection schemes described in the previous section, and offspring are generated by swapping the features of the chromosomes after a randomly selected locus. Such an operator has also been extended to individuals of variable length (note that the name of the operator in such circumstances is 'cut and slice'), see Elshamli, Abdullah and Areibi (2004), Hu and Yang (2004), Nearchou (1999), and Tu and Yang (2003). In addition, the crossing site can be deterministically selected rather than randomly in order to satisfy the specific constraints or improve search performance. Such a strategy has been adopted by a number of authors (Ahn and Ramakrishna 2002; Davies and Lingras 2003; Wu and Ruan 2004).

A straightforward extension to single point crossover is to select more than one crossing point and exchange the genetic material lying between the two points. While the quantity of genetic information interchanged is increased by multiple point crossover (De Jong 1975; Goldberg 1989 and 2002), it would not necessarily improve convergence due to the increased probability of disruption of useful sequences. Conventional multiple point crossover operators randomly determine the crossing sites, but Davies and Lingras (2003) restricted the positions of crossing sites to the common genes between the parents with the gene segment between the crossing loci being kept in the same order, thereby ensuring both connectivity and feasibility of the resulting offspring paths.

In uniform crossover exchange occurs at the gene rather than segment level (Mitchell 1996; Osyczka 2002; Syswerda 1989), with each gene having a chance (known as

mixing rate and typically taking a value of 0.5) of being swapped between two parents. However, the major disadvantage of uniform crossover is the increased probability of destroying building blocks making it unsuitable in many applications, including the path planning problem.

The above crossover operators do not alter the content of each allele directly, only changing it by swapping genetic information, from their parents. In arithmetic crossover, however, the genetic contents in the offspring may differ from those of their parents (Michalewicz 1994; Osyczka 2002). Arithmetic crossover operates on floating-point chromosomes to provide a means of local search, defining a linear combination of two parents in the following form.

$$C_1 = \alpha \cdot P_1 + (1-\alpha) \cdot P_2 \qquad \text{Equation 2.1}$$

$$C_2 = (1-\alpha) \cdot P_1 + \alpha \cdot P_2 \qquad \text{Equation 2.2}$$

where C_1 and C_2 are the offspring following genetic modifications on P_1 and P_2, and α is weighting factor randomly chosen from the interval [0, 1]. Cai and Peng (2001) used this operator with a fixed-length representation for the path planning problem. For a chromosome of variable length, it may be necessary to determine which gene of a parent should be combined with a particular gene of the other parent. Therefore, its application has rarely been found in the path planning problem, though some algorithms (Smierzchalski and Michalewicz 2000 and 2006; Xiao *et al.* 1997; Zheng, Ding and Zhou 2003) can directly operate in a continuous workspace.

The mutation operator (Goldberg 1989 and 2002; Holland 1975; Mitchell 1996; Watanabe and Hashem 2004) is the most common asexual operator and which functions by occasionally inverting a single gene of individuals, normally with a small probability of being applied. The application frequency needs to be carefully selected; too high a value results in a random search, but too low and the population will have little diversity. As an appropriate compromise is hard to achieve, the rate for mutation has been adaptively modified during the search process in a number of GAs (Glickman and Sycara 2000; Li *et al.* 2006; Wang, H.J. *et al.* 2005). In a case

study, Glickman and Sycara (2000) showed that a self-adapting mutation rate can lead to premature convergence. Ahn and Ramakrishna (2002) modified the conventional mutation operator in the way such that the partial segment after a randomly selected mutation point of an individual is regenerated by the initialisation mechanism. A similar mutation strategy was utilised by Ji, Iwamura and Shao (2007).

There are many other asexual operators, such as repair (Hocaoglu and Sanderson 2001; Hu and Yang 2004; Smierzchalski and Michalewicz 2000 and 2006; Xiao *et al.* 1997) and insertion (Hocaoglu and Sanderson 2001; Smierzchalski and Michalewicz 2000 and 2006; Xiao *et al.* 1997), which often applied using specific knowledge of the particular application. Due to their relevance to the current work, a detailed description of these operators will be given in later chapters.

Many variants of genetic operators have been devised for specific applications. To improve their effectiveness and efficiency, the current trend is to incorporate domain knowledge as heuristics to guide the search more directly to the more promising regions. Although rapid convergence may be achieved by heuristic guidance, there is risk that search may become quickly trapped in a local minimum. Heuristics involved in the application of the operators may be complex and so result in substantial computational overheads.

2.5 Premature convergence and diversity

Premature convergence refers to the phenomenon where the evolutionary process becomes stagnated in sub-optimal solutions and further improvement cannot be realised by the further application of genetic operations as the population does not contain suitable genetic material (Affenzeller and Wagner 2004; Fogel 1994; Goldberg 2002). This phenomenon can also be considered as a loss of diversity in the population. The similarity of individuals can be measured based on the genotypes or phenotypes (Burke, Gustafson and Kendall 2002; Luerssen 2005; Sareni and Krahenbuhl 1998). Due to high selection pressure, elite genes may become dominant quickly in the population before the other areas in the search space have been fully

examined. Section 2.3 described common selection strategies and recent attempts to deploy selection methods in such a manner so as to preserve the population diversity.

Even in the absence of selection, population members will converge to a given point in the search space due to the accumulation of stochastic errors; this important phenomenon is known as genetic drift (Affenzeller and Wagner 2004; Chaiyaratana and Zalzala 1997; Mahfoud 1994). More specifically, a predominant gene may propagate into the entire population, driving out other alleles, and resulting in a loss of genetic variation. The replacement strategy determines the survival of individuals and so can also influence both population diversity and convergence performance. In generational GAs, the entire population is normally replaced at each generation by the new offspring (Goldberg 1989 and 2002; Mitchell 1996), but such a replacement strategy may result in the loss of elite individuals, as there is no guarantee that the offspring generated are of better fitness than their parents. To overcome this drawback, an elitism strategy (Dumitrescu *et al.* 2000; Reed, Minsker and Goldberg 2001; Rudolph 1994) was developed to clone the best individuals and ensure their presence in the next generation (perhaps in addition to their offspring depending on whether the individuals were involved in genetic operations). The drawback of the replacement scheme adopted in the steady-state GAs is a loss of population diversity (Rogers and Prügel-Bennett 1999a; Smith 2007). The elite individuals discovered early in the evolutionary process may well cause the entire population to converge to the same niche, but this may only contain a locally optimal solution, whereas those individuals in different niches, one of which may contain the global optimum, are likely to die out rapidly.

To combat loss of diversity, the mutation operator introduces new genes into the population. In particular, the diversity introduced by the mutation operator in the later evolution phases helps the escape from sub-optimal regions. Another strategy to promote population diversity is reseeding, in which the population is augmented by new individuals. For example, Rasheed (1998) developed a GA in which a set of randomly generated individuals were inserted into a highly converged population in order to increase diversity. More recently, Amor and Rettinger (2005) used self-

organising maps to mine data from previous evolutionary processes and performed reseeding be re-introducing a set of individuals not presented in earlier generations. Niching methods are inspired by an ecological phenomenon in which the individuals in separate but otherwise identical ecological niches compete with one another for limited resources so that a range of species can be retained (Dick 2005; Mahfoud 1995a and 1995b; Sareni and Krahenbuhl 1998; Singh and Deb 2006).

One of the earliest attempts at maintaining population diversity is the so-called 'pre-selection' scheme of Cavicchio (Cavicchio 1970; Mahfoud 1992) and is based on the principle that only if an offspring is fitter than the worst parent is that parent replaced. Holland (1975) introduced the sharing method to which further contributions were made by Goldberg and Richardson (1987). In the sharing method, the resources consumed by individuals should be proportional to the number of individuals in the niche. An important variant of the sharing method is called clearing (Dick 2005; Pétrowski 1996; Sareni and Krahenbuhl 1998), where, instead of sharing the niche resources amongst all members, all available resources are given to the winners in the niche. More recently, the localised niching concept was introduced by Dick (2005) and an implementation described as local clearing was applied to each location in one dimensional space, formulated as a ring structure by connecting the ends of the space (line). The individuals were equally placed around the ring and demes were constructed with a certain radius around each individual and the clearing method applied locally to each deme. Sharing approaches have been employed in the multimodal optimisation problems, such as Hiroyasu, Miki and Watanabe (1999), Lin and Wu (2002), Singh and Deb (2006), and Zhang *et al.* (2006). The deployment of the sharing method in conjunction with other algorithms was applied by Feng *et al.* (2006) where the sharing GA was used to perform global search whereas local search was executed by a bit-climbing technique. However, sharing and clearing methods both suffer from the need to determine a niche radius and an appropriate value for this parameter is difficult to estimate.

De Jong (1975) described a crowding technique in which only a fraction of the population, designated by the degree of generation gap, is involved in reproduction

during each generation and where the same number of individuals is always replaced. The replacement scheme is as follows. A sub-population of the size specified by the crowding factor is randomly drawn from the global population and the resulting offspring replace the individual that is genetically the most similar (based on phenotypic distance). In such a replacement strategy, especially when the crowding factor is small, replacement error will occur since the individuals to be replaced by offspring may not be similar to the offspring. Deterministic crowding (DC) was designed to reduce the replacement error and maintain the diversity by constraining the competition between the children and parents in identical niches (Mahfoud 1995a; Sareni and Krahenbuhl 1998). Two sets of tournaments are performed in DC, the first involving offspring C_1 pitted against parent P_1 and offspring C_2 against parent P_2 and the second involving offspring C_1 against parent P_2 and offspring C_2 against parent P_1. A parent is replaced by the offspring should the latter have better fitness. Another variant to crowding is probabilistic crowding (PC) (Mengsheol and Goldberg 1999) which performs DC but with probabilistic replacement. The winner of a competition in the tournament is determined probabilistically rather than deterministically based on their fitness values, so that the less fit individuals still have a chance to survive. PC is, however, likely to result in a slow convergence as compared with DC. Affenzeller and Wagner (2004) described a new mechanism to form the next generation by a competition between the offspring and their parents. The idea is similar to 'pre-selection' scheme of Cavicchio (1970), but the unsuccessful offspring are still given the chance to enter the next generation after a pre-defined fraction of the next generation is filled by successful offspring.

A number of authors (Abbass and Deb 2003; Bui, Branke and Abbass 2005; Toffolo and Benini 2003) introduced diversity into an objective and then applied a multi-objective evolutionary algorithm (MOEA) (Abraham, Jain and Goldberg 2005; Osyczka 2002). The solutions optimised by MOEA give a pareto-optimal trade-off between the diversity and any other objectives. Such an approach may be questionable since the diversity is really a means to improve search quality rather than being an objective. Furthermore, the solutions obtained in this way may not yield the global optimum with respect to other objectives as the optimal solutions of other

objectives may be dominated by the solutions of the objectives combined with diversity.

In summary, the diversity introduced by mutation operation is not sufficient to prevent premature convergence in many practical applications and consequently a number of approaches to improve diversity have been developed. Inserting a new set of individuals may promote population diversity to some degree, but there may be significant overlap between the newly introduced individuals and those individuals already presented and examined in the previous evolution, resulting in inefficient repeated evaluations. Sharing and clearing techniques require the determination of the radius of each niche which is difficult to obtain without *a priori* knowledge of the optimisation problems. MOEA introduces artificial objective of diversity often inhibiting the discovery of the true global optimum compared to the original single-objective optimisation approach. DC that replaces the parent individuals by fitter offspring in an identical niche not only preserves the diversity initially introduced, but also makes progress towards highly fit solutions. In contrast, PC is found to exhibit slow convergence to preserve the population diversity. It should be noted that the execution time is as important as solution quality in the project in this thesis, but quality needs to remain acceptable. DC may require a sufficiently large initial population to ensure sufficient building blocks are present in the gene pool to permit convergence to the global optimum when new genes are reinserted into the population less frequently. Consequently, it is likely that an appropriate population size will need to be determined experimentally (Mitchell 1996; Vekaria and Clack 1998).

2.6 Conclusions

This chapter has discussed the current state of research with respect to steady-state GAs, genetic representations, selection schemes, genetic operators and both premature convergence and diversity. The review of GAs given here is by no means exhaustive and, in particular, numerous successful applications have been reported from a wide range of fields that have not been covered. A range of introductory

publications (Goldberg 1989 and 2002; Holland 1975; Mitchell 1996; Nolfi and Floreano 2000; Osyczka 2002) are worthy of examination for background reading. It is important to stress that many issues concerning GAs remain unsolved; specifically, guidance for selecting appropriate genetic representations, selection schemes, genetic operators, and optimal parameter values, and the extent of the capabilities of GAs have yet to be fully established.

Chapter 3

NAVIGATION SYSTEMS REVIEW

Based on the recent relevant work in GAs, the previous chapter extended the introduction of GAs presented in section 1.3 by discussing those aspects related to the algorithms developed in this research project. This chapter reviews relevant work found in the literature and, in particular, navigation algorithms based on both evolutionary concept and hybrid approaches. Recent work in the literature that is most closely related to the work in this thesis is described in greater depth in the relevant chapter.

3.1 Planner-based navigation systems

Planner-based navigation systems are mainly concerned with path planning for mobile robots. The mobile robot path planning task can be described as finding a collision-free route from a specified start location to a desired goal destination while satisfying certain optimisation criteria (Xiao 1997; Yap 1987). Even in simple environments containing few obstacles, planning tasks are categorised as being both NP-complete and PSPACE-hard (Nearchou 1998; Smierzchalski and Michalewicz 2000). The model of the environment is normally constructed and maintained as a centralised representation in this type of navigation system. One of the first issues

that needs to be resolved when implementing any navigation system is to determine a suitable representation of the environment. The approaches to modelling spatial information can generally be classified as *topological* and *metric* mapping (see also section 1.2.2). Then, the information contained in the world model is used to generate an appropriate sequence of actions with certain constraints. Next, the design of a suitable planning algorithm needs to be considered. In order to optimise the efficiency of the process, the planning algorithm should generally be designed to match the representation method used to model the environment, although many algorithms in the literature can be applied to different models of the environment. Since no geometrical information is generally contained within topological maps, planning based on such maps focuses on those regions in the environment where a unique feature can be identified. On the other hand, metric information is often required in the quality evaluation of a generated path.

3.1.1 Environment representations

Topological representations have become increasingly popular in mobile robotics with the advance of sensing technology. A network-like graph is constructed containing a set of nodes corresponding to distinct places (such as corridors, halls) or landmarks (such as gateways) which are connected by arcs. Examples of this type of map can be found in (Gaspar, Winters and Santos-Victor 2000; Kortenkamp and Weymouth 1994; Ranganathan, Menegatti and Dellaert 2006; Remolina and Kuipers 2004). The major advantage of topological maps is that they are compact and consequently facilitate rapid planning. However, building topological map requires high quality sensors capable of identifying the unique features of landmarks. Furthermore, any error generated from inaccurate feature extraction may result in a failure of localisation. On the other hand, a single, global coordinate system is used in metric maps to represent geometric information regarding the robot's environment. The most representative approaches to representing metric information are grid (Arambula Cosio and Padilla Castaneda 2004; Payton, Rosenblatt and Keirsey 1993; Wang, Yong and Ang Jr. 2002), meadow (Arkin 1989), Voronoi diagrams (Latombe 1991; Mahkovic and Slivnik 2000), and visibility graphs (Latombe 1991; Maaref and Barret 2002). The grid representation is probably the simplest metric scheme and is

frequently employed in the situations where the exact representation of the objects is not a rigid requirement. The environment is decomposed into small square cells comprising a grid. If a cell is occupied, at least partially, by an obstacle, the cell will be marked as an occupied, otherwise, it will be labelled as unoccupied. A connective graph for path planning can be generated by linking vertices representing unoccupied cells. The meadow representation method transforms the free space (not occupied by any obstacle) into a set of convex polygons. A characteristic of the convex polygon is that it can be guaranteed that the robot will always traverse free space when it moves between any pair of vertices of a convex polygon. Paths which are equidistant from the closest pair of obstacles can be used to form the edges of a Voronoi diagram and the points where three or more such paths meet are presented by the vertices of the Voronoi diagram. A visibility graph is an undirected graph representing a number of inter-visible locations. The nodes of the graph denote the vertices of the configuration obstacles and each edge represents a visible connection between a pair of nodes. As such maps are easier to maintain, these techniques do, to some degree, simplify the planning task, however, their construction involves considerable computational overheads in terms of both execution time and memory usage. Simultaneous localisation and mapping (SLAM) (Frese, Larsson and Duckett 2005; Masson, Guivant and Nebot 2003; Smith and Cheeseman 1987) is a popular technique that is able to create an accurate metric map for an unknown environment while concurrently maintaining the robot's location. Noisy sensor readings and inaccurate motion models give rise to uncertainties that accumulate as the robot progresses, significantly distorting the map. SLAM employs statistical techniques, such as Kalman filters (Armesto and Tornero 2004) or particle filters (Adams, Zhang and Xie 2004; Howard 2006) to compensate for the uncertainty. The calculation complexity that results from the need to incrementally build a map and to maintain the constituent covariance matrices makes SLAM unsuitable for implementation in most embedded real-time systems. Further difficulties may arise when constructing a metric map using SLAM in environments that contain closed paths, although this can be alleviated by the recording of detailed feature information for later matching. Recent mapping scheme tends to integrate metric and topological representations to yield a hybrid solution. One way to construct a hybrid map is to annotate metrical

information into a topological map (Aguirre and González 2003; Ryu and Yang 1998). Alterbatively, the hybrid map can be created by extracting topological maps from metric ones (for example Jia, Zhou and Chen 2004; Poncela *et al.* 2002; Thrun *et al.* 1998; Urdiales *et al.* 2003a;). Hybrid maps are often found in the hybrid architectures (Aguirre and González 2003; Muñoz-Salinas *et al.* 2005) (see section 3.3, section 6.1 and section 7.1) since such a representation strategy facilitates fast planning and modifies local movements using the metric information in an on-line manner. Section 6.1 and section 7.1 discuss the hybrid representation strategies found in the literature.

3.1.2 Path planning approaches

As the environment is often modelled as a graph, the path between two specified locations can be generated using graph search algorithms. Breadth-first search algorithms begin at the root node and explore all neighbouring nodes before searching the next level, whereas the depth-first search algorithm searches each branch as far as possible before exploring the other branches (Cormen *et al.* 2001; Latombe 1991). Algorithms that perform complete examination of all nodes in order to find a path with minimum cost can guarantee that optimal paths are found, but are computational expensive, particularly in navigation systems that adopt metric representations. Dijkstra's algorithms (Dijkstra 1959) are commonly used to solve the shortest path problem for a graph with single source. Such a search begins from the source node on a directed graph containing non-negative weighted edges and iteratively performs expansion by adding an additional node with a minimum weight edge connecting the node with a minimum weight to the source node. In this way, the shortest path can be generated without visiting all nodes on the graph. During the search, a tree with all nodes visited and the length for each branch are maintained and the branch with the minimum cost is selected for expansion. The A* algorithm (Hart, Nilsson and Raphael 1968; Latombe 1991; Murphy 2000) is an extension of Dijkstra's algorithm that has the aim of reducing the number of nodes for investigation and this is achieved by incorporating a heuristic evaluation function that estimates the distance from any given node to the destination. The total lengths of the paths passing through candidate nodes can then be estimated by adding the distance

travelled so far to the estimate obtained from the heuristic function. The smallest value among the total lengths identifies the current path to explore further. As long as the path length estimated by the heuristic function is equal to or shorter than the actual path length, A* will always generate an optimal path. However, care is needed to derive a suitable heuristic function that ensures the lengths of unexplored paths are never overestimated as heuristic functions that do not meet this criteria are likely to reduce search efficiency or generate a non-optimal solution (Luger 2002). Due to the significant increase in computational complexity with problem size (normally measured by the number of nodes), Dijkstra's algorithm and A* are limited to the solution of small to medium sized problems (Soltani *et al.* 2002) and memory usage (the number of cells) is likely to increase significantly with problem size as all nodes visited during the search need to be stored. Dynamic A* (D*) (Stentz 1994) as an extension to A* is able to perform dynamic planning for mobile robots in unknown or partially-known environments, and has spawned several variants, including focused D* (Stentz 1995), D* lite (Koenig and Likhachev 2002), and anytime dynamic A* (AD*) (Likhachev *et al.* 2005). The D* algorithm operates in the manner described below. The initial plan is generated by Dijkstra from the goal to the start point based on known information regarding the environment (typically represented by a grid map) and stores the path length from each node to the goal. When the current path becomes impassable due to changes detected, the A* algorithm can be adapted to repair the path based on the updated states. As the most nodes remain unaltered and the path length from these nodes to the goal has already obtained during the initial plan, only relatively small portions of the tree need to be repaired by the A* algorithm, thereby gaining efficiency. However, the D* algorithm and its variations do not explicitly consider velocity factors in planning and memory is always needed to store the information gained during the initial plan or in an updated plan in order to enhance the efficiency of later planning that is required due to unexpected changes.

The field methods form gradients or potentials to derive a path flowing in an artificial field, defined by field values stored in the cells of a grid. Although a number of approaches exist (such as steepest descent (Snyman 2005), and wave front propagation (Murphy, Marzilli and Noll 1999)), the most popular of these is the

potential field (PF) method (Arambula Cosio and Padilla Castaneda 2004; Ren *et al.* 2007; Ren, McIsaac and Patel 2006; Wang, Yong and Ang Jr. 2002), in which the environment is characterised as an equivalent potential field containing sources and sinks. A smooth path can be generated by a combination of two virtual forces exerted on the robot: it is attracted by the virtual force generated from the attractive potential field around the goal point and repelled by the repulsive force of potential fields around obstacles. Since Khatib's seminal work (Khatib 1985 and 1986) on artificial field methods for robot path planning, PF has been the subject of investigation by many authors (examples are Arambula Cosio and Padilla Castaneda 2004; Ge and Cui 2000; Hussein and Elnagar 2002; Koren and Borenstein 1991; Ren *et al.* 2007). One of the reasons for the popularity of this method is its mathematical elegance and simplicity, although a number of deficiencies were found by Koren and Borenstien (1991). The most serious limitation inherent in the PF method is that the robot can easily become trapped in local minima when the virtual forces sum to zero. Substantial effort has been applied in addressing the limitations. Although fields free of local minima can be constructed by the approaches using simulated fluid mechanics (Rosell and Iñiguez 2002) or electro magnetic fields (Hussein and Elnagar 2002), the complexity involved in field construction prevents practical application. Field methods in general suffer considerable computational cost overheads due to the large number of data values involved in constructing the artificial field itself. A number of authors (Ge and Cui 2002; Kurihara *et al.* 2005) investigated the application of PF in dynamic environments containing multiple moving obstacles. The construction of local PFs took into account the relative velocity information between the robot and the obstacles detected. However, such repeated construction of local fields in response to the sensed changes in positions of the moving obstacles is computationally expensive.

Genetic algorithms are a class of adaptive search algorithms based on genetic and evolutionary principles. A genetic algorithm searches for one or more solutions by modifying a population of candidate solutions through the application of artificial genetic operators. To generate optimal solutions, GAs use only the fitness of individuals in the population and do not require gradient information or other internal

knowledge of the problem to be solved. A more detailed discussion of GAs is provided in chapter 2 and here only work related to path planning found in the literature is discussed. This section gives only a general description of the relevant work and the more detailed discussion with respect to recent developments in areas related to the specific topics of the work in this thesis is provided in the relevant chapter.

GAs have been employed in path planning problems largely due to the advantages discussed above and in section 1.3. However, in order to improve search performance, most work has employed GAs tailored to the specific problem and, in path planning, the representation strategy and genetic operators are commonly adapted.

Sugihara and Smith (1997) used the simplest representation, namely binary with fixed length, in their planning algorithms. However, a fixed length representation is not sufficiently flexible when representing paths with a variable number of intermediate nodes and so variable length representations have been generally adopted. Nearchou (1999) used a binary string of variable length to represent a sequence of actions to perform the movement between adjacent cells and a similar scheme was adopted by Tu and Yang (2003) with each gene containing four binary bits, three representing the direction and the fourth the distance the robot will move during the next step. Floating point representations have been adopted by a number of authors (Chen and Xu 2005; Elshamli, Abdullah and Areibi 2004; Hu and Yang 2004; Nikolos *et al.* 2003; Trojanowski, Michalewicz and Xiao 1997; Xiao 1997; Xiao *et al.* 1997; Xiao, Michalewicz and Zhang 1996; Zheng, Ding and Zhou 2003; Zheng *et al.* 2005). Chen and Xu (2005) and Geisler and Manikas (2002) used floating point chromosomes which were of equal length. A tree representation was used in the genetic-based path planning algorithm proposed by Hocaoglu and Sanderson (2001).

The genetic operators used in path planning have differed considerably between authors. The one-point crossover has been commonly used without significant modification (Elshamli, Abdullah and Areibi 2004; Geisler and Manikas 2002; Hermanu *et al.* 2004; Hocaoglu and Sanderson 2001; Hu and Yang 2004;

Michalewicz and Zhang 1996; Nikolos *et al.* 2003; Sedighi *et al.* 2004; Smierzchalski and Michalewicz 2006; Sugihara and Smith 1997; Trojanowski, Michalewicz and Xiao 1997; Xiao 1997; Xiao *et al.* 1997; Zheng *et al.* 2005). Other works adopted the two-point crossover operator (Davies and Lingras 2003; Nearchou 1999) or uniform crossover operator (Tu and Yang 2003). Although the standard mutation operator has been commonly used, a range of asexual operators have been specifically designed in order to affect the candidate path to become feasible, smooth and safe. Randomly-generated initial populations are very likely to contain infeasible paths (intersecting with one or more obstacles) and conversion to a feasible path is difficult to achieve using standard operators. Many new operators, such as repair and insert, were developed to enable rapid transformation from infeasible to feasible paths (Elshamli, Abdullah and Areibi 2004; Hocaoglu and Sanderson 2001; Hu and Yang 2004; Smierzchalski and Michalewicz 2006; Trojanowski, Michalewicz and Xiao 1997; Xiao *et al.* 1997). In order to minimise the presence of sharp turns, specific operators were proposed to swap, insert or delete notes by Elshamli, Abdullah and Areibi (2004), Hu and Yang (2004), Michalewicz and Zhang (1996), Smierzchalski and Michalewicz (2006), Trojanowski, Michalewicz and Xiao (1997), Xiao *et al.* (1997), Zheng, Ding and Zhou (2003), and Zheng *et al.* (2005). Other publications (Smierzchalski and Michalewicz 2006; Trojanowski, Michalewicz and Xiao 1997; Xiao *et al.* 1997) described genetic operators that aid the robot in its avoidance of obstacles. In general, while these operators incorporate problem-specific knowledge to bias the search, their disadvantages are that they may trap the search into local minima and that, compared with conventional operators, their implementation is rather complex and so time consuming.

The genetic-based approaches found in the literature operate in either discrete or continuous space (although, in practice, continuous space is also effectively represented in a discrete manner when implemented on digital computer). The distinction between the two types is normally based on whether the environment is divided into discrete cells. Algorithms that operate on discrete space are described by Chang, *et al.* (2005), Geisler and Manikas (2002), Hermanu *et al.* (2004), Hu and Yang (2004), Nearchou (1999), Nikolos *et al.* (2003), Sedighi *et al.* (2004), Soltani *et*

al. (2002), Sugihara and Smith (1997), and Tu and Yang (2003). The continuous planning approaches that search the entire space for solutions and so eliminate the need to construct a map were proposed by Elshamli, Abdullah and Areibi (2004), Fujisawa *et al.* (2000), Hocaoglu and Sanderson (2001), Michalewicz and Zhang (1996), Smierzchalski and Michalewicz (2006), Trojanowski, Michalewicz and Xiao (1997), Vannoy and Xiao (2004), Xiao (1997), Xiao *et al.* (1997), Zeng (2003), Zheng, Ding and Zhou (2003), and Zheng *et al.* (2005).

One of important findings from the literature review is that the evolutionary navigator/planner (EP/N) (Lin, Xiao and Michalewicz 1994; Trojanowski, Michalewicz and Xiao 1997; Xiao 1997; Xiao *et al.* 1997; Xiao, Michalewicz and Zhang 1996) is among the best-performing evolutionary planners and it has been the subject to extensive refinement and extensions as described in a series of frequently cited papers (Ashlock, Manikas and Ashenayi 2006; Buyurgan *et al.* 2007; Elshamli, Abdullah and Areibi 2004; Geisler and Manikas 2002; Hermanu *et al.* 2004; Hocaoglu and Sanderson 2001; Hu and Yang 2004; Nearchou 1999; Nelson *et al.* 2004; Patnaik and Karibasappa 2005; Sedighi *et al.* 2004; Tarokh 2007; Zheng, Ding and Zhou 2003; Zheng *et al.* 2005). Consequently, this thesis has used EP/N as the basis as a standard by which to compare new navigation approaches. Section 3.1.3 is dedicated to a detailed description of EP/N. For dynamic environments, a modified version of EP/N, termed 9EP/N++ (Smierzchalski and Michalewicz 2000 and 2006), was proposed as a decision support system for a ship to voyage without collision on the basis of environmental information obtained from automatic radar plotting aids. This planner, which is closely related to the work presented in chapter 5 (an extension to the work described in chapter 4), is discussed further in section 3.1.4. Further justification on these choices for comparative studies is provided in chapters 4 and 5.

3.1.3 Evolutionary Planner/Navigator

EP/N uses the same algorithm for both off-line planning and on-line navigation, incorporates a problem-specific chromosome structure and specifies a number of application-specific operators. Only a brief overview of EP/N is given here; full details of the EP/N algorithm can be found in the works by Lin, Xiao and

Michalewicz (1994), Trojanowski, Michalewicz and Xiao (1997), Xiao (1997), Xiao *et al.* (1997), and Xiao, Michalewicz and Zhang (1996).

The EP/N algorithm is shown in Figure 3.1. A path is represented by a series of nodes. In the initial generation, each chromosome is produced by randomly choosing intermediate nodes within the search space (environment), apart from the first and last nodes, which indicate the specified start and goal points respectively. If the terminating condition is not met, an operator is selected from the eight candidates according to a given probability and a roulette wheel is then used to choose a single parent (or a pair of parents if the crossover operator is selected) based on the ranks of the individuals. The resulting offspring replace the worst individuals in the current population to form the new generation. In order to minimise the calculation time of each generation, a steady-state evolutionary algorithm is adopted in EP/N, that is, each generation involves only a single application of one operator. The evolutionary process terminates after a number of generations determined by the user or dynamically by the algorithm, and the selected individual is the one describing the best path found. Storing the results of previous navigation tasks can improve the efficiency of later planning in EP/N. If *a priori* knowledge exists, the initial population is generated from the paths stored for previous tasks instead of performing random initialisation.

```
procedure EP/N planning algorithm
  begin
    if there exists a previous population with relevant paths then
      input the previous population P
    else
      initialise P
    end if
      evaluate P
    while the termination condition is not reached do
      use the operator probabilities to select an operator O
      select parent(s) for the operator
      produce offspring by applying operator O to selected parents(s)
      evaluate the new offspring
      replace the worst member(s) of the population P by new offspring
      select the best individual p from P
      every n^th step
        if algorithm is operating online manner and p is feasible then
          move one step along path p while sensing environment
          modify the values in all individuals to a new starting position
          if there is any change needed to the existing plan then
            update the object map
          end if
          evaluate P
        end if
      end every
    end while
  end
end procedure
```

Figure 3.1 Pseudocode for the EP/N algorithm.

There are two working modes in the EP/N system, namely off-line and on-line. In off-line mode, paths are determined based only on information about known obstacles in the environment. In on-line mode, the algorithm navigates the robot along the current best known path found while monitoring for unknown objects. Taking the robot's current position into account, as well as any newly-detected obstacles, EP/N evaluates the best path obtained by the evolutionary process, and, if better, will use it to replace the path currently being followed.

3.1.4 Evolutionary navigator ϑEP/N++

ϑEP/N++ extends EP/N to deal with the problem of avoiding collisions, particularly with moving ships at sea from the perspective of an evolutionary process. This is achieved by introducing a number of parameters, including time, the variable speed of the ship and time-varying constraints on other ships in the vicinity. The fixed navigation constraints (such as land, canals and shallow waters) are represented approximately by a number of polygons, both convex and concave. Moving ships are

modelled as hexagons whose shapes are defined by considering safe distance, speed ratio, and bearing. As the length of the own-ship is small with respect to the maximum length of the areas representing moving ships, the dimensions of the own-ship are ignored in the planning process. The evolutionary algorithm shown in Figure 3.2 is applied for both off-line and on-line planning. In off-line planning stages, all parameters reflecting the motion of the moving ships are assumed to be constant. With the support of automatic radar plotting aids (ARPA), the values of all parameters are monitored on a continuous basis. The on-line planning is performed in response to any observed changes in the motion of other ships with the current trajectory of the own-ship being corrected by the evolutionary process.

```
procedure 9EP/N++ planning algorithm
  begin
    number of generations g = 0;
    input operation parameters;
    input environmental information from sensors;
    initialise the population P(g);
    construct dynamic obstacles;
    evaluate population P(g);
    while (not termination condition) do
      increment the number of generations g = g + 1;
      randomly select operator Oj;
      select parents from P(g);
      apply the operator Oj to produce offspring;
      build dynamic obstacles;
      evaluate population P(g);
      replace worst member in population by offspring;
      select the best individual p from P(g);
    end while
  end
end procedure
```

Figure 3.2 Pseudocode for the 9EP/N++ algorithm.

9EP/N++ preserves the steady-state structure of EP/N and only adds one operator (to modify the speed of the own-ship from a set of discrete speeds available) to the eight inherited from EP/N. A chromosome represents a path containing a series of nodes with additional bits indicting the feasibility and speed value for each path segment. The evolutionary process starts by assigning control parameters and the initial population is randomly generated, then one of nine genetic operators is randomly selected and applied to the individual (or a pair of individuals if crossover operator is chosen) selected by a roulette wheel based on its rank (or their ranks) in the

population. The best individual is selected as the current trajectory after the evolutionary process has run for a specified number of generations, or after evolution has stalled.

3.2 Reactive navigation systems

Active research on the reactive paradigm began in the 1980s with the introduction of the subsumption architecture (Brooks 1985), the seminal work on the behaviour-based approach. In reactive navigation, the robot generates emergent behaviours through interactions between its primitive behaviours, implemented in a number of separate layers, and its environment. Sensory information is shared among the layers and the responses of the various behaviours are selected or fused in some manner to produce a response. Such coordination of behaviour is accomplished under the control of competitive or cooperative rules. In the case of competitive behaviours, only one will access the robot's actuators at a particular instance, whereas cooperative behaviours will combine to establish an action. Although some high-level deliberative behaviour may be incorporated, such as mapping, the response to perceived stimuli is generally accomplished through only the trained behaviours. The main features of reactive architectures were introduced in section 1.2.1. In brief, reactive navigation systems have their sensors and motors directly linked through a control scheme that is embedded in an array of simple perception-action stimuli (Nolfi and Floreano 2000; Mali 2002; Muñoz-Salinas *et al.* 2005; Murphy 2000; Orebäck and Christensen 2003; Stoytchev and Arkin 2004; Urdiales *et al.* 2003b). In contrast to the planner-based navigation systems surveyed in section 3.1, no internal representation of the external world is generally required, but onboard sensors are necessary. Reactive systems are able to provide robust and real-time navigation in dynamic environments where unpredictable events may occur frequently; however, researchers working on this type of system have often received criticism for rejecting the importance of world model.

3.2.1 Brief survey of reactive approaches

There are various reactive navigation systems proposed in the literature and some examples are provided here. The reactive approaches can be generally classified into two categories according to whether training is carried out.

The most behaviour-based systems belong to the category where the system is trained, as this is normally required to facilitate a set of high-performance behaviours (even though some behaviours may exist from the outset). Fuzzy-logic controllers (for example Hagras, Callaghan and Colley 2004; Malhotra and Sarkar 2005; Nefti *et al.* 2001; Zhu and Yang 2004) and neural networks (for example Low, Leow and Ang Jr 2003; Kubota 2004; Min 2005; Zalama *et al.* 2002) have been adopted in behaviour-based systems to mimic a set of reactive behaviours. In the fuzzy-logic systems, the parameters of the controllers need to be optimised by other techniques, such as GAs (Rajapakse, Furuta and Kondo 2002). GAs have also employed in the selection of the most suitable artificial neural network controller (Leon, Tosini and Acosta 2004). The reactive navigation achieved by case-based reasoning (CBR) (for example Kira and Arkin 2004; Urdiales *et al.* 2003a, 2003b and 2006) and decision-tree based techniques (for example Cocora *et al.* 2006; Shah-Hamzei and Mulvaney 2000; Swere, Mulvaney and Sillitoe 2004) also require a training process to generate a set of control rules. CBR techniques create a set of cases for the situations encountered during the training stage and expand the case base by including new cases discovered during actual navigation. The closest match of the current perception to that found in the case base is used to retrieve the most appropriate action. In decision-tree based navigation, an appropriate tree can be induced from training data with a number of attributes and the tree is then used to decompose a set of control rules that directly maps the stimuli to the corresponding actions. For these training-based approaches, a suitable learning technique is often desirable in order to reduce both the memory usage and the training period while maintaining a satisfactory quality of navigation performance. Earlier work conducted by the Electronic System Design Group at Loughborough aimed to overcome these drawbacks in a decision-tree based system that was adopted in the work presented in this thesis and this implementation is described in more detail in section 3.2.2. The main reason for using this method in the

work reported in the thesis is that it is fully available both in source code and as an executable.

The reactive approaches introduced below do not require a period of training. Some reactive systems have used potential field based approaches (PF) (for example Ge and Cui 2002; Kurihara *et al.* 2005; Zelek 1999), in which only a local artificial field is constructed from obstacles observed within the range of the sensors. Local navigation is achieved by the force generated from the local field, but the repetitive construction of the local field is time consuming. The nearness diagram (ND) navigation method (Minguez and Montano 2004 and 2005; Minguez, Montesano and Montano 2004; Minguez, Osuna and Montano 2004; Montesano, Minguez and Montano 2005) determines an avoidance command by the control law associated with the configuration (selected from the five pre-defined configurations) that is the most similar to that perceived. A detailed description of the ND approach will be given in chapter 7 due to its relevance. A number of reactive approaches, namely dynamic windows (DW) (Fox, Burgard and Thrun 1997; Ogren and Leonard 2002 and 2005; Stachniss and Burgard 2002), velocity obstacles (VO) (Fiorini and Shiller 1998; Large, Laugier and Shiller 2005) and vector field histogram (VFH) (Borenstein and Koren 1991; Ulrich and Borenstein 1998 and 2000) were designed specifically for avoiding moving obstacles. The DW approach constructs a space (window) of velocities achievable by the robot taking into account specific constraints and potential collisions that may occur within a defined time frame. A suitable velocity is then selected from the velocity space using an evaluation function. VO-based methods construct velocity obstacles that represent the set of robot velocities that would give rise to collision with a moving obstacle and then, using certain optimisation criteria, select an avoidance velocity not present in the VOs. The VFH approach uses an occupancy grid map that is generated by and updated from sensor information. The grid map is converted into a histogram representing the free space available to the robot and the motion direction and velocity of the robot are determined based on the histogram. Such a mechanism is in sharp contrast to training-based reactive approaches which do not have explicit reasoning in determining instantaneous responses.

The advantages and disadvantages of these reactive approaches are further discussed in section 7.1 when the dynamic avoidance approach proposed in the thesis is introduced. Note that although the decision-tree based technique has been adopted in the hybrid navigation systems presented in chapters 6, 7 and 8, other reactive methods, such as fuzzy logic, neural networks, and PF, can be used without modification of the remainder of the hybrid systems.

3.2.2 Decision-tree based reactive system

In the research described in this thesis, the reactive control is realised by suitable learning of a decision tree (DT); an approach adopted in earlier work at Loughborough (Mulvaney *et al.* 2005; Swere, Mulvaney and Sillitoe 2004). DT learning has been applied to mobile robot navigation by a number of authors. Sillitoe *et al.* (2001) analysed echoes received by an array of sonar sensors to train a DT to classify the contours of obstacles, so enabling the identification of specific object types for use in reactive navigation. Shah-Hamzei and Mulvaney (2000) trained DTs in an off-line manner to allow a robot to learn new reactive behaviours. The DTs generated were used to synthesise appropriate control rules to navigate a robot in unseen environments.

The reactive system used here is a novel frequency-table based learning technique (Mulvaney *et al.* 2005; Swere, Mulvaney and Sillitoe 2004) that was originally developed to overcome memory and calculation time limitations of incremental DT methods, such as ID5 (Utgoff 1988), ID5R (Utgoff 1989) and ITI (Utgoff, Berkman and Clouse 1997). Such as solution was chosen as a mobile robot often requires that the navigation system is able to run in limited memory and in many cases need to learn incrementally. In the current work, the frequency table is incrementally learned from initially random movements of the robot that were rewarded by being entered into the table if the movement resulted in the robot moving nearer to a goal. The entries in the frequency table can be used to provide data for use by entropy-based learning methods (Swere, Mulvaney and Sillitoe 2004), but in the current work an ID3-based DT (Quinlan 1983) is generated for navigation purposes. This form of

representation allows for a 'lazy learning' approach in which the DT learning is only applied when required (Aha 1997). For example, in order to reduce the computational requirements of a real-time application, the DT could be re-computed whenever new feature vectors are acquired, or when it is recognised that the current tree no longer correctly classifies the data.

The formulation of the frequency table can be described as follows. Let the feature vectors in the training set each be of the form $[f_1, f_2, \ldots, f_i, \ldots, f_n, c]$, where the set of attribute values for feature f_i is given by $\{a_{i1}, a_{i2}, \ldots, a_{iu}, \ldots, a_{iw}\}$ and the class c is taken from the set of available class values $[c_1, c_2, \ldots, c_j, \ldots, c_m]$,. The frequency table can then be considered to be composed of n separate two-dimensional tables each of whose mw_i rows form the attribute values of feature i for all classes and whose columns are the attribute values of all features other than i. When a training vector is acquired, its attribute values determine the element that is to be incremented in each of the two-dimensional tables. Figure 3.3 shows an example of the type of frequency table used for a mobile robot with four range sensors equally spaced circumferentially. The frequency table is shown for a one of the four sensors, where the rows are the attribute values for the sensor and the columns are the attribute values for features other than sensor 1. When a new feature vector is acquired, the feature values dictate which particular entry in the table is incremented. The frequency tables for the other features (angle to goal and sensors 2, 3 and 4) are similarly updated on receipt of a new feature vector.

class (new heading)	sensor 1	number of examples	angle to goal				sensor 2		sensor 3		sensor 4	
			zero	small	med	large	near	far	near	far	near	far
forward	near	0	0	0	0	0	0	0	0	0	0	0
	far	206	0	87	49	70	206	0	206	0	206	0
right	near	213	0	61	132	20	213	0	213	0	62	151
	far	109	0	54	12	43	88	21	109	0	0	109
left	near	64	1	44	14	5	38	26	64	0	59	5
	far	81	0	57	22	2	0	81	81	0	54	27
reverse	near	1	1	0	0	0	1	0	1	0	1	0
	far	0	0	0	0	0	0	0	0	0	0	0

Figure 3.3 An example of a frequency table used in the reactive navigation of a mobile robot.

3.3 Hybrid navigation systems

In practical applications, there are many situations where a complete model of the environment is initially unavailable or is only partially available due to unpredictable changes. To build a deliberative model, a number of reactive navigation movements, either designed or as part of exploration activities, can be specified through such an environment. Consequently, such navigation problems require a hybrid solution to compensate for the individual drawbacks of reactive and deliberative paradigms and also to adapt to unexpected changes within dynamic environments. Nevertheless, an extensive world model is normally computationally expensive to construct and any representation of the environment must be compact to meet memory constraints of a practical application.

Arkin (1998) described the findings of an in-depth survey of earlier attempts to design hybrid systems. The survey classified hybrid systems according to the form of interface strategy adopted, namely selection, advising, adaptation, and postponing. Planning is viewed as a hybrid system that employs a selection strategy, which determines the behavioural composition and parameters used during execution. The autonomous robot architecture (AuRA) (Arkin 1986 and 1987) is a representative hybrid architecture of this type. This architecture contains two distinct components: a deliberative hierarchical planner and a reactive controller. In hybrid systems using an advisory interface strategy, the planer offers advice on the course of action to be taken by the reactive layer, which is then at liberty to ignore the advice. A typical example of this type is that described by Atlantis (Gat 1991a and 1991b). Planner-reactor (Lyons and Hendriks 1992 and 1995) used an adaptation strategy to integrate the deliberative and reactive components, such that the planner continuously alters the reactive model according to the changes made within the environment and to task requirements. A more effective course of action within hybrid architectures can, in some cases, be achieved by a postponing strategy that defers planning until it is required, rather than generating an overarching unchangeable single plan. An example implementation of the postponing strategy is the procedural reasoning system (PRS) proposed by Georgeff and Lansky (Georgeoff and Lansky 1987). From

the structural point of view, one common feature found in a number of hybrid designs, such as Atlantis (Gat 1991a and 1991b), SSS (Connell 1992), and 3T (Bonasso *et al.* 1997), is that three major modules comprise the hybrid system. The three modules embedded in those three hybrid architectures roughly equate to a reactive feedback control module, a deliberative planner, and a sequencing module that controls the operations of the first two modules. A comparison with the earlier three-layered architectures was conducted by Gat (1998), revealing that having three major components in the architecture normally corresponds to the following three states: a current state (the reactive component), a state reflecting the past (the sequencing component), and a state predicting the future (the deliberative component).

Recent developments in hybrid navigation systems are introduced below and more detailed description can be found in sections 6.1 and 7.1 in their comparison with the hybrid architectures proposed in this thesis.

Recent hybrid architectures largely inherit the three-layer architecture of earlier designs, even though the three layers may not be arranged hierarchally, for example, the reflexive, reactive, and functional layers, in the nested-loop architecture (Santos, Castro and Ribeiro 2000) were arranged in a nested manner. Aguirre and González (2003), Liu, Hu and Gu (2006), Minguez and Montano (2005), Minguez, Montesano and Montano (2004), and Muñoz-Salinas *et al.* (2005) all described hierarchical implementations consisting of reactive and deliberative components connected through a middle layer. The hybrid architectures of Low, Leow and Ang Jr (2002 and 2003), Maaref and Barret (2002), and Wang, Yong and Ang Jr. (2002) contained directly coupled reactive and deliberative components. When the middle layer is absent, the local navigation is accomplished in the reactive layer without explicit intervention of the high-level deliberative layer. Urdiales *et al.* (2003b) proposed a hierarchical hybrid architecture with four layers in which two layers, *geometrical modelling* and *topological modelling*, construct and maintain the environmental information in two separate mapping approaches. The two mapping layers can be grouped into one, as they have the same function of representing environmental knowledge, though in different formats. In the four-layer architecture proposed by Li

et al. (2004), the additional layer, called the hardware abstraction, is related to the low-level control of sensors and actuators respectively. The low-level control is normally integrated in the reactive component in other hybrid systems as this component directly operates on the raw sensory information and outputs the final action.

Based on the review of recent hybrid architectures, the following points have been identified. A hybrid representation scheme has been frequently employed in navigation architectures. As discussed in section 3.1.1, topological representations can simplify the path planning task, thereby shortening the planning period, and metric information can be used to enhance the local navigation by eliminating the uncertainty due to identical landmarks. The experimental work carried out for most architectures were limited to indoor environments which tend to be well structured. Furthermore, only Santos, Castro and Ribeiro (2000) have discussed hybrid with other architectures, although no practical results were provided.

3.4 Conclusions

The survey of planner-based system included mapping methods and planning algorithms. Planners inspired by the evolutionary concept have been given particular emphasis, with two GA planners being described in detail as they are directly related to the work developed in this research. A short review of the reactive methods was given, followed by the introduction of the reactive system employed in the research. The final part of this survey focused on navigations systems that have a hybrid architecture. Where appropriate, the navigation approaches found in the literature will be discussed further in the chapter where the related navigation system developed in the current work is introduced.

To meet the navigation task requirement, a deliberative navigator will produce a plan based on knowledge of the environment. Re-planning is required to respond to unexpected and unmodelled changes that occur during the execution of the plan. Due to the long computation time involved in calculation of deliberative plans, these

approaches run the risk that collision may occur before re-planning is complete. Reactive methods, on the other hand, are well suited to tackle uncertainty and dynamic changes, but lack any means of predicting future events. The fact that the reactive approach ignores cognition, often limits reactive robots to mimicking simple forms (Arkin 1998). Hybrid systems hold the promise of taking advantage of the benefits of both deliberative and reactive approaches and consequently improving navigation performance by incorporating world models and being able to respond on the timescales dictated by dynamic environments.

In chapters 4 and 5, the research described in this thesis implements a deliberative planning method that enhances the performance of existing evolutionary planners (described in sections 3.1.3 and 3.1.4) by reducing the search to a small number of points in the environment. In chapters 6 to 8, a hybrid architecture is adopted that takes advantage of a novel abstract representation of the environment. This hybrid system is subsequently revised firstly by developing an avoidance behaviour to interact with moving objects and secondly by a generalised version that is able to navigate between any specified start and goal points.

Chapter 4

VERTEX PLANNER

This chapter describes the vertex planner developed with the aim of overcoming the principal drawbacks of the existing planning approaches described in section 3.1. This work was presented in IEEE International Conference on Cybernetics and Intelligent Systems (CIS 2006) and awarded 'best paper'.

4.1 Related work in path planning in static environments

Path planning can, as discussed in section 3.1.2, be viewed as an optimisation problem (such as shortest path, shortest travel time, minimum energy consumption, or some combined optimisation criteria) with certain constraints (such as collision-free). Genetic algorithms as an adaptive search technique have seen increasing application to optimisation problems due to their robustness, simple mechanism, no requirement for gradient information, operation in a range of parameter spaces (see section 1.3). These advantages were instrumental in the decision to adopt genetic algorithms for the planner in the current work. Consequently, this section only presents approaches based on the evolutionary concept and the examples of other methods can be found in chapter 3.

The genetic based approaches found in the literature can be classified into two categories according to the search space, discrete or continuous, on which the algorithms operate.

In the discrete search space, the grid method is probably the most common method of modelling given environments. In generating the grid, a suitable resolution is needed in order to deal with the range of obstacles likely to be encountered. For example, obstacles of more complex shape may require a representation of higher resolution in order to prevent a collision, whereas a high resolution of grid makes planning inefficient if only obstacles of simple shape are present in the environment. Furthermore, although shorter paths are more likely to result from higher resolution representations their determination requires more planning time. Consequently, an appropriate trade-off between path quality and planning time is difficult to achieve in practical situations. Three different genetic encoding mechanisms have been investigated. The simple binary genetic representation with fixed length was an early implementation by Sugihara and Smith (1997) that was adapted by Geisler and Manikas (2002) to use the standard GA. In this approach, the environment is modelled by a grid map of n rows (or columns), where the length of each chromosome is restricted to n genes. A real value corresponding to each gene represented the selected columns (or rows), whereas the rows (or columns) were indicated by the locus of each gene. However, such a genetic representation can be so biased that possible solutions may not be found even when they exist. In order to overcome this limitation, an efficient genotype structure was developed that incorporated an orientation bit which functioned to select either row-wise or the column-wise representation (Hermanu *et al.* 2004; Sedighi *et al.* 2004). An alternative solution that relaxes this restriction is to use the coordinates of the cells of each gene. Hu and Yang (2004) incorporated problem-specific knowledge into their GA in the form of designed genetic operators, such as *node_repair* (moving a node that is inside an obstacle to the outside) and *line_repair* (repairing an infeasible line segment by adding a new node) to improve efficiency. Note that the length of an individual in the GA was variable and a number of grids was selected as intermediate nodes. Nearchou (1999) developed an evolutionary-based planning algorithm that used a tailored set of genetic operators, in which each individual was a binary string of variable length representing a sequence of actions to achieve transitions between adjacent cells. A variable genetic representation was also used by Tu and Yang (2003) where each gene contained four binary bits, three of which represented the direction

and the fourth the distance the robot will move during the next step. However, it was later reported by Hu and Yang (2004) that many hours were needed to deliver a solution. Discrete space has also been used in planning solutions for 3D environments. For example, Nikolos *et al.* (2003) proposed an evolutionary-based path planner for unmanned aircraft, with each gene being the coordinates of a control point selected within the desired constrained space represented by a mesh. To generate a smooth path, the control points were used in the construction of a spline curve.

Most planning algorithms that search continuous space adopt a floating point representation scheme, their genetic operations are also conducted on a fraction of population in each generation, and, to accelerate evolution, problem-specific knowledge is often taken into account when designing the genetic operators. One of the most cited planning algorithms based on the evolutionary concept operating in continuous space, is the evolutionary planner/navigator (EP/N) (Lin, Xiao and Michalewicz 1994; Trojanowski, Michalewicz and Xiao 1997; Xiao 1997; Xiao *et al.* 1997; Xiao, Michalewicz and Zhang 1996). The absolute coordinates of any point in the environment can be encoded as the elementary information for a potential gene, obviating the need to establish a configuration space. This is an important advantage of the coordinate approach with respect to the grid approach, namely that an often arbitrary, or at best, difficult to establish, minimum resolution for the representation does not need to be determined. Eight genetic operators were developed by incorporating problem-specific domain knowledge and each of them is applied for each iteration (the steady-state genetic algorithm) according to operator probabilities. A more detailed description of EP/N was given in section 3.1.3. Hocaoglu and Sanderson (2001) described a evolutionary planning algorithm having a binary tree structure to represent a path whose intermediate vertices are determined by a modified Gram_Schmidt orthogonalization process. Their path planner used the steady-state genetic algorithm with one crossover and six mutation operators, where the mutation operators were similar in purpose to the genetic operators of EP/N, in that they provided a means to fix or repair an infeasible path, swap two nodes in a path, insert a new intermediate node, or fine-tune clearances. Based on EP/N, a GA planner (Elshamli, Abdullah and Areibi 2004) was developed with a set of genetic

operators modified for dynamic environments where initially unknown static obstacles became to be completely known during the planning. EP/N was modified in its application to a 3D manipulator for planning movements (Vannoy and Xiao 2004). Once a feasible trajectory had been generated, the robot followed a trajectory that was improved by evolution carried out during movement. Zheng, Ding and Zhou (2003) developed a route planner for an unmanned aircraft by considering multiple constraints, such as minimum route leg length, flying altitude, and maximum turning angle. A variable length representation was employed with each gene consisting of coordinate information for a selected point in the workspace together with a state bit to indicate feasibility of the path segment. A small number of members of the population was involved in the genetic reproduction, each generation being evolved using one crossover and six mutation operators. The planner was extended (Zheng *et al.* 2005) for route planning of multiple unmanned aircraft. Smierzchalski and Michalewicz (2000 and 2006) developed ϑEP/N++, an extension of EP/N applied to planning problems at sea consisting of multiple moving ships. ϑEP/N++ was introduced in section 3.1.4 and, due to its relevance to the current work, is further discussed in chapter 5.

A quantitative comparison between a conventional GA, Dijkstra's algorithm and A* search was conducted by Soltani *et al.* (2002) in their application to a material-delivery routing problem in a construction site represented by a multi-dimensional grid. The measures for comparison purpose were three single optimisation criteria, namely, path length, safety, and visibility, and, overall, the GA search algorithm performed the best in that less time was required to deliver the solution with a similar quality to those produced by the Dijkstra and A* methods. The efficiency of the GA algorithm was attributed to the smaller number of grid points visited by GA compared to those examined by the other methods. The experimental results also indicated that the GA is more competent for larger problems as the time complexities of both the Dijkstra and A* approaches increase significantly with problem size.

The modification of genotype structures and the inclusion of problem-specific knowledge both aim to shorten planning time. The decomposition of the continuous

environments into cells can reduce the search space (Hu and Yang 2004), yet the pre-calculation of a suitable cell representation (such as a grid map) is time consuming (Hocaoglu and Sanderson 2001; Xiao *et al.* 1997; Zheng *et al.* 2005). Furthermore, flexibility in allowed movements may be restricted if a discrete map used as a basis for planning; for example, the robot's turning circle may be constrained to ensure the locus includes the centres of adjacent cells.

Most of the above research seeks the shortest path as the optimisation goal, though occasionally optimisation criteria, such as safety, smooth trajectories, and restrictions on turning angles, have been used in planning algorithms (Nikolos *et al.* 2003; Smierzchalski and Michalewicz 2000 and 2006; Xiao *et al.* 1997). Energy saving has also been considered as an optimisation goal by a small number of authors (Garg and Kumar 2001; Huang, Xu and Liang 2005). The reduction of space to be searched while incurring little in pre-processing demands is the principal motivation in developing the vertex planner with the optimisation goal of minimising the path length. As stated by Hand *et al.* (2005) researchers do not base their work on navigation results found in standard environments. Although Hand *et al.* (2005) developed benchmarking program, this is available only for grid-based approaches and is unsuitable for methods that operate in other space, such as that used in this thesis.

In comparison with the map-based methods (as described in section 3.1.1 and above), the vertex planner requires only a simple vertex graph, thereby significantly alleviating the pre-processing demands. Whereas EP/N (Trojanowski, Michalewicz and Xiao 1997; Xiao 1997; Xiao *et al.* 1997) plans using the entire continuous space of the robot's environment, the vertex planner limits the search space to only the vertices of the obstacle. It is reasonable to assume that imposing such a constraint would result in a considerable shortening of the search process, this being the principal reason for proposing the vertex approach.

In the vertex planner, each gene represents a single obstacle vertex selected as an intermediate node and a variable length representation is used for the individuals to

minimise memory usage. In order to be able to compare the efficiency and effectiveness of the proposed algorithm with the EP/N system (Trojanowski, Michalewicz and Xiao 1997; Xiao 1997; Xiao *et al.* 1997), the current work has adopted a similar steady-state genetic algorithm, as described in section 2.1. As EP/N provides a relatively well-established solution, matured through a series of revisions, the underlying approaches taken by the EP/N researchers, continuous workspace and problem-specific knowledge, has been frequently adopted by other researchers (Elshamli, Abdullah and Areibi 2004; Zheng, Ding and Zhou 2003; Zheng *et al.* 2005), but without significant improvement. Additionally, the continuous approach adopted in EP/N makes its comparison with grid-based approaches difficult, due to the large number of different resolutions of grid map that a researcher can choose.

4.2 Vertex planning algorithm

The principal novelty of the vertex planner is that it does not need to search the entire continuous environment for a suitable path, but rather the GA acts directly on the vertices of the obstacles encoded in the genes. The pseudo-code for the vertex planning algorithm is shown in Figure 4.1 and the stages in its implementation are described below.

```
Procedure vertex planning algorithm
  begin
    t ← 0
    enlarge the obstacles
    encode the vertices of the obstacles
    initialise P(t)
    decode P(t)
    evaluate P(t)
    while (not terminating condition) do
      t ← t + 1
      select operator o_i with probability p_j
      select parent(s) from P(t)
      apply the operator o_j to produce offspring
      decode offspring
      evaluate new offspring
      replace worst member in P(t) by offspring
    end
    select the best individual p from P(t)
  end
end procedure
```

Figure 4.1 The pseudo-code for the vertex planner.

4.2.1 Enlargement of the obstacles

So as to allow the planner to regard the robot as a single point, the obstacles are enlarged by a value determined from the minimum distance that the robot can approach obstacles without collision, taking into account its physical dimensions. Figure 4.2 shows an example of the obstacle enlargement.

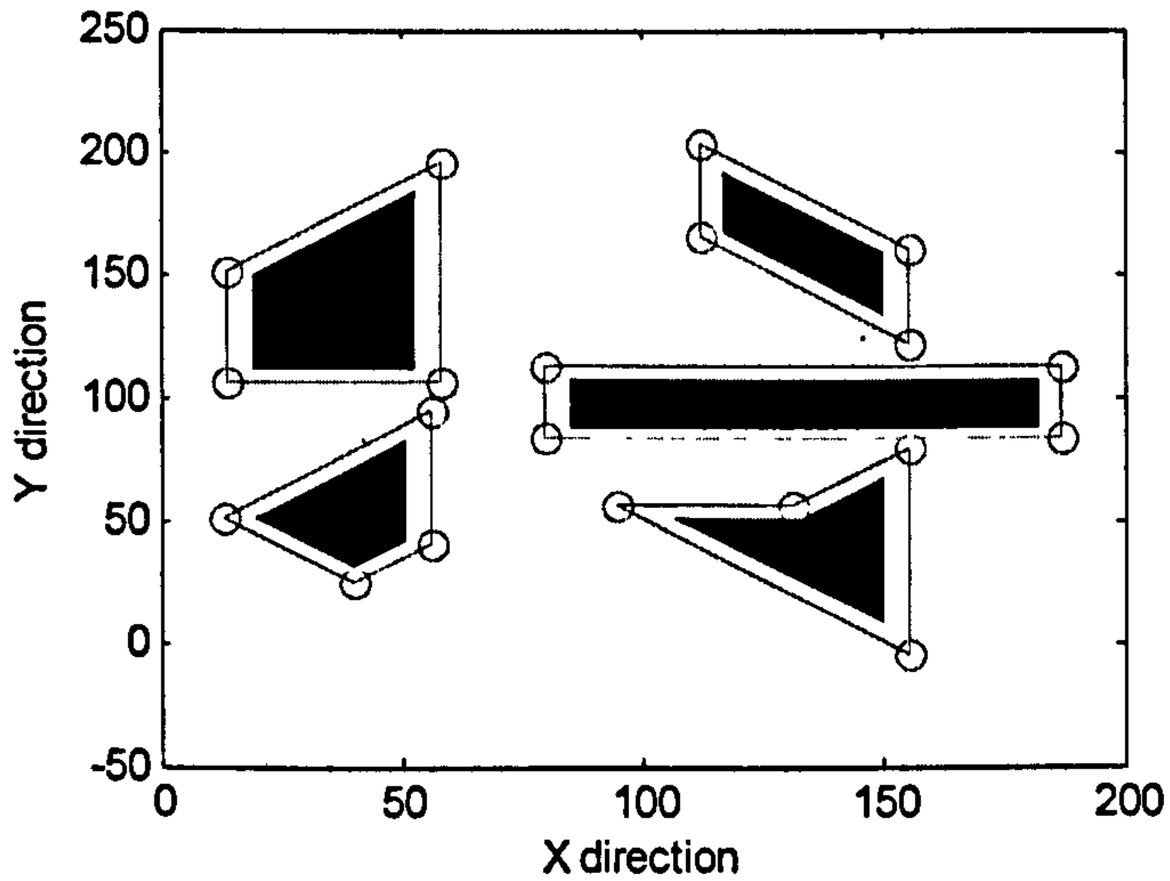


Figure 4.2 An example of the enlargement of the obstacles for the purposes of planning.

4.2.2 Encoding and decoding

The intermediate nodes of a path contain only vertices of enlarged obstacles and each gene contains a reference to a vertex, rather than holding its absolute coordinates. The vertices of the enlarged obstacles are randomly numbered and used as potential genes for a chromosome whose phenotype presents a path between two specified locations.

4.2.3 Genetic representation and initialisation

The structure of the chromosomes is illustrated in Figure 4.3. A chromosome contains a total number of genes l, whose minimum value is two (a path containing only the start and goal nodes) and whose maximum value is N+2, where N is the total number of vertices of all the obstacles in the environment. Consequently, each chromosome represents a path with a number of intermediate genes (nodes) in the range 0 to N between the start and goal points (that are also both included as genes in the chromosome). Each intermediate gene contains a reference, V_i, $i \in \{0, l\text{-}2\}$, to exactly one of the N vertices. A 'feasibility bit' for each gene is used to indicate whether the

path segment originating from the vertex referenced by the node is feasible. If the path segment connecting two consecutive vertices referenced in the chromosome (including from the start point to any vertex or from any vertex to the goal point) intersects one or more obstacles, then the feasibility bit of the first node of the path segment is assigned 1 to mark this segment as infeasible. If there is no intersection, 0 is used to indicate the segment is feasible.

start gene	gene 1		gene *l*-2	goal gene
0	V_1	...	V_{l-2}	N+1
0/1	0/1		0/1	0

Figure 4.3 The structure of a chromosome. The first gene in the chromosome refers to the start point (numbered 0) and the last gene refers to the goal point (numbered N+1). The l-2 intermediate genes refer to the vertices of obstacles in the environment. The lower row in the gene indicates the feasibility of the path segment that starts at that gene.

The initial population, $P(0)$, is generated by randomly choosing, for each chromosome, the number of genes and their vertex references. The vertex references are constrained so that they are not repeated within a chromosome, and that the start and goal points are always the first and last genes of a chromosome respectively.

4.2.4 Evaluation functions

The optimisation goal is to find a collision-free path with minimum path length. To achieve this goal, the feasibility of each chromosome is determined before the evaluation of path quality is carried out. The approach for feasibility examination is to check if there is any intersection between two straight lines (Pavlidid 1982). The generated chromosomes represent feasible paths if none of the individual path segments is infeasible, otherwise the chromosome is infeasible.

In the vertex planning method, the evaluation function, E_f, which is used to assess the quality of the feasible paths, is simply the length of the generated path as the intermediate nodes are selected from the vertices of the enlarged obstacles rather than being any point in the environment. E_f is given by

$$E_f = \sum_{i=0}^{l+1} d(V_i, V_{i+1}) \qquad \text{Equation 4.1}$$

where $d(V_i,V_{i+1})$ denotes the distance between the pair of vertices. Such design favours shorter paths over longer ones. The evaluation function E_i indicates how deeply an infeasible path segment intersects with an obstacle and is given by

$$E_i = \mu + \eta \qquad \text{Equation 4.2}$$

where μ denotes the number of obstacle intersections along the entire path and η is the mean number of intersections per infeasible segment. The reason for setting this function is to qualify and rank the infeasible paths. If less effort is needed to correct an infeasible path, a lower cost will be assigned. Given the two evaluation functions, the aim of the optimisation process of the vertex planning method is to minimise the values of E_f and E_i for their respective populations.

When the population contains both feasible and infeasible paths, the infeasible paths are all assumed to be worse than the worst feasible path.

4.2.5 Operator selection

The two standard genetic operators, *crossover* and *mutation*, and an additional operator, *repair*, are utilised in the algorithm. Only one of these operators o_j is needed in each steady-state generation and its selection is based on a roulette method whose slots widths are in proportion to the probabilities of the operators.

4.2.6 Application of the operators

The crossover operation is performed by a conventional one-point operator, where the crossover points are randomly selected and the parts that follow the crossover points of the two parent individuals are swapped. This operator is the simplest form for exchanging information between two selected individuals in order to generate better offspring. A faster convergence may result from having two or more crossover points during the first evolutionary phase (see section 4.2) from infeasible to feasible paths. However, when a two or more point crossover operator is applied between two feasible paths, the number of new path segments is greater than that produced by single point crossover, thereby increasing the probability that infeasible paths result. Note that the use of a uniform crossover operator is not suitable as it has the same

disruptive effect on the feasible paths as two or more point crossover. Also, as the search space consists of a discrete set of obstacle vertices, the arithmetic crossover operator cannot be applied. The mutation operation mutates the genes with small probabilities. The role of mutation is to prevent premature convergence and promote population diversity by introducing a small perturbation to the gene values, but too high a rate of mutation will result in a random search. The repair operator adjusts a randomly selected infeasible segment of an infeasible path, so that it circumnavigates the obstacles previously intersected. Figure 4.4 illustrates the repair operation. This genetic operator utilises problem-specific knowledge in order to produce efficiency during the evolutionary process that alters an infeasible path into feasible one.

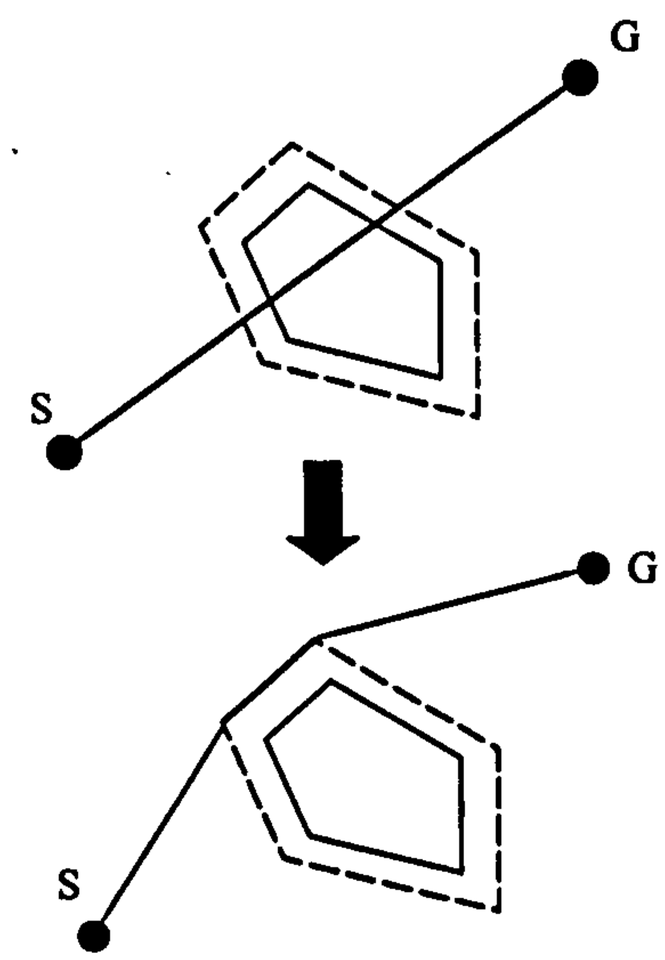


Figure 4.4 An illustration of the application of the repair operator that uses the vertices of the enlarged obstacle (shown by a broken line) to determine a feasible path around an obstacle.

4.2.7 Evaluation

Before evaluating the offspring, decoding of the genotype to its corresponding phenotype is required. In order to retain a constant selection differential, the individuals are sorted according to their fitness and a quadratic ranking scheme (De Jong 1992) (see section 2.3) is used to determine the number of offspring. The worst individual in terms of fitness is discarded, while the remainder and any new offspring created form the next generation. The evolutionary process terminates if no improvement is observed in the fitness of the best individual for a specified number

of generations, or if a user-defined number of generations is exceeded. When this occurs, the best individual is selected as the path planning solution.

4.3 Experiments and Results

The proposed planning algorithm was evaluated in four simulated environments, as shown in Figure 4.5, using version 6.5 of MATLAB (Mathworks 2006) running under Windows 2000 on a 2.8GHz Pentium P4. The first environment contains a number of obstacles of irregular shape to simulate unstructured environments. The second environment contains a similar number of obstacles to the first, but contains more vertices providing increasingly irregular obstacles. The third environment contains a relatively large proportion of infeasible space, and could be considered to represent walkways through a series of manufacturing cells or along corridors in an office building. The fourth simulates an open-plan office workspace. The relative complexities of the test environments are reflected by the total number of obstacle vertices, which doubles in each successive environment; the values being 20, 40, 80, and 160 vertices in environments 1 to 4 respectively.

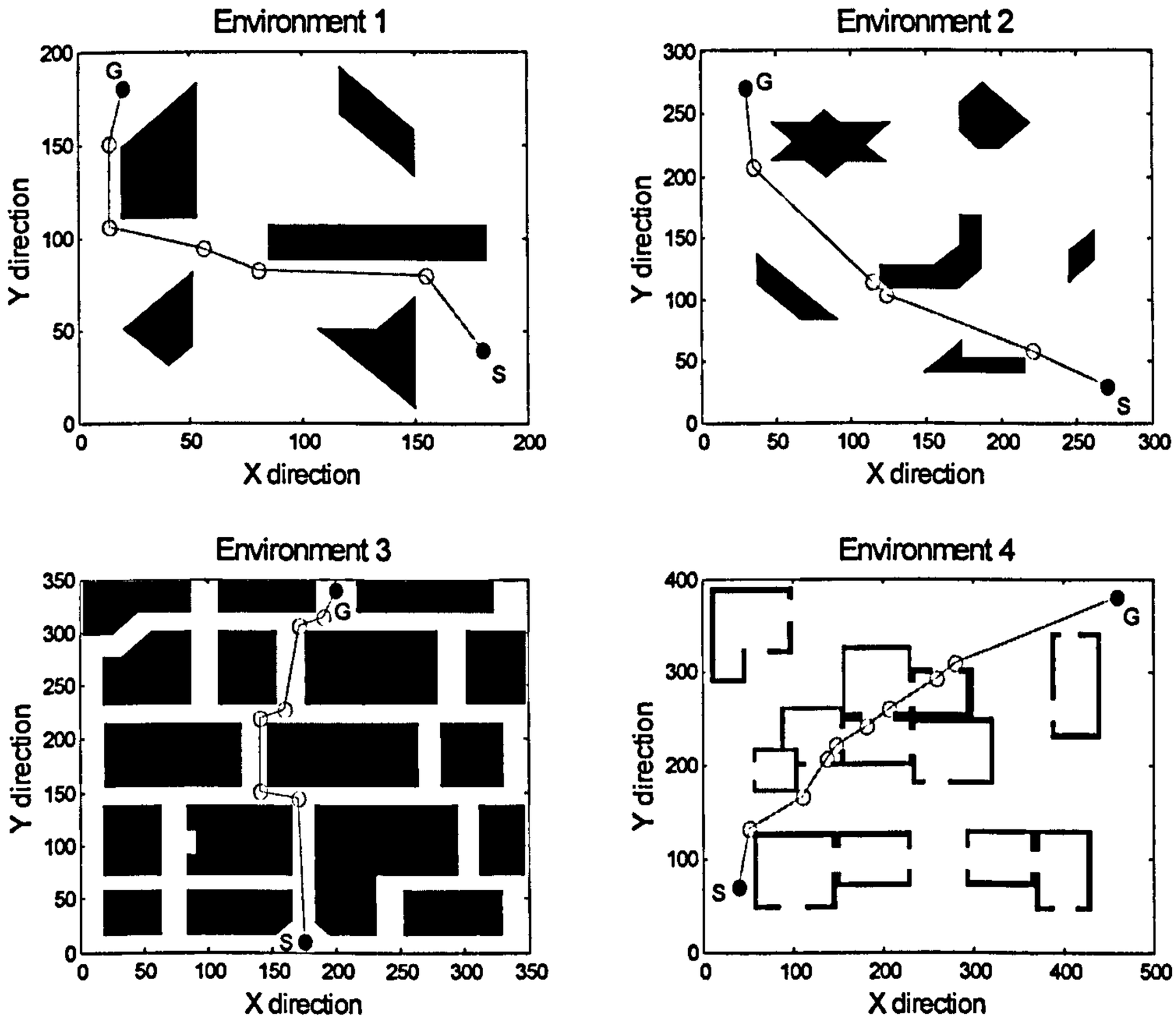


Figure 4.5 The paths generated by the vertex planner for each of the simulated environments. The circular markers show the generated intermediate nodes.

EP/N (Xiao 1997; Xiao *et al.* 1997) was used in order to provide a comparative test of the performance of the vertex planner. EP/N is a relative mature evolutionary planner having been refined through a series of revisions (Lin, Xiao and Michalewicz 1994; Trojanowski, Michalewicz and Xiao 1997; Xiao 1997; Xiao *et al.* 1997; Xiao, Michalewicz and Zhang 1996) and has frequently been cited by other researchers (Ashlock, Manikas and Ashenayi 2006; Buyurgan *et al.* 2007; Elshamli, Abdullah and Areibi 2004; Geisler and Manikas 2002; Hermanu *et al.* 2004; Hocaoglu and Sanderson 2001; Hu and Yang 2004; Nearchou 1999; Nelson *et al.* 2004; Patnaik and Karibasappa 2005; Sedighi *et al.* 2004; Tarokh 2007; Zheng, Ding and Zhou 2003; Zheng *et al.* 2005). Furthermore, due to the difficulty in justifying a specific grid resolution, the most appropriate comparison of the vertex planner is with another continuous planner. As far as possible, the system parameters of the vertex planner were defined so as to mimic those adopted by Xiao (1997) for EP/N. To facilitate direct comparison, these parameters are maintained for all experiments in the four

different environments, and are listed in Tables 4.1 and 4.2 for EP/N and the vertex planner respectively. For both planners, each generation contained 30 individuals and evolution was terminated when there was no further improvement in the fitness of the best individual over 300 generations. For both EP/N and the vertex planner, the maximum length of an individual in the initial generation was limited to be the sum of the number of vertices, the start point and the goal point in the environment under test and the minimum length was set to be two, this being the start and goal points only. When *mutation_2* is selected as a genetic operator for EP/N, only one intermediate node will be affected.

Table 4.1 System parameters for EP/N

operator probability								weights			rate			
crossover	*mutation_1*	*mutation_2*	*insert_delete*	*delete*	*swap*	*smooth*	*repair*	w_d(path length)	w_s (smoothness)	w_c(clearance)	*mutation_1*	*insert_delete*	*delete feasible node*	*delete infeasible node*
0.6	0.8	0.5	0.5	0.5	0.5	0.9	0.8	1	0	0	0.3	0.6	0.1	0.3

Table 4.2 System parameters for the vertex planner

operator probability			mutation rate	safe distance
crossover	*mutation*	*repair*		
0.6	0.5	0.8	0.3	5

The paths generated by the vertex planner for the four environments are shown in Figure 4.5. Note that under visual inspection, little discernible difference was apparent between the paths produced by the two planners.

To evaluate the efficiency of the vertex planner, the following were recorded: (a) the execution time to obtain the first feasible path; (b) the number of generations required to determine the first feasible path; (c) the execution time to obtain the final path; (d) the number of generations required to determine the final path and (e) the length of

the final path. The final path is deemed to have been reached when E_f remains unaltered for 300 consecutive runs.

Figure 4.6 shows the calculation time for the first evolutionary phase, namely that during which all the individuals in the population represent infeasible paths. Note that the plotting of the curves terminates as soon as one individual in the population is feasible. Table 4.3 shows both the execution times and the number of generations required to determine the first feasible path. The results in both Figure 4.6 and Table 4.3 are median values obtain over 50 runs.

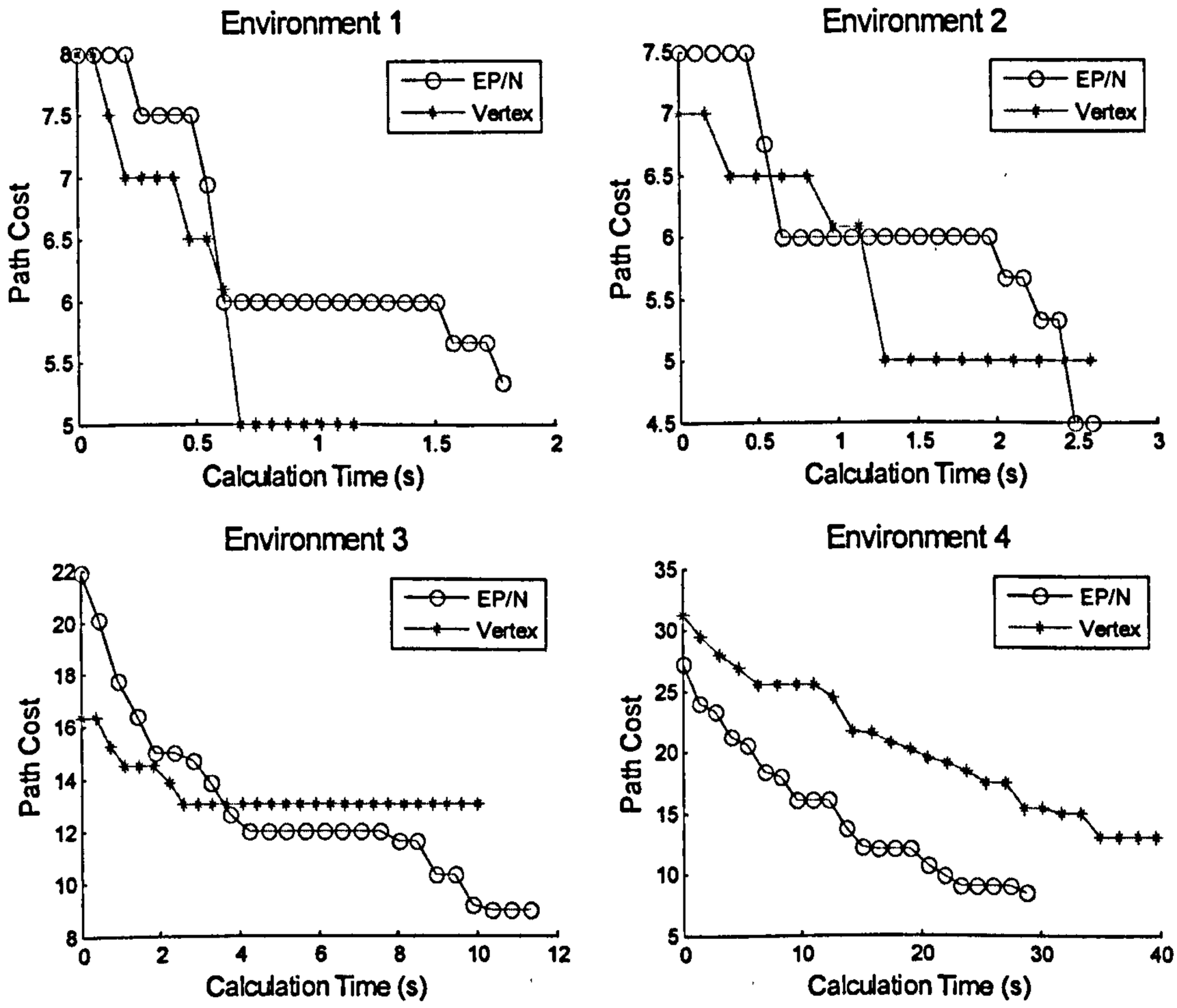


Figure 4.6 The infeasible path costs for the best individual from the first generation to that when the first feasible path is generated. The results are the medians obtained over 50 runs.

Table 4.3 Execution time and number of generations to determine the first feasible path. The results are the medians obtained over 50 runs.

environment	planning method	execution time to obtain the first feasible path (s)	number of generations to determine the first feasible path
1	EP/N	1.78	26
	Vertex	1.15	17
2	EP/N	2.59	23
	Vertex	2.58	16
3	EP/N	11.6	49
	Vertex	9.99	27
4	EP/N	28.8	63
	Vertex	39.7	50

Although the absolute probabilities of the three genetic operators in the vertex planner are the same as those of the corresponding operators in EP/N, as the vertex planner has fewer operators (and only one can be applied in each generation), each of its three operators are likely to be applied more frequently. This includes the repair operator which, from manual inspection of the progress of the development of the individuals, appears to be responsible for the relatively rapid convergence of the vertex planner in the first evolutionary phase.

The results for the second evolutionary phase, during which the aim is to translate feasible paths to high-quality feasible paths, are shown in Figure 4.7 and Table 4.4. Compared with EP/N, the vertex planner shows a significant improvement in execution time performance during the second evolutionary phase, with reductions in the range 20% (environment 4) to 78% (environment 1). A number of the EP/N operators are computationally more expensive to execute than those in the vertex planner, for example *insert*, *delete* and *smooth* can add new nodes to the selected path, and such enlarged individuals will subsequently require additional time to process. In contrast, the vertex planner naturally limits the maximum possible number of intermediate nodes in each individual path to the total number of vertices in the environment. Consequently, compared with the reductions in execution time, the corresponding reductions in the number of generations required by the vertex planner are not so great, but are significant nonetheless.

The effectiveness of the vertex planner can be seen in the path length results shown in the final column of Table 4.4, where both planners produce similar results in terms of path quality for environments 1 and 3. A marginally better path was found by the vertex planner in environment 2, but a more significant improvement was apparent in environment 4.

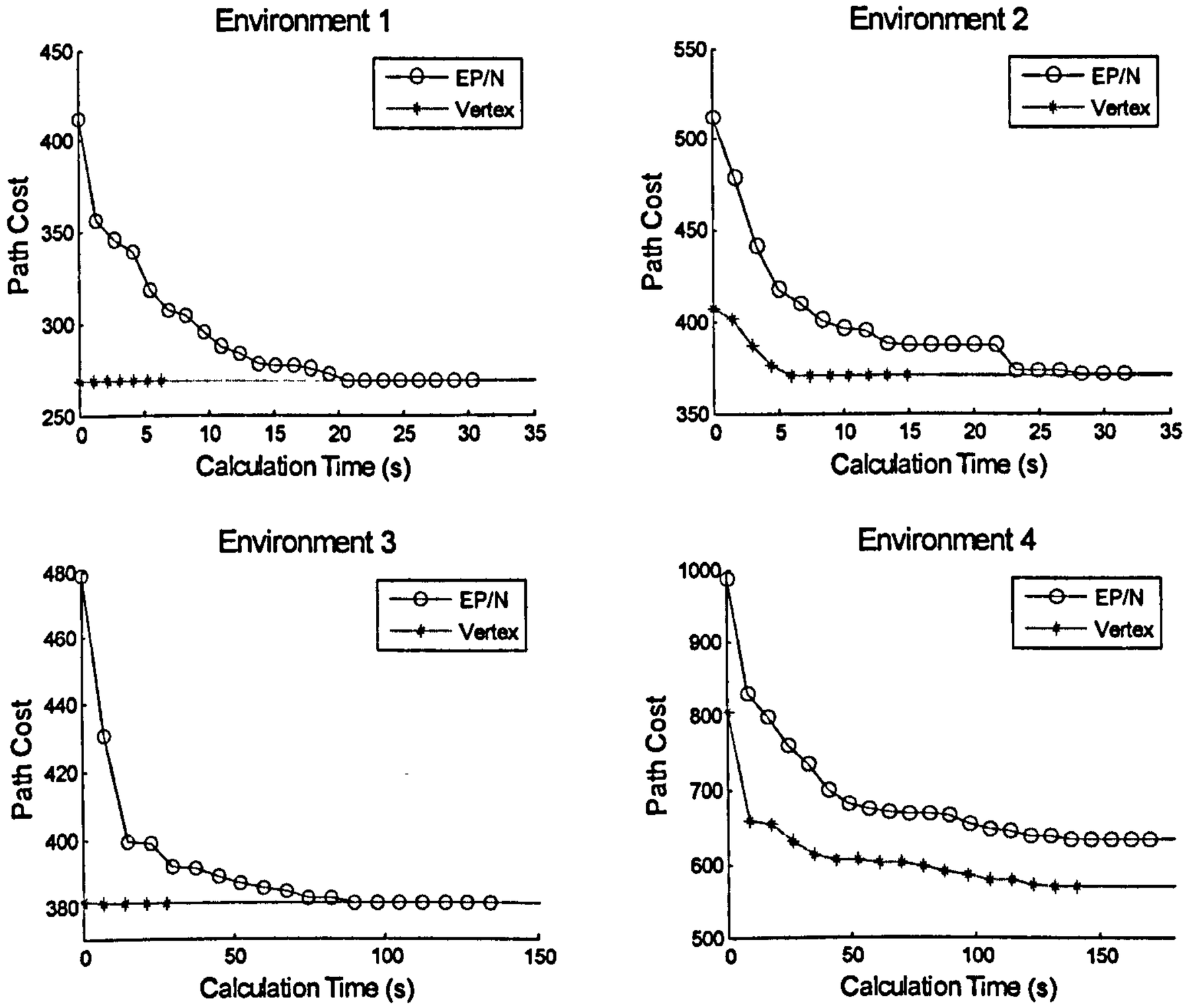


Figure 4.7 The feasible path lengths for the best individual during the second evolutionary phase. Note that the plot is from the first generation in which there existed at least one feasible path, to the generation in which no improvement in the fitness of the best individual has occurred over 300 successive generations. The path length is the median over 50 runs.

Table 4.4 Execution time and number of generations to determine the final path. Also shown is the total path length obtained. The results are the medians obtained over 50 runs.

environment	planning method	execution time to find the final path (s)	number of generations to determine the final path	path length
1	EP/N	31.6	759	268
	Vertex	6.95	328	268
2	EP/N	32.3	644	371
	Vertex	15.6	528	370
3	EP/N	140	1234	380
	Vertex	33.9	487	380
4	EP/N	176	1235	634
	Vertex	141	912	566

In the comparative results above, the experiments were conducted using 30 individuals in all cases. As this population size is unlikely to be optimal, additional experiments were carried out to assess the effect of the population size on both the calculation time and the path cost. In each case, 50 runs were performed at each of 10 different population sizes. The calculation times and the path costs for the four environments are shown in Figure 4.8 and Figure 4.9 respectively. As expected, Figure 4.8 shows that the calculation times increase with the population size, but Figure 4.9 indicates that in all environments the path length is largely independent of the population size. Outliers found in the path length results indicate that there were experiments in which a significantly sub-optimal solution was produced. In all the environments, these outliers are more prevalent at smaller population sizes and only at a population size of 100 individuals are no outliers present in any of the environments.

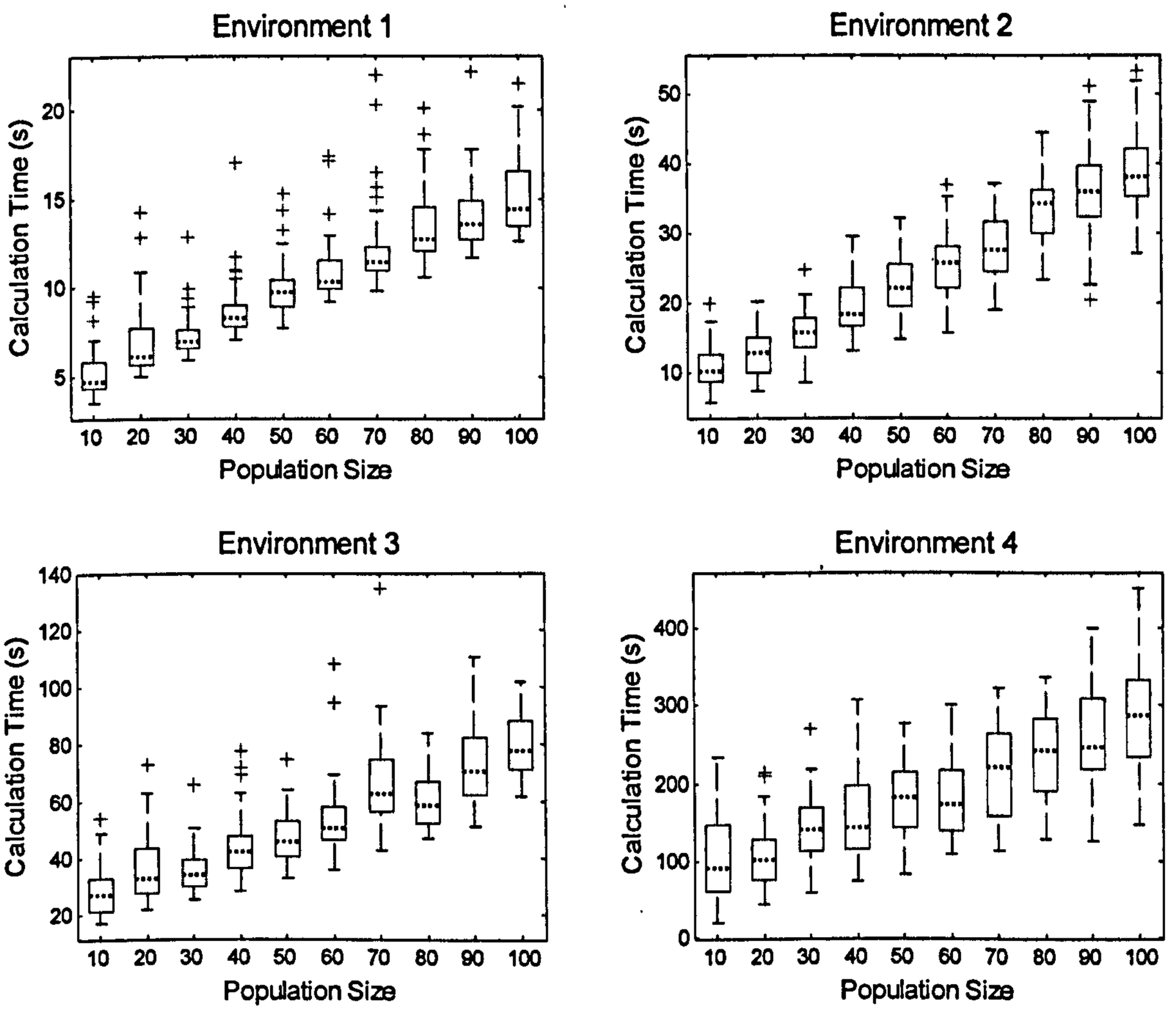


Figure 4.8 Effect of altering the number of individuals in the population on the calculation time for the vertex planner. For each population size, the calculation time of the best individual has been obtained over 50 runs. In the box plot, the box itself contains 50% of the samples, so the top and bottom edges of the box indicate the upper and lower quartiles respectively. The broken horizontal line within the box is the median value, the 'whiskers' above and below indicate values 1.5 times that of the inter-quartiles and the outliers beyond this range are denoted by a plus sign. More details on box plots can be found in Nelson (1989).

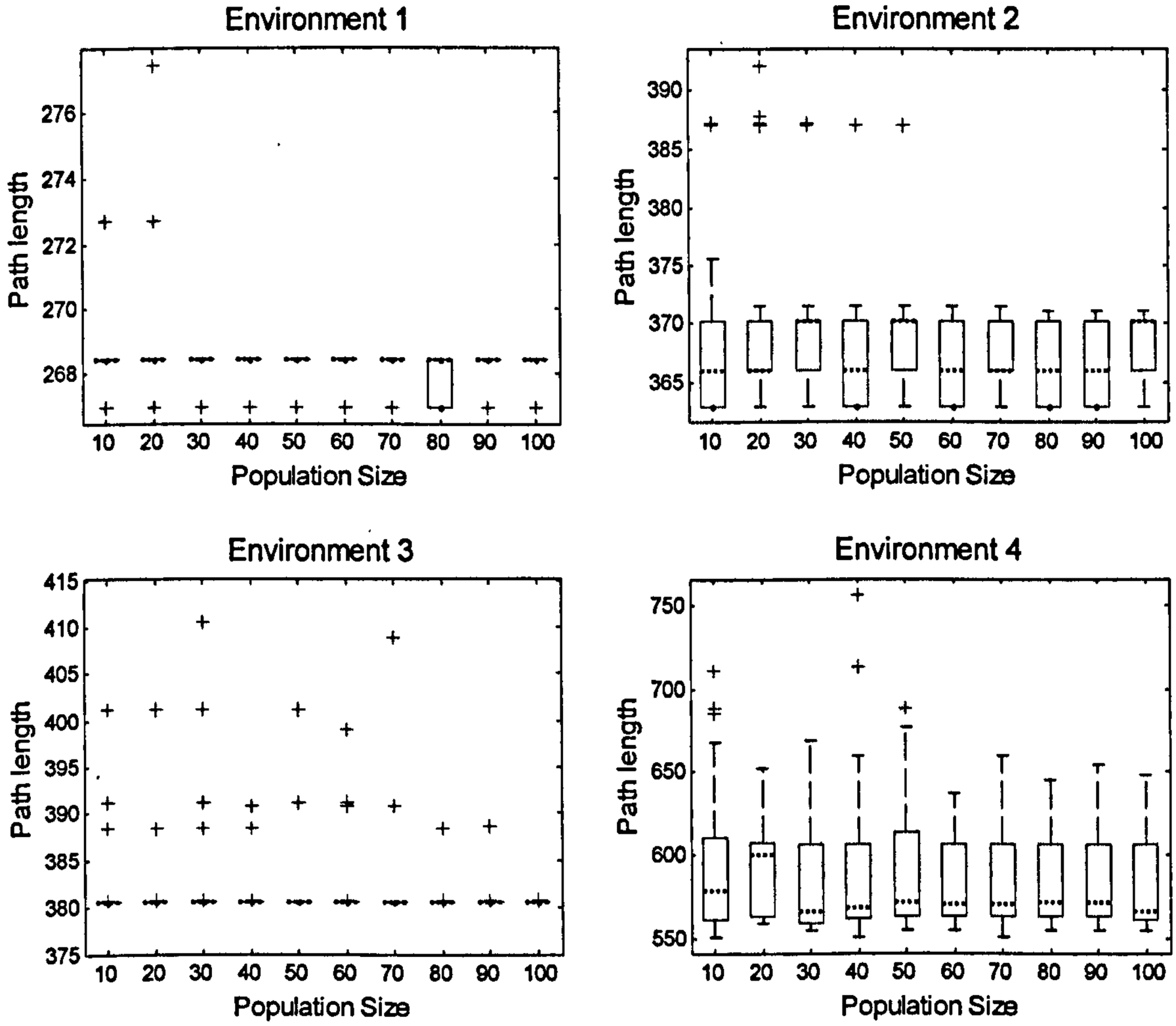


Figure 4.9 Effect of altering the number of individuals in the population on the path length for the vertex planner. For each population size, the path length of the best individual has been obtained over 50 runs.

4.3 Conclusion

The series of experiments presented in this chapter have investigated the efficiency and effectiveness of a new vertex planner. Compared with previous methods, the vertex planning approach operates in a substantially reduced search space, as a result of adopting the obstacles' vertices as the intermediate nodes in the path. In particular, the results show significant performance improvements in comparison with EP/N which searches the whole of the free space for a path (assuming that in most environments encountered, relatively little area is occupied by obstacles). However, given its search space advantage, the convergence performance of the vertex planner is perhaps not as good as might reasonably be expected. As the current work has adopted the operator probabilities that have been optimised for EP/N, it is likely that further experimentation to obtain probabilities better suited to the vertex planning

method may yield improved performance. In those cases where the free space is not dominant, perhaps due the presence of many large obstacles (such as in environment 3), it may be expected that the performance is adversely affected. In contrast, free space dominates in environment 4, but the mean number of vertices per obstacle is greater. Of the environments investigated, this was the only one in which the vertex planner appeared to encounter much difficulty in evolving a feasible path and it is possible that a higher mutation rate should be considered in order to escape local minima. Nevertheless, the vertex planner with a variable-length path representation has been shown to be both efficient and effective in finding solutions of suitable quality for the path planning problem.

The new planning algorithm was presented and a discussion on the experimental results was given in this chapter. However, the static environments simulated in the tests imply a limited application of the vertex planner. Therefore, an augmented planner was subsequently developed for a robot operating in dynamic environments. That work is introduced in the next chapter.

Chapter 5

VERTEX++ PLANNER

This chapter presents vertex++, a genetic-based algorithm for path planning in dynamic environments (those in which one or more moving obstacles are present), and which has the ability to deal with both static and dynamic constrains simultaneously. Although designed initially as an off-line algorithm, vertex++ is also appropriate for use in on-line planning, where its operation can be triggered in response to changes in the expected movements of the dynamic obstacles. The vertex++ navigation approach is an enhancement of the vertex planning method for static environments described in chapter 4. The on-line planning aspects of vertex++ planner have been presented at the 2007 IEEE International Conference on Robotics and Automation and an additional submission will be made to the IEEE Transactions on Evolutionary Computation.

By restricting the planning to obstacle vertices rather than considering the entire environment, the vertex++ planner is able to significantly reduce the calculation time compared with other GA approaches that have been applied in dynamic environments, in which all points in the environment are considered as potential nodes in a path (regardless of whether they are in free space or within an obstacle). A further novel achievement of the new planning approach is the inclusion of robot speed into the planning process, which takes into account the time at which obstacles are encountered, thereby allowing the consideration of a much greater range of possible avoidance paths.

5.1 Related work in path planning in dynamic environments

Planning a path for a mobile robot in dynamic environments is more complex than in static environments, as additional parameters, such as time and velocity, need to be considered in order to generate a collision-free path. A general description of the existing methods that have been applied to solve planning problems in dynamic environments was given in section 3.1, and, this section only addresses approaches based on evolutionary concepts. It is important to note that relatively little work has been reported in the literature that has utilised genetic algorithms as the basis for path planning in dynamic environments.

A number of researchers (Elshamli, Abdullah and Areibi 2004; Patnaik and Karibasappa 2005; Thomaz, Pacheco and Vellasco 1999; Xiao *et al.* 1997) have employed genetic algorithms in the environments where suitable sensors can capture the obstacles that newly appear. However, as these detected obstacles remain stationary only a re-planning process is needed using modified geometric information. As this results in environmental representation much simpler than those needed for the solution of planning problems in dynamic environments, this work is not closely related to that described in this chapter.

Fujisawa *et al.* (2000) described a planning method using only a mutation operator to generate a suitable action (velocity and steering controls) based on 'anytime sensing', that is, to begin the search with a low quality of sensing information, but to gradually improve its quality as the search progresses. More specifically, all obstacles are initially classified into a circular virtual obstacle that is sufficiently large enough to include all obstacles, and, as the search advances, the virtual obstacle gradually fragmenting into smaller virtual obstacles each containing fewer obstacles. In this way, the solution is incrementally improved while processing the sensory information. However, in practical applications the range of influence of the algorithm is rather local due to the limited view obtained by the sensors.

Smierzchalski and Michalewicz (2000) developed an evolutionary based algorithm, termed ϑEP/N++, to generate a collision-free trajectory at sea. This algorithm extends EP/N that was designed to operate in static environments (Xiao *et al.* 1997) by considering the speed of the other moving ships to search the entire and continuous environment for an optimal or near optimal solution. A more detailed description of this planner was provided in chapter 3. Path planning for ships in a continuous environment was also investigated by Zeng (2003) who encoded not only position and speed parameters, but also noise resulting from tide, wind, and wave effects. His algorithm applied a crossover operator that determines the crossing site between two parents of variable length according to the phonotypic function of parameters (position, speed, and noise) rather than using their genetic locus. Following crossover, one gene selected from a fitter chromosome is altered by mutation. The potential for collision between the ship and other moving objects is assessed based on the values of the closest points of approach. If the distance is smaller than the sum of radii of the safe areas around the ship and the moving object, a collision risk exists.

Chen and Xu (2005) added three operations to the standard genetic algorithm to improve global optimality in a planning problem. The first two operations, termed restoration and reconstruction respectively, reinstate or rebuild a previous population if the best individual after crossover and mutation is less fit than the previous best. Otherwise, the new fitness will be remembered by recording the better operation. A unique length representation is used in the GA, and, to reduce the string length, two dimensional data (that is, coordinates with the original workspace *XOY*) were projected to one dimension by converting the original workspace into a new coordinate system, with the origin at the start point and the x axis drawn to goal point. The locus of a chromosome thus represents the x coordinates of a set of via points equally spaced along the x axis, while the content of each gene is the y coordinate. However, such a representation may result in the planning algorithm not finding the optimal solution, as x coordinates are recorded in monotonic order and no reordering operator is available. To achieve dynamic obstacle avoidance, the via points need to be selected to ensure the distance from each via points to an obstacle is longer than the sum of the radii of the robot and that of the obstacle under consideration. In

addition, the method also assumes that the number, the positions and the speeds of obstacles can be completely captured by the sensor system.

GAs have been also applied to the path planning of multiple mobile robots. Liu, S., Tian and Liu, J. (2004) planned a collision-free shortest path for each robot from its initial location to its destination while ignoring the presence of other robots. A reactive strategy was adopted to resolve locally the conflicting paths. Each individual was encoded as a linked list of variable length where each gene contained the coordinates of a cell in a grid. The initial population was constrained to the range bounded by the coordinates of the start point and the goal point, the crossover and mutation operator was modified to cope with the planning problem and two evaluation functions were developed to assess the quality of the feasible and infeasible paths. The fitness function used for the infeasible paths, defined as the inverse of the number of infeasible segments plus one, is not an intuitive parameter, since the single straight line between the start point and the goal point is necessarily the fittest one among infeasible paths. A similar approach in which a global geographical path was generated and then collisions avoided using a set of behaviours was adopted by Fu *et al.* (2006). The practical problem considered was to navigate a planet rover from a start point to a target point and then return to the start point, while passing through a set of sub-targets (and is largely equivalent to a travel salesman problem). A local planning algorithm was designed to generate a path between pairs of adjacent sub-targets according to the gradients of the 3D terrain modelled by a grid map.

GAs have also been used for planning problems in 3D workspaces containing moving objects. One example is the path planning of an unmanned aircraft (Rathbun *et al.* 2002), where path was composed of a set of straight lines and constant radius curves joined smoothly. Four mutation operators were developed; at each generation one was chosen randomly for application to each individual to alter one or more of the following parameters: length, radius, and speed. The check for potential collision with obstacles considered the expected position and velocity, safe approach radius and uncertainties of the position and velocity of the obstacles. A second example is

genetic-based local path planning designed to avoid moving obstacle when navigating an autonomous underwater vehicle (Chang *et al.* 2005). A rectangular grid map representing the environment sensed was rotated such that the straight line from the start point (the origin) to the goal point becomes the *x* axis. The location and velocity of moving obstacles were assumed to be observed by the sonar sensor and the avoidance was achieved by changing the heading. In addition to the standard operators, namely, one-point crossover and mutation, an additional operator, termed the moving operator, was designed to navigate the vehicle close to the previously planned path while avoiding collision. The avoidance constraint was set to be larger than the dimension of the vehicle plus the radius of the obstacle.

The approaches to navigation in dynamic environments have adopted a grid representation or have operated on the original continuous working environments. Although the grid representation may reduce the search space to some extent, the construction (or reconstruction based on updated information about the dynamic environments) requires careful selection of grid dimensions and considerable pre-processing. Converting the *x* axis into a line connecting the start point and the goal point can reduce the lengths of individuals in the genetic representation, but does not allow any movement in the negative *x* direction and consequently some feasible paths may never be discovered. To accelerate the search process or improve the path quality, a conventional GA is rarely employed directly, but rather some modifications are made based on problem-specific knowledge. It is important to note that in all the work found in the literature, the proposed algorithms were only evaluated in the specific environments developed by the respective authors and none was assessed in their comparison with other algorithms.

The vertex++ planner presented in this chapter is able to perform planning operations in dynamic environments and is an extension of the vertex planner described in chapter 4. As 9EP/N++ is the extended version of the EP/N to operate in dynamic environments, it was used to provide a benchmark for the vertex++ planner.

5.2 Planning algorithm

This section describes the types of environment used in the experiments carried out with vertex++, details the internal structure of the GA used for the planner and describes the operation of the vertex++ during both off-line and on-line planning.

5.2.1 Operating environment and constraints

The working environment for the mobile robot consists of a set of stationary obstacles whose shapes either are defined to be, or are approximated by, bounding polygons. In addition, the robot movement between two specified locations may be interrupted by the presence of one or more dynamic obstacles that are also represented by polygons. If the motion parameters (here the current heading and current speed) of those dynamic obstacles remain constant, a safe trajectory for the robot can be generated by the vertex++ planner in an off-line manner. In addition, the path generated off-line can be adaptively revised in response to any changes in the motion characteristics of the dynamic obstacles.

In the off-line planner, it is assumed that complete motion knowledge of the moving obstacles in the environment is available. Consequently, an inherent assumption in the off-line planner is that the information gained remains unchanged following the generation of a safe path. In the on-line planner, it is assumed that changes to the motion parameters of the moving obstacles are made available whenever one comes within the sensor range. Although no particular sensor type or configuration is specified, it is assumed that in order to allow the robot to be guided so as to avoid any potential collisions with obstacles, there is an adequately large time interval between the detection of obstacle movements and the implementation of newly generated actions. Note that this assumption may be relaxed if guidance is achieved by reactive navigation, such as in (Mulvaney *et al.* 2006).

For purposes of planning, the static obstacles are enlarged by a value determined from the minimum distance (herein referred to as the safe distance) that the robot can approach obstacles without collision, to account for the robot dimensions (see Figure

5.1 for an example). Such a representation allows the physical dimensions of the mobile robot to be neglected and regarded as a single point.

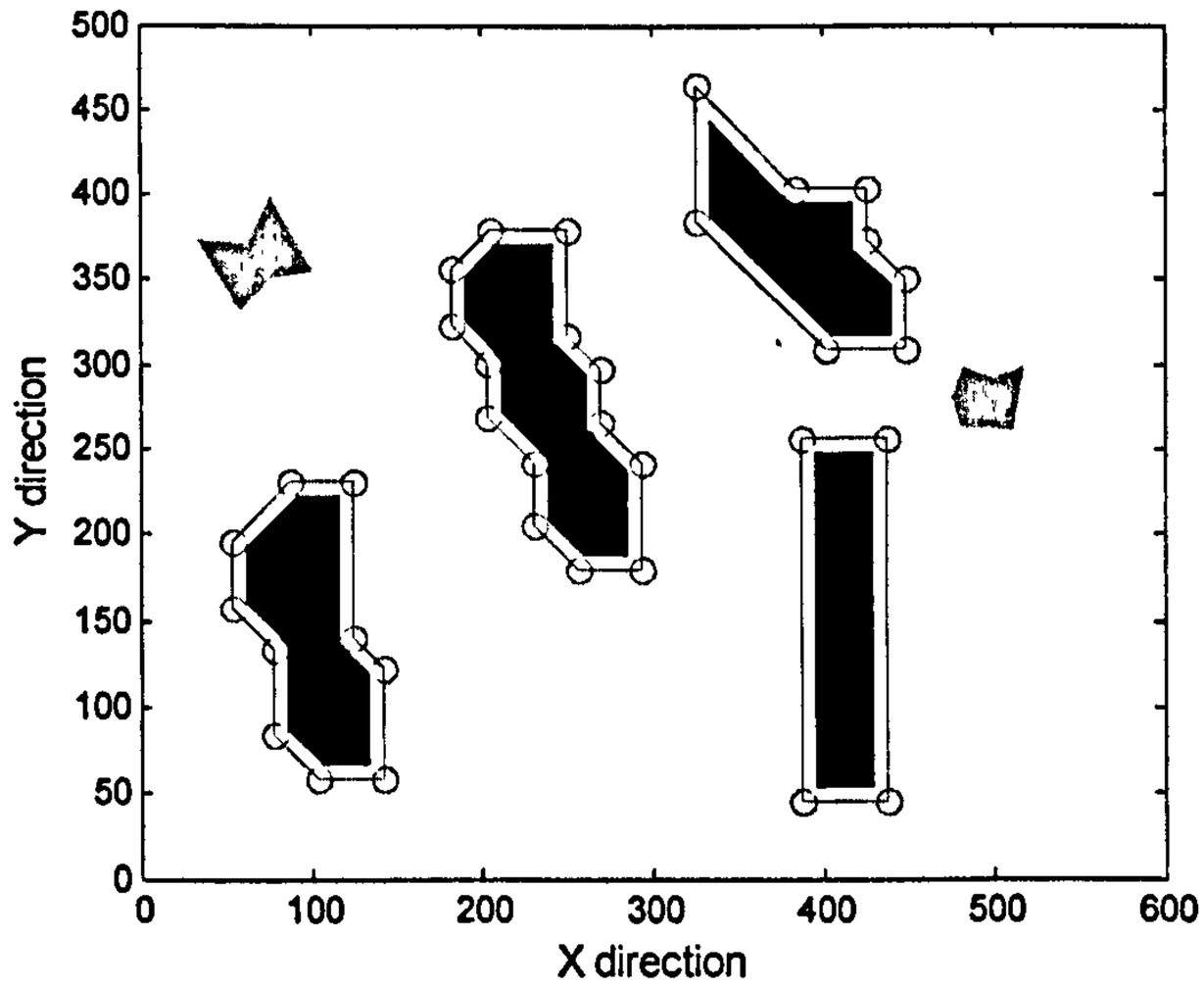


Figure 5.1 An example of the environment representation in vertex++ planner. The grey polygons represent exclusion areas surrounding the moving obstacles; the black obstacles are static.

To model the motion of the dynamic obstacles in the vertex++ planner, the same strategy as in 9EP/N++ (Smierzchalski and Michalewicz 2000) is adopted. In brief, for each obstacle, its motion is described by a trajectory, consisting of a series of one or more segments, each having start and finish coordinates between which the heading (defined by the coordinates) and the speed of the obstacle are fixed.

In order to assess the possibility of the robot colliding with dynamic obstacles, the following method has been developed and implemented in vertex++. The first crossing point between the robot path proposed by the planner and the trajectory of a moving obstacle is calculated before examining the possibility of collision. Based on the time t required for the robot to cover the distance from the current position to the first crossing point, the instantaneous location of the moving obstacle can be calculated and, consequently, the exclusion area for this obstacle. If the crossing point falls within this area, a collision would occur between the robot and the moving obstacle. An example of such an occurrence is shown in Figure 5.2. Note that the

safety margins for the longitudinal and lateral dimensions of the moving obstacle are unlikely to be the same when constructing this area, as the speeds of the robot and the moving obstacle need to be taken into account in addition to the dimensions of the robot. For the problem in Figure 5.2, the time t is firstly calculated for the robot to travel from its current location to the crossing point determined from the generated path. The instantaneous location of the moving obstacle after time t can then be calculated according to the motion information relating to the dynamic obstacle, allowing a region to be identified for assessment of feasibility using the algorithm for checking polygon clipping given in (Pavlidid 1982).

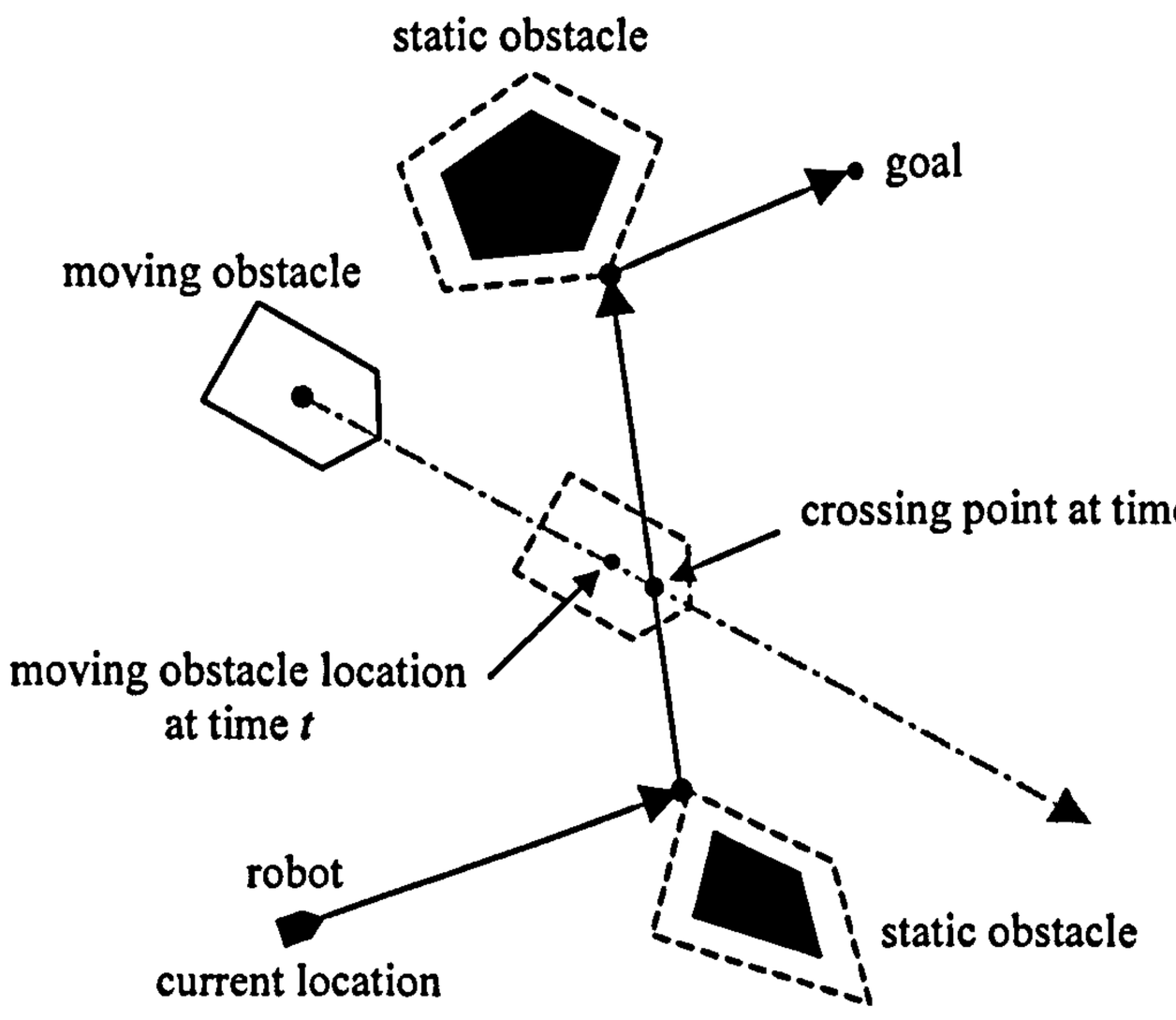


Figure 5.2 Evaluation of the possibility of collision with a moving obstacle. Note that the intermediate nodes of the generated path illustrated are vertices of the enlarged static obstacles.

5.2.2 Pseudo-code of the vertex++ planning algorithm

The vertex++ planner preserves the structure of the steady-state GA vertex planner described in the last chapter. The pseudo-code for the vertex++ planning algorithm is shown in Figure 5.3 and its implementation is described in the following sections.

```
Procedure Vertex++ Planning Algorithm
  begin
    number of generations g = 0;
    input environmental information from sensors;
    enlarge the static obstacles;
    initialise the population P(g);
    evaluate P(g);
    while (not termination condition) do
      increment the number of generations g = g + 1;
      randomly select Oj from four operators
          (crossover, mutation, repair, and speedmutation);
      select parents from P(g);
      apply the operator Oj to produce offspring;
      evaluate new offspring;
      replace worst member in P(g) by offspring;
    end while
    select the best individual p from P(g);
  end
end procedure
```

Figure 5.3 The vertex++ algorithm.

5.2.3 Genetic representation and initialisation

Candidate paths are represented by a chromosome (Figure 5.4) consisting of a total number of genes l, where l has a minimum value of two (a path containing only the start and goal points) and maximum value of N+2, where N is the total number of the vertices of all obstacles (both static and dynamic) in the environment. The absolute coordinates of the vertices (x_i, y_i), i=1,..,l-2, are used directly in the gene representation rather than a reference to one of the N vertices (Wang, Mulvaney and Sillitoe 2006). The robot's speed, s_i in the segment originating from each gene is selected from a set of available discrete speeds. A single bit is also provided in each gene to indicate the feasibility of the path that originates from the gene; if the path segment connecting two consecutive vertices intersects one or more obstacles, then the infeasibility bit of the gene representing the originating node is assigned 1 to mark this segment as infeasible (it is 0 otherwise).

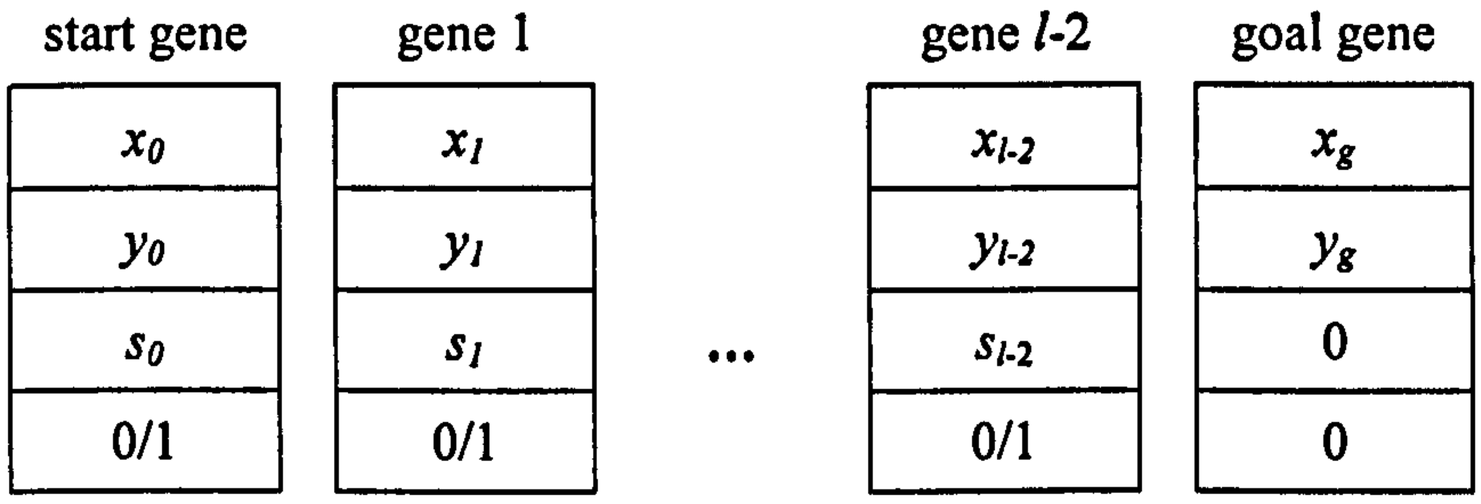


Figure 5.4 The structure of a chromosome. The first and last genes in the chromosome indicate the start and goal points respectively. The l-2 intermediate genes represent the vertices of obstacles in the environment as well as the speed and feasibility of the segment that originates from the gene.

The initial population $P(0)$, is generated by randomly choosing for each individual both its length (in the range 2 to N+2) and the coordinates of the vertices contained therein, with the constraints that no vertex is repeated in a individual and that the first and last genes are always the start and goal points respectively. The speed for each gene is selected randomly from the set of discrete speeds. Compared with a fixed-length chromosome approach, the variable length strategy not only reduces the memory storage requirement, but also the processing time, as more time is generally needed for the fitness calculations if the individuals are longer.

5.2.4 Evaluation functions

A check for the feasibility of each chromosome is performed before evaluation of the chromosomes. Separate evaluation functions are applied to assess the qualities of infeasible and feasible paths.

Two parameters, path length and travel time, are considered in the evaluation function E_f for feasible paths. E_f is given by

$$E_f = w_d \bullet \sum_{i=0}^{l+1} d(V_i, V_{i+1}) + w_t \bullet \sum_{i=0}^{l+1} t(V_i, V_{i+1}), \qquad \text{Equation 5.1}$$

where w_d and w_t are the weights for path length and travel time respectively, $d(V_i, V_{i+1})$ denotes the distance between the pair of vertices and $t(V_i, V_{i+1})$ represents the time needed to cover each segment from vertex V_i.to V_{i+1} which can be calculated by

$$t(V_i, V_{i+1}) = d(V_i, V_{i+1}) / s_i . \qquad \text{Equation 5.2}$$

where s_i denotes the speed of the robot when travelling from V_i to V_{i+1}. The infeasible paths are evaluated using the function E_i by considering the deepness of an infeasible path's intersections with obstacles and is given by

$$E_i = \mu + \eta , \qquad \text{Equation 5.3}$$

where μ denotes the number of obstacle intersections along the entire path and η is the mean number of intersections in the infeasible segments. Given the two evaluation functions, the aim of the optimisation process in the vertex++ planning approach is to minimise the values of E_f and E_i for their respective populations.

When the population contains both feasible and infeasible paths, all infeasible paths are assumed to be no better than the worst feasible path. A sufficiently large constant C is added to the costs for the infeasible paths to ensure the evaluation values of any given infeasible path is worse than the values for all feasible paths. This constant value C is given by

$$C = (N+2) \cdot D , \qquad \text{Equation 5.4}$$

where N+2 indicates the maximum possible number of genes in an individual and D denotes the maximum length of a path segment (for example, this would be the diagonal in a rectangular environment).

5.2.5 Genetic operators and their selection

Three of the total of four genetic operators are the same as those used in the vertex planner in the previous chapter, namely, *crossover*, *mutation* and *repair*. The fourth operator, termed *speedmutation*, has been introduced in this work in order to mutate the robot speed indicated in a gene and it is selected with a small probability. In order to keep the number of system parameters to a minimum, the selection of an operator from the four available is made randomly at each generation rather than being based on pre-defined probabilities for those operators. The crossover operation is performed by a conventional one-point operator, following which individuals are examined for repeated vertices, and, in order to eliminate circular paths, those replicated vertices of lower locus are removed (as the fitness of such individuals is worse than those

without a ring branch). The reason for choosing single point crossover is that when there are two or more crossover points there is a greater probability of making currently feasible paths infeasible. When the mutation operator is selected, only one bit is modified in the chosen individual. Mutation is inhibited if the replacement genes are already present in the individual. Note that the mutation rate will depend on the length of the individuals; for example if the average length of the individuals is 10 bits for a certain planning task, then, on average, only 1 bit will be mutated in every fourth generation (there being four operators), giving a mutation rate of 0.025. This is why a single gene for mutation is selected from an individual rather than being based on a pre-defined probability. Consequently, the mutation rate is effectively adapted during the search process. The repair operator adjusts a randomly selected infeasible segment of an infeasible path, so that it circumnavigates all obstacles previously intersected, as illustrated in Figure 4.4 in the previous chapter.

5.2.6 Evolutionary process

As a steady-state GA has been adopted, only (following crossover) a single pair of individuals differs between consecutive generations. The generational operation begins with the random selection of a genetic operator and a quadratic ranking scheme (De Jong 1992; Watanabe and Hashem 2004) (see section 2.3) is used to retain the constant selection differential after evaluation. The parent (or parents for the crossover operation) that is involved in the genetic operation is determined by a roulette wheel whose slots are sized in proportion to the fitness as scaled by the ranking technique. To form a new generation, the newly generated offspring replaces the least fit individual (or pair of individuals if crossover is applied). The evolutionary process continues until a termination condition is satisfied, which can be defined to be a number of generations specified by the user or determined dynamically by monitoring specified performance criterion. When evolution terminates, the fittest individual is selected as the path planning solution.

5.2.7 On-line planning

On-line planning is triggered automatically to adapt to any changes in the movement characteristics of the dynamic obstacles that have occurred since the off-line plan was

computed. On-line planning is instigated only when such changes are detected within range of the robot's sensors, otherwise the robot continues to follow the previously-planned trajectory. Information gathered from the robot's sensors, with regard to the motion changes of obstacles, is supplied to the vertex++ planner which then uses the current state of the robot as the start configuration for its on-line evolutionary planning, and evolves a new path for the robot. The on-line planning algorithm is the same as that used in off-line planning, but with the additional assumption that the planning time is relatively short compared with that needed for the robot to implement motion changes to avoid collision with dynamic obstacles.

5.3 Experiments and results

Three separate experiments were carried out using MATLAB (Mathworks 2006) simulations on a 2.8GHz Pentium P4, each applied in the same set of four simulated environments. In the first, the robot speed is constrained to remain constant over the entire trajectory from start to goal, in the second, the robot speed was allowed to vary between segments and in the third, the on-line planner was required to respond to a number of motion changes made by the obstacles and the path was separately optimised for both travel time and path length. In the simulations, the trajectories of the dynamic obstacles along a segment joining any pair of adjacent nodes are approximated as linear. The number of obstacles present in each of the test environments is summarised in Table 5.1.

Table 5.1 The numbers of obstacles in the four test environments.

environment	number of static obstacles	number of dynamic obstacles
1	4	2
2	5	3
3	9	4
4	14	5

To provide realistic challenges to the planner, the four environments were designed to reflect a representative range of applications in which mobile robots may be expected to operate. Simple trajectories for the dynamic obstacles were designed for the first

two environments, with the obstacles simply traveling to and fro between two specified locations. More complex paths for the dynamic obstacles were defined for the remaining two environments, involving speed changes and movement between a series of nodes. For off-line planning, the information regarding the motions of the dynamic obstacles is assumed to be completely known for the four environments before planning. Apart from the robot speed and the optimising criteria, all parameters remained unchanged throughout the set of experiments and they are listed in Table 5.2.

Table 5.2 System parameters for the vertex++ planner. Note that mutation acts on only one gene to alter either the vertex or the speed of the selected segment and that the value of the safe distance is determined from the minimum distance that the robot can approach obstacles without collision, taking into account its physical dimensions.

population size	mutation rate (for node)	mutation rate (for speed)	repair rate	safe distance (m)
30	one gene	one gene	one infeasible segment	1

5.3.1 Robot of constant speed

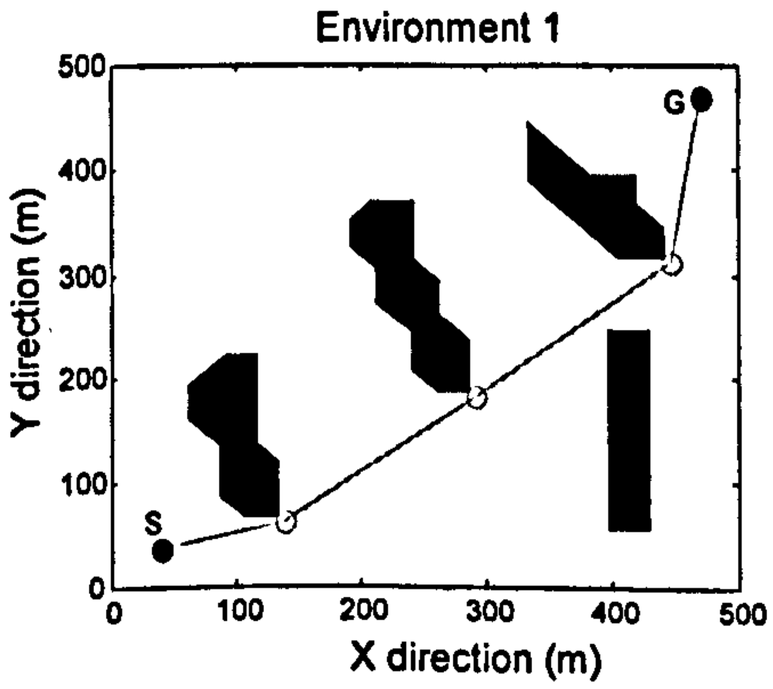
The speeds of the obstacles in the different environments are chosen from a uniform distribution in the interval $0.01ms^{-1}$ to $1ms^{-1}$ and an example of generated values are shown in Table 5.3. The results in Figure 5.5 show the paths produced for the robot traveling at constant speed (here assumed to be $0.5ms^{-1}$) in the four environments first in the absence of dynamic obstacles (left column) and secondly in their presence (right column). In the experiments, 1000 generations was normally more than sufficient to generate a suitable solution. The execution times to obtain feasible solutions in the dynamic environments were longer than those for the environments containing only static obstacles, but even in the more complex environments, consistent results were produced in less than 60 seconds. Little variation was evident when the navigation task was repeated. It can be seen that, in comparison with the results from environments where all obstacles are static, the paths generated in the presence of the moving obstacles are generally longer. In the dynamic environments, the planner often chose an alternative route around the static obstacles in order to avoid potential collisions with dynamic obstacles, thereby producing a shorter overall path through the entire obstacle population. Although this did not occur in

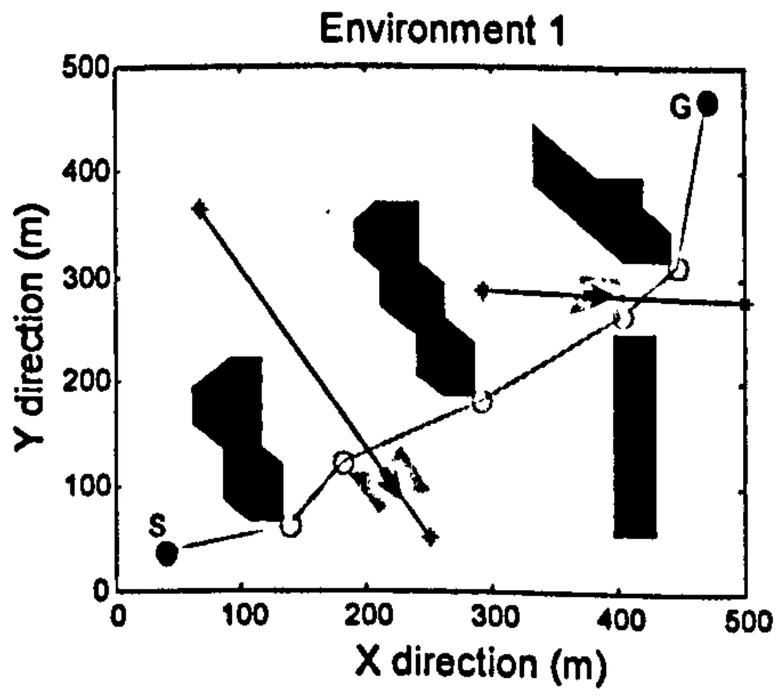
environment 1, where only a minor deviation was needed around the first dynamic obstacle encountered, in environments 2, 3 and 4 alternative routes around at least one of the static obstacles was chosen.

The quantitative experimental results are presented in Table 5.4, where it can be seen that when moving obstacles are present there is an increase in both the execution time and the number of generations required to determine feasible paths. This increase is most pronounced in environment 3 and this appeared to be due to the complex interactions that result from all four moving obstacles operating simultaneously in a region that needs to be traversed by the robot. From Table 5.4, it can be seen that, for the dynamic environments, the number of generations required to find the first feasible path for environment 1 with moving obstacles is double that obtained in dynamic environment 2, but a similar execution time is required. One possible reason is that although the application of the four genetic operators is randomly selected, their execution times are different. Therefore, the operators needing longer calculation times may be selected more frequently in the experiments carried out in dynamic environment 2 than in dynamic environment 1. Although the costs shown in Table 5.4 are travel times, the corresponding path lengths are linearly related due to the constant speed of the robot.

Table 5.3 Examples of speeds (in ms^{-1}) for the moving obstacles in the experiments conducted for the robot with constant speed

environment	obstacle 1	obstacle 2	obstacle 3	obstacle 4	obstacle 5
1	0.75	0.35	-	-	-
2	0.15	0.28	0.32	-	-
3	0.81	0.39	0.18	0.47	-
4	0.53	0.35	0.18	0.43	0.27





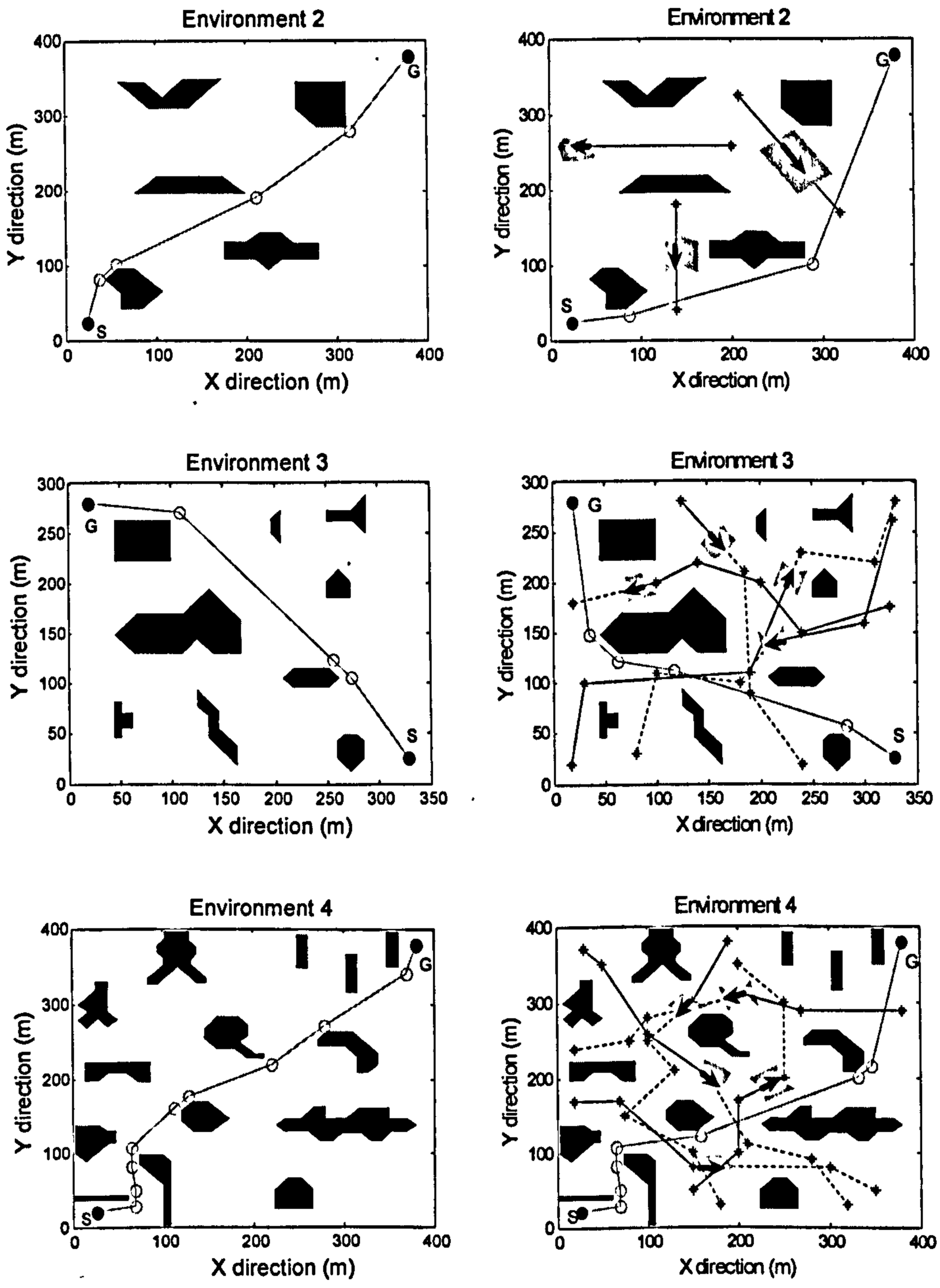


Figure 5.5 Comparison of the generated paths for the four environments both with no moving obstacles (left column) and with moving obstacles (right column). The paths generated for the robot follow a series of nodes marked by the symbol 'o', that indicate the intermediate vertices. The start and goal points for the robot are indicated by the solid dots marked 'S' and 'G' respectively. The trajectories for the moving obstacles (with nodes marked by the symbol '*') are shown solid in those parts already traversed up to the first crossing point (see Figure 5.4) and broken for the remainder of the trajectory yet to be followed. The positions of the moving obstacles shown are those at the time the robot reached the first crossing point. If crossing does not occur, the position of the moving obstacle shown is that taken at the time the robot reaches the goal. The heading of the obstacle at the moment of the first crossing (or at the time the robot reaches the goal) is shown by an arrow in the body of the obstacle.

Table 5.4 Experimental results for the robot with constant speed in the four environments

environment		cost - travel time (s)	generations to find first feasible path	execution time to find first feasible path (s)	execution time for 1000 generations (s)
1	no moving obstacles	1327	8	2.25	20.4
	2 moving obstacles	1338	26	3.81	39.2
2	no moving obstacles	1053	35	3.38	22.5
	3 moving obstacles	1142	13	3.78	40.2
3	no moving obstacles	838	26	3.80	30.9
	4 moving obstacles	911	54	14.0	66.1
4	no moving obstacles	1102	155	27.3	99.7
	5 moving obstacles	1191	166	39.4	118

5.3.2 Robot of variable speed

In these experiments, the robot's speed between a consecutive pair of vertices was selected from the set {0.3, 0.4, 0.5, 0.6, 0.7} ms^{-1}, whereas the speeds of the dynamic obstacles were obtained from the range [0.1, 1.2] ms^{-1} with a resolution of 0.1ms^{-1}; examples of generated obstacle speeds are shown in Table 5.5.

Table 5.5 Examples of speed parameters (in ms^{-1}) randomly generated for each path segment of the moving obstacles for the experiments conducted with the robot operating at variable speed.

environment	obstacle 1	obstacle 2	obstacle 3	obstacle 4	obstacle 5
1	0.5, 0.1, 0.2, 0.4	0.1, 0.5, 0.2, 0.4	-	-	-
2	0.6, 0.1, 0.4, 0.7	0.3, 0.6, 0.1, 0.4	0.1, 0.3, 0.2, 0.4	-	-
3	0.8, 1.1, 0.7, 0.3, 0.5	0.4, 0.5, 0.6, 0.4, 0.3	0.3, 0.5, 0.4	0.3, 0.6, 0.5, 0.7, 0.2	-
4	0.6, 1.2, 0.3, 0.2	0.7, 1.2, 0.6, 0.4, 0.5	0.3, 0.2, 0.5, 0.1, 0.6	0.2, 0.7, 0.6, 0.2, 1.0, 0.4	0.4, 0.9, 0.7, 0.2, 0.5, 0.2

The robot paths for this set of experiments are shown in Figure 5.6. In comparison with the results in Figure 5.6, it can be seen that the trajectories generated have been modified as a result of the range of speeds now available to the planner. In particular,

in environment 1, the planner now chooses to drive the robot in a longer path around the first static obstacle in order to avoid the first dynamic obstacle and, in environment 2, the planner has determined changes to the robot speed allow a path to be followed that more closely resembles that taken for the same environment in Figure 5.5 where only static obstacles were present. Although the paths generated in environments 3 and 4 pass through the same vertices as those in the corresponding dynamic results in Figure 5.5, the instantaneous locations of the moving obstacles are different due to the range of speeds now allowed for the robot and dynamic obstacles. As only travel time is considered in the evaluation function for the feasible paths, in its attempts to minimise the travel time, the GA tends to choose the maximum speed from those available. The details of the number of generations and execution times required for generating the solutions depicted in Figure 5.6 are shown in Table 5.6. In particular, it can be seen that the ability of the planner to select the operating speeds for the individual segments has permitted the more rapid identification of suitable individuals in the GA, and resulted in a significant reduction in the execution time in environment 3.

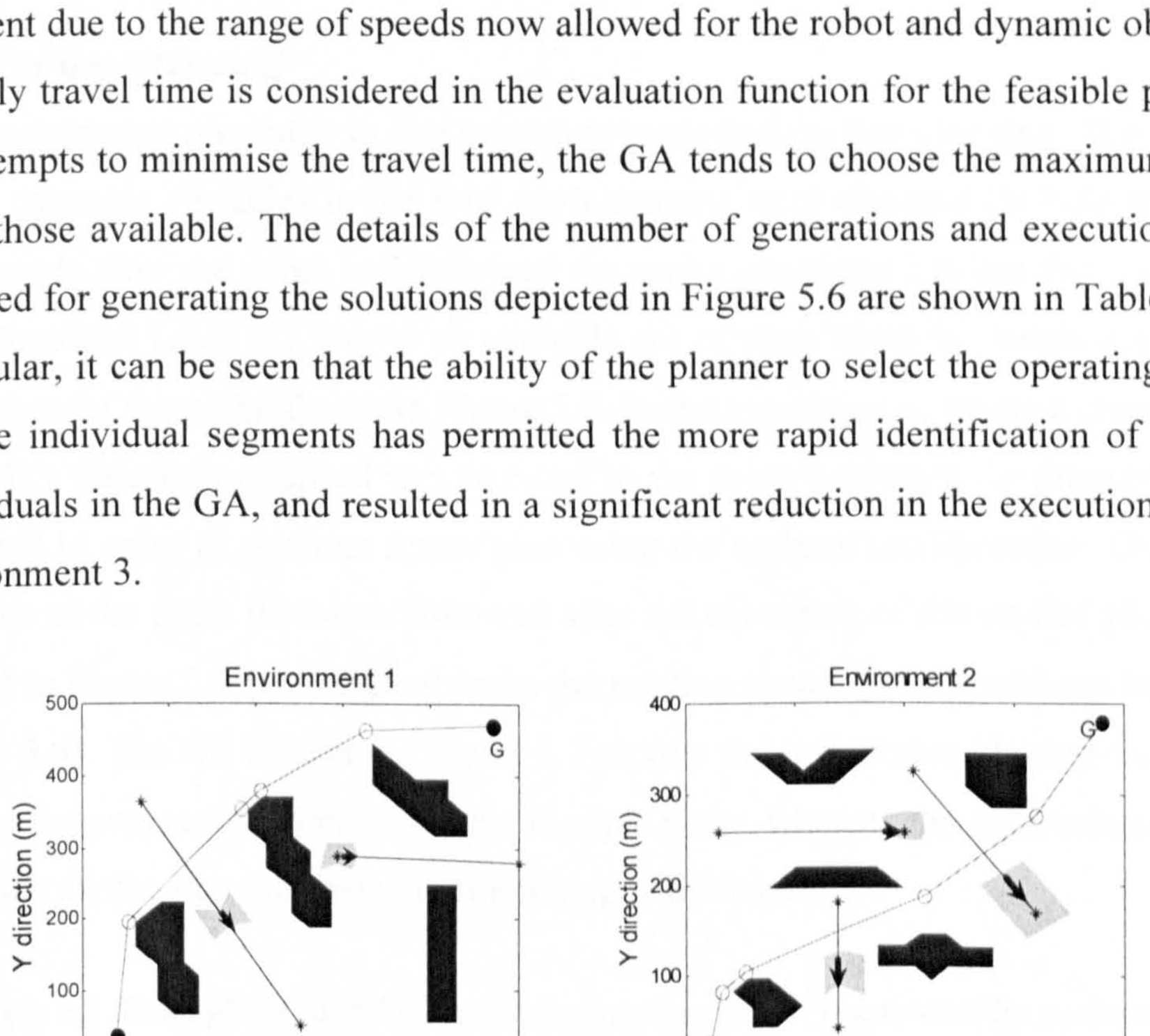


Figure 5.6 The paths generated for a mobile robot with speeds determined by the GA.

Table 5.6 Results in the four environments for the robot with variable speed.

environment	cost - travel time (s)	number of generations to find first feasible path	execution time to find first feasible path (s)	execution time for 1000 generations (s)
1	982	53	4.02	39.4
2	752	54	6.31	40.8
3	651	52	7.78	59.5
4	851	151	36.4	124

5.3.3 On-line planning

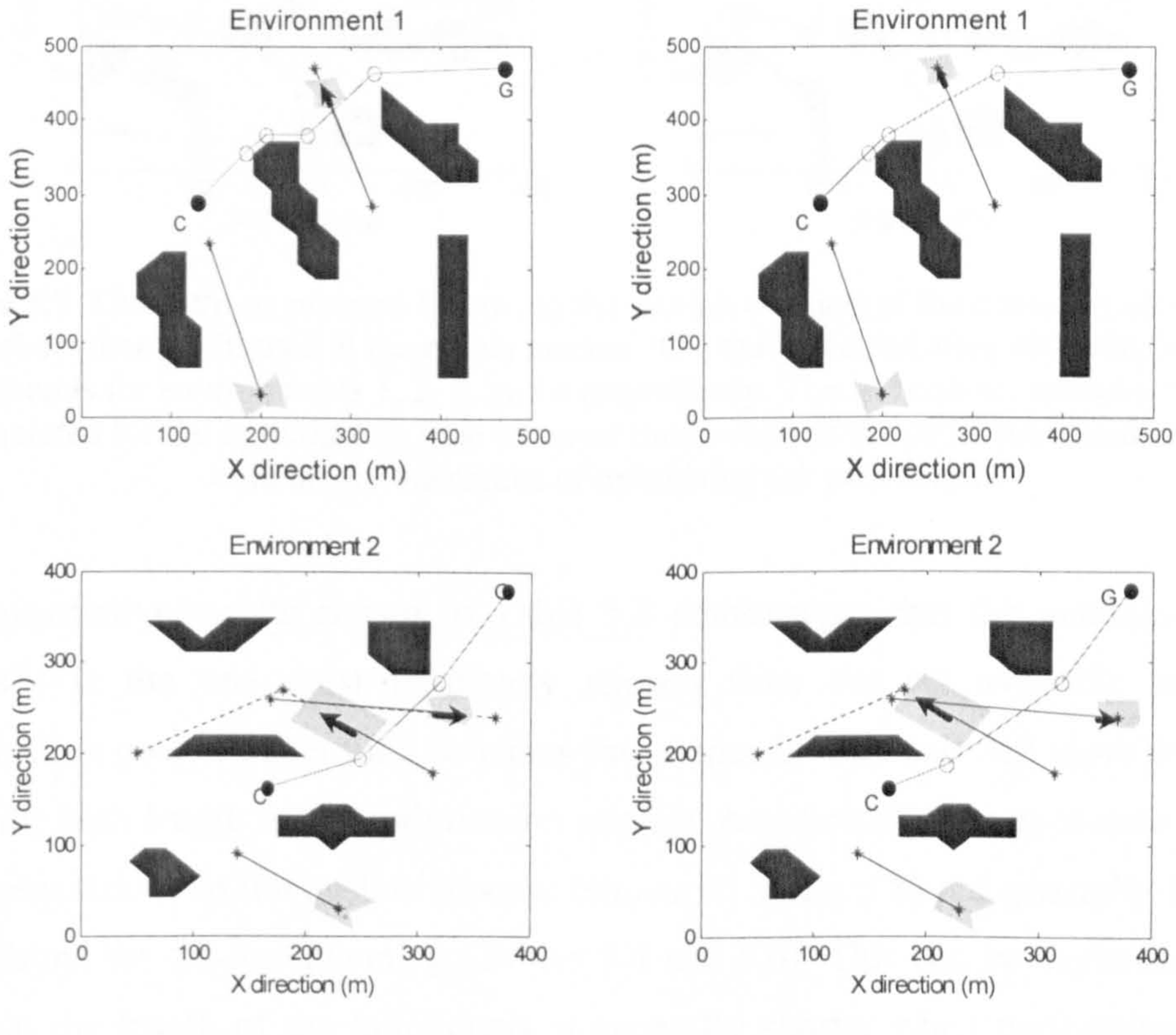
The experiments presented in this section investigated on-line planning. The motions of the dynamic obstacles in the four environments were changed (in both trajectory and speed) after the robot had followed the paths generated off-line for a specified time duration. Table 5.7 shows an example set of such obstacle speeds used in the generation of the paths shown in Figure 5.7. In the experiments, where a change in an obstacle's trajectory or speed was detected by the robot's sensors, the planner was re-executed in order to generate a new plan using the updated configuration. Only those portions of the paths that were followed after the execution of the on-line planner are plotted in Figure 5.7 (the original paths the moving obstacles followed can be seen in Figure 5.6). The left column in Figure 5.7 reports the trajectories planned with travel time as the evaluation criterion for the feasible paths, whereas the right column shows the paths evolved on the basis of minimising path length.

Table 5.7 Example of modified speed parameters (in ms^{-1}) generated for each moving obstacle path segment.

environment	obstacle 1	obstacle 2	obstacle 3	obstacle 4	obstacle 5
1	0.1	0.55	-	-	-
2	0.4, 0.1	0.2	0.5	-	-
3	0.3, 0.5	1.1, 0.4	0.7, 0.4, 0.6	1.2, 0.5, 0.4, 0.8	-
4	0.9	1.4, 0.7	1.1, 0.2, 0.5	0.3, 1.1	1.6, 0.5, 0.6

Comparing Figure 5.6 and Figure 5.7, the path generated for environment 1 when optimising for travel time (left column) is modified in order to avoid the moving obstacle, but, when optimising for path length (right column in Fig 5.8), the same path as that obtained in Figure 5.6 can be followed. For environment 2, the re-planned

path in the right column of Figure 5.7 appears to be the same as that planned before the obstacle changed trajectory, whereas, on careful examination, it can be seen that the trajectory in the left column has required minor modifications to the choice of intermediate nodes. The paths presented in both columns of Figure 5.7 for environment 3 selected, as intermediate nodes, the same vertices in same order. However, neither path for environment 4 is the same as that generated off-line (Figure 5.6), since the planned robot movements were forced to change by the modified motion of the dynamic obstacles. In general, it can be seen that as a result of different optimisation criteria, the trajectories illustrated in the right column for path length are generally shorter than those in left column for path length.



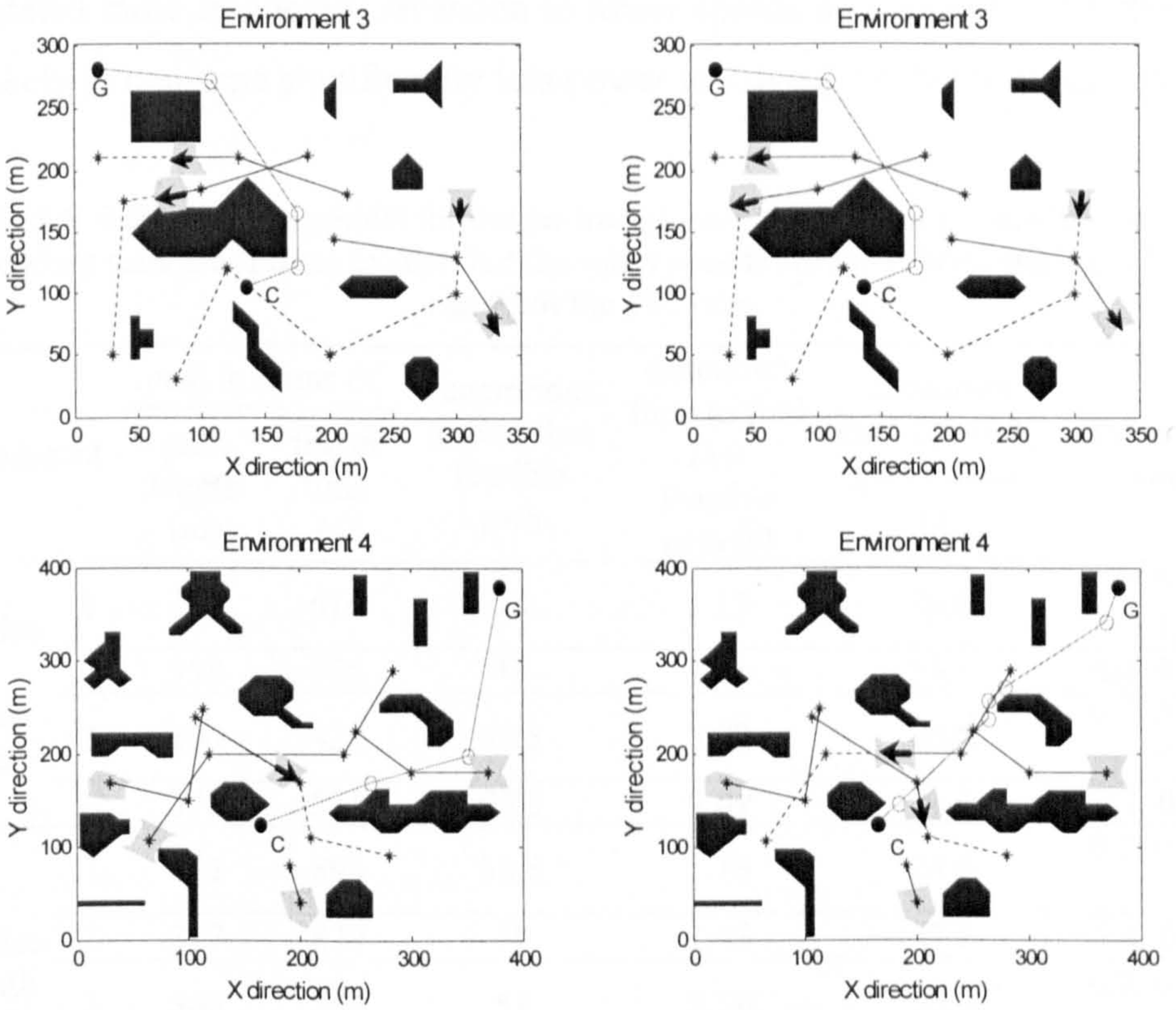


Figure 5.7 The paths as planned following the motion changes of the obstacles (detected by the robot when positioned at the points marked 'C') that occurred after 400, 380, 300, and 320 seconds for environments 1, 2, 3, and 4 respectively. The left column contains the paths generated for the optimisation goal of travel time, whereas the paths presented in right column are the results of optimising the path length.

The quantitative results shown in Table 5.8 demonstrate that the maximum speed ($0.7ms^{-1}$) is the one most frequently chosen from the set available when the optimisation goal is travel time, whereas lower speeds were selected more frequently when the path length is the optimisation criteria. Additionally, the execution time for 1000 generations in the on-line process (shown in Table 5.8) are generally less than those found for off-line planning (Tables 5.4 and 5.6). This can be explained by the fact that the length of the individuals is generally shorter when performing on-line planning, simply because the robot has advanced closer to the goal and there will likely be fewer intermediate vertices (the number of intermediate nodes for the final paths can be seen in Figure 5.5, 5.7, and 5.8). Table 5.8 also shows that when optimising for travel time, although the maximum speed was frequently chosen, occasionally a faster path could be obtained by reducing speed to avoid more efficiently a dynamic obstacle. In contrast, optimising for minimum path length

necessitated more frequently reversion to lower speeds and consequently these paths were likely to consume significantly less power to drive the robot to its destination.

Table 5.8 Experimental results for the on-line planning. Note that the medians over 100 independent runs are shown, except that the robot speeds are the representatives of a single run from the 100 runs.

environment		cost in terms of		generations to find first feasible path	execution time to find first feasible path (s)	execution time for 1000 generations (s)	robot speed (ms^{-1})
		path length (m)	travel time (s)				
optimise for travel time	1	419	616	38	1.13	26.6	0.7, 0.6, 0.7, 0.7, 0.7
	2	358	555	43	1.61	35.6	0.4, 0.7, 0.7
	3	300	428	58.5	3.72	56.5	0.7, 0.7, 0.7, 0.7
	4	372	531	93.5	9.09	81.3	0.7, 0.7, 0.7
optimise for path length	1	407	893	38.5	1.16	25.9	0.7, 0.3, 0.3, 0.3
	2	317	829	39	1.43	35.2	0.4, 0.3, 0.3
	3	300	552	58	3.56	56.2	0.7, 0.3, 0.7, 0.7
	4	366	817	95.5	9.63	80	0.6, 0.3, 0.3, 0.5, 0.3, 0.3

5.3.4 Comparison

This section evaluates the performance of the vertex++ planner in its comparison with 9EP/N++ (Smierzchalski and Michalewicz 2006), and using optimisation goals of path length and travel time for the on-line planning process described in the previous section. Although only results for on-line planning are presented, the comparison is also valid for off-line planning process, as each planner uses the same algorithm for their respective on-line and off-line planners. The system parameters of vertex++ were kept as close as possible to those of 9EP/N++ and are shown in Table 5.9. Note that the operator for each generation was selected on a random basis from those available (Smierzchalski and Michalewicz 2000 and 2006) and the same evaluation function for feasible paths (see section 5.2.4) was used for both planners. The system parameters for the vertex++ planner are shown in Table 5.2.

Table 5.9 System parameters for the 9EP/N++ planner. Note that the weight for distance (or time) is 1 if the optimisation is distance (or time), otherwise it is 0. One of nine operators is selected randomly for each generation.

population size	rate						weights				safe distance (m)
	Mutate_1	Mutate_2	Insert_ Delete	Delete feasible nodes	Delete infeasible nodes	Speed mutation	clearance	distance	smooth	time	
30	0.3	one gene	0.6	0.1	0.3	one gene	1	1/0	0	0/1	1

The experiment was arranged for two separate optimisation criteria, travel time and path length, and the path quality and execution time were evaluated for the search process until no improvements in fitness were detected for the best individual over 300 consecutive generations. For each of the two algorithms over 100 independent runs were conduced for the same planning problems and the median values calculated (as mean values are often distorted by extreme experimental data for small sample sizes).

The first experiment conducted was to optimise the travel time for the environments. Figure 5.9 illustrates the changes in path cost of the best individual for each generation before obtaining the first feasible path. The quantitative results are shown in Table 5.10.

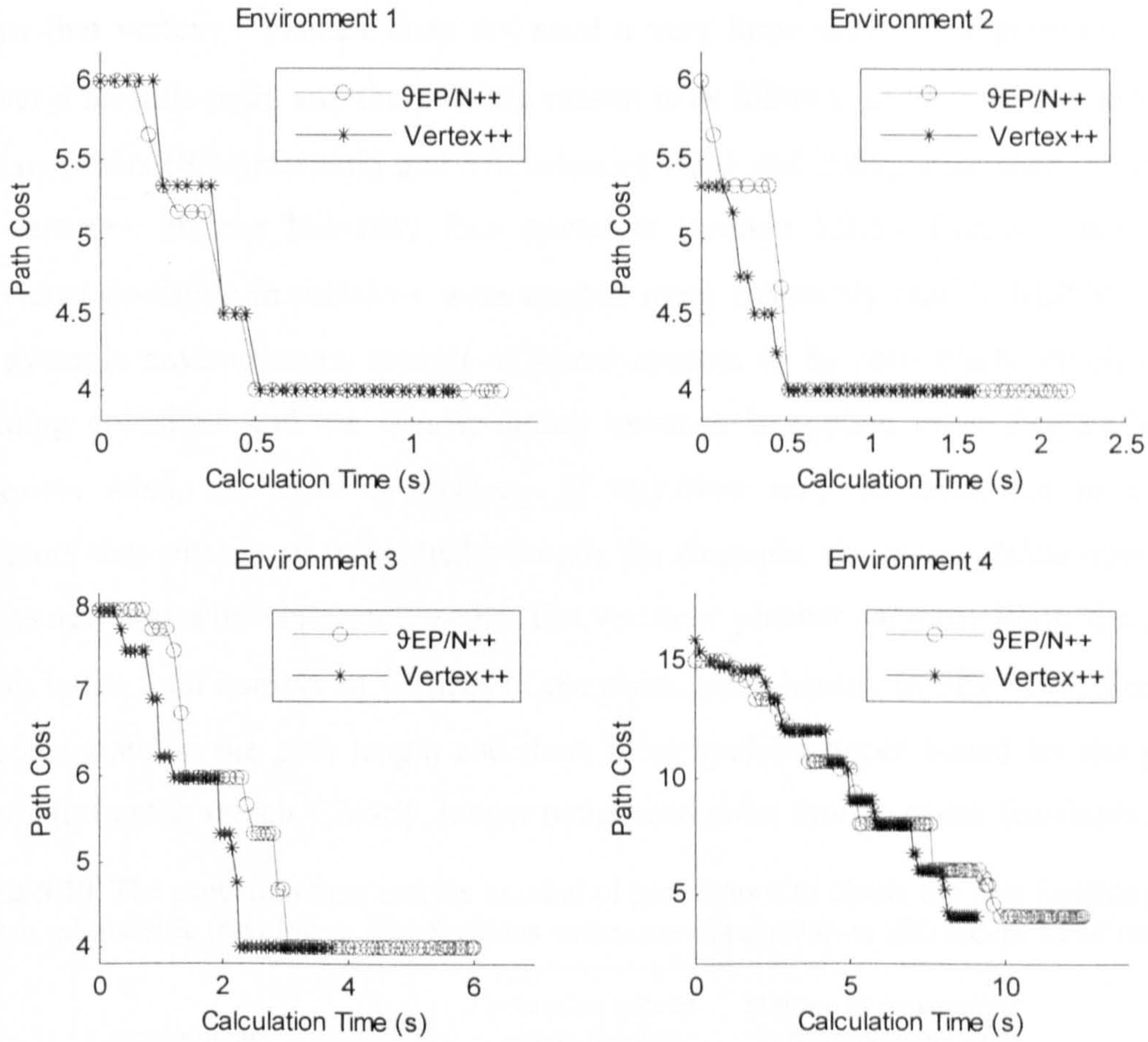


Figure 5.9 The path cost for the best individual of each generation obtained for both planning algorithms with the optimisation goal of minimising travel time for the first evolutionary phase (containing all infeasible paths) during the on-line process. Note that the medians of the path costs over 100 independent runs are shown.

The improvements achieved by vertex++ in terms of execution time can be observed in both Figure 5.9 and Table 5.10, whereas similar path qualities were obtained at the end of the first evolutionary phase during which only infeasible paths exist. However, the number of generations required by vertex++ to obtain the first feasible path is generally larger than that needed by ϑEP/N++. This is probably because in vertex++ the search space constrained to the vertices of the obstacles, whereas in ϑEP/N++ free space dominates for making it relatively easy to obtain a feasible path. In those environments that contain far more space that is unoccupied than that containing obstacles, it would perhaps be expected that the difference in the number of generations required would be substantially greater than that actually found. Further evidence can be seen in the experimental work reported in chapter 6. However, it

seems that vertex++ planner does not need a very large number of generations to deliver a feasible path, and the possible reason is as follows. In 9EP/N++, a total of nine operators (Smierzchalski and Michalewicz 2000 and 2006) were used, whereas the vertex++ planner has only four operators (section 5.2.5). Consequently, the individual operators in vertex++ were applied more frequently than in 9EP/N++. In the dynamic environments, control of speed appears to be particularly efficient in avoiding collisions and the corresponding operator is applied more frequently in vertex++. Also, the time inefficiency of 9EP/N++ may be attributed to those operators that enlarge an individual's length, for example, the *insert_delete* operator inserts new nodes into infeasible paths. The vertex++ planner naturally limits the path length to the total number of vertices of the obstacles, whereas, in 9EP/N++, there is no constraints on the path length and there is no specific upper bound for the path length during the search. Clearly, longer paths need more time to check feasibility.

Table 5.10 The execution time and the number of generations to obtain the first feasible path when minimising travel time. The medians values are obtained over 100 independent runs.

environment	planning method	execution time to obtain the first feasible path (s)	number of generations to determine the first feasible path
1	9EP/N++	1.28	30
	Vertex++	1.13	38
2	9EP/N++	2.16	39
	Vertex++	1.61	43
3	9EP/N++	6.02	66.5
	Vertex++	3.72	58.5
4	9EP/N++	12.5	93
	Vertex++	9.09	93.5

Figure 5.10 shows the results obtained by the two approaches in the second phase as feasible paths are evolved to optimal or near optimal paths. The results to find the final path is summarised in Table 5.11, and are obtained from both evolutionary phases.

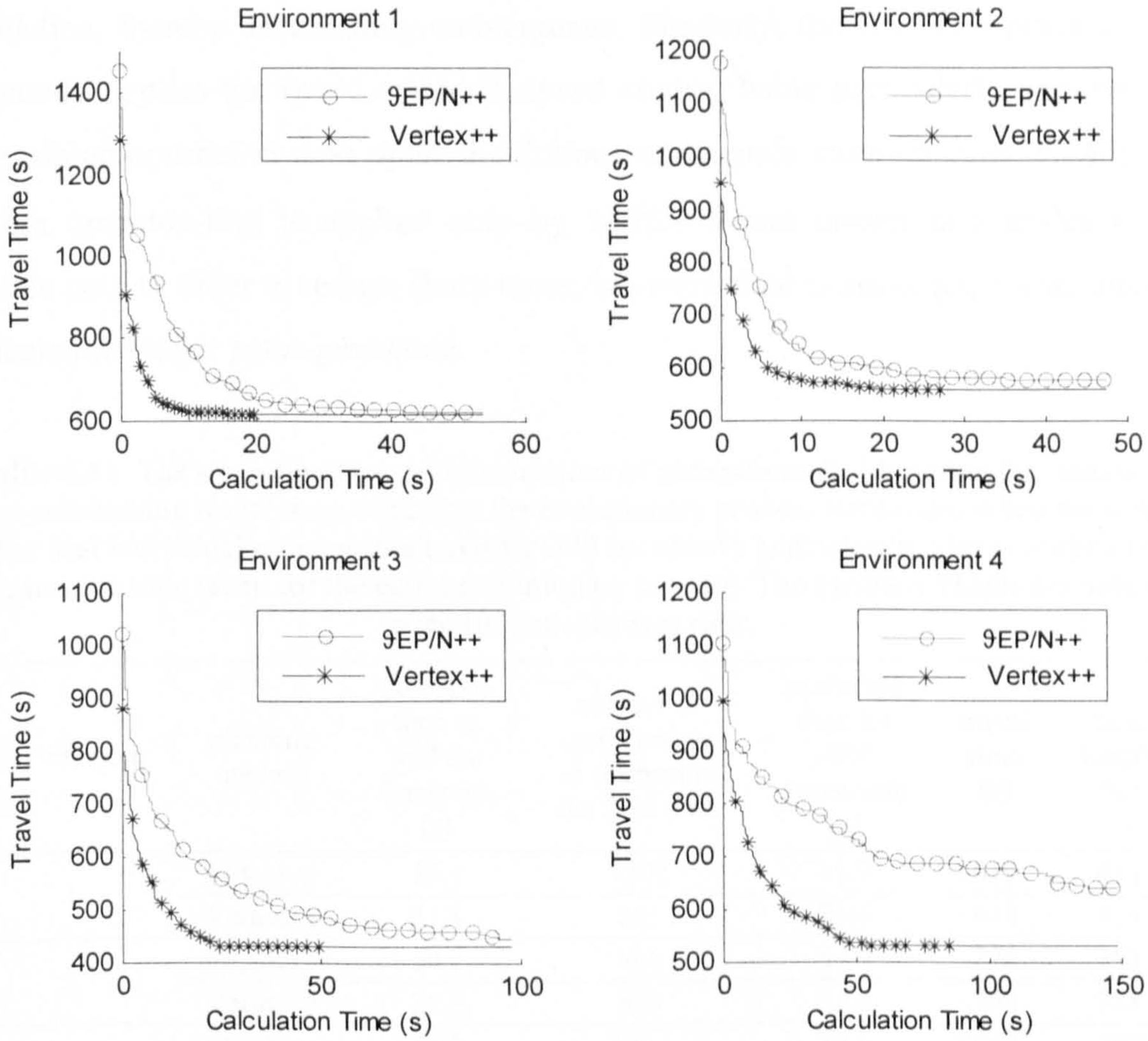


Figure 5.10 The path cost of the best individual of each generation by both planning algorithms with the optimisation goal of minimising travel time during the second evolutionary phase (containing all feasible paths) during the on-line process. Note that the medians of the path costs for 100 independent runs are shown. The evolutionary process for each run terminated when the fitness of the best individual remained unchanged for 300 successive generations.

From Figure 5.10, it can be seen that a significant improvement in execution time is exhibited by the vertex++ planner when compared with ϑEP/N++ during the second phase. The path quality obtained by the vertex++ planner remains similar to that of ϑEP/N++ for environments 1, 2, and 3, while a better path was generated by the vertex++ planner in environment 4. As ϑEP/N++ operates in the entire search space of the environment, the first feasible path obtained is likely to be far from optimal, but, as only a few feasible paths exist when the search space constrained to the vertices of the obstacles, the first feasible path generated by vertex++ is likely to be shorter. As the rate of application of the crossover operator is relatively high in the vertex++ planner, there will be a faster propagation of elite genes among the

population, thereby accelerating convergence. Similarly, the vertex++ planner more frequently applies the speed operator; speed control being particularly important in determining optimal or near optimal solutions in dynamic environments. Finally, the smooth operator that is applied only by 9EP/N++ and inserts new nodes into a feasible path in order to reduce sharp turns, but extra time is subsequently required to evaluate the longer paths generated.

Table 5.11 The execution time and the number of generations to determine the final path when minimising travel time. Note that the evolutionary process terminates when the fitness of the best individual remains unaltered for 300 successive generations. The execution time indicates the time taken for the entire evolutionary process. The medians values are obtained over 100 independent runs.

environment	planning method	execution time to find the final path (s)	number of generations to determine the final path	execution time for 1000 generation (s)	travel time (s)	path length (m)
1	9EP/N++	55.1	1202	45.7	619	411
	Vertex++	21.4	805	26.6	616	419
2	9EP/N++	50.1	908	54.5	576	357
	Vertex++	29.9	839	35.6	555	358
3	9EP/N++	105	1226	82.9	443	307
	Vertex++	53.1	939	56.5	428	300
4	9EP/N++	163	1241	129	641	401
	Vertex++	98.1	1103	81.3	531	372

Table 5.11 shows the reduction in execution time by the vertex++ planner in comparison with 9EP/N++ for the entire search process. Overall, a reduction of around 50% was obtained for the four environments, although a modest reduction in the number of generations was achieved. Clearly, this implies that the average time taken by 9EP/N++ for each generation is longer, and this can also be seen from the execution time required by the two algorithms for 1000 generations. Two possible reasons may explain the calculation time difference: one is that one or more problem-specific operators of 9EP/N++, such as swap and smooth, are rather time consuming, whereas the simple operators (such as crossover, mutation, and speed mutation) that were also applied to feasible paths by vertex++ took rather less time to execute; the other is the number of genes in the path has been increased by the smooth operator increasing the application time of subsequent operators. Compared to 9EP/N++,

Table 5.11 shows that minor improvements can be seen in terms of travel time by the vertex++ planner for environments 1, 2 and 3, whereas the vertex++ planner was generally able to generate a much better path for environment 4. For path length, a longer path was generated by the vertex++ planner for the environments 1 and 2, but a shorter one for environments 3 and 4. This does not imply the vertex++ planner is worse than 9EP/N++, as the optimisation goal in the experiments was travel time.

Figure 5.11 and Table 5.12 report the results obtained by the two planning algorithms when the minimisation of path length was the sole optimisation goal. Compared with 9EP/N++, a reduction on the execution time for the first evolutionary phase was observed for the vertex++ planner. Again, the average time for each generation required by the vertex++ planner is relatively short as the number of intermediate nodes of a path was naturally limited to be no more than the number of the vertices of the obstacles. The reduction in the number of generations required by vertex++ for environments 3 and 4 may be due to the application rate of repair operator which is relatively high compared with 9EP/N++, aiding the rapid transformation from infeasible to feasible paths. However, this effect is not so apparent in short search processes as found in environments 1 and 2

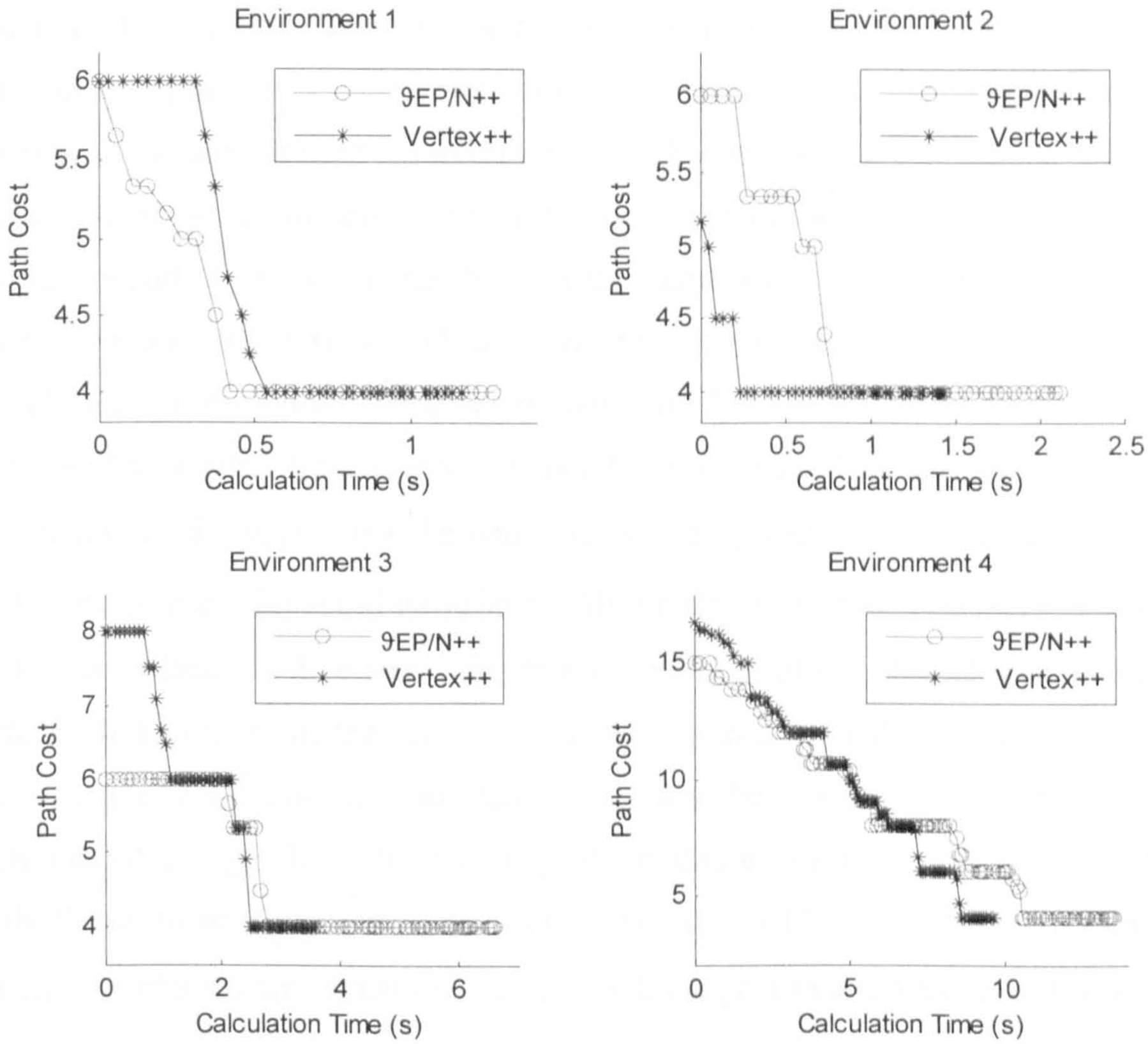


Figure 5.11 The path cost of the best individual of each generation produced by both planning algorithms with optimisation goal of path length for the first evolutionary phase (containing all infeasible paths) during the on-line process. Note that the medians of the path costs over 100 independent runs are shown.

Table 5.12 The execution time and the number of generations to obtain the first feasible path when minimising path length. The medians obtained over 100 independent runs are shown.

environment	planning method	execution time to obtain the first feasible path (s)	number of generations to determine the first feasible path
1	ϑEP/N++	1.25	29
	Vertex++	1.16	38.5
2	ϑEP/N++	2.13	39
	Vertex++	1.43	39
3	ϑEP/N++	6.6	78.5
	Vertex++	3.56	58
4	ϑEP/N++	13.5	103
	Vertex++	9.63	95.5

The results for the second evolutionary phase is illustrated in Figure 5.12 and the quantitative measures for the complete process are presented in Table 5.13. The

reduction of execution time for vertex++ is evident, with the most significant reduction being in the first environment (approximately 67%), whereas the smallest reduction is in environment 4 (around 46%). The number of generations was also reduced, but not so significantly. Again, the simpler set of genetic operators available to vertex++ and the natural upper limit to the number of intermediate nodes yield a reduction in execution time and the reduction in the number of generations is probably due to the search space being constrained to the set of the vertices of the obstacles. The vertex++ planner was generally able to produce the final paths with same quality as ϑEP/N++, but the paths generated by ϑEP/N++ for environments 3 and 4 were shorter. To avoid collisions with moving obstacles, ϑEP/N++ is able to select nodes within the free space far from the vertices of the obstacles to produce a shorter path. However, as the vertex++ planner was constrained to the vertices of the obstacles, the collision-free path generated may be longer due to geometrical constraints. Although the optimisation goal for this experiment was set to be path length, the corresponding time is also shown in Table 5.13, where it can be seen that the vertex++ planner was generally able to reach the goal in less time than ϑEP/N++.

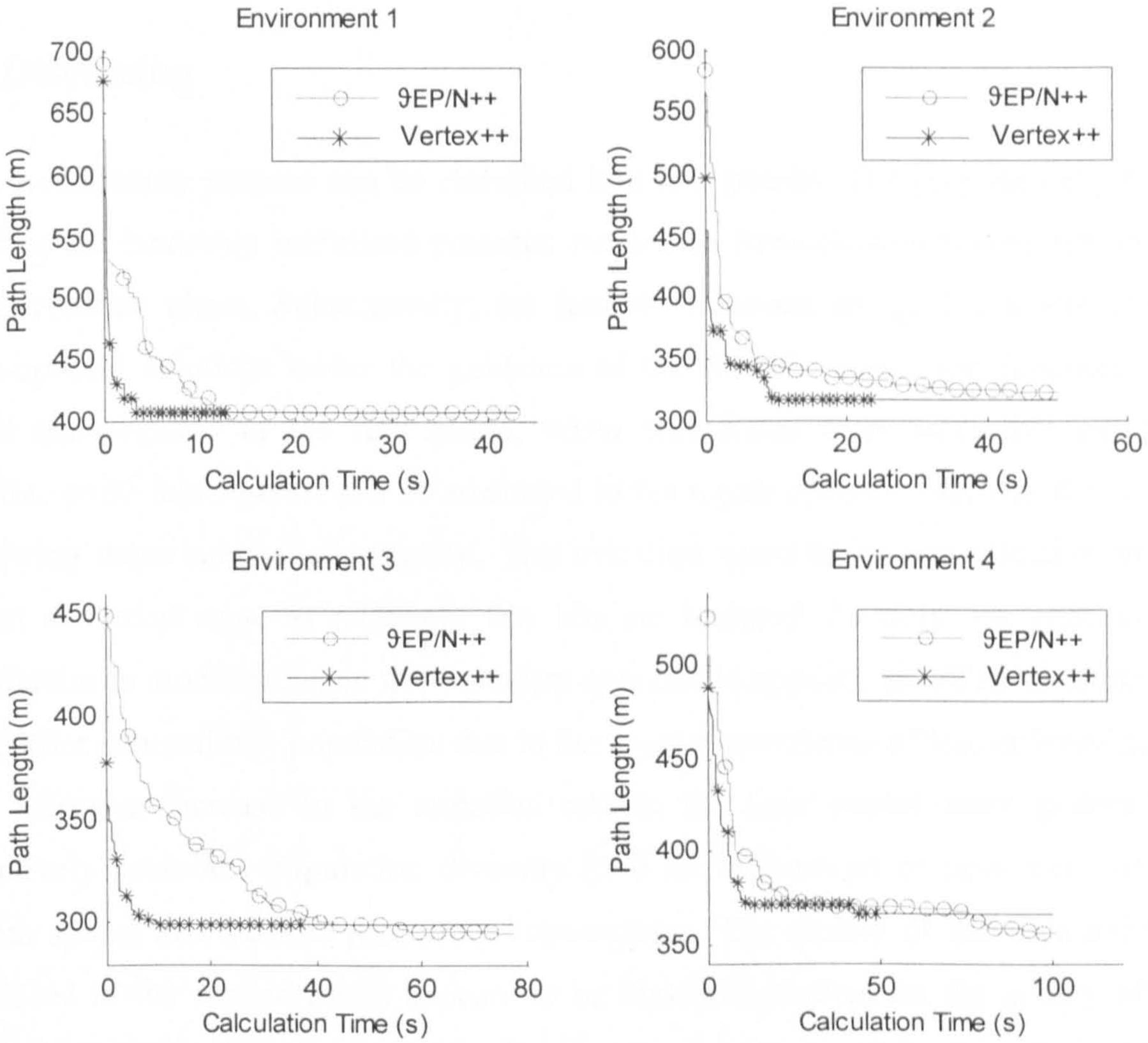


Figure 5.12 The path cost of the best individual of each generation produced by both planning algorithms with the optimisation goal of path length for the second evolutionary phase (containing all feasible paths) during the on-line process. Note that the medians of the path costs over 100 independent runs are shown. The evolutionary process for each run terminated when the fitness of the best individual remained unchanged over 300 successive generations.

Table 5.13 The execution time and the number of generations to determine the final path when minimising path length. Note that the evolutionary process terminates when the fitness of the best individual remains unaltered for 300 successive generations. The execution time indicates the time taken for the entire evolutionary process. The medians are obtained over 100 independent runs.

environment	planning method	execution time to find the final path (s)	number of generations to determine the final path	execution time for 1000 generation (s)	travel time (s)	path length (m)
1	ϑEP/N++	44.4	977	45.6	912	407
	Vertex++	14.6	554	25.9	893	407
2	ϑEP/N++	52.7	974	54.4	841	322
	Vertex++	25.4	722	35.2	829	317
3	ϑEP/N++	85.0	1057	82.3	674	295
	Vertex++	42.5	756	56.2	552	300
4	ϑEP/N++	115	934	125	888	356
	Vertex++	61.7	754	80	817	366

5.4 Discussion

The evolutionary process can be classified into two phases. The proposed algorithm evolves the randomly initialised potential paths into free-collision trajectories in the first evolution phase. Subsequently, the feasible solutions are guided to optimal or near-optimal solutions under the guidance of the feasible evaluation function. The rapid convergence in the first phase, when conversion from infeasible paths to feasible paths takes place, can be attributed to the repair operator that was developed following experimental observation. The mutation operator is less effective in the initial evolution stage as relatively few bits are mutated. As only one gene in the population is modified when the mutation operator is applied, its effect is diluted in the earlier generations' population due to the greater prevalence of longer individuals. The effective increase in the mutation rate in the later evolutionary generations effectively promotes population diversity (and so exploration of new areas of the search space) and inhibits premature convergence. The quality of the final solution produced in the second phase appears to be highly dependent on the quality of the initial individuals that are supplied following the operations of the first phase. If the initial supply for the second evolution phase is sparse (in that all the necessary building-blocks for the global optimal solution are not present), the process may be led into a local minimum. The higher mutation rate apparent in later evolutionary steps promotes the diversity by modifying the inherited building blocks; however the mutation is not sufficiently dominant in the process to necessarily avoid the GA becoming trapped in a local minimum. The vertex++ planning algorithm has been demonstrated as being capable of generating an optimal or near-optimal path for the robot in a relatively short time compared with 9EP/N++ which is also able to operate in environments containing dynamic obstacles. The search space has been constrained to the vertices of the obstacles and reduces the number of generations required by 9EP/N++ to generate an optimal or near optimal solution. Furthermore, it appears that the application of a small number of operators that performs only simple tasks is able to improve the performance in terms of execution time. The current implementation has been carried out in MATLAB and a significant improvement (reducing the

calculation time by a factor of five to ten times) is to be expected when executed in a compiled language such as C.

5.5 Conclusions

A number of experiments have been presented for a range of assessment criteria in order to verify the capability of the vertex++ planner using navigation problems in dynamic environments. The planner searches for the optimal solution in a space limited to the vertices of the static and dynamic obstacles, rather than requiring the entire environment. By modelling the moving obstacles involved in the robot environment, it has been shown to be possible to determine a collision-free path between two specified locations in an off-line fashion. On-line planning is appropriate when the motions of the obstacles change from those values known at the time of generation of the original plan computed off-line.

From the experimental results, it can be seen that a suitable solution can be produced by the proposed planner for relatively complex planning problems in a reasonable time. Where travel time is crucial it was found that, in the environment considered, full speed can be applied in most segments of the trajectory. However, environments could easily be conceived in which operating at full speed throughout the trajectory would result in a much extended path due to the robot's circumnavigation of dynamic obstacles. If minimising the path length is more important, for example to conserve energy, lower speeds can be planned and this is likely to result in a shorter overall path.

One of the next objectives of this research is to relax the assumption that details of the obstacles' motion changes need to be completely known and that there is always sufficient time for re-planning without significant impact on robot travel times. The investigations in this area are discussed in the following three chapters. The next chapter introduces the waypoint-based navigation system for stationary environments. This system was augmented to include the navigational ability to interact with moveable objects, which is described in chapter 7. A generalised navigation system

that allows paths to be planned between any two pairs of points in the environment is described in chapter 8.

Chapter 6

WAYPOINT-BASED NAVIGATION IN STATIC ENVIRONMENTS

The previous two chapters contribute to the development of the planners by enhancing path planning performance. However, both of the planners rely heavily on world models even though a new plan could be generated for an environment containing moving obstacles. In unknown environments, the deliberative model-based approaches are not appropriate and earlier knowledge of the environment need to be accumulatively acquired from navigation that has been performed reliant on sensory information. Although navigation quality may be improved by a detailed model of the environment, its construction (where possible) involves considerable complexity in terms of execution time and memory usage. Any errors arising during map building may give rise to poor performance or complete failure in subsequent navigation tasks. Reactive navigation systems, on the other hand, afford real-time robust reaction in a varying world, and are tightly coupled to the available sensory information. However, their movement (at a non-local level) is unlikely to be optimal and may cause the robot to become trapped in local minima. A new hybrid system is developed and presented in this chapter with the aim of addressing these issues by combining the advantages and overcoming the drawbacks of the deliberative and reactive approaches. The work in this chapter has been submitted to Robotica, and

part of the work was also presented at the 6th World Congress on Intelligent Control and Automation in 2006.

The new work presented here employs a reactive navigation system to determine and navigate between suitable locations, termed waypoints, in the robot's environment space. The waypoint in the proposed system is defined as a location where the robot changes its behaviour as a result of reacting to the perceived environment. However, due to the ill-defined boundary between the primitive behaviours in the reactive implementation adopted, heading and sensory information are used in determining the locations of the waypoints. Specifically, when under reactive control, a location is marked as a waypoint if the robot deviates from the path it is currently following in response to the presence of one or more detected obstacles. A more detailed explanation of the definition of a waypoint and a summary of the use of waypoints in other navigation systems are described in section 6.2. A genetic based approach was developed for deliberative planning that determines the sequence of previously-discovered waypoints that need to be followed to satisfy future mobile robot tasks. The deliberative navigation system also manages exploration to allow additional waypoints in the robot's environment to be discovered if time permits. The exploration is directed by the known waypoints to investigate relevant and promising areas. This simplification of the required representation of the environment reduces considerably both the computation and memory requirements without significantly affecting navigation performance. Moreover, the method does not require *a priori* knowledge of the environment and, in unseen environments, the recorded waypoints can be used to facilitate escape from certain obstacle configurations that would normally trap robots under the control of a reactive navigation system.

The following section presents the related work, section 6.2 outlines and explains the new waypoint system, section 6.3 explains the planning algorithm developed, section 6.4 describes how the waypoint navigation method is able to escape from certain obstacle shapes, section 6.5 presents the experimental procedure, section 6.6 shows the experimental comparison of the navigation methods, section 6.7 presents further detailed investigations of the waypoint navigator in its application to complex

environments, and section 6.8 compares the proposed system with several recent hybrid architectures described in the literature.

6.1 Related work in hybrid systems in static environments

The earlier attempts by a number of authors to implement navigation systems using a hybrid architecture were introduced in section 3.3. The more recent work found in the literature largely use a three-layered architecture (following the earlier work of Atlantis (Gat 1991a and 1991b), SSS (Connell 1992), and 3T (Bonasso *et al.* 1997)) arranged in a hierarchical fashion.

The nested-loop architecture (Santos, Castro and Ribeiro 2000) consists of three loops, namely reflexive, reactive, and functional, arranged in such manner that a loop of a lower level of abstraction is nested into the immediately higher one. In each loop the information follow is from sensor input to generated actions. Emergency situations, such as imminent collision and actions to escape traps, are handled in the reflexive loop and a direct command is given immediately on their detection. The reactive loop performs local navigation according to the path generated in functional loop, which is responsible for determining the strategy used in local movements. The authors also provided a comparison with other architectures in the text that described the new method, but no experimental comparison was reported. Although such a system may have benefits for navigation in complex environments, most actions generated are not tightly linked to the raw sensory information but are directed by the functional loop and consequently the computational load is increased when providing local demands.

The hybrid architecture developed by Aguirre and González (2003) consists of three layers, which, in order of hierarchy were: planning, executive, and control layer. The generation of a safe path with minimum cost is carried out in planning layer based on the topological map of the environment using either Dijkstra (Cormen *et al.* 2001; Latombe 1991) or A* (Hart, Nilsson and Raphael 1968; Murphy 2000). The middle layer, the executive layer, determines which primitive behaviours are activated to

accomplish the plan generated, and has the additional function of monitoring the robot performance so that potential failures can be identified. Activated behaviours are combined to produce a single action in the lowest layer, the control layer, which is responsible for the motion control. The sensory information is transformed to produce suitable inputs to the mapping unit, the executive layer and the control layer, whereas the abstracted map is shared between the planning and executive layer. The hybrid system was tested in an office-like environment, but its suitability to other environments is unclear. Also, no comparison with other architectures to further evaluate the developed system is given and in the absence of the executive layer, (which co-ordinates the individual behaviours), the success when navigating to the goal and how the robot behaves are not reported.

The set of agents in the hybrid system described by Muñoz-Salinas *et al.* (2005) for navigation in office-like environments was organised into three layers: deliberative, execution and monitoring, and control. According to the mission assigned by an external operator, the deliberative layer generates a path consisting of a sequence of rooms and corridors that the robot needs to navigate in order to reach the goal. A* search (Hart, Nilsson and Raphael 1968; Murphy 2000) was used to generate the navigation plans using a topological representation of the environment either supplied *a priori* or produced autonomously by exploration. The local navigation for partial plans was managed by the monitor agent (in the execution and monitor layer) that was able to perform transformation to an appropriate skill (for example find a door in room) in order to accomplish the sub-goal. Skills were able to activate a set of fuzzy behaviours to achieve the sub-goal. The lowest layer, the control layer, was further subdivided into the behavioural and hardware levels. In behavioural level, a vision agent was used to identify landmarks, and a navigation agent implemented the specified behaviours. Two agents are contained in the hardware level: one to deal with the communication between the agents and the second to operate the pan-tilt unit on which a camera for detecting landmarks is mounted. The hierarchical layers were connected in a manner that allowed bi-directional communication of navigation tasks to be ordered in a top-down manner and errors from the lower layer to the upper layer. The world model was shared among all agents and whether navigation would be

successful in absence of the higher-level guidelines remains unclear. Compared with the nested loop architecture of Santos, Castro and Ribeiro (2000), there appears to be no agent to deal with emergent situations. No comparison with other hybrid systems was carried out and so the advantage of this system with respect to the existing alternatives is not clear. The experiments, being limited to office-like environments, seem to restrict its application and modification may be required for application to other environments.

A hybrid control architecture for navigating a robotic fish was introduced by Liu, Hu and Gu (2006). Three layers, *cognitive layer*, *behaviour layer* and *swim pattern layer*, organised in a hierarchical manner, comprised this architecture. The central layer contained a set of behaviours realised by fuzzy-logic controllers with individual behaviours being activated according to sensor states and combined through the behaviour coordination component to determine a particular swim pattern. The lowest layer, *swim pattern layer*, converts the specified swim pattern into the control of the individual joints of the robotic fish. The cognitive layer produced a set of actions to lead the robotic fish from the initial configuration to the goal configuration, with the output being the parameters for the coordination module in the behaviour layer rather than the actions themselves. In this way, the emergent behaviour for each movement is largely influenced by the output of the cognitive layer. However, any errors generated in the planning process (possibly due to inaccuracies in the representation of the environment) may produce ill emergent behaviour. Additionally, no clear strategy to construct the world model was reported in the paper.

Maaref and Barret (2002) combined global planning and local reactive methods into a hybrid system. In unknown environments, the robot relies on a set of fuzzy behaviours to move, whereas navigation could be accelerated by fast tracking the path generated by the global planner if the partial or entire world model were available. In order to avoid deadlock problems often found in purely reactive systems, separate strategies to coordinate the behaviour have been developed for convex and concave obstacles. If a convex obstacle is detected, goal seeking behaviour and a behaviour to move the centre of the collision-free space will be combined. Escaping the concave

obstacles relies on the coordination of behaviours used for convex obstacles and wall following behaviour. In order to follow the generated path and avoid unmodeled obstacles, a fuzzy decision module is used to generate the final action by taking into account the commands from a virtual robot moving in the known environment and the robot in the actual environment. However, a longer execution time in conjunction with a high-resolution world is needed to generate the final action of producing a sufficient accurate output for the virtual robot at each step. Such a high quality model requires the allocation of significant memory, otherwise positional errors may accumulate to such an extent that localisation with respect to the real robot becomes too poor for practical purposes. In the described architecture, the generation of behaviours and the coordination between activated behaviours are not directly guided by the deliberative module, hence basic navigation is achievable in unknown environments.

Wang, Yong and Ang Jr. (2002) proposed a hybrid approach to navigation in an indoor environment. Global path planning was achieved using the *distance transform* that was based on a known grid map, whereas the local navigation was guided by a *potential field.* The navigation system was implemented for known environments represented by a grid map, where each cell was assigned a distance from the goal location by the distance transform method that propagated the distance value relative to the goal location. Cells occupied by obstacles were labelled with a very high value relative to unoccupied ones. A collision-free optimal path could then be generated by the steepest descent method based on cell values in the distance map. In response to the environmental changes, local navigation was directed by a potential field to follow a set of sub-goals that are points with the smallest distance between the pre-planned path and the circumference of a circle (which may not be in sensory range) centred on the robot. The experimental study demonstrated that the local minima that often results from the potential field approach can be overcome with the aid of global planning. To construct an *a priori* grid map and a potential field is computationally expensive. In addition, the method to determine the radius of the circle in generating the sub-goal is not well defined and needs to be obtained empirically. Also, whether

the robot can actually escape from local minima is questionable when it enters an unknown environment and global planning is infeasible.

A hybrid architecture was proposed in by Li *et al.* (2004) for indoor navigation. This system consists of 10 components which can be classified into four categories. The deliberative components, *pathplanning*, generate a path according to the target specified through *userinterface*. The *obstacledetecting* component, which belongs to the monitor type, discovers the obstacles *en route*. The three components of reactive type, *goto*, *orienting*, and *obstacleavoiding*, perform basic navigation to rotate and head towards the goal (or sub-goals if any) while avoiding any obstacles detected. The fourth type of components, called hardware abstraction, contains *lasersever* and *robotbasesever*, which function as interfaces from other components to the laser sensors and actuator respectively. No specific algorithm for each component was described in the paper. This architecture clearly lacks components for map building, localisation and exploration, and the system was not evaluated by comparison with any other system.

An architecture built principally on the behaviour-based control was proposed by Na and Oh (2003), in which environment was classified by a neural network into one of 16 situations. The output of this unit then selects an appropriate behaviour from a set of neural network behaviours to control the steering angle and to configure a potential field to generate the speed command. Although the authors described the architecture as hybrid, such description is probably not appropriate as it did not contain a deliberative component. It is clear that the absence of proper management in the navigation is one of the system's major disadvantages. Moreover, the experimental studies were performed in only indoor environments and no comparison to other systems was carried out.

Other approaches whose relationship to the current work is not as close as those already mentioned, include the contribution made by many of them, Santos, Castro and Ribeiro (2000), Aguirre and González (2003), and Muñoz-Salinas *et al.* (2005), who all used abstract representations of environments rather than a detailed model.

The disadvantages of constructing a detailed model are that it can be difficult to construct in many applications, it is often unnecessary for satisfactory navigation in most practical applications, calculation time is long and considerable memory is required for storage, and it is difficult to incorporate dynamic changes into the world model. Most architectures (Aguirre and González 2003; Li *et al.* 2004; Maaref and Barret 2002; Muñoz-Salinas *et al.* 2005; Wang, Yong and Ang Jr. 2002) were evaluated only in indoor environments which tend to be well structured. Consequently, the performance of such systems in outdoor unstructured environments is unknown. Furthermore, none hybrid systems were compared by experimental study and only the nested loop architecture was compared to other earlier systems (and even then by discussion rather than by experimental evaluation).

6.2 Waypoint navigation system

The waypoint navigation system shown in Figure 6.1 is composed of three main units.

1. The *reactive unit* provides reactive navigation for the robot, which, in the current work is achieved by a frequency-table based learning technique developed by the Electronic System Design Group at Loughborough and that is able to generate a decision tree (Mulvaney *et al.* 2005; Swere, Mulvaney and Sillitoe 2004). Although other reactive approaches may be used in this unit rather than the decision-tree based method, the main reason for using this method in the current implementation is that it is fully available both in source code and as an executable.
2. The *knowledge base* contains the robot's acquired knowledge of the environment represented as a set of waypoints and paths between waypoints. To construct a detailed world model requires high quality sensors, longer processing time and a large memory capacity (see section 6.1). The approach assumes that a detailed model is unnecessary for every navigation task and only a set of points relevant to the navigation task are recorded, thus providing a highly abstracted map for the navigation task between specified locations.
3. The *deliberative unit* is a high-level control unit used for navigation when existing knowledge of the environment is available. It contains three elements, namely localisation, exploration and planning. This unit is included as late navigation

through the same environment can be improved by cognitive reasoning using previous navigation experiences (resulting from either exploration or previous navigation tasks). Also, the knowledge of the environments presented as waypoints can aid the solution of the localisation problem for the robot.

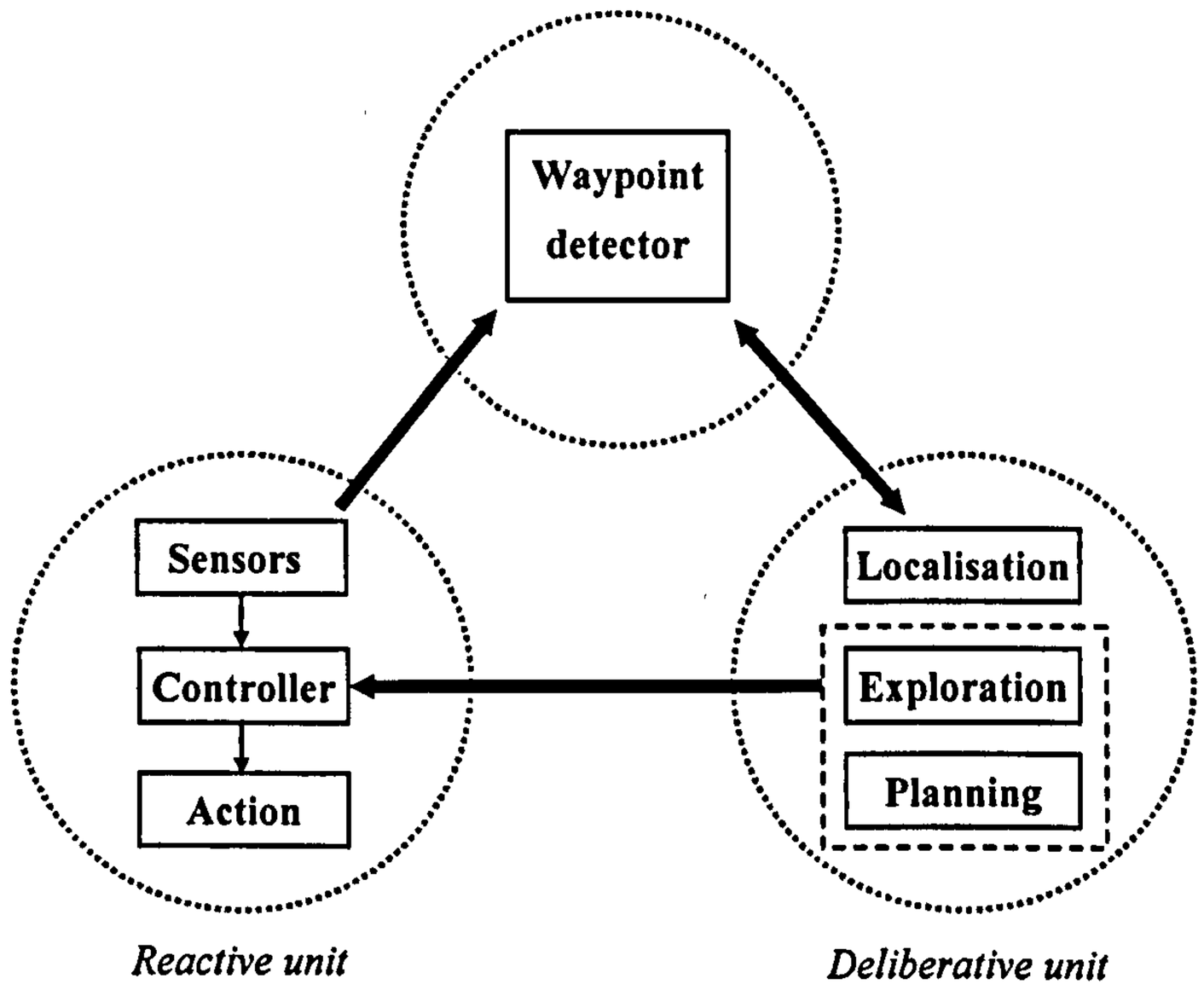


Figure 6.1 Block diagram of the waypoint navigation system.

The three separate units in Figure 6.1 reflect three different internal states. The reactive unit responds to the current state input from the sensors. As the reactive unit operates on the instant state, much processing on the raw sensory data is not necessary and should be kept minimum to reduce the response time. The previous states are selected and stored in the knowledge base. As not every detail from previous actions is valuable for the future navigation, data recorded are limited by removing previous records either periodically or dynamically if rarely used. Clearly the memory capacity and the decisions regarding the deletion of data may influence the performance in future navigation. Section 6.2.2 discusses further how suitable information for storage is determined. The deliberative unit predicts and plans future navigation actions by reasoning using previous experiences, with the internal states generated in the unit providing a future direction for the robot to follow. Note that some modules (such as localisation) in a unit may represent state other rather than the

main state reflected by the unit. Although the current action is not directed by the historical knowledge in the system (that is, there is one-way communication between the reactive unit and knowledge base), the behaviour activated by the current stimuli has been obtained through the previous training (off-line learning) and navigation experience (on-line learning).

The detailed description of each of the units is given in the following subsections, but a brief overview of the general operation is appropriate here. Assuming the waypoint navigation system has no previous knowledge of its environment, it attempts to reach a goal under reactive control, while continuously transmitting sensor information and its current action to the knowledge base. Here, the waypoint detector selects and records suitable locations. A point in the robot's environment is marked as a waypoint when the robot needs to deviate from its current path due to the presence of an obstacle. These waypoints are entered into the knowledge base and are then available to the deliberative unit for use in exploration and planning. Suitable waypoints for the environment can be obtained following off-line simulation or generated on-line either as a result of executing previous navigation tasks or by purposely invoking exploration. Note that information transfer between the waypoint detector and the localisation unit is two-way, since localisation needs to access to the stored waypoints in order to instruct the waypoint detector to remove duplicate entries. The solution to a navigation task is presented by the planner as a path defined by a sequence of waypoints.

6.2.1 Reactive unit

The reactive unit is used in the current system to investigate new areas of the robot's environment, whether this is towards a given goal or to explore previously uncharted regions. In the waypoint navigation system, the deliberative unit provides only a sequence of waypoints to follow and the reactive system is required to perform the local navigation between consecutive pairs of waypoints.

The reactive approach adopted in this hybrid system was developed previously in our Research Group at Loughborough University and was introduced briefly in section

3.2.2. Further information on this reactive system can be obtained in Mulvaney *et al.* (2005) and Swere, Mulvaney and Sillitoe (2004). In Passone, Chung and Nassehi (2006) and Urdiales *et al.* (2003b and 2006), the basic reaction to the perceived environment was achieved by a case-based reasoning approach. A well-known drawback of case-based approaches is that the performance is unpredictable when a corresponding case has not been presented during the training stage. Potential field approaches have been used in a number of navigation systems (Arambula Cosio and Padilla Castaneda 2004; Ren *et al.* 2007; Ren, McIsaac and Patel 2006; Wang, Yong and Ang Jr. 2002). Although potential fields give an elegant solution for navigation, they require the construction of an artificial field, whereas decision tree approaches does not require additional functionality in the coordination of the set of primitive behaviours presented but other systems (Aguirre and González 2003; Liu, Hu and Gu 2006) require. It should be noted that, after training, the robot can autonomously move between any pair of locations without any intervention or extra guidance. When the navigation task was not presented in the training environment (or where additional optimisation criteria is required later) the resulting movement may not be optimal, but demonstrates the ability to continue navigation with degraded quality.

6.2.2 Waypoint knowledge base

To be able to plan future movements in an autonomous and intelligent manner, a robot requires memory to record where it has already been. One approach that greatly enhances the navigation efficiency is to build a map of the environment. However, most mapping techniques (a short review on mapping approaches was provided in section 3.1.1) involve considerable computational overheads and significant memory capacity that is difficult to constrain as more of the environment is discovered. Since only a small number of points need be recorded, the waypoint technique provides an alternative to mapping with greatly reduced resource requirements. This section first reviews the use of the waypoints in other navigation systems, and then describes the strategy used to determine the waypoints in the proposed system.

A general definition for a waypoint is the end point of a path segment decomposed from the planned path (Murphy 2000). Dixon, Dolan and Khosla (2004) defined the

waypoints as a sequence of locations through which the robot must pass. A number of researchers (Ghaffari *et al.* 2004; Guo 2006; Kim and Shim 2003; Kumon *et al.* 2006; Maalouf, Saad and Saliah 2005; Parasuraman *et al.* 2005) used the waypoint in navigation problems without giving an explicit definition. The navigation task described by Macfarlane and Croft (2003), Shimoda, Kuroda, Iagnemma (2007) and Zhu, Sun and Zhou (2007) was pass through a set of pre-defined waypoints, but no clear description was provided regarding how the waypoints were determined. Berman, Edan and Jamshidi (2003) described a navigation approach for autonomous ground vehicles to move through the locations defined by a set of waypoints that had been pre-determined by a planning algorithm according to the layout of a manufacturing environment. However, neither the definition nor the strategy to determine the locations, with the waypoints was documented in the paper. In addition to the location, Wendt, Irwin and Cressie (2004) incorporated a time parameter in the waypoint representation.

The approaches in the previous paragraph did not provide a specific method to determine the waypoints, whereas the waypoints used in papers introduced below were generated by specified procedures. An algorithm inspired by ant trail following was described in Vaughan *et al.* (2002) and applied to a resource transportation task performed by a robot team. A waypoint indicated a location where a specified event occurs (such as where a resource is received or dropped). The waypoints were classified as either global task-level landmarks (which are not physical), or local waypoints, named as crumbs, that were recorded periodically along the trail from a particular location to a landmark. A crumb's coordinates as well as its distance (in terms of travel time) to the landmark were recorded, together with the name of the event that occurred. This event-driven waypoint determination is different from the approach proposed in this chapter which can be better described as behaviour-driven waypoint identification. In addition, the exploration strategy used by Vaughan *et al.* (2002) employed random exploration, but the exploration in the current work (see section 6.2.3) was directed by the waypoints already discovered. In a layered goal-oriented fuzzy algorithm for motion planning (Yang, Moallem and Patel 2005), a set of intermediate goals, called waypoints, were determined in the uppermost layer of

the planner using readings obtained from long-range sensors. Following the calculation of the direction composed in the waypoint representation (based on the collision-free area in a favourable direction towards the goal), the location of the waypoint itself was determined based on the calculated direction, distance to the detected obstacle from the robot, and robot size. The second layer of the planner used the waypoints supplied by the uppermost layer as sub-goals to direct the navigation, but with the aid of short-range sensory information. The waypoints were generated periodically to deal with the situation where the waypoint as a sub-goal cannot be reached due to environmental changes. In such a case, if the robot does not reach the sub-goal within a time threshold (the time interval between two successive generations of the waypoints), a new waypoint was generated to replace the current sub-goal for the robot to seek. The generated path was reported as being similar to that produced by the visibility graph based approach, as the waypoints were located either along the edges or around the vertices of obstacles. If the time taken to generate a waypoint is relatively long, the robot may take considerable time in an attempt to reach a previously recorded sub-goal that is now unreachable. Consequently, more waypoints will be generated and recorded due to the increased planning frequency (in the first layer), thereby increasing the computation complexity and memory usage, but improving navigation quality. In practice, this trade-off may be difficult to optimise. The authors did not exploit other uses of the discovered waypoints in their paper, such as applying the previously-generated waypoints to help the robot escape from local minima; improving the navigation performance of the future tasks by using previously-generated waypoints to generate a plan for future navigation; the use of discovered waypoint to help the robot in localisation.

In the current work, the selection and recording of waypoints is implemented as follows. A point in the robot's environment is marked as a waypoint when the robot needs to deviate from its current path due to the presence of an obstacle. The currently-adopted reactive control approach uses a combination of several primitive behaviours and exhibits no clear boundary between each elemental behaviour. Therefore, rather than identifying behaviour change, the current method of determining waypoints employs heading change as its cue. During the robot's

movements, when operating under reactive control, should the heading change by more than a pre-defined threshold angle while one or more obstacles are being detected, the geometrical location at which the robot begins to change its heading is recorded as a waypoint. Following the initial identification of a waypoint, two possible new headings are available to the robot, as shown in Figure 6.2. As only one of these two can be investigated immediately, a waypoint is marked as unexplored until the second heading has been taken following a subsequent visit. In contrast, the waypoints used by Vaughan *et al.* (2002) indicate the locations where certain events occur. Although the behaviour is needed to adapt the occurrence of a particular event at the waypoints, those waypoints do not implicitly or explicitly contain information on the obstacle distribution. Similarly, no information about the obstacle locations is reflected in the waypoints selected by the approach in Yang, Moallem and Patel (2005), as a waypoint is determined according to the free space and goal. Therefore, the waypoints determined by either those approaches are not able to offer alternative paths to circumnavigate obstacles encountered.

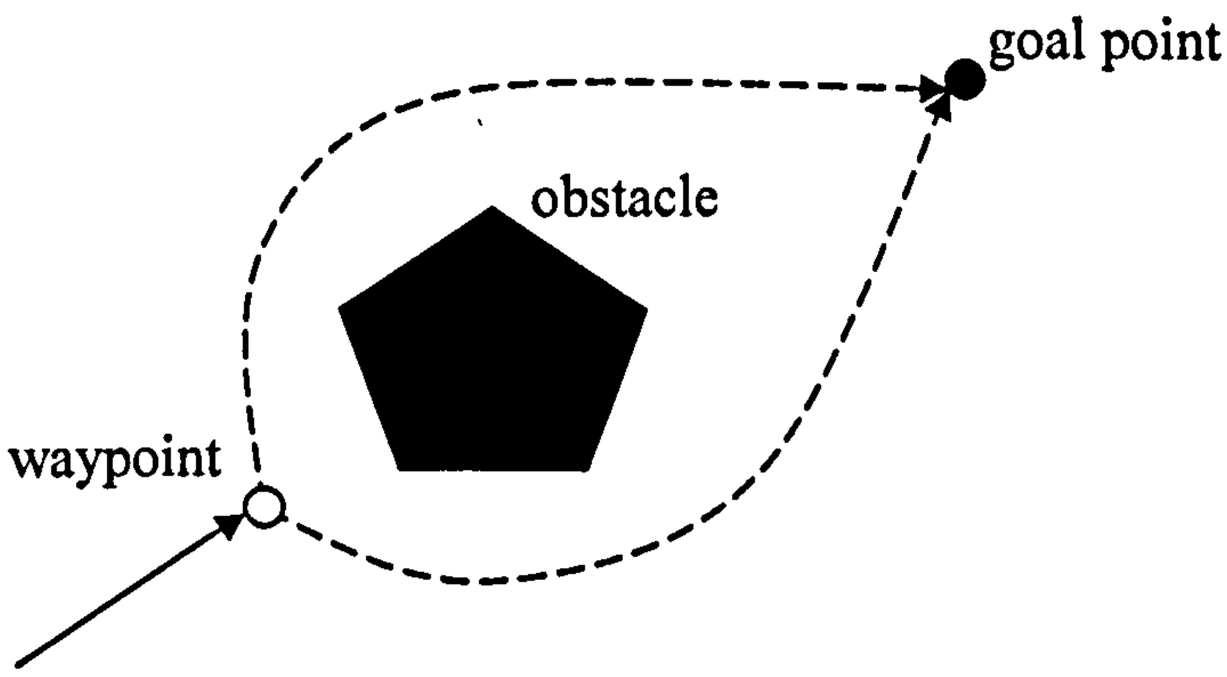


Figure 6.2 On sensing the presence of an obstacle, the robot has a choice of following one of two paths. Under control of the reactive navigator, should the robot need to turn through an angle greater than the pre-defined threshold, a waypoint is recorded.

The value of the pre-defined threshold angle clearly influences whether a given location is selected as a waypoint and, consequently, the physical dimensions of the region explored in its immediate vicinity as a result. Conversely, as the environment within which the robot moves may be large, lowering the threshold may dramatically increase the number of waypoints generated, resulting in unnecessary computational and memory costs.

When a waypoint is detected, its coordinates and the subsequent heading taken by the robot are recorded, as shown in Figure 6.3. The time taken for the robot to travel from the previous waypoint (which may be the start point) to the current waypoint is also stored. The final waypoint is the goal point, which contains the time to travel the final segment. Also included in the waypoint structure is an indication of the previous waypoint. The order in which the set of waypoints is recorded in the knowledge base is that of their discovery. If the waypoints are viewed as a set of artificial or virtual landmarks, a topological map similar to those used by Aguirre and González (2003) and Muñoz-Salinas *et al.* (2005) can be constructed from the collected waypoints, with nodes indicated by the waypoints' coordinates and edges between two adjacent nodes weighted by the distance (in terms of travel time) contained in the waypoint. Such virtual landmarks do not require the capture of distinct features as landmarks as needed in many navigation systems. The waypoints determined by the approach described by Vaughan *et al.* (2002) contain two pieces of information: the first is the direction to the goal point and the second is the estimated time required to reach the goal. In contrast, the waypoints recorded in this work contain additional information regarding an alternative heading for avoiding the obstacles as well as information about the previous waypoint. This extra information permits back tracking and provides a valuable indicator to guide future exploration.

x coordinate	y coordinate	heading	travel time	previous waypoint

Figure 6.3 The structure of a waypoint.

6.2.3 Deliberative unit

The deliberative control system contains three sub-units, namely localisation, exploration and planning.

Localisation Since the waypoints are progressively acquired, the robot can identify from where it accumulated knowledge of the environment. As it is possible that a same location or a location near to the previous waypoints is selected again as a new waypoint, this duplicated information is not new to the robot but requires additional

memory if recorded. In an event driven system (Vaughan *et al.* 2002), if a newly-found waypoint locates the same event as an existing waypoints, the existing waypoint was replaced by the new waypoint. The other waypoint-based systems introduced in section 6.2.2, however, did not address the issue of duplicated waypoints. In our system, a simple pragmatic strategy is used to combine waypoints. If a new waypoint is generated during the exploration phase, a test is made as to whether any waypoints already recorded in the knowledge base lie within the detection range of the sensors. Only if no previously recorded waypoint is within this sensor range is the new waypoint recorded, otherwise it is assumed that the local area has already been explored. It is reasonable to set this distance measure between two waypoints as the sensor range, as the robot can localise itself in this range by only one waypoint. Not combining waypoints or the use of a shorter range may improve reliability, but there is a clear trade-off in respect of memory use, as discussed in the previous sub-section.

Exploration In order to generate waypoints for use in planning, exploration of the environment is required. Given the practical task of moving from a start position to a goal point, two practical approaches to the generation of waypoints have been implemented. The first approach is appropriate when the robot is introduced to a new environment and the assumption is made that it is completely known (as is the case for EP/N and the vertex planner). The movement to the goal point (as well as relevant exploration), is then simulated off-line and the waypoints so generated can be used to plan the best path. The second approach is to determine suitable paths using the waypoints already entered into the knowledge base arising from previous navigation tasks or planned explorative movements. In this case, the robot can collect the environmental information represented by waypoints under reactive control with no requirement for *a priori* knowledge of the environments. This is achieved due to the presence of the reactive system in the hybrid architecture. The navigation systems mainly based on the planner (Sedighi *et al.* 2004; Wang, Yong and Ang Jr. 2002; Zheng *et al.* 2005) need an *a priori* model of the environment in order to perform navigation. On the other hand, the systems with reactive components (Liu, Hu and Gu 2006; Muñoz-Salinas *et al.* 2005; Vaughan *et al.* 2002) rely on information captured

by sensors to fulfil the navigation in unknown environments. Note that the results presented in this chapter have been obtained using waypoints generated by the second approach.

Exploration can be invoked deliberately (perhaps when the robot is introduced to a new environment) or can be permitted when no tasks are currently assigned to the robot. As each waypoint generally defines a branch in a path indicating possible alternative routes around an obstacle, exploration from existing waypoints could potentially provide paths better than those already discovered. Moreover, if one of the generated paths becomes impassable due to the movement or introduction of a new obstacle, the robot may be able to follow one of the alternative paths found during exploration instigated in non time-critical phases, thus reducing the need to explore during situations when the aim is to reach the goal in minimal elapsed time. To begin exploration from an unexplored waypoint, the robot rotates through an angle equal and opposite to that between the goal direction and the previous heading taken from the waypoint. From this point, the robot will rely on reactive control to navigate to the goal point or to a waypoint already discovered. Note that, during such navigation, obstacles may be encountered and further waypoints discovered. Exploration continues unless the robot is requested to execute a higher priority task. With the aim of learning navigation information regarding the environment by exploration, the strategy developed should minimise the covered distance and avoid repeated exploration while maximising the area investigated (Dessmark and Pelc 2004; Fleischer and Trippen 2005; Gartshore, Palmer and Illingworth 2005; Panaite and Pelc 1999; Su and Tan 2005). In a completely unknown environment, the simplest strategy is to perform exploration at random (Barto, Sutton and Watkins 1990). Yamauchi (1997) proposed a frontier-based exploration approach, in which the frontier is first established between the observed and unseen areas and the location to explore is determined according to selection strategies (Burgard *et al.* 2005; Freda and Oriolo 2005; Gonzalez-Banos and Latombe 2002). To improve efficiency, Poncela *et al.* (2002) proposed an algorithm to select the next action by optimising the view range based on a utility function in an attempt to obtain geometrical information of the obstacles while taking a minimum number of steps. A similar idea

was developed in Jia, Zhou and Chen (2004) by taking into account time-saving (or energy-saving) based on a different utility function. Using the accumulated knowledge of the environment, a plan can be generated to direct the exploration to unvisited areas. For example, in a partially known topological map, a plan may be generated as a sequence of nodes (places) with aim of minimising the straight-line distance and with the exploration task being the discovery of the area between two successive nodes. Exploration planning is then similar to solving the travelling salesman problem. A number of techniques (Dessmark and Pelc 2004; Fleischer and Trippen 2005; Panaite and Pelc 1999; Poncela *et al.* 2002) have been proposed to solve this problem, but all attempt to create a complete model for the environment. In contrast, in this work, the exploration is directed in such a way that only those parts of the environment relevant to specific navigation tasks are investigated.

Planning The introduction of waypoints as part of the navigation process gives the opportunity to search for a feasible path using only the recorded collection of waypoints rather than attempting to search the whole environment. A suitable planning method that employs waypoints is described in detail in the next section, and, to assess its performance, two alternative planning approaches, EP/N and the vertex planner are also considered for comparative purposes.

6.3 Planning approach

The evaluation of the effectiveness and robustness of the proposed waypoint navigation approach was made in its comparison with two other planning methods in their application to four simulated environments (section 6.6). The EP/N and vertex methods can be considered as planning approaches in their own right, whereas the waypoint method based on steady-state GAs operates with the support of a reactive system and follows the architecture of the system shown in Figure 6.1. None of the hybrid architectures found in earlier work (section 6.1) conducted an experimental comparison, perhaps because the architectures developed were very dependent on the specific type of robot platform adopted and so a fair comparison between two distinct hybrid architectures could not be conducted. However, in the current work,

comparative studies were made feasible: with the vertex method as the same robot platform was adopted and with EP/N as it was entirely re-implemented by the author. Note that indoor navigation has been the focus for the most of the hybrid system surveyed in section 6.1, but the hybrid system proposed in this chapter has been verified for outdoor unstructured environments. A subjective comparison with a number of hybrid architectures is provided in section 6.8. The vertex planner was introduced in chapter 4 and a review of EP/N was given in section 3.1.3 and hence only the planning approach designed for waypoint navigation is described here.

The algorithm for the waypoint navigator is shown in Figure 6.4. This GA follows the steady-state architecture and incorporates a deterministic crowding mechanism. Although the underlying GA has largely the same structure as both the EP/N and vertex planning techniques, here the intermediate nodes for a path are a selection of the waypoints generated during the exploration phases under reactive control.

```
procedure waypoint navigator
  begin
    produce P with the constraint that all paths generated are feasible
    evaluate P
    while the termination condition is not reached do
      select an operator O
      if the crossover operator is selected then
        select parents P1 and P2 using a roulette wheel based on individuals' rank
        produce offspring C1 and C2 by crossover of the parents P1 and P2
        evaluate the offspring C1 and C2
        if distance(P1, C1)+distance(P2, C2) < distance(P1, C2)+distance(P2, C1) then
          if fitness of C1 > fitness of P1 then C1 replaces P1 end if
          if fitness of C2 > fitness of P2 then C2 replaces P2 end if
        else
          if fitness of C1 > fitness of P2 then C1 replaces P2 end if
          if fitness of C2 > fitness of P1 then C2 replaces P1 end if
        end if
      else
        generate an offspring C using the insertion operator
        evaluate the offspring C
        replace the worst individual in P with the offspring C
      end if
    end while
    select the best individual from P
  end
end procedure
```

Figure 6.4 The pseudocode for the waypoint navigator algorithm.

6.3.1 Chromosome initialisation

Based on the heuristic knowledge gained during exploration, a set of path segments between a pair of adjacent waypoints is made available. Rather than performing

random initialisation, the initial population is generated in conjunction with the heuristic knowledge, so that the chromosomes in the population represent only the feasible paths, where a path is a series of segments. Consequently, the maximum length of a chromosome is constrained to be the longest of the potential feasible paths (in terms of the number of waypoints). All paths commence with the first waypoint encountered after leaving the start node and thus this is the chromosome's first gene. The next gene is identified from the set of waypoints that are connected by segments to the current gene and, if there is more than one branch extending from the current waypoint, one of them is randomly selected to be the next gene. Subsequent genes are defined in a similar manner until the goal gene is reached. To obviate the need for encoding and decoding, the same genetic representation is employed as that already used to store waypoints in the knowledge base unit.

6.3.2 Genetic operators

Two genetic operators, a multi-point *crossover* and an *insertion*, are used to produce offspring. To exclude any infeasible offspring generated by the crossover operation, the crossing points are chosen deterministically, by searching for pairs of waypoints that are common in the parents. The segments lying between the identified pairs of waypoints that are common to the parents are then interchanged to form the offspring and therefore the number of crossing points varies with the number of common nodes between the parent individuals. This is different from the crossover operator used in EP/N (section 3.1.3) and in the vertex planner (introduced in chapter 4), which both of used single point crossover. The likelihood of producing less fit individuals is increased if the crossover operator used in this chapter were applied to the feasible individuals in the EP/N and vertex planners. Note that the crossover sites are determined at random in each of the EP/N and vertex planners, with the commensurate possibility of generating infeasible paths. Although infeasible paths can be gradually converted into feasible paths by the repair operator in the EP/N and vertex planners, the process may take many generations. On the other hand, the planning algorithm presented in this chapter excludes infeasible paths during the evolutionary process to take advantage of exploration or previous navigation and thereby eliminating the evolutionary process to convert infeasible paths to feasible

paths as found in the EP/N and vertex approaches. Such a crossover operator used ensures that significant exchange of genetic information occurs between the parents, while population diversity is still being promoted by the application of the insertion operator. The insertion operator plays a similar role in the evolution to a conventional mutation operator, but is different in the sense that the insertion operator randomly generates a new individual using the initialisation mechanism rather than mutating genes of the selected parent based on some chosen probability value. The two operators are applied on alternate generations rather than being selected based on pre-defined probabilities, as in EP/N, reducing the number of system parameters that need to be defined.

6.3.3 Selection scheme

In the waypoint navigator, offspring are generated using a crossover operator that acts on a pair of individuals. The pair is selected by a roulette wheel whose slots are sized in accordance with the ranks of the individuals. A quadratic ranking technique (De Jong 1992; Watanabe and Hashem 2004) (see section 2.3) was implemented to scale the raw fitness before selection, so that the selective pressure is independent of the fitness distribution of the population while the selection is biased towards the favoured individuals. An alternative would have been to adopt a purely proportionate selection scheme, but although this method tends to exhibit rapid convergence due to the high selection pressure during the initial phase of evolution, there is less selective differential in the later evolution, providing little incentive for the GA to make the appropriate selection between competing individuals (De Jong 1992; Sareni and Krahenbuhl 1998).

6.3.4 Evaluation

As only feasible paths are involved in the evolutionary process, the quality of a path can be determined simply by its length.

$$E_f = \sum_{i=1}^{n} L_i \qquad \text{Equation 6.1}$$

where L_i denotes the length of the segment i of the path containing a total of n segments. Alternative assessment criteria, such as the number of waypoints (the number of turns) or travel time, could be accommodated where required.

6.3.5 Replacement strategy

Deterministic crowding (DC) (Mahfoud 1995a; Sareni and Krahenbuhl 1998) is used as the replacement scheme for the offspring generated by the action of the crossover operator. The competition between offspring and parents of identical niches (closest competition) helps to maintain the diversity of the population. DC yields two set of tournaments, the first involving offspring C_1 pitted against parent P_1 and offspring C_2 against parent P_2 and the second involving offspring C_1 against parent P_2 and offspring C_2 against parent P_1. A parent is replaced by the nearest offspring should the latter have better fitness. Similarities among the individuals are defined based on phenotypic distance. Equation 6.2 shows the similarity S defined for this problem,

$$S = \frac{N}{N_P} + \frac{N}{N_c} \qquad \text{Equation 6.2}$$

where N is the number of common waypoints between the parent and offspring, and N_p and N_c denote the number of waypoints in the parent and filial paths respectively. The aim of applying the DC technique is to retain population diversity so as to increase the probability of evolving optimal solutions. Sharing approaches (Goldberg and Richardson 1987) and clearing techniques (Pétrowski 1996; Sareni and Krahenbuhl 1998) both require the determination of niche radius, which is difficult in absence of domain knowledge (section 2.5). Path quality may be improved if DC is applied to the EP/N or vertex planner, but experimental observation showed considerable time was needed to check similarity (defined as how many obstacles lay between two paths with fewer obstacles implying greater similar). Consequently, the work described in the previous two chapters did not use DC to minimise the planning time. Note that the single offspring generated by insertion replaces the worst individual in the population.

6.4 Investigation of the waypoint method in escaping from 'U-shaped' traps

As no memory of previous decisions is stored, most purely reactive systems are unable to escape from the dead-end or 'U-shaped' obstacles found in many practical environments. However, during reactive navigation in the waypoint system, a means of providing such memory by recording recently-visited locations is readily available. Figure 6.5 depicts the sequence of processes that the waypoint method uses to escape successfully from a U-shaped obstacle. In (a), the robot enters the obstacle, generates a waypoint and continues within the obstacle in such a direction that it reduces the distance to the goal. In (b) and (c), the robot begins what appears to be oscillatory motion and generates two further waypoints. By adopting a rule that if a newly-generated waypoint is determined by the localisation unit to be close to one already detected in the current exploration cycle and at which the robot has a similar heading, the system will instigate behaviour to escape from the obstacle. This behaviour involves generating an apparent goal location for temporary use as a target point by the reactive navigation system. This new location is determined from the original goal location by translation in the same direction as that of the robot heading when the duplicated waypoint was identified, as shown in (d). The translation distance needs to be sufficiently large to ensure escape and in practical cases this can be easily achieved by moving the apparent goal location to the edge of the navigation environment. Only when the robot's heading is directly towards the apparent goal, as shown in (e), is the goal returned to its original position, as in (f). Note that, throughout this process, the mobile robot has continued to operate purely under reactive control. Many approaches reported in the literature to escape from U-shaped obstacles (Minguez, Osuna and Montano 2004; Na and Oh 2003; Suzuki *et al.* 2005) used a sensor that can capture the characteristics of the U-shape obstacle from a single point of view. Often, the robot may remain trapped if the U-shaped obstacle cannot be identified correctly and such a limitation has been experimentally examined by Antich and Ortiz (2006). In the current work, the robot can, solely under reactive control, escape U-shaped obstacles recognised by the sensor at single sampling instant, since this is characterised by a single waypoint and the similar situation has

been presented in the training stage. If such situation is detected, the robot will reverse its heading immediately out of such a U-shape obstacle. The use of waypoints as introduced in this section is intended to escape the traps involving in U-shape obstacles whose features cannot in practice be identified by a sensor at a single sampling instant, due to the relatively large size of the obstacle.

In order to avoid such deadlock problems that are often found in purely reactive systems, schemes reported in the literature can be roughly categorised into two types: one to incorporate wall following behaviour and the second to establish a temporal virtual target. In the first category, Maaref and Barret (2002) co-ordinated behaviours for convex and concave obstacles: if a convex obstacle is detected, goal seeking behaviour and reaching the middle of the collision-free space behaviour was combined, whereas the escape of concave obstacles relied on the coordination of behaviours used for convex obstacles and wall following behaviour. The approach of adopting wall following behaviour in order to avoid traps can be also seen in the work by Antich and Ortiz (2006). In the second category, Xu and Tso (1999) and Xu, Tso and Fung (1998) proposed an approach to escape the trap in U-shaped obstacles using local target switching. Here, a dummy target will be switched to the opposite direction at which the real target exists with respect to the robot, with the potential trap being detected by identifying an abrupt change that the robot heading suddenly rotates from the left (or right) side to right (or left) side of the goal. Only once the robot detects an opening on its left (or right) side will the dummy target be switched back to coincide with that of actual target. In contrast, the approach described here, monitoring of the sensory information is not required (thereby eliminating false triggering of behaviour change) and the direction to the temporal goal (G') is used instead. In the approach described by Arambula Cosio and Padilla Castaneda (2004), four auxiliary attraction points were positioned horizontally or vertically around the goal point and assigned suitable strengths in the grid map. The trap could then be escaped by applying, to the robot, the combination of five forces.

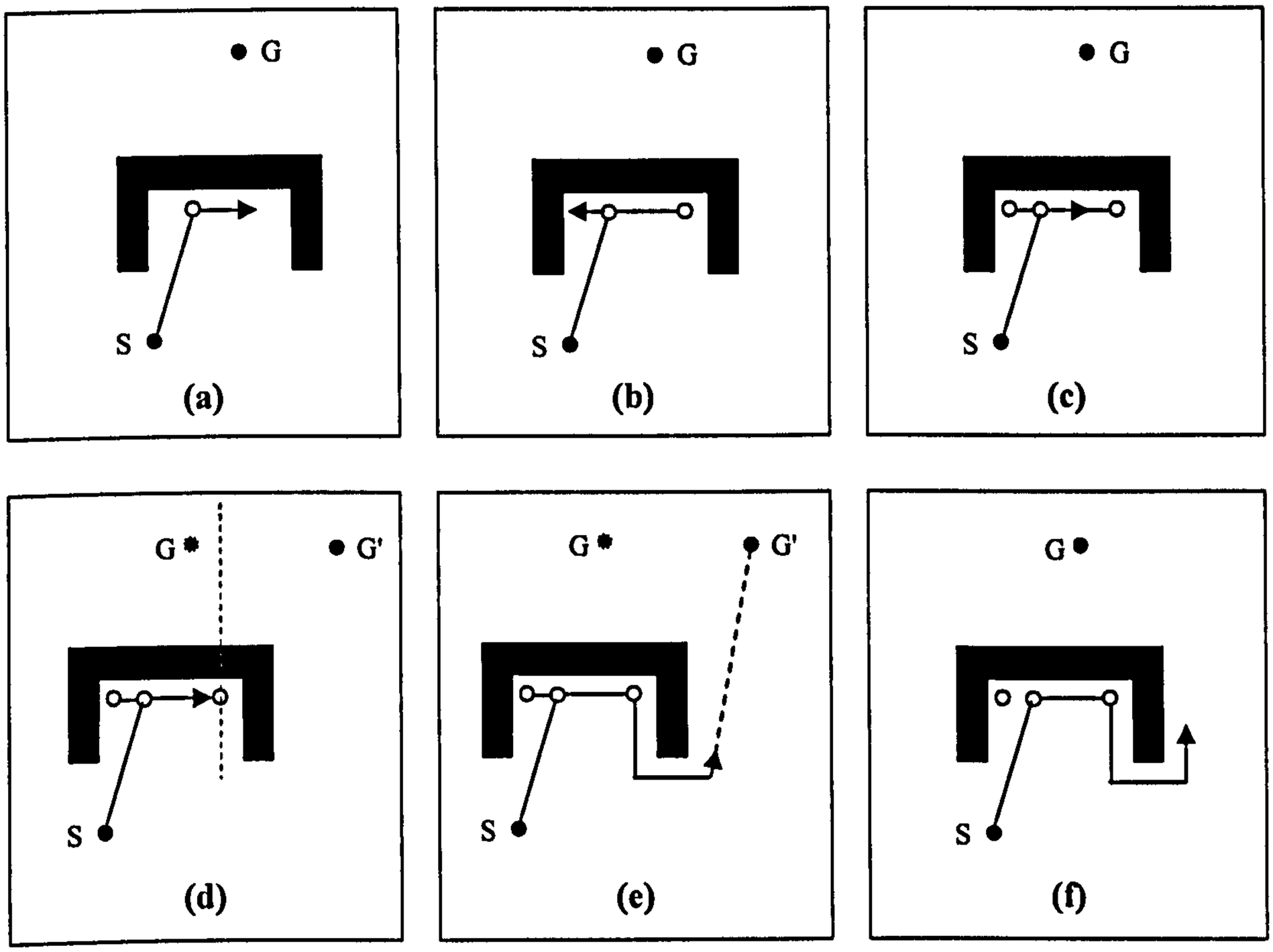


Figure 6.5 Illustration of the mobile robot escaping from a U-shaped obstacle. The circular markers show the waypoints and the solid circular markers denote the start (S), goal (G) and apparent goal (G') points.

6.5 Experimental procedure

The simulations used an autonomous mobile robotics toolbox (Brno University of Technology 2006) running on MATLAB (Mathworks 2006) version 6.5. The toolbox allows the definition of virtual environments and simulates the behaviour of one or more robots, each of which can be equipped with a range of configurations of ultrasonic sensors and laser scanners. The toolbox consists of two separate applications, namely *editor*, that allows the user to create virtual environments, define robot configurations, edit the control algorithms and load and save simulations, and *simulator*, that provides a graphical view of dynamic movements and allows data to be recorded for later analysis.

6.5.1 Toolbox features

The principal features of the toolbox are as follows (Brno University of Technology 2006).

- A graphical user interface providing simulation, creating, viewing and editing.
- The simulator supports as many as 254 robots each with its own control algorithm.
- Control algorithms can be designed as standard MATLAB functions.
- Ultrasonic sensor and laser scanner simulators are provided in the toolbox and their number and position can be configured for each robot.
- The virtual environment consists of static obstacles and simulated robots can be used as moving obstacles.
- Movements for each robot are recorded and can be saved and replayed.

6.5.2 Kinematic modelling

The toolbox model of a robot has three wheels, as shown in Figure 6.6; at the front is an undriven castor, and at the rear are two conventionally actuated and steered wheels equipped with velocity feedback. This type of chassis provides only two degrees of freedom and the simulated robot in the experiments presented in this thesis was constrained to be incomplete (that is, the robot is unable to rotate without any translational displacement except that the robot can reverse its direction immediately when facing narrow dead end), so that the algorithm can also be applied to the robots wiouth any translational displacement when rotating. In order to provide rotational movement the rear wheels are driven differentially.

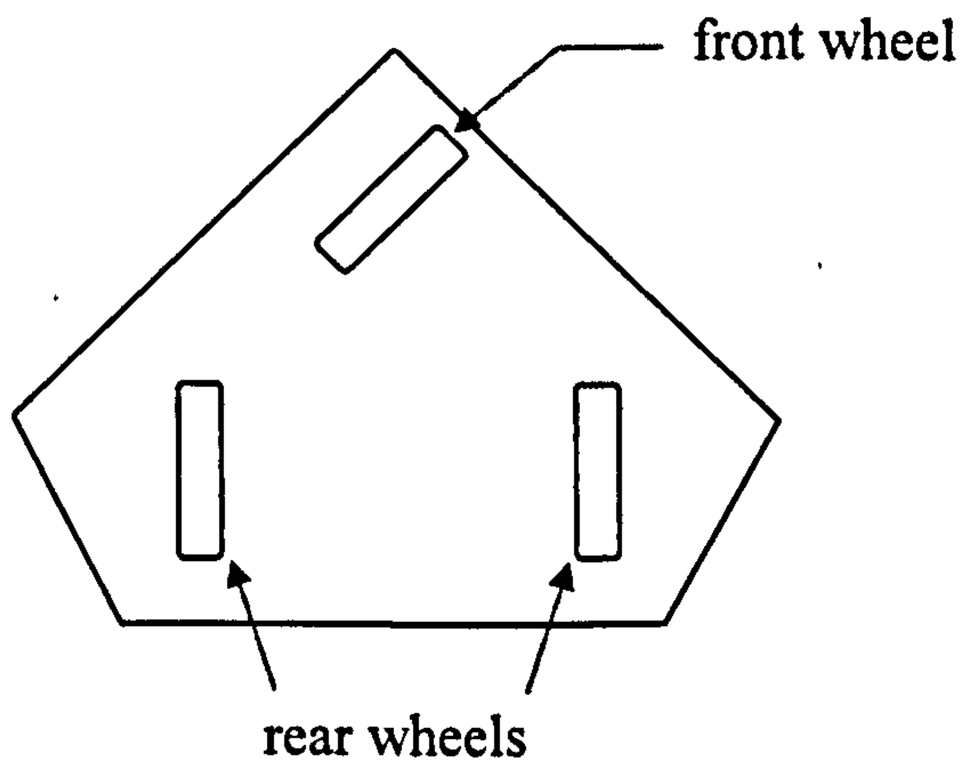


Figure 6.6 Locomotion mechanism for the simulated robot.

This simulator has previously used in our research group at Loughborough University for developing an autonomous navigation system. To extend the previous work directly, the hybrid systems presented in this thesis adopted the same simulator. Recently, this simulator was employed by Baklouti and Alimi (2007) for the navigation of mobile robots using a fuzzy-logic controller. The toolbox allows multiple robots to populate the environment each with their own control algorithm. This feature was used in chapter 7 when investigating navigation problems in dynamic environments and in which robots were used to model a set of moving obstacles. User-defined control algorithm can be easily developed with the support of MATLAB functions and the experimental process and results can be displayed and analysed in the MATLAB environment.

6.6 Results of the comparisons between the navigation methods

The three navigation systems (EP/N, vertex planner and waypoint navigator) have been implemented to carry out a range of tasks in four simulated environments designed to exercise the robot in diverse activities. The first environment, called *shopping mall*, includes a large number of small obstacles; the second, *park*, contains a smaller number of larger obstacles; the third, *office*, simulates an open-plan office workspace; and the fourth, *manufacturing*, contains a number of walkways through a series of manufacturing cells or along corridors in an office building. The reactive unit was developed in C in such a manner that it can be called as a separate stand-alone executable, whereas the waypoint detector and deliberative units were realised in MATLAB directly.

In the set of experiments that follow, the control parameters used in the three planning methods are as listed in Table 6.1; all were determined experimentally and remained unaltered throughout the practical experiments. For all the methods, a quadratic ranking strategy (De Jong 1992; Watanabe and Hashem 2004) (see section 2.3) was used to scale raw fitness and a roulette wheel approach was adopted for selection purposes.

Table 6.1 The control parameters for the three planning algorithms

algorithm	operator probability								weights			Rate of *mutation_1*	Insert rate of *insert_delete*	rate for *delete*	
	crossover	*mutation_1*	*mutation_2*	*insert_delete*	*delete*	*swap*	*smooth*	*repair*	w_d (for path length)	w_s (for smoothness)	w_c (for clearance)			Feasible node	Infeasible node
EP/N	0.6	0.8	0.5	0.5	0.5	0.5	0.9	0.8	1	0	0	0.3	0.6	0.1	0.3

(a) The system parameters for EP/N. The maximum length of an individual in the initial generation was limited to be the sum of the number of vertices, start point and goal point in the environment under test and the minimum length was set to be two, these being the start and goal points only. When *mutation_2* is selected as a genetic operator, one intermediate node will be affected.

algorithm	operator probability			mutation rate	safe distance (m)
	crossover	*mutation*	*repair*		
vertex planning	0.5	0.3	0.9	0.1	0.2

(b) The system parameters for the vertex planning algorithm. The maximum number of intermediate nodes for a path randomly generated in the initial generation is same as the number of obstacle vertices in the test environments and the start and goal points are always selected.

algorithm	operator probability	
	crossover	*insertion*
waypoint navigator	every other generation	every other generation

(c) The system parameters for the waypoint navigator. The maximum length of an individual in the initial generation is equal to the total number of waypoints in the longest feasible path that could potentially be generated. In our simulation, the threshold angle through which the robot needs to turn to trigger the recording of a waypoint was set to 33.3° in all experiments.

In the following comparison of the navigation approaches, the results of the experiments are considered in terms of the following: the subjective quality of the generated path, the relative path lengths, the number of individuals required to generate the first feasible solution and the times to obtain the first feasible path.

6.6.1 Generated path quality

Figure 6.7, 6.8, and 6.9 show the paths generated following the application of the three planning methods to the same task in each environment. Although the paths shown for the three techniques were obtained over 2000 generations, around 1000 generational cycles were normally found to be sufficient to generate an optimal or a near optimal solution for the environments. Note that these planning algorithms do not always generate an optimal, but rather a near optimal path (as there is not exact world model available to the robot). Consequently, the paths presented in Figure 6.7, 6.8 and 6.9 do not necessarily represent the optimal solutions.

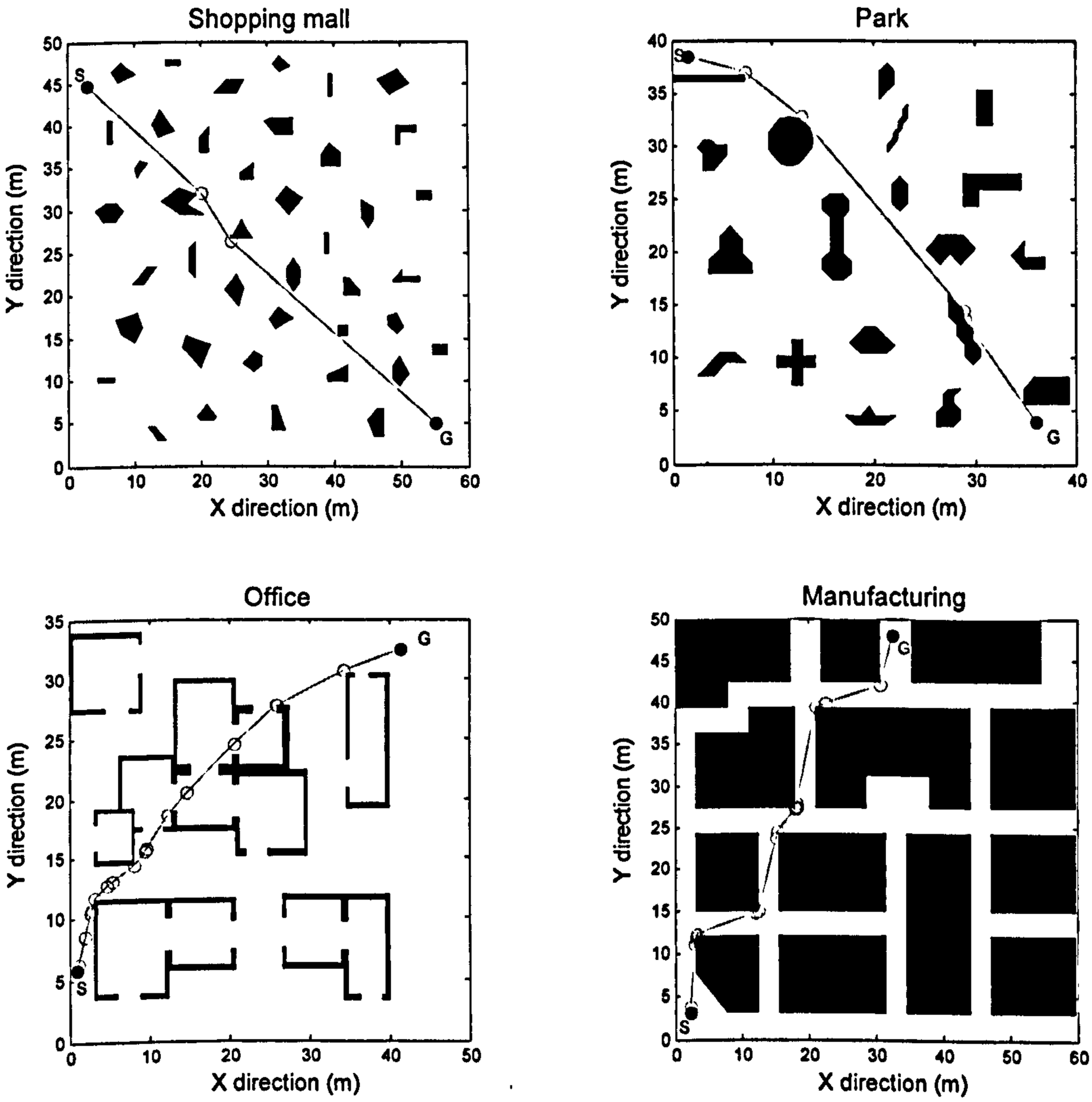


Figure 6.7 The paths generated by the EP/N technique. The circular markers show the intermediate nodes generated by EP/N and the solid circular markers denote the start (S) and goal (G) points.

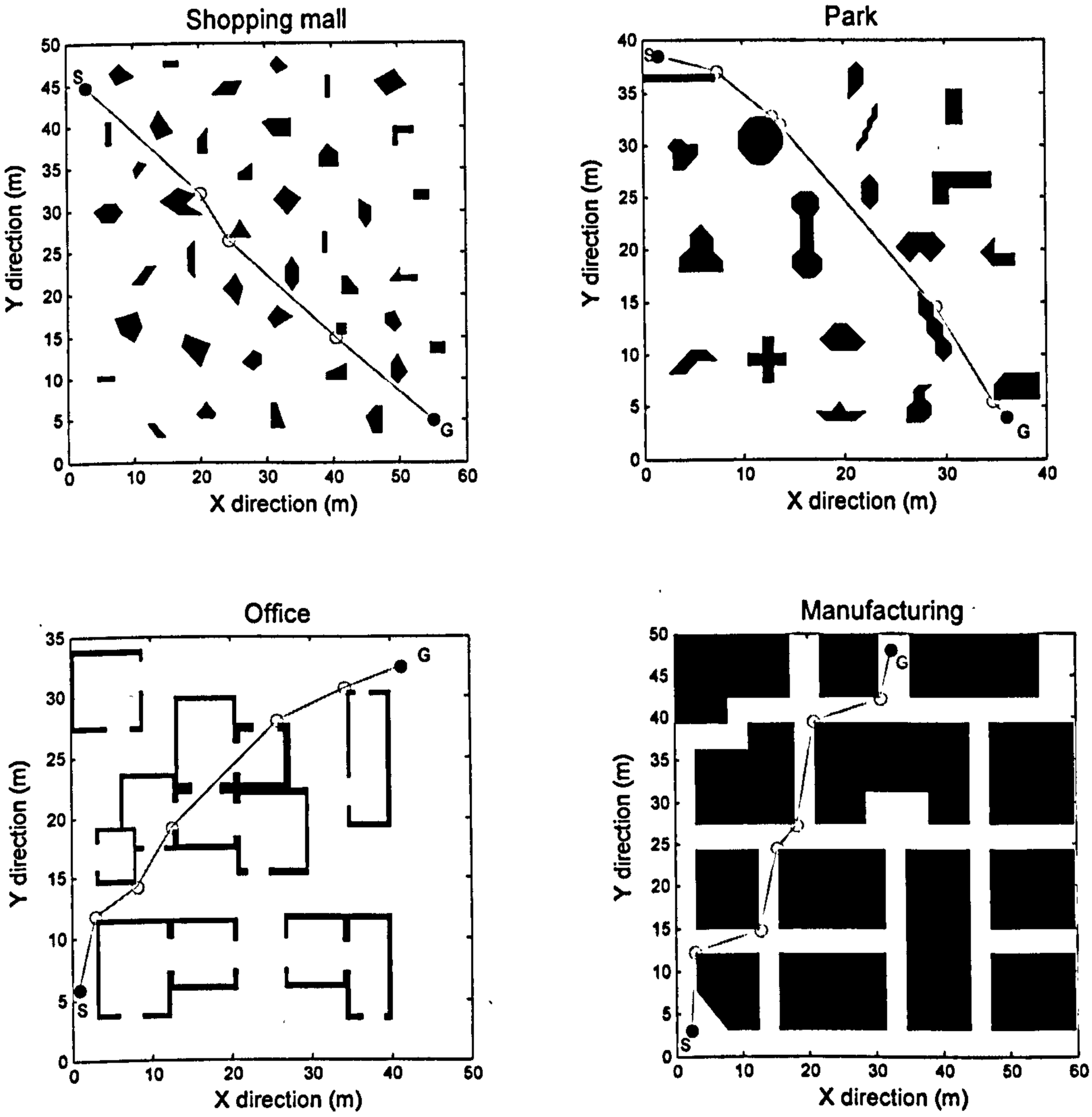


Figure 6.8 The paths generated by the vertex planning technique. The circular markers show the intermediate nodes generated by the vertex method.

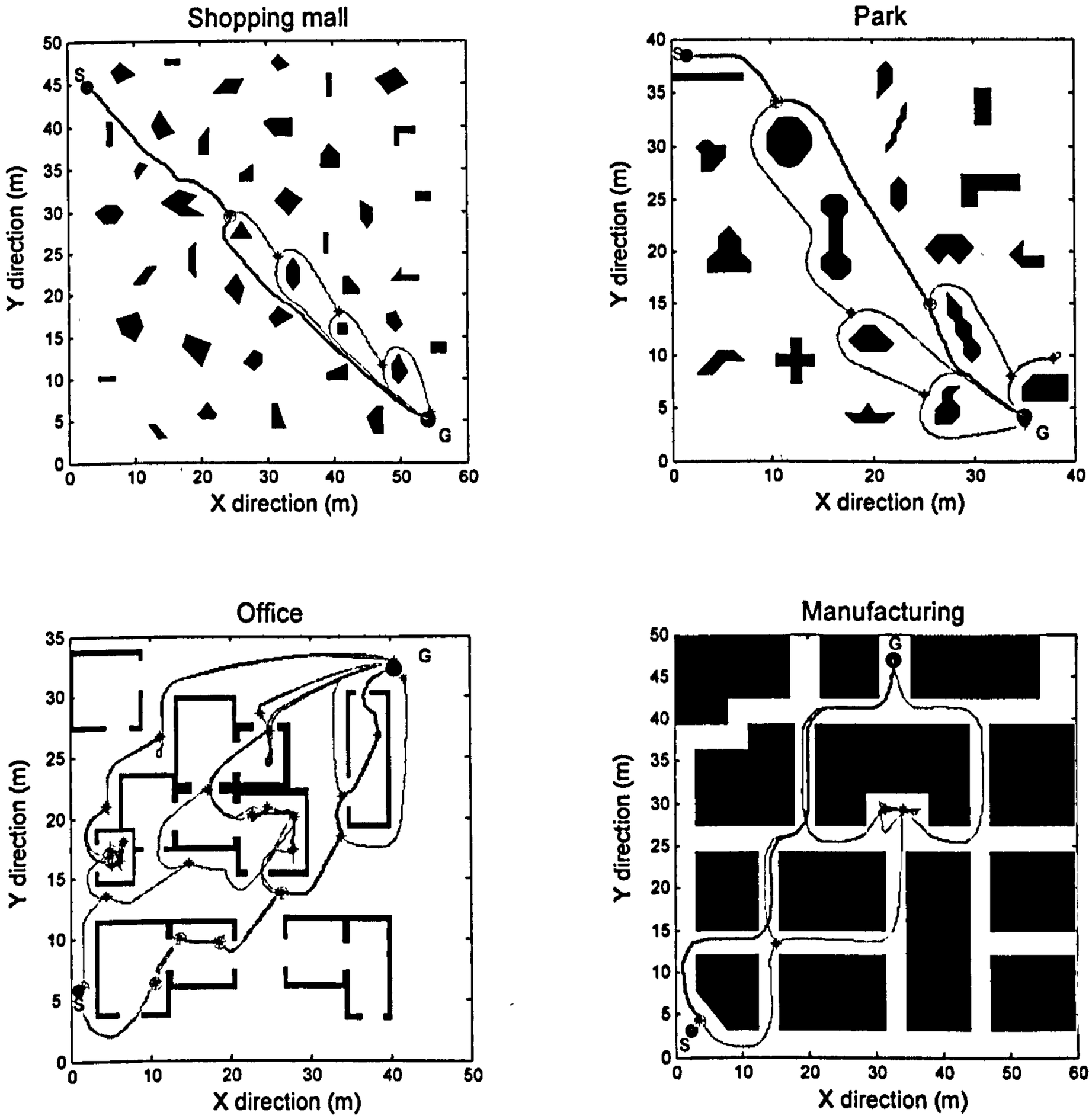


Figure 6.9 The paths generated by waypoint technique. The starred points indicate all the generated waypoints, the circular markers indicate the waypoints selected by the planner and the planned paths are shown as thickened lines. Note that although many additional waypoints were defined for the environment, only those relevant to the specific planning task are shown.

Apart from the intermediate nodes, the paths generated by the EP/N and vertex planning approaches in each of the respective test environments are similar. The circular obstacle towards the top left of the diagram in the park environment could only be approximated for use in checking path feasibility and when performing repair operations in the EP/N and vertex planning methods. As both these methods require that obstacles are represented by a finite number of vertices, a trade-off needs to be made for certain obstacles shapes in terms of representation for efficient yet collision-free motion of the mobile robot and the time that will ultimately be consumed in assessing feasibility. The waypoint navigation system, however, has no requirements

with regard to obstacle shape and hence no such compromise needs to be made. In the waypoint navigation system results, it can be observed that the 'best' path through the identified sequence of waypoints does not necessarily correspond to the global optimum. Although for the shopping mall and manufacturing environments the paths generated by the waypoint method are similar to those determined by the EP/N and vertex methods, it can be observed in the park and office environments that the waypoint planner produces somewhat longer paths than those obtained using either the EP/N or vertex method. As the environment is modelled in the waypoint system as a set of waypoints, only a very limit area of the environment is involved in the planning and so it is unlikely that it will produce paths with the same quality as those found by the EP/N and vertex approaches that operate on an detailed map.

6.6.2 Number of individuals needed to produce feasible paths

The following experiments determine the minimum population size needed to obtain feasible paths over 30 separate runs of the three navigation algorithms. Each evaluation was performed either until no improvement in the convergence towards the optimum was observed, or until identical but near-optimum solutions were observed over ten consecutive generations. The results of the investigation to determine the minimum number of individuals in the population that is needed to evolve a feasible path for each algorithm are shown in Table 6.2. It can be seen that only two individuals were needed in order for the EP/N and vertex planning algorithms to be able to generate a feasible path for most environments. Further investigations showed that this was largely due the effective conversion of infeasible paths to feasible paths by the repair operator. As the evolution process carried out in the waypoint navigator always starts with a population of feasible paths, the minimum number of generations required to generate the initial feasible paths is always zero.

Table 6.2 The minimum number of individuals in a population needed to obtain a feasible path for each algorithm. The minimum was found by determining the smallest population capable of producing feasible paths in no fewer than 19 out of 20 tests each of length 600 generations.

	shopping mall	park	office	manufacturing
EP/N	2	2	20	2
vertex planner	2	2	2	2
waypoint navigator	0	0	0	0

6.6.3 Path length

In order to compare the qualities of the paths generated by the three navigation methods, the geometric lengths of the paths produced by each were calculated and compared. The results presented in Figure 6.10 are the median values of the lengths of the fittest path produced during the first 1000 generations. In the cases where the population size for the EP/N or vertex planning algorithm fell to a value of two, the number of individuals in the population was increased by one in order to prevent the loss of a feasible path on application of the crossover operator. To ensure a fair comparison, the population size used for the waypoint navigator is maintained at the minimum population size (three individuals) adopted in the EP/N and vertex planning algorithms. Although two individuals are sufficient for evolutionary progress, such a small population frequently led to non-convergence. This arose due to the worst individual in the population being replaced by an offspring; it frequently being the case that an offspring produced by insertion was less fit than the worst individual ejected from the current population. Children resulting from a subsequent crossover operation would be likely to have lower fitness, inhibiting the evolutionary process and leading to premature termination.

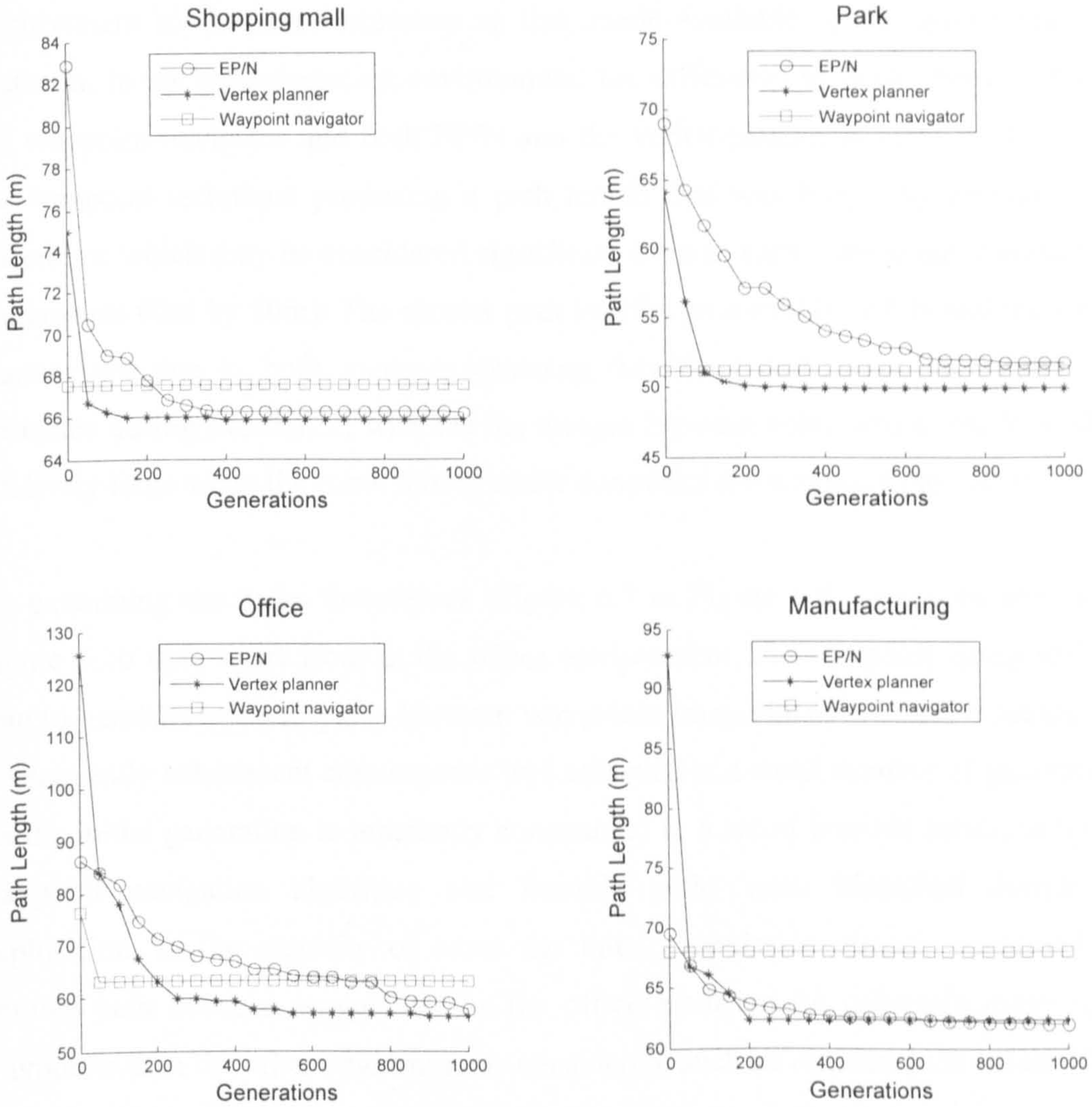


Figure 6.10 The path length for the best individual for generations containing paths that are all feasible. The median value of the path cost over 30 runs is presented.

In the shopping mall, park, and office environments, the obtained path lengths all reached similar values by 1000 generations, although the evolutionary progress of the three algorithms followed different trends and the lengths of paths generated by the waypoint navigator are marginally longer than those produced by EP/N and the vertex planner. As both the EP/N and vertex planning techniques conduct global planning based on complete knowledge of the obstacles' dimensions, a global optimal path can often be generated, whereas, in the waypoint navigator, the robot is under reactive control and only acquires knowledge of its environment accumulatively. In addition, although reasonably large regions of the office environment can be explored, it is probably unrealistic to expect the waypoint method to obtain details about the

environment to the same accuracy as that made available to the global planning methods. In the manufacturing environment, the difference in performance between the waypoint navigator and both EP/N and the vertex planner is most marked, with the waypoint technique producing a path length that was longer by around 6m, a difference which may be considered significant even in such a large environment (of dimensions 60m by 50m). The shorter path lengths produced by EP/N and the vertex planner are due to both methods ensuring that the robot often passed close to obstacles during avoidance, whereas the margin between robot and obstacle is often relatively large when the robot moves under control of the waypoint navigator.

On examining the paths themselves (Figure 6.7 to Figure 6.9), it can be seen from Figure 6.10 that, apart from in the office environment, the waypoint navigator was able to generate optimal paths between waypoints immediately evolution began, and consequently subsequent convergence was achieved in a small number of generations. As the initial generation is inherently constrained to a set of feasible solutions by the waypoint navigation algorithm and feasible paths were identified during the exploration, in the majority of cases the initial population already contained the optimal path between waypoints. As the office environment generates many more waypoints than the other environments considered, such an optimal path is less likely to be present in the initial population rendering further evolution necessary.

In the shopping mall, park, and manufacturing environments, it is apparent that the waypoint navigator has extracted sufficient heuristic knowledge from exploration to allow any one of a range of conventional optimisation or search methods to be applied rather than necessitating a solution using GAs, which would appear somewhat superfluous in this case. However, it will be shown in section 6.7 that for more complex environments, where the number of waypoints is greatly increased, the use of GAs in generating the robot path will again be appropriate. It is clear that in comparison with the EP/N and vertex-based techniques, the waypoint method not only allows the assumption that the robot needs to be equipped with complete knowledge of its environment before planning to be relaxed, but also represents the

information from the explored environment in such a way that it can be used in a highly efficient manner for planning purposes.

6.6.4 Time to obtain the first feasible path

Table 6.3 lists the median time needed to obtain the first feasible path in the four sample environments, determined from a set of 30 runs. It can be seen that the vertex planning method is able to provide a modest reduction in calculation time compared with EP/N, except in the manufacturing environment where the improvement is more apparent. This may be due to the fact that the manufacturing example has a considerably higher proportion of the environment occupied by obstacles and so a significant number of the nodes, selected initially at random by EP/N, are likely to be within an obstacle. The waypoint navigator takes advantage of the prior explorations that result in a set of feasible candidates for the initial population, and consequently no evolutionary process is required for generating the first feasible path.

Table 6.3 The median calculation times (in seconds) to obtain the first feasible path. Each value was obtained over a series of 30 identical planning tasks.

	shopping mall	park	office	manufacturing
EP/N	20.0	18.0	33.2	14.9
vertex planner	18.6	16.4	25.6	4.0
waypoint navigator	0	0	0	0

Figure 6.11 shows the times taken for each of the three navigation methods to calculate planning solutions during the first 1600 generations. In each case, the minimum population size shown in Table 6.3 is used in the first 600 generations and evolution from the first feasible path to the final path occurs during the final 1000 generations.

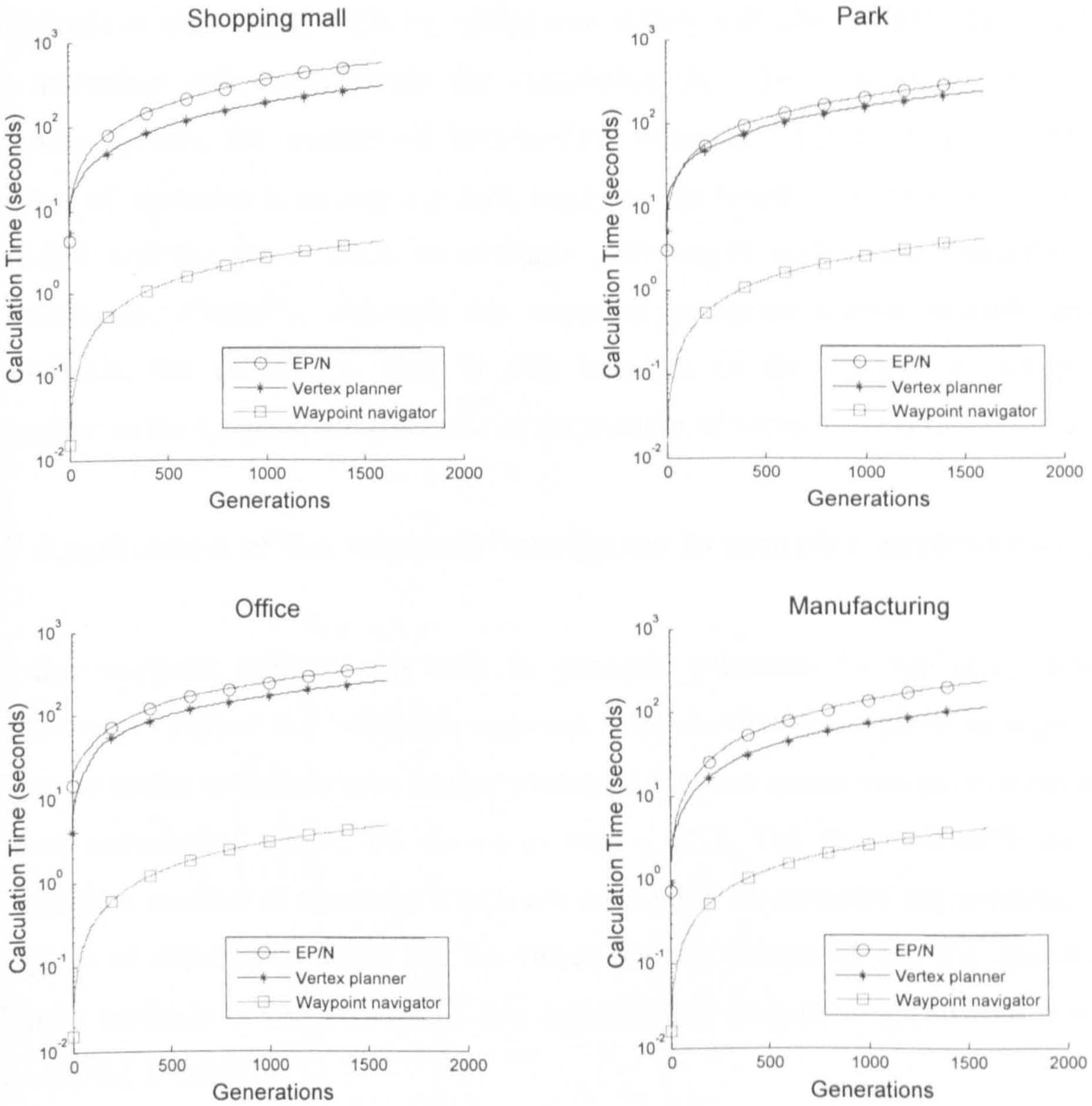


Figure 6.11 The median values for the calculation times averaged over 30 runs.

The results in Table 6.3 and Figure 6.11 demonstrate that the calculation times to obtain the first feasible path and to complete the defined number of generations by the waypoint technique are considerably shorter (around two orders of magnitude) than those of either the EP/N or vertex planner. Although the repair operator in the EP/N and vertex planning systems is effective in achieving feasibility for a path, it is one of the most costly in terms of calculation time. EP/N operators such as crossover, insert, smooth and repair all increase the length of the individuals and consequently adversely affect the number of calculations that need to be performed to assess path length. Also, EP/N often attempts to convert infeasible paths to feasible ones using the repair and insert nodes, again resulting in longer individuals whose feasibility will then take longer to check. Further, the smooth operator, that attempts to reduce the

magnitude of the turning angle by adding new nodes, will also increase the length of the individual and consequently the calculation time. In contrast, in the vertex planning system, the number of intermediate nodes is limited as the number of vertices of obstacles is known *a priori*, implying the length of each chromosome is bounded and the times taken to evaluate path length and assess feasibility are deterministic. Similarly, although the waypoint navigator allows variable length individuals, the calculation time is also bounded as the number of waypoints contained in the longest path (in terms of the number of waypoints) is constrained.

6.7 Application of the waypoint navigator to complex environments

As the waypoint method was able to generate solutions for the environments considered in section 6.6 with such apparent ease, the GA-based planning algorithm of the waypoint technique was further evaluated for four additional environments of greater complexity, which are shown in Figure 6.12. The four test environments contained a number of obstacles which are unsuitable for accurate representation by polygons as required by EP/N and the vertex planner. Consequently, the application of these methods is likely to result in a considerable computational overhead when considering feasibility.

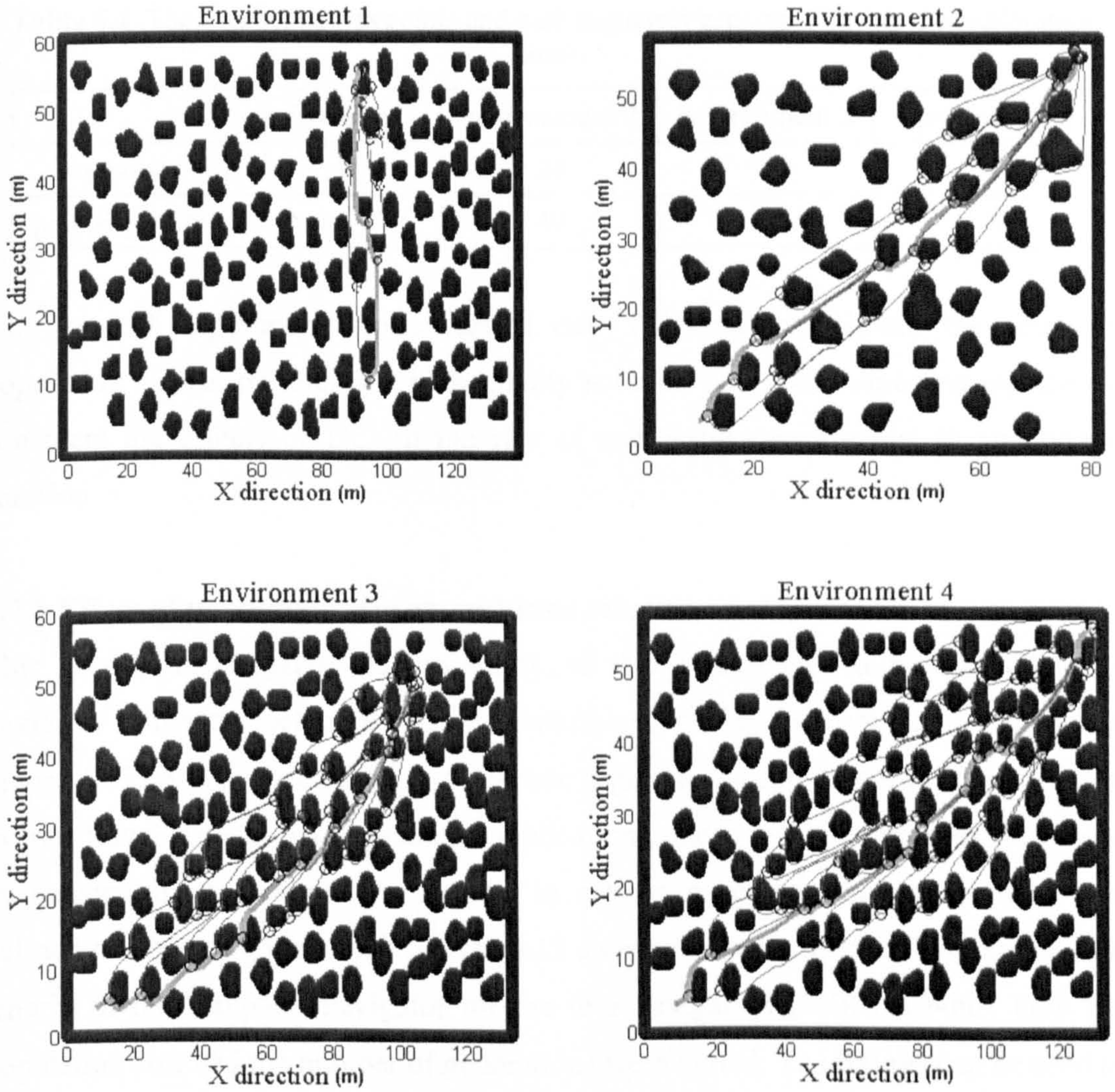


Figure 6.12 The paths generated by waypoint technique for the four complex environments. The circular markers indicate the recorded waypoints and the planned paths are shown as thickened lines. In the first environment, the robot's task is to move from the bottom to top, while the robot moves from the bottom-left corner to the top-right corner in each of the other three environments.

To illustrate the increased complexity of the planning problem now to be solved, Table 6.4 shows the number of waypoints and path segments generated during the exploration phase. As the robot has fully explored all the waypoints in these environments, two paths emanate from each (apart from the start point) and hence the number of path segments generated is simply one more than twice the number of generated waypoints.

Table 6.4 The number of waypoints and path segments generated during the exploration phase.

	environment 1	environment 2	environment 3	environment 4
waypoints	11	24	39	54
path segments	23	49	79	109

Two sets of experiments were carried out. The first investigates the effect of population size in producing a high quality solution in a fixed time and the second considers the effects of DC on the rate of successful convergence to an optimal solution.

6.7.1 Effect of population size on real-time solution quality

This section investigates the optimality of the plan produced by the waypoint navigator using the set of waypoints determined during exploration, as shown in Figure 6.12. As the time taken to determine a plan has a significant influence on the overall travel time in many practical applications, these results have been generated by constraining the planner to operate in a fixed execution time rather than in a defined number of generations. Figure 6.13 shows the performance (in terms of path length) of the waypoint navigator for the four navigation problems while both the population size and the number of generations were varied. The figures can be used to aid an assessment as to whether increasing the population size or increasing the number of generations will be the more effective in achieving a path of the desired quality in a given execution time. Note that the results presented in Figure 6.13 were obtained from 1000 independent runs for each navigation problem.

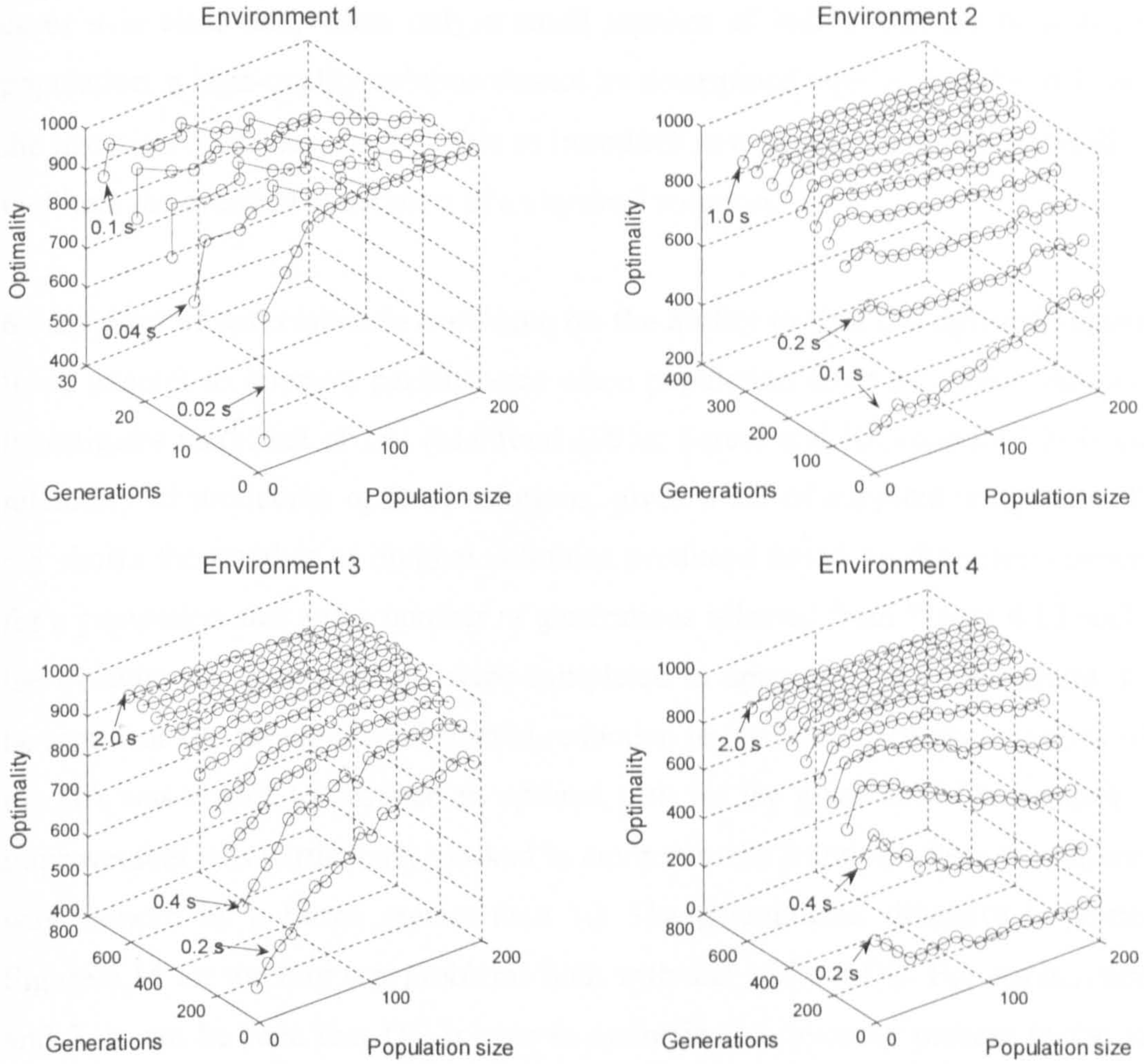


Figure 6.13 The optimality achieved by the proposed planning algorithm for the environments shown in Figure 14 with the population size being changing from 10 to 200 individuals in steps of 10 for the test environments.

From Figure 6.13, it is evident that, in many cases, good optimality with respect to the available set of waypoints can be achieved by the planning algorithm with a modest population size in under two seconds. For the first three environments, the performance of the planning algorithm improves as the population size is increased. However, in environment 4 this is not apparent for the shorter execution times and is only observed when more generations become available and when the time constraints are not so rigid: it can also be seen that the increased population size cannot compensate for the evolutionary effect of the reduced number of generations. There appears to be a certain threshold in terms of the number of generations, below which one should not fall in order to achieve a high quality solution. In a number of

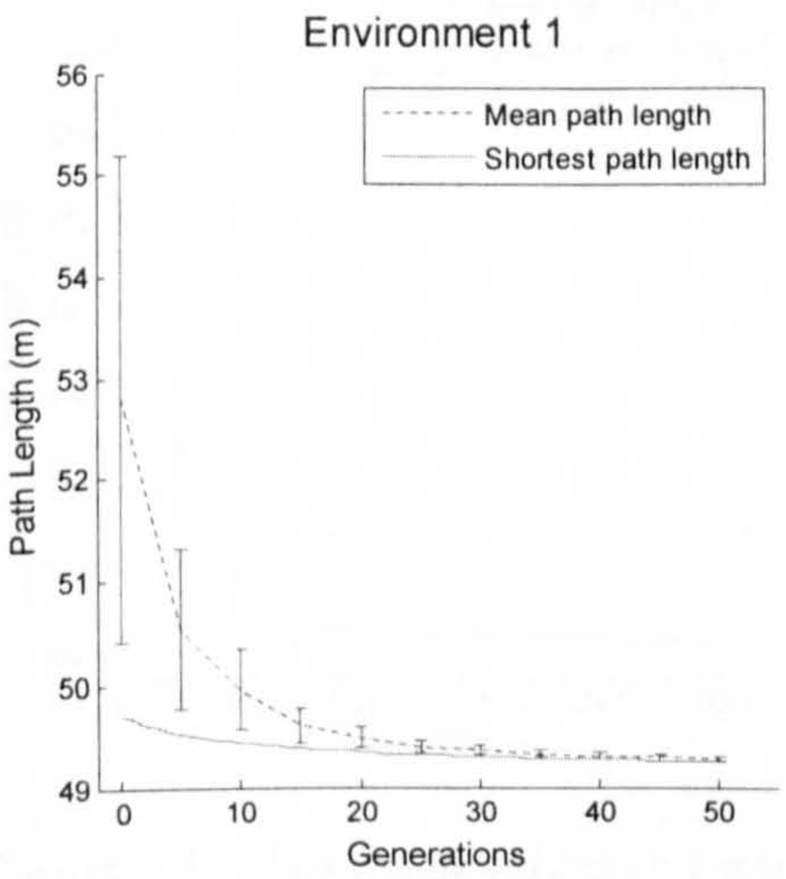
cases it is clear that, when only a small number of individuals are present in the population, a high-quality solution cannot be determined rapidly, but the inclusion of the insertion operator, which is able to introduce new genetic material, generally will facilitate the eventual generation of an optimal solution.

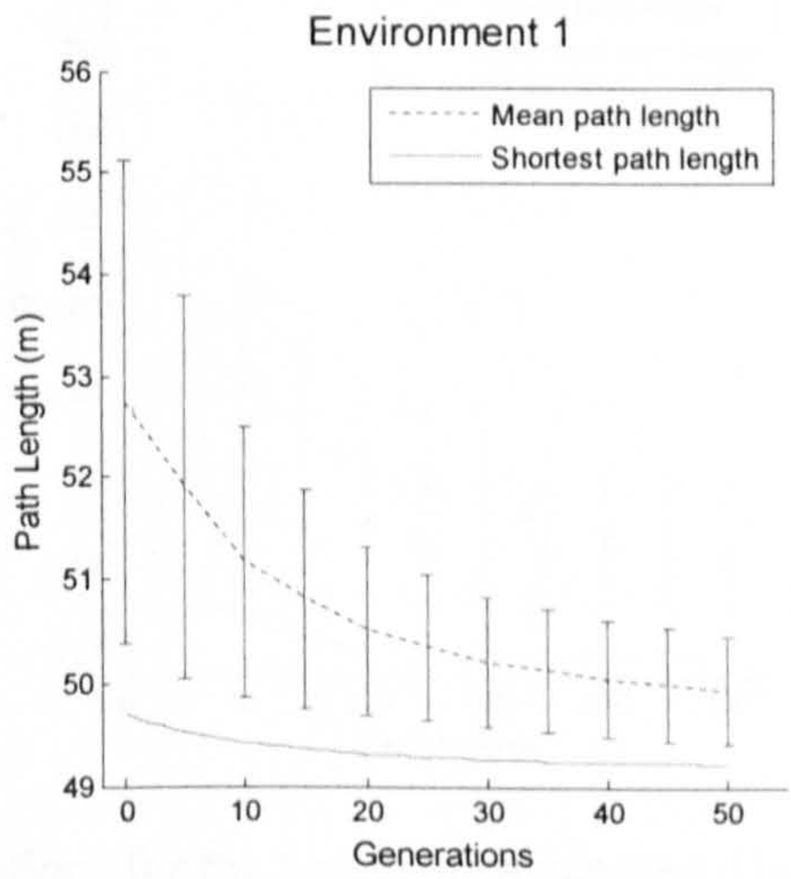
6.7.2 Effect of deterministic crowding on the ability to find the optimal solution

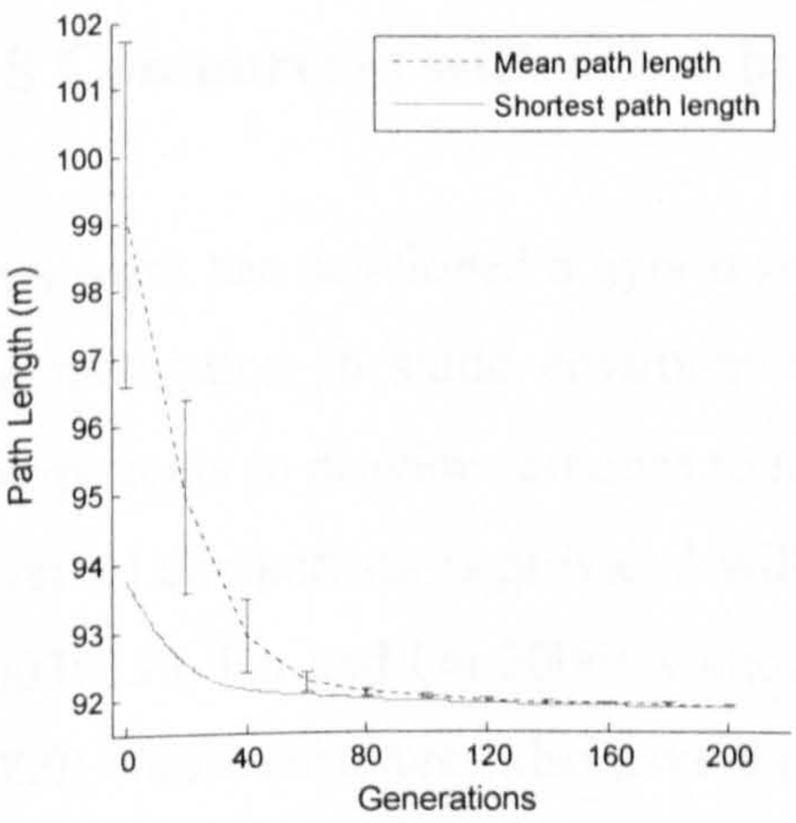
In an attempt to improve performance when population sizes are small, this section investigates the effect of DC (Mahfoud 1995a; Sareni and Krahenbuhl 1998) on the reliability of producing optimal solutions, given a set of supplied waypoints. Table 6.5 shows the number of optimal solutions produced based on the results generated for a population size and a number of generations selected from Figure 6.13 such that the evolutionary computations were completed in approximately one second. It can be seen that DC provides a substantial reduction in the number of occasions on which the GA was unable to produce an optimal path for the given set of waypoints. This improvement was particularly marked in environments 2 and 4, where the failure rate was reduced by a factor greater than 10. The generational diversity is plotted in Figure 6.14 for the four test problems both with and without DC. For environments 2 and 3, it can be seen that DC is able to maintain the diversity present in the initial generation, and, although the results for environments 1 and 3 show some tendency towards convergence when DC is used, the population convergence is far more rapid without DC. Furthermore, it can be seen in Figure 6.14 that the shortest path lengths available in the respective populations for environments 2 and 4 were reduced when using DC, and, in fact, improvements were also observed but were less apparent in environments 1 and 3. In fact, in the general case, for any given generation during the evolutionary process, the best individual produced when using DC was better in terms of path length than the corresponding individual produced without DC.

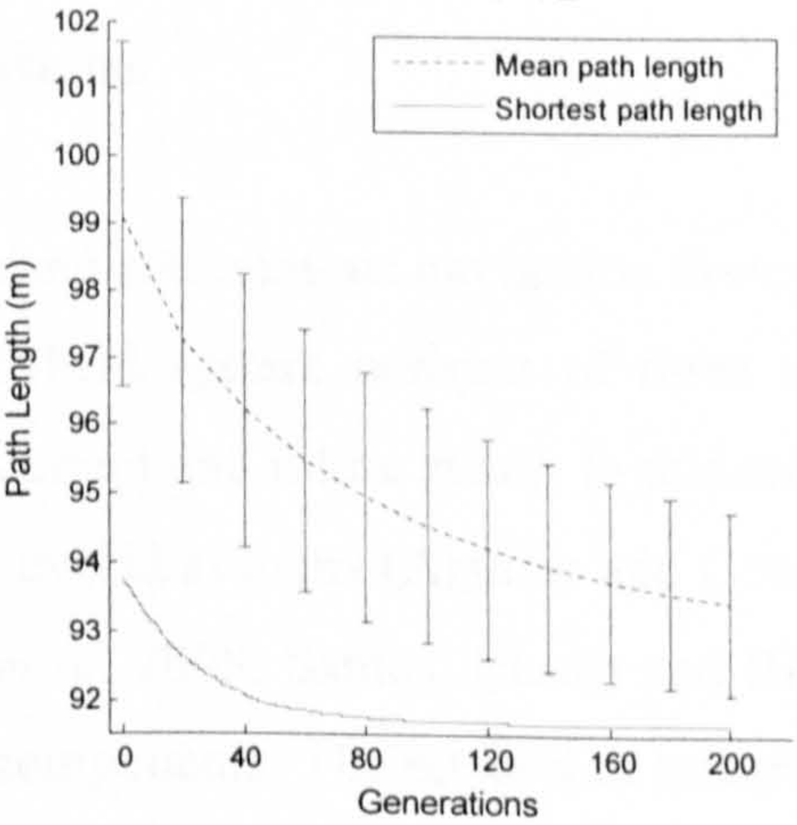
Table 6.5 Percentage of runs that achieve optimality for the four sample environments.

environment	population size	generations	algorithm scheme	failure to achieve optimality over 1000 runs (%)
1	10	50	without DC	8.1
			with DC	3.5
2	20	200	without DC	37
			with DC	2.2
3	60	200	without DC	15.7
			with DC	8.9
4	30	300	without DC	52.0
			with DC	3.5









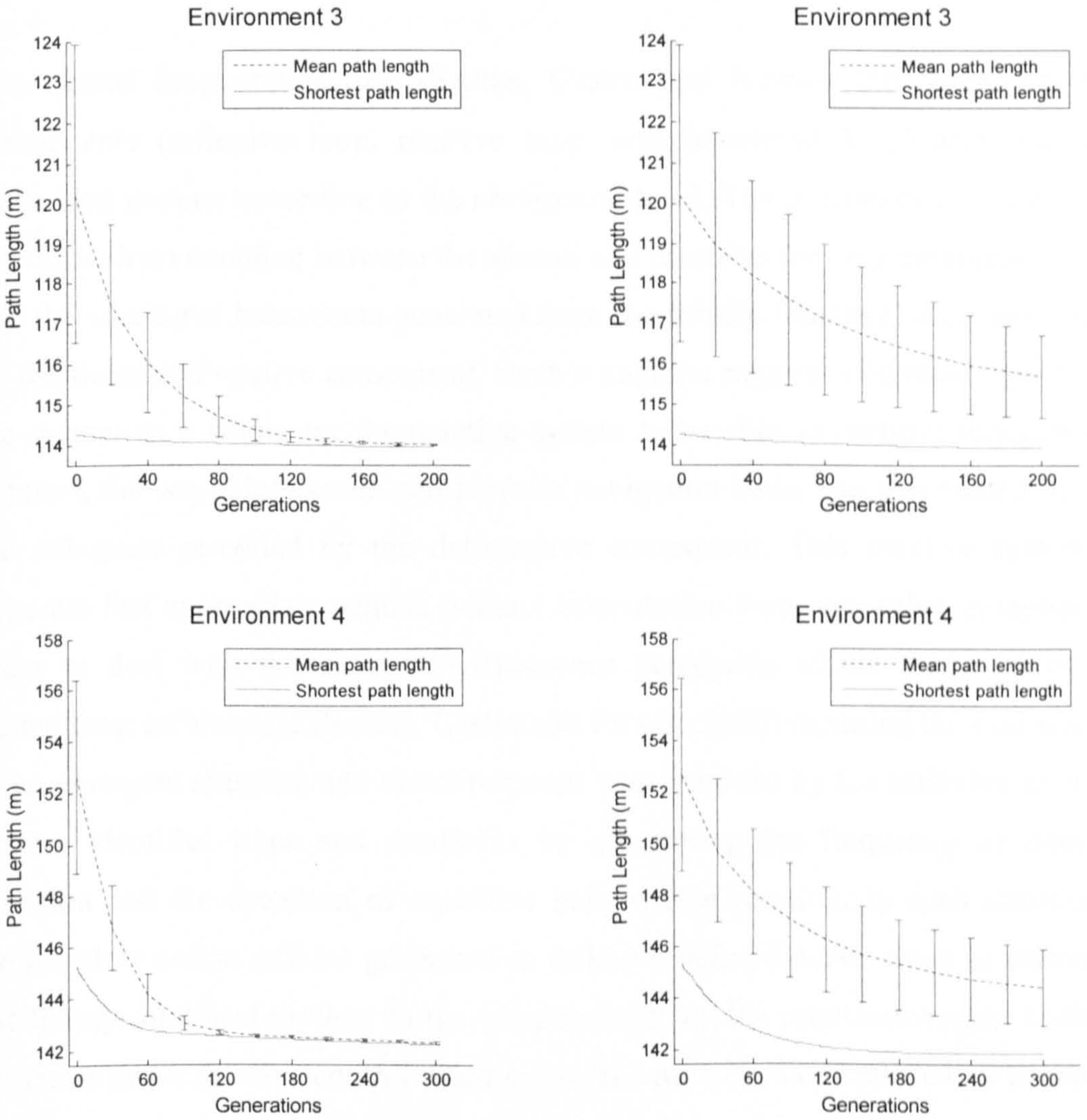


Figure 6.14 The mean values and standard deviations for the populations generated by the waypoint navigator both without DC (left column) and with DC (right column).

6.8 Comparison with other hybrid systems

This work has developed a hybrid waypoint-based autonomous navigation system for the navigation in static environments. The hybrid system consists of three major components to provide response to historical, current and future states. In this section, a verbal comparison is provided with existing hybrid systems (Aguirre and González 2003; Liu, Hu and Gu 2006; Muñoz-Salinas *et al.* 2005; Santos, Castro and Ribeiro 2000) whose structures also have three major components. The method of integration, the content of each major component and the functions of each major component in these earlier hybrid architectures all differ from that of the waypoint system.

The nested loop architecture (Santos, Castro and Ribeiro 2000) had its three components (reflexive loop, reactive loop, and functional loop) arranged in an embedded manner according to the abstraction level. The innermost loop (reflexive) provides direct coupling between the stimuli and response for only emergent situation and the additional behaviours generated from the middle (reactive) loop are directed by the outer deliberative component. Such a solution requires continual operation of the system as a whole for the reactive system to be able to perform navigation. In contrast, the waypoint system can perform navigation under reactive control to fulfil the sub-goals provided by the deliberative component. This reactive system can generate fast appropriate actions without intervention from any other component in order to deal with the current instantaneous perception of the environment. The nested loop architecture (Santos, Castro and Ribeiro 2000) regarded the trap situation as an emergent situation and direct response was provided by the reflexive loop. The system identifies traps and deadlocks by monitoring the frequency of imminent collision and the detection of repetitive halts during rotation. In such situations, a simple slow action will be generated to follow a defined curve so as to prevent an oscillatory rotational motion. In the waypoint system, the robot can escape U-shaped obstacles under reactive control when such a situation can be identified by the current perception, otherwise, waypoints collected previously are used to detect traps and to generate a temporal target to facility the escape. In the nested loop approach, three motion modes are defined in the reactive loop in order to deal with local motion. Such motion can be generated externally by the teleportation mode, or locally by either path following or local navigation modes. This is different from the reactive component in the waypoint system which uses no external intervention and both local navigation and path following can be realised by the reactive system after training. The functional loop in the nested loop architecture is responsible for path planning, localisation and determination of the local navigation strategy whenever the local navigation mode in the reactive loop is selected. In comparison, while the deliberative unit in the waypoint system provides exploration, it is not involved in the decision marking process during the local navigation. The authors of the nested loop architecture realised it was unnecessary to build a detailed map for the environment,

but the strategy was not described in the paper. In the waypoint system, the waypoints that reflect the estimated location and distribution of the obstacles clustered in the environment provide the abstract of the world model. In addition, unexplored waypoints provide heuristics for task specific exploration.

The hierarchical hybrid architecture proposed by Aguirre and González (2003) consists of planning, executive and control layers. A safe shortest path is generated in the planning layer and the executive layer selects appropriate behaviours to fulfil the plan. The control layer is responsible for the control of the robot according to the results of the perception process. The navigation task can be viewed as a process of top down decomposition, with any error occurring in a lower layer being reported to the higher layer. The system produces a topological map that contains metric information. The waypoints in the new system described in this chapter can be viewed as a set of virtual landmarks that are unique in that no duplication is allowed within sensor range and the waypoints themselves contain metric information. The executive layer in Aguirre and González (2003) activates and weights a set of behaviours according to the context of applicability to the current state, with the combination of those weighted behaviours producing the final output. The control layer contains elementary fuzzy behaviours, whereas there is no executive layer to coordinate the different behaviours in the waypoint system. In fact, the two layers, executive and control layers, are effectively fused to form the single reactive layer, since the navigation rules are generated by the training and a single steering action is produced based on the rules. Although the authors claimed that the map can be constructed through exploratory tasks, no detailed explanation of the strategy was provided in the paper, and it appears that, for the purposes of the work presented, a complete map was built that minimised the travel distance between key locations. In contrast, the waypoint system autonomously explores and collects waypoints along with the appropriate metric information. The hierarchical three layer architecture was evaluated only in the indoor environments and no report was given on how the system dealt with deadlock problems.

A three-layer hybrid architecture was proposed by Muñoz-Salinas *et al.* (2005) involving a set of agents allocated in *deliberative*, *execution* and *monitoring*, and *control* layers. The *deliberative* layer generates an appropriate plan for a specified navigation task, which, once received, creates several navigation skills (a skill contains a set of activated behaviours to achieve a sub-goal) in the *execution* and *monitoring* layer in order to achieve a sequence of sub-goals for the plan. At the same time, the *execution* and *monitoring* layer examines the implementation of the plan for any failure in its implementation. The *control* layer contains a set of agents that implements fuzzy and visual behaviours. The navigation system requires a topological map that includes geometric information, and the authors state that the map can be either supplied externally or constructed autonomously, but no detailed description of this process was provided in the paper. The navigation task can be considered to be decomposed by layer, with the function of the middle layer being the translation of the plan generated in *deliberative* layer into a sequence of skills. In practice, the navigation skill is effectively organised into primitive behaviours in order to achieve sub-goals. This arrangement is different from that in the waypoint architecture, where the reactive component requires no specific layer to arrange the individual behaviours, since their coordination was facilitated during the training period. The three-layer hybrid architecture was tested in office-like environments with a complete topological map, whereas, in the waypoint system, only the waypoints related to the current navigation task (current, initial and final locations) are recorded and serve as a basis for planning.

Liu, Hu and Gu (2006) developed a hybrid architecture containing three layers, namely (in order of hierarchy) cognitive, behaviour, and swim pattern. Instead of generating a path, the cognitive layer infers a set of parameters to coordinate the behaviours in the behaviour layer. To achieve this, states reflecting physical events are abstracted from sensory information rather than from any spatial representation of the environment. The swim pattern layer is designed specifically for robotic fish navigation and converts the behaviours into the control of joints of the robotic fish. The representation adopted by this hybrid system was highly abstracted according to physical events, whereas the world model used in the waypoint system can be

regarded as being a behaviour driven representation, in that the waypoints indicate the locations where the significant changes in the behaviour occur. This may cause undesirable behaviour if any errors occur in the cognitive layer, since the combination of the primitive behaviours is also directed by the cognitive layer. It is unclear whether local minima exist in the robotic fish navigation, as this issue was not discussed in the paper. In the behaviour layer, the *wander* behaviour performs random exploration in the environment (a fish tank). In this way, a complete map can be generated, but it is inefficient in that the same area may be explored for many times. In contrast, task-oriented approach was used in the waypoint system.

Apart from in the nested loop architecture, the lowest layer in the hybrid systems found in the literature are closely related to the low-level implementation of the behaviours to drive the robot, but this transformation was performed in the reactive unit in the waypoint system. Two of the above architectures were evaluated only in indoor environments and none created its own a detailed world model, but where present, topological maps with metric information were built. The waypoint representation is highly abstracted and is even able to avoid the modeling of areas not related to the current task.

6.9 Conclusions

A waypoint navigation system has been developed for mobile robot navigation and compared with two evolutionary planning techniques, namely EP/N and a vertex planner. That suitable paths can be generated for the sample environments confirms that the waypoint system is able to navigate appropriately to the goal and to collect waypoints during the exploration. The waypoint technique is able to reduce the search from the initially large physical area to just a small number of representative points previously traversed under reactive navigation. Only these waypoints then need to be considered in producing a suitable plan to reach the goal. The statistical results obtained by repetitive measurement confirm that the waypoint technique is able to significantly reduce the time taken to produce a plan. In the EP/N and vertex planning systems, determining whether a plan is feasible is time consuming, but is unnecessary

in the waypoint method as the paths between pairs of waypoints are already known to exist.

The global optimisation that can be achieved by EP/N and the vertex planner is clearly evident in the path length results. However, this can only be achieved by supplying complete knowledge of the environment, and this is unrealistic in most applications and does not allow general solutions to the robot navigation problem. The new waypoint method does not require any prior knowledge of its environment, (which becomes gradually more familiar to the robot through exploration), keeps minimal information regarding the nature of its surroundings and rapidly generates suitable paths. The waypoint system is extremely robust in terms of its ability to find a suitable path and in terms of response time, as it is always able to resort to reactive navigation should a plan not be available in a timely fashion or if operating in a dynamic environment.

The effectiveness of the proposed planning algorithm has been investigated in a series of navigation problems of varying complexity. In the simpler environments, only short evolutions are needed to produce an optimal solution given a set of supplied waypoints and the use of GAs is probably not required. However, in problems of greater complexity, the use of a GA in finding suitable combinations of waypoints to solve the navigation problem is appropriate. The frequency at which optimal solutions could be produced in the more complex environments was shown to be improved significantly through the use of deterministic crowding.

The hybrid structure has been evaluated by comparison to a number of recent hybrid architectures reported in the literature. The waypoint navigation approach has been arranged in such a manner as to take into account past, current and future information. The system is robust and real time in that the basic reactive navigation can be realised by tight coupling between stimuli and responses without any intervention from other components. Furthermore, the memory requirement is kept low as only a set of waypoints need be stored. Efficient exploration is performed in the more promising

areas closely related to the current navigation task. The plan generated is a sequence of waypoints (sub-goals) that allows local adaptation during robot movements.

The following chapter investigates an enhanced version of the waypoint navigation approach that involves reactive interaction with moving obstacles.

Chapter 7

WAYPOINT-BASED NAVIGATION IN DYNAMIC ENVIRONMENTS

The waypoint-based system introduced in the previous chapter concerns navigation in static environments. This chapter extends this navigation system, so that it can also operate in dynamic environments containing multiple moving obstacles. To achieve navigation in dynamic environments, two novel extensions were made with respect to the approach described in the previous chapter. The first extension equips the robot with the ability to avoid dynamic obstacles, based on the identification of the necessary properties of a single dynamic obstacle which must be satisfied in order for the robot to guarantee avoidance. The second extension uses the information gathered over a series of exploration tasks to develop a statistical model of the extent to which paths between pairs of waypoints are disrupted by the presence of dynamic obstacles. However, this does not imply that the motion of the obstacles are constrained. If the obstacles move randomly in the environment, the navigation behaviour may not be improved as few heuristics are provided by the statistical model. The work described in this chapter is being prepared for submission to the International Journal of Robotics Research.

The identification of waypoints utilises the same strategy as that in the previous navigation system, but is constrained to static obstacles only. This is a common strategy for navigation in dynamic environments (for example Low, Leow and Ang Jr 2002 and 2003; Minguez and Montano 2005; Minguez, Montesano and Montano 2004; Urdiales *et al.* 2003a and 2003b; Vazquez-Martin *et al.* 2006) where paths are first generated globally for the static aspect of the environment and avoidance is achieved locally. This chapter introduces related works, the proposed navigation system, gives a description of each module and presents the results obtained for the three separate experiments designed to verify the two extensions proposed. A comparison with other hybrid architectures found in the literature is then presented. A discussion of the results obtained for the navigation system is provided before the chapter is concluded.

7.1 Related work in hybrid systems in dynamic environments

The previous chapter included a description of recent works related to the hybrid architecture for navigation in static environments and only previous works related to navigation in dynamic environments (those containing moving obstacles) are presented in this section. Following a short survey of hybrid systems in dynamic environments, two extensions made to the hybrid system introduced in the previous chapter are discussed in greater detail.

Low, Leow and Ang Jr (2002 and 2003) proposed an architecture that integrated planning and motion control. The planning module produced a sequence of checkpoints between the start and end points using cell decomposition, whereby those cells that can be connected vertically or horizontally are assigned the same label and a greedy method is applied that searches the grid map marking as checkpoints those closed points along the boundary between cells of different labels. Clearly, the generation of checkpoints requires an *a priori* grid map. The reactive module consists of three lower-level modules: target reaching, obstacle avoidance, and homeostatic control. A neural network in the target reaching module produces a sequence of control commands to guide the robot between a pair of checkpoints, the obstacle

avoidance module keeps the robot away from sensed obstacles and the homeostatic control module maintains internal stability by monitoring the internal states. The final output of the reactive module is a combination of the commands originating from the three modules. The work was extended by the authors (Low, Leow and Ang Jr 2006) to the navigation of multiple robots.

Urdiales *et al.* (2003b) proposed a hierarchical hybrid architecture with four layers that are neither strictly deliberative nor reactive scheme. The *geometrical modelling layer* uses on-board sensors to create a grid map from which a topological map is abstracted by the *topological modelling layer*. A global collision-free path is generated in the *route planner* and the path generated is decomposed into a set of sub-goals for tracking by the *local navigation layer*, that is implemented by a case-based reasoning (CBR) technique. One of the significant drawbacks of CBR is that the robot may fail to generate an appropriate response when a significant discrepancy exists between the current situation and the cases stored (Liu *et al.* 1994; Ni *et al.* 2003; Park, Kim and Chun 2006). A solution proposed by Urdiales *et al.* (2003b) is to use a simplified version of a potential filed methods and to combine the generated force vector with the closest matched case and store the outcome as a new case. The system was evaluated through experiment for indoor environments with both static and moving obstacles, but the results when avoiding moving obstacles was not clearly documented. A slightly different version of this architecture was reported in an earlier work (Urdiales *et al.* 2003a) using a potential field method as a reactive scheme rather than CBR. A similar architecture was used for outdoor environments in the work described by Vazquez-Martin *et al.* (2006).

A hybrid system described by Minguez and Montano (2005) and Minguez, Montesano and Montano (2004) consisted of three components: model builder, planning and reactive motion. The objective of the model builder is to construct a grid map, incrementally constructed from a moving binary grid of a fixed size centred on the robot. The planner computes a path by gradient search on the grid map whose cells are labelled with the values of the distances from the goal point to the cell, and collision-free motion is generated by the reactive navigation module. The nearness

diagram (ND) (Minguez and Montano 2004 and 2005; Minguez, Montesano and Montano 2004; Minguez, Osuna and Montano 2004; Montesano, Minguez and Montano 2005) navigation method is used in this module to match a configuration of five pre-defined configurations and to guide the robot more closely so that its configuration corresponds to one of the five configurations. The sensory information flows into the model builder and is subsequently directed to the planner and reactive navigation module. The output from the planner guides the reactive navigation. In certain cases, the planner may fail to deliver a plan to the reactive module, and the local navigation will rely solely on the reactive method. In the synchronous planner-reactor architecture (see section 3.3), both the planner and the reactive module operate on the current model to provide a prediction for the immediate future, however the resulting trajectory is unlikely to be optimal as no past experience is explicitly considered in the generation of a plan. No exploration component was incorporated into the system for map building purposes and the grid map will normally require more memory in comparison with the waypoint system that records only a small number of points. The ND reactive method can be used to escape the traps arising from U-shaped obstacles, but is limited to those obstacles whose characteristics can be identified by the sensor at single sampling point.

In dynamic environments, it is normally unrealistic to have *a priori* knowledge regarding the motion of the obstacles. Therefore, following a collision-free path that has been generated before navigation commences (examples can be found in section 3.1.2) is not feasible. To avoid moving obstacles in dynamic environments requires timely monitoring of environments using an appropriate configuration of sensors. Suitable avoidance manoeuvres can be achieved by modifying the steering angle or the velocity of the robot. The methods that have been used to compute the desired steering commands are CBR (Kira and Arkin 2004; Urdiales *et al.* 2003a, 2003b and 2006) and ND (Minguez and Montano 2004 and 2005; Minguez, Montesano and Montano 2004; Minguez, Osuna and Montano 2004; Montesano, Minguez and Montano 2005). CBR techniques create a set of cases for the situations encountered during the training stage and during actual navigation and the instantaneous reaction is determined by the actions associated with the cases that most closely match sensory

information. When the case base lacks a sufficient case, a solution recalled from a secondary case is unlikely to be appropriate (Liu *et al.* 1994; Ni *et al.* 2003; Park, Kim and Chun 2006). An alternative approach is to build a large case base, but this requires additional memory. Another major drawback of CBR techniques is high sensitivity to noise (Ni *et al.* 2003; Park, Kim and Chun 2006), potentially resulting in the failure to deliver a final appropriate solution. The ND method generalised five configurations for obstacles avoidance and subsequent matching with respect to the five general cases allowed a steering action to be computed from the formulas associated with each case. These approaches have been integrated into a hybrid architecture (Minguez and Montano 2005; Minguez, Montesano and Montano 2004) for dynamic environments. Section 7.6.3 exams this method in detail. The avoidance of obstacles using robot velocity control has been investigated using dynamic windows (DW) (Fox, Burgard and Thrun 1997; Ogren and Leonard 2002 and 2005; Stachniss and Burgard 2002), velocity obstacles (VO) (Fiorini and Shiller 1998; Large, Laugier and Shiller 2005) and vector field histogram (VFH) (Borenstein and Koren 1991; Ulrich and Borenstein 1998 and 2000). A description on DW, VO and VFH was provided in section 3.2.1. Although good avoidance performance can be achieved by those approaches (both DW and VO) at high velocities, they generally decompose the continuous velocity space that guarantees collision avoidance into a discrete set of velocities, so that the search for an optimal velocity command can be performed quickly. However, the trade-off between the solution resolution and the time for computing such solution may, in practice, be difficult to make, as the VO varies in size and shape. In addition, DW is liable to become trapped in a local minima (Kunwar and Benhabib 2006; Stachniss and Burgard 2002). A number of authors (Ge and Cui 2002; Kurihara *et al.* 2005) incorporated velocity information regarding the robot and obstacles to construct an artificial potential field (PF). The local potential around the robot was constructed at regular intervals in order to provide an adequate response to avoid unexpected moving obstacles. Such a repetitive construction of the potential field containing velocity information is time consuming. The performance of the DW, PF, and VFH approaches largely depends on a number of parameters that are difficult to optimise for general application (Urdiales *et al.* 2006), and, although those approaches are efficient in that the

commands for the next step can be generated at high frequency, their purely sensor-based approach results in a sub-optimal solution in that only the current state is considered.

The behaviour-based avoidance techniques, such as fuzzy logic (for example Malhotra and Sarkar 2005; Zhu and Yang 2004), neural networks (for example Kubota 2004; Low, Leow and Ang Jr 2003; Min 2005), are different from the above-mentioned approaches in that direct mapping between sensor inputs and action output is established through an appropriate learning process performed either offline or online. Once moving obstacles are detected, several primitive behaviours are activated and coordinated to produce a single action without explicit reasoning. However, the training processes involved are often complicated and the training samples may be so large that they cannot be guaranteed to be limited to a pre-defined memory constraint. The coordination of the different activated behaviours needs careful design to eliminate poor behaviour. Additionally, the number of rules generated by fuzzy-logic controllers increases rapidly with the number of input variables, slowing the search for appropriate rules to be activated and consuming additional memory, both of which are undesirable in dynamic environments (Yang, Watanuki and Zhao 2005). Decision-tree based reactive techniques (Cocora *et al.* 2006; Shah-Hamzei and Mulvaney 2000; Swere, Mulvaney and Sillitoe 2004) have been successfully applied to navigation problems in static environments, but the literature contains no specific reports on moving obstacle avoidance. To generate an appropriate decision tree requires a learning process which takes into account the motion features (such as direction of motion and velocity) of the obstacles in addition to the geometrical ones (such as the distance between the robot and the obstacles).

Those approaches introduced above generate an action only for next step and no look ahead is involved, with the potential of allowing the robot to become caught in a local trap and so preventing it from achieving its objective. Instead of computing the next movement, some approaches (for example Minguez *et al.* 2001; Stachniss and Burgard 2002; Ulrich and Borenstein 2000) attempt to compute a sequence of movements for several steps ahead, depending on the time available. Ulrich and

Borenstein (2000) developed VFH* which analyses the consequence of each candidate motion direction through a look-ahead verification technique in an attempt to eliminate the possibility of being caught in a local trap. Planning was incorporated into the ND reactive scheme (Minguez *et al.* 2001) and the DW method (Stachniss and Burgard 2002) to improve the performance of the local navigation and avoid local minima. These approaches effectively extend the reactive schemes into a hybrid architecture, but the grid representation of the environments adopted in those approaches requires proper determination of the trade-off between the number of cells in the grid map and the map resolution, since a fine resolution consumes additional memory space and requires computational overheads, but produces a solution better equipped to avoid narrow gaps between obstacles. The waypoint-based moving obstacle avoidance strategy introduced in this chapter establishes constraints on the movements of the dynamic obstacles so as to reduce the occurrence of inevitable collisions (that is, where avoidance of the moving obstacle as detected by a specific sensor configuration is impossible whatever action is taken by the robot). In contrast to the behaviour-based approaches, no training is required to deliver the avoidance solution and it can be applied in completely unknown environments even when a model of the static aspects of the environment is unavailable. Moreover, the trajectory generated is still optimal as the sequence of avoidance actions are determined by look-ahead verification. The planning methods introduced in chapters 4 and 5 are not appropriate for this purpose as the dynamics of the robot (the robot is unable to rotate without any translational displacement) constrains the actions to be taken to avoid the moving obstacles. Although the avoidance algorithm was developed for a single moving obstacle, it is possible to generate avoidance manoeuvres by selecting the intersection of the avoidance commands for each moving obstacle perceived concurrently (if the intersection is not empty).

The exploration schemes reported in the literature have been mainly designed for map building purposes. It would not be appropriate to record the geometrical information of the moving obstacles in a global frame as such information rapidly becomes out of date. The statistical exploration technique introduced in this chapter is novel in that it

is different from previously reported methods since it summarises the dynamic temporal features of the moving obstacles.

7.2 Waypoint navigation system for dynamic environments

The waypoint navigation system shown in Figure 7.1 consists of three modules.

1) The low-level control is performed by the *static behaviour module*. In the current work, this is implemented as a decision tree generated by a frequency-table based learning technique (Mulvaney *et al.* 2005; Swere, Mulvaney and Sillitoe 2004). This reactive solution has been adopted due to the availability of the source code and access to the designers of the approach.
2) High-level behaviours in the *dynamic avoidance module* respond to the detection of moving obstacles by favouring directions which avoid collisions. Information regarding the locations of the static obstacles is used to assist in the selection avoidance direction. In dynamic environments, the robot should be able to react to the presence of a moving obstacle in order to complete the navigation task. It would not be sufficient to use those avoidance techniques employed in the static behaviour module, as the assumption made was that the attributes of the environment were fixed.
3) The *deliberative module*, a high-level control unit, manages the navigation at a global level. It consists of three sub-modules, namely localisation, exploration, and planning each of which operates on a set of waypoints which in turn, indirectly, represents the position of the static obstacles within the environment. The planner plays a tactical role by providing a sequence of sub-goals to the robot giving the remaining modules freedom with regard to the local movements. Due to the time variant characteristics of moving obstacles, locations detected previously become invalid when producing a future plan. However, the dynamic aspects of the environment are indicated in the statistical exploration model.

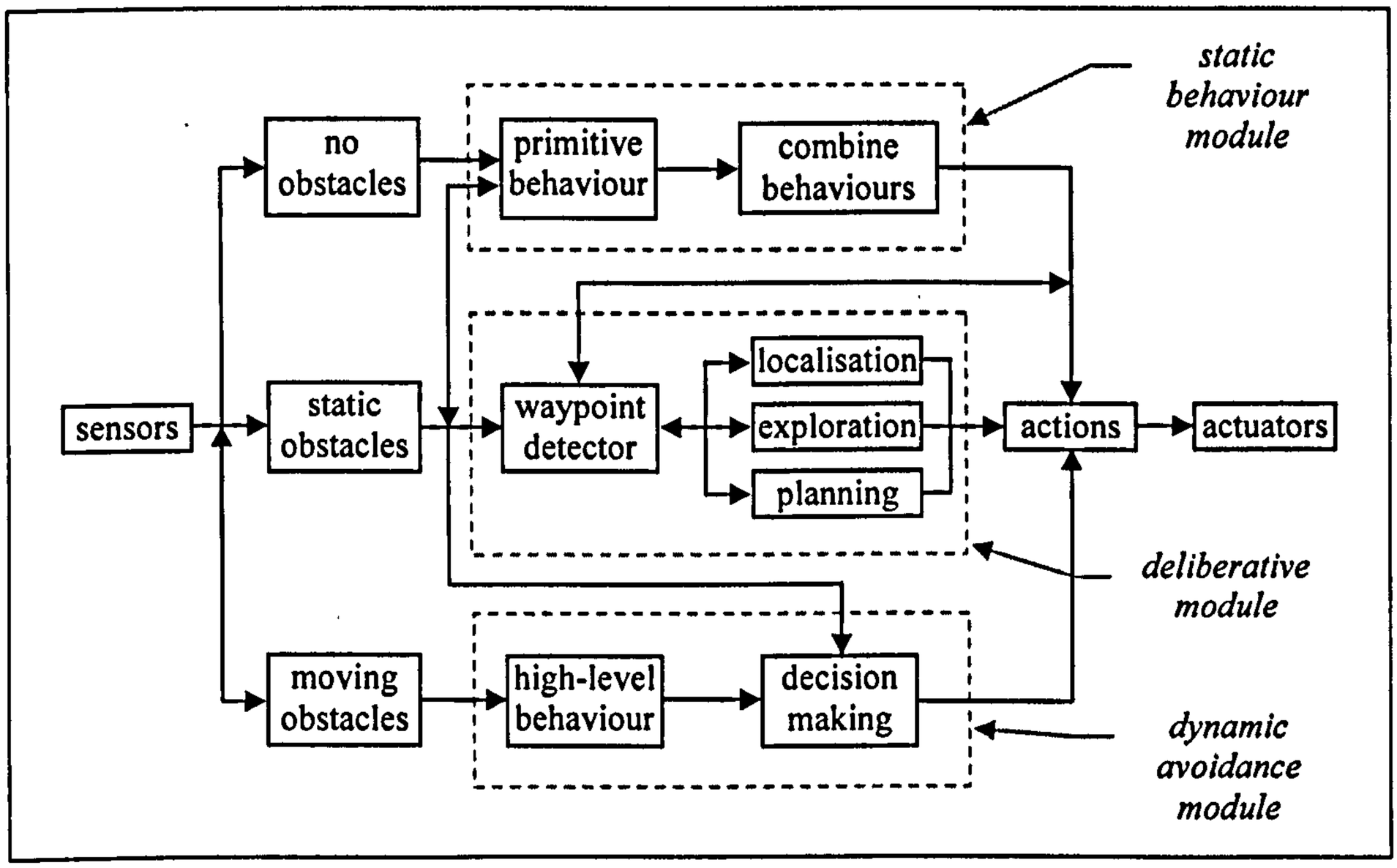


Figure 7.1 The architecture of the waypoint navigation system.

The development of this architecture mainly relies on the partitioning of the environment into three different types: free space, static obstacles, and moving obstacles. None of the architectures reported in the literature (see section 7.1 for a survey and further discussion on those architectures is provided in section 7.7) was constructed based on such a characterisation of different types of environmental information. This architecture is somewhat different than that introduced in the previous chapter, since, in this hybrid system, the modules dealing with the past and future states have been combined into one module mainly to account for the static obstacles in the environment. However, this hybrid architecture inherits the close connection from history to current to future that was used as the basis for the development of the system in the previous chapter. There are two principal reasons to develop such hybrid architecture: 1) three different characteristics of the environment need to be addressed independently; 2) the past experience, the current practices, and future prediction should be considered and combined into a single hybrid solution to create high-level intelligence.

Each of individual modules shown in Figure 7.1 is discussed in greater depth in later sections, but a brief overview of the entire architecture is given here for clarity. In an environment which has not previously been encountered, and assuming no obstacles

are detected, the robot's initial behaviour will be to seek a goal specified by deliberative module. Once an obstacle is encountered, the resulting avoidance behaviour is dependent on the motion and sensed dimension of the obstacle. If the obstacle is stationary, the static behaviour module is activated, whereas whenever moving obstacles are detected (regardless of whether any static obstacles are perceived simultaneously), the dynamic avoidance module is activated. During navigation, when the static behaviour module is activated, the current sensory information and the corresponding actions are taken reactively and these are transmitted to the deliberative module for interpretation by the waypoint detector. The waypoint detector's function is to save selectively, from the stream of sensory data, a set of locations as waypoints for further reference. A location is selected as a waypoint whenever the robot must deviate from its current path due to the presence of a static obstacle, in order to circumnavigate it. The set of waypoints is used later for exploration, localisation, and planning. When the robot is free of any specified navigation tasks, the exploration function can be activated to discover uncharted regions, and thereby gather additional waypoints. Exploration behaviour can be interrupted by any task that has a higher priority. In an environment that has previously been encountered, the planner uses its current knowledge of the environment to generate an optimal or near optimal path as a sequence of waypoints.

7.3 Static behaviour module

The reactive control undertaken when the robot is within stationary surroundings is realised within the static behaviour module, as a combination of trained behaviours. The same behaviour is used during exploration and when navigating between the waypoints generated by the deliberative unit, having the effect of directing the robot toward the desired goal while avoiding potential collisions. The method employed in this module was detailed in section 3.2.2. Note that this module was designed for behaviour-based reactive approaches, even though no explicit coordination mechanism is required by the employed method, as this is carried out in the training process.

7.4 Dynamic avoidance module

Since the navigational skills learned in static environments are insufficient to avoid collisions in dynamic environments, additional high-level behaviours are needed. It is recognised that conditions within dynamic environments may arise such that a collision is unavoidable, no matter what the robot's reaction. Such collisions are referred to as inevitable collisions in the following discussions. Inevitable collisions are mainly due to the limitations of sensor system, as their range of operation is restricted and the ability to consider all avoidance solutions is constrained by computational capability. In order to reduce the number of inevitable collisions, a number of cases have been investigated in this chapter in which the relative position and movements of the robot and a single moving obstacle were used to determine the constraints on the dimensions and maximum velocity of the obstacles. However, the inevitable collisions may still exist if multiple moving obstacles simultaneously advance towards the robot. This does not mean that the application is restricted to avoid single obstacle. Earlier work (Fiorini and Shiller 1998; Fox, Burgard and Thrun 1997; Large, Laugier and Shiller 2005; Ogren and Leonard 2005; Stachniss and Burgard 2002) chose avoidance commands from the space that contains all collision-free solutions, but none discussed whether or how the solution space was constrained to be non-empty and the response that resulted if solution space was empty. In Yang, Watanuki and Zhao (2005), the condition of collision avoidance was established by considering a single obstacle moving in an arbitrary direction that remained unaltered in order to estimate the collision condition. The concept of inevitable collision states was introduced by Fraichard and Asama (2004) to be one in which a collision between the robot and an obstacle will eventually occur whichever future trajectories are followed. Once inevitable collision states have been identified, an avoidance solution can be determined so as to ensure collision-free movement. However, no constraints on the obstacle movements or dimensions were described. The avoidance algorithm introduced in this chapter for moving obstacles can be applied to multiple moving obstacles simultaneously, since the avoidance actions can be determined by choosing the intersection of the set of actions for each moving obstacle. This strategy was also adopted by the VO approaches (Fiorini and Shiller 1998; Large, Laugier and

Shiller 2005) to account for similar circumstances. In this chapter, a path is referred to be feasible if it is collision-free. The following section derives the necessary constraints and then describes the additional behaviours required to avoid moving obstacles.

7.4.1 Constraints on moving obstacle dimensions and velocity to avoid collision

The approach taken here, as shown in Figure 7.2, is to divide the robot's sensor field of view into three sub-ranges ($R_1 > R_2 > R_3$), namely maximum sensor range R_1, detection range R_2, and effective range R_3. The maximum sensor range R_1 is the maximum distance that the robot's sensor can detect within an acceptable confidence level. When the static obstacle falls within the effective range R_3, reactive control in the static behaviour module is used to navigate the static obstacle without collision. This range, assigned by the user, should be larger than the minimum turning radius R of the robot to avoid any possible contact with obstacles. However, large values of R_3 may lead to the robot following a much longer path than the robot needs to take to circumnavigate obstacles. The detection range R_2 needs to be specified by the user before navigation takes place. When a moving obstacle enters the detection range, the robot will evaluate whether a collision may occur if the current path is maintained, or whether a new collision-free path to avoid the obstacle needs to be produced. The range R_2 should be greater than the effective range R_3, as the motion of the obstacle needs to be taken into account to ensure a collision-free path. On the other hand, it is apparent that this range is bounded within the maximum sensor range R_1. Note that the ability to avoid moving obstacles is dependent on the value of R_2 for a given R_1 and R_3 and is defined to be that point at which the robot must make a decision regarding avoiding the moving obstacle. To be able to make this decision, the robot must be able to gain knowledge of the entire width of the moving obstacle. Therefore, the widths of the moving obstacles in the environment have to be constrained to ensure that the robot is able to estimate whether the robot's current path leads to a collision-free traverse across the moving obstacle, together with the velocity information about both the robot and the obstacle when the moving obstacle enters the detection range. The maximum width of the moving obstacle can be determined given the values of R_1 and R_2.

Modelling the obstacle as a bounding rectangle (as shown in Figure 7.2), the maximum width of the moving obstacles can be deduced by minimising the following equation with respect to the angle θ, the vertex of which is the centre of the robot, one side of which is opposite to the moving direction of the obstacle, and the other side of which is the straight line from the robot to the nearest obstacle vertex.

$$W = \sqrt{R_1^2 - (R_2 \cos\theta)^2} - R_2 \sin\theta \quad (0 \le \theta \le \pi/2) \qquad \text{Equation 7.1}$$

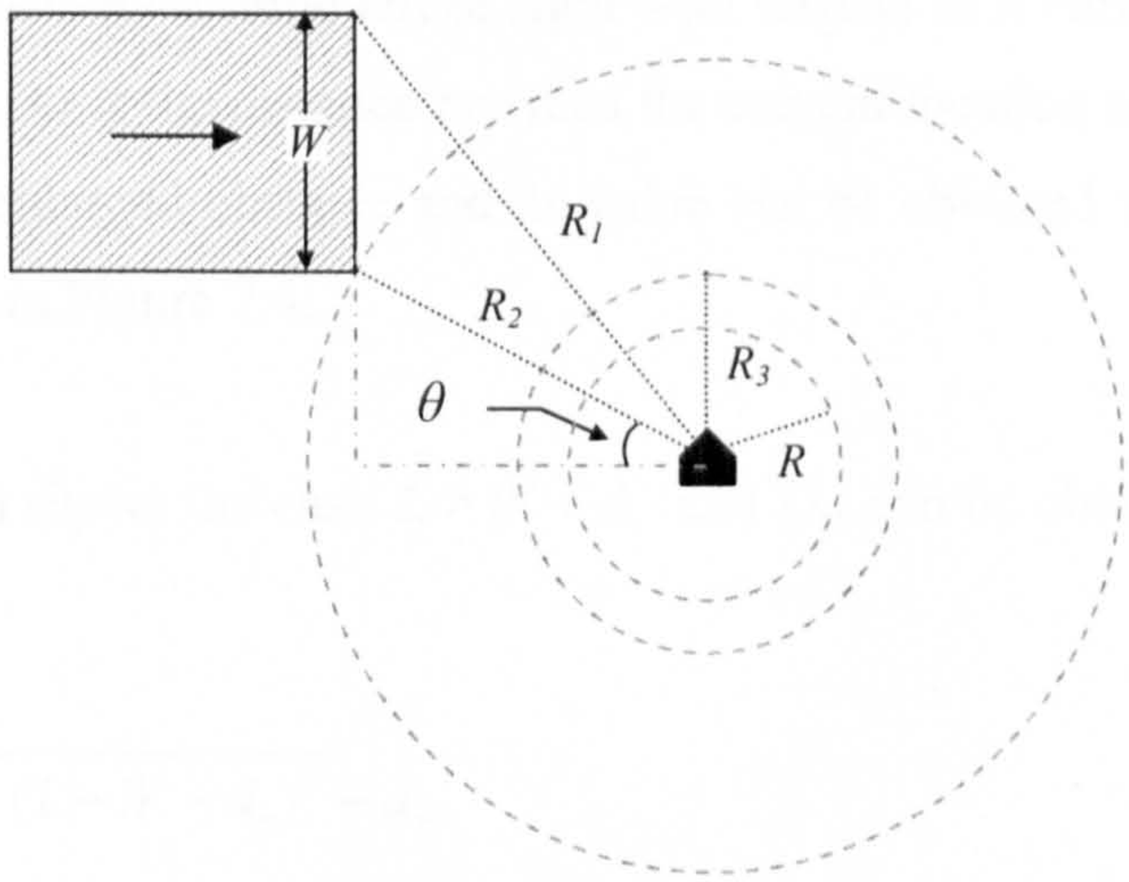


Figure 7.2 Determination of the maximum width W of the moving obstacle.

Given the lateral dimension W of the moving obstacle, Figure 7.3 illustrates the determination of the range of the obstacle velocities that can be permitted without an inevitable collision given two assumptions. The first assumption is that the moving obstacle has only a simple translational motion with respect to the robot while any part of the obstacle is within R_2. The second is that the maximum velocity of the obstacle can be measured. The range of the velocity of the obstacle is determined with respect to the angle, α, subtended between the robot heading and the moving direction of the obstacle with the vertex of the angle being the centre of the robot. Here we consider only the situation where the robot moves towards the moving obstacle ($\pi/2 < \alpha \le 3\pi/2$), this range of α being critical when determining the physical characteristics of the moving obstacle.

Figure 7.3 shows an example of a robot of cylindrical cross-section and diameter d_r moving at the velocity V_r, rotational velocity ω_r with a turning radius, R, as it encounters an obstacle of width W. The obstacle, shown as a solid rectangle, moves at a velocity V_b in the direction opposite to the component of the velocity of the robot which is projected along the longitudinal dimension of the obstacle. The rectangle with a grey border is the obstacle enlarged by d_r on three sides, allowing the physical dimensions of the robot to be ignored in the calculations.

For the case illustrated in Figure 7.3, the robot may be forced to give way to the moving obstacle by turning left or right with respect to it current heading by an angle of β_{ol} or β_{or}. D_{rb} is the distance between the current location of the robot and the front edge of the enlarged obstacle and its value can be obtained with respect to the three cases shown in Figure 7.4.

Figure 7.4(a) shows the case $L > W + d_r$ and D_{rb} can be obtained from the following equation

$$D_{rb} = \sqrt{R_2^2 - (L - W - d_r)^2} - d_r \qquad \text{Equation 7.2}$$

For the case $d_r \le L \le W + d_r$, shown in Figure 7.4(b), D_{rb} is as follows:

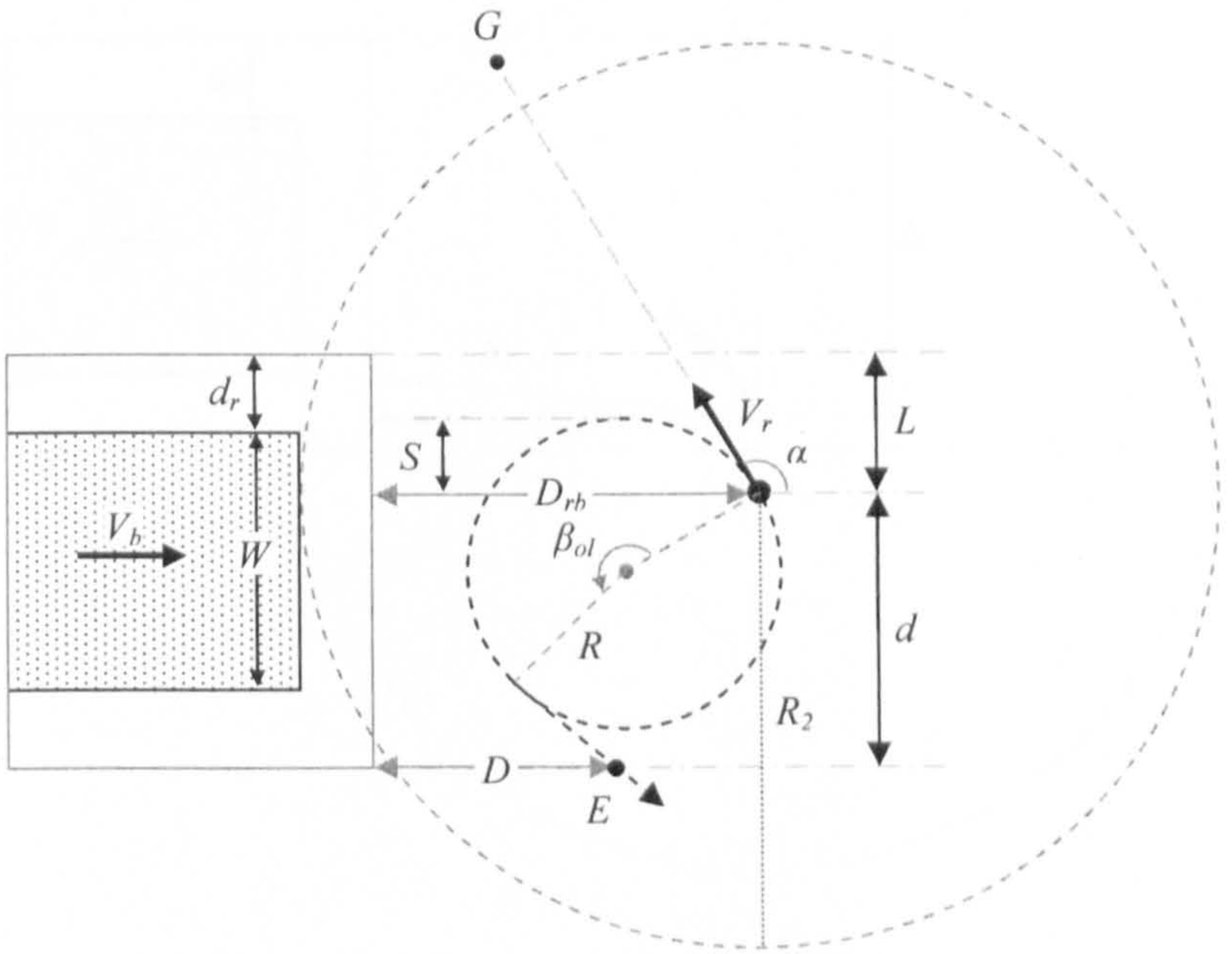
$$D_{rb} = R_2 - d_r \qquad \text{Equation 7.3}$$

The third case, as shown in Figure 7.4(c), is $L < d_r$, and D_{rb} can be calculated by Equation 7.4.

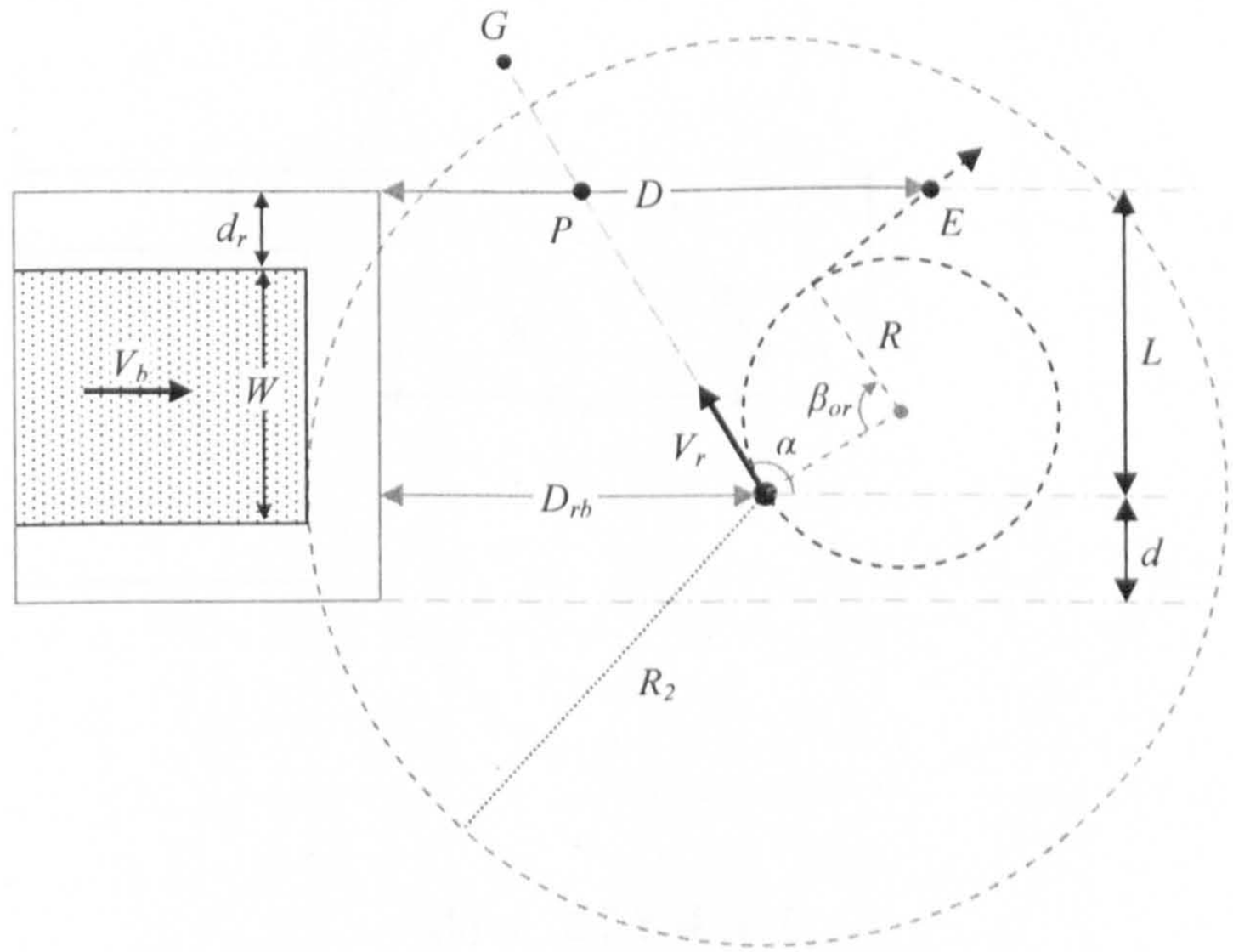
$$D_{rb} = \sqrt{R_2^2 - (d_r - L)^2} - d_r \qquad \text{Equation 7.4}$$

The distance D indicates the distance that the leading edge of the obstacle moves from the position when it is first sensed in the detection range to the escape point E, as illustrated in Figure 7.3. The escape point, marked as E in Figure 7.3, is where the robot completely leaves the path of the moving obstacle, following the modified path in an attempt to avoid the moving obstacle. L represents the lateral distance from the robot's current location to the path edge of the moving obstacle nearest to the goal

point, and d is the lateral distance to the path of the moving obstacle furthest from the goal point.

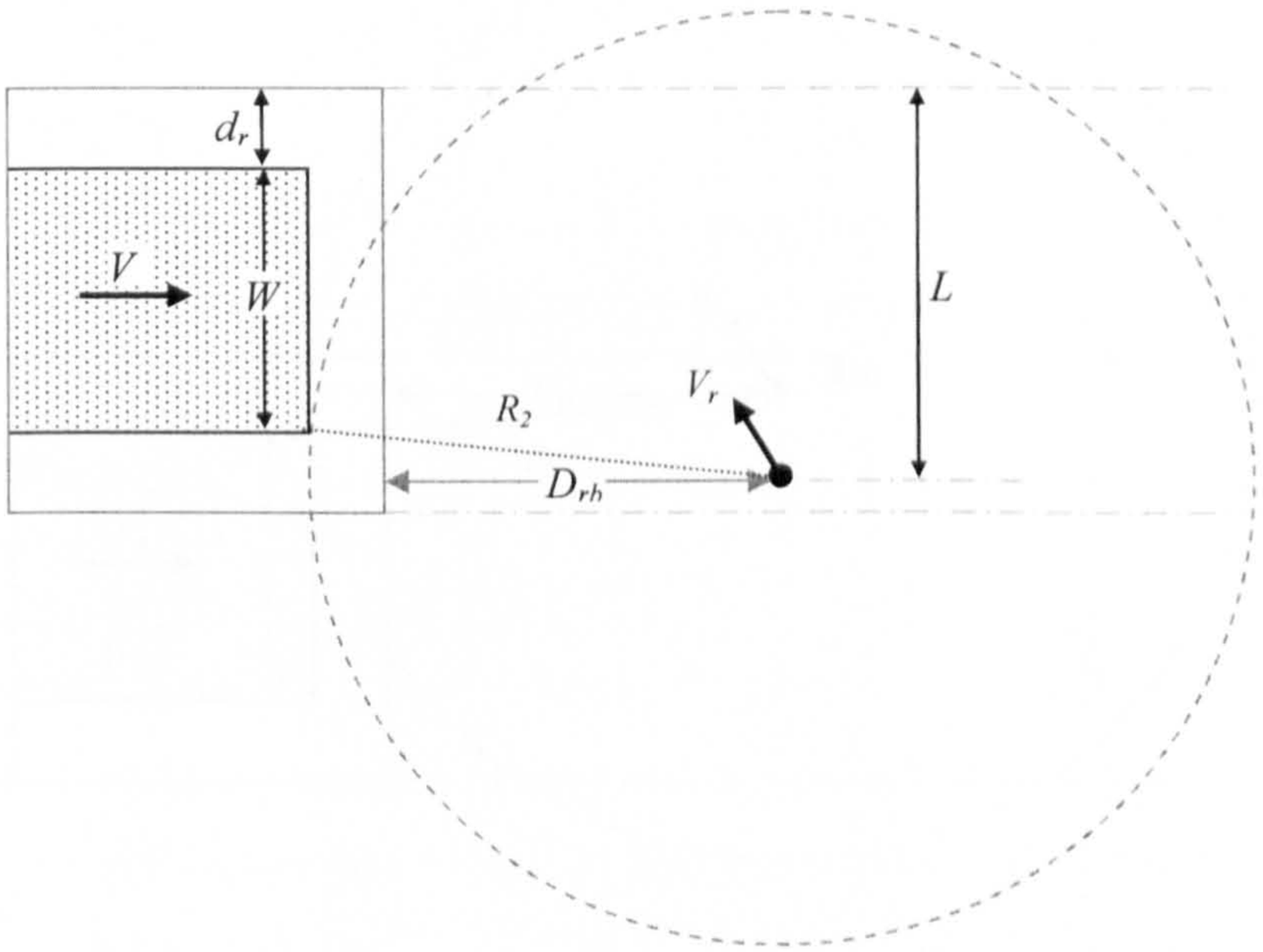


(a) turning left

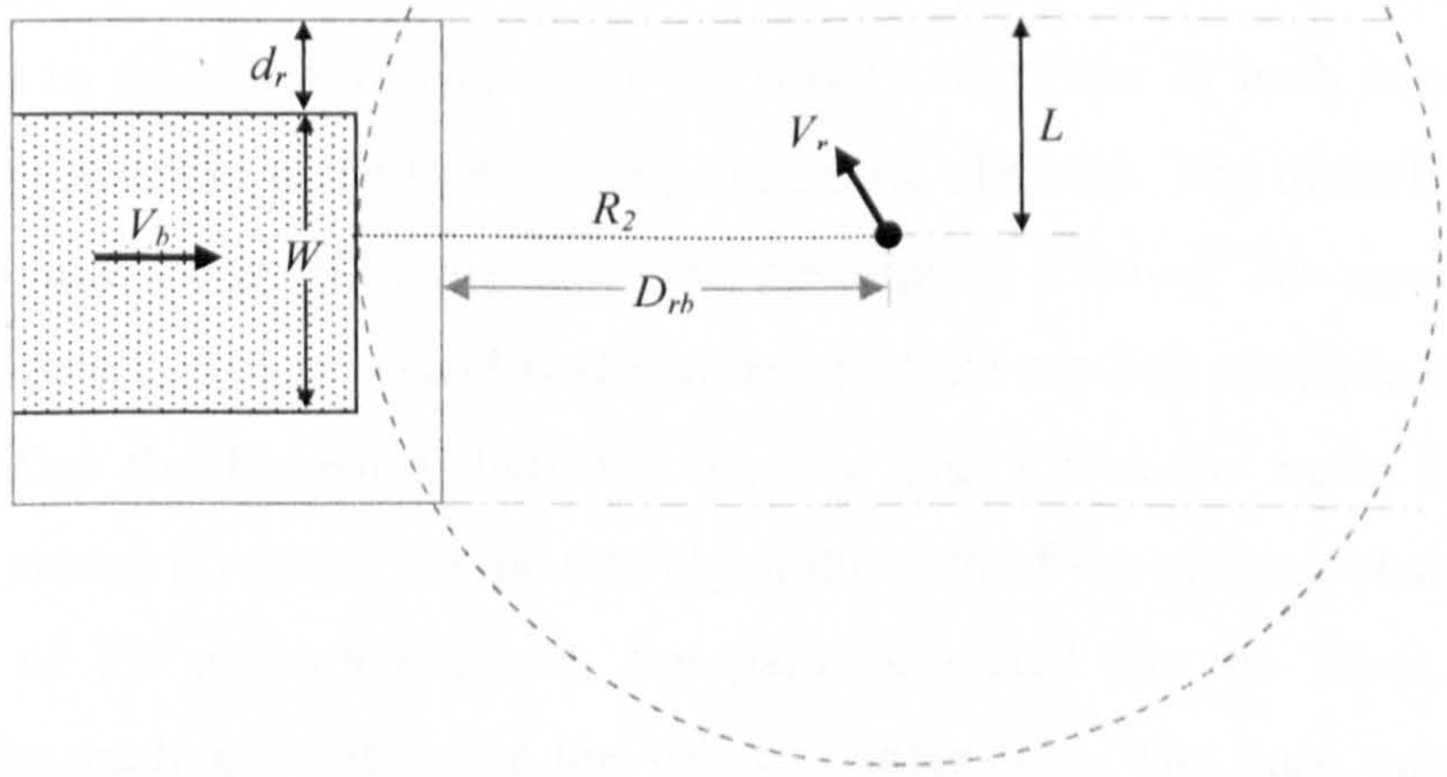


(b) turning right

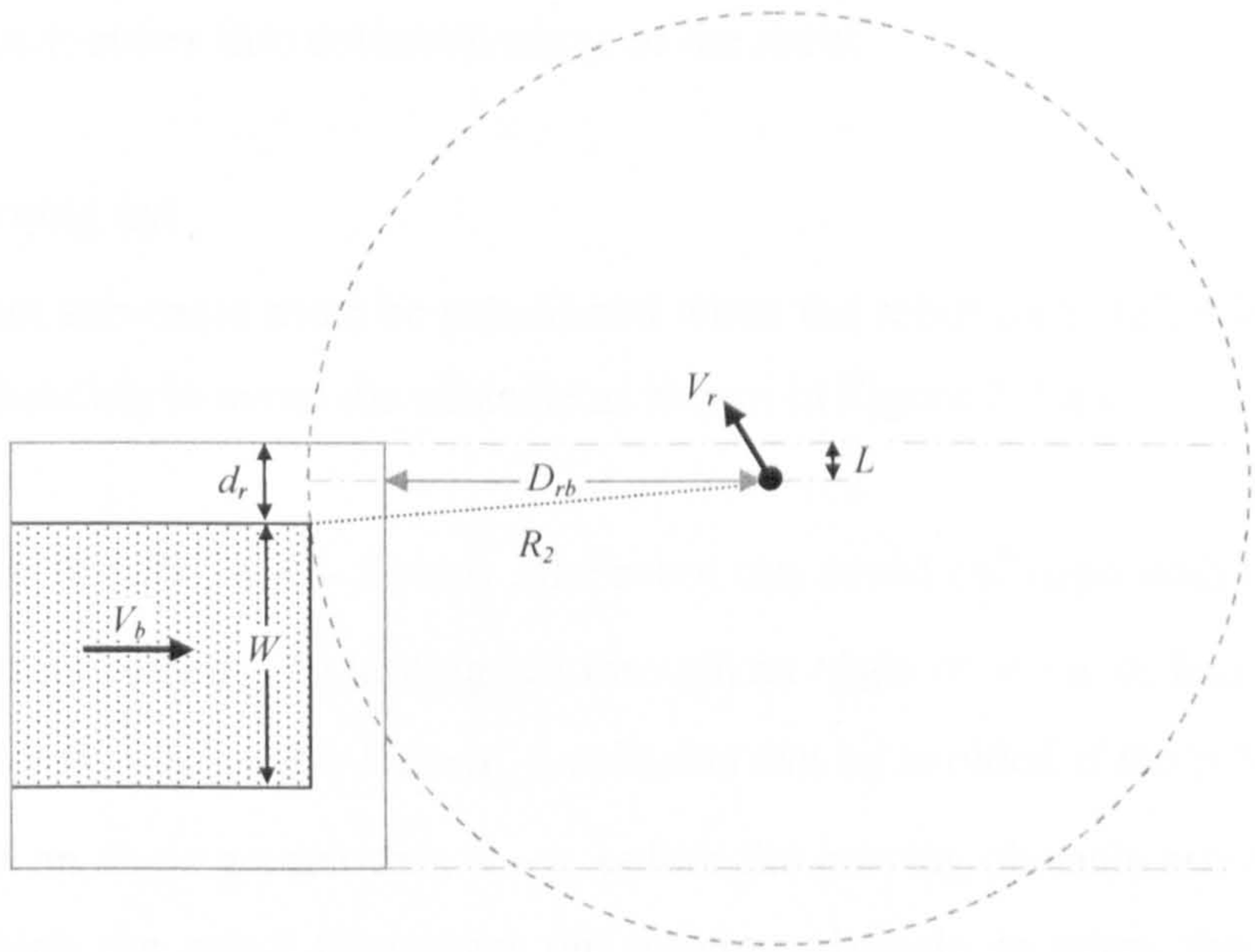
Figure 7.3 Example of the robot moving with a component of its velocity in the direction opposite to the moving direction of the obstacle, when the component of the velocity of the robot is projected along the longitudinal dimension of the moving obstacle. (a) shows the robot turning left to avoid the collision and (b) the robot turning right.



(a) $L > W + d_r$



(b) $d_r \leq L \leq W + d_r$



(c) $L < d_r$

Figure 7.4 The determination of D_{rb} when
(a) $L > W + d_r$, (b) $d_r \le L \le W + d_r$, and (c) $L < d_r$

The following sections derive the conditions for obstacle avoidance for two separate cases. Firstly, in which the robot turns left and secondly in which the robot turns right with respect to the current heading of the robot and where in both cases the robot must identify possible paths to avoid the oncoming obstacle. The underlying concept is given first and the detailed mathematical description follows. The condition where the robot is able to follow a modified path to meet the moving obstacle at the escape point, specifies the boundary between feasible and infeasible cases. If the robot crosses the escape point toward the outside of the path of the moving obstacle prior to the arrival of the moving obstacle, the path generated for the robot is feasible, otherwise the path generated for the robot is infeasible. The case employed here bounds a range of velocities for the moving obstacle within which a collision-free path can always be generated. Furthermore, it is assumed that a new path can be generated for the robot to avoid the moving obstacle immediately when the moving obstacle enters into the detection range. In the following discussions, the time T indicates the time that the robot requires to move to the escape point from the location where the moving obstacle is first sensed in the detection range, and the distance D,

as defined above, is that the obstacle moves to reach the escape point from the location once it enters into detection range of the robot.

A. Robot turning left

Three distinct sub-cases must be considered when the robot turns left with respect to the current heading to avoid the obstacle as shown in Figure 7.3(a).

(i) When $L > W + 2d_r + R + R\cos\alpha$, the robot can avoid collision with the obstacle, regardless of its velocity, by turning left through an angle of $\pi - \alpha$ or less.

(ii) When $L = W + 2d_r + R + R\cos\alpha$, a collision can be avoided if the robot is able to turn through an angle greater than $\pi - \alpha$ before the moving obstacle arrives. The first point at which the robot may meet the moving obstacle is when the robot turns through an angle β_{ol}, equal to $\pi - \alpha$, and where the travel time T and the travel distance D to the escape point are expressed as follows:

$$T = \frac{\pi - \alpha}{\omega_r} \qquad \text{Equation 7.5}$$

$$D = D_{rb} - R\sin\alpha \qquad \text{Equation 7.6}$$

(iii) When $L < W + 2d_r + R + R\cos\alpha$, the robot turns by an angle of β_{ol} with respect to the current heading of the robot and then moves forward following the heading after turning β_{ol} before meeting the moving obstacle at escape point. Here, the travel time and point of closest approach are

$$T = \frac{\beta_{ol}}{\omega_r} + \frac{R\cos\alpha - R\cos(\alpha + \beta_{ol}) - d}{\sin(\alpha + \beta) \cdot V_r} \qquad \beta_{ol} \in (\pi - \alpha,\ 2\pi - \alpha) \qquad \text{Equation 7.7}$$

and

$$D = D_{rb} - R\sin\alpha + R\sin(\alpha + \beta_{ol}) + [d - R\cos\alpha + R\cos(\alpha + \beta_{ol})]\cot(\alpha + \beta_{ol})$$

Equation 7.8

where $d = L - W - 2d_r$.

For cases (ii) and (iii), the maximum velocity of the moving obstacle with respect to different escape points can be deduced as follows.

Firstly, the maximisation of Equation 7.9 is carried out with respect to β_{ol} (note that for case (ii) only a single value for β_{ol} is considered.)

$$V_{bl} = \frac{D}{T} \qquad \text{Equation 7.9}$$

$$V_{bl\max} = \max V_{bl}(\beta_{ol}) \qquad \beta_{ol} \in (\pi - \alpha,\ 2\pi - \alpha) \qquad \text{Equation 7.10}$$

In order to ensure obstacle avoidance for any value of L, the minimum of Equation 7.10 needs to be determined as follows

$$V_{l\max} = \min V_{bl\max}(L) \qquad L \in (0,\ W + 2d_r + R + R\cos\alpha] \qquad \text{Equation 7.11}$$

B. Robot turning right

Three separate cases need to be considered when turning right with respect to the current heading of the robot by an angle of β_{or}.

(i) When $L > W + 2d_r + R - R\cos\alpha$, the avoidance strategy is independent of the obstacle velocity.

(ii) When $L = W + 2d_r + R - R\cos\alpha$, the robot needs to turn an angle greater than α before the moving obstacle arrives at the escape point. Again, only the first point at which the robot could possibly meet the moving obstacle is considered, as this maximises the distance that the moving obstacle traverses prior to any contact with the robot within the minimum travel time. The time T and distance D after the robot has turned an angle of α can be written as follows

$$T = \frac{\alpha}{\omega_r} \qquad \text{Equation 7.12}$$

$$D = D_{rb} + R\sin\alpha \qquad \text{Equation 7.13}$$

(iii) When $L < W + 2d_r + R - R\cos\alpha$, avoidance can be realised by first turning by an angle of β_{or} with respect to the current heading and then moving forward following the heading after turning by an angle of β_{or}. Equation 7.14 and 7.15 express the time T and the distance D respectively

$$T = \frac{\beta_{or}}{\omega_r} + \frac{L + R\cos\alpha - R\cos(\alpha - \beta_{or})}{\sin(\alpha - \beta_{or}) \cdot V_r} \qquad \beta_{or} \in (0,\ \alpha) \qquad \text{Equation 7.14}$$

$$D = D_{rb} + R\sin\alpha - R\sin(\alpha - \beta_{or}) + [L + R\cos\alpha - R\cos(\alpha - \beta_{or})]\cot(\alpha - \beta_{or})$$

Equation 7.15

Thus, the maximum velocity of the moving obstacle when it meets the robot at different escape points can be deduced by following manipulations.

Firstly, Equation 7.16 is maximised with respect to β_{or} (note that only a single value of β_{or} is considered for case (ii)).

$$V_{br} = \frac{D}{T} \qquad \text{Equation 7.16}$$

$$V_{br\max} = \max V_{br}(\beta_{or}) \qquad \beta_{or} \in (0, \alpha) \qquad \text{Equation 7.17}$$

Secondly, Equation 7.17 is minimised with respect to L, so that obstacle avoidance can be realised for any value of L.

$$V_{r\max} = \min V_{br\max}(L) \qquad L \in (0,\ W + 2d_r + R + R\cos\alpha] \qquad \text{Equation 7.18}$$

Finally, the maximum velocity for the moving obstacle is the maximum of those obtained for turning left and right, as found in Equations 7.11 and 7.18, namely

$$V_{b\max} = \max(V_{l\max}, V_{r\max}) \qquad \text{Equation 7.19}$$

Therefore, $[0, V_{b\max})$ is the range of the velocity that the moving obstacles can select and this range ensures that there is at least a path that can be generated for the robot to avoid the moving obstacle.

According to the above analysis, the detection range bounds the maximum width of the moving obstacle that can be viewed before taking actions. On the other hand, the moving velocity of the obstacle is also constrained for given detection range to avoid inevitable collisions. Although a larger detection range provides more time for the robot to avoid the moving obstacle, in such a case the maximum lateral dimension of

the moving obstacle to ensure a collision-free path would be reduced. The introduction of the fixed detection range in this chapter simplifies the analysis and provides a single value for the maximum width.

7.4.2 Mobile robot dynamic avoidance behaviour

If the current path becomes infeasible due a moving obstacle, the robot must determine whether it should turn left or right with respect to its current heading to avoid the collision. Should only one direction be feasible, this will be taken. However, the situation becomes more complex if both are feasible and a choice must be made based upon the characteristics of the possible path. The robot could select the turning direction according to the relative lengths of the alternative paths to the goal, the time consumed in avoiding collision threats, or the turning angle. Here, the selection of the route to be taken is based on the length of the revised path to the goal, where the robot chooses the shorter path and the positions of other static obstacles within the detection range. Figure 7.5 illustrates the method by which the robot chooses the avoidance path.

```
if the current path is infeasible
  assess the possibilities of turning left and right
  if turning left (or right) is feasible and the other is infeasible
    turn left (or right)
  else if turning left and right are all possible
    check if any of them is blocked by static obstacles
    if turning left (or right) is blocked and the other is not
      turn right (or left)
    else
      select shorter path
    end if
  end if
end if
```

Figure 7.5 Algorithm for moving obstacle avoidance.

A. Feasibility of the current path

When determining the feasibility of a path (that is, to ensure no collision with the obstacle will occur), the moving obstacle's dimensions are enlarged by the value of the robot diameter, allowing the robot to be treated as a point. The algorithm first determines the crossing point between the current path of the robot (with the robot treated as a point and the path simply being a line) and the path edge of the moving

obstacle which the robot is currently approaching. This crossing point, P, is illustrated in Figure 7.3(b) with the straight broken line shown from the robot to the goal representing the current path. The time from the current position of the robot to P and the time from the current position of the obstacle to P can then be estimated in conjunction with the velocity of both the robot and the obstacle. If the robot takes a longer time to reach P than does the moving obstacle, the current path is infeasible. In such a case, the robot will attempt to avoid possible collision with the moving obstacle by turning left or right with respect to its current heading.

B. Turning left or right

Figure 7.3 shows the situation where the robot moves with a component of the velocity of the robot in the direction opposite to the moving direction of the obstacle, when the component of the velocity of the robot is projected along the longitudinal dimension of the obstacle. For such situation ($\pi/2 < \alpha \leq 3\pi/2$), a set Θ of angles β for turning left (substituting β with β_{ol}) or right (substituting β with β_{or}) can be deduced from the inequality in Equation 7.20. The inequality indicates that the distance at which the obstacle moving at V_b over time T is less than D, the distance to the escape point from the position when the moving obstacle first enters the detection range. This ensures that the robot escapes from the path of the moving obstacle before its arrival. The set of angles is given by

$$\Theta = \{\beta \mid D > V_b T, 0 < \beta < 2\pi\} \qquad \text{Equation 7.20}$$

where T and D can be calculated from Equations 7.4 and 7.5 for turning left, when $L < W + 2d_r + R + R\cos\alpha$, or Equations 7.11 and 7.12 for turning right, when $L < W + 2d_r + R - R\cos\alpha$. If $\Theta \in \{\}$, then turning left or right is infeasible, otherwise, the set Θ contains all feasible turning angles and the robot can select one according to the optimisation goal. For the situation where $L = W + 2d_r + R + R\cos\alpha$ for turning left or $L = W + 2d_r + R - R\cos\alpha$ for turning right, the robot should turn by an angle greater than $\pi - \alpha$ for left or α for right. When $L > W + 2d_r + R + R\cos\alpha$, the moving obstacles are extended along its longitudinal dimension, so that the robot can negotiate the extended obstacle under reactive control. Such manipulation enables

the moving obstacle to be treated as being stationary with respect to its lateral dimension. Furthermore, such a distance between the robot and the extended obstacle (note this distance is larger than the minimum distance that the robot is required to turn) ensures that the robot is able to avoid the moving obstacle. In the experiments presented in this chapter, the extension length was set to be sufficiently long to prevent the robot from crossing the moving obstacle from the front of the obstacle.

Figure 7.6 shows examples of the robot turning (a) left and (b) right to avoid a potential collision with an obstacle moving in the same direction as the component of the velocity of the robot which is projected on to the longitudinal dimension of the obstacle (i.e., $0 \le \alpha \le 2/\pi$ or $3/2\pi < \alpha < 2\pi$). The set Θ of of angles β that the robot is able to turn left (substituting β with β_{sl}) or right (substituting β with β_{sr}) without collision can be derived from Equation 7.20, in which T and D can be deduced from each of the two different turning directions.

For turning left, the travel time T is

$$T = \frac{\beta_{sl}}{\omega_r} + \frac{L - R\cos\alpha + R\cos(\alpha + \beta_{sl})}{V_r \sin(\alpha + \beta_{sl})}$$

$$L \in (0,\ R\cos\alpha + R) \text{ and } \beta_{sl} \in \left(0, \cos^{-1}\frac{R\cos\alpha - L}{R} - \alpha\right] \qquad \text{Equation 7.21}$$

and D is

$$D = D_{rb} - R\sin\alpha + R\sin(\alpha + \beta_{sl}) + \left[L - R\cos\alpha + R\cos(\alpha + \beta_{sl})\right]\cot(\alpha + \beta_{sl})$$

Equation 7.22

When $R + R\cos\alpha \le L < W + 2d_r + R + R\cos\alpha$, Equations 7.21 and 7.22 remain applicable for T and D with $\beta_{sl} \in (0, \pi - \alpha)$. The robot should turn an angle greater than $\pi - \alpha$ to avoid the moving obstacle, for the case when $L = W + 2d_r + R + R\cos\alpha$. When $L > W + 2d_r + R + R\cos\alpha$, the robot can, under reactive control, avoid the moving obstacle that has been extended along its longitudinal dimension.

The values of T and D when turning right, are as follows. When $L > W + 2d_r + R - R\cos\alpha$, the robot is able to turn through its minimum radius to avoid the oncoming obstacle, and this can be implemented by extending the bounding rectangle of the moving obstacle along its longitudinal dimension and then allowing the robot to operate under reactive control. For the case where $L = W + 2d_r + R - R\cos\alpha$, the robot must turn by an angle greater than α to avoid collision. For the case where $R - R\cos\alpha < L < W + 2d_r + R - R\cos\alpha$, T and D are determined separately for two ranges of turning angles. When the turning angle is less than α, T and D can be determined from Equations 7.23 and 7.24,

$$T = \frac{\beta_{sr}}{\omega_r} + \frac{L + R\cos\alpha - R\cos(\alpha - \beta_{sr})}{V_r \sin(\alpha - \beta_{sr})} \qquad \beta_{sr} \in (0, \alpha)$$

Equation 7.23

$$D = D_{rb} + R\sin\alpha - R\sin(\alpha - \beta_{sr}) + [L + R\cos\alpha - R\cos(\alpha - \beta_{sr})]\cot(\alpha - \beta_{sr})$$

Equation 7.24

otherwise ($\beta_{sr} \in \left(\alpha, \cos^{-1}\frac{R\cos\alpha + d}{R} + \alpha\right]$), T and D are given by

$$T = \frac{\beta_{sr}}{\omega_r} + \frac{d + R\cos\alpha - R\cos(\alpha - \beta_{sr})}{V_r \sin(\alpha - \beta_{sr})} \qquad \beta_{sr} \in \left(\alpha, \cos^{-1}\frac{R\cos\alpha + d}{R} + \alpha\right]$$

Equation 7.25

and

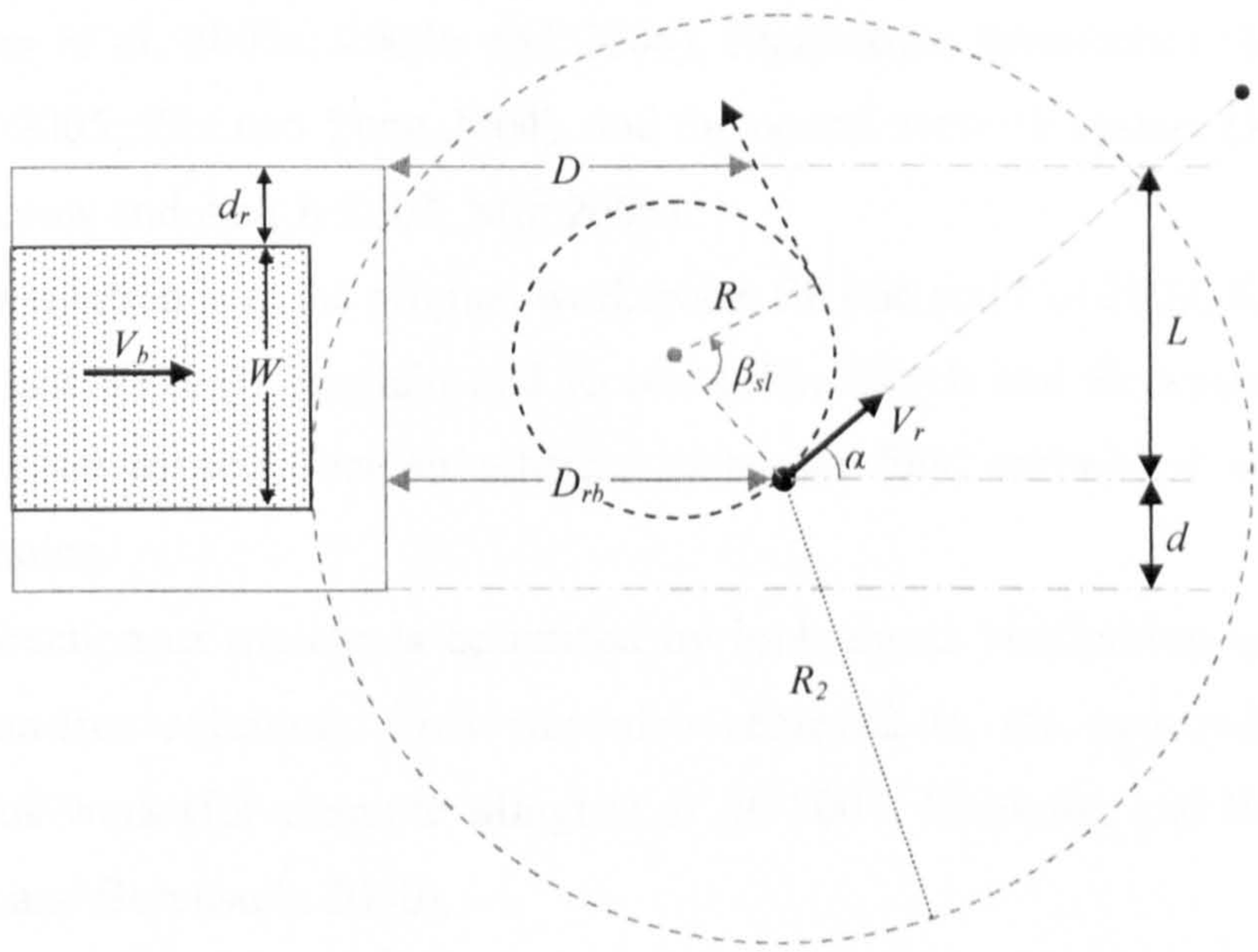
$$D = D_{rb} + R\sin\alpha - R\sin(\alpha - \beta_{sr}) + [d + R\cos\alpha - R\cos(\alpha - \beta_{sr})]\cot(\alpha - \beta_{sr})$$

Equation 7.26

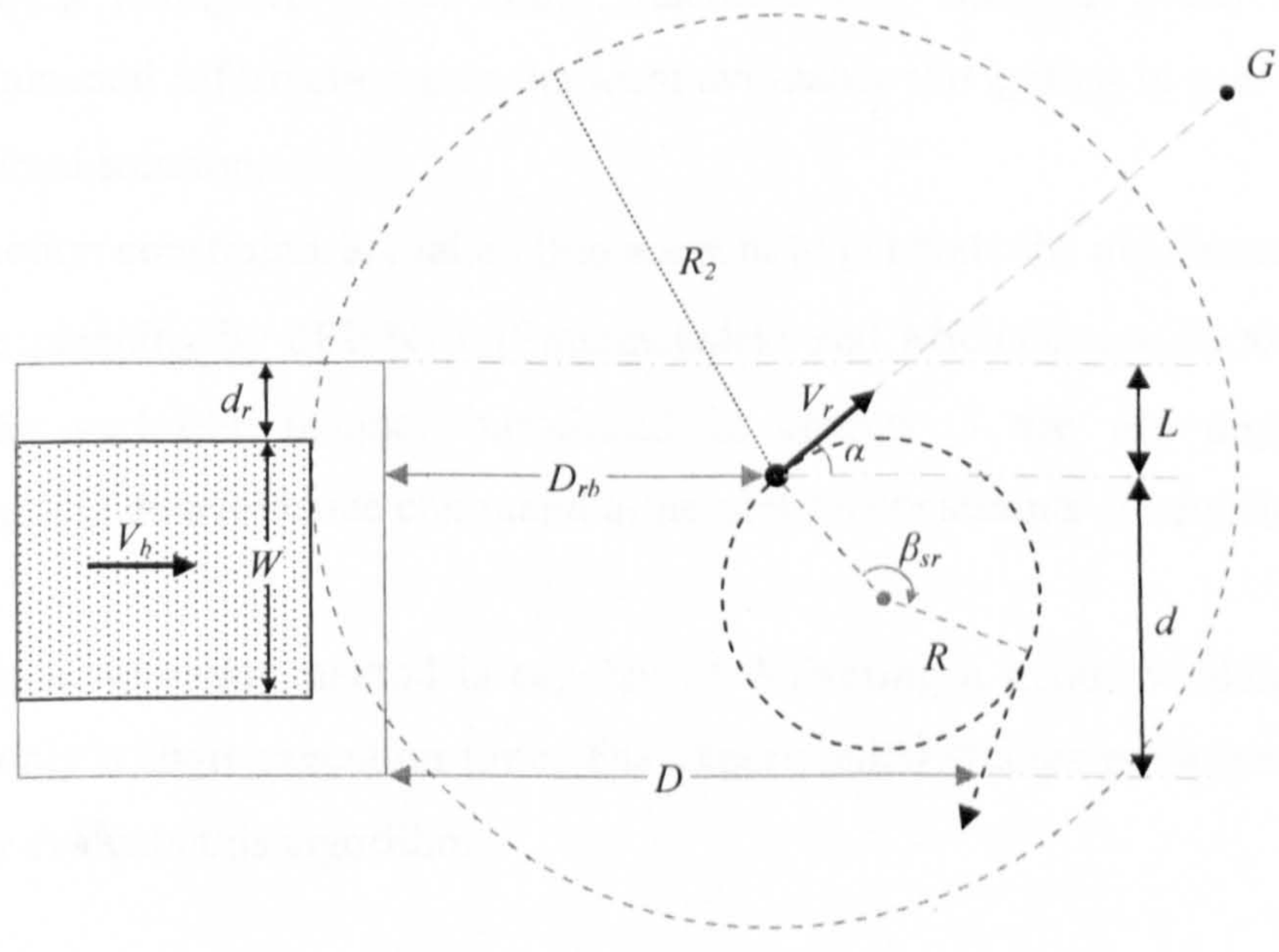
where $d = L - W - 2d_r$.

Note that a turning angle β_{sr} equal to α is considered to be infeasible as the robot following such path is unable to escape from the moving obstacle.

When $L \leq R - R\cos\alpha$, T and D can be expressed by Equations 7.23 and 7.24 with $\beta_{sr} \in \left(0, \alpha - \cos^{-1}\frac{L + R\cos\alpha}{R}\right]$.



(a) turning left



(b) turning right

Figure 7.6 An example where the robot moves with a component of its velocity in the same direction as the moving obstacle, when the component of the velocity of the robot is projected along the longitudinal dimension of the moving obstacle. (a) shows the robot turning left to avoid the collision and (b) turning right.

The dynamic waypoint method delivers the following benefits compared with existing approaches.

1. No training process is required. In contrast, an appropriate training process has to be carried out to ensure proper avoidance behaviour in CBR (Kira and Arkin 2004; Urdiales *et al.* 2003a, 2003b and 2006), fuzzy-logic approaches (Malhotra and Sarkar 2005; Zhu and Yang 2004), and the neural network system (Kubota 2004; Low, Leow and Ang Jr 2003; Min 2005).
2. Implementation is in the original workspace. PF (Ge and Cui 2002; Kurihara *et al.* 2005) and VFH (Borenstein and Koren 1991; Ulrich and Borenstein 1998 and 2000) need to construct an artificial potential field embedded with temporal information.
3. The direction of motion is optimised by look-ahead verification using a set of collision-free solutions. This was also achieved in the approaches in some previous work (for example Minguez *et al.* 2001; Stachniss and Burgard 2002; Ulrich and Borenstein 2000).
4. No prior environmental information is required. A number of existing avoidance techniques (Minguez *et al.* 2001; Stachniss and Burgard 2002) incorporate environmental information into the local avoidance navigation in order to deliver an optimal solution.
5. The motion constraints are taken into account to generate the avoidance command. Online planning by 9EP/N++ (Smierzchalski and Michalewicz 2000 and 2006) and the vertex++ planner introduced in chapter 5 are not appropriate in determining an avoidance command as no motion constraints are considered.

The dynamic waypoint method is capable of delivering a good avoidance solution and with only a short execution time. The experimental studies presented in section 7.6 further evaluate this algorithm.

7.5 Deliberative module

The set of waypoints, determined by the waypoint detector, forms a compact description of the environment and greatly reduces the memory capacity requirement.

The strategy for the selection and recording of waypoints is same as that used in the navigation system introduced in the previous chapter. However, the selection of waypoints is constrained to those generated from the presence of static obstacles only. Following the initial identification of a waypoint, two possible new headings are available to the robot to circumnavigate the static obstacle indicated by the waypoint. As only one of these two can be investigated, a waypoint is marked as unexplored until the second heading has been taken (perhaps at a later stage as part of a separate navigation task).

Three functions are undertaken as waypoints are acquired, namely localisation, exploration and planning. These three functions are directly inherited from the previous navigation system with an extension made for the exploration function. Exploration of the segments between pairs of waypoints is performed in a statistical manner (further explanation is provided in section 7.5.1) in order to provide estimates of likely future travel times. This permits complete paths from start to goal positions to be generated using only the waypoints and the mean travel times for each segment. Further explanations of the additional functions can be found in the previous chapter.

7.5.1 Statistical exploration

A technique, termed as 'statistical exploration', was developed in an attempt to summarise the dynamic characteristics of the environment. This is realised by performing navigation between a pair of waypoints for a number of times, and the travel time, taken for planning the segment connecting the pair of waypoints, is the mean value of the travel times recorded for the number of navigations between the pair of waypoints. A learning process can be employed to achieve the statistical navigation between a pair of waypoints, when the robot is free from any assigned navigation task. Alternatively, the dynamic characteristics between a pair of waypoints may be estimated as a result of previously navigations between the two waypoints. In this work, the first method is used for statistical navigation between a pair of waypoints, where movement between all pairs of waypoints been followed for the same number of times, as specified by user. This technique may bring benefit when planning a suitable path in dynamic environments, particularly where a large

number of moving obstacles are likely to be found between certain pairs of waypoints, but a few are likely found between other pairs of waypoints. Section 7.6 reports the experimental studies for this technique. In static environments, the experience gathered from previous travels can be used to improve performance in future navigation activities. If dynamic obstacles exhibit repetitive characteristics or their collective movements in the environment are non-uniform in nature, such features can be extracted into a statistical representation for use in future operations. However, it needs to be acknowledged that such a statistical representation will provide little or no useful information in environments where the movement of the dynamic obstacle is random.

7.6 Experimental results

To verify the operation of the enhanced navigation system, three sets of experiments were conducted using a set of simulated environments. The simulations used an autonomous mobile robotics toolbox (Brno University of Technology 2006) modified to render it compatible with version 7.0 of MATLAB (Mathworks 2006). A description on the simulator and the robot was given in section 6.5 where justification for such an experimental arrangement was also provided. The function of the first set of experiments was to test the performance of the navigation architecture in the presence of moving obstacles. The second set illustrates the effect of the path statistics generated upon the choice of path. The final set compares the proposed avoidance technique with the ND algorithm (Minguez *et al.* 2001; Minguez and Montano 2004; Minguez, Osuna and Montano 2004; Montesano, Minguez and Montano 2005) as used in the implementation of a hybrid architecture (Minguez and Montano 2005; Minguez, Montesano and Montano 2004). The ND algorithm was originally developed as a novel reactive approach for the obstacle avoidance problem (Minguez *et al.* 2001; Minguez and Montano 2004), but was extended for application to navigation in environments containing moving obstacles (Montesano, Minguez and Montano 2005). Although the authors discussed the comparative performance of their approach with other avoidance techniques and summarised the benefits of their technique (further details can be found in Minguez *et al.* 2001; Minguez and Montano

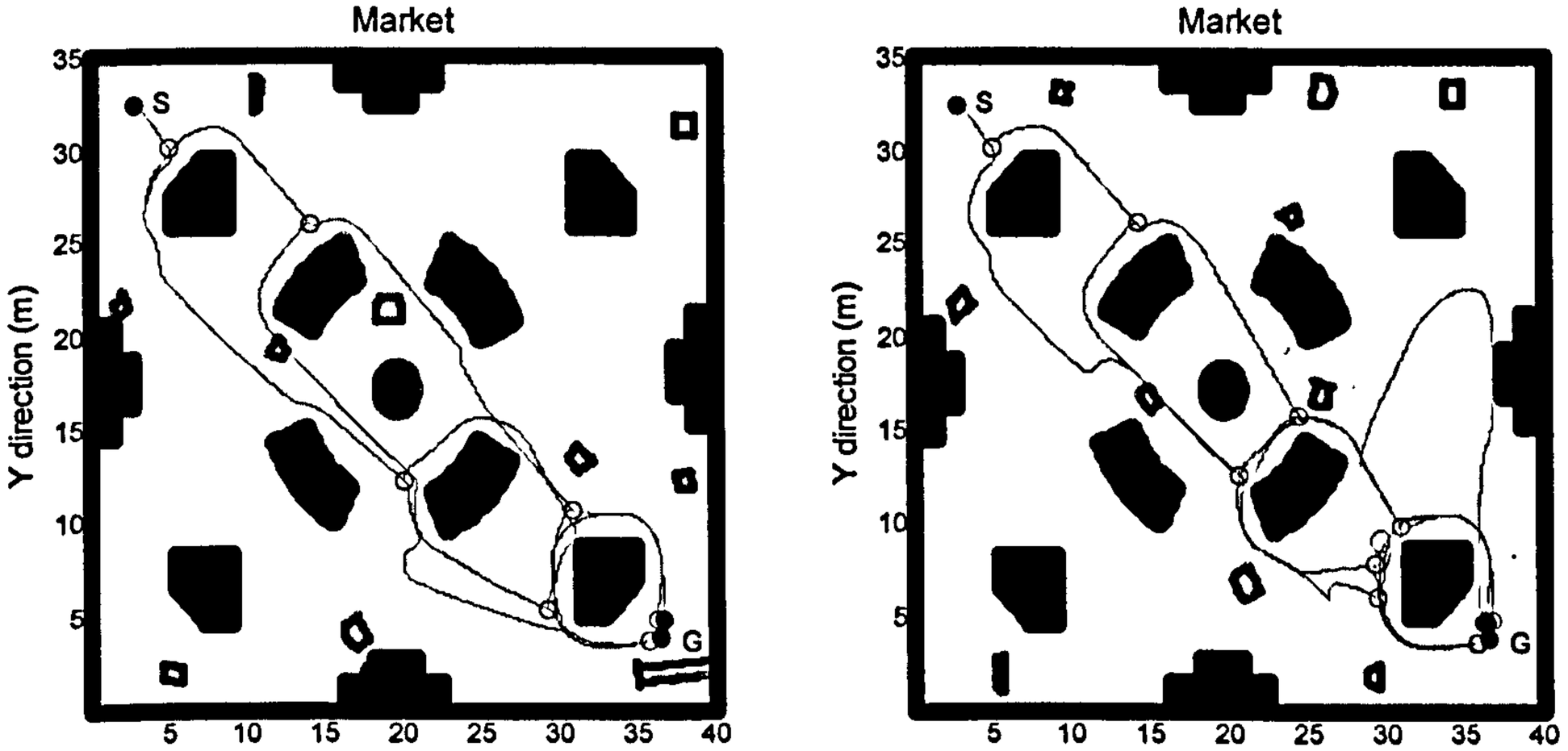
2004), no experimental comparison was carried out. A brief description of the ND algorithm is provided in section 7.6.3 in order to better understand the comparison results. ND has been chosen for comparison purposes as it operates in the original workspace and, as it computes avoidance commands according to a small set of pre-defined general patterns, ND does not require a training process. The other methods found in the literature are not suitable for comparison here, as they either transform the workspace to an alternative representation (DW, VO, VFH, PF) or require a training process (CBR, fuzzy logic, neural networks).

7.6.1 Dynamic avoidance

Figure 7.7 shows examples of trajectories followed by the robot in an environment in which the initial placement of the moving obstacles is random. For the test presented in Figure 7.7, a simple algorithm was designed to determine the trajectory of the moving obstacles, namely that the each obstacle continues to move forward following its current heading unless it detects another obstacle, whereupon it turns left. An additional constraint on the motion of the obstacles is that they maintain their current direction when within the detection range of the robot. The robot may fail to avoid collisions when turning left or right due to the presence of several obstacles approaching the robot from the front and two sides of the robot. In such circumstances, the moving obstacles were designed to be able to reverse direction in an attempt to avoid collision. Additional behaviour, termed 'waiting behaviour', was developed to deal with the situation where both dynamic and static obstacles simultaneously hinder the robot's progress. In this situation, the robot stops and waits for the path to become clear if it is not possible to avoid collision by turning left or right. This behaviour is apparent in the later experiments presented in Figure 7.10 where the robot waits for the junctions to become clear.

The first experiments were conducted with dynamic obstacles constrained to move at a constant speed, Figure 7.7(a), before this constraint was relaxed, in Figure 7.7(b). The range of the velocity for the moving obstacles was determined according to their widths, using the mathematical description in section 7.4. The constant speed for the moving obstacles used in the experiments was $0.52ms^{-1}$ which was slightly less than

the upper bound of the determined range (as the upper bound was not included in the determined range), while the robot travelled at a constant speed of $0.5ms^{-1}$ (this value was used in the training of reactive behaviours). A series of experiments was conducted in which the robot successfully avoided potential collisions with the moving obstacles by virtue of its high-level behaviours.



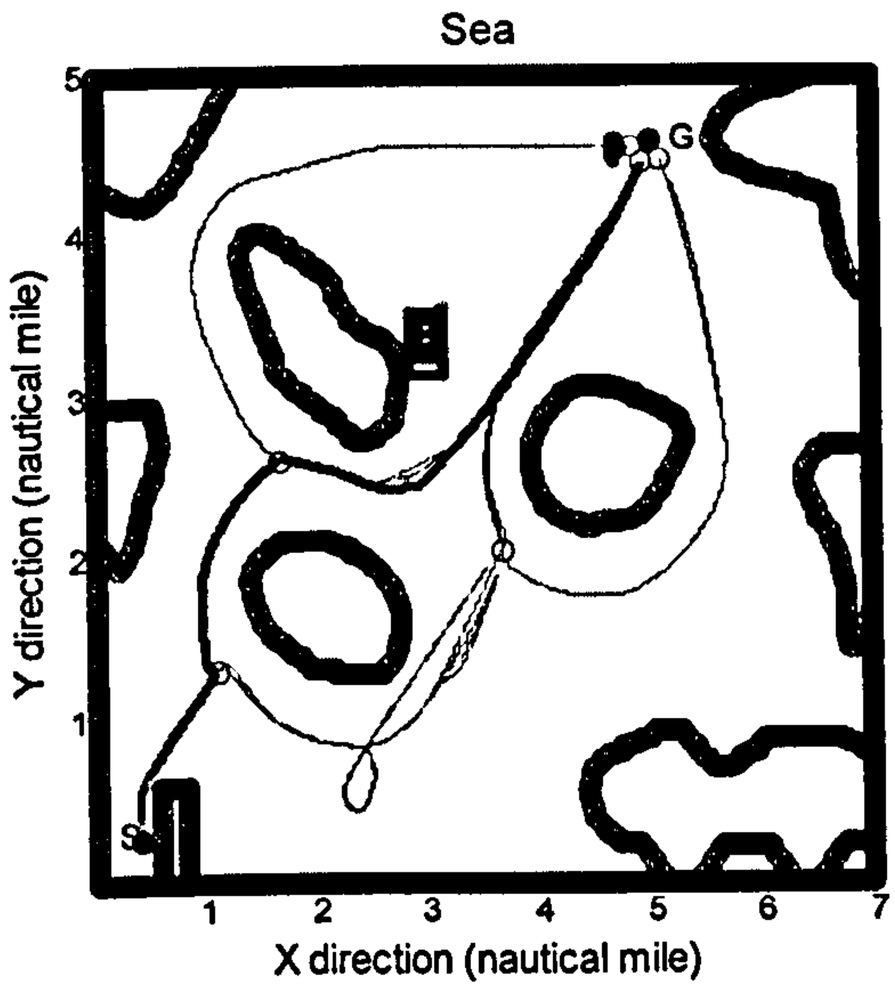
(a) obstacles moving at constant speed **(b)** obstacles moving at random speed

Figure 7.7 Trajectories followed by the navigation system. The solid objects represent the static obstacles and the moving obstacles are illustrated by non-solid objects. The solid markers with labels, 'S' and 'G' indicate the start and goal points respectively, and the circular markers represent the waypoints generated.

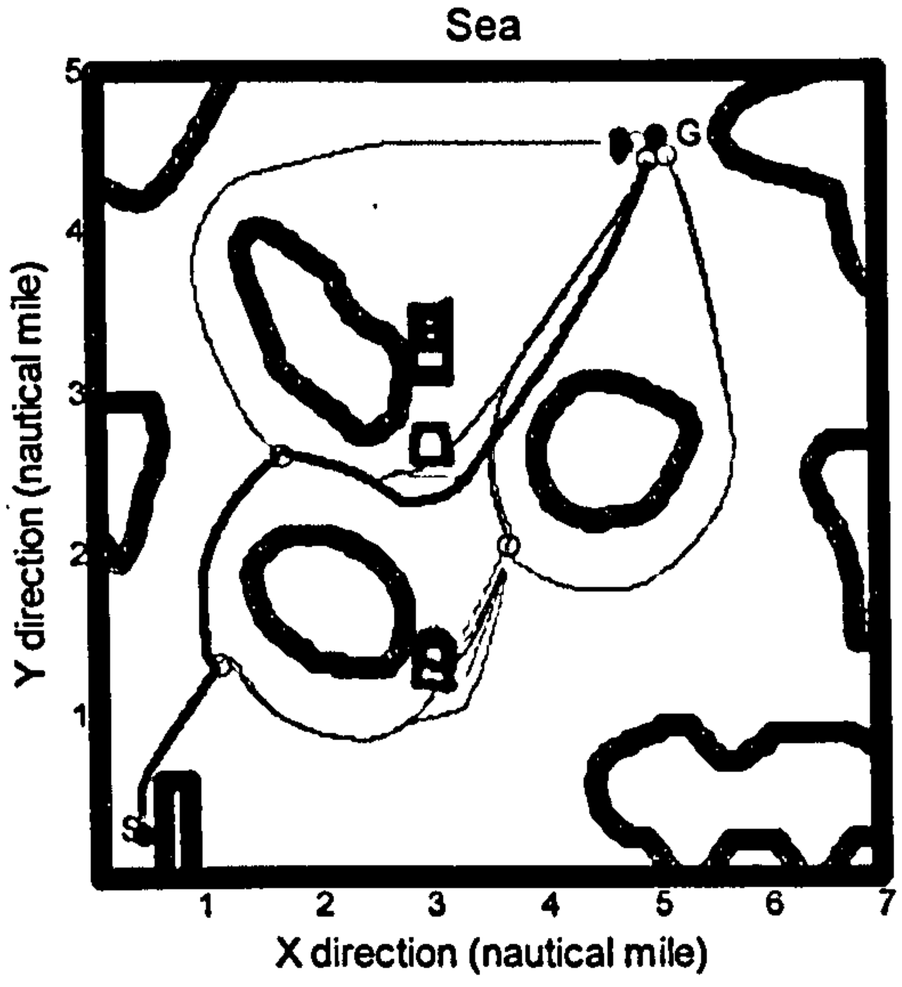
7.6.2 Statistical exploration

The 'sea' environment simulates a set of ships cruising at sea, with the ship under navigation moving goods repeatedly between two specific ports, while the channel between the ports is populated with static islands and a number of ships moving in a shipping line passing vertically through the environment. The moving ships travel at a constant speed of 8 knots and appear at an arrival rate of λ according to a Poisson distribution. The navigated ship also travels at 8 knots if the path is feasible, but is able to stop and wait if the current path becomes infeasible due to the possibility of collision. Figure 7.8 shows the navigation results for the sea environment when the moving obstacles follow the Poisson distribution with a range of arrival rates. In each case the navigated ship explored each path 10 times and the subsequent best paths

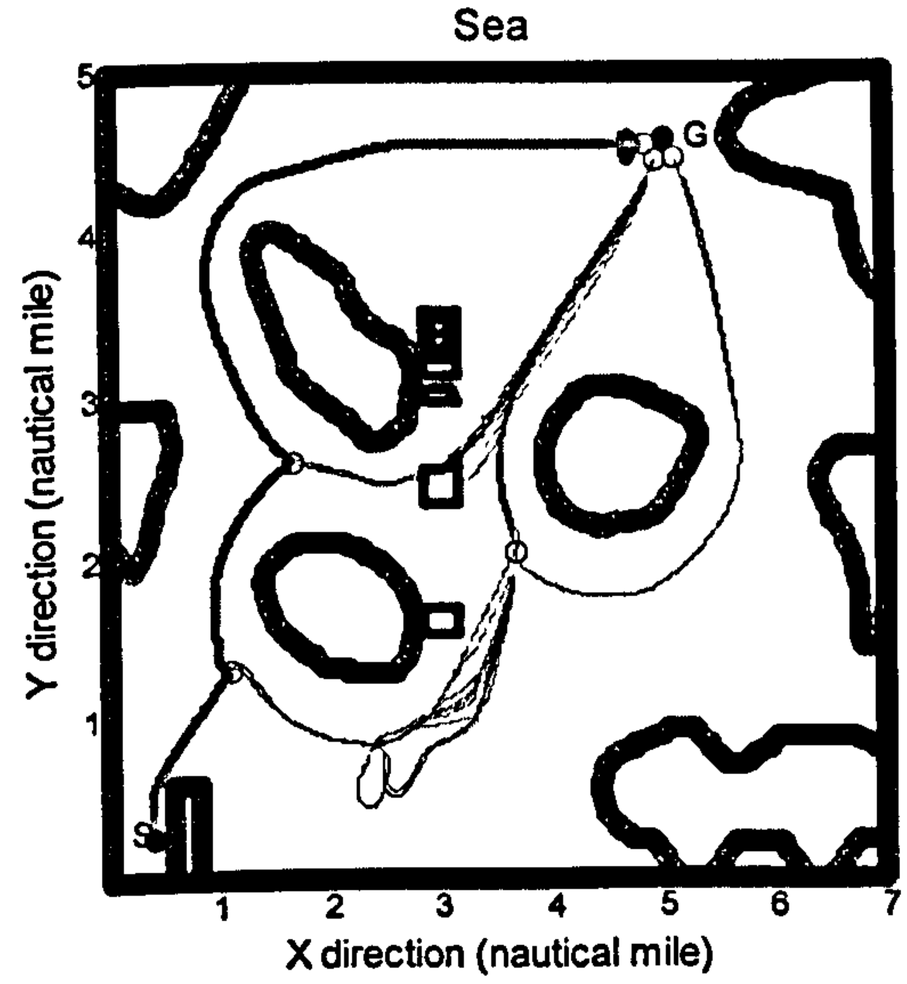
were then generated based on the mean travel time for each path segment. The aim of the planner was to minimise travel time, and the results are shown in Table 7.1.



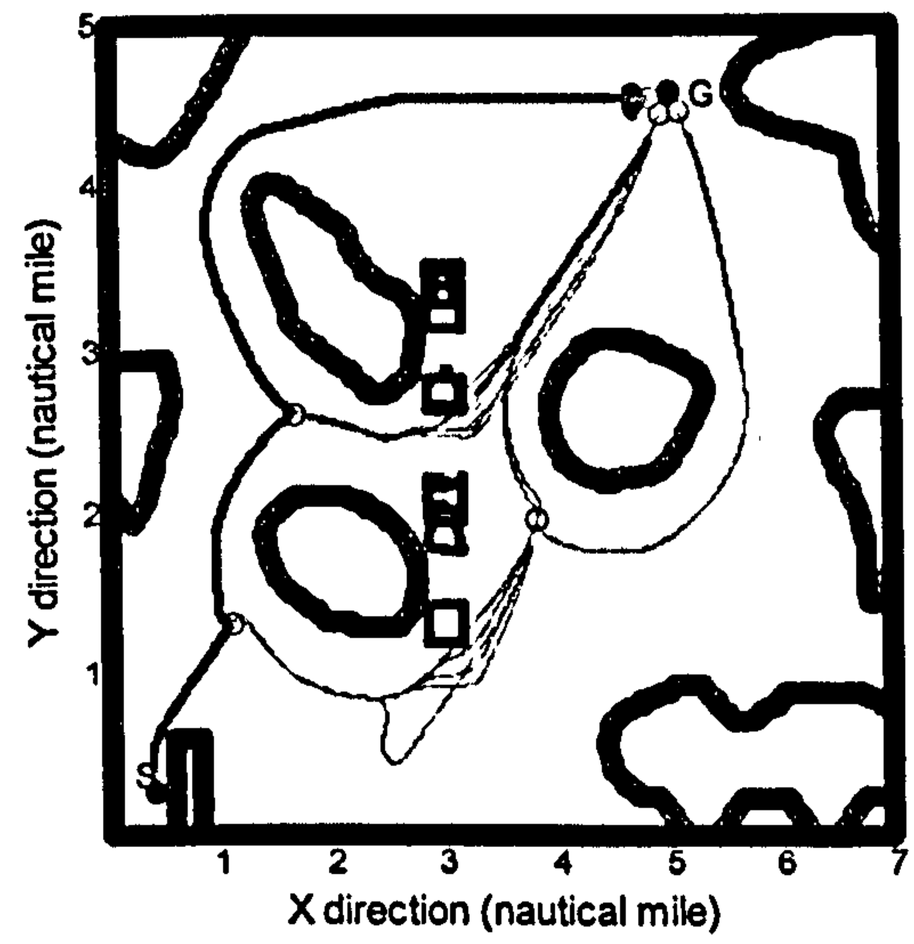
(a) $\lambda = 2$ ships/hour



(b) $\lambda = 4$ ships/hour



(c) $\lambda = 6$ ships/hour



(d) $\lambda = 8$ ships/hour

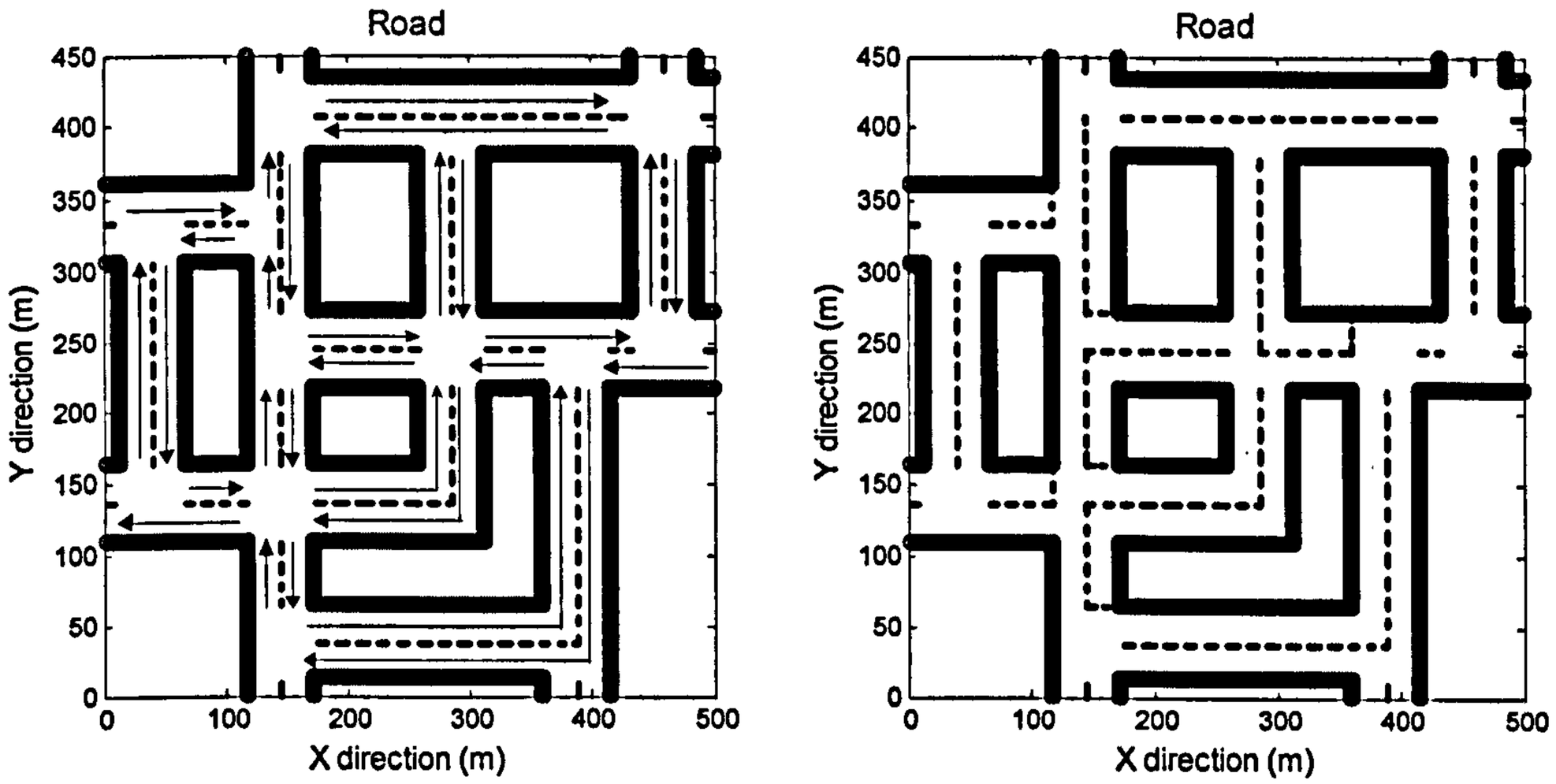
Figure 7.8 The generated paths for the sea environment for a range of arrival rates of moving obstacles. The markers, 'S' and 'G', indicate the start and goal points respectively and circular markers denote the generated waypoints. The thickened line is the path generated by the planner.

Table 7.1 Path costs for the best path in terms of mean travel time found from 10 traverses of each path segment.

λ (ships/hour)	2	4	6	8
travel time (hours)	0.91	0.94	1.06	1.06

From Figure 7.8 and Table 7.1, it can be seen that the navigated ship selects a more circuitous path to reduce the longer expected delay that is likely to results when the number of ships in the channel increases. Examples of such cases can be seen in Fig 7.8(c) and Figure 7.8(d) where a longer path around the uppermost island is taken in order to avoid the congested area in the centre of the channel. When few ships are likely to be present in the channel (Figure 7.8(a) and 7.8(b)), the navigated ship continues to travel through the more direct central area, on some occasions choosing to wait for the channel to clear. This strategy appears better able to keep travel time to a minimum in conditions of light traffic.

The 'road' environment in Figure 7.9 simulates a network of roads and mimics traffic levels found at different times of day. Figure 7.9(a) shows the environment which incorporates the rule that all moving obstacles must drive on the left, as illustrated by the arrows. However, as the robot was not equipped with knowledge of traffic regulations, a modified version of the environment was produced to ensure the robot's adherence to traffic rules, as shown in Figure 7.9(b) (note that the environment was modified for only the navigation task illustrated in Figure 7.10). At each sampling time, two separate sets of sensor readings are obtained; the set from Figure 7.9(a) producing information regarding the dynamic environment and that from Figure 7.9(b) reflecting the static one. In a practical implementation, the robot could be equipped with two sets of sensors; one for sensing road markings and the other to detect dynamic objects in the environment.

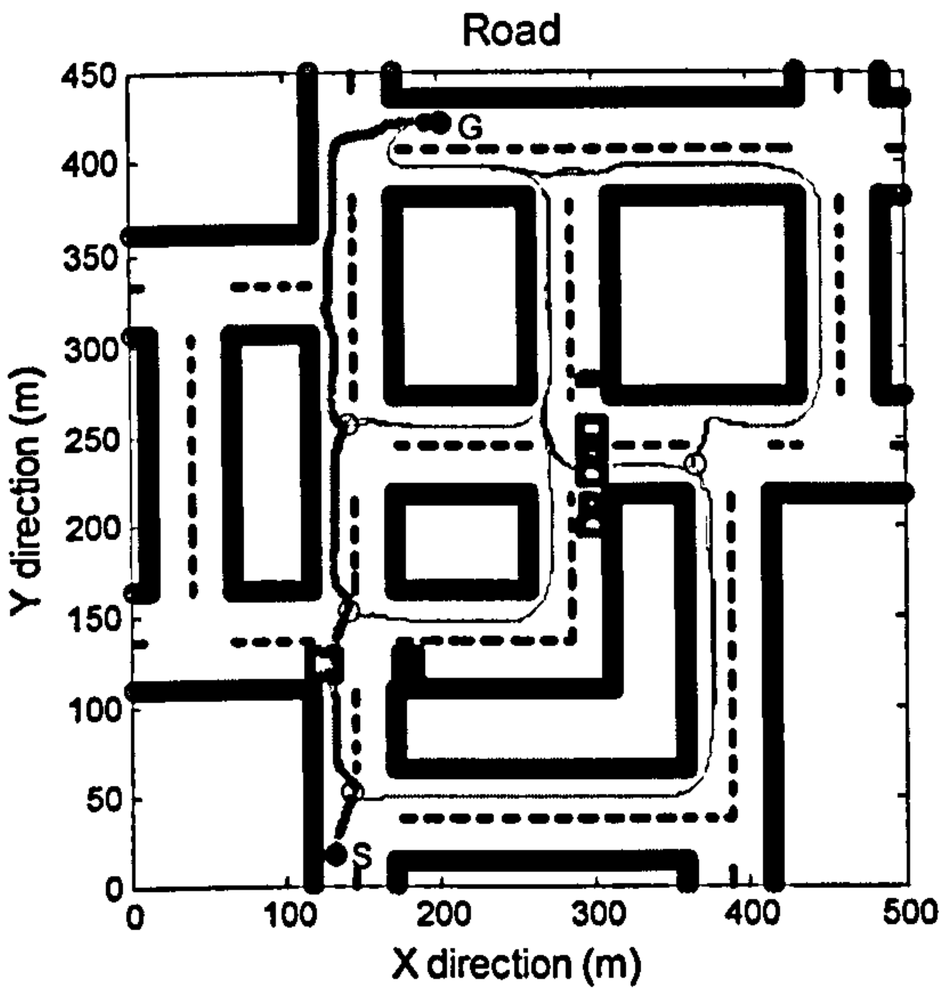


(a) original environment

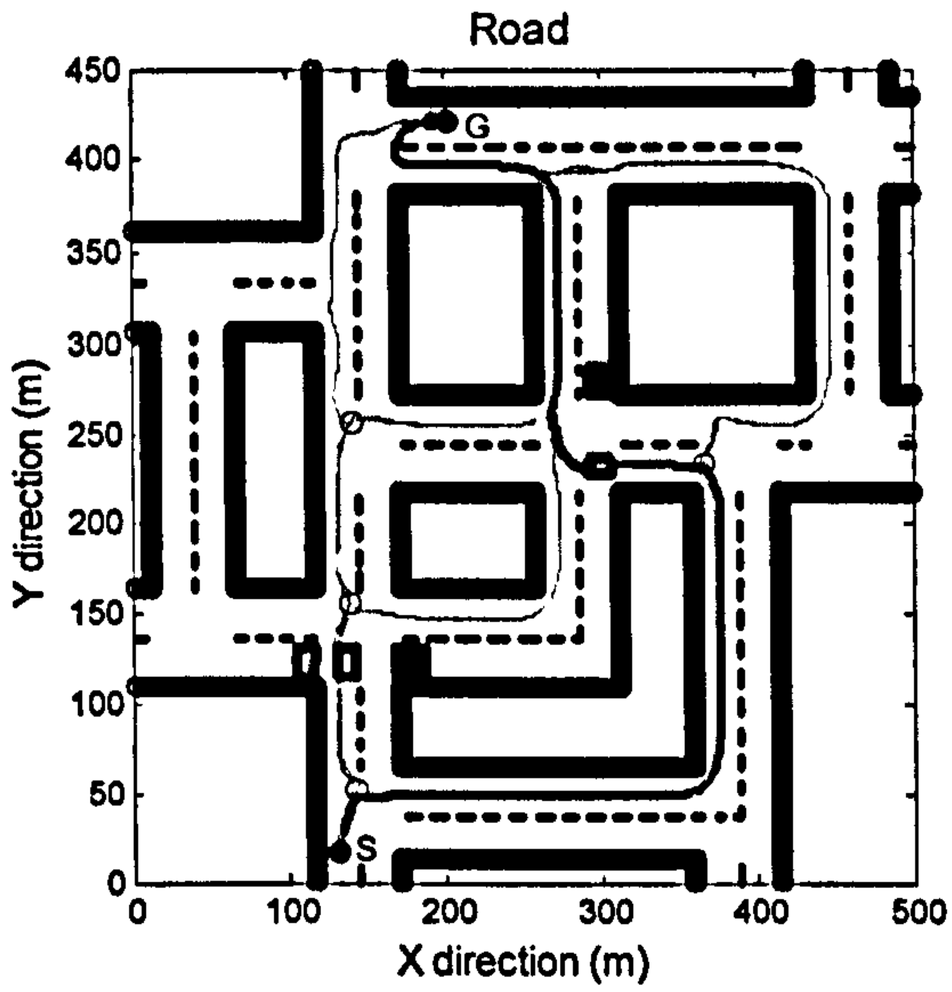
(b) modified environment to include traffic rules

Figure 7.9 Robot movements in a road environment. The arrows in (a) indicate the moving directions of the vehicles and (b) is the modified environment to ensure conformance with traffic regulations.

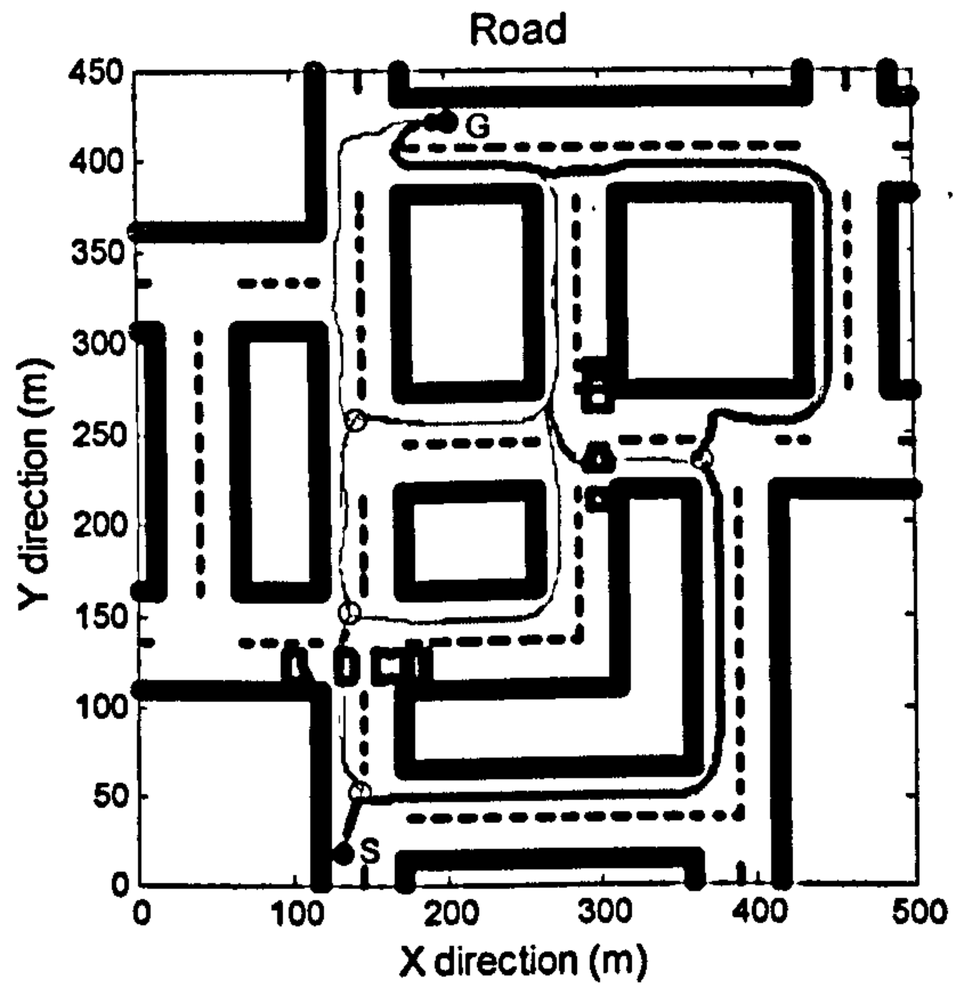
The mission for the robot to conduct in the test environment was to travel between two specific locations on a number of occasions during a day. The robot can then plan a best path in terms of travel time based on the waypoints obtained during the pertaining traffic patterns. The vehicles move at a constant speed of 45kmh^{-1} when passing through junctions, and the same speed was also assigned to the robot, except that the robot was able to stop and wait for junctions to become clear. The waiting behaviour was realised by viewing the road edges and traffic markings as static obstacles. Varying traffic conditions were simulated with three separate values of arrival rate with the aim of minimising travel time from start to goal. The paths generated are presented in Figure 7.10 and the times needed to follow the planned paths the three situations are listed in Table 7.2.



(a) λ = 15 vehicles per minute for each of the two junctions



(b) λ = 30 vehicles per minute for the left junction and λ = 15 vehicles per minute for the right junction



(c) λ = 30 vehicles per minute for each of the two junctions

Figure 7.10 The paths generated for three traffic conditions. The markers 'S' and 'G', represent the start and goal points for the robot and circular markers denote the generated waypoints. The thickened lines indicate the path generated by the planner.

Table 7.2 Path costs for the best path in terms of travel time obtained over 10 experiments for each path segment.

λ (vehicles/min)	left junction	15	30	30
	right junction	15	15	30
travel time (s)		41.4	74.4	78.6

It can be seen from the above results (Figure 7.10 and Table 7.2) that when there was heavier traffic at the junctions (larger values of λ), the robot selected the longer path to the destination as this route is likely to save travel time. Figure 7.10(a) has light traffic and the robot can take the shortest path (in length) to the destination with few delays at the two junctions. In Figure 7.10(b), more vehicles passed through the left junction, whereas the arrival rate remained the same the right junction. As more time is required on average for the robot to pass through the left junction, the robot may choose the longer path to avoid the increased traffic. The heaviest traffic conditions for the two junctions are shown in Figure 7.10(c) and here the robot took a longer path to the destination to reduce travel time.

7.6.3 Comparison of dynamic avoidance behaviour

This section first introduces the ND method (Minguez and Montano 2004 and 2005; Minguez, Montesano and Montano 2004; Minguez, Osuna and Montano 2004; Montesano, Minguez and Montano 2005), which has been chosen to provide a comparison with the proposed avoidance technique, then presents the results of the comparison which are finally discussed. The reason to select ND algorithm for comparison study was given in section 7.6.

The ND algorithm solves the avoidance problem by performing simplifications and applying the motion laws corresponding to each situation identified from the five pre-defined environmental configurations. A binary tree was used to match the sensor information received with the five configurations. The inputs of the binary tree are the obstacles, the robot and goal locations, and the tree outputs one of the pre-defined configurations by selecting a branch according to the criteria based on inputs and their relations. According to the motion law associated with the configuration identified, a motion command can be computed to produce the best behaviour suited to the current configuration. For each moving obstacle, the location at which a collision occurs in the direction of motion of the robot is computed using the current robot and obstacle velocities, with both the robot and the moving obstacle being assumed to move at constant linear velocities. The obstacle is then placed on the

collision location as a temporary static obstacle and is taken into account in the situation perceived when generating avoidance commands. Further information can be found in the literature (Minguez and Montano 2004 and 2005; Minguez, Montesano and Montano 2004; Minguez, Osuna and Montano 2004; Montesano, Minguez and Montano 2005).

The first comparison was conducted for three simple configurations where the robot encountered a single moving obstacle, these being representative of situations where the angle between the moving directions of the robot and the obstacle is $(0, \pi]$. The paths generated by the ND method and by the dynamic waypoint avoidance (DWA) technique are shown in the left and right columns of Figure 7.11 respectively and the quantitative results are given in Table 7.3. Figure 7.11 illustrates three situations of potential collision between the robot and an obstacle moving from right to left. Case 1 presents the situation where the horizontal component of the velocity of the robot is in the opposite direction to that of the moving obstacle and its vertical component is not zero, so that the angle between the directions of motion of the robot and the moving obstacle is in the range $(\pi/2, \pi)$, before the obstacle is detected. In the second configuration, labelled as Case 2, the robot moves in such direction that the horizontal component of the velocity of the robot is in the same direction as that of the moving obstacle and therefore the angle between the robot and moving obstacle is in the range $(0, \pi/2)$, before the obstacle is sensed. Case 3 shows the robot initially moving in the opposite direction to the moving obstacle so that the angle between the robot and the moving obstacle is π.

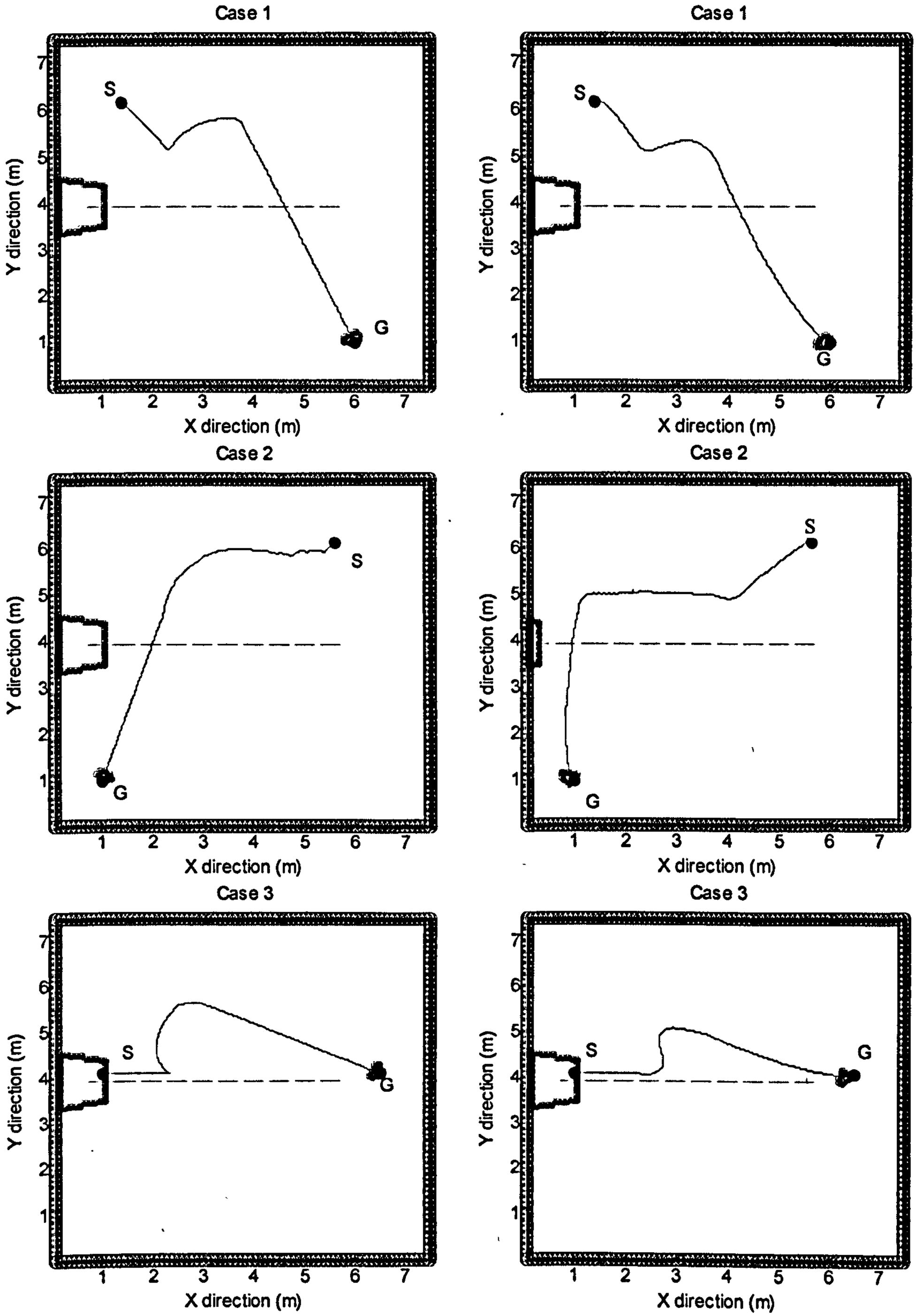


Figure 7.11 The paths generated by ND (left column) and the DWA algorithm (right column) for three cases. The markers 'S' and 'G', represent the start and goal points for the robot. The solid lines indicate the trajectories taken by the robot, and the dashed lines present the paths of the moving obstacle.

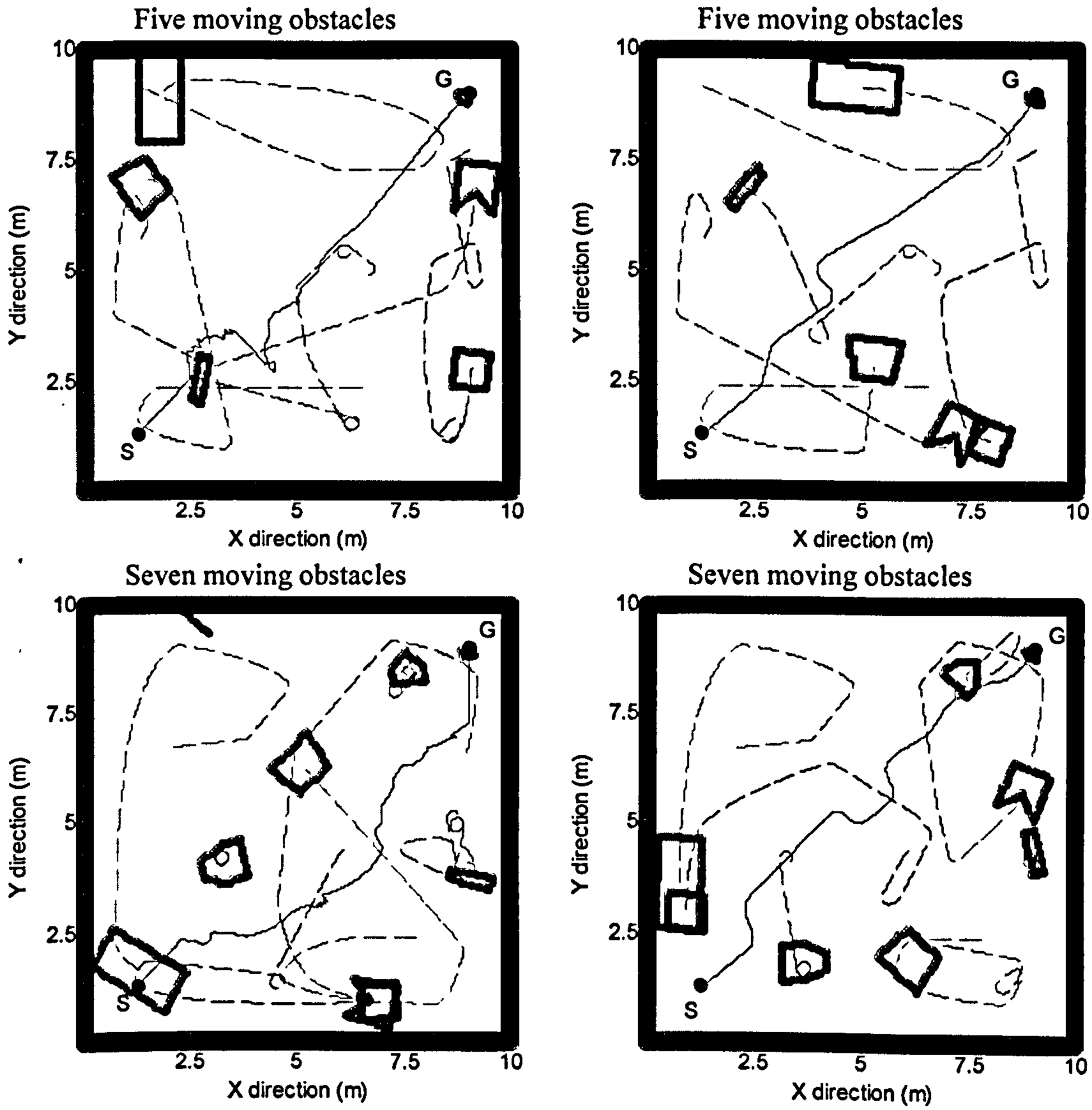
Table 7.3 Path length and execution time taken by ND and the DWA technique for three cases.

case	algorithm	path length (m)	execution time (s)
1	ND	8.2	5.08
	DWA	7.58	8.06
2	ND	8.15	4.97
	DWA	8.9	10.8
3	ND	7.25	4.69
	DWA	6.33	6.84

It can be seen that the paths generated by DWA are shorter than those produced by ND for cases 1 and 3. When moving obstacles are detected, ND attempts to adjust the direction of motion towards free space close to the goal point, but the robot attempted to move away from the obstacle, as shown for cases 1 and 3. Under the control of DWA, the robot narrowly avoids the moving obstacle, as motion is based on the criteria of minimising path length. For case 2, the robot took a longer trajectory under navigation by DWA than by ND. It can be seen that the first part of the path is different for two algorithms, with obstacle avoidance under ND beginning earlier than that under DWA, even though the detective sensor range was the same for the two navigation systems. To follow closely the goal direction, the robot controlled by DWA avoids the moving obstacle only when necessary. In DWA, the robot moved further to the left in avoiding the obstacle compared with that in ND, implying that the robot under ND control tends to be slower in its movements as the robot often performed rotational movements when the distance between the robot and the moving obstacle is close to the limit of the sensor range. More specifically, when the robot's distance from the obstacle is slightly less than the sensor range, the robot attempts to move away from the obstacle, so that it can no longer be detected and the robot then resumes motion towards the goal point. The robot may then detect the presence of the obstacle again and the cycle repeats. Note that ND was designed for a robot that has no translational displacement while under rotation, which is a principal limitation discussed by ND's authors (Minguez and Montano 2004). The execution times for the complete trajectory as shown in Table 7.3 for the two algorithms, indicate that DWA

produced a slightly slower response than did ND, but the mean execution time required for each step of decision-marking is approximately 30ms, a promising result for real-time navigation.

Figure 7.12 shows the paths followed by the robot under the control of ND and DWA for the same start and goal points in each of four environments containing identically 5, 7, 9 and 11 moving obstacles. Simple motion ability was designed for each obstacle, in that they continued to travel forwards unless any obstacle other than the robot is detected, where it turns to the left by a pre-defined angle. Table 7.4 shows the results of the quantitative evaluation of the path length and execution time.



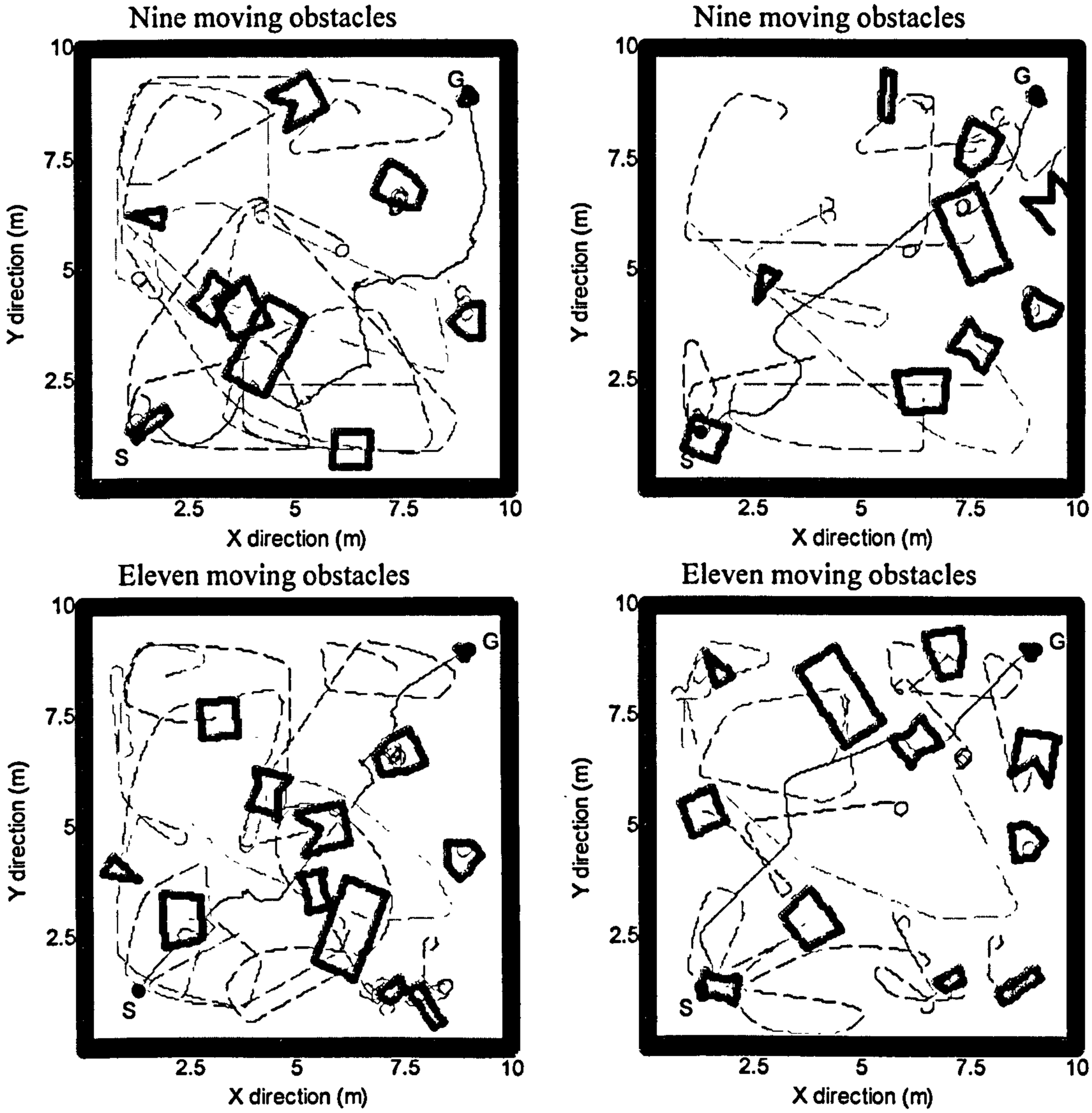


Figure 7.12 The paths generated by ND (left column) and DWA (right column). The markers 'S' and 'G', represent the start and goal points for the robot. The solid lines indicate the trajectories taken by the robot, whereas the dashed lines are the paths of the moving obstacle.

Table 7.4 Path lengths and execution times taken by ND and DWA for four configurations shown in Figure 7.12.

number of dynamic obstacles	algorithm	path length (m)	execution time (s)
5	ND	15	24.6
	DWA	11.7	27.8
7	ND	14.4	31.6
	DWA	12.2	34.2
9	ND	21.3	53.6
	DWA	12.4	42.2
11	ND	16.9	51.3
	DWA	12.7	52.3

The paths computed by DWA are subjectively smoother than those produced by ND, probably due to the oscillations that arise in the latter algorithm, as discussed earlier. The shorter path generated by DWA results from the narrow margins used to avoid moving obstacles, whereas ND tended to err on the side of caution and gave greater leeway to moving obstacles. During the experiments, it was observed that the robot under ND control frequently collided with the moving obstacles on more than one occasion as it headed to the goal. Recalling that in the ND algorithm calculations are performed by assuming the moving obstacle is at collision location, it appears this strategy may cause problems when the robot is unable to completely leave the path of the moving obstacle once it is at the collision location. From table 7.4, ND can be seen to have a shorter execution time, except for the situation where 9 moving obstacles were present. Since the execution of the robot and moving obstacles were performed in a sequential manner, the execution time increases with the number of moving obstacles.

7.7 Comparison with other hybrid architectures

A number of hybrid architectures reported in the literature were designed for navigation problems where a number of moving obstacles are presented. A brief description of those architectures was given in section 7.1 and the reasons for selecting architectures for comparison with DWA are as follows: (a) the planning modules in those architectures have a tactical role in guiding the robot; (b) the

reactive parts are in charge of local navigation and (c) the hybrid systems were claimed to be able to navigate in environments containing moving obstacles.

The systems implemented by Low, Leow and Ang Jr (2002 and 2003), Minguez and Montano (2005), Minguez, Montesano and Montano (2004) and Urdiales *et al.* (2003b) all operated in grid environments, whereas DWA operates on the original workspace of the robot to directly generate a set of waypoints for the static aspect of the environment. No metric map is required to generate an abstract topological map or a path.

The hybrid architecture proposed by Low, Leow and Ang Jr (2002 and 2003) was arranged with a short time scale for the reactive model and long time scale for the deliberative planning. The inputs for the target reaching, obstacle avoidance, and homeostatic control of the reactive model are checkpoints, local obstacles, and the internal states respectively. Planning operates on the world model. Four layers were developed for the architecture in Urdiales *et al.* (2003b), with the *geometrical modelling layer* constructing a grid map through sensors, the *topological modelling layer* abstracting a topological map from the grid, the *router planner* operating on the topological map, and the *local navigation layer* reacting directly to the sensory information. The planning and reactive motion modules in the navigation system described by Minguez and Montano (2005) and Minguez, Montesano and Montano (2004) operate on the grid map constructed from the sensor readings with the path generated in the planning module assisting the local navigation. In contrast, DWA was specifically designed for application in dynamic environments in that three layers were defined according the different environmental information coupled to each layer and the information accounting for the three different aspects of the dynamic environments.

7.8 Discussion

The experimental results presented in section 7.6 verify the strategy proposed for avoiding collisions with moving obstacles and the effectiveness of statistical

explorations for the identification of characteristics of dynamic environments. Although the avoidance technique presented in this paper was designed to deal with a single moving obstacle, an avoidance command can be determined for the robot facing multiple moving obstacles by selecting the intersection of the avoidance directions of each individual moving obstacle. Furthermore, additional behaviour was developed to deal with the situation where both dynamic and static obstacles hinder the robot's progress simultaneously. The waiting behaviour is more apparent in the traffic environments (see Figure 7.10 for an example). As reversing is normally not permitted in traffic roads, the robot stops and waits in front of the congested junctions until they become clear. Further investigation is required to verify the navigation strategy when multiple moving obstacles are presented concurrently. However, it must be recognised that, in realistic environments that do not embed any intelligence in the moving obstacles, it cannot be guaranteed that collisions can always be avoided. The experimental results reported in the previous section show the effectiveness of the statistical exploration in planning a suitable path while minimising travel time. For environments that exhibited no apparent dynamic characteristics, the statistical exploration may provide few heuristics to direct the planning.

The dynamic avoidance behaviour proposed in this chapter assumes the heading of the moving obstacle remains the same when entering the detection range. This assumption can be relaxed by the inclusion of the angular velocity of the moving obstacle when generating a new path. To avoid inevitable collisions with the robot, the angular velocity of the moving obstacle needs to be constrained within a certain range when the moving obstacle appears in the detection range. The method presented here can be used with minimum modification for the situation where the moving obstacle can rotate when in the detection range.

Although the experimental results reported in this chapter were obtained for the robot moving at constant speed, the method proposed is still applicable to a robot with variable velocity.

Also, it would be interesting to investigate what behaviours might result if the same control algorithm is applied to multiple robots, and communication between robots regarding traffic rules (such as walk on the left) could potentially greatly reduce the likelihood of collision.

7.9 Conclusion

This chapter has taken an existing hybrid navigation technique that is able to avoid static obstacles and has extended it for application in dynamic environments. The necessary constraints on the physical dimensions and velocity of the moving obstacles to avoid collisions with the robot have been determined. A series of experiments in which dynamic objects moved in a random manner demonstrated that a robot equipped with this high-level behaviour can successfully avoid collision. Statistical recording of the times taken for the robot to traverse paths between waypoints during exploration was shown to be effective in planning appropriate future paths. However, one of the limitations inherited from previous navigation systems is that the planning of a suitable path is performed between the same start and goal points as those used when collecting the waypoints. An approach designed to overcome this limitation is reported in the next chapter.

Chapter 8

GENERALISED WAYPOINT-BASED NAVIGATION SYSTEM

The navigation system presented in this chapter is a generalised version of the waypoint-based navigation system introduced in the previous two chapters. The navigation systems described thus far are limited to planning robot movements between the same start and goal points as were assigned in earlier navigation tasks and the waypoints used in the previous two chapters are task-oriented. This chapter describes the results of implementing a scheme to allow navigation to be planned between arbitrary start and goal points. This is achieved by preserving the main structure of the waypoint system, but adding an extension that is able to record and apply information stored in a compact fashion regarding earlier navigation activities between pairs of waypoints. The work presented in this chapter will be submitted to the IEEE Transactions on Systems, Man and Cybernetics, part B.

This chapter firstly reviews the relevant literature, and gives an overview of the generalised navigation system, before giving the formal definition of the waypoints and describing in more detail the environmental knowledge stored in the database. The planning algorithm itself is presented in section 8.4, with emphasis given to differences from the waypoint algorithm of earlier chapters. The experimental results are presented in section 8.5.

8.1 Related work to path approximation and graph search by genetic algorithms

The navigation system introduced in this chapter preserves the main structure of the hybrid architectures presented in the previous two chapters, albeit with modifications to its components, knowledge base and deliberative module, so as to empower the planning for future navigation. This section concentrates on the literature related only to the modifications.

Spline techniques (de Boor 1978; Kvasov 2000; Noggle 1993) have been widely used to generate trajectories by interpolating a set of points that the robot needs to traverse (for example Connors and Elkaim 2007; Dyllong and Visioli 2003; Guan *et al.* 2005; Hao and Agrawal 2005; Magid *et al.* 2006; Nikolos *et al.* 2003; Park and Bobrow 2005). The main reason to adopt splines rather than straight line segments is that trajectories constructed by splines are smooth avoiding sharp turns and accommodating the dynamic constraints of the robot (such as being unable to rotate without translational displacement). In robot navigation, cubic splines are the most popular choice as they possess the important feature of second order continuity, that is continuity of location, speed and acceleration (Guan *et al.* 2005; Korb and Troch 2003). A number of authors describe the application of splines to curve-fitting problems in robotics. Simon and Isik (1993) applied trigonometric splines to approximate the desired robot path within a given knot tolerance. Gu and Owens (1998) developed a system in which a parametric cubic spline was used to represent the observed motion of a human operator to provide an imitation by a robot. Korb and Troch (2003) developed a data reduction algorithm for manipulator path planning using cubic splines to approximate a given path created by linear interpolation using a least-squares method within a certain error bound. The aim was to represent the original path by a cubic spline within the specified error bound and with as few knots as possible. The principal idea was to assign each knot a weight estimated according to its significance in the approximation with the knots being deleted in order of least importance if a proposed spline satisfies the specified accuracy. However, if the approximation accuracy is not met, the algorithm will insert additional knots close to

where the error bound was exceeded. Bessonnet, Seguin and Sardain (2005) presented an algorithm for walking pattern synthesis where the generalised coordinates of motion were approxiamted by a spline fitted at specified knots uniformally spaced in time along the path. A least-squares method was used to fit the curves, there being no rigid requrement to pass through the knots.

The use of GAs when operating in grid-based or in the original workspace was reviewed in chapters 4, 5 and 6, but a specific area not covered and of relevance to the work in this chapter is recent work in which GAs have been applied as a graph search technique, particularly for the shortest path problem. This literature is reviewed in this section. In recent literature, there has been an increasing number of reports investigating the application of GAs to the shortest path problem, defined as finding a path between two designated nodes with minimum total length or path cost. It is a fundamental problem for many applications, such as transportation, routing and communication (Davies and Lingras 2003; Gen, Cheng and Oren 2001; Sniedovich 1988). The GA proposed in (Ahn and Ramakrishna 2002) randomly initialised a set of routes constrained to be feasible paths in which all nodes in the path visited only once. The site for the one-point crossover operator was restricted to a pair of common genes of the parents, randomly selected by the pairwise (the tournament size is two) tournament selection strategy. The mutation operator altered the partial route from the mutation node to the destination node using the initialisation mechanism. Furthermore, the efficiency of the proposed GA was verified by favourable experimental comparisons with Dijkstra's algorithm and two other GAs reported in the literature. GAs with a similar structure were developed or adopted by a number of researchers (Davies and Lingras 2003; Wu and Ruan 2004).

Davies and Lingras (2003) proposed a GA to solve the rerouting problem in dynamic and stochastic networks. The initial population contained a group of candidate paths of variable length representing a number of nodes connecting the current node to the destination node. By use of specifically modified crossover and mutation operators, the paths produced during the evolution were constrained to be feasible and a strategy was adopted that combined an elitist approach with the application of a roulette wheel

to favour fitter individuals. To prevent incomplete paths appearing in gene pool, the one-point crossover operator was constrained to sites having common genes in the parents and the two-point crossover required the two sets of common genes constrained to be present in both parents in the same order. The mutation mechanism randomly selected two genes and the part between the two genes was reproduced. To account for the dynamic changes in the network, a weight was assigned according to the length of waiting time assigned to each node. A GA with a similar structure was developed by Wu and Ruan (2004) to operate in a connectivity information matrix that was predefined to indicate the connectivity of nodes. As is the work by Davies and Lingras (2003), the selection scheme employed was a combination of the roulette wheel and elitism methods, but the mutation operation was slightly modified in that the partial path after mutation node was generated from the end node to the mutation node by an initialisation mechanism. The two GAs proposed above are generational GAs, since the entire or at least a large fraction of the population is altered by genetic operations carried out between successive generations. In the GA implemented by Ji, Iwamura and Shao (2007), a smaller number of individuals were modified by the genetic operations. The initial population contained a set of chromosomes of variable length, representing only the feasible paths. A rank-based selection scheme was used to determine the parents for genetic operations. The crossover operation was performed such that the parts after the nodes (randomly chosen from the common nodes between a pair of parent paths) were swapped. The mutation point was randomly selected and a new path from the mutation node to the final destination node was generated by chromosome initialisation. An evolutionary algorithm was proposed by Mooney and Winstanley (2006) to solve multicriteria path optimisation problems using a pareto-elitist approach. The randomly generated initial population contained a set of feasible paths without loops and the individuals were assessed using pareto domination and a number of elite individuals were copied directly into the next generation. The pairwise (the tournament size is two) tournament selection scheme was used to select parents and further valid paths were generated through one-point crossover and mutation. The result obtained from real world road networks demonstrated that the new algorithm could outperform that of Dijkstra's algorithm. To solve the multiobjective problem for multicast routing, Garrozi and Araujo (2006)

used a single objective function combined nonlinearly for three separate objective goals. Each row of a table-like chromosome represented a route from a single source to one of the destinations and crossover operators that incorporated problem-specific knowledge were used to exchange information between parents chosen based on their relative fitness. The two mutation operators functioned as follows: the first one inserted a partial path after the mutation points and the second eliminated any loops that may have been generated during evolution.

A number of authors attempted to combine GAs with other algorithm for shortest path problem. Those approaches are less relevant to the new planning algorithm described in this chapter, and only very brief introduction are given below. Duan and Yu (2003) proposed an algorithm, termed genetic shortest path algorithm, to optimise a power distribution system, in which a local optimisation method was developed to find local optima, from which the global optimum was generated by a GA. A hierarchical approach was developed by Wu and Ruan (2006) that incorporated the Floyd algorithm into a GA in order to solve the shortest path problem with fixed intermediate nodes but without constraints on their order in the chromosome.

8.2 Generalised waypoint navigation system

The generalised version of waypoint navigation system preserves the hybrid architecture of the waypoint navigation systems of the previous two chapters, with the knowledge base functioning as an interface between the reactive and deliberative layers. Reactive control is derived from the decision trees generated by a series of robot movements through a set of simulated environments designed to embed suitable control rules. Deliberative planning takes advantage of knowledge of the environment gradually acquired during earlier navigation tasks and stored as waypoints. As explained in section 6.2, this hybrid architecture attempts to create high-level intelligent navigation behaviour by selectively recording the past experience, timely monitoring the current information, and tactically planning the future navigation. The waypoints used to represent the environment consume little memory, but their number is dependent on the navigation task currently performed. Overcoming this

limitation is the main purpose behind the development of the generalised hybrid architecture. One principal addition in the generalised version of waypoint navigation system is that more information is recorded regarding the reactions that result as a consequence of the presence of static obstacles. This extra knowledge is used to extend the capability of the deliberative control to allow path planning to be performed between any two locations in the environment. As memory capacity is a scarce resource in embedded systems, the raw information gathered is progressively processed in order to reduce the data storage requirement. The information so recorded permits paths previously taken to be approximately restored with little computational effort to enable future path planning or to provide a localisation reference. To permit this extra information to be used in the generation of paths between any specified two points in the environment, the path planning approach has also required extensive modification.

A high-level overview of the operation of the generalised waypoint navigation system is now given. Assuming no initial knowledge of the environment, the robot navigates to the goal under reactive control with waypoints being determined as static obstacles are encountered. Two types of waypoints are now obtained. The first is the same as that in the previously-described waypoint system, in that the waypoints indicate that the robot is approaching a newly encountered obstacle and its heading is being forced to change. The second type of waypoint is introduced in the generalised waypoint system and the waypoint marks the location where the robot is no longer avoiding the obstacle and instead reverts to heading directly towards the goal. On generating a waypoint of the first type, memory is temporarily assigned to record the tracking of the path segment that the robot follows around the obstacle. Tracking stops once a waypoint of second type is generated. As there is an alternative path to explore to circumnavigate the obstacle starting from a waypoint of first type, such waypoints can be labelled as unexplored until the alternative path has been followed. As the second type of waypoint is the end point of the path segment, no similar exploration is required. Following the recording of a waypoint of the second type, the robot processes the information regarding the segment and stores only that necessary to permit the deliberative navigation module to plan future paths involving the segment.

When the robot finishes its current task and no further navigation tasks are required to be conducted immediately, the exploration function can be invoked to follow either alternative paths that start from unexplored waypoints or to investigate uncharted regions by selecting locations randomly, or heuristically if the robot has knowledge of the distribution of the obstacles.

Using the knowledge gained during reactive navigation and exploration, paths that can take advantage of the recorded segments can be generated by the deliberative system prior to beginning a new navigation task. As in previous waypoint implementations, the path generated will consist of a set of sub-goals for the robot to follow successively in the new navigation task, with the actual movements between these sub-goals remaining under the control of the reactive layer.

8.3 Knowledge base

In navigation problems, memory is often required for the deliberative navigation system to maintain information about its environment in order to localise the robot and generate a plan for future navigation. An important issue in the practical realisation of hybrid navigation systems is not just the nature of this information, but also its storage, as memory capacity is often limited. Some hybrid systems described in the literature construct an exhaustive model for the environment (as discussed in section 3.1.1) in order to achieve good navigation performance. Such models not only consume substantial memory, but dealing with such large quantities of memory involves significant computational overheads (Santos, Castro and Ribeiro 2000; Urdiales *et al.* 2003b). A number of hybrid systems reported in literature adopted hybrid maps or topological maps abstracted from a metric map (see section 6.1) to enhance the planning efficiency. The construction of hybrid maps and abstract topological maps is computationally complex. The implementation of the navigation system presented in this chapter requires that only a relatively small number of points be extracted to record the trajectories around obstacles. The memory usages for the experiments are reported in section 8.5.5.

8.3.1 The two types of waypoint

This section defines the two types of waypoints employed in the navigation system introduced in this chapter. In the waypoint navigation systems described in chapters 6 and 7, a location is marked as a waypoint only when the robot needs to deviate from its current path due to the presence of a static obstacle (see section 6.2.2). In contrast, two types of waypoints are defined in the new navigation system. As the particular reactive system used in the current work neither identifies nor internally represents boundaries between primitive behaviours, the identification of waypoints is instead based on robot heading changes and sensor readings. A survey of other literature related to waypoints was presented in section 6.2.2. No publications that determine waypoints based on behaviour changes have been found in the literature, and so the idea to build behaviour-driven waypoints is claimed by the author as being novel.

An example of the first type of waypoint found in previous waypoint navigation systems is shown in Figure 8.1 represented by a circular marker. Such a waypoint is recorded when the robot's behaviour changes from one of goal-seeking to one that avoids static obstacles detected by its sensors. The reason to record those waypoints is to indicate the approximate location of obstacles and those points in the environment where the robot can take one of two alternative paths to negotiate obstacles. On the first recording of a waypoint, only one of the two paths around an obstacle is taken, with the waypoint being marked as unexplored until the second path has been taken. The first type of waypoint also marks the beginning of a path segment around the obstacle (shown by broken lines); such as a segment being the principal method of recording information about the environment in the generalised waypoint navigation approach.

The location at which the robot begins to move away from the obstacle that forced the robot to change its direction and revert to heading towards the goal, is recorded as a waypoint of the second type and is indicated by a star marker in Figure 8.1. This location is in fact that at which avoidance of the obstacle ends and the robot reverts to goal-seeking behaviour. The recording those points is useful as they give the boundary between the obstacle and the free-space with respect to the current

navigation task. A stored path segment will always end at such a location, although this type of waypoint may, in some cases, indicate the initial point for a second path segment around the obstacle. This is particularly likely to occur when the robot is circumnavigating a concave obstacle edge. Although this second type of waypoint is effectively a node that can be selected to aid obstacle avoidance in future planned paths, it is not used to initiate exploration.

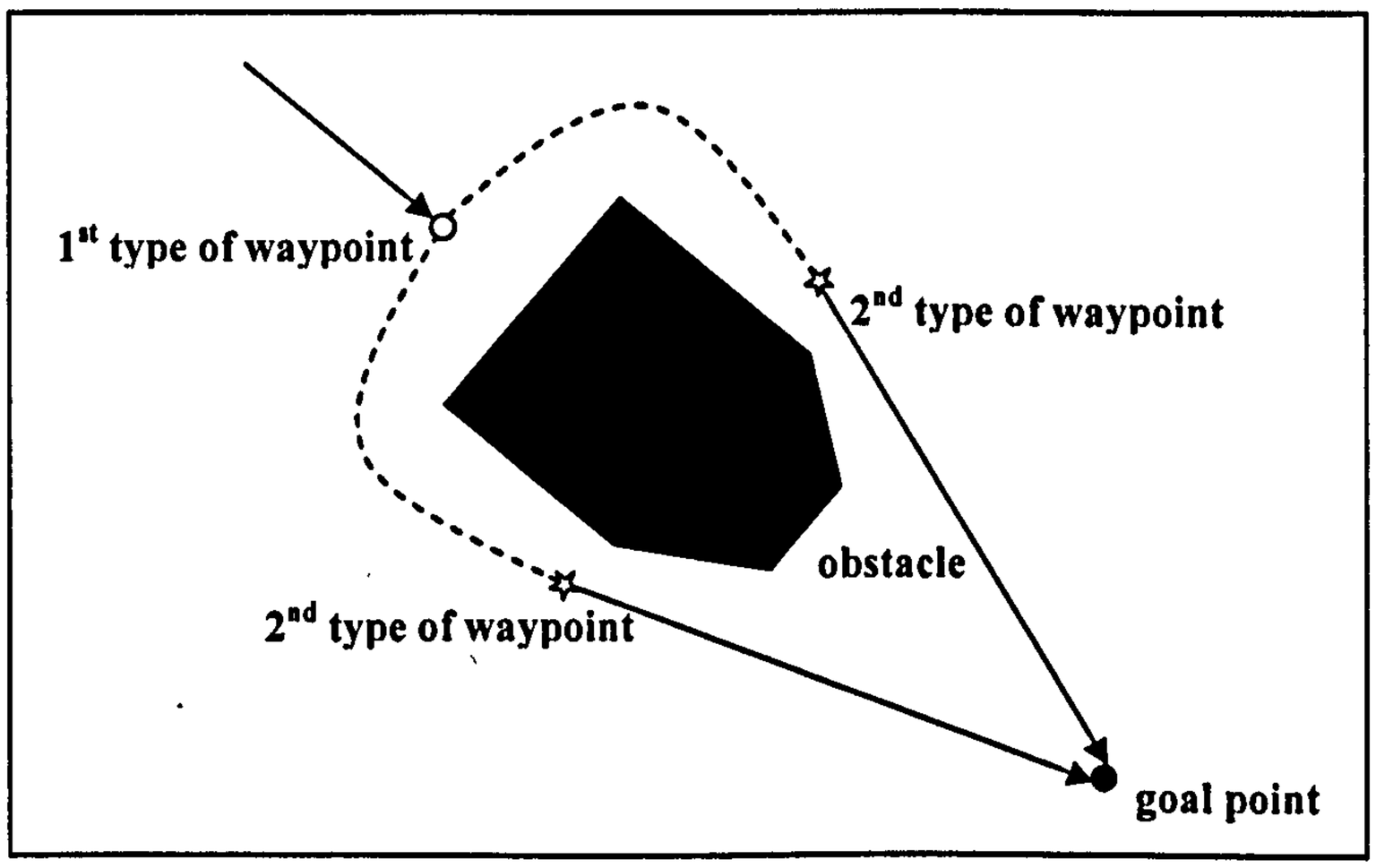


Figure 8.1 An example of the two types of waypoint. The circular maker denotes the first type of waypoint, whereas the second type of waypoint is indicated by star markers. The broken lines between the two types of waypoint are path segments that the robot has followed while avoiding the obstacle.

8.3.2 Curve segment representation

In the generalised navigation system, the path segments, such as those in Figure 8.1 originating from a waypoint of the first type and terminating at a waypoint of the second type, are recorded for path planning purposes. The segment itself is a trajectory generated by the reactive navigation system as a consequence of its obstacle avoidance behaviour. The shape of the segment reflects that of the obstacle the robot is avoiding. The previous navigation systems proposed in chapters 6 and 7 do not retain the useful information provided by such segments traversed during earlier navigation tasks. To take advantage of such valuable information for future planning is the main reason to record the segments instead of a few orphan points.

In order to be able to reproduce the segment for path planning purposes, a method is needed of recording the sequence of coordinates progressively accumulated as the robot reactively navigates around on obstacle. To reduce the quantity of information that needs to be retained, a spline approximation technique (de Boor 1978; Kvasov 2000; Noggle 1993) has been adopted. Splines generally provide an efficient solution to the representation of complex shapes, as the piecewise polynomial approximation that results is generally of a lower order than that obtained when approximating using a single polynomial, thereby reducing the storage requirement for any given accuracy of fit.

To make clear why a suitable approximation to the actual path taken by the robot when performing obstacle avoidance is needed by the planner, consider the representation of the path segment by a straight line connecting the end points of a path segment. Although the line can be easily manipulated, it is unable to provide a sufficiently accurate representation of planned paths that avoid obstacle collisions. Figure 8.2 shows an example in which the planner is attempting to determine a suitable path between waypoints A and B. As part of this assessment, a test of the intersection of line AB with all path segments is required. If the path segment between waypoints C and D is recorded as a straight line, no intersection is detected. If the path segments around obstacles are instead recorded as splines, the planner will identify the intersection with the spline CD and a closer representation of the actual path the robot will need to take can be planned. Clearly this example does not identify a specific accuracy requirement and this is discussed further in section 8.4.1.

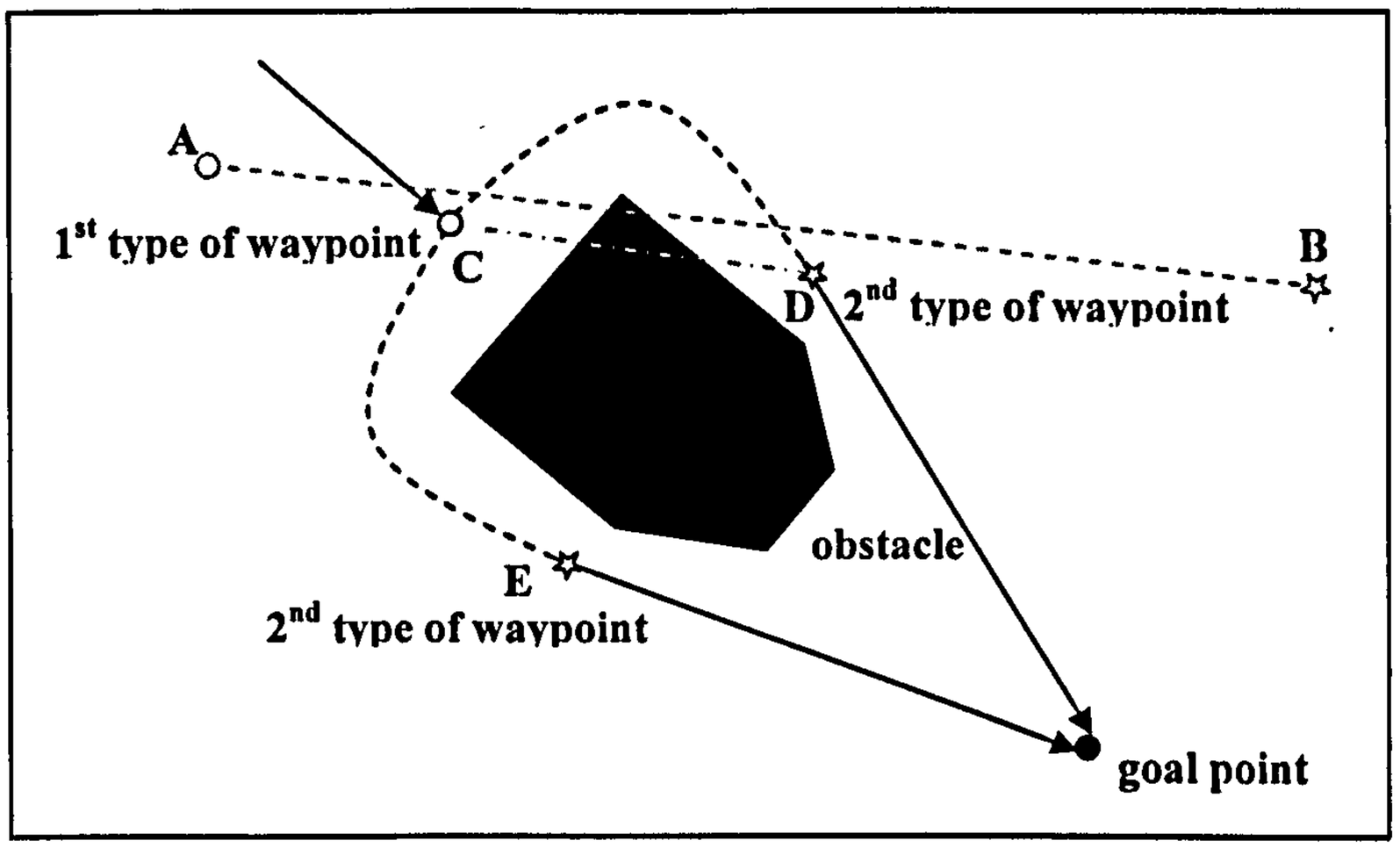


Figure 8.2 The path segment represented by a straight line. The straight line between C and D is indicated by the grey 'dash-dot' line, whereas the dark broken line denotes the actual curve between the pair of waypoints. The grey broken line between A and B is a proposed path connecting two waypoints located near other obstacles.

The conventional approach when fitting a curve is to use a least-squares approach (Guo, Gui and Yang 2003; Noggle 1993). However, in the current implementation, cubic spline interpolation expressed in parametric form is used in place of least-squares approximation to ensure that the spline passes through the end points of the path segment. Although the elegant feature of second order continuity of cubic splines makes it a popular choice for modelling trajectories (Guan *et al.* 2005; Korb and Troch 2003), the choice of cubic spline here arises more from the need to control the computation complexity and memory usage. Cubic polynomials are chosen as a reasonable compromise; they are more easily manipulated than polynomials of higher order, whereas, to achieve the same resolution, splines of lower order are likely to require additional pieces (Guan *et al.* 2005), thereby increasing the memory needed to store the coefficients and breaks for the spline. Figure 8.3 illustrates the difference between the fitting results of using least-squares approximation method (the upper path segment) and interpolation (the lower path segment). As the spline curve generated by least-squares approximation approach does not pass through the end points that are the two waypoints, it may not be possible to identify correctly all intersections that occur between the true path segment and a path that coincidently passes through or near those waypoints. In contrast, this problem can be avoided if

the interpolation is used as the approximation technique, as breaks will coincide with the end points of the actual path segment. Singularity problems can arise when using the explicit equation $y = f(x)$ to represent a planar curve which is closed or multiple valued and has vertical tangents in a fixed Cartesian coordinate system. Although these limitations inherent to the explicit form can be overcome by using the implicit form $f(x, y) = 0$ to represent the curve, its determination is computationally complex, requiring the solution of a non-linear equation for each data point. The parametric form $x = h(t)$ and $y = g(t)$ has advantages over both the explicit and implicit forms: it is able to represent curves that the explicit equation cannot, while also having reduced computational complexity compared with the implicit form. The path segment is expressed in the following form, where the parameter *t* is travel time.

$$x = h(t) = a_1 t^3 + b_1 t^2 + c_1 t + d_1 \qquad \text{Equation 8.1}$$

$$y = g(t) = a_2 t^3 + b_2 t^2 + c_2 t + d_2 \qquad \text{Equation 8.2}$$

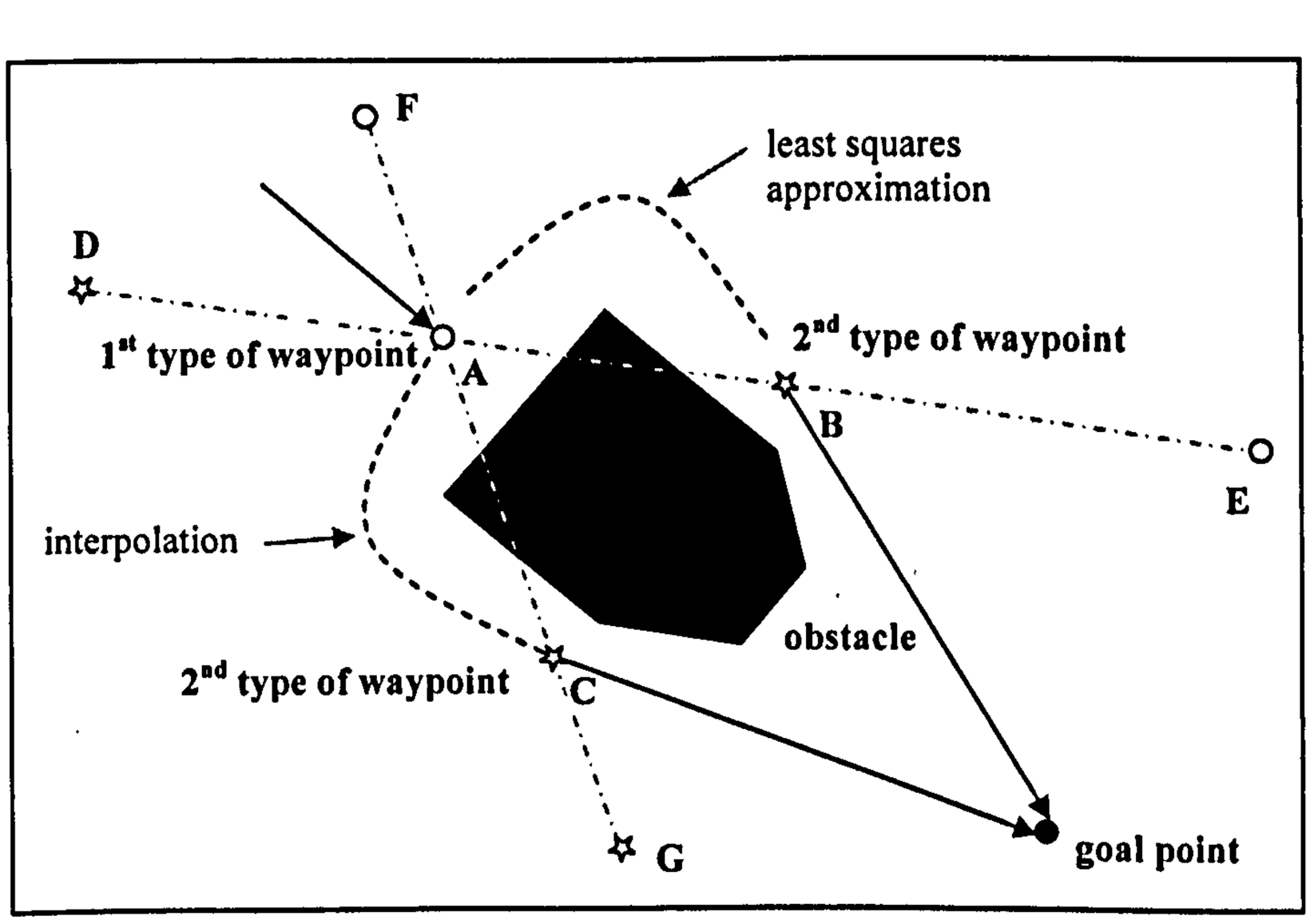


Figure 8.3 An example of intersections between straight line paths (illustrated as grey dash-dot lines) and spline curves (shown as black broken lines), generated by using least squares approximation (between A and B) and interpolation (between A and C).

The knot sequence is a list of monotonically non-decreasing values and one of the critical components in constructing a spline for a given set of data points. The

uniform selection scheme (Bessonnet, Seguin and Sardain 2005; de Boor 1978) is used to determine a knot sequence for spline construction. Residual analysis is performed between the data and the values generated by the constructed spline for each data site. If the evaluation indicates the current spline is unable to meet the requirement determined from the residual criteria, the algorithm proposes a new spline with an additional cubic polynomial and evaluates the goodness of fit of the new spline. This process is guaranteed to complete in a finite number of iterations as the spline will pass through all the data points exactly once the number of polynomials in the spline is one less than the number of data points. Under normal circumstances, process terminates once a spline meets the residual criteria. More exacting criteria result in a better fitting spline, but one which takes longer time to process. The residual value needs to be set according to the requirements of the planner, rather than by the navigation actions, which are performed in a reactive manner. Currently, the residual value is set to be equal to the step length of the robot, namely the distance it moves between navigation decisions, as this is approximately the error involved when recording the path segment data. Once the spline fitting is complete, the robot stores the coefficients of the spline generated as well as the breaks for each polynomial for future use in planning operations, while the original data of the path segment are discarded.

8.4 The planning algorithm

When a new navigation task is assigned, the robot needs to generate a path between the start and goal points. For a suitable path to be determined, the estimated overall path length is found from the individual path length between pairs of waypoints stored in a cost matrix. To perform planning, a genetic algorithm is applied that has a steady-state structure, in the sense that successive generations differ by only a single individual or by one pair of individuals (depending on the genetic operator applied). The reason for choosing the steady-state structure is that the planning can be interrupted at anytime and the best path evolved thus far can be used as the current solution, while the generational computation is kept to a minimum (as explained in section 2.1). A deterministic crowding technique is used to maintain the population

diversity. The overall operation of the evolutionary planning is given before describing each component in detail in the subsequent sections. The evolutionary process starts with the initial population randomly generated based on the cost matrix. The population is evaluated and the individuals are ranked according to their fitness. A roulette wheel is constructed and its slots are sized according to the ranks of the individuals and a parent or a pair of parents is selected by the roulette wheel to produce offspring. The new generation is formed by inserting the winners of the competition between offspring and parent(s) in the identical niche. The best individual is selected as the path after a suitable number of iterations. As the planning algorithm retains the main structure of the algorithm introduced in section 6.3, a repeated description is avoided here, and, instead, only the components that have been modified are discussed.

8.4.1 Cost matrix

The cost matrix (Wu and Ruan 2004) is square and represents the visibility graph in a form that is convenient for the planning algorithm to manipulate. An example of a cost matrix and its corresponding environment are shown in Figure 8.4 and Figure 8.5 respectively. The algorithm initially constructs an empty matrix with dimension equal to the number of waypoints so far detected plus the start and goal points. Each cell in the matrix is the cost in terms of path length of the path segment originating from the waypoint indicated by the row label and ending at the waypoint indicated by the column label. The cost matrix is then used to plan a path that is a sequence of path segments each originating and terminating in a waypoint. Those cells corresponding to path segments that the robot has previously visited show the actual path lengths and are stored as positive values. Zeroes are permanently assigned to the cells along the diagonal, to cells of paths originating from the goal and to those cells whose paths end at the start point. The cost values in the remaining cells are estimated as follows. The straight line path between the waypoints is assessed to determine whether it is infeasible, that is, it intersects with any of the path segments represented by splines. In practice, this evaluation simply involves solving the equation of a straight line with the two parametric equations (Equation 8.1 and 8.2). If the evaluation indicates a intersection between the straight line path and one of the spline curves, the cell

corresponding to this straight line segment is set to be zero, otherwise the length of the straight line is assigned to the cell but prefixed with a negative sign to denote this path has not previously been followed.

	Start	WP1	WP2	WP3	WP4	WP5	WP6	Goal
Start	0	0	0	0	-4.16	-6.15	-2.03	0
WP1	0	0	2.55	0	0	3.08	0	0
WP2	0	2.55	0	2.38	-4.63	0	0	-2.04
WP3	0	0	2.38	0	2.58	-4.42	3.9	-1.46
WP4	0	0	-4.63	2.58	0	0	0	-2.86
WP5	0	3.08	0	-4.42	0	0	-4.16	-5.29
WP6	0	0	0	3.9	0	-4.16	0	0
Goal	0	0	0	0	0	0	0	0

Figure 8.4 An example of a cost matrix used for planning a path for the navigation task shown in Figure 8.5.

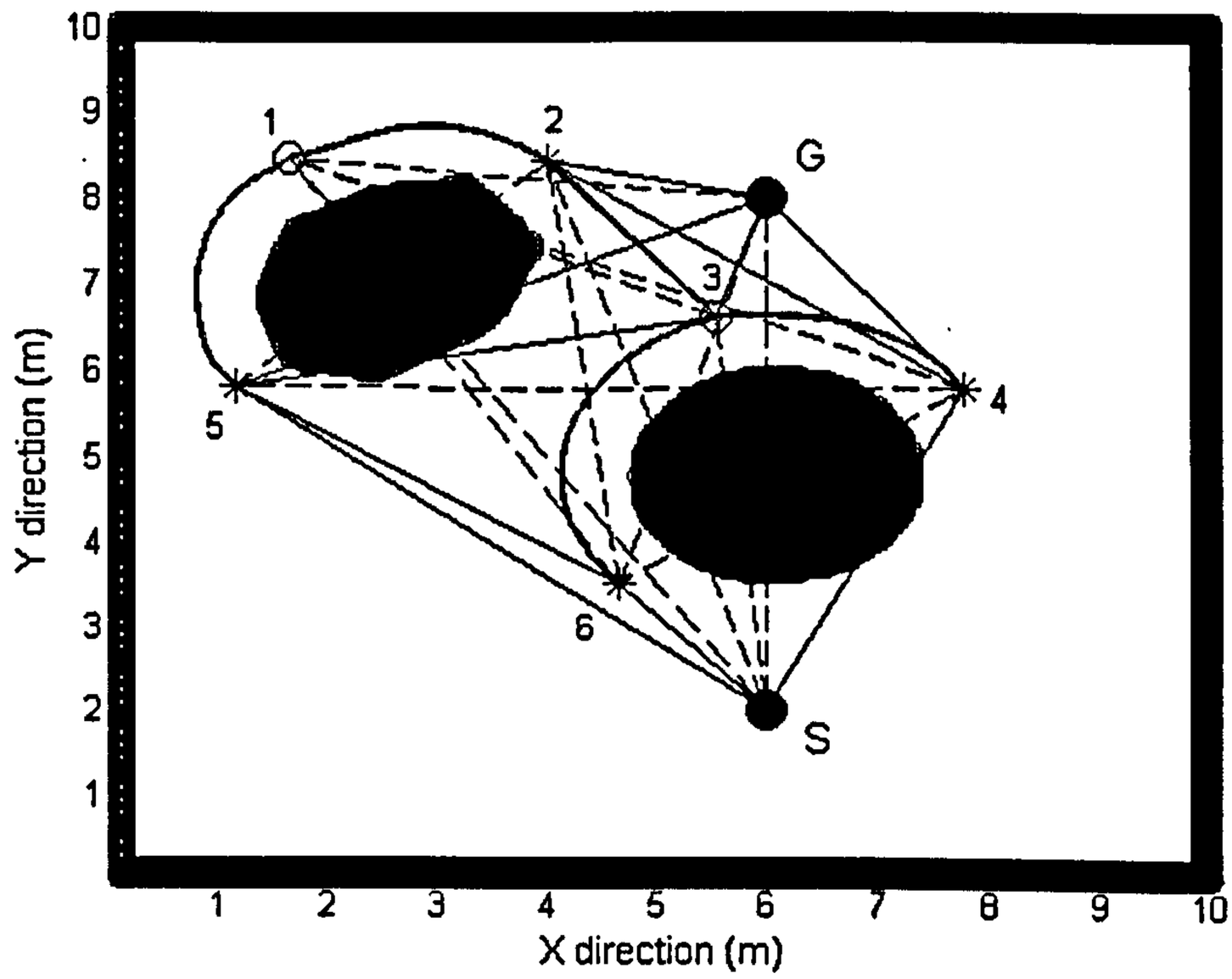


Figure 8.5 Illustration of feasibility evaluation during the calculation of values in the cost matrix. The circular and star markers indicate the first and second type of waypoints respectively. The feasible paths are depicted as solid lines and the infeasible ones as dashed lines. The cost matrix for this visibility graph is shown in Figure 8.4.

In addition, paths between two waypoints of the second type are regarded as infeasible if they connect to a common waypoint of the first type. As can be seen in Figure 8.5, these paths are liable to be infeasible. In the figure, the infeasible

connections are indicated by dashed lines and the solid lines are the feasible paths (both known and unknown). Note that the first row and the last column are all negative values indicating the navigation starts from a new location and will terminate at a new destination. As the cost matrix is symmetrical, then, only the upper triangle is retained.

In the implementation, if a part of the straight line segment and an intersecting curve segment are similar in length, these values can remain in the matrix. Figure 8.6 shows an example where the proposed straight-line path AB intersects the spline CD at points P1 and P2. If the length of the straight line between P1 and P2 is within 5% of the length of the curve between P1 and P2, the straight line AB will be regarded as a feasible path segment. This value of 5% could be optimised according to the navigation task, navigation ability of the robot, and whether intersections with other path segments occur in the vicinity of P1 or P2. This strategy means the robot does not exclude such paths that are likely to be feasible. It is important to note that once robot movement begins, the actual navigation between waypoints will be carried out reactively and so the route followed by the robot will likely be along the path segment between P1 and P2.

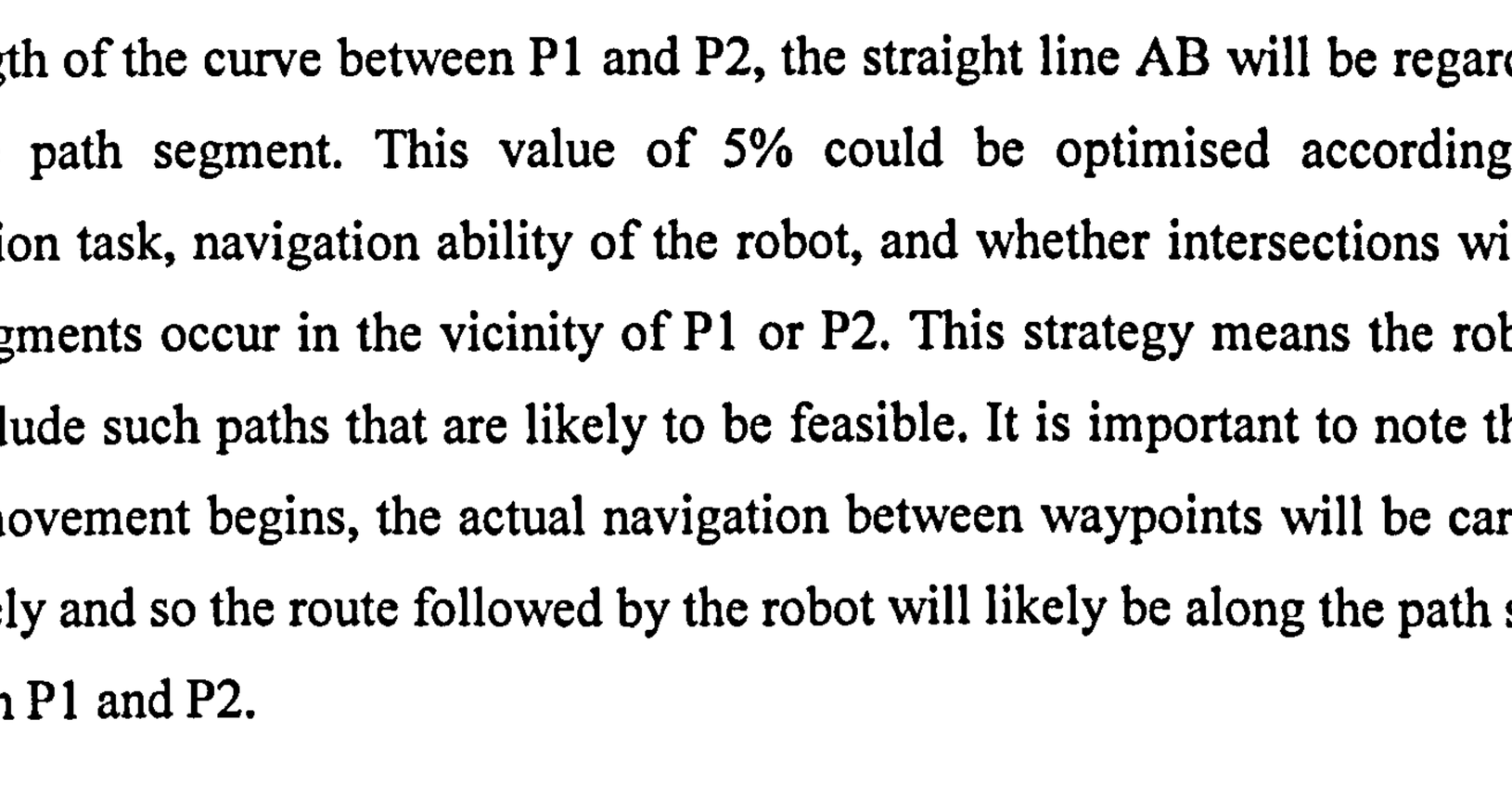


Figure 8.6 Intersection between the straight line AB and the spline CD at points P1 and P2.

The search for an optimal or near optimal path is constrained to the space containing the cost matrix values of feasible path segments, some being known and others not. The elimination of all infeasible path segments makes it possible that a number of individuals will terminate at a waypoint instead of the goal, however, such paths are then also infeasible and so not involved in the evolutionary process.

8.4.2 Initial population

The initial population is generated by incorporating domain knowledge so that all individuals are feasible. The process first randomly selects a path segment originating from the start point (the highlighted cell in the first row in the cost matrix). As illustrated in Figure 8.7(a), the first row and the column of the target waypoint are not considered in the remainder of the generation process, in order to prevent the robot's return to the same location. Using the row for WP5, the random selection of WP3 as the third gene has the effect of emptying the row for WP5 and the column for WP3 as shown in Figure 8.7(b). This process continues until either the goal is selected (as in Figure 8.7(c)), or the row corresponding to the current node is blank or all zeroes. The individuals generated in this way are not of fixed length. In each of Figure 8.7(a), (b) and (c), the lengths of the individual segments are shown; note that the signs of the lengths are ignored when determining the total path length. For this example, it becomes clear that visiting the same node more than once can be easily avoided by careful use of the cost matrix.

	Start	WP1	WP2	WP3	WP4	WP5	WP6	Goal
Start	0	0	0	0	-4.16	-6.15	-2.03	0
WP1	0	0	2.55	0	0	3.08	0	0
WP2	0	2.55	0	2.38	-4.63	0	0	-2.04
WP3	0	0	2.38	0	2.58	-4.42	3.9	-1.46
WP4	0	0	-4.63	2.58	0	0	0	-2.86
WP5	0	3.08	0	-4.42	0	0	-4.16	-5.29
WP6	0	0	0	3.9	0	-4.16	0	0
Goal	0	0	0	0	0	0	0	0

individual

	gene 1	gene 2
nodes	start	WP5
segment length	0	-6.15

	Start	WP1	WP2	WP3	WP4	WP5	WP6	Goal
Start								
WP1	0	0	2.55	0	0		0	0
WP2	0	2.55	0	2.38	-4.63		0	-2.04
WP3	0	0	2.38	0	2.58		3.9	-1.46
WP4	0	0	-4.63	2.58	0		0	-2.86
WP5	0	3.08	0	-4.42	0		-4.16	-5.29
WP6	0	0	0	3.9	0		0	0
Goal	0	0	0	0	0		0	0

(a) the first gene is the start point and the second gene has been randomly selected as WP5

	Start	WP1	WP2	WP3	WP4	WP5	WP6	Goal
Start								
WP1	0	0	2.55	0	0		0	0
WP2	0	2.55	0	2.38	-4.63		0	-2.04
WP3	0	0	2.38	0	2.58		3.9	-1.46
WP4	0	0	-4.63	2.58	0		0	-2.86
WP5	0	3.08	0	-4.42	0		-4.16	-5.29
WP6	0	0	0	3.9	0		0	0
Goal	0	0	0	0	0		0	0

individual

	gene 1	gene 2	gene 3
nodes	start	WP5	WP3
segment length	0	-6.15	-4.42

	Start	WP1	WP2	WP3	WP4	WP5	WP6	Goal
Start								
WP1	0	0	2.55		0		0	0
WP2	0	2.55	0		-4.63		0	-2.04
WP3	0	0	2.38		2.58		3.9	-1.46
WP4	0	0	-4.63		0		0	-2.86
WP5								
WP6	0	0	0		0		0	0
Goal	0	0	0		0		0	0

(b) from the row for WP5, WP3 has been randomly selected

	Start	WP1	WP2	WP3	WP4	WP5	WP6	Goal
Start								
WP1	0	0	2.55		0		0	0
WP2	0	2.55	0		-4.63		0	-2.04
WP3	0	0	2.38		2.58		3.9	-1.46
WP4	0	0	-4.63		0		0	-2.86
WP5								
WP6	0	0	0		0		0	0
Goal	0	0	0		0		0	0

individual

	gene 1	gene 2	gene 3	gene 4
nodes	start	WP5	WP3	goal
segment length	0	-6.15	-4.42	-1.46

	Start	WP1	WP2	WP3	WP4	WP5	WP6	Goal
Start								
WP1	0	0	2.55		0		0	
WP2	0	2.55	0		-4.63		0	
WP3								
WP4	0	0	-4.63		0		0	
WP5								
WP6	0	0	0		0		0	
Goal	0	0	0		0		0	

(c) finally, the goals have been randomly selected

Figure 8.7 An example of the random generation of an individual using the cost matrix in Figure 8.4, corresponding to the environment illustrated in Figure 8.5.

8.4.3 Genetic operators

Two operators are defined, namely *crossover* and *insertion*; the crossover operator is extensively applied whilst the insertion operator is occasionally used.

A conventional one-point *crossover* operator is used to exchange the genes between a pair of individuals. The random selection for the crossover site is constrained to the nodes that are in common between the pair of individuals. The offspring generated competes with their most similar parent for survival into the next generation. The definitions of the similarity and the replacement strategy were described in detail in section 6.3.5. One-point crossover has been used, rather than the multi-point crossover used in the previous systems (see chapter 6), because the graph structures are largely different. In the waypoint network presented in chapter 6, if all waypoints

have one branch input and two branch outputs after the waypoint has been explored, the number of the common nodes between a pair of parents should be one. However, in most situations, waypoints may originate from more than one ancestor waypoint in order to converge to the single goal point. Therefore, multi-point crossover is efficient in such situations in that more information can be interchanged between the parents. In contrast, each waypoint in the graph generated for the navigation problem in this chapter have multiple input and output waypoints. This implies a significantly increased probability of potential paths around single waypoints that need to be exploited. Furthermore, the better efficiency of one-point crossover with respect to multi-point crossover was observed during the experiments.

The second operator, *insertion*, introduces a new individual by the mechanism used to generate the initial population, thereby promoting population diversity. The offspring produced replaces the worst fitting member of the current population. Sections 6.3.2 and 6.3.5 provide a full discussion of this operator.

8.4.4 Evaluation

The evaluation function is a weighted sum as expressed in Equation 8.3 and has been formulated to account for both the known and unknown path segments.

$$E = w_1 \cdot \sum_{i=1}^{m} L_i + w_2 \cdot \sum_{j=1}^{n} L_j \qquad \text{Equation 8.3}$$

For the known segments, the sum of the m known paths lengths is multiplied by a weight w_1 and the sum of the n straight-line unknown path segments are multiplied by a weight w_2. The weights w_1 and w_2 can be set to pre-determined values according to whether the robot is required to give priority to exploration or to follow the known path segments where possible. In the current implementation, w_1 is simply set as 1, whereas w_2 is the ratio of the estimated actual path length to the straight line path. A value for the ratio can be obtained from the navigation tasks, in section 8.5.4. Pareto domination (Mooney and Winstanley 2006) have been used for multicriteria optimisation problems, but the determination of their domination requires extra computation effort. Garrozi and Araujo (2006), combined three separate objective functions into a single nonlinear evaluation function, but such a single function is

difficult to determine. In the navigation system presented here, a linear combination of known and unknown path segments is used and the different emphases placed on each part can easily be adjusted through the changes on their weight values.

8.5 Experiments and results

The navigation system proposed in this chapter has been verified through a series of five experiments conducted in simulated environments. As the architecture of the proposed navigation system is inherited from the navigation systems presented in previous two chapters and the comparison with other architectures reported in the literature can be found in the previous two chapters, this section only presents the result of the comparison between the generalised waypoint-based navigation algorithm and other two algorithms (Ahn and Ramakrishna 2002; Ji, Iwamura and Shao 2007). The reasons for choosing those two benchmarks are explained in section 8.5.6.

The experimental arrangement is same as that described in chapter 6, but with modifications to the autonomous mobile robotics toolbox (Brno University of Technology 2006) to render it compatible with version 7.3 of MATLAB (Mathworks 2006). Three investigations were carried out in the first environment, which was used to produce a range of different levels of environmental knowledge for use by the planner. A second environment was used to conduct the final two experiments: experiments 4 and 5 investigated the effects of the weight w_2 on the plan produced. Experiments 3, 4 and 5 employ a learning process to enable the robot to become familiar with its environment. Table 8.1 summarises the parameters used; these were determined experimentally and have not been optimised. The weight, w_1, is applied to the sum of the costs of the path segments already visited and is set to unity as their path length values are accurately known. The costs for the unknown path are scaled by w_2 which is obtained based on previous navigational experiences.

Table 8.1 The system parameters for the planning algorithm (w_2 is estimated from the previous navigations).

experiment	population size	number of generations	weights		operator probability	
			w_1	w_2	*insertion*	*crossover*
1	250	100	1	1.12	every 100 generation	other generations rather than every 100 generation
2	300	2500	1	1.12		
3	500	4500	1	1.12		
4	500	4800	1	1.85		
5	500	4800	1	1		

8.5.1 Path generated with little knowledge of the environment

The first test is a simple navigation problem where limited information of the environment is known from a previous navigation task that generated fully explored waypoints when moving from the lower-left corner to upper-right corner of the environment. Figure 8.8 shows as thin lines the trajectories that the robot previously followed and the thickened line indicates the path segments for a new navigation problem that consists of two segments through unknown territory and a known segment in the central part of the path.

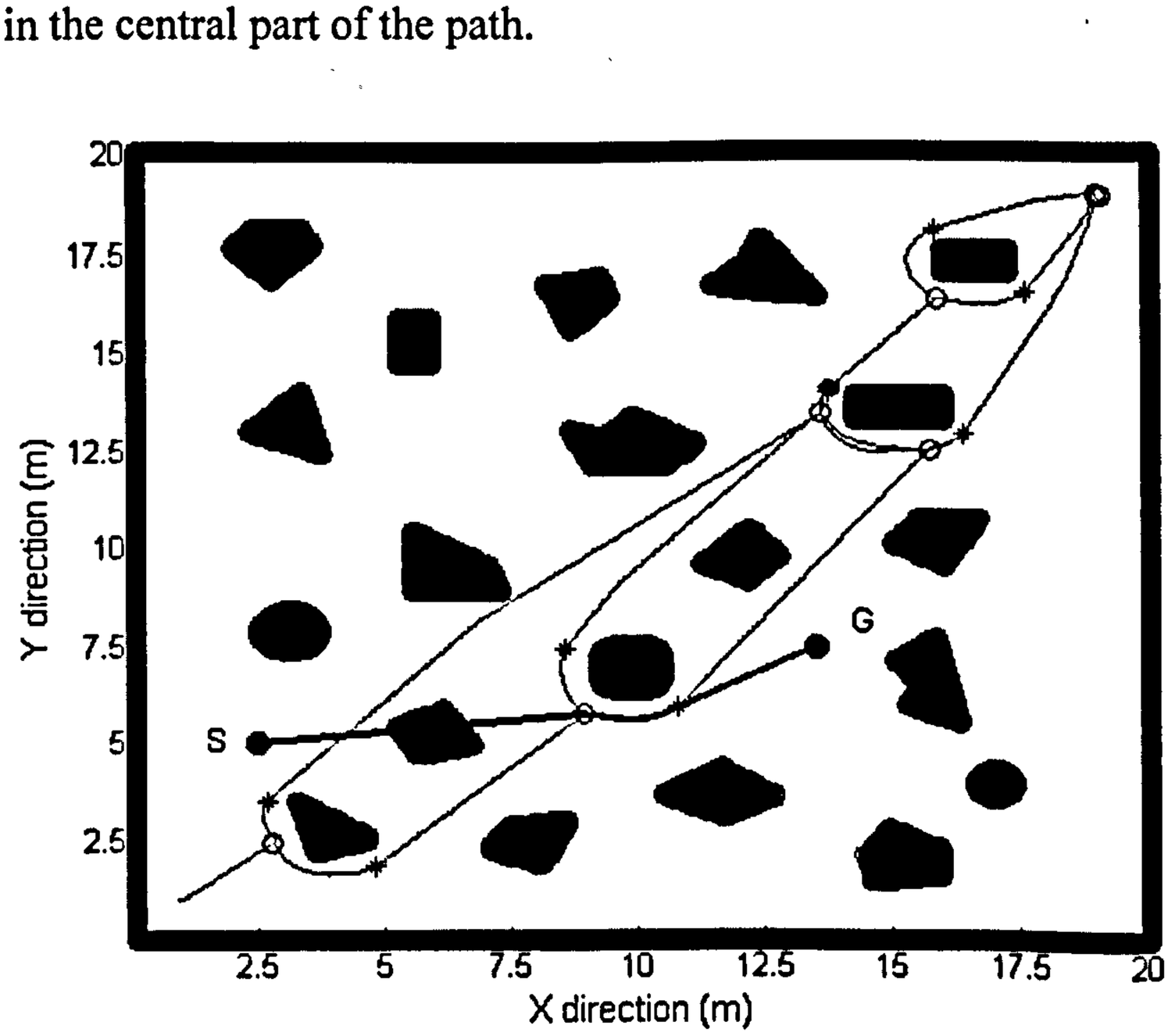


Figure 8.8 A path generated based on knowledge acquired during previous navigations. The thickened and thin lines respectively represent paths generated by the deliberative planner and that travelled by the robot during previous navigations. The new start and goal points are indicated by the markers 'S' and 'G'.

To illustrate the operations of the planner, Figure 8.9 shows a thickened line that represents splines constructed to approximate the path segments around obstacles. Note that the spline fit is constrained to approximate the actual path segment with a maximum error of 5cm, which is the step length of the robot.

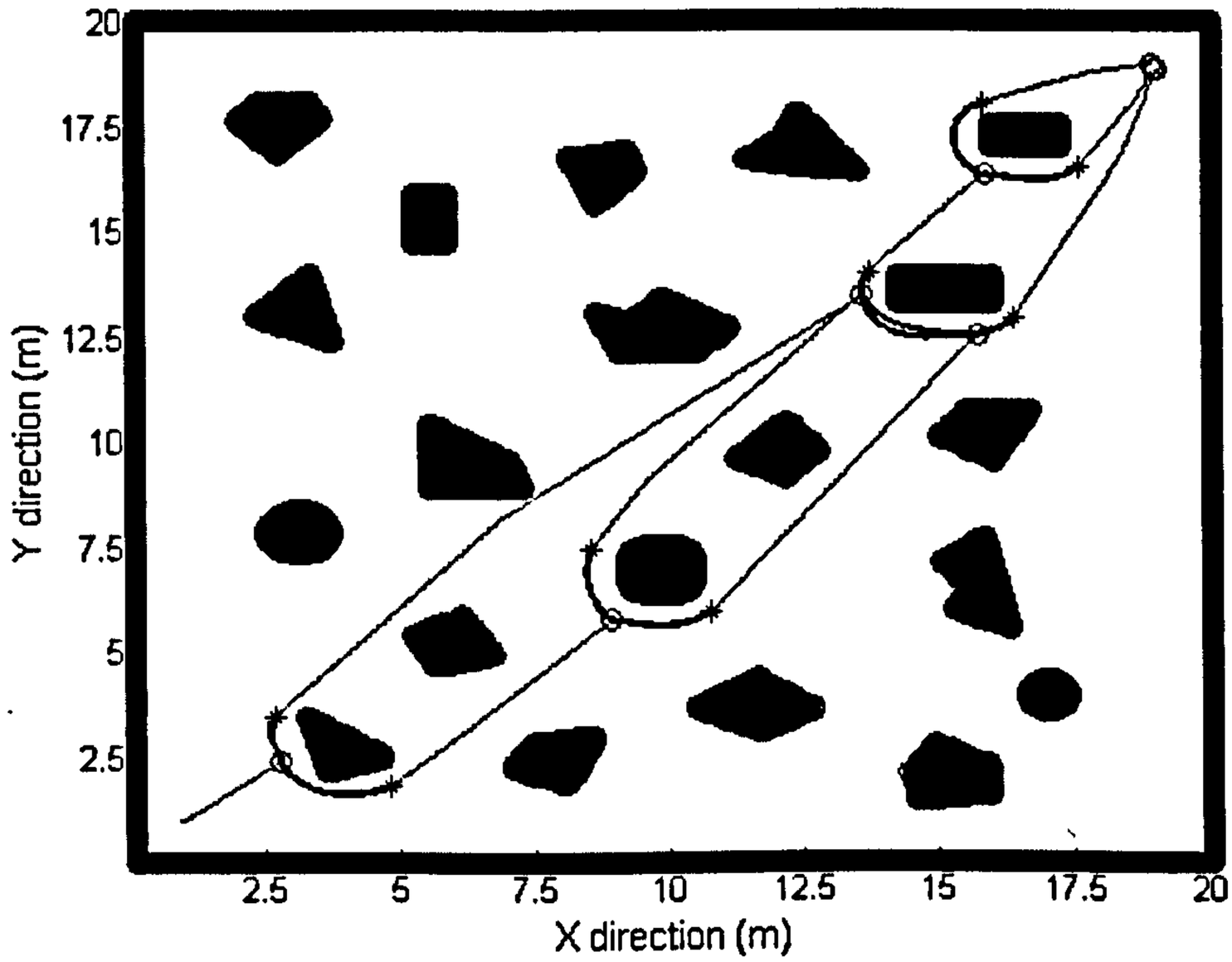


Figure 8.9 The spline approximations to the actual path segments around the obstacles. The spline curves are indicated by the thickened lines, whereas the thin lines present the path that the robot had taken previously.

Figure 8.10 shows the cost matrix generated for the new navigation task shown in Figure 8.8 (from S to G) and all the paths in the cost matrix are shown in Figure 8.11. The negative values in Figure 8.10 correspond to the costs estimated for proposed path segments that have not been followed in a previous robot activities and are shown as thin lines in Figure 8.11.

	Start	WP1	WP2	WP3	WP4	WP5	WP6	WP7	WP8	WP9	WP10	WP11	WP12	WP13	Goal
Start	0	0	-1.5	-14	-14.4	-17.5	0	-3.9	-6.5	-8.3	-15.2	0	-18.8	-6.5	0
WP1	0	0	1.1	0	0	0	0	2.4	0	0	0	0	0	0	0
WP2	0	1.1	0	14.9	-15.3	-18.4	0	0	-6.7	-8.5	-15.9	0	-19.7	-7.1	0
WP3	0	0	14.9	0	0.6	0	0	0	0	-8.1	0	3.4	0	8	-6
WP4	0	0	-15.3	0.6	0	3.2	-4.6	0	0	-8.6	3.4	0	0	0	0
WP5	0	0	-18.4	0	3.2	0	1.9	0	0	0	0	-3.4	2.2	0	0
WP6	0	0	0	0	-4.6	1.9	0	0	0	0	0	-3.8	0	0	0
WP7	0	2.4	0	0	0	0	0	0	5.7	-7.3	0	0	0	-6.8	-10.4
WP8	0	0	-6.7	0	0	0	0	5.7	0	2	0	0	0	1.8	0
WP9	0	0	-8.5	-8.1	-8.6	0	0	-7.3	2	0	8.3	0	0	0	-3.2
WP10	0	0	-15.9	0	3.4	0	0	0	0	8.3	0	0.8	0	-8.8	-5.5
WP11	0	0	0	3.4	0	-3.4	-3.8	0	0	0	0.8	0	0	0	0
WP12	0	0	-19.7	0	0	2.2	0	0	0	0	0	0	0	0	0
WP13	0	0	-7.1	8	0	0	0	-6.8	1.8	0	-8.8	0	0	0	-4.9
Goal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8.10 The cost matrix constructed for path planning. The positive values represent the paths travelled by the robot during previous navigation tasks, whereas the estimated paths are indicated by a negative sign.

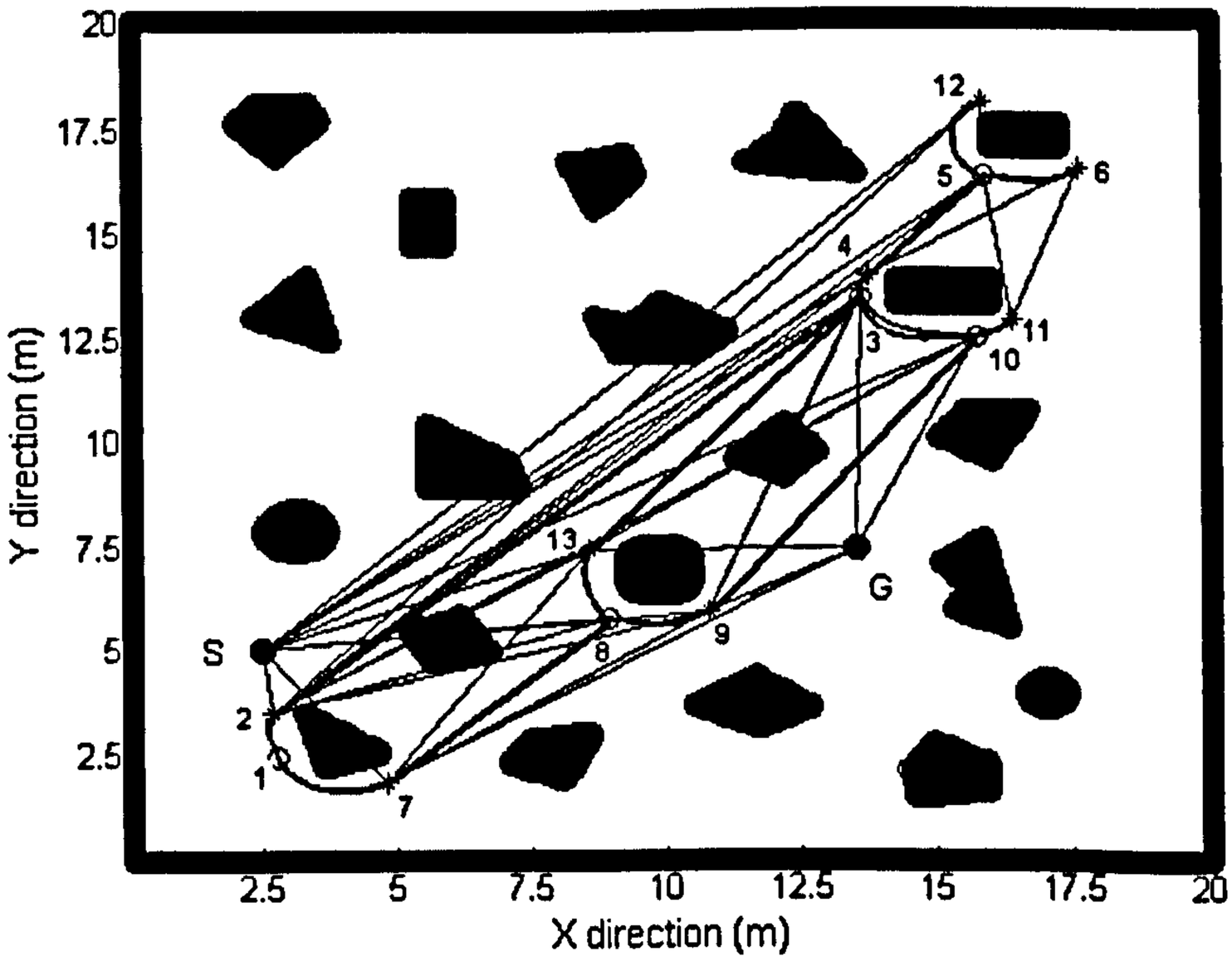


Figure 8.11 The visibility graph corresponding to the cost matrix shown in Figure 8.10. While the thin lines indicate the paths that have not been followed by the robot, the paths that the robot has followed in previous navigations are shown by thickened lines.

Figure 8.11 shows the visibility graph corresponding to the cost matrix constructed for a new navigation task that starts at point S and ends at point G. A small number of path segments (indicated by thickened lines in Figure 8.11) have been followed by the robot in the previous tasks and a larger number of path segments (shown as thin lines) are previously unfollowed paths between waypoints from which the robot can also choose to generate navigation solutions.

8.5.2 Path generated with greater confidence of the working environment

The planning result presented in this section was obtained after the robot has performed a number of different navigation activities in the same environment and consequently much more domain information was available for planning purposes. Navigation performed from four different pairs of start and goal points are illustrated as thin lines in Figure 8.12 and the thickened line is the path generated for the new start and goal points. From Figure 8.12, it can be seen that no known segments are used by the best path generated.

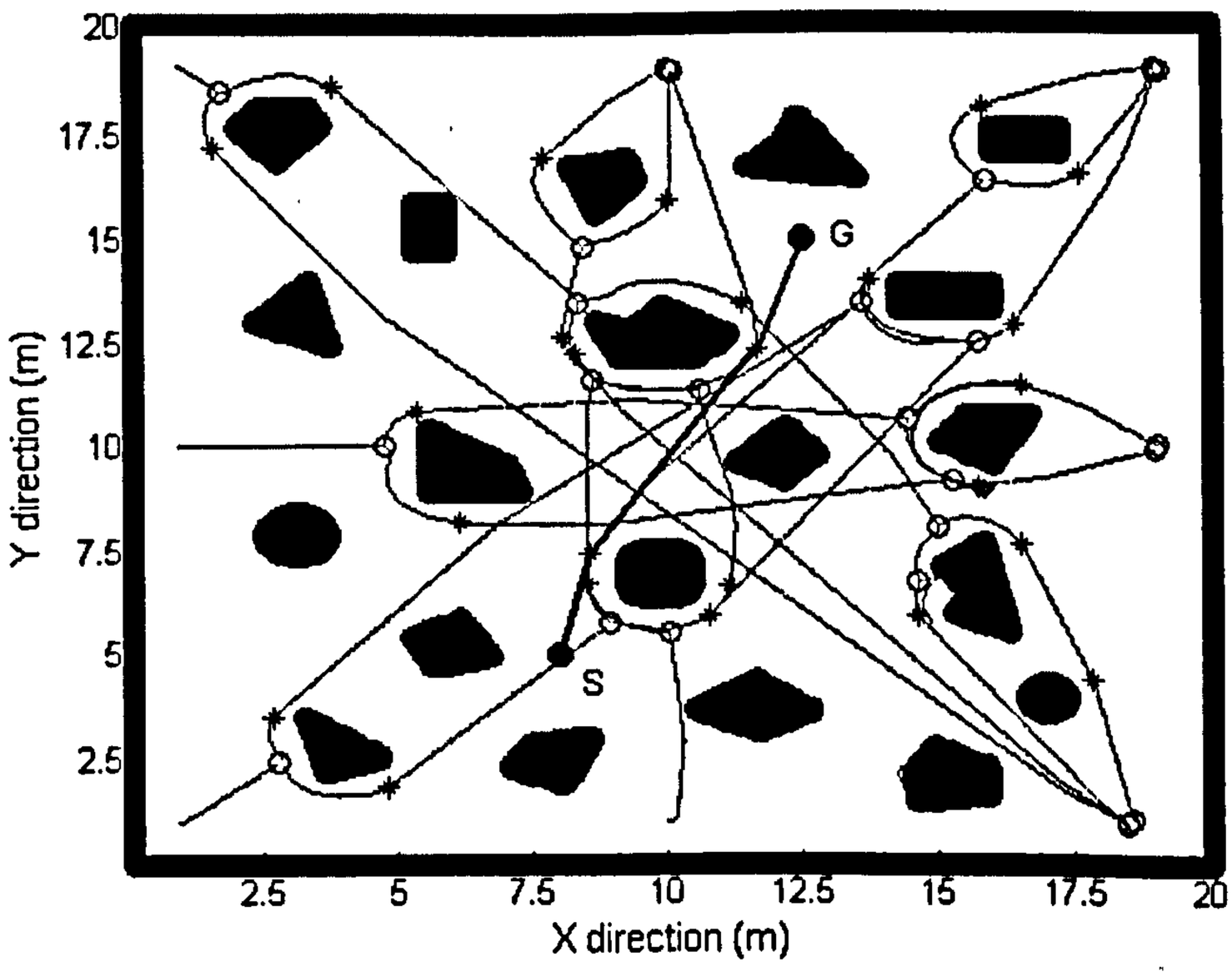


Figure 8.12 The path generated based on the environmental knowledge gained from four earlier navigation tasks. The markers 'S' and 'G' are the start and goal points for the current navigation activities.

The thickened lines in Figure 8.13 are the approximations achieved by a set of splines fitted to the path segments generated when avoiding obstacles. The cost matrix generated for the new start and goal points is shown graphically in Figure 8.14.

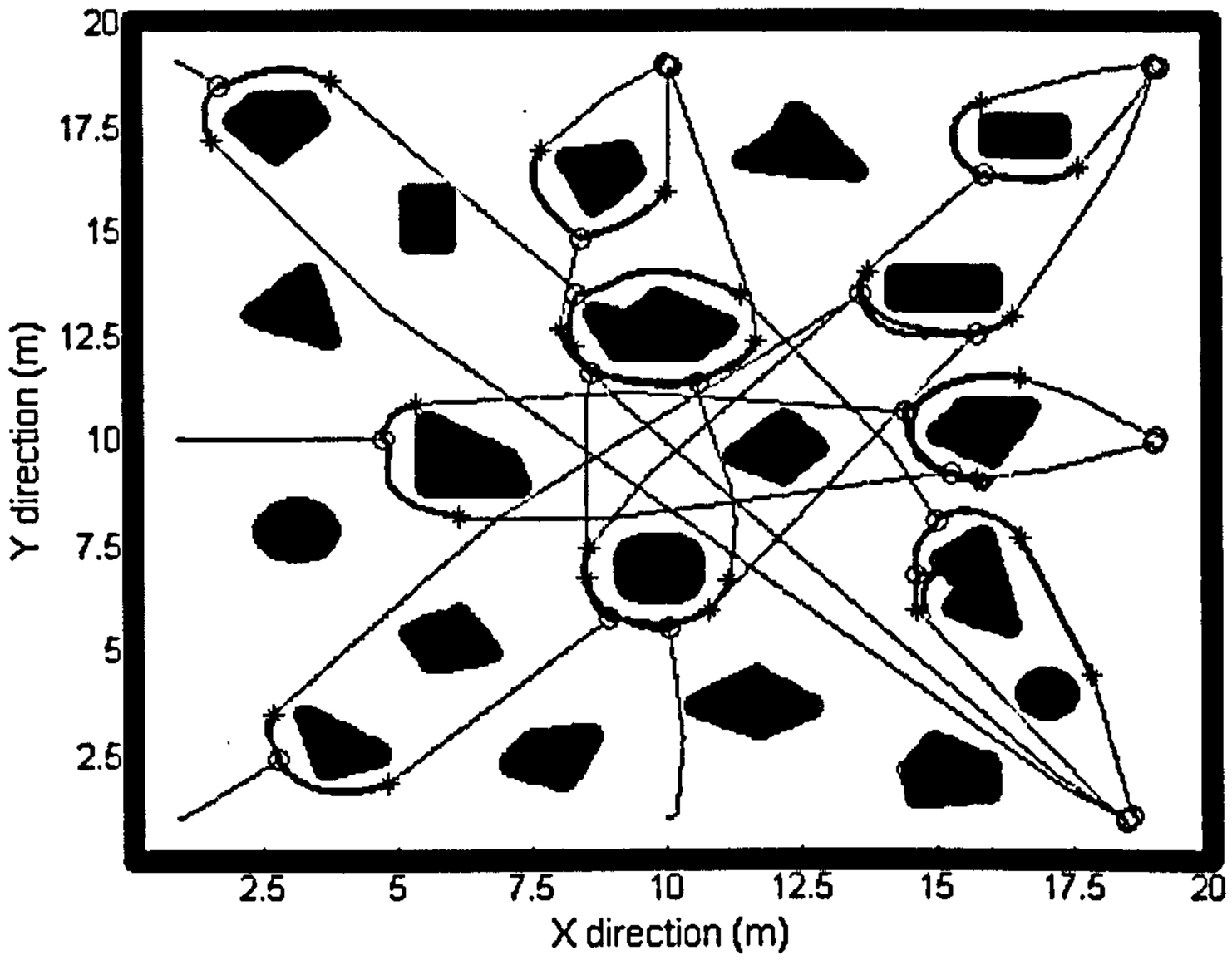


Figure 8.13 The spline curves representing the actual path segments. The thickened lines indicate the fitted curves and the thin lines the paths previously travelled.

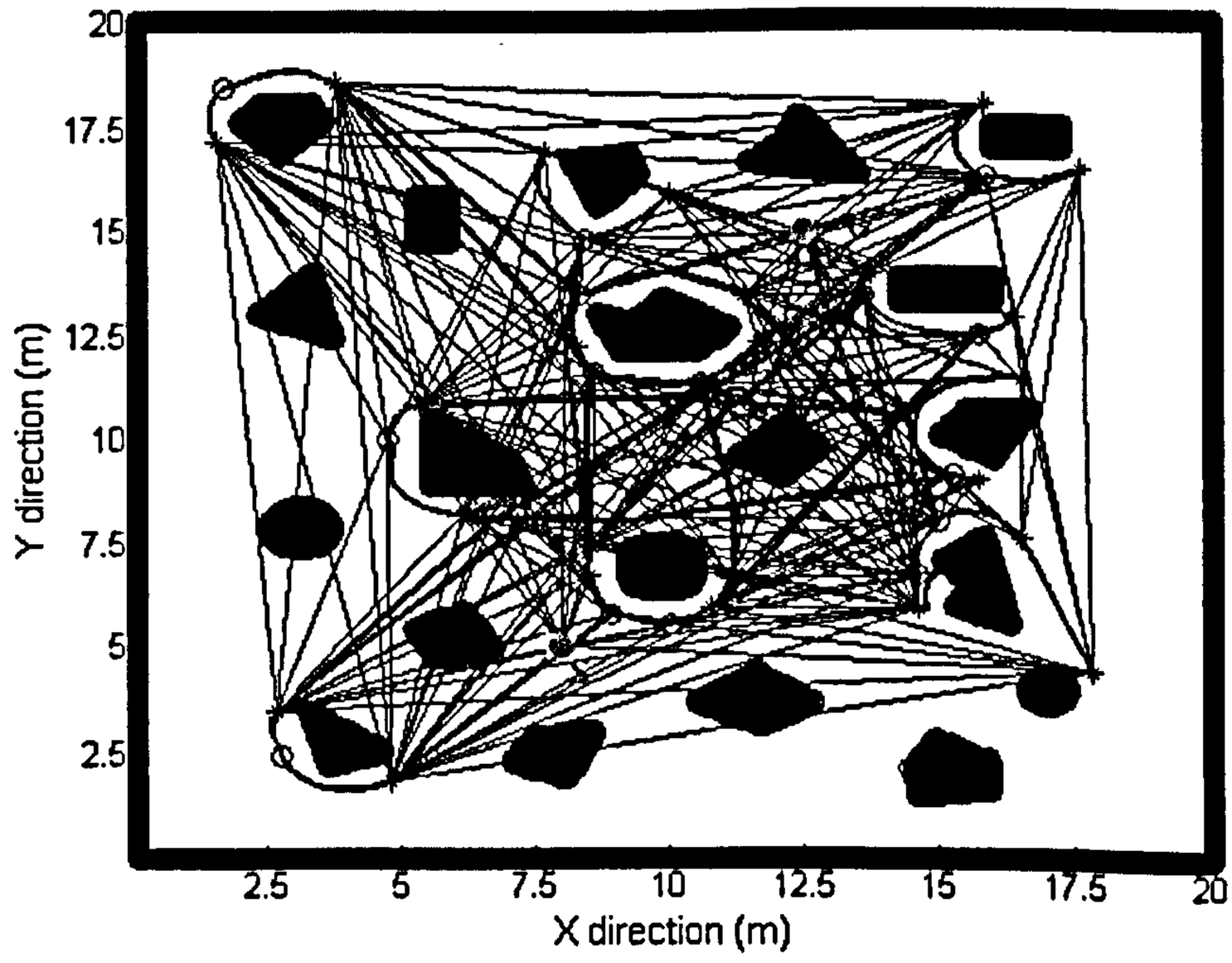


Figure 8.14 The visibility graph corresponding to the cost matrix generated. The segments unvisited are indicated by thin lines, whereas the thickened lines represent segments that have been discovered.

8.5.3 The paths generated based on learning

The planning performed by the proposed algorithm was further evaluated by increasing the search space. This was realised by a learning process that repeatedly performed navigation between pairs of randomly selected locations, each of which must be in free space and at a distance no less than the sensor range from any obstacle. The thin lines in Figure 8.15 are the trajectories of these navigation tasks and the thickened line is the path planned for a new navigation task. It can be seen from the result obtained that a path can be generated with minimum cost in terms of path length, though all path segments have not been explored in the previous navigation tasks.

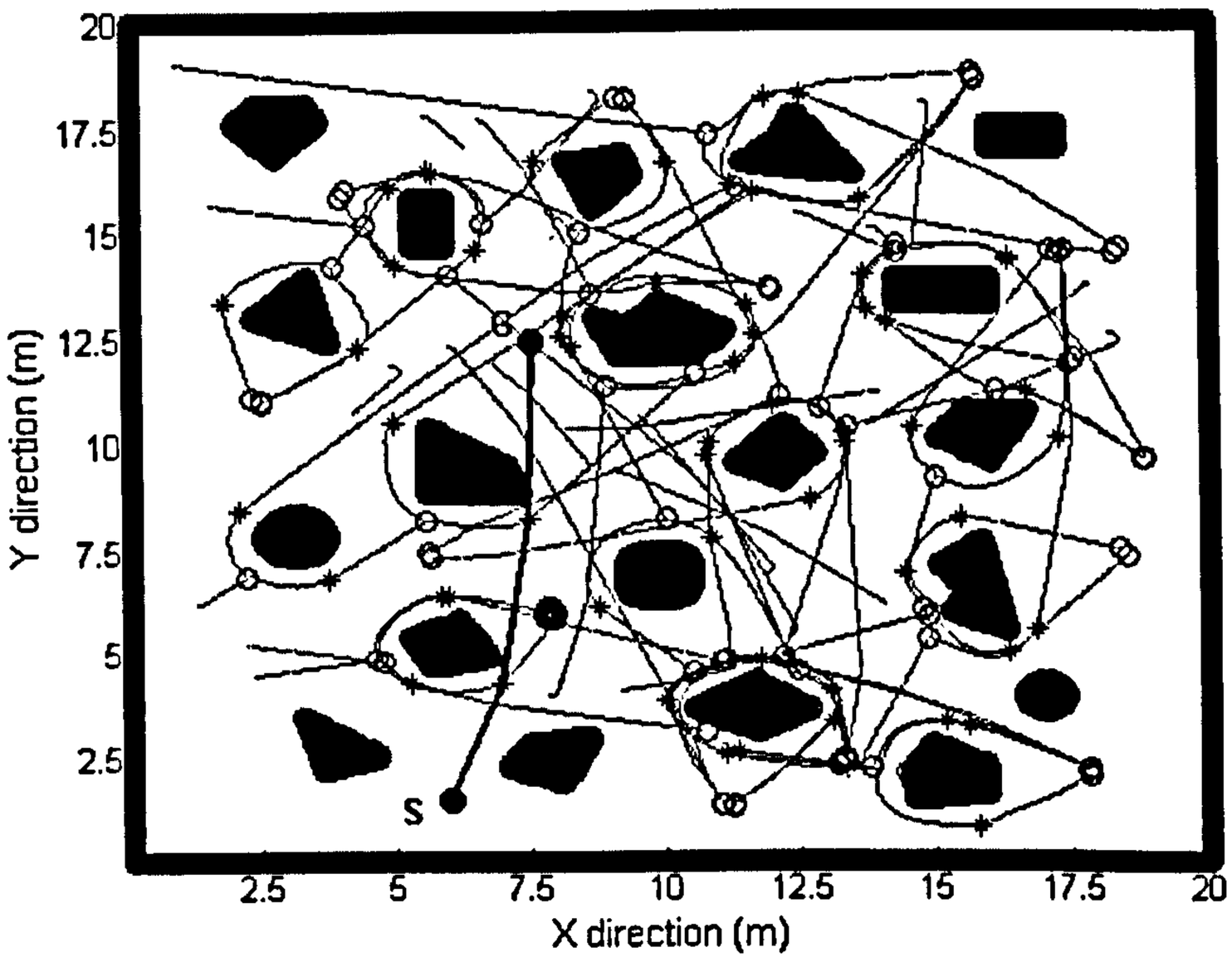


Figure 8.15 The path generated following a sequence of learning in the environment. The thickened line is the path generated for a planned navigation task from 'S' to 'G' and thin lines indicate the paths that have been travelled previously.

8.5.4 The effect of weight values on path planning

The final two experiments investigated the effect of the weight values for unknown path lengths for use in planning. The cost for the unknown part of the planned path is the length of the straight line segments scaled by this ratio. The value of w_2 is

effectively estimated from the previous navigations through the same environment by comparing the true path length to the length of the path assumed as a straight line. This value is likely to be larger when the environment is heavily populated with obstacles. The weight w_2 is the ratio of the actual path length taken to that if a straight-line path could have been followed, and in the current work is estimated from previous navigation tasks conducted between randomly selected pairs of locations. To demonstrate the effect of the weight w_2 on the path planned, two experiments are now described. In the first experiment, the value of w_2 is set to 1.85 and the path produced for a new navigation task is shown in Figure 8.16 as a thickened line. Apart from the first and final segments, all the remaining segments have been previously taken by the robot. In contrast, the path planned in Figure 8.17 when w_2 is set to unity (so that unknown path lengths are taken to be the straight-line costs) tends to follow previously untaken paths as these are not penalised. In contrast, the path shown in Figure 8.16 as the thickened line was generated with a different weight w_2 being 1.85 estimated from the previous navigations for the unvisited part of the path.

It is apparent that high values of w_2 tend to lead the planner to prefer previously taken paths, whereas low values of w_2 promote the following of unknown paths. As high values of w_2 reflect an environment that has a large proportion of the environment filled by obstacles, known paths are more likely to be more attractive to the planner as they guarantee obstacle avoidance. The current system includes a learning process that uses previous navigations to increase the confidence of the value of w_2. The accuracy of the estimated value can be further improved as more explorations are performed in the same environment.

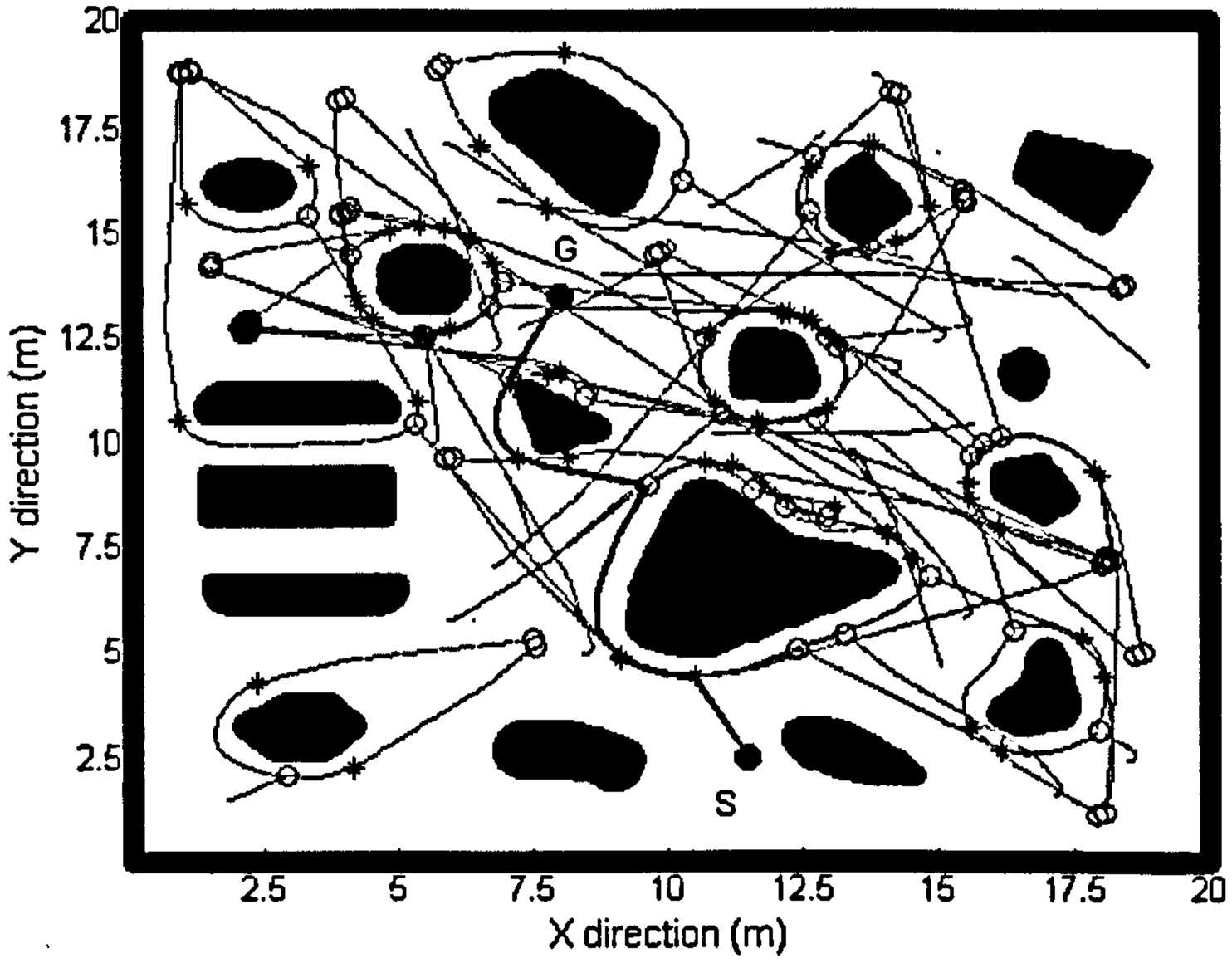


Figure 8.16 The path generated using a weight value $w_2 = 1.85$.

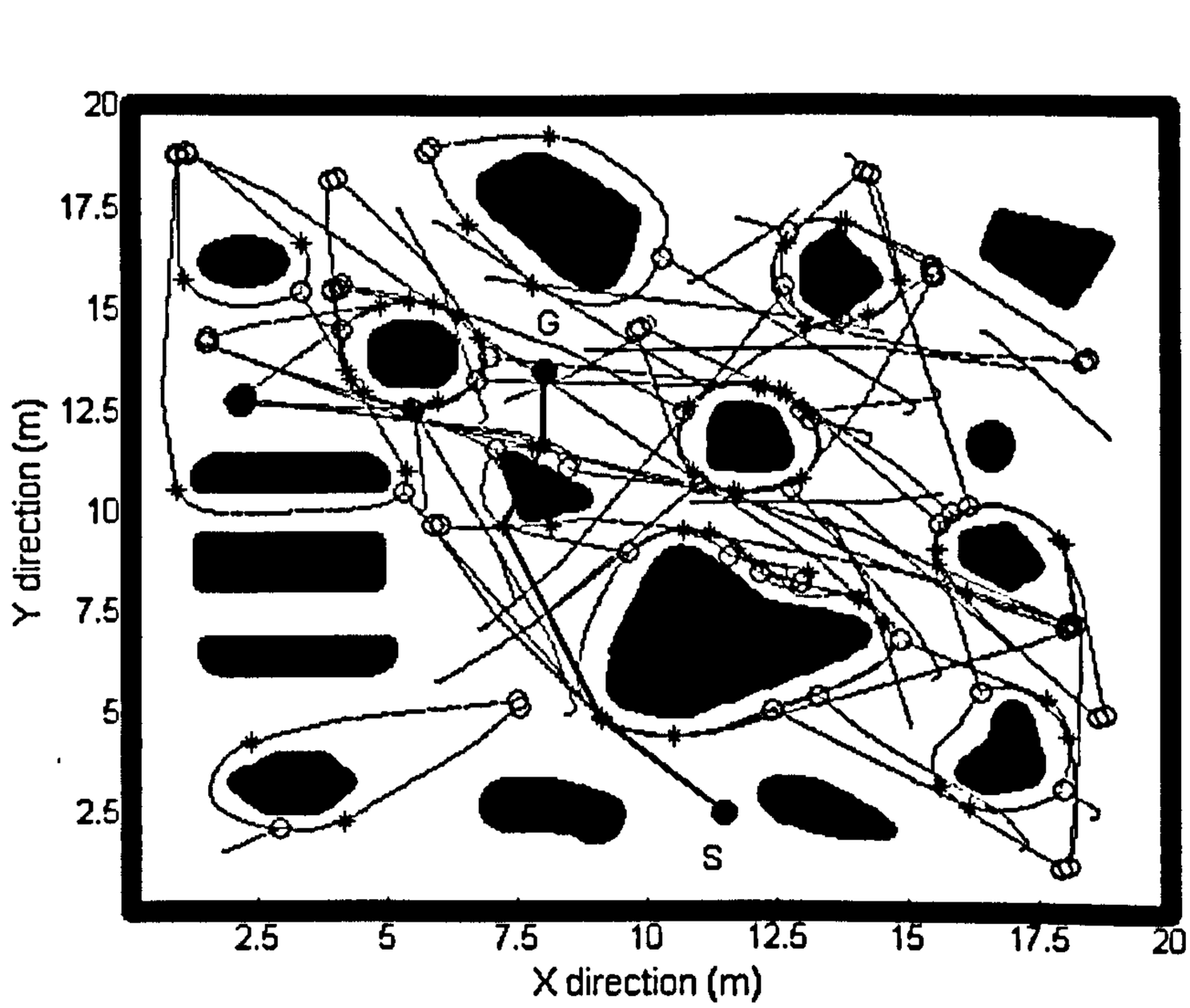


Figure 8.17 The path generated using a weight value $w_2 = 1$.

8.5.5 Summary of results

Table 8.2 summarises the planning performances for each experiment. The number of waypoints and splines produced by the experiments reflects the complexity of the task that needs to be performed by the path planner. The use of splines rather than the actual recorded data has allowed memory usage to be reduced by around an order of magnitude. The memory usage in the spline fit could be further reduced by relaxation of the error bound in the approximation. The planning performance in terms of optimality (defined as the percentage of attempts that successfully evolved to the optimal path) was evaluated by performing 300 independent runs for each experiment. The calculation times shown in Table 8.2 were obtained from the mean values of 300 independent runs rather than median values, as there was little presence of extreme values. For a reasonably complex planning problem, such as those in experiments 3, 4, and 5, a suitable path can be generated in well under 10s. Although this is achieved by a large population over a large number of generations (as shown in Table 8.1), the number of fitness evaluations that need to be performed is considerably smaller than the number of candidate paths. Even the simplest planning problem shown in Table 8.2 contains a large number of possible paths to search. To determine the total number of alternative paths that can be generated from the cost matrix, a depth-first algorithm was employed. As a steady-state GA is adopted for the planner, only a single genetic operation is performed in each generation and consequently a greater number of generations is required compared with conventional GAs. The steady-state GA, however, is prone to retaining potential elites until their genes have possibly been propagated to later generations. During experiments, it was observed that the insertion operator, compared with crossover, was responsible for fewer improvements in the fitness of the population, even though it promoted population diversity to some degree. The use of DC was found to promote the retention of the initial diversity and the initial supply was observed to be more critical in progressively evolving an optimal solution compared with the results of applying the insertion operation. It was found that the frequency of application of the insertion operator could be reduced, firstly as this would generally lengthen the survival rate of potential elite genes until they had been more closely scrutinised, and secondly because its importance in

maintaining diversity was limited. Note that insertion operator was not applied to the simplest environment, as the initial supply of genes is adequate.

Table 8.2 The planning performances for the five experiments, showing the memory needed to store the splines. The compression ratio is the mean value of the ratio of the memory occupied by the spline coefficients to that occupied by the raw coordinate data. The results for the calculation time and optimality were obtained for 300 independent runs and the mean values are shown.

experiment	number of waypoints	number of splines	memory usage (kB)	compression ratio (%)	optimality rate (%)	calculation time (s)
1	13	10	12.1	8.17	93.0	0.13
2	41	33	41.6	10.6	91.6	2.34
3	90	73	108	11.3	93.7	4.72
4	92	68	93.7	12.2	92.7	6.05
5	92	68	93.7	12.2	97.0	6.03

8.5.6 Comparison of the planning algorithm with existing systems

To further evaluate the planning algorithm proposed in the navigation system, a comparison with two existing genetic-based algorithms (Ahn and Ramakrishna 2002; Ji, Iwamura and Shao 2007) was conducted for the five experiments introduced previously in this section. These two algorithms are closely related to the planning algorithm proposed in this chapter, since only two genetic operators, one is used for exchanging information between a pair of parents and the other for randomly altering genes, were used. Ahn and Ramakrishna (2002) proposed a generational GA and a number of other authors (Davies and Lingras 2003; Mooney and Winstanley 2006; Wu and Ruan 2004) recently reported similar approaches but with minor modifications. Ahn and Ramakrishna (2002) performed a comparison to Dijkstra's algorithm and the results obtained indicates their GA outperformed Dijkstra in terms of calculation time and, moreover, the computation time required by their GA does not increased significantly with problem size, in contrast with Dijkstra's algorithm. The algorithm developed by Ji, Iwamura and Shao (2007) uses only a fraction of the population in the genetic operations for each generation.

Tables 8.3 and 8.4 summarise the system parameters used for the two algorithms (denoted as Ahn's GA and Ji's GA) respectively. In order to produce a fair comparison, the population size and weights for both the known and unknown parts

were assigned the same values, and the probabilities of the genetic operators used the values documented in the literature. The probabilities of crossover and mutation listed in Table 8.3 indicates Ahn's GA is a crossover-intensive algorithm, but Ji's GA used the two genetic operators with equal probability. Note that the source code for Ahn's GA was downloaded from the author's website (Ahn 2007), whereas Ji's GA was translated from the paper (Ji, Iwamura and Shao 2007).

Table 8.3 The system parameters for Ahn's algorithm (w_2 is estimated from the previous navigations).

experiment	population size	weights		operator probability	
		w_1	w_2	*crossover*	*mutation*
1	250	1	1.12	1	0.05
2	300	1	1.12		
3	500	1	1.12		
4	500	1	1.85		
5	500	1	1		

Table 8.4 The system parameters for Ji's algorithm (w_2 is estimated from the previous navigations).

experiment	population size	weights		operator probability	
		w_1	w_2	*crossover*	*mutation*
1	250	1	1.12	0.2	0.2
2	300	1	1.12		
3	500	1	1.12		
4	500	1	1.85		
5	500	1	1		

Two measures, optimality (measured as the percentage of attempts that successfully evolved to the optimal path) and the number of fitness evaluations, were performed in a series of five experiments used previously in this section. The results shown in Table 8.5 were obtained for 300 independent runs when the evolution terminated after the calculation time listed in Table 8.2, as our aim was to maximise the performance (in terms of optimality) with minimum calculation time. Cantu-Paz (2000) suggested to use the number of fitness function evolutions as a criterion of convergence performance when compared with other algorithms until equal quality of solutions is obtained. However, it is difficult to obtain an identical solution quality if the high quality is required, and low quality may indicate the convergence performance for

partial evolution and the later convergence towards high optimality may never be explored. Consequently, the comparison strategy adopted is instead to constrain the evolutions so that they terminate over a specified execution time.

It can be concluded from Table 8.5 that Ahn's and Ji's algorithms slightly outperformed the waypoint-based planning algorithm in terms of optimality for the simplest test, but high optimality was achieved by the waypoint-based planning algorithm for the remaining tests. That the most rapid convergence of the waypoint-based planning algorithm was achieved with a relatively smaller number of fitness evaluation functions for all tests implies that DC is important in ensuring good convergence. Ahn's and Ji's algorithms have a similar performance in the five test problems and are capable of generating optimal solution more frequently for the simple planning problem (experiment 1). The smallest number of fitness evaluations was required by the waypoint-based planning algorithm indicating more time was consumed in similarity evaluations between offspring and parents.

Table 8.5 Comparison of the performances of the planning algorithms. Optimality was obtained for 300 independent runs and the number of fitness evaluations are estimated according the population size and operator probabilities.

experiment		1	2	3	4	5
Ahn's GA	optimality rate (%)	94.8	66.7	52.6	37.8	56.2
	fitness evaluations	729.7	20608	18216	10296	10476
Ji's GA	optimality rate (%)	96	57.3	43	45.7	67.7
	fitness evaluations	701.3	6446	11573	12058	12094
waypoint planner	optimality rate (%)	93	91.6	93.7	92.7	97
	fitness evaluations	450	5275	9455	10052	10052

8.6 Conclusions

This chapter has described a generalised waypoint navigation system that has extended the waypoint navigation system introduced in chapters 5 and 6, so that it is able to plan navigation tasks that can start and end at any point in the robot's environment. To achieve this objective, additional information regarding the paths taken to avoid obstacles is recorded and stored in a compact manner. A suitable path is generated by the generalised waypoint algorithm operating on the search space

represented by a cost matrix which is constructed taking into account the new start and goal points. Statistical evaluation of the planning performance implies the robustness of the proposed algorithm in finding an optimal solution within a short time even for reasonability complex problems. The planning performance required to obtain the optimal solutions were evaluated by comparing its performance with that of two other methods reported in the literature.

The current navigation system has a number of shortcomings. From Figure 8.15, 8.16 and 8.17, it can be seen that a number of spline curves determined for in the same obstacle overlap to a certain degree. At present, no method of combining this information has been developed and each future investigation of the same area will result in the storage of duplicated information. This will give rise to scaling problems, as additional waypoints will continue to be generated even when no additional useful information is being acquired. The scaling problem can be addressed by the implementation of a suitable localisation technique. Scaling issues, when constructing the cost matrix, will arise as more waypoints are discovered.

Chapter 9

CONCLUSIONS

The research project work presented in the thesis is summarised and the objectives for the research are critically reviewed. Future investigations are suggested to overcome some of the shortcomings of the navigation systems.

9.1 Summary

The broad concepts relating to the type of robots studied in the project, the robot navigation and genetic algorithms were introduced in chapter 1. The research objectives were stated and the contributions of the project to knowledge listed.

Chapter 2 discussed the recent development in GAs research, including descriptions of steady-state GAs, genetic representations, selection schemes, genetic operators, and premature convergence and diversity. These features of GAs are closely related to those adopted in the planning algorithms in this thesis.

Research results published in the literature that are relevant to the current project were reviewed in chapter 3. This included an introduction to the deliberative, reactive and hybrid architecture and described work related to the various types of planners, particularly those based on evolutionary techniques. The reactive component of the navigation system used in the work described in chapters 6 to 8 was also introduced.

Chapter 4 proposed a genetic-based approach using vertex heuristics for mobile robot path planning and presented the results for a set of simulated environments. In comparison with an earlier genetic-based planner (EP/N), the path planning performance was similar, but the calculation time was considerably reduced. The results demonstrated that the performance can be further improved if the system parameters are optimised.

The vertex++ planner introduced in chapter 5 is an extension of the vertex planner and was designed to deal with the planning problems for dynamic environments. The advantages and drawbacks of the vertex++ planner observed from the experimental results are as follows. Due to the reduction in search space and simple genetic operators, the planner can rapidly establish an optimal or near optimal path for environments containing multiple obstacles. New information observed regarding changes in the environment can be incorporated into the planning process to modify the current path and avoid potential collisions. Unfortunately, such planner-based navigation systems are generally unable to deliver a satisfactory real-time solution when unexpected changes occur.

The new navigation system presented in chapter 6 adopted a hybrid architecture with the waypoints representing the robot's environment in attempt to combine the advantages and overcome the disadvantages of the reactive and deliberative navigation. The waypoint navigation system required no *a priori* knowledge of the environment, stored the information elicited regarding the environment in highly abstract and compact form and was able to deliver real-time operation by navigating in a reactive manner between waypoints. The experimental results showed that, compared with EP/N and the vertex planner, the calculation time required to generate a path was significantly reduced without compromising on the quality of the path generated. The experiments performed in complex environments indicated that a stable optimality was achieved by the proposed planning algorithm.

Chapter 7 introduced a navigation system for dynamic environments based on waypoint-based navigation system presented in chapter 6. The results showed that a

suitable path can be effectively generated by incorporating statistical knowledge of the dynamic characteristics of the environment gained from the previous navigations. The experiments conducted for the hybrid system verified the proposed policy for avoiding both individual and small groups of moving obstacles, even though the strategy was mainly designed for only single dynamic obstacles. The technique developed for avoiding moving obstacles was further evaluated by comparison with a reactive strategy found in the literature.

The generalised version of the waypoint-based navigation system was presented in chapter 8. In order for the robot to plan future navigation between any start point and any goal point, a new method of representing information obtained regarding the environment was developed and this was recorded in a manner that is not expensive in terms of memory usage or planning time. The chapter obtained path planning results for a series of experiments in environments exhibiting a range of different complexities and a comparison with other two algorithms reported in the literature was conducted to benchmark the performance of the planning algorithm.

9.2 Review of research objectives

The aim of the research, to design an autonomous navigation system for a mobile robot that has no *a priori* knowledge of the environment, was met to an extent by the implementation of the waypoint navigation system in chapter 6, but with a limitation that restricted the choice of start and end points for the navigation. The removal of this limitation was accomplished in chapter 8, where the waypoint method was extended to allow the determination of segments around obstacles that described paths for obstacle avoidance that could be used in later planning activities.

The achievements with respect to the specific objectives outlined in section 1.4 are discussed below.

1. Compared with existing implementations described in the literature, a reduction in the time taken to generate plans for static environments was

achieved by the vertex planner that restricted the search space to only the vertices of the obstacles.

2. The vertex++ planner extended the vertex planner in such a way that the navigation could also be carried out for dynamic obstacles. This was realised by incorporating speed parameters into planning process. The improvement in the calculation time of the planning algorithm can be attributed to the reduction apparent in the search space and application of only simple genetic operators.
3. The waypoint-based navigation system provided a simple mechanism to automatically gather information regarding suitable paths to avoid static obstacles in the environment. This is a hybrid solution, in which the reactive navigator identifies and stores the location at which a new obstacle is first encountered. This location can then be used by a high-level deliberative system for generating future plans.
4. The waypoint navigation system was enhanced by augmenting the behaviours of the reactive component so that specific actions could be instigated on encountering moving obstacles.
5. The generalised version of the hybrid navigation system was able to provide navigation from any start point to any goal point in the environment. This was achieved by acquiring additional information regarding the obstacles and designing suitable paths that could be following to ensure avoidance.

9.3 Shortcomings of the planners and future work

This section discusses some of the drawbacks of the current approaches described in this thesis and outlines future work to overcome these drawbacks.

9.3.1 Planner-based navigation systems

The improvement in the performance of the planners proposed in chapters 3 and 4 apparently results from the reduction in search space that requires only the vertices of obstacles need be considered. However, objects in the test environment that were not polygonal were approximated by bounding polygons. The trade-off between accuracy

of representation and both planning time and plan viability was not investigated, and could be the subject of further study.

9.3.2 Waypoint-based navigation system

Although the waypoint system is clearly an important contribution to mobile robot planning, further development and investigation is needed prior to practical implementation. One of the principal shortcomings of the generalised waypoint navigation system is scalability as the cost matrix is augmented with new waypoints. Environments investigated in the study contained many obstacles and planning can be carried out on a reasonable timescale, but this cannot be guaranteed for larger numbers of obstacles. Practical solutions for limiting the effect of scalability exist, for example, it may be possible to subdivide the waypoints into independent sets (for example those contained in separate rooms) that do not need to be fully interconnected.

No combination of the path segments around the obstacles that are produced in the generalised waypoint system was carried out. In many cases these segments describe largely coincident or intersecting paths, wasting memory for the storage of information and lengthening processing time for future planning. A method to combine waypoints needs to be developed. Such a solution would also alleviate the scalability problems to some extent.

9.3.3 Experimental procedure

The work presented in the thesis was conducted using simulation and implementation on a physical robot is important in demonstrating practical feasibility. This is also an area for future research.

9.4 Conclusions

This chapter has summarised the work presented in this thesis, reviewed the objectives, and proposed the future work based on the outline of possible shortcomings of the navigation approaches introduced in this thesis.

References

Abbas, H.M. and Bayoumi, M.M. 2006. Volterra-system identification using adaptive real-coded genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, **36**(4), 671-684.

Abbass, H.A. and Deb, K. 2003. Searching under multievolutionary pressures. In: *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization, Apr 8-11, 2003, Faro, Portugal*. New York: Springer Publishing Company, 391-404.

Abraham, A., Jain, L. and Goldberg, R. (eds.). 2005. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. London: Springer-Verlag Publisher.

Adams, M., Zhang, S. and Xie, L.H. 2004. Particle filter based outdoor robot localization using natural features extracted from laser scanners. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 26-May 1, 2004, New Orleans, USA*. Los Alamitos, CA: IEEE Computer Society Press, 1493-1498.

Affenzeller, M. and Wagner, S. 2004. The influence of population genetics for the redesign of genetic algorithms. *Journal of Systems Science*, **30**(4), 53-60.

Affenzeller, M. and Wagner, S. 2005. Offspring selection: a new self-adaptive selection scheme for genetic algorithms. In: *Proceedings of the 7th International Conference on Adaptive and Natural Computing Algorithms, Mar 21-23, 2005, Coimbra, Portugal.* Vienna: Springer-Verlag Publisher, 218-221.

Agrawal, A. *et al.* 2005. Dynamics in proportionate selection. In: *Proceedings of the 7th International Conference on Adaptive and Natural Computing Algorithms, Mar 21-23, 2005, Coimbra, Portugal.* Vienna: Springer-Verlag Publisher, 25-28.

Aguilar-Ruiz, J.S., Giraldez, R. and Riquelme, J.C. 2007. Natural encoding for evolutionary supervised learning. *IEEE Transactions on Evolutionary Computation*, **11**(4), 466-479.

Aguirre, E. and González, A. 2003. A fuzzy perceptual model for ultrasound sensors applied to intelligent navigation of mobile robots. *Applied Intelligence*, **19**(3), 171-187.

Aha, D.A. 1997. *Lazy Learning*. Dordrecht: Kluwier Academic Publishers.

Ahn, C.W. 2007. Genetic algorithm for shortest path routing problem. http://www.evolution.re.kr/, Aug 15, 2007.

Ahn, C.W. and Ramakrishna, R. 2002. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation*, **6**(6), 566-579.

Amor, H.B. and Rettinger, A. 2005. Intelligent exploration for genetic algorithms: using self-organizing maps in evolutionary computation. In: *Proceedings of the Genetic and Evolutionary Computation Conference, Jun 25-29, 2005, Washington, D.C., USA*. New York, NY: Association for Computing Machinery, 1531-1538.

Antich, J. and Ortiz, A. 2006. Bug-based T2: A new globally convergent potential field approach to obstacle avoidance. In: *Proceedings of the International Conference on Intelligent Robots and Systems, Oct 1-15, 2006, Beijing, China.* Los Alamitos, CA: IEEE Computer Society Press, 430-435.

Arambula Cosio, F. and Padilla Castaneda, M.A. 2004. Autonomous robot navigation using adaptive potential fields. *Mathematical and Computer Modelling*, **40**(9-10), 1141-1156.

Arkin, R.C. 1986. Path planning for a vision-based autonomous robot. In: *Proceedings of SPIE Conference on Mobile Robots, Oct 30-31, 1986, Cambridge, MA, USA*. Bellingham, WA: SPIE Press, 240-249.

Arkin, R.C. 1987. Motor schema based navigation for a mobile robot: An approach to programming by behaviour. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Mar 31-Apr 3, 1987, Raleigh, NC, USA*. Los Alamitos, CA: IEEE Computer Society Press, 264-271.

Arkin, R.C. 1989. Navigational path planning for a vision-based mobile robot. *Robotica*, **7**(1), 49-63.

Arkin, R.C. 1998. *Behavior-Based Robotics*. Cambridge, MA: MIT Press.

Armesto, L. and Tornero, J. 2004. SLAM based on Kalman filter for multi-rate fusion of laser and encoder measurements. In: *Proceedings of the IEEE Conference on Intelligent Robots and Systems, Sep 28-Oct 2, 2004, Sendai, Japan*. Los Alamitos, CA: IEEE Computer Society Press, 1860-1865.

Asada, M. *et al.* 1999. RoboCup: today and tomorrow - what we have learned. *Artificial Intelligence*, **110**(2), 193-214.

Ashlock, D.A., Manikas, T.W. and Ashenayi, K. 2006. Evolving a diverse collection of robot path planning problems. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jul 16-21, 2006, Vancouver, Canada*. Piscataway, NJ: IEEE Press, 1837-1844.

Bäck, T., Fogel, D. and Michalewicz, Z. 1997. *Handbook of Evolutionary Computation.* New York: Oxford University Press and Institute of Physics Publishing.

Bäck, T., Hammel, U., and Schwefel, H.-P. 1997. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, **1**(1), 3-17.

Baker, J.E. 1985. Adaptive selection methods for genetic algorithms. In: *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications, Jul 24-26, 1985, Pittsburgh, PA, USA*. NJ: Lawrence Erlbaum Associates, 101-111.

Baker, J.E. 1987. Reducing bias and ineciency in the selection algorithm. In: *Proceedings of the 2nd International Conference on Genetic Algorithms, Jul 28-31, 1987, Cambridge, MA, USA*. NJ: Lawrence Erlbaum Associates, 14-21.

Baklouti, N. and Alimi, A.M. 2007. Motion planning in dynamic and unknown environment using an interval type-2 TSK fuzzy logic controller. In: *Proceedings of the IEEE Conference on Fuzzy Systems, Jul 23-26, 2007, London, UK*. Piscataway, NJ: IEEE Press, 1-6.

Ballester, P.J. and Carter, J.N. 2003. Real-parameter genetic algorithms for finding multiple optimal solutions in multi-modal optimization. In: *Proceedings of the Genetic and Evolutionary Computation Conference, Jul 12-16, 2003, Chicago, IL, USA*. Berlin: Springer-Verlag Publisher, 706-717.

Barto, A.G., Sutton, R.S. and Watkins, C.J.C.H. 1990. Learning and sequential decision making. In: Gabriel, M. and Moore, J. (eds.), *Learning and Computational Neuroscience: Foundations of Adaptive Networks*. Cambridge, MA: MIT Press, 539-602.

Beasley, D., Bull, D. and Martin, R.R. 1993. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, **15**(2):58-69.

Bekey, G.A. 2005. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. Cambridge, MA: MIT Press.

Berman, S., Edan, Y. and Jamshidi, M. 2003. Navigation of decentralized autonomous automatic guided vehicles in material handling. *IEEE Transactions on Robotics and Automation*, **19**(4), 743-749.

Bessonnet, G., Seguin, P. and Sardain, P. 2005. A parametric optimization approach to walking pattern synthesis. *International Journal of Robotics Research*, **24**(7), 523-536.

Blickle, T. and Thiele, L. 1997. A Comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4), 361-394.

Bonasso, R.P. *et al.* 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2), 237-256.

Booker, L.B. 1982. *Intelligent Behavior as An Adaptation to The Task Environment*. Ph.D. thesis, University of Michigan.

Borenstein, J. and Koren, Y. 1991. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3), 278-288.

Brindle, A. 1981. *Genetic Algorithms for Function Optimization*. Ph.D. thesis, Department of Computer Science, University of Alberta.

Brno University of Technology, Czech Republic. 2006. *Autonomous mobile robotics toolbox*. http://wes.feec.vutbr.cz/UAMT/robotics/simulations/amrt/, Jul 21, 2006.

Brooks, R.A. 1985. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, **2**(1), 14-23.

Bui, L.T., Branke, J. and Abbass, H.A. 2005. Multiobjective optimization for dynamic environments, In: *Proceedings of the IEEE Congress on Evolutionary Computation, Sep 2-5, 2005, Edinburgh, UK.* Piscataway, NJ: IEEE Press, 2349-2356.

Bullinaria, J.A. 2004. Generational versus steady-state evolution for optimizing neural network learning. In: *Proceedings of the IEEE International Joint Conference on Neural Networks, Jul 25-29, 2004, Budapest, Hungary.* Piscataway, NJ: IEEE Press, 2297-2302.

Burgard, W. *et al.* 2005. Coordinated multi-robot exploration. *IEEE Transaction on Robotics*, **21**(3), 376-386.

Burke, E., Gustafson, S. and Kendall, G. 2002. A survey and analysis of diversity measures in genetic programming. In: *Proceedings of the Genetic and Evolutionary Computation Conference, Jul 9-13, 2002, New York, USA.* San Francisco, CA: Morgan Kaufmann Publishers, 716-723.

Buyurgan, N. *et al.* 2007. Real-time routing selection for automated guided vehicles in a flexible manufacturing system. *Journal of Manufacturing Technology Management*, **18**(2), 169-181.

Cai, Z. and Peng, Z. 2001. The application of a novel encoding mechanism in path planning for a mobile robot. *Robot*, **23**(3), 230-233.

Cantu-Paz, E. 2000. *Efficient and Accurate Parallel Genetic Algorithms.* Norwell, MA: Kluwer, 2000.

Carrano, E.G. *et al.* 2006. Electric distribution network multiobjective design using a problem-specific genetic algorithm. *IEEE Transactions on Power Delivery*, **21**(2), 995-1005.

Cavicchio, D.J. 1970. *Adaptive Search Using Simulated Evolution.* Ph.D. thesis, University of Michigan.

Chaiyaratana, N. and Zalzala, A.M.S. 1997. Recent developments in evolutionary and genetic algorithms: theory and applications. In: *Proceedings of the 2nd International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Sep 2-4, 1997, Glasgow, UK.* London: IEE Press, 270-277.

Chambers, L. (ed.) 2000. *The Practical Handbook of Genetic Algorithms: Applications*. 2nd ed. Boca Raton, FL: CRC Press.

Chang, S.J., Hou, H.S. and Su, Y.K. 2006. Automated passive filter synthesis using a novel tree representation and genetic programming. *IEEE Transactions on Evolutionary Computation*, **10**(1), 93-100.

Chang, Z. *et al.* 2005. GA path planning for AUV to avoid moving obstacles based on forward looking sonar. In: *Proceedings of the International Conference on Machine Learning and Cybernetics, Aug 18-21, 2005, Guangzhou, China.* Los Alamitos, CA: IEEE Computer Society Press, 1498-1502.

Chen, D., Lee, C.Y. and Park, C.H. 2005. Hybrid genetic algorithm and simulated annealing (HGASA) in global function optimization. In: *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence table of contents, Nov 14-16, 2005, Hong Kong, China.* Washington, DC: IEEE Computer Society Press, 126-131.

Chen, H. and Xu, Z. 2005. Path planning based on a new genetic algorithm. In: *Proceedings of the 2nd International Conference on Neural Networks and Brain, Oct*

13-15, 2005, Beijing, China. Los Alamitos, CA: IEEE Computer Society Press, 788-792.

Choset, H. *et al.* 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementation.* Cambridge, MA: MIT Press.

Cocora, A. *et al.* 2006. Learning relational navigation policies. In: *Proceedings of the International Conference on Intelligent Robots and Systems, Oct 9-15, 2006, Beijing, China.* Los Alamitos, CA: IEEE Computer Society Press, 2792-2797.

Conceicao Antonio, C.A. 2006. A hierarchical genetic algorithm with age structure for multimodal optimal design of hybrid composites. *Structural and Multidisciplinary Optimization*, **31**(4), 280-294.

Connell, J. 1992. SSS: a hybrid architecture applied to robot navigation. In: *Proceedings of the IEEE Conference on Robotics and Automation, May 12-14, 1992, Nice, France.* Los Alamitos, CA: IEEE Computer Society Press, 2719-2724.

Connors, J. and Elkaim, G. 2007. Analysis of a spline based, obstacle avoiding path planning algorithm. In: *Proceedings of the 65th Vehicular Technology Conference, Apr 22-25, 2007, Dublin, Ireland.* Piscataway, NJ: IEEE Press, 2565-2569.

Cormen, T.H. *et al.* 2001. *Introduction to Algorithms*, 2nd ed. New York: MIT Press and McGraw-Hill.

Critchlow, A.J. 1985. *Introduction to Robotics.* New York: Macmillan.

Davies, C. and Lingras, P. 2003. Genetic algorithms for rerouting shortest paths in dynamic and stochastic networks. *European Journal of Operational Research*, **144**(1), 27-38.

de Boor, C. 1978. *A Practical Guide to Splines.* Berlin: Springer-Verlag Publisher.

De Jong, K.A. 1975. *An Analysis of the Behaviors of A Class of Genetic Adaptive Systems*. Ph.D. thesis, University of Michigan.

De Jong, K.A. 1992. Genetic algorithms are NOT function optimizers. In: Whitley, D. (ed.), *Foundations of Genetic Algorithms 2*. San Francisco, CA: Morgan Kaufmann Publishers, 6-18.

De Jong, K.A. and Sarma, J. 1992. Generation gaps revisited. In: Whitley, L.D. (ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann Publishers, 19-28.

De Jong, K.A. and Spears, W. 1993. On the state of evolutionary computation. In: *Proceedings of the 5th International Conference on Genetic Algorithms, Jul 17-21, 1993, San Mateo, CA, USA*. San Francisco: Morgan Kaufmann Publishers, 618-625.

Dengiz, O., Dozier, G. and Smith, A.E. 2004. Non-deterministic decoding with memory to enhance precision in binary-coded genetic algorithms. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jun 19-23, 2004, Portland, Oregon, USA*. Piscataway, NJ: IEEE Press, 2166-2172.

Dessmark, A. and Pelc, A. 2004. Optimal graph exploration without good maps. *Theoretical Computer Science*, **326**(1-3), 343-362.

Dick, G. 2005. A comparison of localised and global niching methods. In: *Proceedings of 17th Annual Colloquium of the Spatial Information Research Centre, Nov 24-25, 2005, Dunedin, New Zealand, USA*. New Zealand: University of Otago, 91-101.

Dijkstra, E.W. 1959. A note on two problems in connexion with graphs. In: *Numerische Mathematik*, **1**, 269-271.

Dixon, K.R., Dolan, J.M. and Khosla, P.K. 2004. Predictive robot programming: theoretical and experimental analysis. *International Journal of Robotics Research*, **23**(9), 955-973.

Duan, G. and Yu, Y. 2003. Power distribution system optimization by an algorithm for capacitated steiner tree problems with complex-flows and arbitrary cost functions. *Electrical Power and Energy Systems*, **25**(7), 515-523.

Dumitrescu, D. *et al*. 2000. *Evolutionary Computation*. Boca Raton, FL: CRC Press.

Dyllong, E. and Visioli, A. 2003. Planning and real-time modifications of a trajectory using spline techniques. *Robotica*, **21**(5), 475-482.

Elshamli, A., Abdullah, H.A. and Areibi, S. 2004. Genetic algorithm for dynamic path planning. In: *Proceedings of the Canadian Conference on Electrical and Computer Engineering, May 2-5, 2004, Niagara Fall, Ontario, Canada*. Piscataway, NJ: IEEE Press, 677-680.

Feng, J. *et al*. 2006. A hybrid genetic algorithm with fitness sharing based on rough sets theory. In: *Proceedings of the IEEE 6th World Congress on Intelligent Control and Automation, Jun 21-23, 2006, Dalian, China*. Piscataway, NJ: IEEE Press, 3340-3344.

Fiorini, P. and Shiller, Z. 1998. Motion planning in dynamic environment using velocity obstacles. *International Journal of Robotics Research*, **17**(7), 760-772.

Fleischer, R. and Trippen, G. 2005. Exploring an unknown graph efficiently. *Lecture Notes in Computer Science*, **3669**, 11-22.

Fogel, D.B. 1994. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, **5**(1), 3-14.

Fonseca, C.M. and Correia, M.B. 2005. Developing redundant binary representations for genetic search. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Sep 2-5, 2005, Edinburgh, UK.* Piscataway, NJ: IEEE Press, 1675-1682.

Fox, D., Burgard, W. and Thrun, S. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, **4**(1), 23-33.

Fraichard, F. and Asama, H. 2004. Inevitable collision states - a step towards safer robots? *Advanced Robotics*, **18**(10), 1001-1024.

Freda, L. and Oriolo, G. 2005. Frontier-based probabilistic strategies for sensor-based exploration. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 18-22, 2005, Barcelona, Spain.* Los Alamitos, CA: IEEE Computer Society Press, 3892-3898.

Frese, U., Larsson, P. and Duckett, T. 2005. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, **21**(2), 196-207.

Fu, Y. *et al.* 2006. A navigation strategy based on global geographical planning and local feature positioning for mobile robot in large unknown environment. In: *Proceedings of the 1st International Symposium on Systems and Control in Aerospace and Astronautics, Jan 19-21, 2006, Harbin, China.* Los Alamitos, CA: IEEE Computer Society Press, 1189-1193.

Fujisawa, K. *et al.* 2000. Real-time decision making for autonomous mobile robot using evolution strategy and anytime sensing. In: *Proceedings of the 26th Annual Conference of the IEEE Industrial Electronics Society, Oct 22-28, 2000, Nagoya, Japan*, Los Alamitos, CA: IEEE Computer Society Press, 2809-2814.

Fujita, M. and Kitano, H. 1998. Development of an autonomous quadruped robot for robot entertainment. *Autonomous Robots*, **5**(2), 7-18.

Gaing, Z.L. and Huang, H.S. 2004. Real-coded mixed-integer genetic algorithm for constrained optimal power flow. In: *Proceedings of the IEEE Region 10 Conference on Analog and Digital Techniques in Electrical Engineering, Nov 21-24, 2004, Chiang Mai, Thailand.* Piscataway, NJ: IEEE Press, 323-326.

Garg, D. and Kumar, M. 2001. Optimal path planning and energy minimization via genetic algorithm applied to cooperating robotic manipulators. In: *Proceedings of the American Society of Mechanical Engineers, Dynamic Systems and Control Division, Nov 11-16, 2001, New York, USA.* New York NY: American Society of Mechanical Engineers, 71-79.

Garrozi, C. and Araujo, A.F.R. 2006. Multiobjective genetic algorithm for multicast routing. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jul 16-21, 2006, Vancouver, Canada.* Piscataway, NJ: IEEE Press, 2513-2520.

Gartshore, R., Palmer, P. and Illingworth, J. 2005. A novel exploration algorithm based on an improvement strategy. *International Journal of Advanced Robotic Systems*, **2**(4), 287-294.

Gaspar, J., Winters, N. and Santos-Victor, J. 2000. Vision-based navigation and environmental representations with an omnidirectional camera. *IEEE Transactions on Robotics and Automation*, **16**(6), 890-898.

Gat, E. 1991a. *Reliable Goal-directed Reactive Control for Real-world Autonomous Mobile Robots*. Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

Gat, E. 1991b. ALFA: a language for programming reactive robotic control systems. In: *Proceedings of the IEEE Conference on Robotics and Automation, Apr 9-11, 1991, Sacramento, CA, USA.* Los Alamitos, CA: IEEE Computer Society Press, 1116-1121.

Gat, E. 1998. On three-layer architectures. In: Kortenkamp, D., Bonnasso, R.P. and Murphy, R. (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. Cambridge, MA: MIT Press, 195-210.

Ge, S.S. and Cui, Y.J. 2000. New potential functions for mobile robot path planning. *IEEE Transactions on Robotics and Automation*, **16**(5), 615-620.

Ge, S.S. and Cui, Y.J. 2002. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, **13**(3), 207-222.

Geisler, T. and Manikas, T.W. 2002. Autonomous robot navigation system using a novel value encoded genetic algorithm. In: *Proceedings of the 45th IEEE International Midwest Symposium on Circuits and Systems, Aug 4-7, 2002, Tulsa, Oklahoma, USA*. Piscataway, NJ: IEEE Press, 45-48.

Gen, M., Cheng, R. and Oren, S.S. 2001. Network design techniques using adapted genetic algorithms. *Advances in Engineering Software*, **32**(9), 731-744.

Georgeoff. M.P. and Lansky, A.L. 1987. Reactive reasoning and planning. In: *Proceedings of the 6th National Conference on Artificial Intelligence, Jul 13-17, Seattle, WA, USA*. Menlo Park, CA: AAAI Press, 677-682.

Ghaffari, M. *et al*. 2004. Design of an unmanned ground vehicle, bearcat III, theory and practice. *Journal of Robotic Systems*, **21**(9), 471-480.

Glickman, M. and Sycara, K. 2000. Reasons for premature convergence of self-adapting mutation rates. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jul 16-19, 2000, La Jolla, CA, USA*. Piscataway, NJ: IEEE Press, 62-69.

Goldberg, D.E. 1989. *Genetic Algorithms in Search Optimization and Machine Learning*. Reading, MA: Addison-Wesley Publishing.

Goldberg, D.E. 2002. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Boston, MA: Kluwer Academic Publisher.

Goldberg, D.E. and Deb, K. 1991. A comparative analysis of selection schemes used in genetic algorithms. In: Rawlins, G.J.E. (ed.), *Foundations of Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann Publishers, 69-93.

Goldberg, D.E., Korb, B. and Deb, K. 1989. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5), 493-530.

Goldberg, D.E. and Rechardson, J. 1987. Genetic algorithms with sharing for multimodal optimization. In: *Proceedings of the 2nd International Conference on Genetic Algorithm, Jul 28-31, 1987, Massachusetts Institute of Technology, Cambridge, MA, USA*. Hillsdale, NJ: Lawrence Erlbaum Associates, 41-49.

Gonzalez-Banos, H.H. and Latombe, J.C. 2002. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research*, 21(10-11), 829-848.

Grefenstette, J.J. and Baker, J.E. 1989. How genetic algorithms work: A critical look at implicit parallelism. In: *Proceedings of the 3rd International Conference on Genetic Algorithms, Jun 4-7, 1989, Fairfax, Virginia, USA*. San Francisco: Morgan Kaufmann Publishers, 20-27.

Gu, L. and Owens, R. 1998. Visual guidance of robot motion. In: *Proceedings of the IEEE International Conference on Intelligent Processing Systems, Oct 11-14, 1998, San Diego, California, USA*. Piscataway, NJ: IEEE Press, 1242-1246.

Guan, Y. *et al*. 2005. On robotic trajectory planning using polynomial interpolations. In: *Proceedings of the IEEE International Conference on Robotics and Biomimetics, Jun 29-Jul 3, 2005, Hongkong, China*. Los Alamitos, CA: IEEE Computer Society Press, 111-116.

Guo, J.F., Gui, Q.M. and Yang, Y.X. 2003. Least squares fitting method based on bivariate nonuniform B-spline and its applications in surveying engineering. *Journal of Surveying Engineering*, **129**(3), 105-109.

Guo, J.H. 2006. A waypoint-tracking controller for a biomimetic autonomous underwater vehicle. *Ocean Engineering*, **33**(17), 2369-2380.

Guo, S.X., Sawamoto, J. and Pan, Q.X. 2005. A novel type of microrobot for biomedical application. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Aug 2-6, 2005, Edmonton, Canada.* Los Alamitos, CA: IEEE Computer Society Press, 1047-1052.

Hagras, H., Callaghan, V. and Colley, M. 2004. Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online fuzzy-genetic system. *Fuzzy Sets and Systems*, **141**(1), 107-160.

Hand, A. *et al.* 2005. Benchmarking of robot path planning algorithms. In: *Intelligent Engineering Systems Through Artificial Neural Networks: Smart Engineering Systems Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life*. New York: ASME Press, 377-384.

Hao, Y. and Agrawal, S.K. 2005. Formation planning and control of UGVs with trailers. *Autonomous Robots*, **19**(3), 257-270.

Hart, P.E., Nilsson, N.J. and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.

Hermanu, T. *et al.* 2004. Autonomous robot navigation using a genetic algorithm with an efficient genotype structure. In: *Intelligent Engineering Systems Through Artificial Neural Networks: Smart Engineering Systems Design: Neural Networks,*

Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life. New York: ASME Press, 319-324.

Herrera, F. and Lozano, M. 2000. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1), 43-63.

Hiroyasu, T., Miki, M. and Watanabe, S. 1999. Distributed genetic algorithms with a new sharing approach in multiobjective optimization problems. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jul 6-9, 1999, Washington, DC, USA*. Piscataway, NJ: IEEE Press, 69-76.

Ho, Y.C. and Pepyne, D.L. 2002. Simple explanation of the No-Free-Lunch theorem and its implications. *Journal of Optimization Theory and Applications*, **115**(3), 115-549.

Hocaoglu, C. and Sanderson, A. C. 2001. Planning multiple paths with evolutionary speciation. *IEEE Transactions on Evolutionary Computation*, 5(3), 169-191.

Holland, J.H. 1975. *Adaptation in Natural And Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Howard, A. 2006. Multi-robot simultaneous localization and mapping using particle filters. *International Journal of Robotics Research*, **25**(12), 1243-1256.

Hrstka, O. and Kucerova, A. 2004. Improvements of real coded genetic algorithms based on differential operators preventing premature convergence. *Advances in Engineering Software*, **35**(3-4), 237-246.

Hu, Y. and Yang, S.X. 2004. A knowledge based genetic algorithm for path planning of a mobile robot. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 26-May 1, 2004, New Orleans, LA, USA*. Piscataway, NJ: IEEE Press, 4350-4355.

Huang, P.F., Xu, Y.S. and Liang, B. 2005. Dynamic balance control of multi-arm free floating space robots. *International Journal of Advanced Robotic Systems*, 2(2), 117-125.

Hussein, A.M. and Elnagar, A. 2002. Motion planning using Maxwell's equations. In: *Proceedings of the International Conference on Intelligent Robots and Systems, Sep 30-Oct 4, 2002, Lausanne, Switzerland.* Los Alamitos, CA: IEEE Computer Society Press, 2347-2352.

Hutt, B. and Warwick, K. 2007. Synapsing variable-Length crossover: meaningful crossover for variable-length genomes. *IEEE Transactions on Evolutionary Computation*, 11(1), 118-131.

Hutter, M. 2002. Fitness uniform selection to preserve genetic diversity. In: *Proceedings of the Congress on Evolutionary Computation, May12-17, 2002, Honolulu, HI, USA*. Piscataway, NJ: IEEE Press, 783-788.

Jablonski, J. and Posey, J. 1985. Robotics terminology. In: Nof, S. (ed.), *Handbook of Industrial Robotics*. New York: John Wiley and Sons Ltd, 1271-1303.

Jerald, J. *et al.* 2005. Simultaneous scheduling of parts and automated guided vehicles in an FMS environment using adaptive genetic algorithm. *International Journal of Advanced Manufacturing Technology,* 29(5), 584-589.

Ji, X., Iwamura, K. and Shao, Z. 2007. New models for shortest path problem with fuzzy arc lengths. *Applied Mathematical Modelling*, 31(2), 259-269.

Jia, M., Zhou, G. and Chen, Z. 2004. An efficient strategy integrating grid and topological information for robot exploration. In: *Proceedings of the IEEE Conference on Robotics, Automation and Mechatronics, Dec 1-3, 2004, Singapore.* Piscataway, NJ: IEEE Press, 667-672.

Jones, J. and Soule, T. 2006. Comparing genetic robustness in generational vs. steady state evolutionary algorithms. In: *Proceedings of the 8th Conference on Genetic and Evolutionary Computation, Jul 8-12, 2006, Seattle, Washington, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 143-150.

Khatib, O. 1985. Real-time obstacle avoidance for manipulators and mobile robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Mar 25-28, 1985, St. Louis, MO, USA*. Los Alamitos, CA: IEEE Computer Society Press, 500-505.

Khatib, O. 1986. The operational space formulation in the analysis, design, and control of robot manipulators. In: *Proceedings of the 3rd International Symposium on Robotics Research, Oct 7-11, 1986, Gouvieux, France*. Cambridge, MA: MIT Press, 263-270.

Kim, H.J. and Shim, D.H. 2003. A flight control system for aerial robots: algorithms and experiments. *Control Engineering Practice*, **11**(12), 1389-1400.

Kim, I.Y. and De Weck, O.L. 2005. Variable chromosome length genetic algorithm for progressive refinement in topology optimization. *Structural and Multidisciplinary Optimization*, **29**(6), 445-456.

Kira, Z. and Arkin, R.C. 2004. Forgetting bad behavior: Memory management for case-based navigation. In: *Proceedings of the IEEE Conference on Intelligent Robots and Systems, Sep 28-Oct 2, 2004, Sendai, Japan*. Los Alamitos, CA: IEEE Computer Society Press, 3145-3152.

Kitano, H. 1990. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, **4**(4), 461-476.

Koehler, G.J. 2007. Conditions that obviate the No Free Lunch theorems for optimization. *Journal on Computing*, **19**(2), 273-279.

Koenig, S. and Likhachev, M. 2002. Improved fast replanning for robot navigation in unknown terrain. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 11-15, 2002, Washington, USA*. Los Alamitos, CA: IEEE Computer Society Press, 968-975.

Koppen, M. 2004. No-Free-Lunch theorems and the diversity of algorithms. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jun 19-23, 2004, Portland, OR, USA*. Piscataway, NJ: IEEE Press, 235-241.

Korb, W. and Troch, I. 2003. Data reduction for manipulator path planning. *Robotica*, **21**(6), 605-614.

Koren, Y. and Borenstein, J. 1991. Potential field methods and their inherent limitations for mobile robot navigation. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 9-11, 1991, Sacramento, California, USA*. Los Alamitos, CA: IEEE Computer Society Press, 1398-1404.

Kortenkamp, D. and Weymouth, T. 1994. Topological mapping for mobile robots using a combination of sonar and vision sensing. In: *Proceedings of the 12th National Conference on Artificial Intelligence, Jul 31-Aug 4, 1994, Seattle, Washington, USA*. Menlo Park, CA: AAAI Press, 979-984.

Kortenkamp, D., Bonasso, R.P. and Murphy, R.R. 1998. *Artificial Intelligence and Mobile Robots*. Cambridge, MA: MIT/AAAI Press.

Kubota, N. 2004. A spiking neural network for behavior learning of a mobile robot in a dynamic environment. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Oct 10-13, 2004, The Hague, The Netherlands*. Piscataway, NJ: IEEE Press, 5783-5788.

Kumon, E.M. *et al.* 2006. Autopilot system for kiteplane. *IEEE Transaction on Mechatronics*, **11**(5), 615-624.

Kunwar, F. and Benhabib, B. 2006. Rendezvous-guidance trajectory planning for robotic dynamic obstacle avoidance and interception. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, **36**(6), 1432-1441.

Kurihara, K. *et al.* 2005. Mobile robots path planning method with the existence of moving obstacles. In: *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation, Sep 19-22, 2005, Catania, Italy.* Piscataway, NJ: IEEE Press, 195-202.

Kvasov, B.I. 2000. *Methods of Shape-preserving Spline Approximation.* Singapore: World Scientific.

Large, F., Laugier, C. and Shiller, Z. 2005. Navigation among moving obstacles using the NLVO: principles and applications to intelligent vehicles. *Autonomous Robots Journal*, **19**(2), 159-171.

Larrañaga, P. *et al.* 1999. Genetic algorithms for the travelling salesman problem: a review of representations and operators. *Artificial Intelligence Review*, **13**(2), 129-170.

Latombe, J.C. 1991. *Robot Motion Planning.* Boston: Kluwer Academic Publishers.

Lee, W.S. 1998. *Robotic Weed Control System for Tomatoes.* Ph.D. thesis, University of California.

Leon, J.A.F., Tosini, M. and Acosta, G.G. 2004. Evolutionary reactive behavior for mobile robot navigation. In: *Proceedings of the IEEE International Conference on*

Cybernetics and Intelligent System, Dec 1-3, 2004, Singapore. Piscataway, NJ: IEEE Press, 532-537.

Leonard, J.J. and Durrant-Whyte, H.F. 1991. Simultaneous map building and localization for an autonomous mobile robot. In: *Proceedings of the IEEE Workshop on Intelligent Robots and Systems, Nov 3-5, 1991, Osaka, Japan*. IEEE Press, 1442-1447.

Leung, K.S., Sun, J.Y. and Xu, Z.B. 2002. Efficiency speed-up strategies for evolutionary computation: an adaptive implementation. *Engineering Computations*, **19**(3), 272-304.

Levitt, T.S. and Lawton, D.T., 1990. Qualitative navigation for mobile robots. *Artificial Intelligence*, **44**, 305-360.

Li, M. and Kou, J. 2005. A novel type of niching methods based on steady-state genetic algorithm. In: *Proceedings of the 1st International Conference on Advances in Natural Computation, Aug 27-29, 2005, Changsha, China*. New York: Springer Publishing Company, 37-47.

Li, Q. *et al*. 2006. An improved genetic algorithm of optimum path planning for mobile robots. In: *Proceedings of the 6th International Conference on Intelligent Systems Design and Applications, Oct 16-18, 2006, Jinan, China*. Los Alamitos, CA: IEEE Computer Society Press, 637-642.

Li, W.F. *et al*. 2004. An architecture for indoor navigation. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 26-May 1, 2004, New Orleans, USA*. Los Alamitos, CA: IEEE Computer Society Press, 1783-1788.

Liang, Y., Leung, K.S. and Lee, K.H. 2006. A novel binary variable representation for genetic and evolutionary algorithms. In: *Proceedings of the IEEE Congress on*

Evolutionary Computation, Jul 16-21, 2006, Vancouver, Canada. Piscataway, NJ: IEEE Press, 536-543.

Liang, Y., Leung, K.S. and Xu, Z.B. 2007. A novel splicing/decomposable binary encoding and its operators for genetic and evolutionary algorithms. *Applied Mathematics and Computation*, **190**(1), 887-904.

Likhachev, M. *et al.* 2005. Anytime dynamic A*: An anytime, replanning algorithm. In: *Proceedings of the International Conference on Automated Planning and Scheduling, Jun 5-10, 2005, Monterey CA.* Menlo Park, CA: AAAI Press, 262-271.

Lim, S. and Cho, S. 2005. Language generation for conversational agent by evolution of plan trees with genetic programming. *Lecture Notes in Artificial Intelligence*, **3558**, 305-315.

Lin, C.Y. and Wu, W.H. 2002. Niche identification techniques in multimodal genetic search with sharing scheme. *Advances in Engineering Software*, **33**(11-12), 779-791.

Lin, H., Xiao, J. and Michalewicz, Z. 1994. Evolutionary navigator for a mobile robot. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 8-13, 1994, San Diego, USA.* Los Alamitos, CA: IEEE Computer Society Press, 2199-2204.

Liu, B. *et al.* 1994. Integrating case-based reasoning, knowledge-based approach and Dijkstra algorithm for route finding. In: *Proceedings of the 10th Conference on Artificial Intelligence for Applications, Mar 1-4, 1994, San Antonio, Texas, USA.* Piscataway, NJ: IEEE Press, 149-155.

Liu, J., Hu, H. and Gu, D. 2006. A layered control architecture for autonomous robotic fish. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Oct 9-13, 2006, Beijing, China.* Los Alamitos, CA: IEEE Computer Society Press, 312-317.

Liu, S., Tian, Y. and Liu, J. 2004. Multi mobile robot path planning based on genetic algorithm. In: *Proceedings of the IEEE 5th World Congress on Intelligent Control and Automation, Jun 15-19, 2004, Hangzhou, China*. Piscataway, NJ: IEEE Press, 4706-4709.

Low, K.H., Leow, W.K. and Ang Jr, M.H. 2002. Integrated planning and control of mobile robot with self-organizing neural network. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 11-15, 2002, Washington, D.C., USA*. Los Alamitos, CA: IEEE Computer Society Press, 3870-3875.

Low, K.H., Leow, W.K. and Ang Jr, M.H. 2003. Enhancing the reactive capabilities of integrated planning and control with cooperative extended kohonen maps. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 12-17, 2003, Taipei, China*. Los Alamitos, CA: IEEE Computer Society Press, 3428-3433.

Low, K.H., Leow, W.K. and Ang Jr, M.H. 2006. Autonomic mobile sensor network with self-coordinated task allocation and execution. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, **36**(3), 315-327.

Luerssen, M.H. 2005. Phenotype diversity objectives for graph grammar evolution. In: Abbass, H.A., Bossamaier, T. and Wiles, J. (eds.), *Recent Advances in Artificial Life, Advances in Natural Computation*. Singapore: World Scientific.

Luger, G.F. 2002. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, 4th ed. London: Addison-Wesley.

Lumelsky, V. 2005. *Sensing, Intelligence, Motion: How Robots and Humans Move in an Unstructured World*. Hoboken, NJ: John Wiley And Sons Ltd.

Lyons, D. 1992. Reactive planning. In: Shapiro, S. (ed.), *Encyclopedia of Artificial Intelligence*. New York: John Wiley and Sons Ltd, 1171-1182.

Lyons, D. and Hendriks, A. 1992. Planning for reactive robot behavior. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 10-15, 1992, Nice, France*. Los Alamitos, CA: IEEE Computer Society Press, 2675-2680.

Lyons, D. and Hendriks, A. 1995. Planning as incremental adaptation of a reactive system. *Robotics and Autonomous Systems*, **14**(4), 255-288.

Maalouf, E., Saad, M. and Saliah, H. 2005. A higher level path tracking controller for a four-wheel differentially steered mobile robot. *Journal of Robotics and Autonomous Systems*, **54**(1), 23-33.

Maaref, H. and Barret, C. 2002. Sensor-based navigation of a mobile robot in an indoor environment. *Robotics and Autonomous Systems*, **38**(1), 1-18.

Macfarlane, S. and Croft, E.A. 2003. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, **19**(1), 42-52.

Magid, E. *et al.* 2006. Spline-based robot navigation. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Oct 9-15, 2006, Beijing, China*. Los Alamitos, CA: IEEE Computer Society Press, 2296-2301.

Mahfoud, S.W. 1992. Crowding and preselection revisited. In: *Proceedings of 2nd Conference on Parallel Problem Solving from Nature, Sep 28-30, 1992, Brussels, Belgium*. Amsterdam: North-Holland, 27-34.

Mahfoud, S.W. 1994. Genetic drift in sharing method. In: *Proceedings of the 1st IEEE International Conference on Evolutionary Computation, Jun 27-29, 1994, Orlando, FL, USA*. Los Alamitos, CA: IEEE Computer Society Press, 67-72.

Mahfoud, S.W. 1995a. *Niching Methods for Genetic Algorithms*. Ph.D. thesis, University of Illinois, Urbana-Champaign.

Mahfoud, S.W. 1995b. A comparison of parallel and sequential niching methods. In: *Proceedings of the 6th International Conference on Genetic Algorithms, Jul 15-19, 1995, Pittsburgh, PA, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 136–143.

Mahkovic, R. and Slivnik, T. 2000. Constructing the generalized local Voronoi diagram from laser range scanner data. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, **30**(6), 710-719.

Malhotra, R. and Sarkar, A. 2005. Development of a fuzzy logic based mobile robot for dynamic obstacle avoidance and goal acquisition in an unstructured environment. In: *Proceedings of the International Conference on Advanced Intelligent Mechatronics, Jul 24-28, 2005, Monterey, California, USA*. Piscataway, NJ: IEEE Press, 1198-1203.

Mali, A.D. 2002. On the behavior-based architectures of autonomous agents. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, **32**(3), 231-242.

Masson, F., Guivant, J. and Nebot, E. 2003. Robust navigation and mapping architecture for large environments. *Journal of Robotic Systems*, **20**(10), 621-634.

Mathworks, 2006. MATLAB. http://www.mathworks.com/, Jul 16, 2006.

Mengsheol, O. and Goldberg, D. 1999. Probabilistic crowding: Deterministic crowding with probabilistic replacement. In: *Proceedings of the Conference on Genetic and Evolutionary Computation, Jul 13-17, 1999, Orlando, Florida, USA.* San Francisco, CA: Morgan Kaufmann Publishers, 409-416.

Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs.* 2nd ed. Berlin: Springer-Verlag Publisher.

Miconi, T. and Channon, A. 2006. The N-Strikes-Out algorithm: a steady-state algorithm for coevolution. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jul 16-21, 2006, Vancouver, Canada.* Piscataway, NJ: IEEE Press, 1639-1646.

Min, H.J. 2005. Navigation of a mobile robot using behavior network with Bayesian inference. In: *Proceedings of the IEEE International Conference on Mechatronics and Automation, Jul 29-Aug 1, 2005, Niagara Falls, Canada.* Los Alamitos, CA: IEEE Computer Society Press, 1479-1484.

Minami, M., Gao, J. and Mae, Y. 2007. Chaos-driving robotic intelligence for catching fish. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 10-14, 2007, Roma, Italy.* Los Alamitos, CA: IEEE Computer Society Press, 85-91.

Minguez, J. *et al.* 2001. Global nearness diagram navigation (GND). In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 19-21, 2001, Seoul, Korea.* Los Alamitos, CA: IEEE Computer Society Press, 33-39.

Minguez, J. and Montano, L. 2004. Nearness diagram navigation (ND): collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation,* **20**(1), 45-59.

Minguez, J. and Montano, L. 2005. Sensor-based robot motion generation in unknown, dynamic and troublesome scenarios. *Robotics and Autonomous Systems*, **52**(4), 290-311.

Minguez, J., Montesano, L. and Montano, L. 2004. An architecture for sensor-based navigation in realistic dynamic and troublesome scenarios. In: *Proceedings of the Conference on Intelligent Robots and Systems, Sep 28-Oct 2, 2004, Sendai, Japan.* Los Alamitos, CA: IEEE Computer Society Press, 2750-2756.

Minguez, J., Osuna, J. and Montano, L. 2004. A "Divide and Conquer" strategy based on situations to achieve reactive collision avoidance in troublesome scenarios. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 26-May 1, 2004, New Orleans, LA, USA*. Los Alamitos, CA: IEEE Computer Society Press, 3855-3862.

Mitchell, M. 1996. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.

Montana, D.J. and Davis, L.D. 1989. Training feedback networks using genetic algorithms. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Aug 13-15, 1989, Detroit, MI, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 762-767.

Montesano, L., Minguez, J. and Montano, L. 2005. Modeling the static and the dynamic parts of the environment to improve sensor-based navigation. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 18-22, 2005, Barcelona, Spain*. Los Alamitos, CA: IEEE Computer Society Press, 4556-4562.

Mooney, P. and Winstanley, A. 2006. An evolutionary algorithm for multicriteria path optimization problems. *International Journal of Geographical Information Science*, **20**(4), 401-423.

Mühlenbein, H. and Schlierkamp-Voosen, D. 1993. Predictive models for the breeder genetic algorithm, I.: continuous parameter optimization. *Evolutionary Computation.* **1**(1), 25-49.

Mulvaney, D.J. *et al.* 2005. Real-time machine learning in embedded software and hardware platforms. In: *Proceedings of the International Workshop on Automatic Learning and Real-Time, Sep 7-8, 2005, Siegen, Germany*. 65-78.

Mulvaney, D.J., Wang, Y. and Sillitoe, I.P.W. 2006. Waypoint-based mobile robot navigation. In: *Proceedings of the IEEE 6th World Congress on Intelligent Control and Automation, Jun 21-23, 2006, Dalian, China*. Piscataway, NJ: IEEE Press, 9063-9067.

Muñoz-Salinas, R. *et al.* 2005. A multi-agent system architecture for mobile robot navigation based on fuzzy and visual behavior. *Robotica*, **23**(6), 689-699.

Murphy, R.R. 2000. *Introduction to AI Robotics*. Cambridge, MA: MIT Press.

Murphy, R.R., Marzilli, A. and Noll, E. 1999. Integrating explicit path planning with reactive control of mobile robots using trulla. *Journal of Robotics and Autonomous Systems.* **27**(4), 225-245.

Na, Y.K. and Oh, S.Y. 2003. Hybrid control for autonomous mobile robot navigation using neural network based behavior modules and environment classification. *Autonomous Robots*, **15**(2), 193-206.

Nearchou, A.C. 1998. Path planning of a mobile robot using genetic heuristics. *Robotica*, **16**(5), 575-588.

Nearchou, A.C. 1999. Adaptive navigation of autonomous vehicles using evolutionary algorithms. *Artificial Intelligence in Engineering*, **13**(2), 159-173.

Nefti, S. *et al.* 2001. Intelligent adaptive mobile robot navigation. *Journal of Intelligent and Robotic Systems: Theory and Applications*, **30**(4), 311-329.

Nelson, A.L. *et al.* 2004. Maze exploration behaviors using an integrated evolutionary robotics environment. *Robotics and Autonomous Systems*, **46**(3), 159-173.

Nelson, L.S. 1989. Evaluating overlapping confidence intervals. *Journal of Quality Technology*, **21**(2), 140-141.

Ni, Z. *et al.* 2003. Integrated case-based reasoning. In: *Proceedings of the International Conference on Machine Learning and Cybernetics, Nov 2-5, 2003, Xi'an, China*. Los Alamitos, CA: IEEE Computer Society Press, 1845-1849.

Nikolos, I.K. *et al.* 2003. Evolutionary algorithm based offline/online path planner for UAV navigation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **33**(6), 898-912.

Noggle, J.H. 1993. *Practical Curve Fitting and Data Analysis*. Chemical and Information Systems Series. Chichester: Ellis Horwood-PTR Prentice Hall.

Nolfi, S. and Floreano, D. 2000. *Evolutionary Robotics*: *the Biology, Intelligence, and Technology of Self-organizing Machines*. Cambridge, MA: MIT Press.

Ogren, P. and Leonard, N. 2002. A tractable convergent dynamic window approach to obstacle avoidance. In: *Proceeding of the IEEE International Conference on Intelligent Robots and Systems, Sep 30-Oct 4, 2002, Lausanne, Switzerland.* Los Alamitos, CA: IEEE Computer Society Press, 595-600.

Ogren, P. and Leonard, N. 2005. A convergent dynamic window approach to obstacle avoidance. *IEEE Transactions on Robotics and Automation*, **21**(2), 188-195.

Orebäck, A. and Christensen, H.I. 2003. Evaluation of architectures for mobile robotics. *Autonomous Robots*, **14**(1), 33-49.

Osyczka, A. 2002. *Evolutionary Algorithms for Single and Multicriteria Design Optimization*. Studies in Fuzzyness and Soft Computing. Berlin: Springer-Verlag Publisher.

Panaite, P. and Pelc, A. 1999. Exploring unknown undirected graphs. *Journal of Algorithms*, **33**(2), 281-295.

Parasuraman, R. *et al.* 2005. A flexible delegation-type interface enhances system performance in human supervision of multiple robots: empirical studies with RoboFlag. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, **35**(4), 481-493.

Park, J. and Bobrow, J.E. 2005. Reliable computation of minimum-time motions for manipulators moving in obstacle fields using a successive search for minimum-overload trajectories. *Journal of Robotic Systems*, **22**(1), 1-14.

Park, Y.J., Kim, B.C. and Chun, S.H. 2006. New knowledge extraction technique using probability for case-based reasoning: application to medical diagnosis. *Expert Systems*, **23**(1), 2-20.

Passone, S., Chung, P.W.H. and Nassehi, V. 2006. Incorporating domain-specific knowledge into a genetic algorithm to implement case-based reasoning adaptation. *Knowledge-Based Systems*, **19**(3), 192-201.

Patnaik, S. and Karibasappa, K. 2005. Motion planning of an intelligent robot using GA motivated temporal associative memory. *Applied Artificial Intelligence*, **19**(5), 515-534.

Pavlidid, T. 1982. *Algorithms for Graphics And Image Processing*. Rockville MD: Computer Science Press.

Payton, D.W., Rosenblatt, J.K. and Keirsey, D.M. 1993. Grid-based mapping for autonomous mobile robot. *Robotics and Autonomous Systems*, **11**(1), 13-21.

Pereira, F.B. *et al.* 2002. GVR: a new genetic representation for the vehicle routing problem. *Lecture Notes in Computer Science*, **2464**, 95-102.

Pétrowski, A. 1996. A clearing procedure as a niching method for genetic algorithms. In: *Proceedings of the IEEE International Conference on Evolutionary Computation, May 20-22, 1996, Nagoya, Japan*. Los Alamitos, CA: IEEE Computer Society Press, 798-803.

Poli, R. 2005. Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms. In: *Proceedings of the 8th workshop on the foundations of genetic algorithms, Jan 5-9, 2005, Aizu-Wakamatsu City, Japan*. New York: Springer Publishing Company, 132-155.

Poncela, A. *et al.* 2002. Efficient integration of metric and topological maps for directed exploration of unknown environments. *Robotics and Autonomous Systems*, **41**(1), 21-39.

Quinlan, J.R. 1983. Learning efficient classification procedures and their application to chess endgames. In: Michalski, R.S., Carbonell, J. and Mitchell, T.M. (eds.), *Machine learning: an artificial intelligence approach*. Palo Alto, CA: Tioga Press, 463-482.

Raghuwanshi, M.M. and Kakde, O.G. 2006. Genetic algorithm with species and sexual selection. In: *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems, Jun 7-9, 2006, Bangkok, Thailand*. Piscataway, NJ: IEEE Press, 1-8.

Rajapakse, A., Furuta, K. and Kondo, S. 2002. Evolutionary learning of fuzzy logic controllers and their adaptation through perpetual evolution. *IEEE Transactions on Fuzzy Systems*, **10**(3), 309-321.

Ranganathan, A., Menegatti, E. and Dellaert, F. 2006. Bayesian inference in the space of topological maps. *IEEE Transactions on Robotics*, **22**(1), 92-107.

Rasheed, K. 1998. *GADO: A Genetic Algorithm for Continuous Design Optimization.* PhD thesis, Department of Computer Science, Rutgers University.

Rathbun, D. *et al.* 2002. An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments. In: *Proceedings of the 21st Digital Avionics Systems Conference, Oct 29-31, 2002, Irvine, Canada.* Piscataway, NJ: IEEE Press, 8D21-8D212.

Reed, P.M., Minsker, B.S. and Goldberg, D.E. 2001. The practitioner's role in competent search and optimization using genetic algorithms. *World Water and Environmental Resources Congress, May 20-24, 2001, Orlando, Florida.*

Remolina, E. and Kuipers, B. 2004. Towards a general theory of topological maps. *Artificial Intelligence*, **152**(1), 47-104.

Ren, J., McIsaac, K.A. and Patel, R.V. 2006. Modified Newton's method applied to potential field-based navigation for mobile robots. *IEEE Transactions on Robotics*, **22**(2), 384-391.

Ren, J. *et al.* 2007. A potential field model using generalized sigmoid functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **37**(2), 477-484.

Rogers, A. and Prügel-Bennett, A. 1999a. Genetic drift in genetic algorithm selection schemes. *IEEE Transactions on Evolutionary Computation*, **3**(4), 298-303.

Rogers, A. and Prügel-Bennett, A. 1999b. Modelling the dynamics of a steady state genetic algorithm. In: Banzhaf, W. and Reeves, C. (eds.), *Foundations of Genetic Algorithms 5*. San Mateo, CA: Morgan Kaufmann Publishers, 57-68.

Rosell, J. and Iñiguez, P. 2002. Hierarchical and dynamic method to compute harmonic functions for constrained motion planning. In: *Proceedings of the International Conference on Intelligent Robots and Systems, Sep 30-Oct 4, 2002, Lausanne, Switzerland*. Los Alamitos, CA: IEEE Computer Society Press, 2334-2340.

Rothlauf, F. 2002. *Representations for Genetic and Evolutionary Algorithms*. New York: Springer-Verlag Publisher.

Rowe, J. *et al*. 2004. Properties of gray and binary representations. *Evolutionary Computation*, **12**(1), 47-76.

Rudolph, G. 1994. Convergence analysis of canonical genetic algorithms. *IEEE Transaction on Neural Network*, **5**(1), 96-101.

Ryu, B.S. and Yang, H.S. 1998. An enhanced topological map for efficient and reliable mobile robot navigation with imprecise sensors. *Journal Robotics and Computer Integrated Manufacturing*, **14**(3), 185-197.

Ryu, B.S. and Yang, H.S. 1999. Integration of reactive behaviors and enhanced topological map for robust mobile robot navigation. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, **29**(5), 474-485.

Sano, Y. and Kita, H. 2002. Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation. In: *Proceedings of the IEEE Congress on Evolutionary Computation, May 12-17, 2002, Honolulu, HI, USA*. Piscataway, NJ: IEEE Press, 360-365.

Santos, V., Castro, J. and Ribeiro, M.I. 2000. A nested-loop architecture for mobile robot navigation. *International Journal of Robotics Research*, **19**(12), 1218-1235.

Sareni, B. and Krahenbuhl, L. 1998. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, **2**(3). 97-106.

Sasaki, H. *et al.* 2006. Steady-state genetic algorithm for self-localization in illuminance measurement of a mobile robot. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Oct 8-11, 2006, Taipei, Taiwan.* Piscataway, NJ: IEEE Press, 1897-1902.

Schnier, T. and Yao, X. 2000. Using multiple representations in evolutionary algorithms. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jul 16-19, 2000, La Jolla, CA, USA*. Piscataway, NJ: IEEE Press, 479-486.

Schumacher, C., Vose, M.D. and Whitley, L.D. 2001. The no free lunch and description length. In: *Proceedings of the Conference on Genetic and Evolutionary Computation, Jul 7-11, 2001, San Francisco, CA, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 565-570.

Sedighi, K.H. *et al.* 2004. Autonomous local path planning for a mobile robot using a genetic algorithm. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Jun 19-23, 2004, Portland, Oregon, USA.* Piscataway, NJ: IEEE Press. 1338-1345.

Shah-Hamzei, G.H. and Mulvaney, D.J. 2000. Intelligent process control using fuzzy ITI. *Neural Computing and Applications*, **9**(1), 12-18.

Shi, Z., Cui, J. and Zhang, H. 2004. Parameter design of integrated altitude control system using steady state genetic algorithm. In: *Proceedings of the IEEE 5th World Congress on Intelligent Control and Automation, Jun 15-19, 2004, Hangzhou, China.* Piscataway, NJ: IEEE Press, 2091-2094.

Shi, Z. *et al.* 2004. Comparison of steady state and elitist selection genetic algorithms. In: *Proceedings of the International Conference on Intelligent Mechatronics and Automation, Aug 26-31, 2004, Chengdu, China.* Piscataway, NJ: IEEE Press, 495-499.

Shimoda, S., Kuroda, Y. and Iagnemma, K. 2007. High speed navigation of unmanned ground vehicles on uneven terrain using potential fields. *Robotica*, **25**(4), 409-424.

Siegwart, R. and Nourbakhsh, I. 2004. *Introduction to Autonomous Mobile Robots.* Cambridge, MA: MIT Press.

Sillitoe, I.P.W. *et al.* 2001. Experiments in robust bistatic sonar object classification for local environment mapping. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 21-26, 2001, Seoul, Korea.* Los Alamitos, CA: IEEE Computer Society Press, 2147-2152.

Simon, D. and Isik, C. 1993. Suboptimal robot joint interpolation within user-specified knot tolerances. *Journal of Robotic Systems*, **10**(7), 889-911.

Singh, G. and Deb, K. 2006. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: *Proceedings of the Conference on Genetic and Evolutionary Computation, Jul 8-12, 2006, Seattle, Washington, USA.* San Francisco, CA: Morgan Kaufmann Publishers, 1305-1312.

Smierzchalski, R. and Michalewicz, Z. 2000. Modeling of ship trajectory in collision situations by an evolutionary algorithm. *IEEE Transactions on Evolutionary Computation*, 4(3), 227-241.

Smierzchalski, R. and Michalewicz, Z. 2006. Path planning in dynamic environments. In: Patnaik, S., Jain, L.C., Tzafestas, S.G., Resconi, G. and Konar, A. (eds.),

Innovations in Robot Mobility and Control. Berlin: Springer-Verlag Publisher, 135-154.

Smith, J. 2007. On replacement strategies in steady state evolutionary algorithms. *Evolutionary Computation*, **15**(1), 29-59.

Smith, R.C. and Cheeseman, P. 1987. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, **5**(4), 56-68.

Sniedovich, M. 1988. A multiobjective routing problem revisited. *Engineering Optimization*, **13**(2), 99-108.

Snyman, J.A. 2005. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. New York: Springer Publishing Company.

Sokolov, A. and Whitley, D. 2005. Unbiased tournament selection. In: *Proceedings of the Genetic and Evolutionary Computation Conference, Jun 25-29, 2005, Washington, D.C., USA*. New York, NY: Association for Computing Machinery, 1131-1138.

Sokolov, A., Whitley, D. and da Motta Salles Barreto, A. 2007. A note on the variance of rank-based selection strategies for genetic algorithms and genetic programming. *Journal Genetic Programming and Evolvable Machines*, **8**(3), 221-237.

Soltani, A.R. *et al*. 2002. Path planning in construction sites: performance evaluation of the Dijkstra, A*, and GA search algorithms. *Advanced Engineering Information*, **16** (4), 291-303.

Soska, G.V. 1985. Third generation robots: their definition, characteristics, and applications. *Robotics Age*, **7**(8), 20-29.

Spears, W.M. 1992. Crossover or mutation? In: Whitley, D. (ed.), *Foundations of Genetic Algorithms 2*. San Francisco, CA: Morgan Kaufmann Publishers, 221-237.

Spears, W.M. 1997, Recombination parameters. In: Baeck, T., Fogel, D. and Michalewicz, Z. (ed.), *Handbook of Evolutionary Computation*, New York: Oxford University Press and Institute of Physics Publishing.

Spears, W.M. and De Jong, K.A. 1998. Dining with GAs: operator lunch theorem. In: *Proceedings of the 5th Workshop on Foundation of Genetic Algorithms, Jul 22-25, 1998, Madison, WI, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 240-256.

Srinivasa, K.G., Venugopal, K.R. and Patnaik, L.M. 2007. A self-adaptive migration model genetic algorithm for data mining applications. *Information Sciences,* **177**(20), 4295-4313.

Stachniss, C. and Burgard, W. 2002. An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In: *Proceedings of the International Conference on Intelligent Robots and Systems, Sep 30-Oct 4, 2002, Lausanne, Switzerland.* Los Alamitos, CA: IEEE Computer Society Press, 508-513.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 8-13, 1994, San Diego, CA, USA*. Los Alamitos, CA: IEEE Computer Society Press, 3310-3317.

Stentz, A. 1995. The focussed D* algorithm for real-Time replanning. In: *Proceedings of the International Joint Conference on Artificial Intelligence, Aug 20-25, 1995, Montreal, Canada*. San Francisco, CA: Morgan Kaufmann Publishers, 1652-1659.

Stoytchev, A. and Arkin, R.C. 2004. Incorporating motivation in a hybrid robot architecture. *Journal of Advanced Computational Intelligence and Intelligent Informatic*, **8**(3), 269-274.

Su, L. and Tan, M. 2005. A virtual centrifugal force based navigation algorithm for explorative robotic tasks in unknown environments. *International Journal of Robotics and Autonomous Systems*, **51**(4), 261-274.

Sugihara, K. and Smith, J. 1997. Genetic algorithms for adaptive motion planning of an autonomous mobile robot. In: *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, Jul 10-11, 1997, Monterey, CA, USA*. Washington, DC: IEEE Computer Society Press, 138-143.

Suzuki, H., Sawai, H. and Piaseczny, W. 2006. Chemical genetic algorithms - evolutionary optimization of binary-to-real-value translation in genetic algorithms. *Artificial Life*, **12**(1), 89-115.

Suzuki, T. *et al.* 2005. Goal-directed navigation strategy for a mobile robot under uncertain world knowledge. In: *Proceedings of the IEEE International Conference on Mechatronics and Automation, Jul 29-Aug 1, 2005, Niagara Falls, Canada.* Los Alamitos, CA: IEEE Computer Society Press, 741-746.

Swere, E., Mulvaney, D.J. and Sillitoe, I.P.W. 2004. Efficient incremental decision tree generation for embedded applications. In: *Proceedings of the IEEE International Conference on Cybernetics and Intelligent Systems, Dec 1-3, 2004, Singapore.* Piscataway, NJ: IEEE Press, 1101-1106.

Syswerda, G. 1989. Uniform crossover in genetic algorithms. In: *Proceedings of the 3rd International Conference on Genetic Algorithms, Jun 4-7, 1989, Fairfax, Virginia, USA*. San Francisco: Morgan Kaufmann Publishers, 2-9.

Syswerda, G. 1991. A study of reproduction in generational and steady-state genetic algorithms. In: Rawlins, G. (ed.), *Foundations of Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann Publishers, 94-101.

Tarokh, M. 2007. A genetic robot path planner with fuzzy logic adaptation. In: *Proceedings of the IEEE/ACIS International Conference on Computer and Information Science, Jul 11-13, 2007, Melbourne, Australia*. Washington, DC: IEEE Computer Society Press, 388-393.

Thomaz, C.E., Pacheco, M.A. and Vellasco, M. 1999. Mobile robot path planning using genetic algorithms. In: *Proceedings of the International Work-Conference on Artificial and Natural Neural Networks: Foundations and Tools for Neural Modeling. Alicante, Spain, Jun 2-4, 1999*. London: Springer-Verlag Publisher, 671-679.

Thrun, S. and Bucken, A. 1996. Integrating grid-based and topological maps for mobile robot navigation. In: *Proceedings of the 13th National Conference on Artificial Intelligence, August 1996, Portland, Oregon*. Menlo Park, CA: American Association for Artificial Intelligence, 944-950.

Thrun, S. *et al.* 1998. Map learning and high-Speed navigation in RHINO. In: Kortenkamp, D., Bonasso, R.P. and Murphy, R.R. (eds.), *AI-based Mobile Robots: Case Studies of Successful Robot Systems*. Cambridge, MA: MIT Press, 21-52.

Toffolo, A. and Benini, E. 2003. Genetic diversity as an objective in multiobjective evolutionary algorithms. *Evolutionary Computation*, 11(2), 151-168.

Trojanowski, K., Michalewicz, Z. and Xiao, J. 1997. Adding memory to the evolutionary planner/navigator. In: *Proceedings of the IEEE International Conference on Evolutionary Computation, Apr 13-16, 1997, Indianapolis, USA*. Los Alamitos, CA: IEEE Computer Society Press, 483-487.

Tu, J. and Yang, S. 2003. Genetic algorithm based path planning for a mobile robot. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Sep 14-19, 2003, Taiwan, China.* Los Alamitos, CA: IEEE Computer Society Press, 1221-1226.

Ulrich, I. and Borenstein, J. 1998. VFH+: reliable obstacle avoidance for fast mobile robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation, May 16-21, 1998, Leuven, Belgium*. Los Alamitos, CA: IEEE Computer Society Press, 1572-1577.

Ulrich, I. and Borenstein, J. 2000. VFH*: local obstacle avoidance with look-ahead verification. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 24-28, 2000, San Francisco, CA, USA*. Los Alamitos, CA: IEEE Computer Society Press, 2505-2511.

Urdiales, C. *et al.* 2003a. Hierarchical planning in a mobile robot for map learning and navigation. In: Maravall, D., Ruan, D. and Zhou, C. (eds.), *Autonomous Robotic Systems - Soft Computing and Hard Computing Methodologies and Applications.* New York, NY: Springer-Verlag Publisher, 165-188.

Urdiales, C. *et al.* 2003b. A hybrid architecture for autonomous navigation using a CBR reactive layer. In: *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology, Oct 13-17, 2003, Halifax, Canada.* Piscataway, NJ: IEEE Press, 225-232.

Urdiales, C. *et al.* 2006. A purely reactive navigation scheme for dynamic environments using Case-Based Reasoning. *Autonomous Robots*, **21**(1), 65-78.

Utgoff, P.E. 1988. ID5: an incremental ID3. In: *Proceedings of the 5th International Conference on Machine Learning, Jun 12-14, 1988, Ann Arbor, Michigan, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 107-120.

Utgoff, P.E. 1989. Improved training via incremental learning. In: *Proceedings of the 6th International Workshop on Machine Learning, Jun 26-27, 1989, Ithaca, New York, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 362-365.

Utgoff, P.E., Berkman, N.C. and Clouse, J.A. 1997. Decision tree induction based on efficient tree restructuring. *Machine Learning*, **29**(1), 5-44.

Vannoy, J. and Xiao, J. 2004. Real-time adaptive trajectory-optimized manipulator motion planning. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Sep 28-Oct 2, 2004, Sendai, Japan.* Los Alamitos, CA: IEEE Computer Society Press, 497-502.

Vaughan, R.T. *et al.* 2002. Lost: localization-space trails for robot teams. *IEEE Transactions on Robotics and Automation*, **18** (5), 796-812.

Vavak, F. and Fogarty, T.C. 1996. Comparison of steady state and generational genetic algorithms for use in non-stationary environments. In: *Proceedings of the IEEE International Conference on Evolutionary Computation, May 20-22, 1996, Nagoya, Japan.* Piscataway, NJ: IEEE Press, 192-195.

Vazquez-Martin, R. *et al.* 2006. Hybrid navigation guidance for intelligent mobiles. In: *Proceedings of the IEEE 63rd Vehicular Technology Conference, May 7-10, 2006, Melbourne, Australia.* Piscataway, NJ: IEEE Press, 2992-2996.

Vekaria, K. and Clack, C. 1998. Selective Crossover in Genetic Algorithms: An Empirical Study. *Lecture Notes in Computer Science*, 1498, 438-447.

Wang, H.J. *et al.* 2005. An improved path planner based on adaptive genetic algorithm for autonomous underwater vehicle. In: *Proceedings of the IEEE International Conference on Mechatronics and Automation, Jul 29-Aug 1, 2005, Niagara Falls, Canada.* Los Alamitos, CA: IEEE Computer Society Press, 857-861.

Wang, L.C., Yong, L.S. and Ang Jr. M.H. 2002. Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment. In: *Proceedings of the IEEE International Symposium on Intelligent Control, Oct 27-30, 2002, Vancouver, Canada.* Los Alamitos, CA: IEEE Computer Society Press, 821-826.

Wang, L.F., Tan, K.C. and Chew, C.M. 2006. *Evolutionary Robotics: From Algorithms to Implementations*. Singapore: World Scientific.

Wang, Y., Mulvaney, D.J. and Sillitoe, I.P.W. 2006. Genetic-based mobile robot path planning using vertex heuristics. In: *Proceedings of the IEEE International Conference on Cybernetics and Intelligent Systems, Jun 7-9, 2006, Bangkok, Thailand.* Piscataway, NJ: IEEE Press, 463-468.

Wang, Z.G. *et al.* 2005. Optimization of multi-pass milling using parallel genetic algorithm and parallel genetic simulated annealing. *International Journal of Machine Tools and Manufacture,* **45**(15), 1726-1734.

Watanabe, K. and Hashem, M.M.A. 2004. *Evolutionary Computations: New Algorithms and their Applications to Evolutionary Robots*. New York: Springer Publishing Company.

Wendt, D.A., Irwin, M.E. and Cressie, N. 2004. Waypoint analysis for command and control. *Naval Research Logistics*, **51**(8), 1045-1067.

Whitley. D. 1989. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: *Proceedings of the 3rd International Conference on Genetic Algorithms, Jun 4-7, 1989, George Mason University, Fairfax, Virginia, USA*. San Francisco, CA: Morgan Kaufmann Publishers, 116-123.

Whitley, D. and Kauth, J. 1988. GENITOR: a different genetic algorithm. In: *Proceedings of the 3rd Rocky Mountain Conference on Artificial Intelligence, Jun 13-15, Denver, Colorado, USA*. Menlo Park, CA: AAAI Press, 118-130.

Whitley, D., Rana, S. and Heckendorn, R.B. 1997. Representation issues in neighbourhood search and evolutionary algorithms. In: Quagliarelli, D., Periaux, J., Poloni, C., Winter, G., (eds.), *Genetic Algorithms in Engineering and Computer Science*. Chichester: John Wiley and Sons Ltd, 39-57.

Wiese, K. and Goodwin, S.D. 1998. Overview of selection schemes and a suggested classification. *Canadian Artificial Intelligence Conference, Jun 18-20, 1998, Vancouver, Canada.*

Wolpert, D. and Macready, W. 1997. No free lunch theorems for optimisation. *IEEE Transactions on Evolutionary Computation*, 1(1), 221-237.

Wu, W. and Ruan, Q. 2004. A gene-constrained genetic algorithm for solving shortest path problem. In: *Proceedings of the 7th international conference on signal processing, Aug 31-Sep 4, 2004, Beijing, China*. Piscataway, NJ: IEEE Press, 2510-2513.

Wu, W. and Ruan, Q. 2006. A hierarchical approach for the shortest path problem with obligatory intermediate nodes. In: *Proceedings of the 8th International Conference on Signal Processing, Nov 16-20, 2006, Guilin, China*. Piscataway, NJ: IEEE Press, 412-416.

Xiao, J. 1997. Evolutionary planner/navigator in a mobile robot environment. In: Bäck, T., Fogel, D. and Michalewicz, Z. (eds.), *Handbook of Evolutionary Computation*. New York: Oxford University Press and Institute of Physics Publishing.

Xiao, J. *et al.* 1997. Adaptive evolutionary planner/navigator for mobile robots. *IEEE Transactions on Evolutionary Computation*, 1(1), 18-28.

Xiao, J., Michalewicz, Z. and Zhang, L. 1996. Evolutionary planner/navigator: operator performance and self-tuning. In: *Proceedings of the IEEE International Conference on Evolutionary Computation, May 20-22, 1996, Nagoya, Japan*. Los Alamitos, CA: IEEE Computer Society Press, 366-371.

Xu, W.L. and Tso, S.K. 1999. Sensor-based fuzzy reactive navigation of a mobile robot through local target switching. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, **29**(3), 451-459.

Xu, W.L., Tso, S.K. and Fung, Y.H. 1998. Fuzzy reactive control of a mobile robot incorporating a real/virtual target switching strategy. *Robotics and Autonomous Systems*, **23**(3), 171-186.

Xu, Z.B. *et al*. 2003. Efficiency speed-up strategies for evolutionary computation: fundamentals and fast-GAs. *Applied Mathematics and Computation*, **142**(2-3), 341-388.

Yamauchi, B. 1997. A frontier-based approach for autonomous exploration. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 20-25, 1997, Albuquerque, New Mexico, USA*. Los Alamitos, CA: IEEE Computer Society Press, 146-151.

Yang, W., Watanuki, K. and Zhao, S. 2005. A quick intelligent control system for a mobile robot to avoid collision with moving obstacles. *Microsystem Technologies archive*, **11**(8), 569-576.

Yang, X., Moallem, M. and Patel, R.V. 2005. A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **35**(6), 1214-1224.

Yap, C.-K. 1987. Algorithmic motion planning. In: Schwartz, J.T. and Yap, C.-K. (ed.), *Advances in Robotics vol. 1: Algorithmic and Geometric Aspects of Robotics.* Hillsdale, NJ: Lawrence Erlbaum, 95-143.

Yildirim, M., Erkan, K. and Ozturk, S. 2006. Power generation expansion planning with adaptive simulated annealing genetic algorithm. *International Journal of Energy Research*, **30**(14), 1188-1199.

Zalama, E. *et al.* 2002. Adaptive behavior navigation of a mobile robot. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, **32**(1), 160-169.

Zelek, J. 1999. Dynamic issues for mobile robot real-time discovery and path planning. In: *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, Nov 8-9, 1999, Monterey, California, USA.* Piscataway, NJ: IEEE Press, 232-237.

Zeng, X. 2003. Evolution of the safe path for ship navigation. *Applied Artificial Intelligence*, **17**(2), 87-104.

Zhao, X.C. and Long, H.L. 2005. Multiple bit encoding-based search algorithms. In: *Proceedings of the IEEE Congress on Evolutionary Computation, Sep 2-5, 2005, Edinburgh, UK.* Piscataway, NJ: IEEE Press, 1996-2001.

Zhang, G. *et al.* 2006. A clustering based GA for multimodal optimization in uneven search space. In: *Proceedings of the IEEE 6th World Congress on Intelligent Control and Automation, Jun 21-23, 2006, Dalian, China.* Piscataway, NJ: IEEE Press, 3134-3138.

Zheng, C., Ding, M. and Zhou, C. 2003. Real-time route planning for unmanned air vehicle with an evolutionary algorithm. *International Journal of Pattern Recognition and Artificial Intelligence*, **17**(1), 63-81.

Zheng, C. *et al.* 2005. Evolutionary route planner for unmanned air vehicles. *IEEE Transactions on Robotics*, **21**(4), 609-620.

Zhu, A. and Yang, S.X. 2004. A fuzzy logic approach to reactive navigation of behavior-based mobile robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation, Apr 26-May 1, 2004, New Orleans, LA, USA*. Los Alamitos, CA: IEEE Computer Society Press, 5045-5050.

Zhu, R., Sun, D. and Zhou, Z.Y. 2007. Integrated design of trajectory planning and control for micro air vehicles. *Mechatronics*, **17**(4-5), 245-253.