Artificial Intelligence Tools for Path Generation and Optimisation for Mobile Robots

by

Mario Augusto Góngora Florián

A thesis submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy at the University of Warwick

Department of Engineering

University of Warwick

September 1998

Summary

The ultimate goal in robotic systems is to develop machines that learn for themselves based on experience. In order to achieve on-line learning some software tools are needed to allow the robots to continually adapt their behaviour in order to constantly optimise their performance. This thesis presents research work focused on path planning for mobile robots with the objective of generating optimal paths for any type of mobile robot in an environment containing any number of static obstacles of any shape.

The research specifically recognises that an optimal path can be defined according to several criteria including distance, time, energy consumption and risk. The easiest and most commonly used measure is to minimise distance, but this does not by itself optimise task performance, and the other criteria are generally far more important. Distance is used mainly because there is no direct method to optimise time, energy and risk as they depend on the characteristics of the robot and the environment. This is solved in this research by using a set of Artificial Intelligence tools working together to perform an optimisation process strictly on the criteria selected.

The path planning system developed consists of an original and novel two-stage process comprising generation followed by optimisation. Path generation is achieved using cellular automata whose behaviour has been determined by a genetic algorithm. A program called Rutar has been written in which the best behaviour found by the genetic algorithm is encoded, and it has been tested and shown to infallibly generate all the non-redundant paths between any two points around any obstacles. An interesting and valuable feature of Rutar is that the time taken to generate paths depends only on the amount of free space available in which the robot can move and therefore the more obstacles there are present, and hence the more complex the layout, the faster the execution time. The paths generated are sub-optimal solutions, which are then optimised according to the user's selection of a combination of Time, Energy, Distance and Risk criteria. The optimisation process is performed by another genetic algorithm. The original scheme used in this work allows any combination of all the desired criteria in a single optimisation process, allowing it to handle very complex non-linear problems.

All of the optimisation criteria can be used in situations where the environment and the robot are considered to be unchanged during the interval in which the robot moves. This optimisation can be performed either off-line or on-line. However, the ability of the developed system to generate and optimise the paths very fast provide an opportunity for dynamic path optimisation, which ultimately can lead to on-line learning. This potential of the tools developed for the path planning system is explored and recommendations for further exploitation are made.

To my wife Irney for her love and patience
To my parents for their support and the education they provided for me
To Susie, and her kind, for showing me how wonderful the world is
To God for creating such a complicated universe in which to research

Table of Contents

Acknowledgements vi

Declaration vii

Chapter 1 Introduction 1

1.1. Why optimise? 5

1.2. What to optimise? 8

1.3. Optimisation aspects considered in this thesis 17

Chapter 2 Background: Optimisation tools 20

2.1. Optimisation and learning 25

2.2. Optimisation tools as used in this work 27

2.3. Optimisation in robotics 29

2.4. Learning in robotics 32

2.5. Learning systems and optimisation 32

Chapter 3 Path generation 34

3.1. Development of the path generation system 38

3.1.1. Using cellular automata to generate paths 42

3.1.2. Finding cellular automata rules 46

3.1.2.1. Definition of the map and its characteristics 47

3.1.2.2. Definition of the cellular automata rules structure 50

3.1.2.3. Definition of the evaluation process 55
3.1.2.4. Definition of the genetic operations and termination criteria 64
3.2. Execution of the genetic algorithm 68
3.3. The final system 72

Chapter 4 Rutar 75
4.1. The best chromosome: Rules if death 78
4.2. Characteristics of Rutar 86
4.3. Use of Rutar in dynamic path generation 94

Chapter 5 Path optimisation 96
5.1. Path optimisation scheme 98
5.2. Generation of the initial population: the chromosomes 103
5.3. Optimisation process and optimisation criteria 108
5.4. Experimental results 116

Chapter 6 Analysis of results in path optimisation 126
6.1. Verification of results and the computers performance 128
6.2. Optimisation criteria results 136
6.3. The chromosome structure and path optimisation 139
6.4. Rutar and the optimisation process 140
6.5. Fitness in genetic algorithms 144

Chapter 7 Conclusions and Recommendations for further work 149
7.1. Conclusions 150
7.2. Recommendations for further work 157
7.2.1. On-line application of Rutar and the path optimisation algorithm 158

7.2.1.1. Hierarchical control 159
7.2.1.2. Simulation aspects 163
7.2.2. Self simulator 164
7.2.2.1. Structure of the mobile robot 165
7.2.2.2. Possible assessment method 167
7.2.2.3. Modularity provided by the self-simulator 169
7.2.2.4. On-line learning 169
7.2.3. Trade-offs for On-line applications 170

References **172**

Appendix A Development of Rutar **183**

Appendix B Rutar executable in disk **191**

Appendix C Structure of the optimisation genetic algorithm **193**

Appendix D Simulator for optimisation tests **196**

Appendix E Optimisation results **214**

Appendix F Self simulator **217**

Acknowledgements

I am very grateful to my supervisors Dr. T.C. Goodhead and Dr. E.L. Hines, for the advice and help they provided during the work in this research.

I would like to thank Colciencias for the financial support they provided during the initial years of this work. And the Pontificia Universidad Javeriana for the time they allowed me over the duration of this research, especially the Department of Electronics Engineering, which provided valuable facilities to enable the work to be performed. I would also like to thank my colleagues that permitted me to take the necessary time to work on my research without moaning too much.

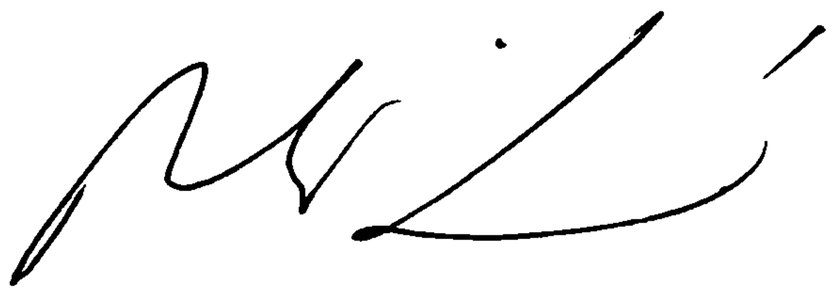
Declaration

The accompanying dissertation entitled "Artificial Intelligence tools for Path Generation and Optimisation for Mobile Robots " is submitted in support of an application for the degree of Doctor of Philosophy at the University of Warwick.

The observations and interpretations described in this thesis are those of the author, except where acknowledgement has been made to results and ideas previously published.

None of the work has been, or is being, submitted for any other degree or diploma to this or any other institution. Part of this work has formed the basis of several conference papers.

I hereby declare that the above statements are true.



Mario Augusto Góngora Florián

September 1998

Chapter 1

INTRODUCTION

The state of the art in robotic systems is well advanced. Many Industrial applications originally envisaged for robots have already been implemented, as well as new applications that once were thought to be impossible. One of the issues to consider now, in some of these applications, is how to optimise the performance of the robot in order to obtain the maximum benefits from using this technology, or even to just make a new application to be performed by a robot economically viable.

Being able to perform a given task with a robot is not very effective if it does not clearly out-perform the previous method such as dedicated automatic machines or manual techniques, in terms of speed, costs, etc. Thus the aspect of optimisation becomes very

important. Maximising the speed, minimising the energy-consumption or the risk involved can all make the difference in terms of whether a robotic system is feasible or not in the performance of a task, compared with other techniques. It is very important to identify the most critical variables to optimise depending on the application and situation of the system, thus the evaluation criteria for the improvement in efficiency that the optimisation process is going to produce, is very important.

In some cases, the optimisation process can be carried out beforehand, when planning the task. This is called *off-line* optimisation. The problem with this kind of optimisation is that it is very difficult to obtain a significant benefit when compared to dedicated machine performance, since any specialised machine can be efficiently optimised on the basis of its design. Thus, normally the benefit obtained is only marginal. When compared to a human operator based task, *off-line* optimisation can yield very high margins of improvement in some cases, especially when it is a repetitive task with little opportunity for creativity.

The other kind of optimisation is the one carried out during the actual performance of the task and is referred to as *on-line* optimisation. This is much more difficult to achieve, but it usually yields a much higher gain in performance in comparison with dedicated machines or human operators. This occurs because it involves *adaptability*, which is the difficult part to realise. Dedicated machines have far fewer adaptable properties than robots, and are also less adaptable than human operators. The problem with human operators is that although they are highly adaptable, they are prone to distraction and boredom, whereas robots are immune to this.

In the robotics area we can define two main types of robots, manipulators and mobile robots. This research will focus on mobile robots. Task optimisation for manipulators, which move in a confined 3 dimensional work envelope, has been studied by a number of researchers [Abe, et. al.] [Cao, et. al.] [Katoh, et. al.] [Rao, et. al.] and some optimisation techniques have been established. In the case of mobile robots, which typically move in an unconfined and often unstructured environment, the problems of achieving 100% reliable task execution have yet to be solved and so optimisation has not been studied to such a great extent.

In the field of Mobile Robots one of the main tasks is path planning. This aspect directly affects the level of autonomy of the robot. The idea of an autonomous mobile robot can be considered only after the robot is at least capable of generating its path. If in addition, the robot is able to optimise its path, it will have a higher degree of autonomy. The less interaction a robot has with a human operator, the more autonomous it is.

The generic optimisation methods presently available allow for very complex and efficient off-line optimisation procedures in some robotic tasks, for instance when using manipulators in certain manufacturing applications, but generally not in connection with mobile robots. It is also possible to perform some on-line optimisation for well defined tasks. Applying the techniques that are going to be presented in this thesis, the optimisation process can be taken a step beyond what is currently possible, where a given Artificial Intelligence (AI) tool will manipulate another Artificial Intelligence tool that in turn will

perform the optimisation process itself. This can be seen as generating a hierarchic AI system, which will contain the combined relative strengths of the AI tools in each of its levels. The AI tools that are going to be used in this work are presented in Chapter 2.

The scheme used in this work for combining AI tools into an optimisation system will allow it to handle complex non-linear problems. These techniques will also be the basis for on-line optimisation and to handle the system's own optimisation process in order to adapt it to a changing environment in real time.

Figure 1.1 shows a diagram of the structure of the hierarchic AI system. The lower level AI tools directly optimise the process. The higher level AI tool monitors the activity of the lower level tools and the behaviour of the system, by comparing the intended action of the control system (performed by the lower level) with the actual behaviour of the system. The higher level AI tool can thus fine-tune the action of the lower level tools. The hierarchy in a system as shown in figure 1.1 does not imply that some AI tools are more important hierarchically than others; the hierarchy is in terms of what type of task each tool performs. Any AI tool could be used at any level of the hierarchy, and for example, making a decision is higher in the hierarchy than controlling a motor.

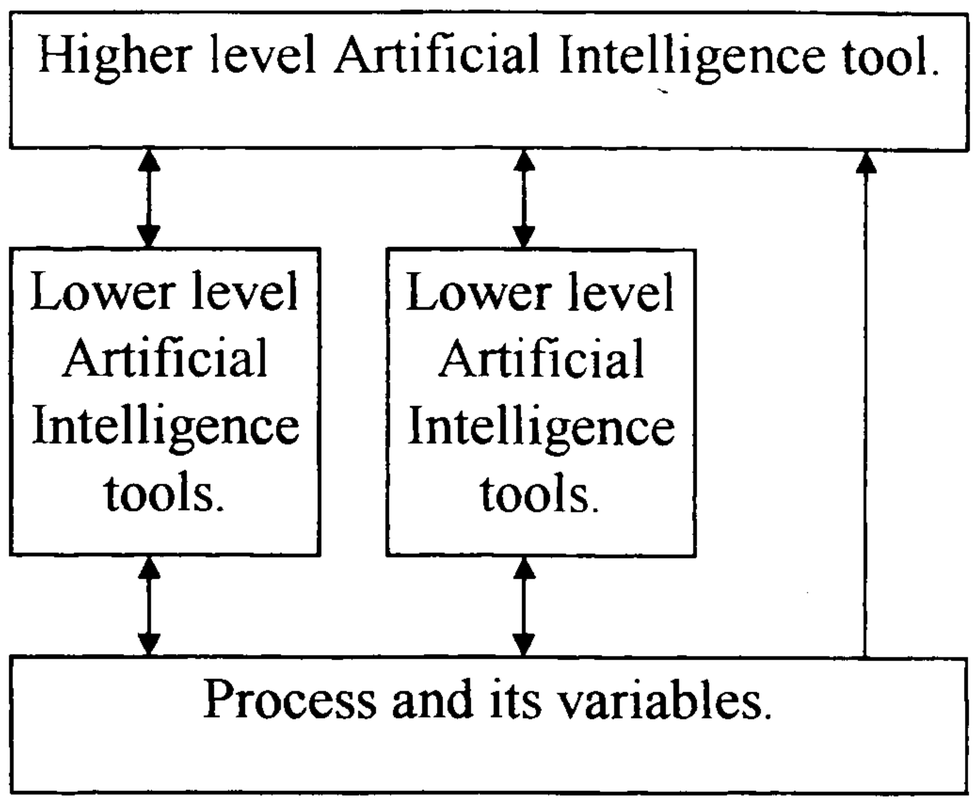


Figure 1. 1 A hierarchic system.

This thesis presents the work done in researching and developing novel schemes for tools for the optimisation of the tasks performed by mobile robots, and further, to allow the robot to update its optimisation schemes periodically depending on the situation. The ability to optimise a task and to further update the optimisation method autonomously is referred to in this thesis as *On-line learning*. The results presented in this thesis will provide valuable tools for developing *On-line learning* systems for robots.

1.1. Why optimise?

The main reason for trying to optimise the task performed by a robot is, as mentioned earlier, to clearly out-perform the alternative methods used for a given task. Additionally, in robotics the optimisation of a task is an important part of the programming of the robot since, in contrast to a dedicated automatic machine, a robot can do many things but in general none of them can be done in the optimum way. The reason can be seen to be a

consequence of the basic definition of an industrial robot "... a multifunctional programmable system ...", a penalty for multiple functionality and versatility being that it is sub-optimal for most tasks. This is also the case with human beings, which are capable of doing many of the tasks other animals do, but usually in a less efficient way than more specialised animals can.

To make up for this disadvantage in robots, the optimisation of the task must always be an important part of the programming. This comes from the fact that the sub-optimal characteristics of the robot are due to the hardware and mechanical design of a multi-functional system, that cannot be optimised (during its construction) for all the tasks that it will be able to do. This is especially true since the tasks that robots perform are not defined by their manufacturers but by the final users. Thus the software, being the only part of the system that can be changed by the user at the speed, cost and convenience that the applications require, has to be the component of the robot that must deal with the characteristics of the different tasks, within the limits introduced by the hardware or other parts of the system. In this way, by optimising the software, the process temporarily adapts the robot to the task it is performing at that time so that its efficiency is enhanced. As mentioned earlier, the optimisation process can take place off-line, while the robot is being programmed; or on-line while the robot is performing its work.

Once optimisation is adopted as a routine part of robot programming, several interesting features can be seen to be specific to the optimisation in robotics compared to optimisation of other less versatile systems:

- In off-line optimisation, the process can be compared to the design optimisation of a dedicated automatic system; with the advantage that the optimisation is performed in the software, thus it is quickly and easily changed or tested.

- An optimisation process in a dedicated automatic system makes it even less adaptable to other possible applications, while it does not limit future re-adaptations of the robot.

- On-line optimisation is possible in robots, and depending on the method, it can potentially be quite simple, and it might also make a significant improvement to performance. In dedicated automatic machines it is very seldom possible, and when possible, it will generally only yield marginal improvements because the machine is already designed for the task.

- On-line optimisation in robotic systems can be set to a continuous automatic mode so that it can adapt continuously to gradual changes in the system, or adapt periodically to unexpected changes in the characteristics of the whole system or environment. This is one of the most important features that will be explored in the recommendations for further work.

- In On-line optimisation processes, other factors, which are not directly concerned with the robot itself, can be included. For instance the optimisation of the space used in a packing task can be performed by the robot's control computer. In contrast, in a

dedicated automatic packing machine this optimisation has to be done as part of the process configuration and has to involve many other subsystems so that the packing machine can optimise the space available.

1.2. What to optimise?

The optimisation processes is a complicated task, therefore the effort should be focused on the most important criteria to optimise, i.e. where to obtain the highest gain.

In general, for robotics applications, optimisation should be focused on the variables that have the most influence on the main process (or processes) of a task. Depending on the application and the type of robot, these aspects can involve the accuracy, distance, time, energy, and risk associated with the task. For mobile robots the last three aspects can be considered to be the most important. Although distance travelled by the robot can readily be seen to be significant, as will be explained later in this section and in more detail in the next chapter, it is not a good optimisation criterion to use in mobile robots because it does not, by itself, affect task performance. Variables to be considered for optimisation include :

- Time is the most obvious variable to optimise in a process because for many applications, process time is the main measure of the efficiency of a process. In mobile robots time is normally assumed to be the travel time between two points.

- Energy is also an obvious variable to optimise. In industry, energy is costly and also an important part of the measurement of how environmentally friendly the process is. In autonomous mobile robots, energy is even more critical since, unlike manipulators, mobile robots invariably use electrical energy stored in batteries. This makes energy a vital and limited resource, thus it is very important to optimise it.

- Risk is a difficult variable to assess. It can imply risk associated with the reliability of the system, the rate of error, the quality of the products, or even in terms of physical damage to the robot. Risk is very seldom considered in the optimisation process. But when its impact is analysed, it becomes apparent how important it is; as will be seen later in section 1.3.

The optimisation of these variables depends directly on the characteristics of the robotics system and is affected indirectly by the task (application).

There are other aspects that may be optimised, which are more directly affected by the task and the characteristics of the process, and are therefore more suitable for off-line optimisation. For example, in the case of optimising the efficient use of space in packing or storing, the minimum space needed can best be determined during the design of the storing process if the parts to be stored are known, and the order in which they reach the storage area or container can be determined in advance. Although these aspects can best be optimised during the design of the process, before programming the robot, some on-line optimisation can be useful in some cases, such as in packing randomly shaped parts. There

is a lot of work to be done in all these areas and many other aspects of optimisation. As was stated at the beginning of this chapter, optimisation will be considered in relation to mobile robots.

Path Optimisation

For mobile robots there are several optimisation problems that are critical and complicated. Path optimisation is usually the most important of these because the chosen path affects all three of the previously identified criteria. A great deal of work has been done on this by the research community, and although important results have been obtained, the optimisation techniques are not yet fully developed. Normally, the variable optimised is distance, which is not a very complex task, and it is often assumed that by optimising distance, time and energy are also automatically optimised; this is not always true. The classical problem in mobile robots is to optimise the path between two points, which have obstacles that prevent a simple straight-line path from being used between the start and destination points. If minimum distance were used as the optimisation criterion, the solution would consist of a path composed of a series of straight lines which avoid obstacles by a specified distance, connected by sharp corners as seen in Figure 1.2.

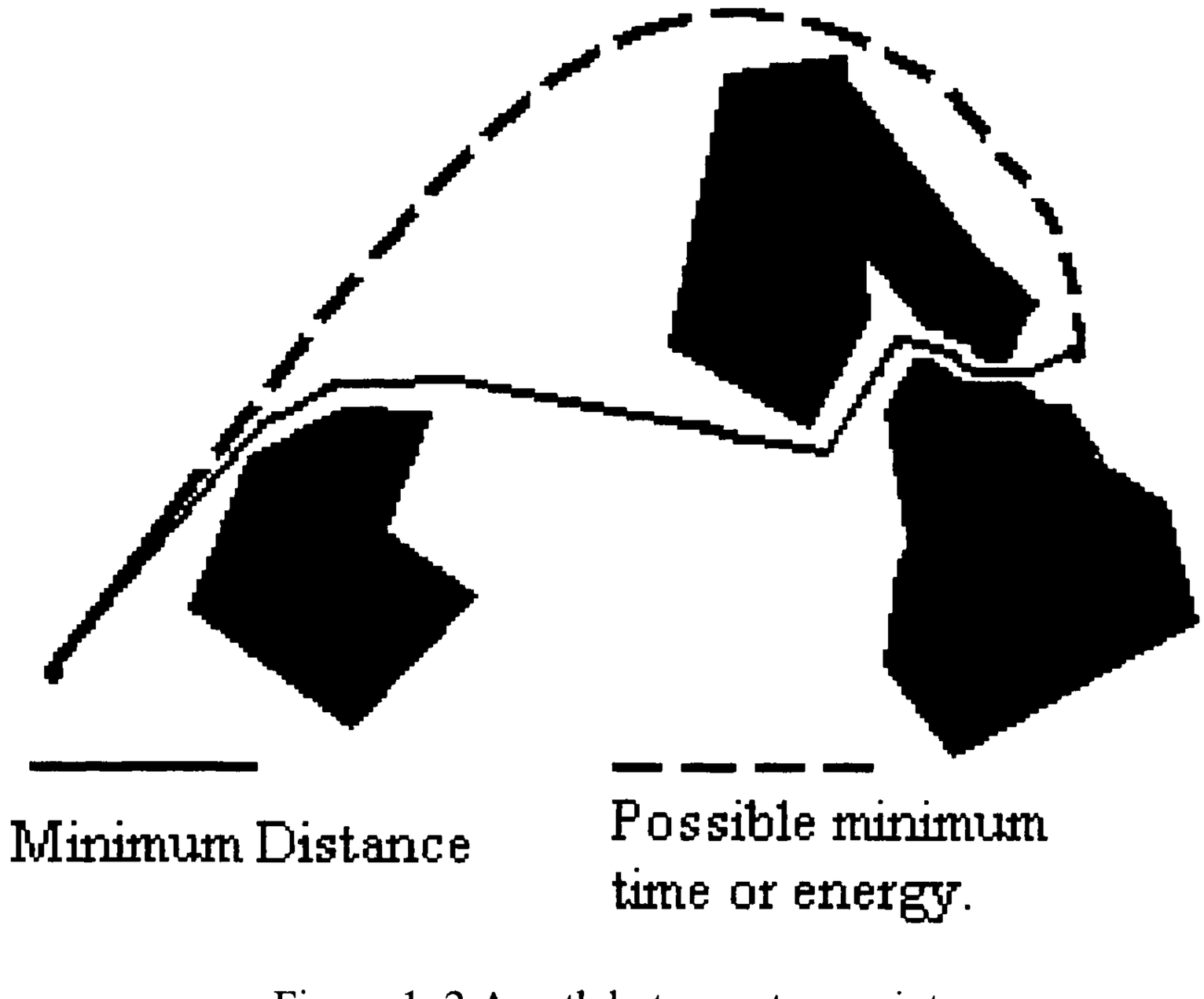


Figure 1. 2 A path between two points.

As this path between start and destination has discontinuities at the corners, when executing the task in practice the robot has to approximate it in one of two main ways:

- It can move from corner to corner and stop very briefly at each one to change orientation. It has to stop since if it tried to change direction while moving, the rate of change of direction would need to be infinite, and is thus impossible in practice. This results in a very accurate but jerky movement, and precisely because of the starting/stopping nature of the movement, the travel time and the energy consumption could be far from optimal. Depending on the characteristics of the robot, the minimum distance path executed in this way, can be very close to or very far from the optimal

path in terms of energy and travel time.

- The robot can round off the corners to make the path continuous, and thus make it possible to execute without stopping. This would result in a smooth path with no sharp corners and it could be more energy efficient due to less acceleration cycles, but the actual route would not be the same as planned which means that the robot could be at risk of colliding with obstacles.

The relationship between distance and travel time, if distance is minimised, would imply that, if the mobile robot chooses to stop at the corners, re-orientates itself and continue with the other straight line path, it would take time to stop, change direction and accelerate. The three of the previous actions require time to execute in practical systems due to inertia and the finite levels of power (and therefore torque) available. If the mobile robot chooses to approximate the path to segments connected through rounded corners, the maximum speed that the robot would be able to maintain at the curves is limited depending on the radius of the curve and the maximum force available to keep it from skidding off the intended curved path or to prevent it from overturning. In both cases the average velocity of the mobile robot throughout the path will be lower than the average possible if the robot travels along a longer but continuous and much smoother path. This implies that most of the time a longer path can be quicker, in terms of travel time, than the shortest one.

Energy optimisation

With respect to energy, a robot uses energy to overcome friction when travelling at constant speed, to produce the necessary forces to accelerate and decelerate, and to change direction. Once a given mechanical design is being used, the source of energy consumption that can be minimised is the acceleration (and deceleration and direction changes). Dynamic friction is fairly independent of the path unless the robot travels at such high speeds that aerodynamic drag becomes significant, which is not the case in most applications. Therefore, if accelerations can be minimised (usually using smooth paths) the energy consumption can be minimised. From this, it is quite obvious that a path that repeatedly requires acceleration and deceleration will consume a substantial amount of energy, while a smooth path, that can be performed at constant speed and with long radii curves may be more energy efficient.

Even in trivial cases, where optimising distance could seem to imply that energy and travel time have been optimised, the assumption can still be wrong. For example, in a problem where the robot has to go from one point to another where there are no obstacles in between, and a single straight line is the shortest path, this does not necessarily mean that it is the best solution. If the orientation of the mobile robot at the starting point and the required orientation at the final point are not the ideal for a straight line as shown in figure 1.3, the optimal path can be something very different.

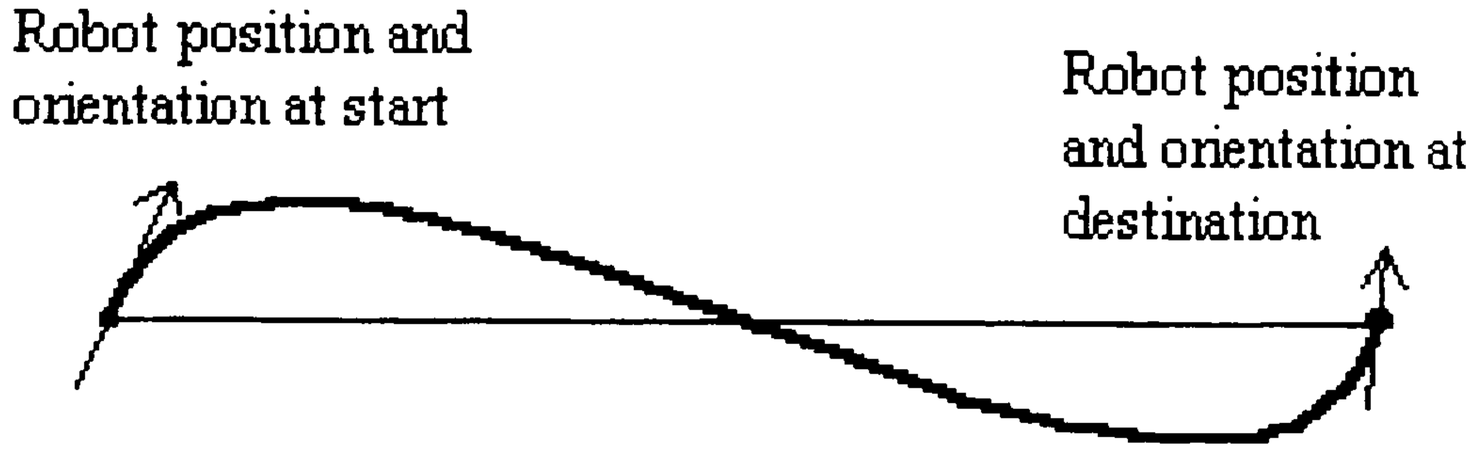


Figure 1. 3 Trivial Path.

The use of the path shown instead of a straight line depends on whether it takes more time or energy to rotate the robot in its place before starting the journey, and again at the end of the journey, compared to the extra energy or time that would be required to execute the longer path.

One of the arguments that could be put forward regarding the energy consumption in a path requiring repeated acceleration and deceleration is that, apart from the frictional losses, the energy needed to accelerate the mobile robot is stored as kinetic energy that in theory can be recovered in the deceleration stage. Of course this is true, and in theory, the same motors that are used to accelerate the robot can be used to generate energy when decelerating it. But in practice this is very difficult to implement for several reasons :

- Decelerating by absorbing energy makes it very difficult to control the deceleration rate.

- Electric motors tend to have different efficiency characteristics when used as generators, and even if this characteristic is fairly symmetrical, it still means that it would not be able to recover all the kinetic energy stored in the robot.

- The power electronics circuits that are used to drive a motor are not suitable, or are very inefficient, to recover energy. Thus it would be necessary to use extra circuitry to control the recovery of energy and this means more cost and more devices that dissipate energy, both quiescent and when acting as power switches/controllers.

- The batteries available at the moment are much more efficient at delivering current than at receiving charge, thus in a symmetrical acceleration-deceleration sequence, the batteries would probably not be able to receive all the energy available. This problem gets more critical as the decelerating rate increases, up to the case of an emergency stop where the energy is released too fast for the battery to accept any of it.

In any case, the optimisation of distance would not guarantee a path resulting in an optimal use of energy and neither would it guarantee the lowest travel time solution. The important point then is, that for time or energy optimisation, the solution is only guaranteed if these criteria are used directly rather than using indirect approximations such as assuming that minimum distance would result also in minimum time or energy.

Risk optimisation.

There is very little reported work on risk optimisation. An additional problem in risk optimisation is that in mobile robots it is not easy to describe or quantify risk. Additionally, risk can be seen more as a set of criteria rather than a single criterion. In the risk factor there can be various aspects such as:

- Risk of colliding with dynamically changing or not completely defined obstacles.

- Risk in the reliability of the robot.

- Risk of damage to the robot when travelling through dangerous areas, for instance where objects appear unexpectedly or where corrosive environments can damage the robot.

- Risk of not being able to reach the final destination or losing time if it has to retrace part of the journey, for instance when a shorter path is available but it may be unexpectedly blocked by an abandoned object, or if it is too narrow for two robots to fit in, and another robot is coming in the opposite direction and one of them has to turn back.

Thus, risk is a set of probabilistic variables rather than a deterministic variable like travel time and energy consumption. The scheme presented in this thesis allows for these types of

variables to be efficiently included with high versatility in a complex and composite optimisation process.

The problem is that optimising distance is much easier that optimising time and energy. There are no practical ways to optimise time and energy directly. In this thesis the problem is solved by using a set of AI tools working together to perform an indirect method of strictly optimising energy, travel time, risk, and any other criterion that might be defined for the system.

All these optimisation criteria can be considered for static paths, which are fixed during the time the robot moves, such as the path required to travel through a space with fixed obstacles. It is also possible to include these criteria for dynamic paths, which vary with time such as a path influenced by other traffic like people or additional robots. Clearly it is more complicated to optimise for dynamic paths, and while static path optimisation can be done either off-line or on-line, dynamic paths can only be optimised on-line.

1.3. Optimisation aspects considered in this thesis.

In this thesis, the optimisation process will be considered by starting with a review of the main techniques which are available for optimisation and learning. Then the development of a novel way for static path optimisation for mobile robots will be presented. Finally, an analysis of an extensive set of guidelines and tools to perform on-line optimisation will be

presented in the recommendations for further work.

The review presents aspects of various *Artificial Intelligence* techniques in optimisation and learning such as *Artificial Neural Networks, Genetic Algorithms, Genetic Paradigm, Cellular Automata,* etc. The inter-relation and differentiation between optimisation and learning with these tools, and some aspects in comparison to other mathematical and conventional techniques will also be considered.

The main body of the thesis is concerned with the development of a novel optimisation technique using static path optimisation and is illustrated by a case study. The optimisation systems use a two-pass (generation-optimisation) technique to find optimal paths for a mobile robot in a space containing static obstacles. The path generation is performed by a Cellular Automata system whose behaviour has been determined using genetic algorithms. The paths generated, are sub-optimal solutions that are further optimised according to the user's selection of a combination of Time, Energy, and Risk criteria. The optimisation process is performed by another genetic algorithm.

The analysis of the optimisation system developed shows that many AI techniques, even genetic algorithms, which are classically considered to be time-consuming when implemented as computer programs, can be used for on-line real time systems. They are also suited for continuous optimisation processes either alone or in conjunction with Artificial Neural Networks.

On the basis of the system developed, the analysis continues toward on-line optimisation for dynamic problems (such as dynamic paths). The additional difficulties, and more radical differences found in dynamic problems, as compared to static problems, are identified. With this, the optimisation techniques are analysed to find the necessary additional or different features that can make them suitable for real-time on-line optimisation in dynamic solutions.

Chapter 2

BACKGROUND:

OPTIMISATION TOOLS

Optimisation processes have always been a fascinating problem for people. Engineers have a great many mathematical techniques available to apply to these processes in a wide range of applications. For many years, traditional mathematical approaches were the main, or only, method to optimise systems, but the optimisation process could be applied only in the design or installation stages of non-adaptable systems. When adaptable systems began to be made available, adaptable solutions had to be developed and traditional mathematical approaches started to become too complicated [Rao, SS, et. al.] [Gill, MAC, et. al.]. These approaches not only became more complicated (due to non-linearity problems and

functions with complicated inverse forms) but also more solutions needed to be obtained faster.

The complex mathematics, in addition to the increasing number of *particularities* or *exceptions* that began to appear increasingly in every new mathematical solution developed, encouraged some people to look at less formal but more practical techniques. This resulted in the *Artificial Intelligence* (*AI*) techniques that are so popular at present for solving many engineering problems. In conjunction with modern formal mathematical techniques, AI has become an even more useful tool for optimisation since it has enabled mathematical methods to be used in a larger variety of problems.

An additional characteristic that was made more available, and powerful by the use of AI techniques, is the intrinsic suitability for Parallel processing that most AI tools exhibit, which is generally more difficult to achieve with other mathematical methods [Madan, S]. Although not all AI methods can be used in all applications with parallel processing schemes, they are generally suitable and this leads to the possibility of very rapid decision making and the opportunity for on-line applications [Cook, D.J. et al]. The efficiency for parallel processing execution that an optimisation tool exhibits, can be very important since it is an alternative method to increase dramatically the processing power of a system when a single processor system cannot be implemented to be faster [Boucher, A].

The term *Artificial Intelligence* is not universally accepted by every author, some use different terms for different techniques, such as *soft computing* or *intelligent systems*.

Without considering any other terms invalid nor absolute, but to use only a single terminology throughout this thesis, the term *Artificial Intelligence (AI)* is going to be used as a generic term. The main AI techniques considered in this work are:

- *Neural Networks*: are based on a network of very basic processing elements (that perform a simple analogue or discrete function) arranged in a similar way to the *neurones* in an animal's brain [Anderson, J.A.]. Although a single *neurone* is not very powerful, a large number of them combined with the correct internal functions results in a very powerful network in terms of being able to deal with non-linear problems for example. Since each *neurone* is a completely autonomous functional unit, *Neural Networks* are highly suitable for parallel or distributed computing techniques, where one or a proportion of the neurones can be evaluated in separate processors. *Neural Networks* are currently most widely used in pattern recognition and signal processing [Kosko, B.] [[Haykin, S.]. Some control applications have also been implemented in this way, as well as some limited decision making applications. These last two aspects suggest that they may be useful in robotics.

- *Genetic Algorithms*: are a non-exhaustive search technique based on the principle of evolution. The solution to a problem is found by evolving a number of initially non-optimal or random solutions to the desired optimal solution [Koza]. A typical Genetic Algorithm (*GA*) starts by defining assessment criteria for the solution it is searching for, then a number of sample problems are solved by an initial population of possible solutions. Their results are evaluated and according to their performance new possible

solutions are created from the previous ones. This process is repeated until a satisfactory solution is found. Only imagination can limit the types of problems that can be tackled by genetic algorithms. Thus, the main problem when developing a genetic algorithm is defining the assessment method for the trial solutions. The "solution evaluation" part of the algorithm is inherently parallel. But the creation of the new generation (solution set) needs a comparison between all (or a significant proportion) of the results, thus this part is not very parallel, but normally it is a very small fraction of the computing time used by a genetic algorithm. Therefore, genetic algorithms are highly suitable for parallel processing techniques in the "solution evaluation" part, which makes up most of the processing work-load.

- *Fuzzy Logic*: is a decision making scheme based on fuzzy parameters. This means that the numbers involved in the operations are grouped within a flexible range of values rather than exact figures or thresholds. It is mainly used in control applications where the control variables are described for instance as Hot, Very Hot, ... Too Hot, rather than specific temperature values. Using these types of fuzzy values in conjunction with a set of decision-making rules and a set of threshold functions, the control actions are taken [Kosko, B]. Fuzzy logic is not always suitable for parallel processing.

- *Cellular Automata*: simulate a colony of simple autonomous co-operative elements (similar to a colony of bacteria). Each element behaves autonomously depending on the state of its neighbours, one or more external variables, and a set of behaviour rules. Normally all elements have the same behaviour rules set. Again, similar to neural

networks, a single element is not very useful, but the statistical behaviour of the whole colony can be very complex and useful. Cellular automata are inherently parallel, and thus highly suitable for parallel processing, since each element is autonomous (such as bacteria in a colony), thus the execution of the behaviour of each automaton can be considered an independent process.

- *Expert Systems*: are "experience based" learning systems based on rules, data, and an inference engine to make decisions, provide reasons why a given decision was taken, and to re-adapt to new data. Expert systems can be used to manage very complex systems (in a similar way to the control of simpler systems with Fuzzy Logic), where the expertise of a human operator can be designed into the system [Nebendahl, D]. Expert systems are not easily implemented in parallel processing schemes.

- Other techniques include some hybrid schemes such as *Neural-Genetic* or *Neural-fuzzy* techniques, but since these types of AI schemes have been used only very recently, they can still be considered as a combination of the previously mentioned techniques. It is possible that in the near future some specific combinations will become particularly important to be considered as a new individual technique. For the moment, reviewing the basic techniques provides the necessary background for their application into problem solving systems.

Each of these methods are based on their own specific principle. Thus, a particular approach has to be used with each technique to make the most of it. This research presents

the work done in studying and developing *Artificial Intelligence* based tools for optimisation of the tasks performed by *Mobile Robots.*

The following sections present the optimisation processes that are going to be studied and the application of some of the presented AI tools in these processes.

2.1. Optimisation and Learning

To properly develop a system a clear definition of both the problem and the method has to be produced. Looking at the method, it is useful to start by making a clear distinction between two key terms, that is the difference between optimising and learning.

It is not easy to make this distinction since applying *learning* to a *non-living* system makes it very closely related to optimising because, at the present time, optimising is one of the main goals of *machine intelligence*. Additionally the terminology used in the literature often fails to differentiate these two [Geng, Z. et. al.] [Song, K.T. et. al.] [Sullivan, J.C.W. et. al.]. Different authors have different views on how to compare Optimisation with Learning. In living creatures there may be a majority of people who consider that learning is more important than optimising, that is, learning is the most important capability of the creature, and once it learns a task, it can optimise it with time. In machines (such a robots) this comparison is very different; a machine may never be assumed to learn anything and still be very efficient at optimising a task.

For this work, it is assumed that the optimisation process is dependent on the Learning capabilities of the system; where learning is defined in this thesis as meaning that the control computer of the robot has the capacity to adapt its optimisation scheme depending on the situation and the task it is to perform. This *Learning* capability is possible only in versatile intelligent machines, such as robots, since it depends on advantages such as:

- In off-line optimisations in robotics, the process can be compared to the design optimisation of a dedicated automatic system; with the advantage that the optimisation is performed in the software, thus it is easily changed while it is being tested.

- Optimising a process in a dedicated automatic system can make the system even less useful for other possible applications. The same case in a robot does not prevent later, and quick re-adaptation, since the optimisation is made predominantly in software.

- On-line optimisation is possible in robots, and depending on the method, it can be quite simple. It also can make a big improvement. In dedicated automatic machines it is very seldom possible, and when possible, it generally yields only a marginal gain because the machine was designed to be optimal at the time it was conceived.

- In On-line optimisation processes other factors that are normally out of the scope of the robot itself can be included. For instance, the optimisation of the space used in a packing task can be performed by the robot's control computer. While in an automatic

packing machine this optimisation has to be done as part of the process configuration and has to involve many other subsystems so that the packing machine can optimise the space properly.

- On-line optimisation in robotic systems can be set to a continuous automatic mode so that it can take account of changes such as the wear and tear of the system, or adapt continuously to uncontrolled changes in the characteristics of the whole system or environment. This is one of the most important features that will be explored for suggestions for further work, at the end of this thesis.

In the following sections the discussion will continue focusing on the optimisation tools, and learning considerations will be discussed in section 2.4 and 2.5.

2.2. Optimisation Tools as used in this Work

Each of the *Artificial Intelligence* techniques outlined in 2.0 can be used in robotics for different applications. For the purpose of illustrating the use that was given to each technique in this work, the main criteria for applying each one of them is explained below:

- Neural Networks: since they are very suitable for implementing complex non linear functions, the application given in this work was the simulation tasks, such as

simulating the mechanics of the robot. This part of the work is reported in section 7.2. where the simulator for the robot is presented.

- Genetic Algorithms, being a very efficient search method, were used for most of the optimisation processes either directly to find a solution, or indirectly to find an optimising scheme. In direct optimisation they were used to find an optimal solution within a set of possible solutions (presented in Chapter 5, on Path Optimisation). In indirect applications they were used to find the set of rules of behaviour for the Cellular Automata (presented in Chapter 3 on Path Generation)), and to maintain the Neural Networks in a continuous training mode (on the Self-Simulator presented in section 7.2.). One of the most useful characteristics of genetic algorithms, the capacity to avoid local minima, was exploited intensively in this work, as will be seen in chapters 3 and 5, where the genetic variety aspect is considered as a crucial variable in this research work.

- Cellular Automata, its important characteristic for this work is that a set of Cellular Automata generates a graphical pattern depending on its particular rules of behaviour. Since the rules of behaviour are the same for all the individuals, it is a very efficient way of compressing information (or programs) to perform relatively complex pattern generation. Since a path in a map can be represented in a graphical form, it was used to generate a pattern that would result in the path to be taken by a robot. This is presented in chapters 3 and 4.

Although all these techniques have been applied by many authors to problems in robotics,

the way they have been applied in this work is a novel approach in that they have been integrated as a set of techniques in one system to optimise and learn. In the following section a review of some applications of these AI techniques in optimisation and machine learning is presented.

2.3. Optimisation In Robotics

The easiest way to differentiate between optimisation and learning in robotics is to compare the type of operations performed in either case. In this work it is assumed that optimisation processes primarily imply a series of complex mathematical operations, managed by a given AI technique, that try to solve a problem in the best way. Learning, on the other hand, is considered to primarily imply a series of logical or decision-making operations that can, in turn, be altered with *experience*. In this way, learning can drive an optimisation process in the best way possible, to find the best solution.

In the field of path optimisation for mobile robots a great deal of work has been done by many researchers. From a review of the literature it can be seen that many complex problems, not previously solvable can now be tackled using AI tools. For instance in the work by [Abe, et. al.] the authors solve the problem of path optimisation in 3 dimensions for a redundant manipulator using a genetic algorithm. Without this tool, trying to optimise this problem using classical inverse kinematics mathematics would have been practically impossible. By using genetic algorithms the scheme works by searching for possible arm

configurations, and then evaluating them through direct kinematics, which is much easier and faster to evaluate than inverse kinematics. In the work by [Cao, et. al.] an approach to a time-optimal solution is presented for two manipulators, and in the work by [Wang, Q. et. al.] for an industrial manipulator, genetic algorithms are used to search for optimal joint configurations that are then evaluated through direct kinematics. Artificial Neural Networks are also used as in the work by [Martin et. al.].

Other research in path optimisation addresses issues like the order in which the destinations are going to be visited rather than the physical trajectory of the path itself, see for example the work by [Sharma et. al.]. By this approach, the accumulated distance, or travel time for a complex route is minimised, but this has no direct effect on the energy consumed or time spent in a given segment of the complete route. This type of optimisation problem is not going to be treated in this work, since they perform path optimisation but without considering any physical characteristic or limitations of the machines, and assume that the chronological order in which the destinations are visited defines the optimal path rather than accounting for energy consumption, physical trajectories, distances, etc.

More particularly [Kang, et. al] worked on path optimisation for mobile robots and for this used genetic algorithms. In this paper the authors assume that minimum distance is equivalent to the optimal path. [Moran et. al.] also assumes this, which further illustrates the problem of assuming equivalent distance optimisation with time or energy. The problem of assuming that minimum distance is equivalent to an optimal path is addressed

in this thesis, and a system developed based on an approach which does not rely on any such assumption is presented.

In [Desaulniers et. al.] the shortest path for a mobile robot is found by working with the constraint of the minimum path turning radius of the vehicle. This in fact could maximise energy consumption at the same time that it minimises distance, so the distance criterion for optimal path is again seen as very different (and possibly contradictory) to optimal energy. And in their work, [Lee J.H. et. al.] assume their minimum distance path to be the minimum time, their primary goal being to achieve a conflict free traffic control for multiple mobile robots. These examples further illustrate the point that the words *optimising a path* are an incomplete statement: it should be stated that a path is being optimised according to a given criteria, since optimisation is rarely a global process.

A reported research work that appears to truly posses the opportunity to minimise energy is an approach used by [IbarraZannatha et. al.] which generates the path from the skeleton of the free space representation of a map of where the robots are going to move. This means that the initial path would be following the centre of gravity of the free space rather that the shortest path. In the second stage of their optimisation, the segments are smoothed; although energy is not directly taken into account in the smoothing, at least it is not incremented by using sharp corners.

2.4. Learning In Robotics

The term *learning* very seldom appears in path planning for robotics, it is easier to find work covering *adaptable* path planning systems. In work like that from [Fujimori et. al] and [Stentz et. al] the system is capable of adapting its path planning to avoid *unexpected* obstacles. In his work, [Noborio] calls his algorithms, learning algorithms, since they adapt to the changing environment.

A basic approach for the design of a learning robot can be to provide the system with the ability to adapt automatically to the environment. This would mean, a robot that changes its characteristics (or parameters) with time, so that they would accommodate to the changes in their environment. This is the most basic form of machine learning and is the approach that most of the research works have taken.

When a robot adapts to changing situations, its performance is optimised to a certain level. This means that it will perform better in a wider range of situations as compared to a robot with no adaptive capabilities. Still, adapting does not necessarily mean that a machine is learning from experience.

2.5. Learning Systems and Optimisation

Taking the term *learning* in robotics a little further from immediate adaptability, it has to

be involved in a long-term adaptable scheme, which will learn with time, and keep the learnt characteristics until it is time to change them. An adaptable system usually has an immediate response to a change in its environment, this change is independent of previous changes, and can be reversed as soon as the environment reverts to its original state. A learning system should keep some sort of historical experience so that a change is not *forgotten* once a new change is made. Additionally, the change should not be immediate but progressive depending not only on the change in the environment but should also be a function of the way it changes. Furthermore, a learning system, should be the manager of the operative part of the system, not necessarily the operative part itself.

In this work, the methods presented for Optimisation and Learning are used in a novel and non-conventional way that produces a whole system or scheme, that can be used to create a learning Robot. This robot would have the capacity to optimise a given task under given criteria on-line, and by learning, the optimisation scheme itself will be kept updated so that it will optimise throughout the different situations that the robot can encounter.

Chapter 3

PATH GENERATION

To optimise a path, a possible path has first to be generated. That is, a way has to be found of getting from the starting point to the destination, without colliding with the obstacles. Although it is possible to generate an optimal path, it is much easier to generate a non-optimal path and then optimise it [Kang et. al.]. The reasons for this are mainly:

- Looking for a possible path avoiding obstacles is not necessarily harmonious with optimising it, especially if specific criteria need to be considered in the optimisation, such as energy or travel time.

- As will be shown later, normally there is not only one possible path to get from the

starting point to the destination. Normally various alternatives are possible and from these alternatives, one could be the one giving the optimal energy consumption, another the optimal travel time and yet another one giving the minimum distance. Figure 3.1 shows an example of a problem where there are various alternative routes to get from one point to another. Therefore the task of finding the path is not necessarily related to how optimal it is or to which parameter is going to be optimised.

- Most of the techniques used for path generation are different from those used for path optimisation, therefore including both at the same time is quite difficult, and if this is done, there is not complete control over the criteria for optimisation. Path generation techniques tend to be geometrical operations [Fujimori et. al.] [Janét et. al.] [DeLaRosa et. al.] [Cao et. al.] while path optimisation techniques tend to use more quantitative arithmetic operations [Cho et. al.] [Meng et. al.] [Moran et. al.] [Shiller et. al]. A few techniques that are suitable for both, like the minimum potential path [Koditschek et. al.] which is a combination of mathematical and geometrical techniques, can generate optimised paths. But these techniques require a great deal of computing power and the optimisation criteria are not very controllable, apart from minimising distance and some level of smoothness.

For the reasons above, in this work, a two step generation-optimisation scheme was used. In this chapter, the generation stage is presented.

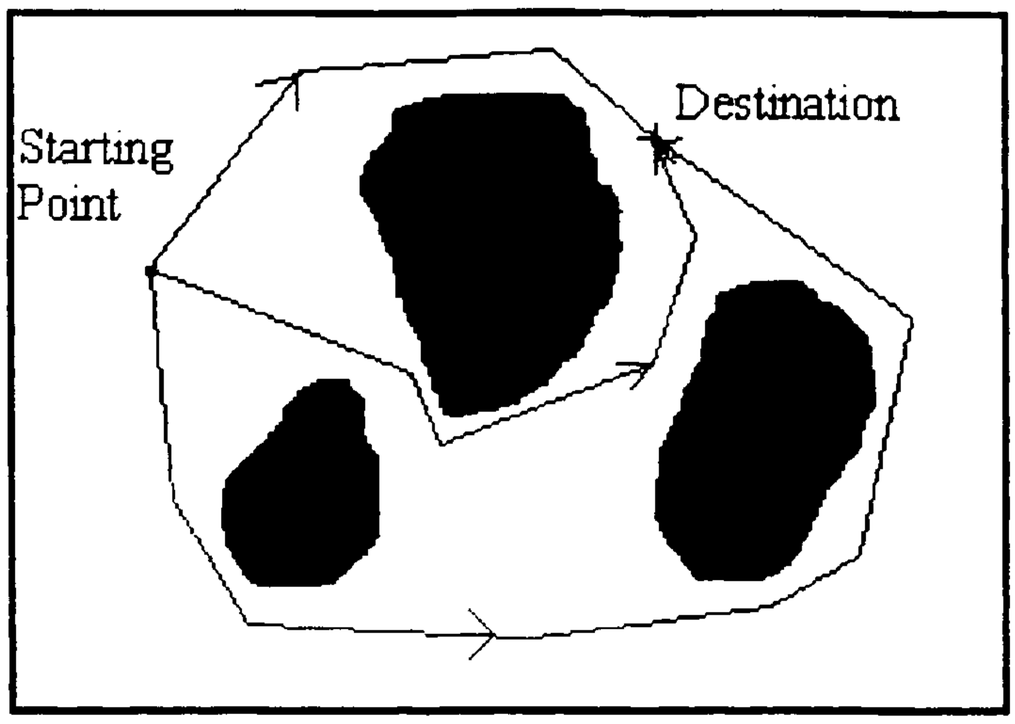


Figure 3. 1 Different alternative paths in a map

A common problem in path optimisation, ie. in minimising a given variable of the path, is the possibility of falling into a local minimum. One way in which this possibility can be minimised is to provide the optimisation system with all the non-redundant possibilities of paths that are practically feasible. That is, all those which take the robot from the origin to the destination without collisions, so that all the global possibilities are explored in the optimisation. From this, the main characteristics of the path generation systems can be determined as follows:

1. It needs to generate paths taking the robot from origin to destination.

2. It has to avoid collisions with obstacles meeting the requirements of 1. above.

3. It has to provide all the non-redundant paths that meet the requirements of 1. and 2. above.

The definition of non-redundant paths is important because otherwise the robot`s computer can spend a lot of time working on paths that being different, can lead to the same optimal path. In theory, a path one millimetre (or whichever small distance) to one side of another, is a different path, thus in any given space there are an infinite number of possible paths. Figure 3.2 shows an example of two paths that would yield the same optimal path, for this case one path can be considered to be redundant and only one of them is needed as a starting point for the optimisation. While in figure 3.1, the three paths shown are non-redundant and the optimisation process would have to consider all three possibilities.

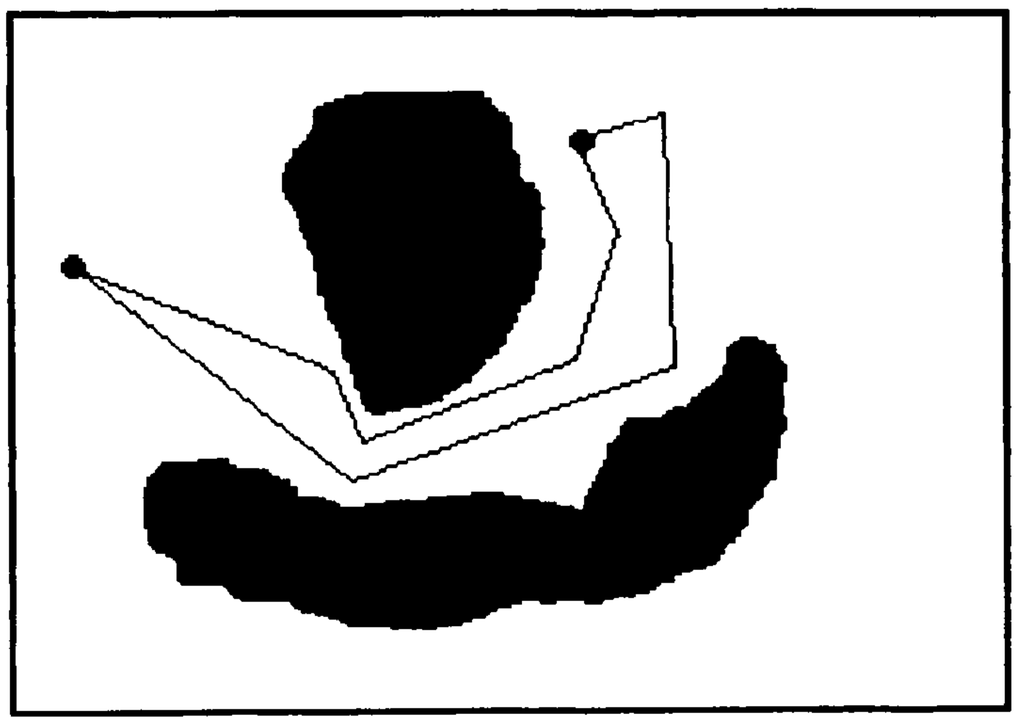


Figure 3. 2 Redundant paths

The three aspects mentioned above were taken into account in the development of the first stage of the path optimisation system used as the case study in this thesis.

3.1. Development of the Path Generation System

There are many techniques available for path generation. One of the first techniques used by the researchers consisted of drawing a line from the origin toward the goal, and when an obstacle was found the line was drawn bordering the obstacle until there was no more of the obstacle in the direction of the goal. This algorithm is very fast if there are only a few obstacles, and if there are no situations where the path could get trapped (such as concave obstacles, dead ends, closed rooms, etc.). But if any of the restrictions mentioned were found, or if the configuration of the obstacles is very complicated, then the algorithm may never get to the goal. Figure 3.3 shows an example where this algorithm would loop forever inside the *mouth* of the obstacle. When this happens, the main problem is the need to find a way to be sure that it is impossible to find a path, and how much time is allowed to elapse before the search is terminated.

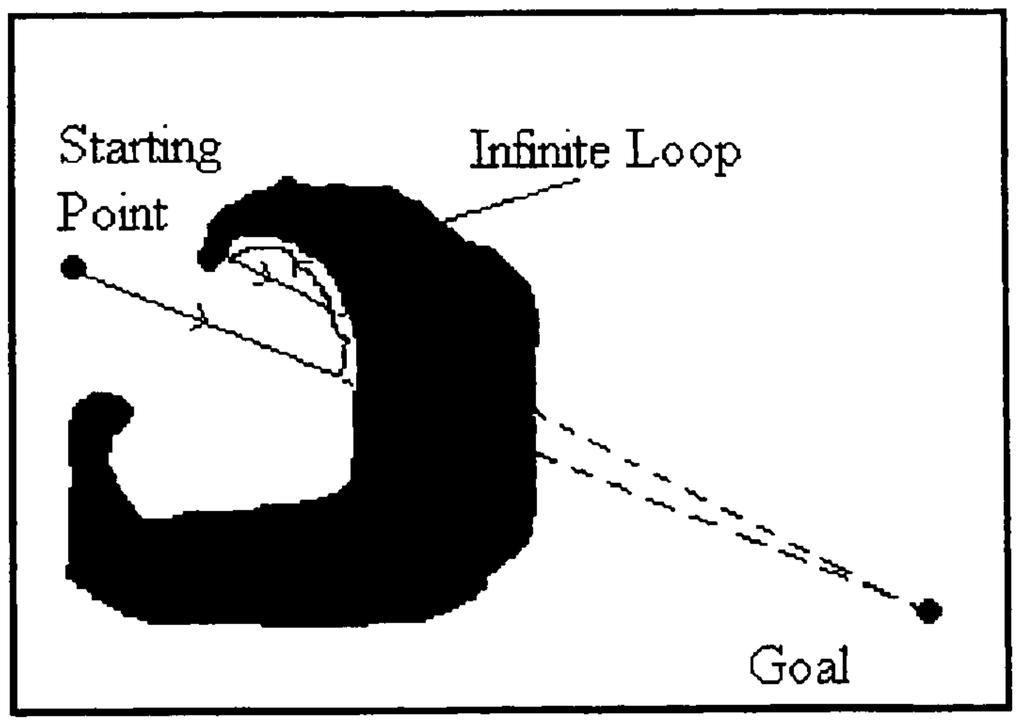


Figure 3. 3 Concave obstacle *trapping* the path

New algorithms have been developed to produce more reliable methods [Hwang et. al.]

[Latombe], these approaches included potential fields methods [Koditschek], Voronoi diagrams [Lee D.T.] [Sugihara], vectorial approaches [Asano et. al.] [Janét et. al.], and cellular automata growth [Tzionas et. al.] among others. In the following paragraphs these methods are going to be reviewed from the point of view of reliability, computational load and suitability for parallel processing.

The potential fields methods are one of the most reliable forms to generate a path, but their computational load is also one of the highest compared to other methods [Hwang et. al] [Koditschek]. It consists of generating a potential forces map, each of the points on the map represent a force vector whose direction and magnitude depends on the total sum of the mass and distribution of the obstacles acting at each point, as if each one of the obstacles produced an attractive (or repulsive) force proportional to its size and distance to each point in space. Figure 3.4 shows a low resolution sketch of this method, the arrows represent the repulsion vectors from the obstacles. The path is then found by following the minimum potential track from the origin to the destination. Despite its high computational complexity, this method can produce local minima dead-end paths that do not reach the destination point, but generally speaking it is reliable. An additional advantage of this method is its suitability for parallel processing, which can increase the computational speed when such a system is available.

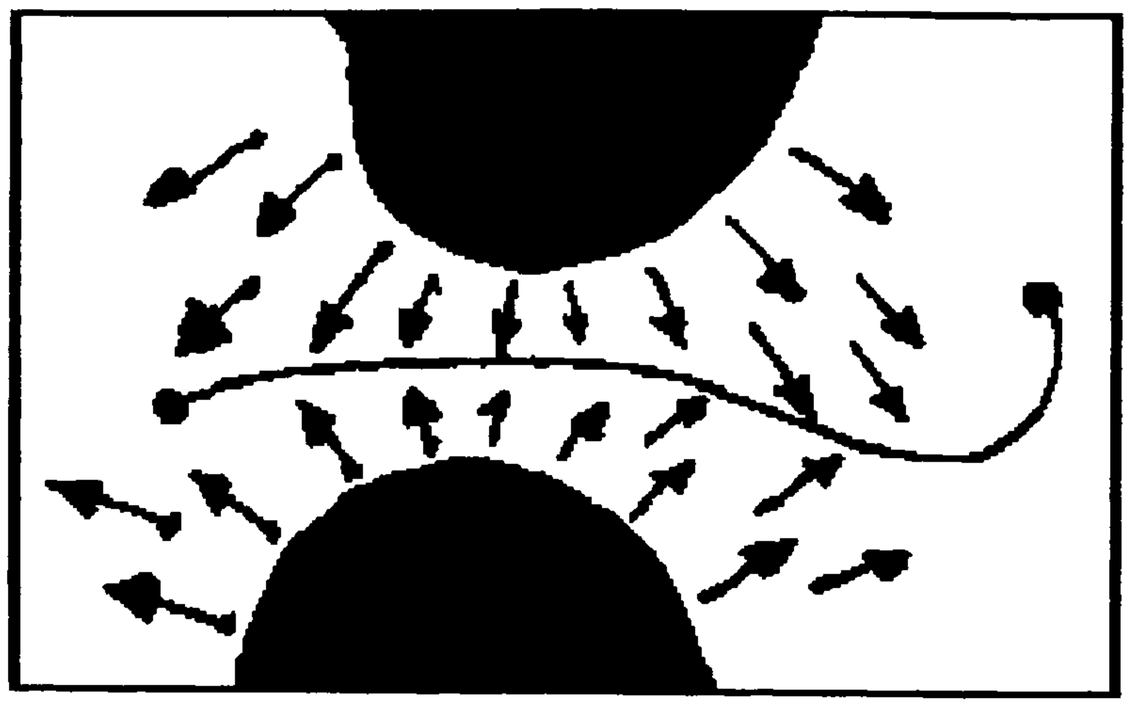


Figure 3. 4 Potential fields path generation

Voronoi diagrams provide a generalised geometrical way to describe free space and obstacles in discrete maps composed of polygonal regions [Lee D.T. et. al.]. They can be used to find paths by moving from a free space region to an adjacent region in the direction of the destination. Depending on the approach used this method can easily get trapped or require relatively high computational power (in terms of CPU time and memory).

Vectorial [Asano et. al.] [Janét et. al.] **and mathematical** approaches [Hwang et. al] [Latombe] tend to get become computationally very complex in the presence of a large number of obstacles. In addition, most of these methods have little suitability for parallel processing.

Cellular automata approaches, such as the approach used by [Tzionas et. al.], are quite reliable and suitable for parallel processing. This method can be seen as an extension of Voronoi diagrams, where the regions become relatively small so that they can be represented as identical cells. In [Tzionas et. al.], by producing and recording the growth

pattern of the cellular automata, a single path can be found, that, as explained by the author, normally corresponds to the minimum distance path. The functions for the behaviour of the cellular automata (interconnections) are very simple logical expressions. Figure 3.5 shows an illustration of the principle used by this technique.

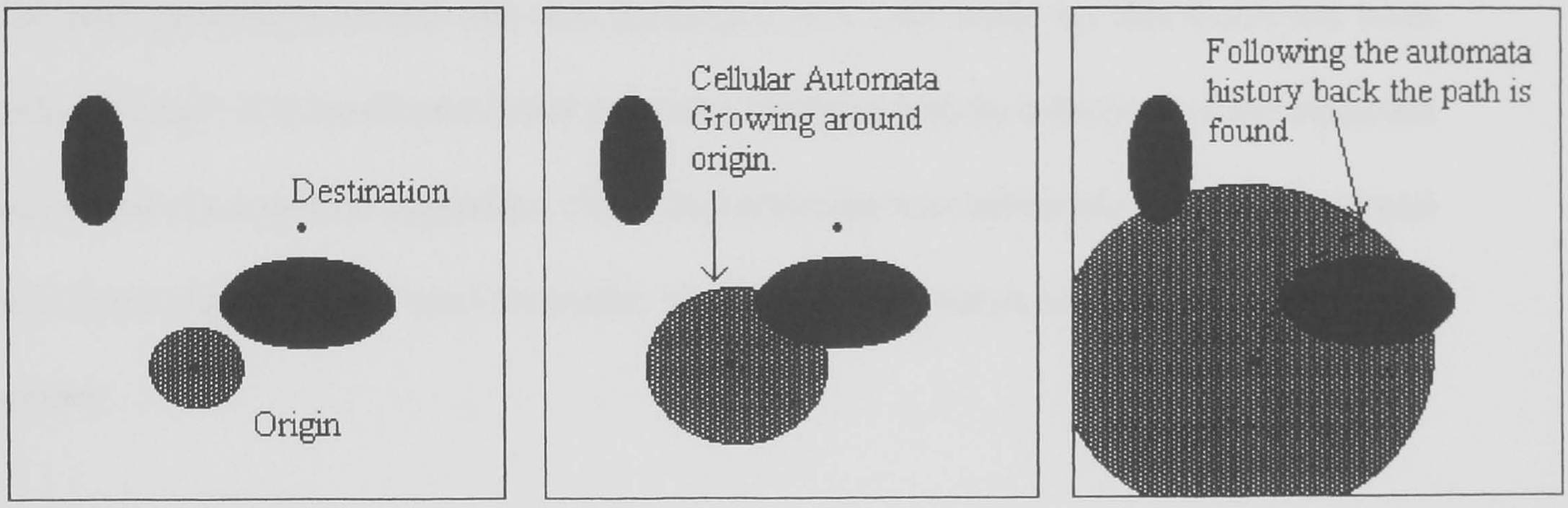


Figure 3. 5 Finding a path with *Cellular Automata* growth

For the path generation in [Tzionas et. al.] the interconnection functions were designed by the authors to grow the cellular automata around the origin. The cellular automata pattern grows filling the free space until it gets large enough to touch the destination. At this moment the growth history is followed backwards to determine the path to follow from origin to destination.

Despite the apparent simplicity of the way in which cellular automata work, relatively complex behaviours in the whole system can be achieved [Peitgen]. The state of a cellular automaton can be calculated very quickly in digital computers and it is highly suitable for parallelism. Thus, if a map is represented in a discrete way where each cellular automaton

is a *pixel*, the whole map of the environment in which a robot is going to travel can easily be covered. This would provide an exhaustive search method for paths, which would mean a completely reliable system. For these reasons, cellular automata were chosen as a very promising approach for the generation of the path.

The path generation system that was developed as a case study for this thesis has been called "Rutar". It is based on cellular automata colonies and the behaviour of the automata was found via a genetic algorithm. Once the behaviour was achieved, it was programmed as a fixed algorithm and used thereafter as the path generation tool for the optimisation system.

3.1.1. Using cellular automata to generate paths

To use cellular automata for path generation, various methods can be employed. For instance [Tzionas et. al.] applied the cellular automata as a high resolution Voronoi diagram and produced a growth, starting with an automaton at the origin, in a recursive way through the free space cells, until one of them reached the destination. Mathematically this method is very simple, but as is the case with Voronoi diagrams, it requires a vast amount of computer memory for large maps.

The approach used in this thesis limits the amount of memory used to the minimum necessary to represent the map itself. To achieve this, the map is represented as a discrete

image where each *pixel* has three different major states: it can be part of an obstacle, free space, or an origin or destination point that the robot has to visit. Since the robot must have an origin and a destination, but it can be required to pass through many points on the way, the number of points that the robot has to visit can be two or more.

The cellular automata are used by representing each free space *pixel* as an automaton. Each of the free space areas contained in the map is therefore represented as a *colony* of cellular automata. There will be as many colonies as there are separate (disconnected) free space areas in the map. Figure 3.6 illustrates this showing two colonies in the map, since the free space outside the 'O' and the free space inside it are not connected.

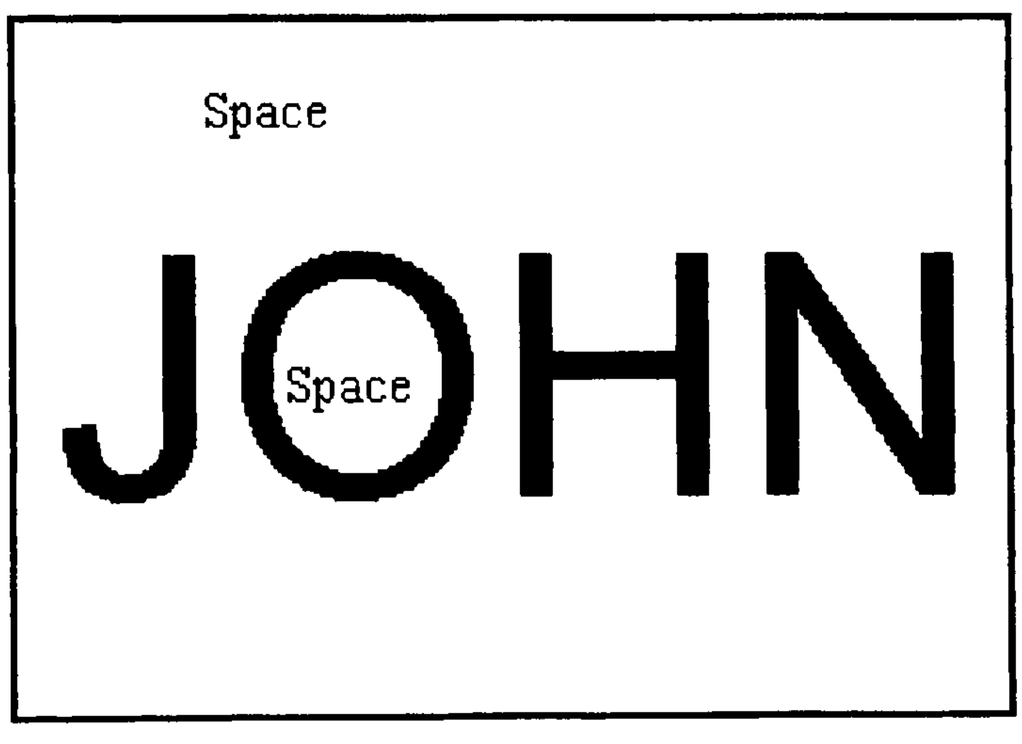


Figure 3. 6 Map containing two separate free-space areas

If the interconnections between the cellular automata are seen as a set of rules of behaviour that each automaton will follow depending on the state of its neighbours, the complete group of automata can be seen as a colony of cells that will follow a given pattern of behaviour as a whole depending on each individual's pattern of behaviour just like a

colony of bacteria. It is important to emphasise that the rules of behaviour of each automaton are identical to the others. This approach can make the cellular automata very useful and yield complex results, but finding the correct combination of rules can be very difficult. The development of these rules of behaviour is the main issue presented in this chapter.

To generate paths with cellular automata, the colony has to shrink in a way such that at the end, there are only *single-cell* width *brick roads* starting at an origin and reaching a destination (this will be seen clearly later when referring to figure 3.7). Therefore these points will anchor the colony they are contained in, so that at the end they will continue to be part of the resultant shrunken colony. For this reason, all the points that must be visited are going to be called the *anchor points*. To achieve the path generation, the behaviour of the colony has to follow one simple basic rule:

- **The colony must shrink (i.e. cellular automata must *die*) to a minimum number of members without dividing the colony into more than one, where the anchor points represent automata that cannot die.**

If this condition is met, the resulting *brick roads* connecting each *anchor point* to the others will represent all the non-redundant possible paths that can be considered for the path optimisation problem. This rule has been produced in this work to fulfil the need for a complex and efficient behaviour of the colony to achieve the path generation task. With the definition of this single clear rule, the search for the appropriate functions of behaviour that

will produce the desired results is going to be performed.

The paths generated at this stage are not supposed to be optimal in any way since no physical characteristic of the robot has been taken into account so far, neither have any optimisation criteria been considered. Still, at this stage it is convenient to take into account some considerations regarding the robot, the computer resources available, and the desired resolution, such as:

- For higher resolutions of the map, or for larger maps, the computer resources requirements will be higher and the computation time will be longer. Later in this chapter and in the next chapter, where the final system is analysed, a precise quantification of the relationship between map size and resolution with the computational load will be established.

- The recommended resolution in most cases is the one where the size of the pixel represents an area approximately the same as the robot size. This is because at this scale, if a path is generated, most probably the robot will fit between the obstacles and be able to follow it. This level of resolution may be referred to hereafter as *critical resolution*. At a higher resolution a generated path might not be feasible due to the size of the robot, and at a lower resolution the obstacles may be misinterpreted or paths that could be feasible may be ignored because it is seen as if the robot will not pass between obstacles.

The last consideration implies that depending on the criteria defined by the task that needs the robot to follow a path, aspects such as computation time and numerical precision can be traded off by means of the resolution used. Low resolution may be useful to generate paths with low computational power. Using critical resolution may be useful to generate the most optimal path. And high resolution may be useful to reliably consider, in an absolute sense, all possible non-redundant paths for evaluation of alternatives if different robots are available.

3.1.2. Finding cellular automaton rules

The behaviour required for the cellular automata to generate the paths has been determined by the rule defined above, and it can be seen to be quite simple. Now appropriate rules that will enable each automaton to follow that behaviour have to be found. Since these rules will determine which automaton disappears or *dies*, these rules will be called *rules of death* for the cellular automata colony. The process of executing these rules until the paths are found will be called executing the *cycle of life* of the cellular automata.

To apply learning to the system, these rules of death are going to be found automatically by means of a genetic algorithm. The genetic algorithm has to evaluate the behaviour of the cellular automata with respect to proper path generation and manipulate the rules until they result in a *correct set*. The configuration required to achieve this task was designed using the following: definition of the map and its characteristics, definition of the cellular

automata rule structure, definition of the evaluation process including the execution of the cellular automata cycle of life and assessment of its performance (fitness), and the definition of the genetic operations and termination criteria. These will now be discussed in turn.

3.1.2.1. Definition of the map and its characteristics

The map will be described as a rectangular array of pixels of a given, and flexible, resolution. Later in the analysis of the computer performance, the practical limits for the resolution or the size of the maps will be discussed. Each pixel has at least four possible states (thus at least two bits are required to describe it):

- It can be part of an obstacle. If the pixel contains part of an obstacle, the whole pixel is considered to be part of the obstacle (the same as a single cell as defined in Voronoi diagrams).

- It can be an anchor point (a point to be visited by the robot).

- It can be free space. Free space pixels correspond to cellular automata, so during the execution of the life cycle each one can be either alive (initially all are alive) or dead (disappear), on termination this meaning will correspond to path or free space.

- In the case where the execution of the cycle of life is performed in a sequential way (such as in most common computers) an intermediate state may be needed, where an automaton can be tagged as condemned to die in the next execution step. In truly parallel systems this state may not be necessary since all the automata will be evaluated (executed) at the same time.

Depending on the machine and programming language used, the memory needed to describe a pixel in the computer can range from a minimum of two bits to a machine's short word (usually a byte). For convenience in the programming and simplicity of the memory structure defined, a byte was used to describe each pixel in the structure of "Rutar", the system that was practically developed as the case study for this thesis. Using a byte means that it can be portable to any computer system.

In summary, the map is represented as a bi-dimensional array of cells that can have a particular value depending on their state. Figure 3.7 shows a very small and low resolution map (for purposes of clarifying the definition), map (a) is the initial state with origin, destination and obstacle (left to right) ; and map (b) the final ideal state where the two non-redundant paths have been found (it is important to keep in mind that the mathematical size of each pixel is square, the fact that in some drawings the pixels appear rectangular is only an optical effect of the particular printer used, but it is irrelevant to the calculations).

Table 3.1 shows the array and its values corresponding to each map's state.

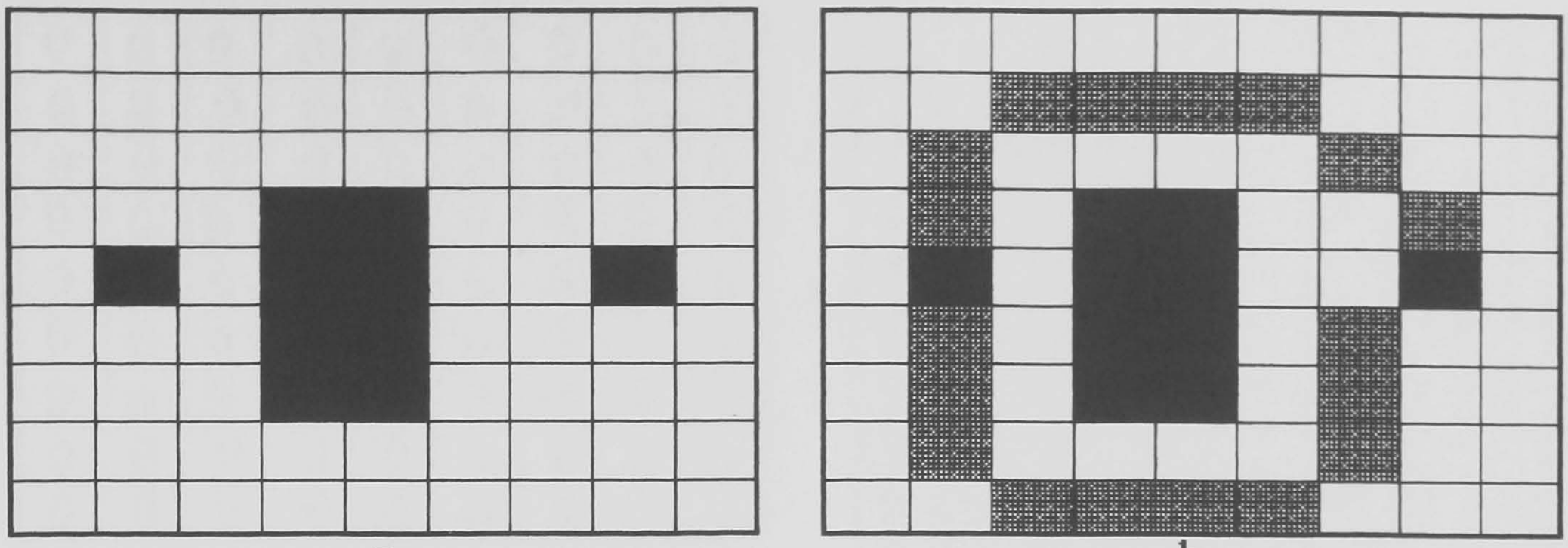


Figure 3. 7 Low resolution map showing the representation of the pixels

The numbers in Table 3.1 represent the states of the automata as follows:

- 0. Free space.
- 2. Automata alive at the end of the cycle, therefore the paths (bricks road).
- 4. Obstacle.
- 8. Anchor points (automata that cannot die).

These numbers are chosen arbitrarily.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	4	4	0	0	0	0
0	8	0	4	4	0	0	8	0
0	0	0	4	4	0	0	0	0
0	0	0	4	4	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

a

0	0	0	0	0	0	0	0	0
0	0	2	2	2	2	0	0	0
0	2	0	0	0	0	2	0	0
0	2	0	4	4	0	0	2	0
0	8	0	4	4	0	0	8	0
0	2	0	4	4	0	2	0	0
0	2	0	4	4	0	2	0	0
0	2	0	0	0	0	2	0	0
0	0	2	2	2	2	0	0	0

b

Table 3. 1 Values representing the state of the pixels in the map

3.1.2.2. Definition of the cellular automata rules structure.

The interesting point about using cellular automata is that no matter how many individuals there are, only one function, or set of rules has to be defined and this unique set will be applied to all the individuals. In the scheme used in this thesis, an additional advantage exists over other applications that use cellular automata [Tzionas et. al] to generate paths, since no global history is saved on the evolution of the cycle of life. In [Tzionas et. al] the path is found by growing the cellular automata and a history of the growth was kept in each cycle so that following the *back-track* after the cellular automata reached the destination would provide a minimum cost path (usually in terms of distance). With their approach, although the cellular automata structure and the map were relatively simple in terms of memory and computing power, the track history is very costly in these terms. In the approach used in this thesis there is no such history to keep, thus the memory and computing power is kept to the minimum necessary for the map and the cellular automata

structure.

The rules of death for the cellular automata are stored as a binary tree that will evaluate to true or false at the root node. Each node can be composed either of a logic operand or a terminal node representing the state of a nearby automaton, as follows:

- The logical operation nodes consist of a two arguments logical AND or OR, that will evaluate as true or false depending on its two branching nodes.

- A terminal node represents a neighbouring automaton, and will evaluate to true or false depending on the state of the automaton it represents. If the automaton is in the same state that the node says, the node will evaluate to true. If the automaton is in a different state to what the node says, it will evaluate to false. The automaton represented can be any of the 8 immediate neighbours or the 16 next level neighbours of the automaton that is being evaluated. And the state can be Anchor Point (always alive), Alive, Condemned (to be dead on next step), Dead (no automaton left), or obstacle pixel. Note that neither an obstacle pixel nor an Anchor Point will be executed as an automaton (does not follow rules of death), but either of them can be neighbours of automata.

When a node is evaluated it transfers its result to the parent node until the complete tree is evaluated. If the overall result is True, the automaton executing the rules of death tree is itself marked for death (condemned) so that at the beginning of the next step it will be cleared. If the result is False, the automaton will do nothing, for the moment.

Figure 3.8 shows an example tree with logical operator nodes and terminal nodes, as would be used to represent a set of rules of death. The logical operation nodes are identified by the OR (+) operator or the AND (•) operator. The terminal nodes are identified with the neighbour identification number and its state. This example is just to show the design of the trees, it may make no sense in terms of the rules of death.

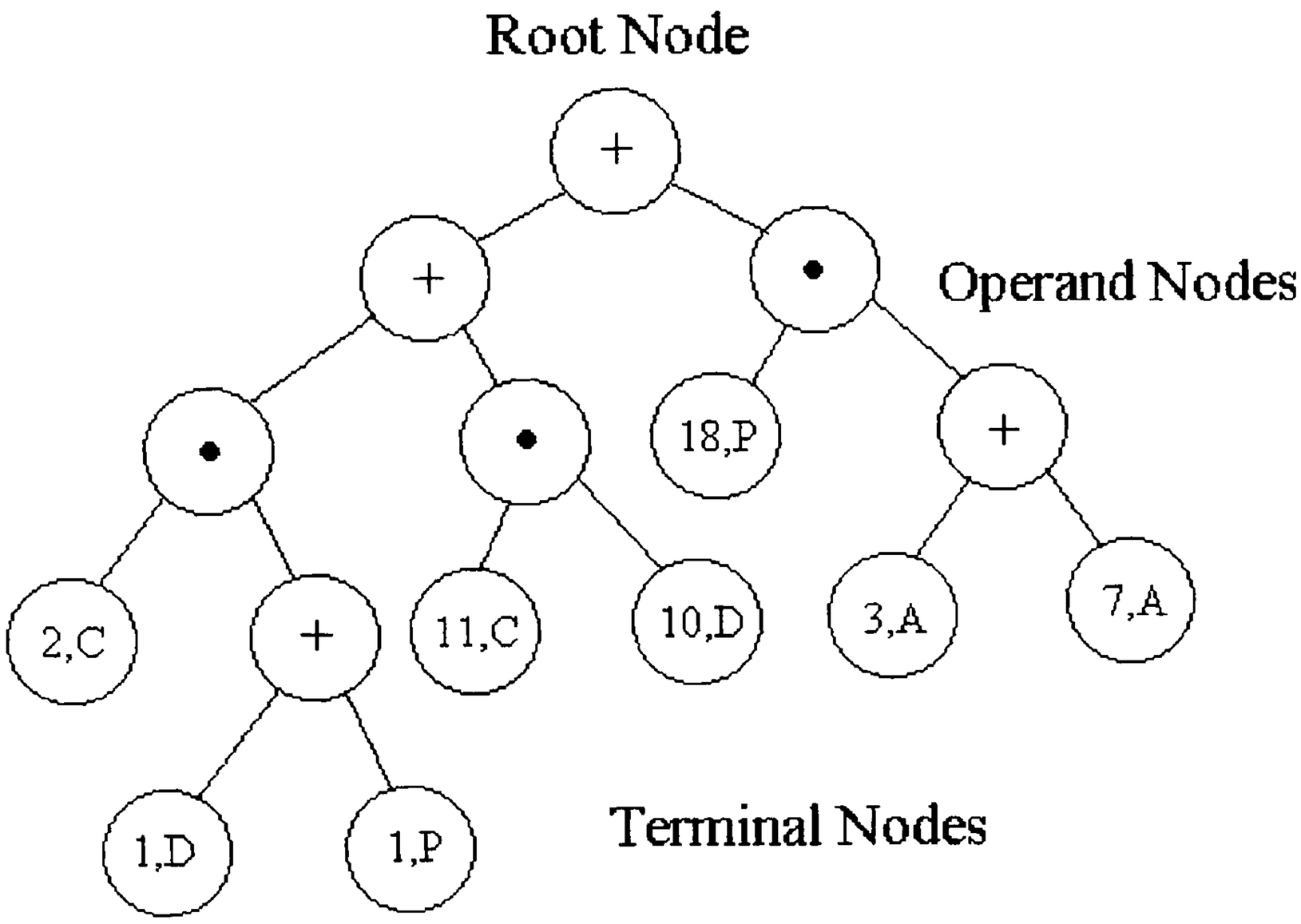


Figure 3. 8 Example binary tree to represent the *rules of death*

Figure 3.9 shows the identification number definition used for the neighbouring automata with respect to the centre one, which is the one currently executing its rules of death. This

number corresponds to the first (left) parameter of the terminal nodes.

The possible states mentioned above for the neighbouring automata, and depicted in the terminal nodes as the second parameter (right), can be described in detail as follows:

- **State P:** Anchor point. As explained earlier, this is a special case of an automaton that cannot die. Thus it is in a fixed live state. For the purpose of evaluation of other cellular automata its state is always alive. For itself it means that it does not have to follow any behaviour (execute rules of death), it just stays there, alive.

- **State A:** A live automaton, for the moment.

- **State C:** Condemned. It is going to die (in the next execution step it will not be there).

- **State D:** Dead (no automaton present, so from now on up to the end it will be free space).

- **State O:** Obstacle pixel. These pixels do not execute any rules, so they will always remain as obstacle pixels. But they have to be recognised by the automata since they can also be neighbours.

9	10	11	12	13
24	1	2	3	14
23	8		4	15
22	7	6	5	16
21	20	19	18	17

Figure 3. 9 Identification numbers for the neighbours of an automaton

Having determined these characteristics, a very simple and structured rule syntax has been defined so that the genetic algorithm can search for a set of rules that will provide the desired behaviour. The chromosome syntax is a LISP type string [Graham, P.] for the binary tree. The example tree in figure 3.8 would be written as follows:

OR(OR(AND(OR((1,D),(1,P)),(2,C)),AND((11,C),(10,D))),AND(OR(OR((3,A),(7,A))),(18,P)))

As it can be seen, the syntax is a simple recursive two operand (always) logic expression. In this way, it is very simple for the genetic algorithm program to manipulate the trees coherently and unambiguously. This chromosome structure will be used to describe each individual in the population of possible "rules of death" that the genetic algorithm is going to manipulate.

3.1.2.3. Definition of the evaluation process.

The evaluation process consists of two parts, the execution of the cycle of life and the assessment of the results that provides the fitness value used by the genetic algorithm to operate on the rules and to evaluate the termination criteria.

The execution is straight forward. It consists only of the use of one (or several) example map(s) on which the system is tested. Some considerations have to be taken into account to use a number of maps with given characteristics so that the genetic algorithm will be able to develop a set of rules with no unexpected flaws. In other words, so that the developed set of rules is generic, suitable for any map on which it may be used at any moment and not suitable only for the examples used. This is achieved by considering the following:

- Obviously, more than one map has to be used, so that various characteristics can be *trained into* the set of rules. In other words, that the cellular automata will learn to work in a generic environment.

- The main cases of complicated paths have to be present in the maps, for example : impossible paths, disconnected paths, paths that need to initially move away from the destination before approaching it again (such as a target point inside the concave part of an obstacle like the case shown in Figure 3.3).

- A trivial path that can be used to quickly evaluate the connection of the *Anchor Points*

and the capacity of the path generator to minimise the number of cellular automata that are alive at the end, which would tend to minimise distance initially.

- A case where the exact number of non-redundant paths can be evaluated.

The example map shown in figure 3.6 can be used to include most of these characteristics. If the letters are considered to be the obstacles, and the anchor points are defined as shown in figure 3.10, the expected resulting paths would be something like what is shown in figure 3.11. From this example the following aspects can be tested:

- Having anchor points both outside the O and inside it, will test for disconnected paths, i.e. there will be one path between the two anchor points inside the O, and 5 paths between the anchor points in the H, but the path inside the O is disconnected from the other 5.

- The path inside the O is trivial on its own. So this one can be used to test for minimum survival.

- The paths connecting the anchor points inside the H need to initially move away from the destination to go around the concavities of the H. Thus this aspect can be tested.

- To connect the anchor points in the H there are five non-redundant paths as shown in figure 3.11.

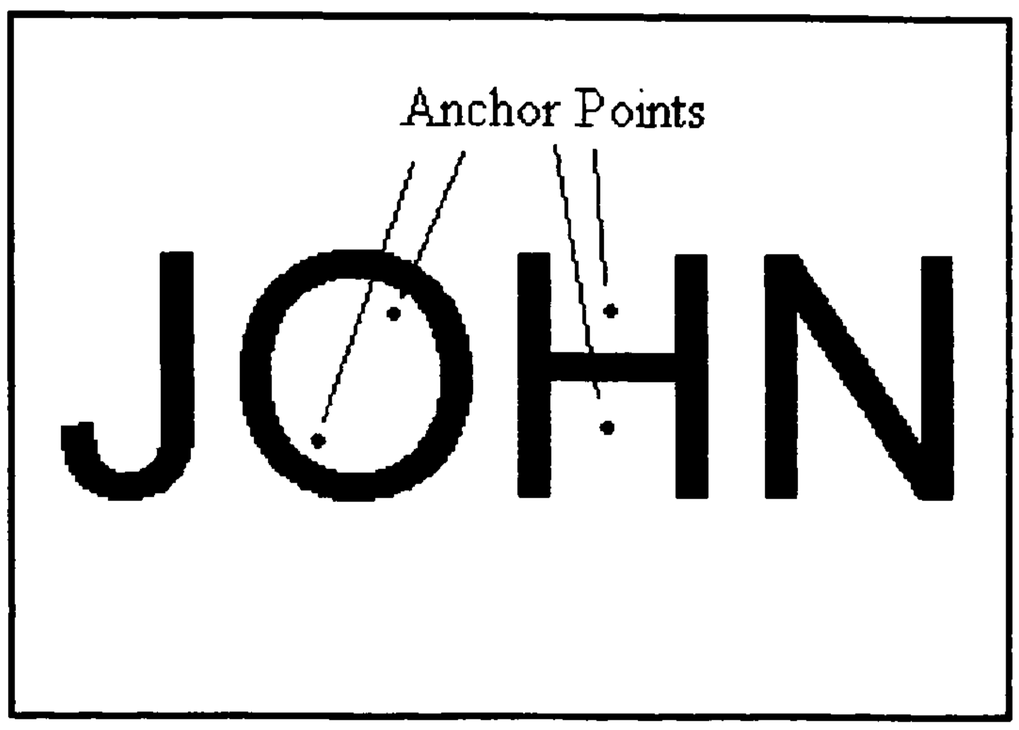


Figure 3. 10 Anchor points defined in the two disconnected free-space regions

As far as the map is concerned, the *Anchor Points* are indistinguishable from one another, thus at this point there is no distinction between origin and destination(s), the non-redundant paths that needed to be generated for this maps are supposed to connect every *Anchor Point* with all the others that it may be possible to connect to. It is the responsibility of the path optimisation system to define the order of visiting the different points and deciding what to do with isolated points or disconnected paths.

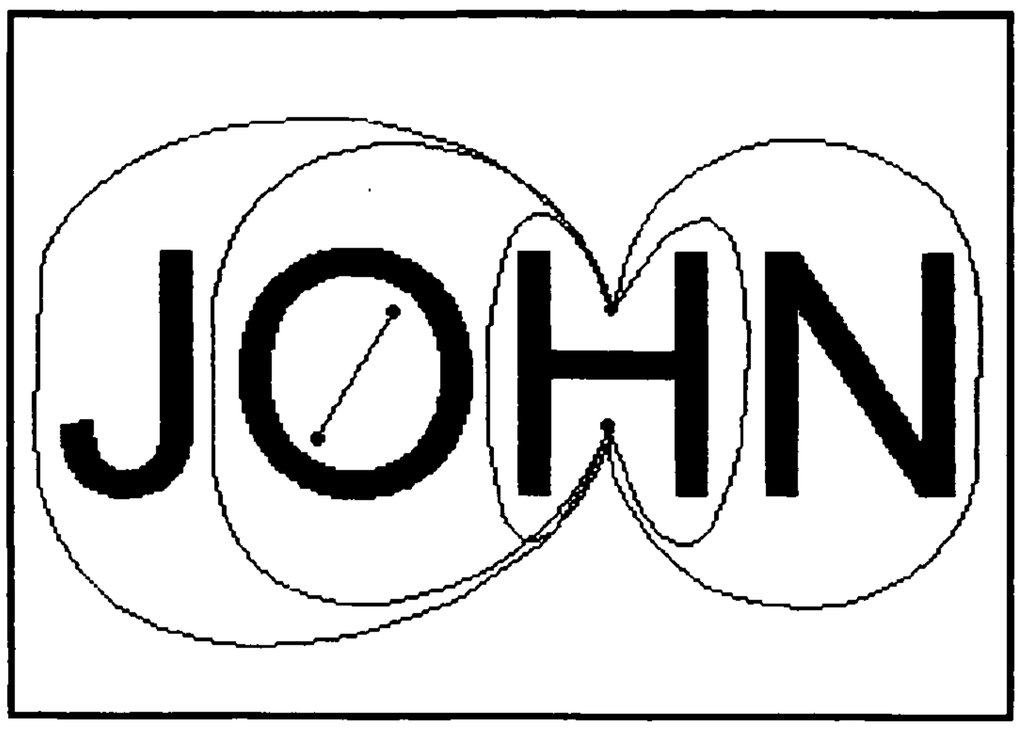


Figure 3. 11 Expected alternative paths to be generated

For the development of "Rutar", two maps were used. One map is the one shown in figures 3.6, 3.10 and 3.11, and the other one is shown in figure 3.12. In this map additional characteristics are evaluated such as connecting multiple anchor points, and having an isolated anchor point.

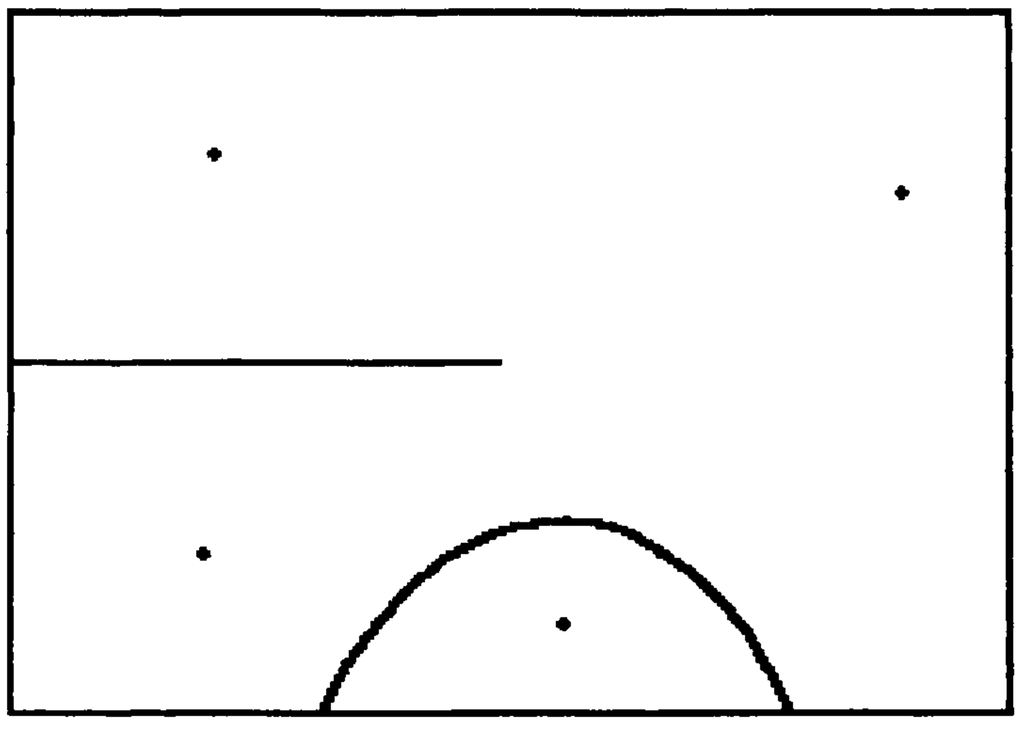


Figure 3. 12 Additional example map

To assess the performance of the rules, the following criteria were used:

- The connection of the anchor points. A pre-defined *answer sheet* was constructed for this purpose. It is just a list of the points that have to be connected with the number of paths. This evaluates connectivity and the number of non-redundant paths. Compliance of at least one connection provides a True in the connectivity table, else a False is given. The number of paths connecting two points is provided so that it is compared with the number that it should be. Table 3.2 shows the format of the answer sheet. Points 1 and 2 are inside the O and points 3 and 4 are in the H.

- Number of automata connecting two points in a trivial unique path. For the case of the path inside the O the minimum possible number of points was hand-counted and given as a value to compare with the actual number of automata forming the path. This evaluated the aspect of minimum survival.

- The time taken to perform the execution. There is no value defined as minimum, only the lower, the better. This provided a measure of the efficiency of the set of rules, which in turn limits the length of the expression.

Answer Sheet **Map: JOHN**

Anchor Point	Anchor Point	Number of Non-redundant paths	Paths Found	Connection	Connection Found
1	2	1		TRUE	
1	3	0		FALSE	
1	4	0		FALSE	
2	3	0		FALSE	
2	4	0		FALSE	
3	4	5		TRUE	

Anchor Point	Anchor Point	Number of pixels	Pixels alive
1	2	32	

Table 3.2. Example answer sheet

A program was written in C to perform this evaluation automatically at the end of each generation. The program starts at one anchor point and follows the *bricks* until the road finishes, arrives at another anchor point, or diverts into two or more roads. This scanning of the roads is made in a recursive way until all the roads originating from the anchor point are explored. It does this with every anchor point in the map.

If the road finishes without arriving at another anchor point the program goes back to the previous bifurcation or to the initial anchor point to look for another direction to start the

scanning again. If there are no more directions to scan in that anchor point, it passes to another anchor point and starts again until it has tried all anchor points in the map.

If the road arrives at another anchor point it places the corresponding TRUE flag in the answer sheet and a count in the number of paths connecting those two anchor points. If the anchor point where it arrives is the same one from where it started or if it arrives at a part of the path previously explored, the action is taken as if the road finished without making a connection. This last characteristic stops the program from getting trapped in a loop or finding paths going twice around an obstacle.

From the beginning (and since the colony does not grow but only shrinks) it is evident that there is no way that the system will connect anchor points from disconnected free space regions (i.e. different colonies) so this case was not evaluated. For the case of the isolated anchor points, the evaluation consisted of counting the number of surviving automata, which ideally should be none apart from the anchor point itself.

If the road diverts, the program will generate a tree type recursive structure to scan all the possible diversions and at the end of each diversion evaluates performance according to the previous criteria. The number of diversions found that resulted in a connection to the same anchor point are logged as the number of paths found for the non-redundant path evaluation.

The recursive algorithm for this program is:

```
Start
     repeat while (select one Anchor point)
          repeat while (select one brick to divert_to)
               follow: (brick)
                    repeat while (one not visited brick
                         adjacent or anchor point adjacent)
                    step to brick
                    mark brick as visited
               if anchor (point adjacent)
                    log this road as TRUE
                    return
               else_if (select one brick to divert_to)
                    follow brick
               else return
          return
End.
```

This program is very simple since the maps used are pre-defined and the answers are also known. To keep this simplicity there was no assessment of path shape, since this would involve a very complex program capable of *quantifying* a shape, in addition to the fact that the supposedly ideal shape is not known. Anyway, the ideal shape is precisely what is going to be found with the complete optimisation system further in the process. At this stage it really does not make much sense since it depends on the mechanical (dynamics and kinematics) characteristics of the robot, which are not considered at this stage (this is going to be considered in Chapter 5).

To compute the final fitness associated with each individual set of rules, four different measures were used rather than a unique number. These were combined to give the total measure of fitness. These values are:

- NC : Number of pairs of anchor point for which a connection was achieved, out of the existing possibilities. NC= number of TRUEs achieved divided by the number of

TRUEs existing in the answer sheet. Maximum = 1. This value is used as a fitness assessment of the individuals while less that 1, for the termination criteria to be met, at least one individual has to have a 1 in this value.

- SA : Extra number of surviving automata compared to the ideal number. Depending on the ideal number defined, a percentage is calculated. Minimum = 0. Any threshold can be defined in the termination criteria for the Genetic Algorithm.

- NP : Number of paths found. Depending on the correct number of non-redundant paths existing in the maps, a symmetrical percentage is calculated. In other words, the percentage of extra paths or the percentage of paths missing. This difference is compared with the correct number of non-redundant paths and a threshold can be defined for the termination criteria of the genetic algorithm.

- ET : Execution time. When the system (computer) provides this data after execution of each individual, this number is used along with the total fitness to provide a small advantage to fast (thus possibly efficient) individuals. Since the ideal value for this variable is not known, it is used in the fitness assessment for each individual, but is not used in the termination criteria of the genetic algorithm.

The results from the execution of the genetic algorithm will be presented later in section 3.2.

3.1.2.4. Definition of the genetic operations and termination criteria

The final stage of the configuration of the genetic algorithm consists of defining the way that the genetic operands are going to be applied to the chromosome, the percentage of chromosomes created with each operand (replication, crossover, and mutation) and the termination criteria.

The genetic algorithm was defined so as to perform the following operations on the chromosomes :

- **Elitism:** when elitism is applied, a given number of the best individuals of the current generation are always copied into the next generation. The individuals chosen for elite copying, are the top in order of fitness. The number of copied individuals can be a percentage or a fixed number, such as the top 10 individuals or the top 5%. This value can be defined at the beginning of each execution. The effect of different values of Elitism used are evaluated in section 3.2.

- **Replication:** a percentage of the new generation's population is chosen from the current generation on the basis of a probability directly proportional to the fitness of the individuals. The fitter an individual, the more probable it is that it will be copied into the next generation. The difference compared with elitism is that it is not necessarily the very best; it has a level of randomness and fitness increases the possibilities of being

copied but there is no guarantee as in elitism.

- **Crossover:** a percentage of the new generation's population is created by combining individuals of the current generation by pairs. As with replication, the fitter an individual, the more probable that it will be chosen as a parent to crossover. The way in which crossover is performed will be described at the end of this section.

- **Mutation:** a percentage of the new generation's population is created by mutating individuals from the current population. Both the choice of which individual to mutate, and which parameter to mutate in the chosen individual are random. The mutation is very valuable to keep the genetic algorithm from falling into local minima. The way in which mutation is performed will be described at the end of this section.

The percentage values for each of the listed genetic operands are parameters given to the genetic algorithm's shell at the moment of executing the program. These values are given by the operator and can be based on experience, trial and error, or based on the performance of the G.A. with some initial trial values.

Once the individuals are chosen, the crossover and mutation schemes are as follows:

- **Crossover** is performed by choosing a random node in each parent's tree, and flipping the parts between them, thus two parents produce two new trees. The method is shown

in figure 3.13. This is the traditional method used by several authors and explained clearly in [Koza J.R.]. Since the trees are binary, any chosen node in any tree crossed with any other, will produce a valid (syntactically) new tree. The chromosome syntax shown in 3.1.2.2. makes the manipulation of them very simple by the genetic algorithm's program.

- **Mutation** is performed in any random node. If the node is a logical operator, the operation is changed (from OR to AND or viceversa). If the node is a neighbour state identifier, either the neighbour number or the automaton state is randomly changed.

The termination criteria used considered the following aspects:

- All the anchor points that can be connected, are connected by the cellular automata. Thus NC=1.

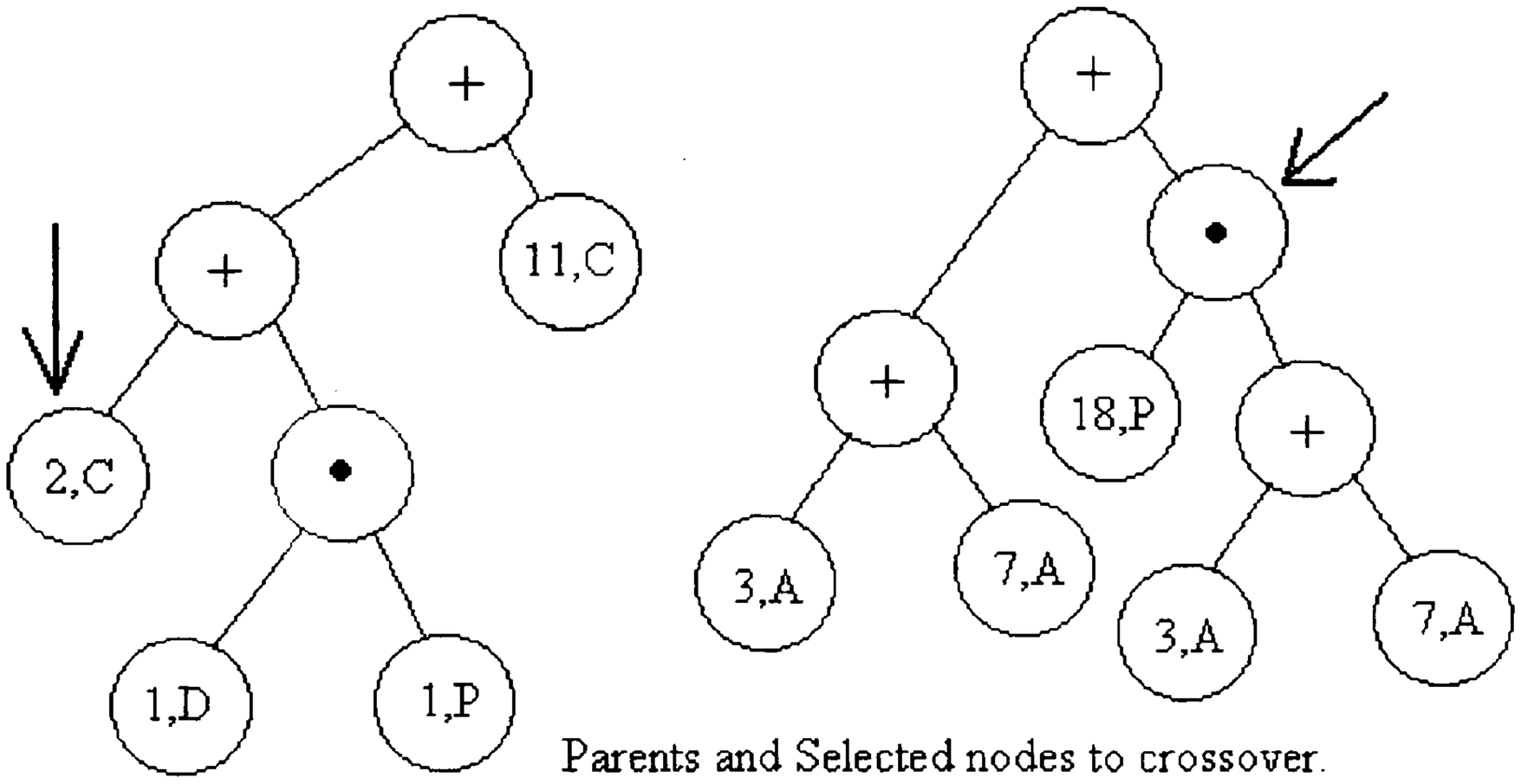
- All the possible non-redundant paths are considered. Thus SC = 0.

- The maximum number of automata surviving at the end of the cycle of life is within a percentage limit. This applies only to the case of trivial paths.

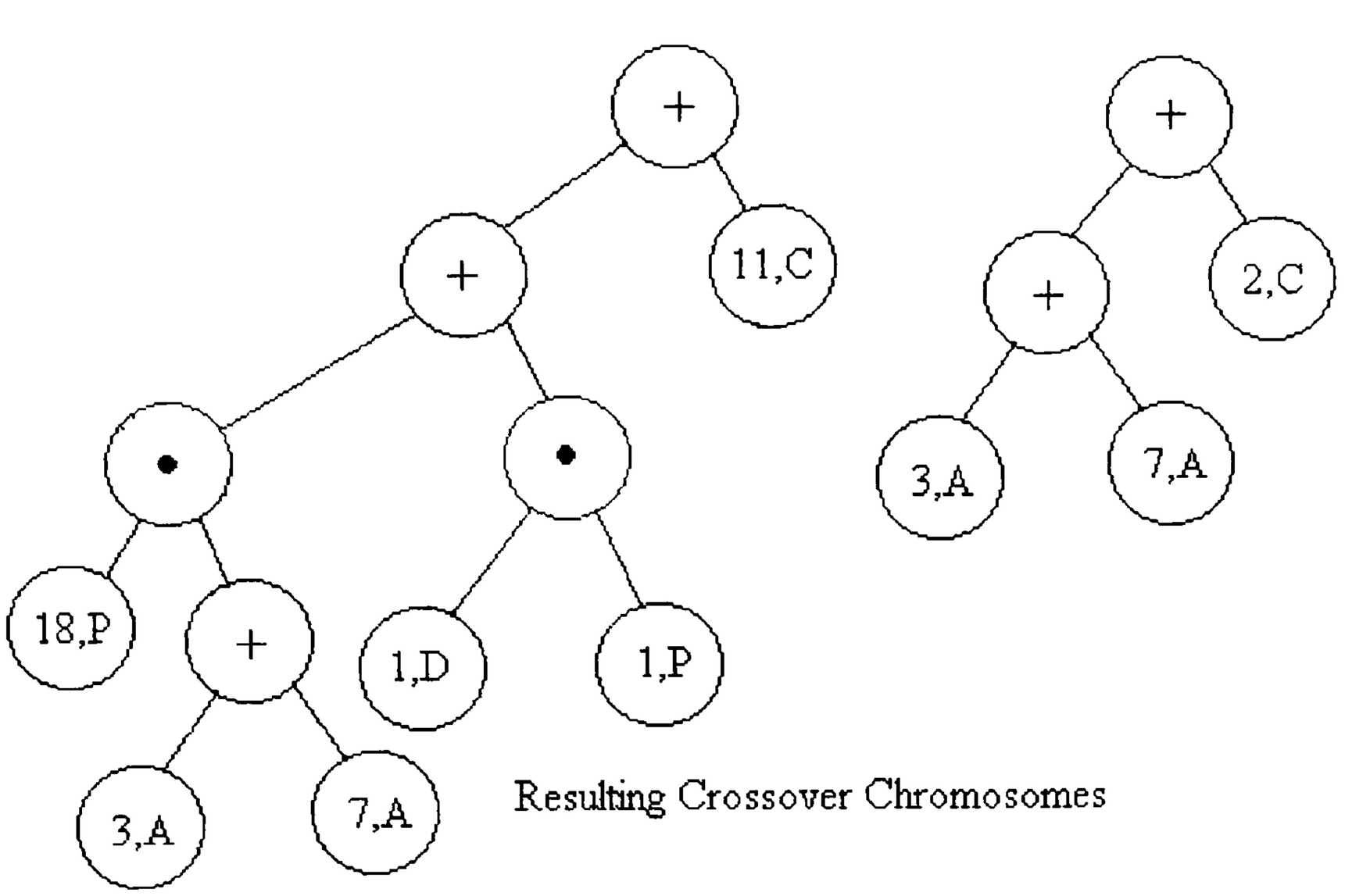
- A maximum number of generations have been executed.

The genetic algorithm is terminated when either all of the first three criteria are met, or the

maximum number of generations allowed have been executed.



Parents and Selected nodes to crossover.



Resulting Crossover Chromosomes

Figure 3. 13 Crossover performed between to trees

3.2. Execution of the Genetic Algorithm

The shell (program) to execute the genetic algorithm was written in C. C was chosen as the language to write all the programs in this thesis because of its suitability for the data management, and the portability due to the availability of the compiler in different platforms (such as PCs, sun workstations, transputer cards, etc.).The flow diagram for the genetic algorithm presented in this section can be found in Appendix A.

The first execution of the genetic algorithm was performed using values chosen empirically from experience gained from the literature and from other work performed previously by the author.

Since most genetic algorithms use most of the percentage in crossover, much less for replication and a marginal percentage for elitism and mutation, the initial values chosen to assess the performance of the algorithm were: 2% for elitism, 8% for replication, 88% for crossover, and 2% for mutation.

The percentage of additional automata surviving over the minimum possible in the trivial path, was set at 5% as a termination criteria, expecting this to have a fairly good minimisation of distance.

From an estimate of the execution time of the algorithm, and to keep the first trial to a

working day, the initial population was set to 1000 individuals, and the time out limit was set to 10 generations. The execution lasted 10 hours on a 12MHz 286 PC computer with 2 MB of RAM. The termination was due to time out, and none of the individuals complied with all of the desired characteristics; in other words, a solution was not found. The fitness analysis showed that there was very little change within generations, and that it was not always increasing, between generations it sometimes decreased and then increased again.

At that time, a set of transputer cards were made available, thus in an effort to increase the computational speed and capacity, the data processing part of the algorithm was recompiled to run in two transputer cards fitted in the computer. The PC was only used as a graphic interface. The transputer cards were two InMos T805 at 25 MHz, and contained 2 MB of RAM each.

The time out limit was doubled, to 20 generations, to test if this would increase the probability of arriving at an answer. The mutation percentage was increased to 10% to test its effect in fitness change (hopefully increase), this extra percentage was subtracted from the crossover, which was set to 80%, since it had the highest value. To stop the high mutation rate from destroying some (or all) of the best individuals, the elitism operation was the first performed, so that those individuals would be exempt from being mutated before being copied. In any case, the mutation was performed in any randomly chosen chromosome, so it is possible to have an additional copy of the best individuals mutated. The execution terminated after about 3 hours, again from time out. Several individuals performed some of the desired path generation except for the minimum survival criterion,

which was not met by any individual. Appendix A shows the trees of the best individuals from this execution.

To look for the minimum survival criteria (not yet achieved), the algorithm was executed again with a time out limit of 100 generations, in an attempt to give enough time for this to happen. Elitism was increased to 5% to increase the number of best individuals being kept between generations, while the replication was reduced to 5%. This change was tried keeping in mind that the elevated mutation rate could make up for the loss in replication, while the increase in elitism would guarantee the survival of a higher average of fitness. The execution terminated after about 12 hours from time out. But this time, all the individuals corresponding to the elitism percentage, performed all the desired path generation functions, again, except for the minimum survival criterion. It was found that many of the successful individuals were equivalent when their expressions were later minimised by hand. Also, the first successful individual appeared at generation 25, very early in the execution, the rest appeared before generation 47. The time out was reached because of the failure to comply with the minimum survival criterion. In other words, none of the individuals generated a straight line path. The minimum survival criterion would have been met if the percentage to meet this termination criterion had been set around 12-14%, which were the values achieved by the best individuals in this execution on the path inside the O in the map JOHN.

These results indicated that the minimum survival criterion is very difficult to achieve while the other criteria, such as connectivity and number of non-redundant paths, was

achieved with nearly a third of the computational time provided (between generations 25 and 47 in a total of 100). The 12.5% extra of automata alive, given by the best individual in the trivial path inside the O of the map JOHN, is not a deterministic figure. As it will be explained in the next chapter, the fact that the execution of the cycle of life of the cellular automata is not parallel but sequential, makes straight paths *bend* in the direction of the sequential scan, thus the deformation of a path depends on the orientation of it and on the way the computer scans the bytes in the array that represents the map. For this reason it is very difficult (or may be impossible) to achieve straight line paths even between points where no obstructions lie between them.

The unusually high mutation rate used (10% as compared with the rates normally used as reported by the authors in the literature on this area, such as 2% or less) demonstrated the power of this operand. When it was found that the fitness was not increasing significantly within generations, the first attempt was to increase mutation just to see its effect on fitness. The dramatic increase of the fitness from generation to generation after it was applied, showed that for this problem it was very appropriate. Of course it does not mean that such a high mutation rate is always necessary, it very much depends on the application, even in most of the cases it may have an adverse effect on the performance of the algorithm.

The replication was replaced partially by elitism by incrementing the percentage of elitism and decreasing it for replication as explained earlier. The difference between the two operations is only in the random characteristic of replication versus the deterministic

elitism. This change did not seem to have produced much effect in the algorithm, but clearly it did not affect it adversely. Given that the mutation rate was so high, the difference between performing the reproduction of the best individuals from one generation to the next by deterministic or probabilistic methods seems to have been greatly reduced. Actually, the high percentage of elitism may have balanced with the high mutation rate, thus the anticipated adverse effects of excessive mutation (such as loss of best individuals, or a decrease in fitness from one generation to the next) mutually with the anticipated adverse effects of elitism (such as a higher probability of falling in local minima).

3.3. The final system

The best individual (chromosome) produced by the genetic algorithms (which will be explained and analysed in next chapter) was embedded into a fixed program, called "Rutar", that was used hereafter as the path generation routine. This best individual is a behaviour rules tree used by the cellular automata to generate paths. The program "Rutar" is the result of executing this individual's cellular automata rules of death on a range maps. Since this set of rules is used identically in each automaton of a cellular automata colony, the program is useful for any number of automata, and is therefore completely independent of the resolution (or size of the map) point of view, giving a high versatility in these aspects. This means that it can be used to process maps of any size or resolution.

The generation of all the non-redundant paths in the map, which is an intrinsic characteristic of Rutar, is an advantage over other methods such as the work by [Tzionas et. al.] where only the first path found is provided as an answer, and no other alternatives are considered.

The results of this part of the work were reported in [Góngora et. al., 1994a] where the development process and details of the operation of the algorithm of Rutar were presented and published.

Rutar's parameters are the map, represented as a bi-dimensional array of bytes, and the size of the map, represented as two integer values holding the maximum values in the x and y axis. The map follows the structure shown in table 3.1 (a) where the obstacles and anchor points should be defined. Rutar creates the cellular automata colonies and executes the cycle of life. It returns the same map with the paths drawn in it as in table 3.1 (b).

Figures 3.14 shows the actual results of Rutar's applied to different maps. Appendix A shows the results of many more map examples, as well as the internet address where the program can be tested in many more maps or user defined maps.

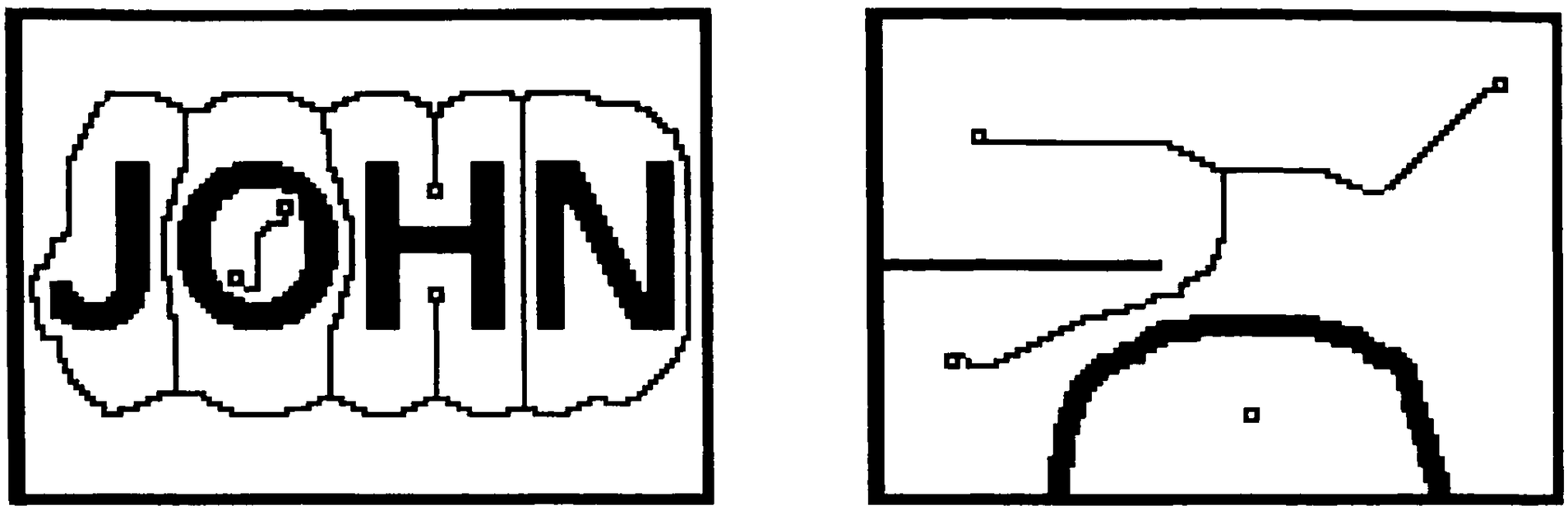


Figure 3. 14 Some examples of Ruta's results

Chapter 4

RUTAR

The development of Rutar involved the use of both of Cellular Automata and Genetic Algorithms coupled with techniques for path generation. A genetic algorithm was used to develop the behaviour rules of a cellular automata system, this behaviour was designed to produce a set of non-redundant paths. The development process presented in the previous chapter can be summarised as follows:

Definition of path generation technique:

- The paths were generated using Cellular Automata applied to a bitmap representation of the environment where the space, obstacles and robot are.
- To generate the paths, a set of rules of death for the cellular automata was needed.

- The set of rules has some key characteristics, which were summarised in one important rule (described in section 3.1.1.).

Development of the set of rules:

- The set of rules for the cellular automata was found by a search using genetic algorithms.
- The evaluation system and the termination criteria for the genetic algorithm were based on the characteristics defined for the set of rules.
- Two sample maps, for which the answers were partially known, were used by the genetic algorithm to develop the set of rules.

Embedding the set of rules in a program:

- Having found a set of rules that complied with all the path generation needs, it was embedded in a fixed program called Rutar.
- Rutar includes the input and output interfaces needed to execute the set of rules in the bitmaps for which the paths need to be found.

As explained in the previous chapter, after the best individual chromosome produced by the genetic algorithm was found, it was embedded in a program that executed this set of *rules of death* on a map represented as a colony of cellular automata. This program is called "Rutar". The function of "Rutar" is to take a standard bitmap file that represents the map where the robot has to find the paths, and execute the rules of death defined in the best individual chromosome from the genetic algorithm, as explained in the previous chapter. After executing "Rutar" the map has a representation of the non-redundant paths as *brick*

roads from each anchor point to the others.

"Rutar" is written in C and consists of a user interface that reads the name of the file that contains the map, reads this file and places it in an array in memory. Then, taking this array as the colony of cellular automata in the format presented in Table 3.1, it executes the rules of death until no more automata can die. It finally returns the array in another bitmap file, which is a representation of the map with the paths in it shown as the surviving automata. Appendix B includes a floppy disk containing an executable graphical MS-DOS version of Rutar, together with instructions for its execution to test it, and some example maps. An illustration of Rutar's input, processing and output structure is as follows:

Inputs:

- Name of the input file containing the map with obstacles and anchor points.
- Name of the output file where the map with the paths is going to be placed.

Processing:

- Load the input map into memory and fill up (represent) free space as cellular automata.
- Apply the rules of death to the map until no more automata can die.

Output:

- Save the resulting map (that now has the paths represented as brick roads) in the file specified as output.

The final set of rules of death, that resulted from the best individual (chromosome) from

the genetic algorithm execution presented in the previous chapter, showed a number of interesting features. For example, one part of the rules of death tree (in other words, a subset of the rules of death) acts on the behaviour of the cellular automata colony at an early stage of the cycle of life, when a large number of automata are alive, and another part of the tree acts in the final stage when the number of surviving automata is low. This will be analysed and discussed in the following sections.

4.1. The best chromosome: Rules of death

The best individual (chromosome) that resulted from the last run of the genetic algorithm and, which was used as the path generation rules in Rutar, was simplified and reorganised. This chromosome, represented in the LISP type syntax, results in the following expression:

OR (OR (OR (OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (1A)), OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (6A)), OR (OR (AND (AND (AND ((4A), (5A)), (3A)), (6A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)),OR (,AND (AND (AND (AND ((1A), (5A)), (6A)), (7A)), (8A))OR (AND (AND (AND (AND ((6A), (7A)), (8A)), (2A)), (1A)), OR (AND (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A))))))))), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6D)), (5A)), (4D)), (3D)), (2D)), (1A), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D)))))), OR (OR (OR (OR (OR (OR (OR ((8P), (7P)), (6P)), (5P)), 4P), 3P), (2P)), (1P))), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND

(AND (AND (OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A)),
(2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A),
(8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND
(OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)),
(6A)), OR (AND (AND (AND (OR (OR (OR (OR ((1A), (2A)), (5A)),
(4A)), (3A)), (6D)), (8D)), (7A)), AND (AND (AND (OR (OR (OR (OR
((7A), (1A)), (5A)), (4A)), (3A)), (6D)), (2D)), (8A)))))))))

This individual was the one that complied the most with all the fitness criteria defined in the genetic algorithm, and therefore it is the one that provides the most generic solution to a path generation process in a map.

After a careful analysis of this tree, a very interesting characteristic was seen. As shown by the different parts in bold in the following listings of the same tree, there are parts in the expression that use different criteria for the death of the automata. To observe these characteristics graphically, "Rutar" was executed with time pauses between each cycle of the scanning of the cellular automata. At the end of each cycle the graphics representing the colony and map were presented on the computer screen. When a path generation process is seen graphically and in *slow motion*, a two part shrinking process of the colony is observed, as shown in Figure 4.1. Appendix B includes the names of the files in which this *slow motion* version of "Rutar" can be run from the disc for illustration purposes. The following listings show the different interesting parts in the tree used for Rutar. These characteristics will be explained following the listings.

Listing 1:

OR (OR (OR (OR (**AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)),**

(1A)), OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (6A)), OR (OR (AND (AND (AND ((4A), (5A)), (3A)), (6A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)),OR (,AND (AND (AND (AND ((1A), (5A)), (6A)), (7A)), (8A))OR (AND (AND (AND (AND ((6A), (7A)), (8A)), (2A)), (1A)), OR (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A)))))))), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6D)), (5A)), (4D)), (3D)), (2D)), (1A), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))), OR (OR (OR (OR (OR (OR (OR ((8P), (7P)), (6P)), (5P)), 4P), 3P), (2P)), (1P))), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A)), (2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)), (6A)), OR (AND (AND (AND (OR (OR (OR (OR ((1A), (2A)), (5A)), (4A)), (3A)), (6D)), (8D)), (7A)), AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (4A)), (3A)), (6D)), (2D)), (8A)))))))))

Listing 2:

OR (OR (OR (OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (1A)), OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (6A)),

OR (OR (AND (AND (AND ((4A), (5A)), (3A)), (6A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)),OR (AND (AND (AND (AND ((1A), (5A)), (6A)), (7A)), (8A))OR (AND (AND (AND (AND ((6A), (7A)), (8A)), (2A)), (1A)), OR (AND (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A))))))))), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6D)), (5A)), (4D)), (3D)), (2D)), (1A), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))), **OR (OR (OR (OR (OR (OR (OR ((8P), (7P)), (6P)), (5P)), 4P), 3P), (2P)), (1P)))**, OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A)), (2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)), (6A)), OR (AND (AND (AND (OR (OR (OR (OR ((1A), (2A)), (5A)), (4A)), (3A)), (6D)), (8D)), (7A)), AND (AND (AND (OR (OR (OR ((7A), (1A)), (5A)), (4A)), (3A)), (6D)), (2D)), (8A)))))))))

Listing 3:

OR (OR (OR (OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (1A)), OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (6A)), OR (OR (AND (AND (AND ((4A), (5A)), (3A)), (6A)), (7A)), OR

(AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)),OR (,AND (AND (AND (AND ((1A), (5A)), (6A)), (7A)), (8A))OR (AND (AND (AND (AND ((6A), (7A)), (8A)), (2A)), (1A)), OR (AND (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A))))))))), **OR (AND (AND(AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6D)), (5A)), (4D)), (3D)), (2D)), (1A), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))**, OR (OR (OR (OR (OR (OR (OR ((8P), (7P)), (6P)), (5P)), 4P), 3P), (2P)), (1P))), **OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A)), (2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)), (6A)), OR (AND (AND (AND (OR (OR (OR (OR ((1A), (2A)), (5A)), (4A)), (3A)), (6D)), (8D)), (7A)), AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (4A)), (3A)), (6D)), (2D)), (8A)))))))))**

Listing 4:

OR (OR (OR (OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (1A)), OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (6A)), OR (OR (AND (AND (AND ((4A), (5A)), (3A)), (6A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)),OR (,AND

(AND (AND (AND ((1A), (5A)), (6A)), (7A)), (8A))OR (AND (AND (AND ((6A), (7A)), (8A)), (2A)), (1A)), OR (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A))))))))), OR (AND (AND(AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6D)), (5A)), (4D)), (3D)), (2D)), (1A), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))), OR (OR (OR (OR (OR (OR (OR ((8P), (7P)), (6P)), (5P)), 4P), 3P), (2P)), (1P))), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND **(OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A))**, (4D)), (8D)), (2A)), OR (AND (AND (AND **(OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A))**, (2D)), (4D)), (3A)), OR (AND (AND (AND **(OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A))**, (2D)), (6D)), (4A)), OR (AND (AND (AND **(OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A))**, (6D)), (4D)), (5A)), OR (AND (AND (AND **(OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A))**, (4D)), (8D)), (6A)), OR (AND (AND (AND **(OR (OR (OR (OR ((1A), (2A)), (5A)), (4A)), (3A))**, (6D)), (8D)), (7A)), AND (AND (AND **(OR (OR (OR (OR ((7A), (1A)), (5A)), (4A)), (3A))**, (6D)), (2D)), (8A)))))))))

The different parts of the tree can be seen as performing the following :

- The bold part in 1., evaluates to TRUE (let the automaton survive) when it is surrounded by more than 5 immediate live neighbours, therefore the automaton will

survive while it is not at the borders of the colony.

- The bold part in 2., evaluates to TRUE (let the automaton survive) when it is touching an Anchor Point (adjacent to it).

- The bold part in 3., evaluates to TRUE (let the automaton survive) when it is in a position that will disconnect the colony. That is, the path is already one automaton wide in that place.

- The bold parts in 4 (sub expressions of the bold part shown in 3.), will be operative when there is a large number of automata isolated in dead end paths. Thus this part is especially important at the end of the shrinking process. When a path generation process is seen graphically (Figure 4.1 a, b, c and d, e and f), it can be observed that this part is responsible for eliminating a lot of spurious branches that are left during the shrinking process (step e to f). This part evaluates to false when an automata has only one neighbour and this, in turn, has less than three neighbours left alive.

This set of rules of death produce the effect of a shrinking colony that moves toward its centre of mass, since it shrinks from the borders toward the centre until there is a line of automata, only one automaton wide, it ends up forming a brick road between the anchor points. Figure 4.1 (a) is the original map with the three anchor points. Once the free space has been filled up with the cellular automata, it starts to shrink under the rules of death. Figure 4.1 (b) shows the colony after a few cycles of life, and (c) through (e) show

intermediate stages. After a cycle of life where no automata die, the program stops with the paths generated as shown by Figure 4.1.(f).

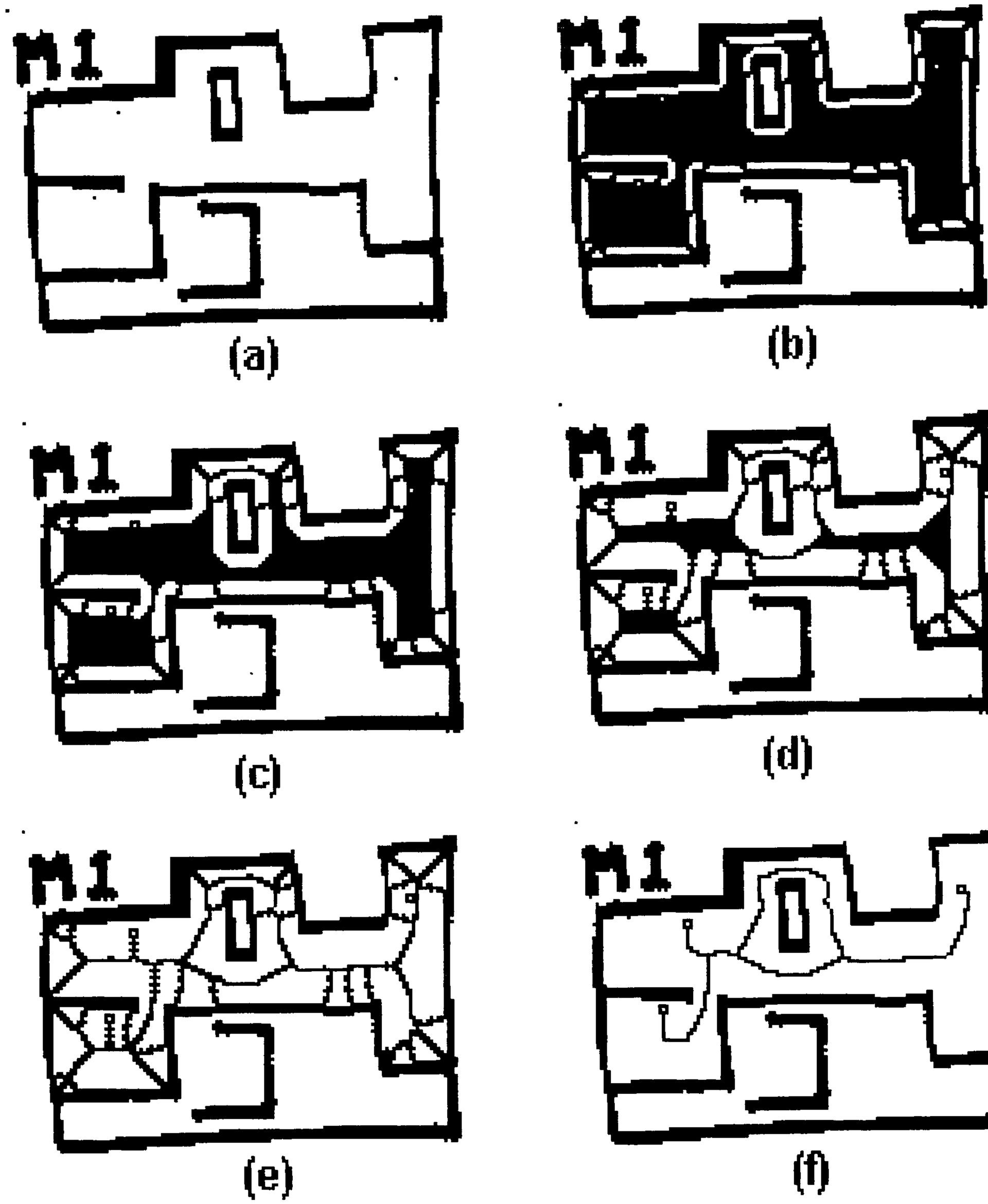


Figure 4. 1 Shrinking process of a map using Rutar

To use this set of rules of death in the maps used for path generation, "Rutar" was developed to integrate the user interface, the file management, and the rules of death in one program. Figure 4.2 shows a flowchart of the functional structure of Rutar.

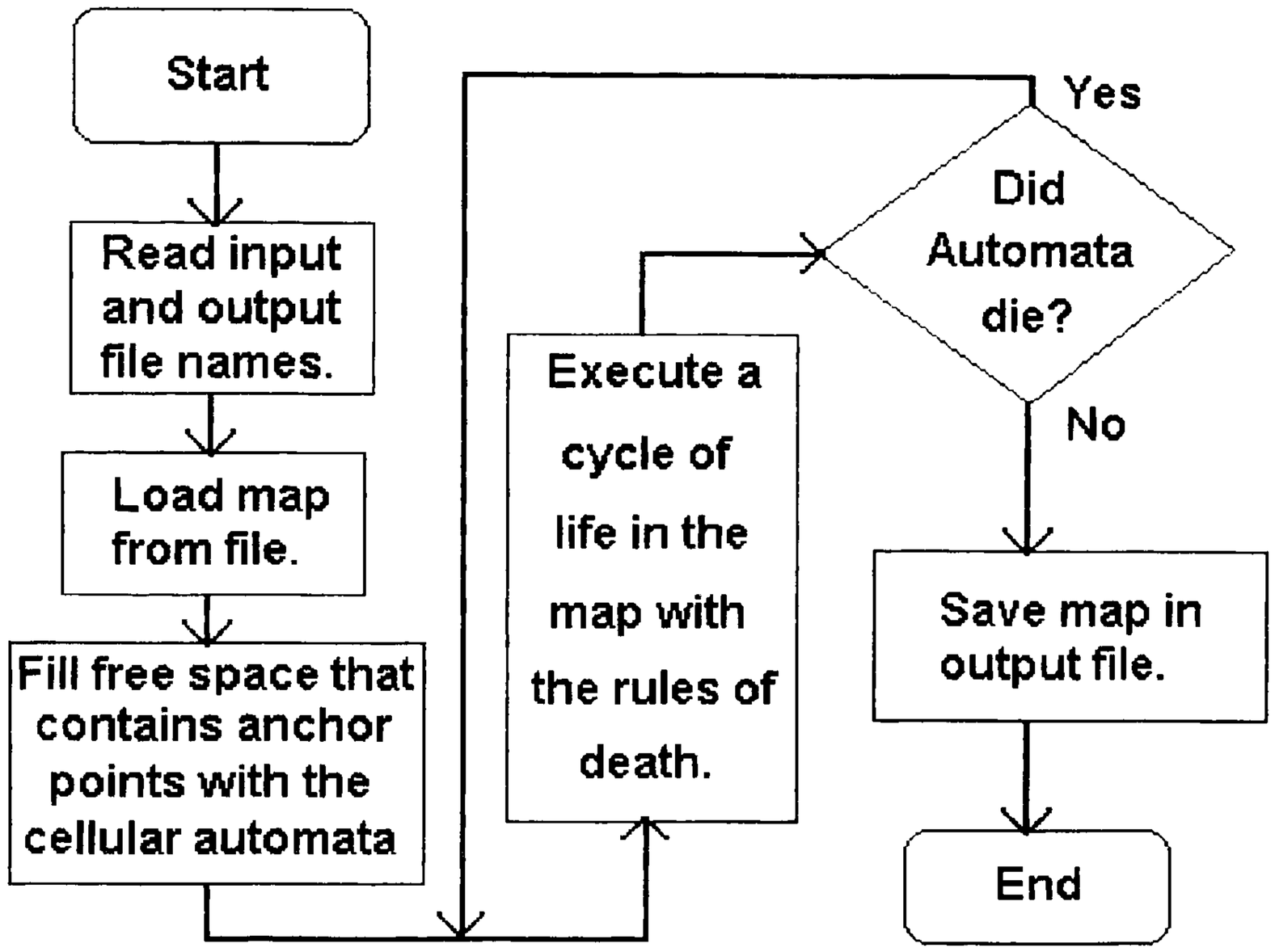


Figure 4. 2 Functional flowchart of Rutar

4.2. Characteristics of Rutar

The main characteristic of "Rutar" is its intrinsic reliability (which will be discussed later in this section) in exploring all possible main routes that can be followed to get from any point to any other. The answer given by "Rutar" is not the actual list of paths, but the *brick*

roads that lead to all possible destinations, as shown in Figure 4.3. This figure shows the answer given by "Rutar" to a map JOHN, with the anchor points as defined in Figure 3.10. At this stage this map is not actually showing which path is which, it merely gives all the possible ways to get from one point to another. As seen here, it is not possible (by following the brick roads) to get from a point in the H to a point in the O. It also can be seen that the path inside the O is not a straight line, it is curved toward the centre of the O.

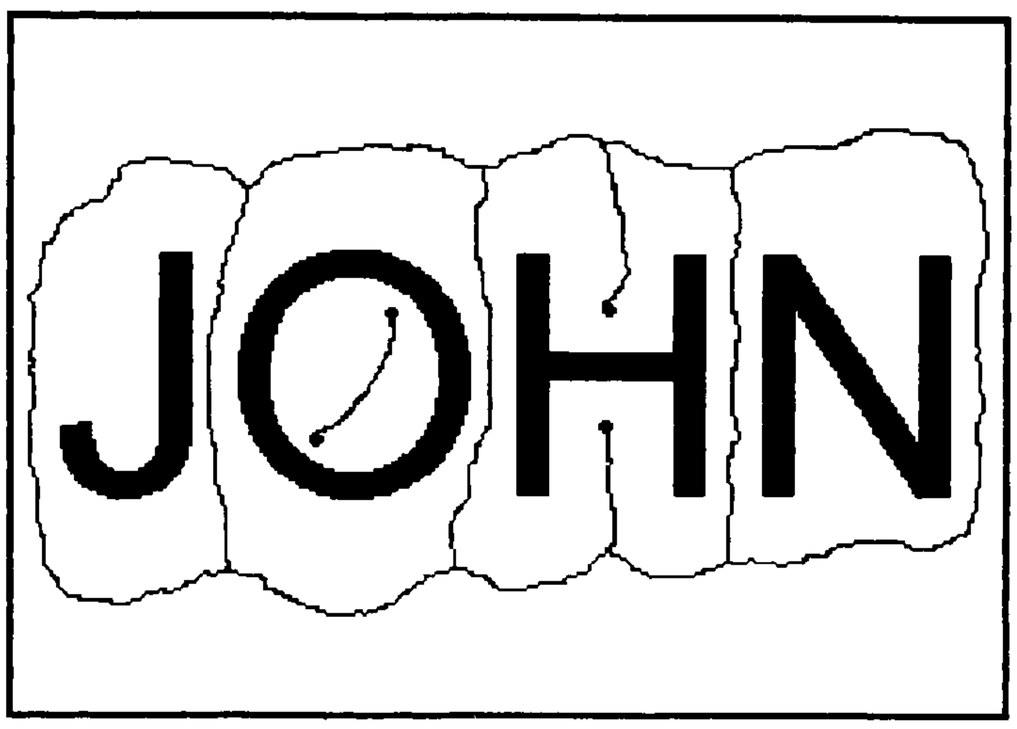


Figure 4. 3 Answer of Rutar

From this map, all possible non-redundant paths can be extracted by using the algorithm described in section 3.1.2.3. This algorithm can be included in the same program that generates paths for an implementation of the whole system in a robot. For the moment, for the purpose of clarity during the development, all modules are treated as separate programs. It is after the use of this algorithm that the actual paths are fully defined, by separating each one of them from the mesh produced by "Rutar". Before using these paths for optimisation, which will be described in chapter 5, at this stage the main characteristics of the path generation can be analysed from the examination of the raw output maps (such

as the one in figure 4.3 or the sequence in figure 4.1).

The first indication that the paths shown are not redundant is given by the fact that within one area of free space, the *brick roads* have no loops or additional branches. For instance, there is only one path between the points in the O, where an infinite number of possible paths exist, but all of them would be redundant in relation to the first. In the paths between the points in the H, it is also clear that the branching occurs only when an obstacle is present, therefore, there are paths only where there is a non-redundant alternative.

Conversely, all possible ways to get from one point to another are found. This can be seen by observing and analysing the process that "Rutar" performs (referring back again to figure 4.1). Interpreting the result of the path generation process the following conclusions can be reached on the generic nature and repeatability of the Rutar:

- First, it fills the space connected to the anchor points with the colony of cellular automata, as shown in figure 4.4. In this figure, **all** the space that is possible to reach from the anchor points in the H is filled with the colony of cellular automata.

- Then, the rules of death are applied until no more automata can die, shrinking the colony to a single automaton wide mesh. Since these rules comply with the condition that the colony **must** remain connected (all the automata must remain connected to the original single colony), the brick roads will travel through **all** the reachable space that was originally considered, thus it will establish **all** the possible routes.

- In other words, by filling **all** the connected (reachable) space with the colony of cellular automata, an exhaustive search (scan) is performed, giving the system the possibility to guarantee that it can find **all** possible paths.

The key point to observe is the fact that an **exhaustive search is performed**. This means that **it is possible** for the system to find all non-redundant paths that can exist. Still, knowing that the system has the possibility to be completely reliable, does not prove that it actually is. A formal proof was considered to be either impossible or very difficult, so instead, thorough and exhaustive testing was used to provide evidence of its reliability. Rutar has been tested with many maps, some of them shown in Appendix B, some others in the enclosed disk, and many more in the web page indicated also in Appendix B.

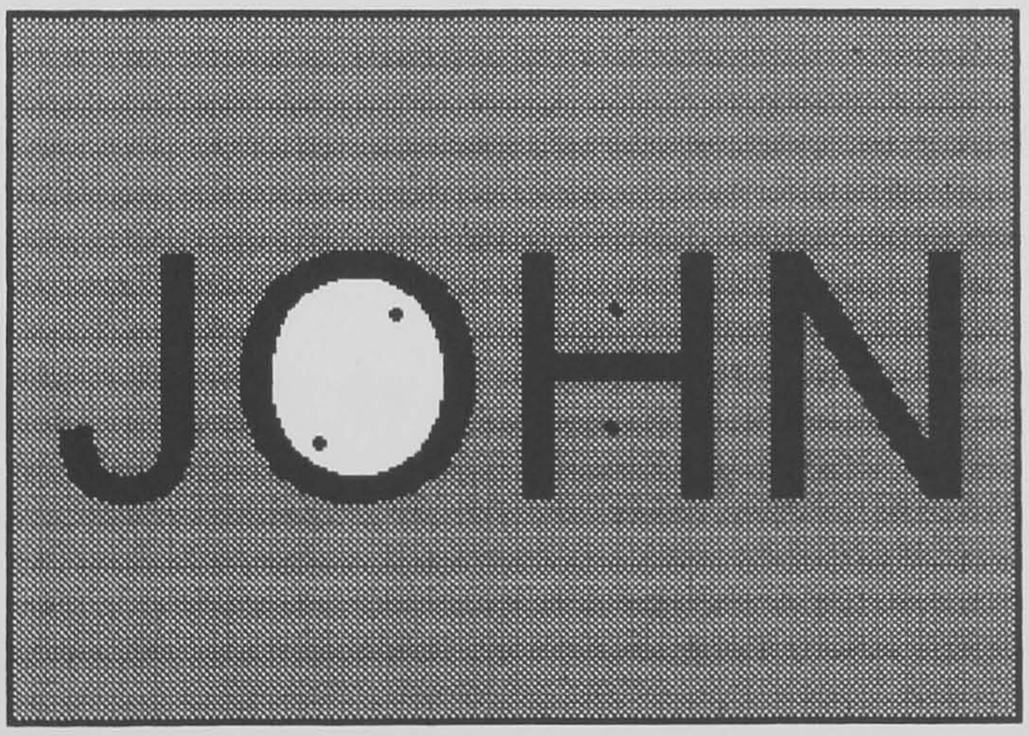


Figure 4. 4 Cellular automata colony for a free space region

Figure 4.4 shows also that the program does not lose any processing time looking

hopelessly for a path between a point in the H and a point in the O. This can be seen clearly since the colony of cellular automata that encloses the two anchor points in the H is completely disconnected from the colony inside the O, therefore, the rules of death will be applied to both colonies as if they were two different problems, having no interaction between them. This can be further illustrated by referring back to figure 4.1, where it is clear that the colony shrinks on its own, so that it is irrelevant if there is another colony separated from it, even if it has anchor points in it.

Another interesting characteristic is the memory requirements to execute this program. Because no global history on the process has to be saved, and no recursive search needs to be performed during the generation, only the memory necessary to store the map is needed. This is because the data needed to store the bitmap figure, regardless of what it represents, can be used to represent the obstacles, the free space, the anchor points, and the cellular automata (including their intermediate states). Therefore, the memory requirements are exclusively dependant on the size of the map. Once the map's size has been defined, the memory usage is fixed independent of the complexity of the paths, the number of paths, anchor points, obstacles, etc. For the case of the maps used by Rutar, the memory necessary to represent a map is one Word per pixel of the map, thus, a 256 x 256 pixels map needs 64K Words. Depending on the computer, the operational system and the compiler used to execute Rutar, the memory can be organised in 8-bit, 16-bit, etc. words. A system capable of using Bytes would need 64K Bytes for the 256 x 256 pixels map, a system using 16-bit integers would need 256K Bytes to store the same map.

The processing time required to execute the rules of death shows an even more interesting and potentially useful characteristic: the more obstacles there are (which generally means a more complex problem to find paths using other methods) the faster the processing will be. This happens because the rules of death are applied only to live cellular automata, therefore, if the space is very crowded with obstacles, the number of individual automata in the colony is less, therefore the processing time will be shorter. To illustrate this, figure 4.5 shows two maps of the same size (area) of 128 x 128 pixels. Map (a) has a few thin obstacles, while map (b) has a more and larger obstacles (and thus there are more alternative non-redundant paths). Since the colony in map (b) has considerably less automata, nearly half of those of the colony in map (a), the number of evaluations in map (b) is nearly half that of map (a), therefore, the execution time in the maps is inversely proportional to the number of pixels occupied by obstacles because it is directly proportional to the number of cellular automata to evaluate. Therefore for these examples, the execution time for complex map (b) is nearly half that of the simpler map (a).

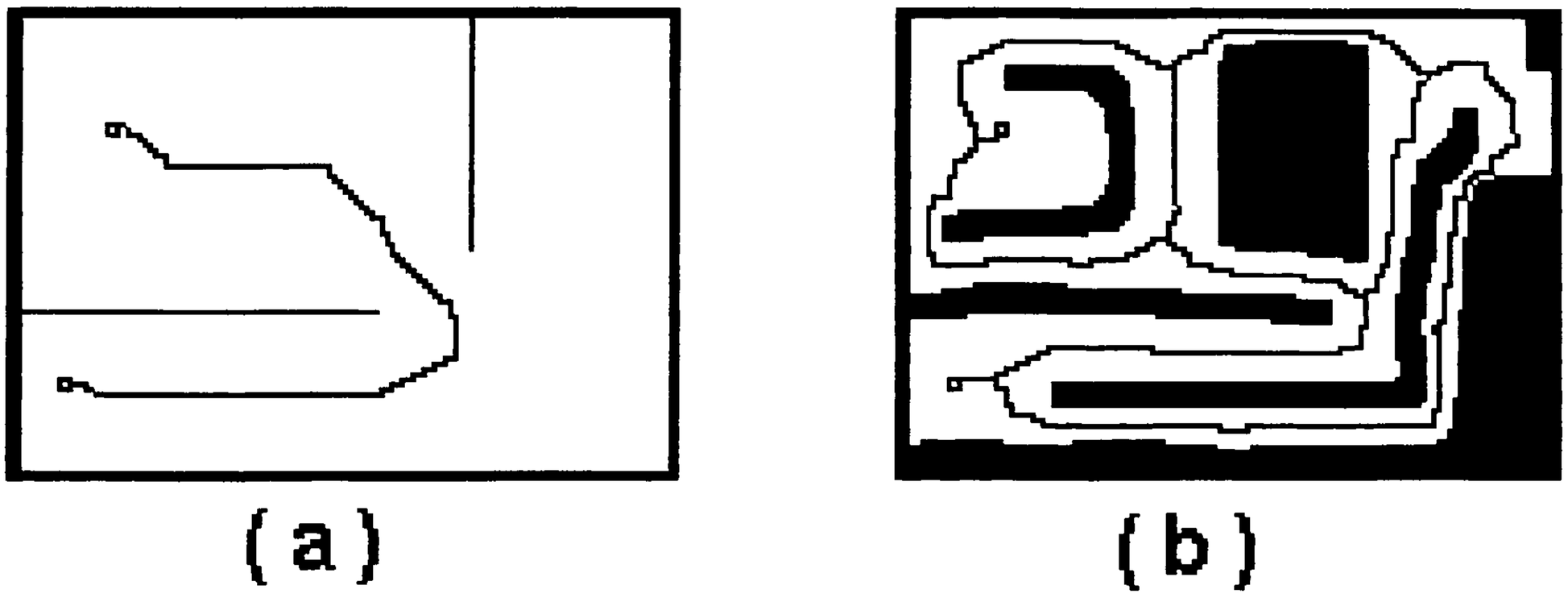


Figure 4. 5 More obstacles mean less automata to evaluate

A further system characteristic, that was observed from the first trials to execute the genetic algorithm presented in chapter 3, is its reluctance to minimise the number of surviving automata. This was the reason why the genetic algorithm terminated because of time-out in the initial executions. By observing the many figures that show the routes generated, it is clear that the rules of death will not be able to minimise distance simply because the final colony (paths) tends to be in the centre of mass of the original colony, so that the shape of the path depends more on the shape (contour) of the surrounding obstacles rather than on the location of the anchor points. The example in figure 4.6 shows this clearly, the shape of the path follows the curved shape of the inner walls of the obstacles despite the fact that it is possible to draw a straight line between the anchor points. This happens because the colony, that initially is all the free space inside the box, shrinks from the borders toward its centre of gravity, and the shape of the borders of the colony are determined by the shape of the obstacles surrounding it.

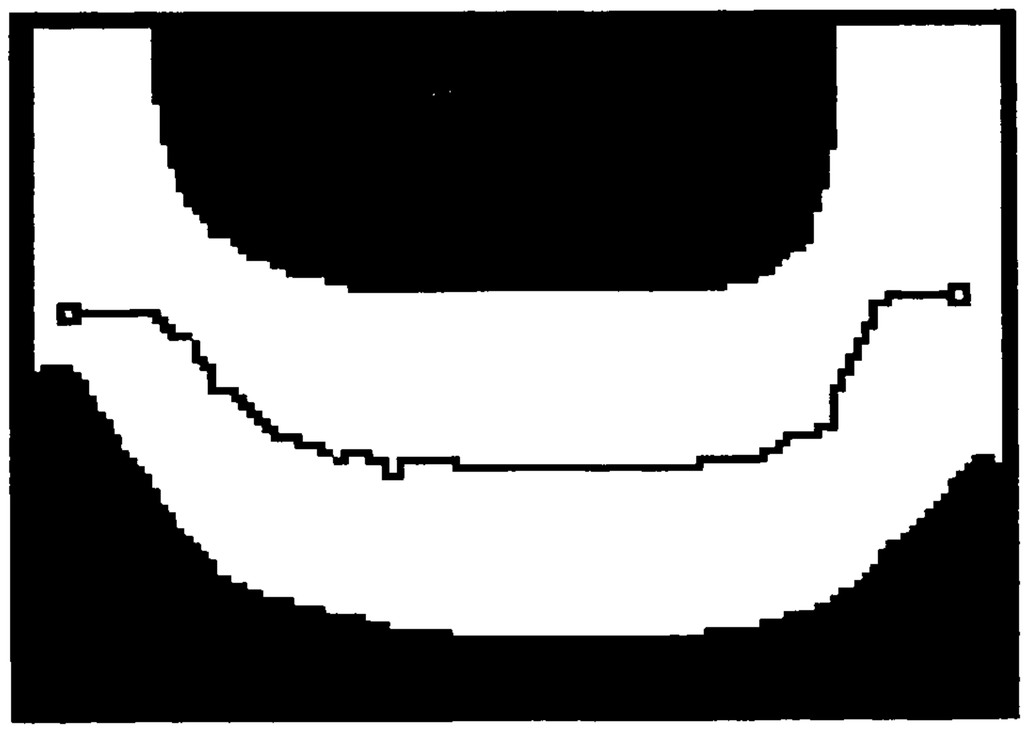


Figure 4. 6 Paths follow the centre of gravity of the free space

For the case of the trivial path, between the points in the O of the maps JOHN, the shrinking toward the centre of gravity and the fact that the scan is sequential, account for the curving of the path. For a path between the points in the H, for example the path going in between the letters O and H, it can also be deduced that for the rules of death to find the shortest path (a straight line from the points to the corners of the H and then a path bordering the H) the rules of death would have to *know* where the points were, and then shrink the colony selectively in that direction. But this is impossible to achieve since the rules apply identically to all automata, regardless of their location and with information only on the immediate neighbours of each one. Therefore it is impossible for an automaton in the middle of the route to know where the path comes from, or where the anchor points are. This was the reason why, despite the large number of generations that were allowed for the genetic algorithm presented in chapter 3, the shortest path was never achieved. So for this particular path, it can be seen that the part of the path that crosses between the O and the H curves a little away from the O (see the JOHN maps in figures 3.14 and 4.3).

After the application of "Rutar" in a great many tests, it has been found to be a time efficient and highly reliable path generation routine. Since the map's structure is very simple, and the rules of death of the cellular automata are represented as logical rules in a binary tree, the execution can be performed on a variety of machines, ranging from some high-end microcontrollers used as on-board computers, to powerful workstations where graphic and user interface routines can be added to make the program very friendly, didactic, and useful for the analysis of the path generation process. In the web page

indicated in Appendix B there are executable versions of Rutar for Unix systems, MS-DOS and T800 series transputers.

Once the bitmap file is saved by "Rutar", the recursive program presented in section 3.1.2.3. isolates the individual paths to be used later in the optimisation process that will presented in the next chapter.

As mentioned already, Apendix B includes the web page address where Rutar can be found for execution and testing. At this address, various versions of Rutar, not available in the enclosed disk, can be found. This web page includes an executable for Sun workstations, which can handle very large maps and a non-graphical executable for PCs (which executes at the real speed of Rutar as compared to the version in the disk). With all these versions of the program, any researcher in the field is invited to test the system and use it to benchmark any other method of path generation.

4.3. Use of Rutar in Dynamic Path Generation

The versatility that "Rutar" provides with respect to the number of automata in the map, gives an interesting versatility in computational speed vs. resolution of a map. The resolution used in a map can be perceived as accuracy in path generation. This makes "Rutar" very convenient for real-time applications since it enables the user to choose a lower resolution for a map to get an answer faster. By trading off resolution, speed can be

gained.

The algorithm is very fast, for example a map of 256 x 256 pixels with no obstacles (the worst case since the maximum number of automata must be evaluated and eliminated) takes less that 200ms to process on a 133MHz pentium based PC computer. This is considered to be a very good resolution in many applications, since, if a pixel is scaled to the size of a robot, and using a robot size of 50cm (its maximum diameter looking at it from the top as a round object) it would give a map size of more that 100 metres on each side. Or if scaled to 3cm per pixel (the width of a rubber tyre in a typical mobile robot), it would give a map size of more that 7 metres on each side. In the onboard computer of a robot it can take even less time if the processor is completely dedicated to this task and does not have the high overhead in terms of its own operating system.

In addition to the efficiency of the rules of death, the use of cellular automata makes the algorithm suitable for parallel processing. If the mobile robot is implemented using a parallel computing system, the execution time of the path generation stage can be reduced even further. Thus making the system even more suitable for real-time applications.

The author believes that Rutar is one of the most reliable and fast static-path generation programs available at the time of writing. However the path is not optimal with respect to any particular criteria and optimisation will be considered in the next chapter. The possible application of Rutar as a tool for on-line optimisation is discussed at the end of the work.

Chapter 5

PATH OPTIMISATION

Having developed Rutar, which was presented in chapters 3 and 4, the path that it generates now has to be optimised. In the optimisation process the characteristics of the mobile robot must be included. As explained in section 1.2, in this thesis, the complete characteristics of the robot are very important in optimisation since this will enable the process to be taken a step further, from simple distance optimisation (where only the geometrical characteristics of the robot are considered), to complete optimisation with respect to additional criteria such as energy, travel time and risk. To use these aspects as optimisation criteria, a weighted combination of these is used to determine the best solution. Section 5.1 presents the scheme that was used for optimisation, section 5.2 presents the way in which the paths are represented for their optimisation, and section 5.3.

explains the way in which the different optimisation criteria are represented and combined. Section 5.4 presents the experimental results for their analysis later in chapter 6.

The characteristics of the mobile robot are used in the optimisation process by means of a simulator. The simulator is used to calculate the result that the real robot would produce when following a given path, so that the path can be assessed in terms of energy, travel time, risk and distance. Thus the more precise the simulation, the more precise the optimisation will be. The simulator has to include the size and manoeuvrability of the robot so that a precise evaluation of the robot's ability to circumnavigate obstacles can be performed. Obviously, it also has to include the ability to calculate the energy consumed by the mobile robot when performing a given path. The time spent travelling is calculated indirectly from the robot's speed and acceleration, and the path distance. The precision of the simulator, in terms of providing the speed and acceleration data, directly affects the precision of the time calculation. The last aspect concerning the simulation process is the way in which the simulator reads or interprets the path provided by Rutar. For the optimisation to be useful in practice, the simulator not only has to simulate the real mobile robot as faithfully as possible, but it also has to interpret the path in the same way that the robot will interpret it. Section 5.3 explains, in more detail, the way in which the simulator is used and the consequences of its performance on the result of the optimisation process.

If the interpretation of the path is properly simulated, important conclusions can be drawn. For example if the optimisation process yields a solution (the optimal answer) that the mobile robot cannot perform in practice, and thus approximates it to what it can

accomplish physically, this may still result in the optimal path in practice. If the simulator is very accurate (both in simulating the robot's interpretation of the path and the robot's dynamics) the path that has been chosen as optimal is likely to be so, even if the robot has to use an approximate path so that it will correspond with the movements it is capable of executing. This will be clarified later in section 5.3 when the optimisation process is explained in detail.

The optimisation criteria used have to be included in the optimisation process in such way that more than one criterion of the criteria (distance, risk, energy and travel time) can be used simultaneously. For example, an optimisation could be performed *mainly on energy minimisation but also to a lesser degree on time minimisation.* In this work this was accomplished by using each criterion in terms of a percentage of importance. In this way, a 100% energy efficient path will ignore travel time and risk completely, while a 33.3% weighting for each of the three criteria, would produce a path that is balanced in terms of the three of them as if each had the same importance as the other two. In this way any combination of trade-off can be performed between the possible criteria.

5.1. Path optimisation scheme

The optimisation of a function implies finding the minimum (or maximum if that is the case) value that the function can take within its solution space. Depending on the type of function, this can be achieved analytically or by calculating the inverse of the function.

Some functions, such as the energy consumed by a mobile robot following a path, are too complicated to be solved in either of these two ways because the inverse of the function is not easy to find. The solution then, is to search in their solution space by some other method that uses the original function directly.

There are many mathematical search methods, divided mainly between exhaustive and non-exhaustive [Goldberg, David E.] [Peitgen]. The exhaustive methods explore all the possible solutions and simply choose the best. But if the number of possible solutions is infinite, or just too large to wait for an exhaustive search to be performed (if the solution space is discrete and finite), a non-exhaustive search method has to be used. When using non-exhaustive search methods two main problems arise: how to know when to stop searching, and how to avoid falling into local minima.

Genetic algorithms are very good at avoiding the local minima when used as a non-exhaustive search method [Goldberg, David E.]. They have already been used in this work to find the rules of death for the path generation system (Rutar), and now they are going to be used to perform the optimisation. The problem of how to decide when to stop searching is up to the user who has to define the termination criteria.

It is important to note that in the case of the path generation system the genetic algorithm was used once to find the path generation rules, and thereafter, these rules are used every time a path needs to be found, but no more genetic algorithms are used. In the case of the optimisation system, the genetic algorithm is going to be used directly to optimise,

therefore, it is going to be executed every time an optimisation is needed.

For a mobile robot moving in a space with obstacles, there are in theory an infinite number of possible paths to chose from. In practice, and after representing the map in a digitised format, the number becomes finite but still with a very large number of possible paths, that makes an exhaustive search impractical. A genetic algorithm can be used to search for the optimal path in terms of chosen criteria. Genetic algorithms were chosen as the best tool to use because their ability to avoid local minima, their good performance in other experiments by the author (such as the finding of the rules of death for "Rutar"), and their good performance reported by other authors [Davis] [Koza] [Marik, V. et. al.] [Meng, QC. et. al.] in various search problems.

In a typical application of a genetic algorithm, an initial population of randomly generated solutions is used as the starting point of the search, such as the case presented in Chapter 3. In the case of the path, using randomly selected paths would mean that the search would have to include, looking not only for the optimal path, but also looking for a feasible path. Since the path generation routine is already available, and the paths generated by this routine are reliable feasible paths, the initial population of paths for the algorithm can be taken from these generated paths.

In a genetic search, it is assumed that the population is composed of a relatively large number of individuals. But from the path generation routine, even if there are a number of alternatives, it is most likely that the number of alternative paths generated is not as large

as expected for a genetic search. For example, a typical population size for a genetic algorithm is from a few hundred individuals to a few thousand (see end of chapter 3 for the case of Rutar), while the number of redundant paths in the example maps presented was only 10 for the most. Moreover, since the generated paths are the non-redundant alternatives, this would more correctly represent a sample of each of the possible local minima available. Thus, although the population must be generated from the path generation results, it has to be converted into a suitable number of individuals, enough to provide the genetic algorithm with the required variety.

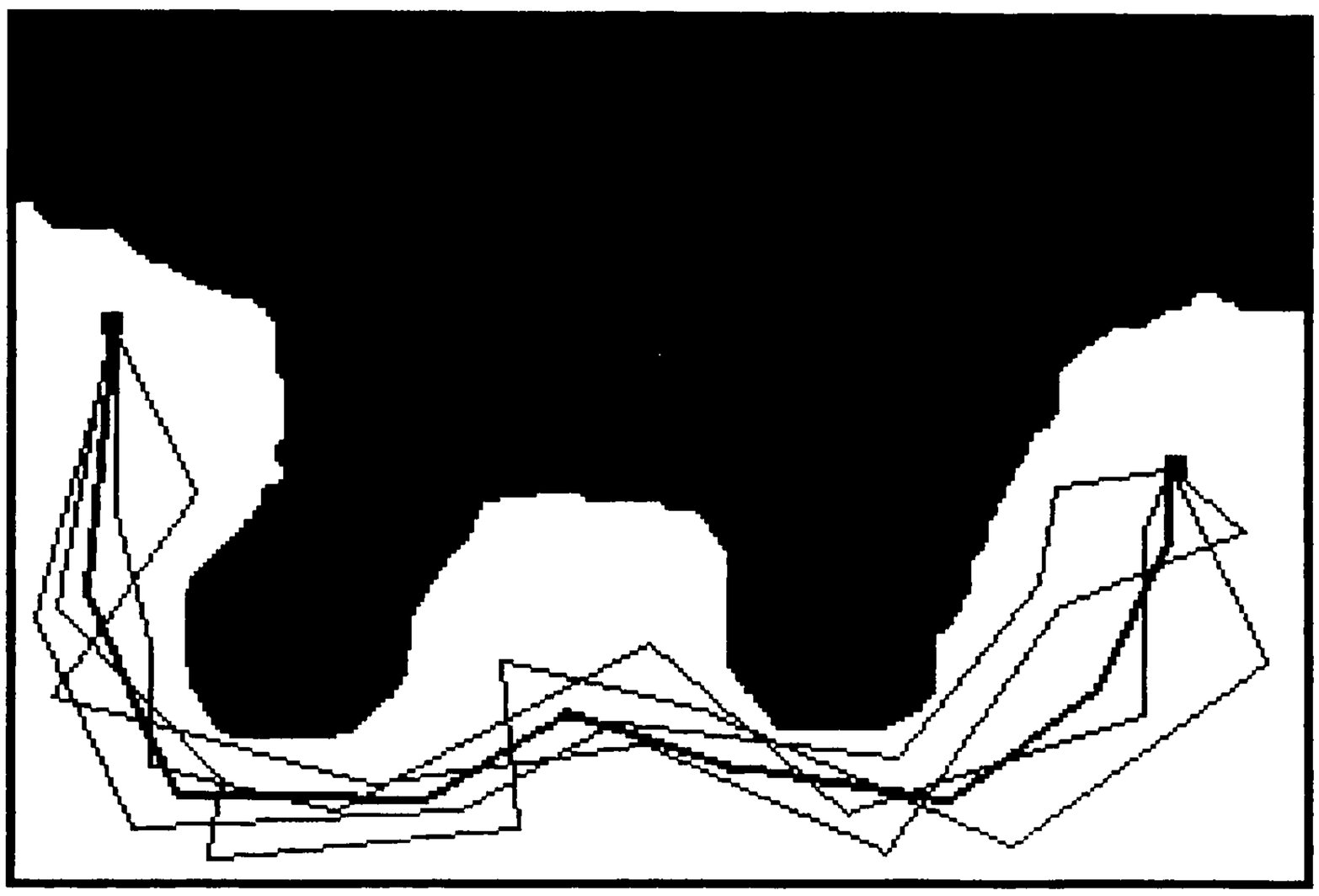


Figure 5. 1 A path and it's variations.

The scheme used to provide the genetic variety in this optimisation process is to test different forms, or versions, of a path to look for the best one, in other words, to look at many redundant forms of each path. Figure 5.1 shows an illustration of this for the case of

a map with one possible non-redundant path. The bold path would be the one provided by Rutar, and the others are variations tried during the search for the optimal path. After trying a large number of variations, the genetic algorithm should find the optimal path. The way in which these variations are produced is explained in section 5.2.

In the cases where the map has more than one non-redundant path, an optimisation process will be carried out with each non-redundant path, and at the end the best of the different non-redundant optimal paths will be selected. In fact, the optimisation of each non-redundant alternative could be performed completely independent of the others since no intermediate paths could be created from combining different alternatives. If each of the non-redundant paths are optimised at the same time, or independently one after the other, this has no effect on the final answer, it also has no effect on the amount of computer power used (the total CPU time). This is because at the end, that same number of individuals (possible optimal paths) are going to be evaluated, regardless of the order in which this is done. Since genetic algorithms have a very parallel nature, the complexity of the algorithm is also independent of the order in which the optimisation of the non-redundant paths is done. Therefore, it is going to be assumed that the optimisation is performed independently for each non-redundant path, and that if the computer platform is capable of parallel processing (or at least of concurrent processing), the optimisation can be done simultaneously for all the paths.

The development of the genetic algorithm for the path optimisation is going to be presented in the following sections. In this presentation, the optimisation of each non-

redundant alternative is going to be addressed as a complete optimisation process, and explanations will be based on the optimisation of one of these single alternatives. The final optimisation will be called the global optimisation solution, which will be extracted at the end of the whole process. This extraction is done by a straight forward comparison between the results from each non-redundant path, the best of these optimal solutions will be chosen as the global optimum. Figure 5.2 shows a flow chart representation of this process.

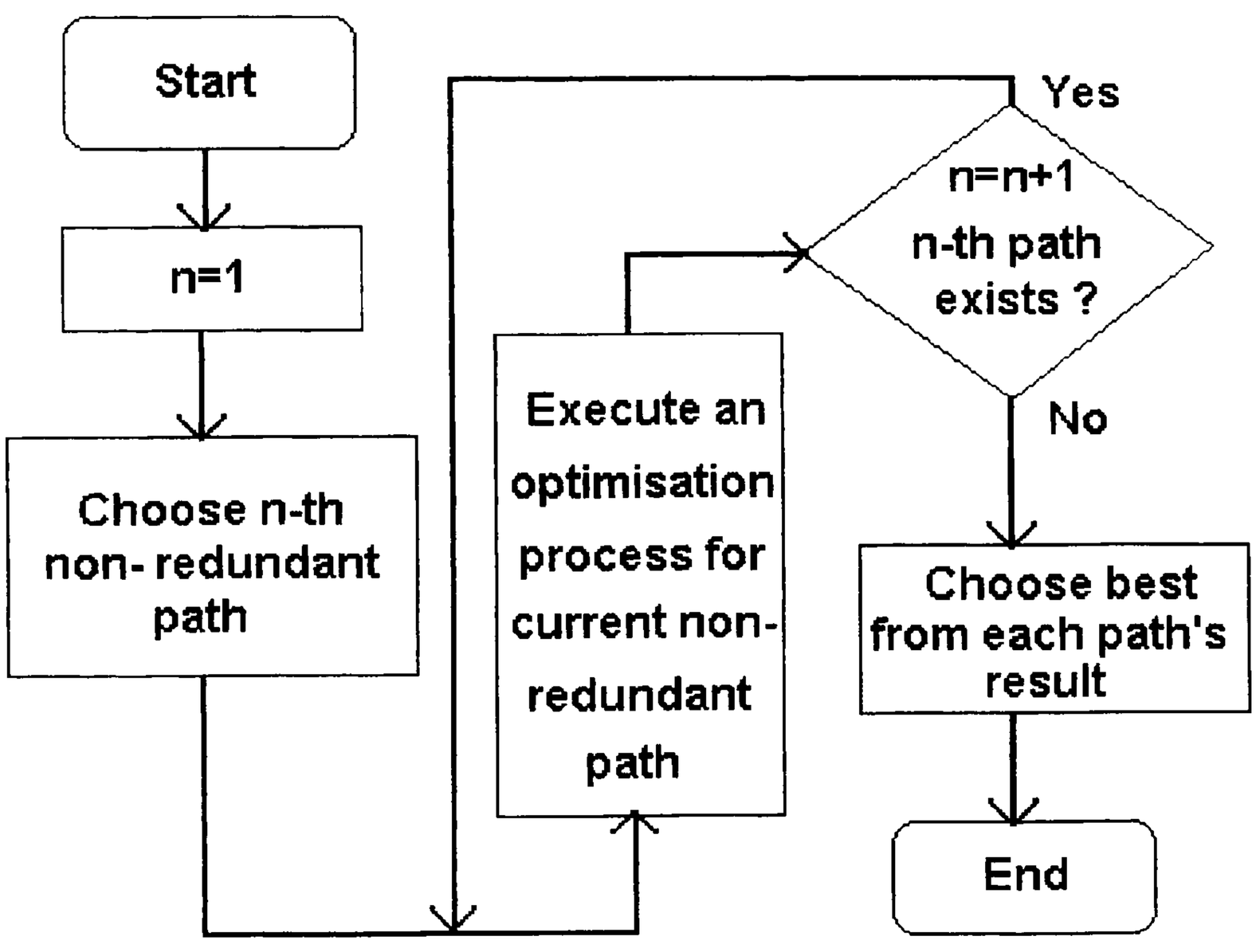


Figure 5. 2 Flowchart of the optimisation process.

5.2. Generation of the Initial Population: the chromosomes

Since the initial population for every optimisation process is one path (each one of the non-redundant alternatives presented by "Rutar"), multiple variations have to be generated from this path to spawn the initial population for the genetic algorithm to work with. The number of individuals in the population is defined at the moment of executing the process, this will be seen in section 5.4, for the moment the point is, that since genetic algorithms work on *populations* of possible solutions to search for the best answer, a population has to be generated from the path provided by Rutar.

The best way to generate this large number of individuals is to make copies of the original path, and have each copy mutated in a different way (the mutation process will be explained in detail further in this section, which is the same mutation process used during the execution of the genetic algorithm). Since mutations are made randomly on each copy, this would be similar to generating the initial random solutions, with the characteristic that all the individuals in the initial population will be feasible solutions, although not necessarily optimum. The mutations have to be performed so that all the new paths are also feasible ; this can be achieved by controlling the mutation process so that the mutated paths do not run into obstacles, or by checking the feasibility of the mutated paths and discarding the unfeasible ones and repeating the *copy and mutate* process to replace them.

This scheme of *copying and mutating* from a single initial possible solution, which was explained above, was an original development made in this research to generate an initial

population for the genetic algorithm used for the optimisation process. This novel approach allows a population to be generated easily when and initial example solution is available, thus eliminating the need for the generation of an initial totally random population. The example of figure 5.1 illustrates the case of an initial population generated by making many mutated copies of the original path presented by "Rutar".

To mutate the original path, a chromosome structure has to be defined, as well as the way in which the genetic operators will be used. The original representation of the path used by Rutar (representing the paths as *brick-roads* by the state of the pixels in the map) can be used as the chromosome structure. Alternatively, a representation to define a chromosome that will make it simpler to operate genetically can be used: it consists of approximating the pixel-based paths to straight-line segments. In doing this, a problem mentioned previously would be encountered: a series of straight line segments implies sharp corners that, if the robot is forced to follow them exactly, could stop it from achieving an optimal path (depending on the robot's characteristics). To solve this problem the straight lines can be used to describe a language based on movement definitions rather than describing the actual exact path that has to be followed by the robot. The solution is then to use a language such as Mitchi, as described and used in [Kanayama Y. et. al.] and [Goodhead, T.C. et. al.].

Using the straight lines to describe the movement of the robot means that the robot will not follow the drawn path, but the described path as interpreted and executed by every particular robot. The actual shape of the path can be smooth, as defined with the Mitchi

language and if the robot's way of following the corners is to make them smooth, similarly, the simulator has to calculate the results that the robot would produce using the same Mitchi language description, and obviously producing a smooth path if that is what the robot would have done.

In other words, the paths drawn represent a description of the path, but the actual paths executed depends on the way in which a given mobile robot will interpret the path and on how it will alter it to accommodate its own mechanical and dynamic characteristics. It is the actual values of the applicable variables from the actual robot's trajectory, that are the ones that are going to be used in the evaluation of the individual paths by the genetic algorithm. As will be explained in the next section, it is necessary for the simulator to have the capacity to use the same language as the robot to analyse the paths, so that the correct values are calculated.

Having described the path with straight line segments, the chromosome structure will consist of the list of segments, represented as vectors having a direction and a length (rather than a magnitude) as shown in figure 5.3. In this way, the genetic operators will act on the vector list in the following way:

- **Crossover:** is performed on two parents (chosen individuals according to their fitness) by randomly selecting a corner on the route of one of the parents, thus a vector origin (which corresponds to a change of direction from one segment to the next), searching the nearest vector origin of the other individual that will be the second parent, and then

interchanging those origins of vectors to form two new offspring individuals. Figure 5.3 illustrates this procedure. Using this scheme as shown in figure 5.3, four possible combinations can be achieved (thus 4 offspring could be produced), but since the population has to be kept constant for the genetic algorithm, only two (any two) of this combinations will be performed.

- **Mutation:** is performed by randomly generating a vector, adding it to a randomly chosen vector origin in the mutating individual, and replacing it. Figure 5.4 illustrates this procedure. This mutation scheme is the one used to create the initial population as well.

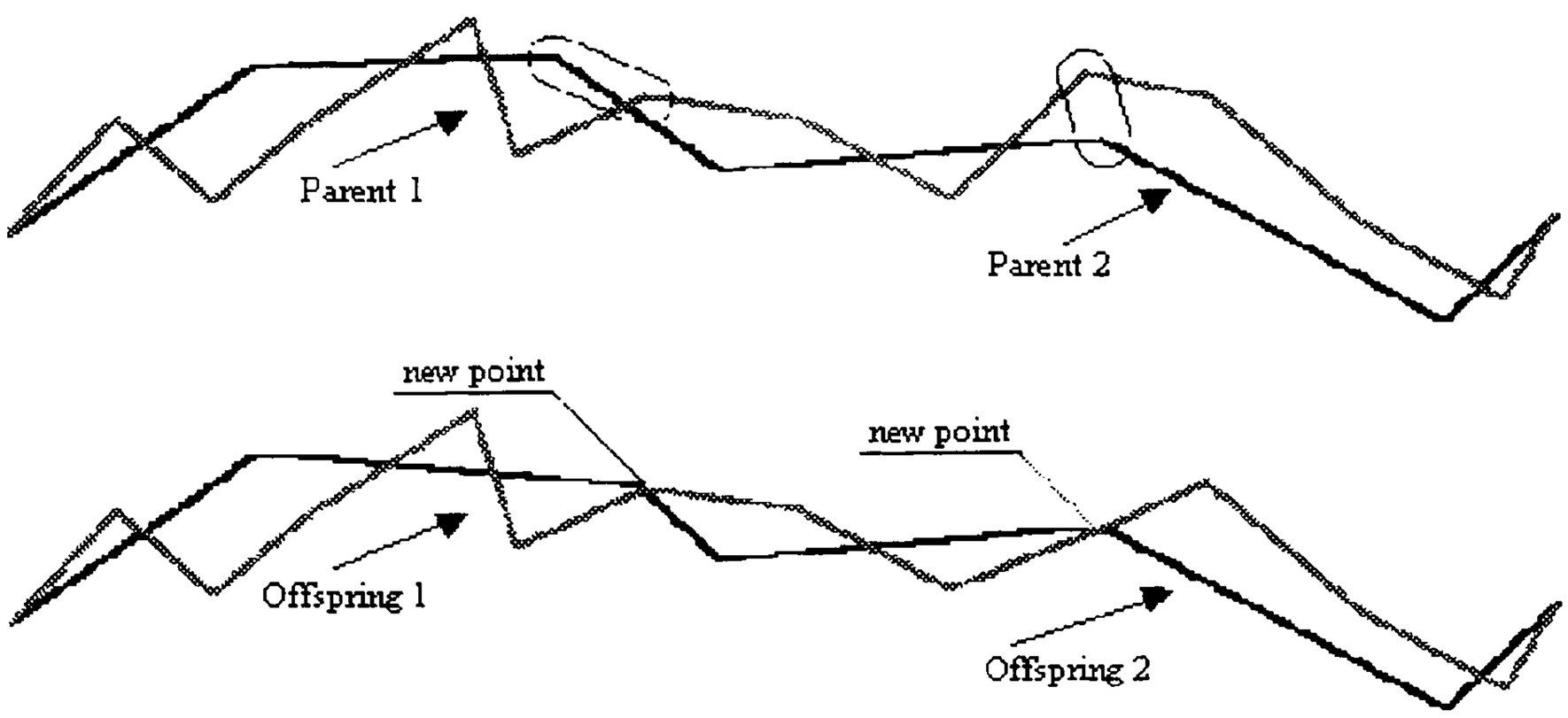


Figure 5. 3 Crossover operation

Using this chromosome structure and these genetic operations, the optimisation is performed as explained in the following section.

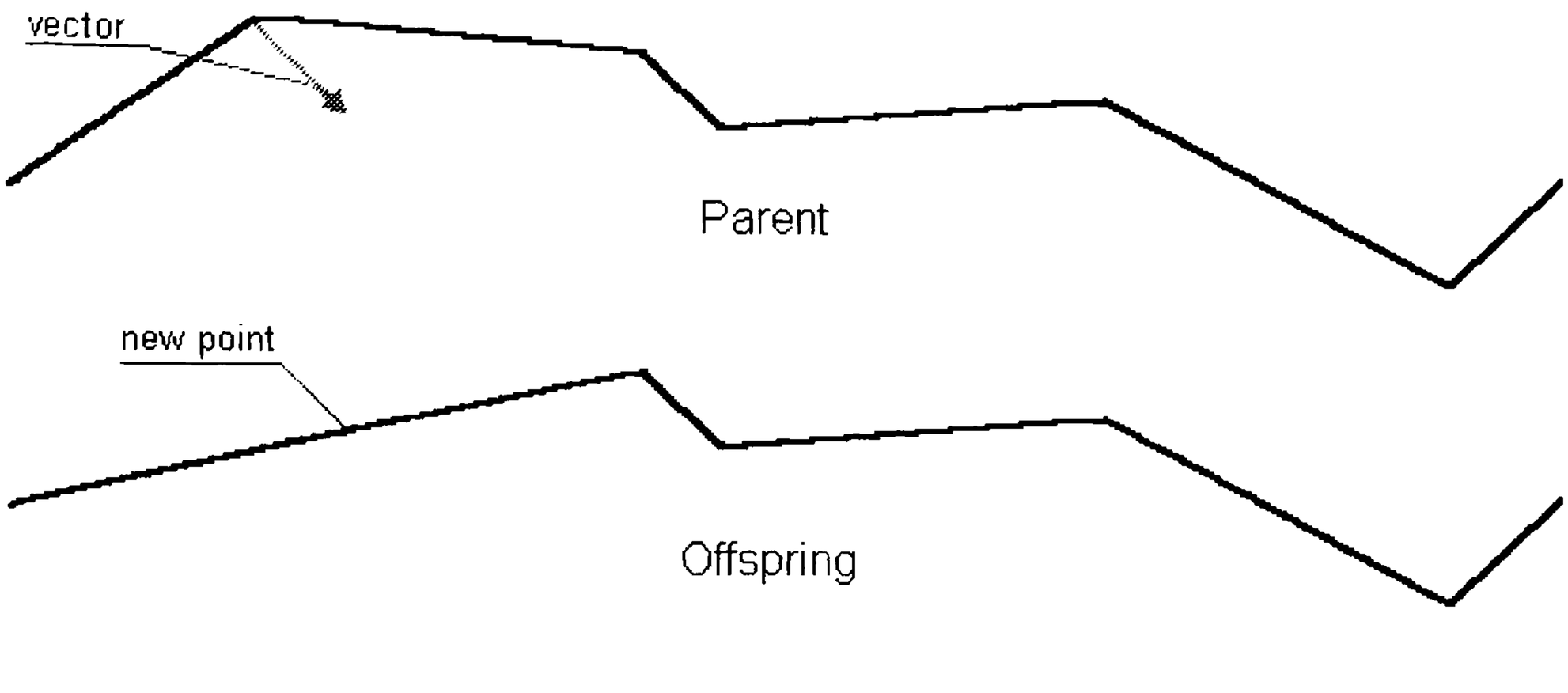


Figure 5. 4 Mutation Operation

5.3. Optimisation process and optimisation criteria

Once the chromosome structure and the genetic operations have been defined, and the initial population has been generated, the optimisation process can take place. To evaluate the fitness of every individual the variables regarding the path have to be determined, i.e. How much energy would the mobile robot consume by following that path? How much time would it take to reach the destination? How much risk is involved?

To solve for these variables, the path has to be tested using the real mobile robot, or simulated. For the case of this research, simulation is a much more convenient way to do this in order for a genetic algorithm to process the data. This means that a robot simulator is required. The relevant characteristics of the mobile robot have to be included in the simulator. The more accurately the simulation calculates the required variables, the more precise the optimisation will be. For this research, the robot simulator is used as a standard

tool, and later, additional general aspects of the simulation will be discussed, as they become relevant to the optimisation process. In the recommendations for further work, a more detailed discussion on some other aspects of the simulator will be presented.

The simulator has to include the geometric characteristics of the mobile robot, such as size and manoeuvrability. This will allow for a precise evaluation of the possibility of the robot circumnavigating the obstacles. It has also to be able to calculate the energy consumed by the mobile robot when performing a given path with detailed calculations on the speed and acceleration. These last two variables will enable it to calculate the time spent travelling and the distance covered.

The last aspect concerning the simulation process is the way in which the simulator interprets the path provided by "Rutar". For the optimisation to be useful in practice, the simulator not only has to simulate the real mobile robot as faithfully as possible, but it also has to interpret the path in the same way that the robot will interpret it. This means, that for the chromosome structure used in this system, the simulator has to interpret the path as a description of movements corresponding to the Mitchi language interpretation of the segments of the path. The simulator has to *follow* the actual path that the mobile robot would follow if presented with the corresponding segments.

If the interpretation of the path is correctly simulated, the fact that the actual path that the mobile robot will follow is a variation of the path represented by straight segments, and not its exact boundaries and corners, does not mean that the optimisation is being performed

wrongly. Since the genetic algorithm uses the values given by the simulator, and the simulator produces the values that the robot would produce, the real path followed by the robot *is* being considered. Thus if the representation corresponds to that path, the correct optimisation process *is* being performed. This means, that at the end of the optimisation process, the resulting path will be *shown* as a series of segments (that do not seem to be optimal). But the actual path that the robot will follow when presented with the Mitchi description of these segments, will be a smoothed version of it, which is the actual optimal path.

The risk assessment is very dependent on the application, and it is not necessarily related to the simulator. For example, if the risk involves a value for the probability that a path might be blocked, the risk can be represented in the map as different background colours for the delimited regions that involve a different level of risk. The total risk of the path would be the accumulated risk level involved depending on what proportion of the path passes through a given region with an associated risk level. This was the scheme used in this work. The free space on the map can have any number of regions with different levels of risk. These regions have no effect on any of the variables when executing a path or when testing its feasibility, but are considered when the fitness of an individual is calculated.

To quantify fitness, the result for each variable provided by the simulator is normalised (how much energy consumed, how much time taken to arrive, how far travelled), as well as the risk regions visited. This value is then multiplied by the percentage importance factor provided by the user for each criterion (as described in section 5.), and added together. The

quantification of each variable is performed as follows:

- **Energy:** A level of available energy is provided, which represents the amount of energy that the robot has available in its batteries. The upper bound of this amount is determined by the battery's capacity at full charge, but this does not mean that the total energy available is equal to the total capacity of the batteries, depending on the characteristics of the battery and on the power drivers of the motors, there will be a maximum amount of energy that it will be possible to use. This amount is what is used to normalise this variable. If the mobile robot requires more energy than the maximum available, the result is cut off at this value. The calculated energy consumed by the mobile robot by following the path is divided by this figure, thus energy related fitness varies from 1 to 0, where the lower is better.

- **Time:** A maximum time allowable to complete the journey is needed. The time limit used can be taken from the real maximum time permissible for the task to be completed, or if available, the time that a known sub-optimal path would take. If the path requires more time to be completed, the time value is cut off at the maximum permitted. The time taken by the robot is divided by the maximum time value to yield a time related fitness between 1 and 0, where the lower is better.

- **Risk:** The accumulated risk calculated is divided by the equivalent risk of the path as if it had always been travelling in a maximum risk region. This yields a value between 1 and 0, where 0 is best since 0 equals no risk.

- **Distance:** If distance is included as one of the optimisation criteria, the distance travelled by the mobile robot is divided by the length of the longest *reasonable* path in the map. This *reasonable* longest path is a variable that has to be produced subjectively depending on the situation in which the optimisation system is used. For example, in an optimisation process controlled by a person, the operator can decide which distance would be the longest path to consider and use a little longer as the upper boundary. If an automated system is controlling the optimisation process, this maximum permitted distance could be extracted from something more than the longest mutated path from the initial population. This value is important since if it is defined too large, the distance fitness of most individuals would be around a small value that would provide little resolution to work with (this means that the fitness value of the worst and the best individuals would be very similar). If the maximum value is chosen too small, it could result in fitness values higher than 1 so normalisation would not be guaranteed. Experience has shown that taking this value as twice the distance of the longest path generated by "Rutar" for each given map is acceptable. Having few exceptions, if the fitness of an individual results in a value higher than 1, it is cut off at 1. Thus, distance fitness also yields a value between 1 and 0, where the lower the better.

After all the individual fitness values are calculated, each one is multiplied by its percentage of relative importance as determined by the user. These percentage values, which determine the relative importance of each criterion in the optimisation, were discussed at the beginning of this chapter. After this all values are added together to get a

consolidated individual fitness. This final fitness will be between 0 and 1. with the lower being better.

The complete optimisation program was arranged as shown in figure 5.5.

- The input module collects the parameters that the user needs in order to define the optimisation process and for the mobile robot, such as filenames to use, percentage importance factors of each criterion, maximum energy available, initial and final orientations of the robot, etc. This module serves as the user interface between the optimisation module and the operator. Appendix C shows a screen dump of the input module and the list and details of each parameter, it also shows the inputs and outputs of the module.

- The simulator module calculates all the variables needed by the genetic algorithm to calculate the fitness for each individual (path) of the population. The simulator module contains the fixed mechanical and dynamic characteristics of the particular mobile robot that is going to be used. The simulator receives a path from the optimisation module, simulates it, and returns to the optimisation module the values of energy, travel time, risk and distance that the mobile robot would yield if it was to follow that path.

- The Optimisation module is the actual genetic algorithm. It calculates the fitness from the simulator's result, performs the genetic operations each generation, and assesses the termination criteria. The termination criteria for this genetic algorithm, as used by most

authors, consists of a time (CPU working time) limit and a fitness achievement. The time limit is given as a maximum number of generations that are going to be executed; the user can specify any number when running the algorithm. The fitness achievement is given as a percentage of the maximum possible global fitness; when executing the algorithm the user can state how close (in terms of a percentage) to the maximum total fitness is considered to be an acceptable or final answer. Whichever criterion is reached first, the genetic algorithm's execution terminates.

- The parameters module has the data to determine the way in which the genetic algorithm is going to use its operators, fitness data, and termination criteria. The percentage of elitism, replication, crossover and mutation used can be determined by the user each time an optimisation process is started. It consists of a file where the user defines this data before each execution of an optimisation process. The data contained also includes the termination criteria. Appendix C shows the structure of the file.

- The output module interfaces with the user to present the answer and useful data for further analysis. Appendix C shows an example of a typical screen dump of an optimisation process output.

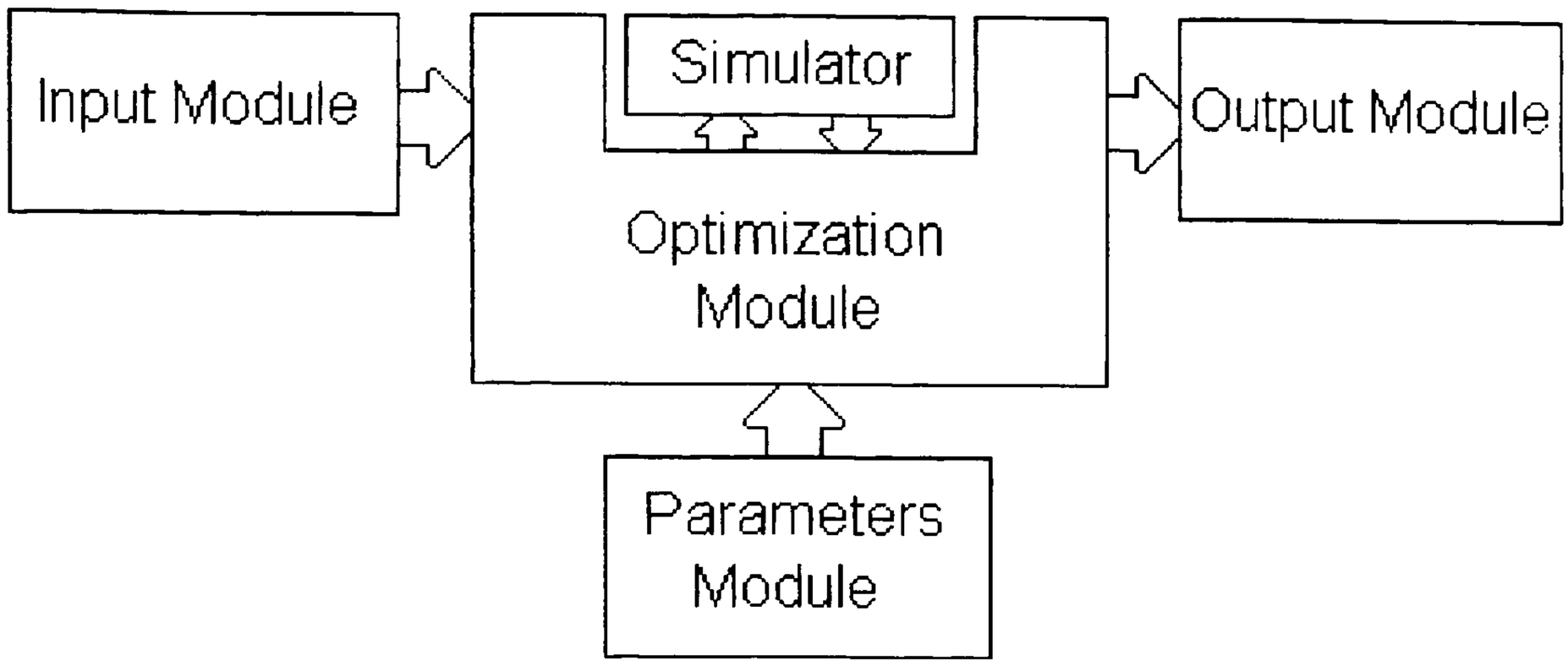


Figure 5. 5 System's Structure

The program is written in the ANSI C language, the same as all the programs presented so far. The simulator is a <simulate.h> type include file, which can be changed to simulate the desired robot. It must execute as a function of the optimisation module, which provides the path description, in Mitchi, as input, and gets the energy, time, risk and distance as outputs. In this way it is a very standard way to include any simulator for any robot.

After the optimisation is finished, there are as many answers (optimal individuals) as initial non-redundant paths, since each one of them has produced a self contained population where an optimisation process was performed, as explained in section 5.1. The optimal paths from each original non-redundant path population are compared, and the global optimal individual is chosen by selecting the path that has the minimal best performance as defined by the relative importance of criteria selected by the user.

5.4. Experimental Results

The optimisation program was compiled to run on two different machines: a DEC Alpha 3500 Unix based workstation and a DOS based 33 MHz 486PC. These two computers were used to evaluate the execution speed and performance on different platforms.

A typical optimisation process would be performed on the paths generated by "Rutar" such as the map shown in figure 5.6. For "Rutar" to generate the paths there is no difference between the origin and the destination, so it is necessary to define them at this stage for the optimisation genetic algorithm. For this example, the point on the left is the origin and the point on the right is the destination. This is defined in the input module as shown in Appendix C.

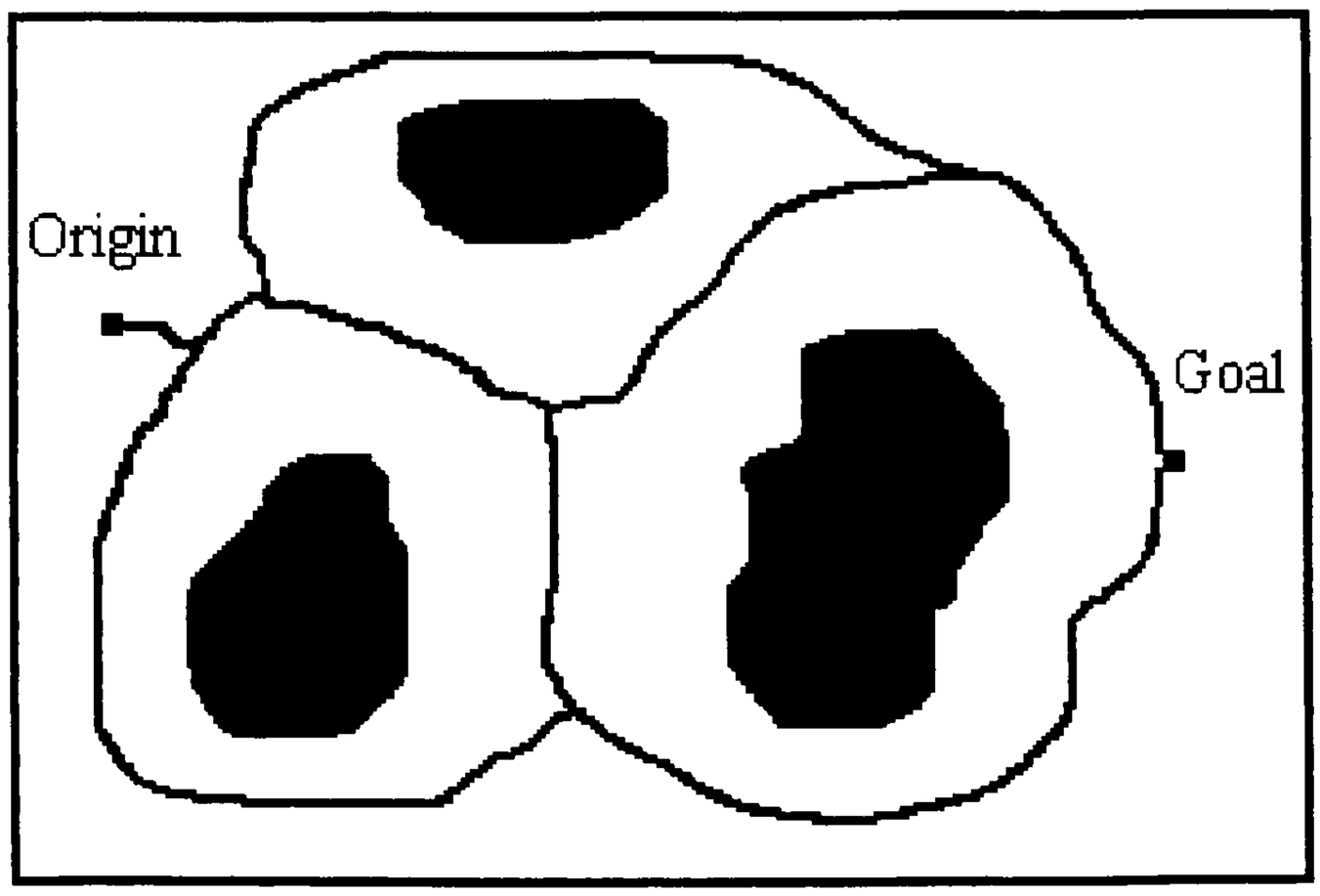


Figure 5. 6 Rutar's Output

One of the parameters of the optimisation system is the number of paths to analyse, since to save CPU time it may be convenient to consider only some of the best paths rather than all the possibilities. The number of non-redundant paths to consider in the optimisation can be any number between one and the maximum. If computing time is not a constraint, then all possible paths can be considered, if computing time needs to be minimised, a reduced number of the initial paths can be considered. For the case of the map in figure 5.6 it will be assumed that only two of the paths are going to be included in the optimisation. By doing this we do not mean that the other paths (for instance the S shaped paths) are unworthy of optimisation as there could be an S shaped optimal path depending on the tightness of the S and on the initial and final orientation of the robot. This only shows that not all paths have to be optimised. Choosing which paths to use and which to ignore can be an arbitrary choice of the user, or could be done by testing the paths and selecting a number of the best paths (best at this stage), or just use them all. Although in some cases there could be the risk of losing the possibility of finding the global optimum by not optimising all paths, it is a choice that the program provides. In an automatic system it may be convenient to use all of them, whereas in a manual system, an experienced user could choose not to optimise a path that he or she might consider to be obviously inefficient. Figure 5.7 shows in bold the two chosen paths to optimise for this case.

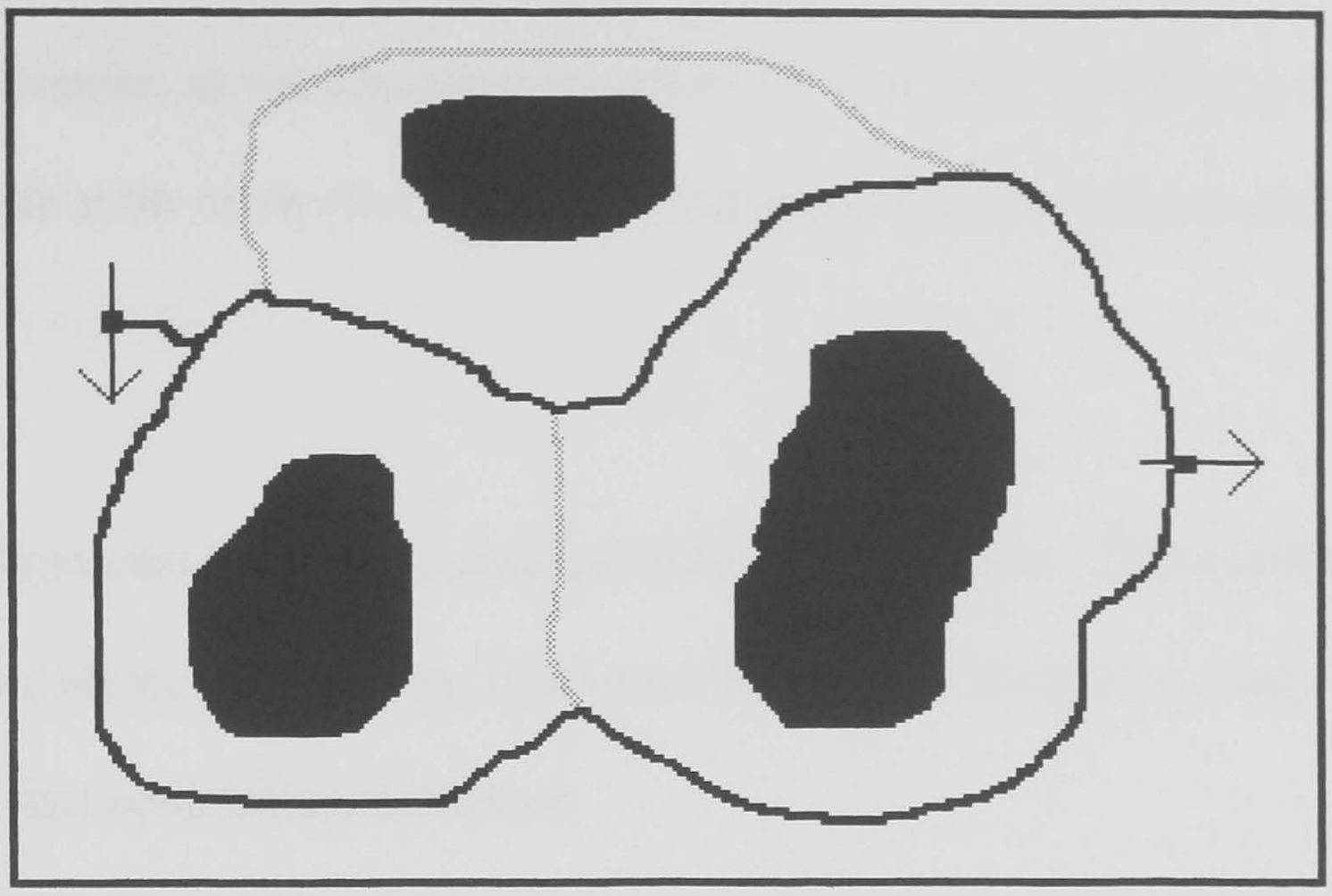


Figure 5. 7 Two chosen paths to optimise

For the purpose of test and evaluation, a map of 256 by 256 pixels was used, and a simulator for a differentially steered type of mobile robot was designed. A simulator was implemented for the purpose of testing the optimisation process [Gongora MA, 1995b] [Gongora MA, 1996a], therefore it was not simulating any real robot, it simulated a hypothetical robot that had general characteristics for which it was possible to perform detailed calculation for all the robot's aspects as discussed in section 5.3. Appendix D shows the characteristics of the theoretical mobile robot used, with the constraints that real robots would have in practice.

Different optimisation criteria were defined for testing the optimisation processes. Tests were made with 100% importance for Energy minimisation, 100% for travel Time minimisation, and 100% for Distance. Later, a test was made with 50% for Energy and

50% to Travel time, this was chosen arbitrarily to test the two most important and difficult criteria to optimise, as was discussed in section 1.2. The mobile robot was assumed to be pointing south at the origin (the left point) and pointing east at the destination (the point to the right).

Figure 5.8 shows the answer for minimum distance optimisation. This result is as expected for a 100% distance optimisation. It is very interesting to compare it with the results for energy and time optimisation as follows.

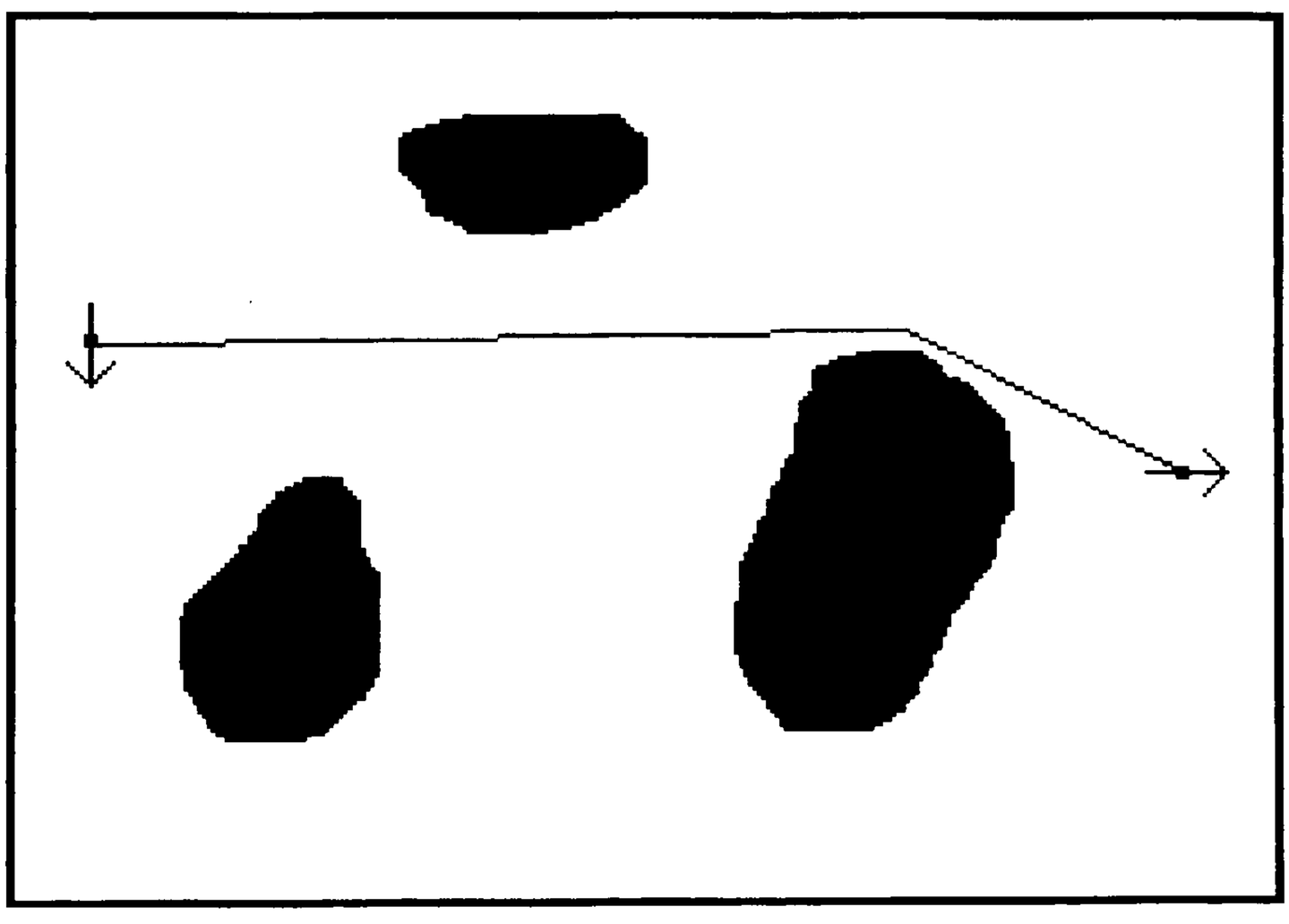


Figure 5. 8 100% distance optimisation result

Figure 5.9 shows the answer for 100% Energy optimisation. It is very different from the distance optimisation, since it resulted from the other one of the two non-redundant paths being considered. Comparing these two results it is clear that the spin turns at the

beginning and end of the paths use up more energy than the extra distance travelled. Moreover, it seems that according to the simulator, the spin at the start expends more energy that the one at the end, since the minimum energy path started straight ahead. It is also possible that the minimum distance path saved most of the energy at the start since at the end it was very difficult anyway (since no matter from which side the robot approached, it had to make a small radius turn because there is very little room for a straight line approach at the correct orientation).

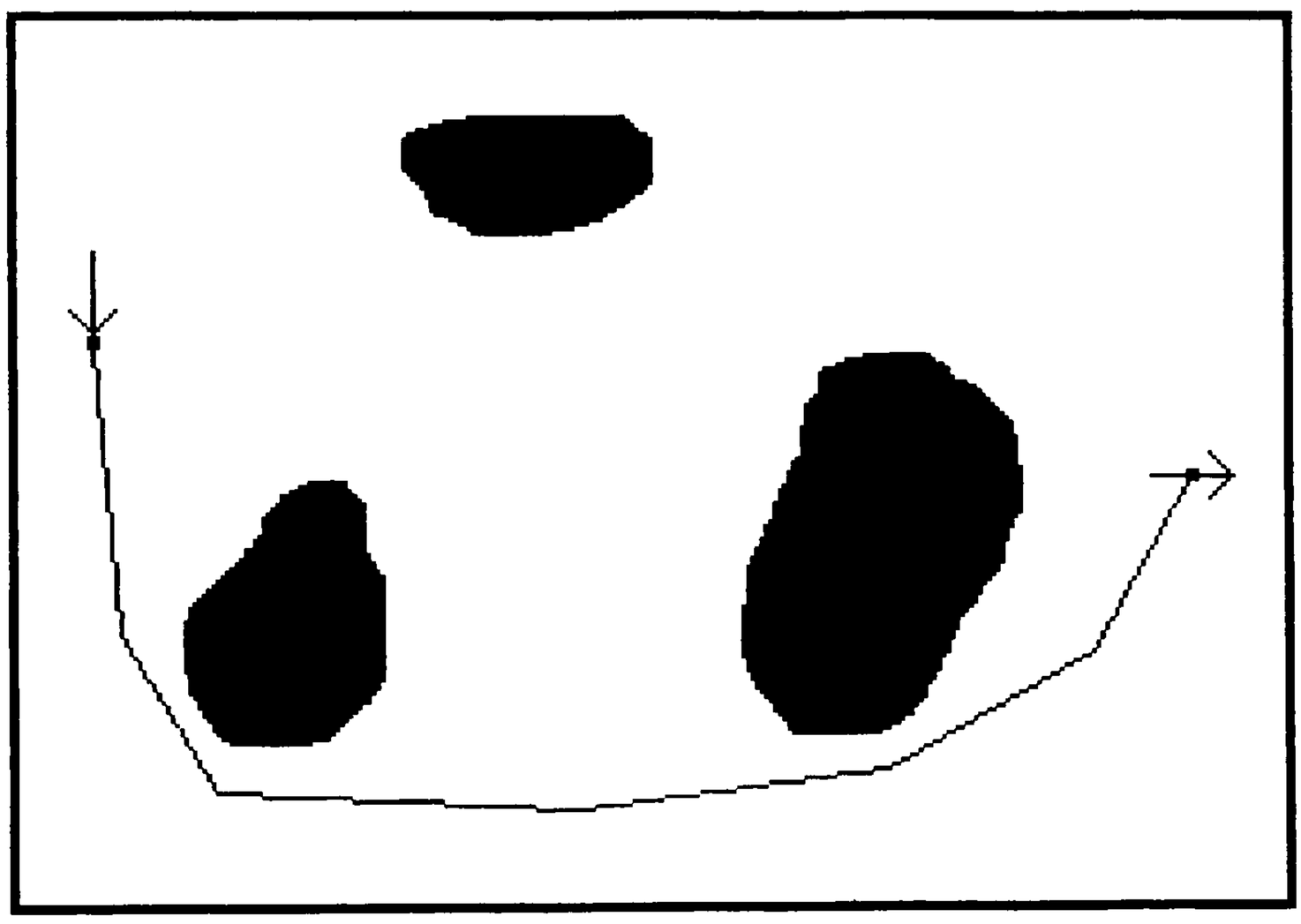


Figure 5. 9 100% Energy optimisation result

Figure 5.10 shows the answer for 100% time optimisation. This time, the path resulted from the same non-redundant alternative that produced the minimum distance, but is has a significant difference in terms of distance; although the robot travelled an extra distance downward, this saved it time by permitting it to start to move faster and keep at a higher

average speed throughout the path.

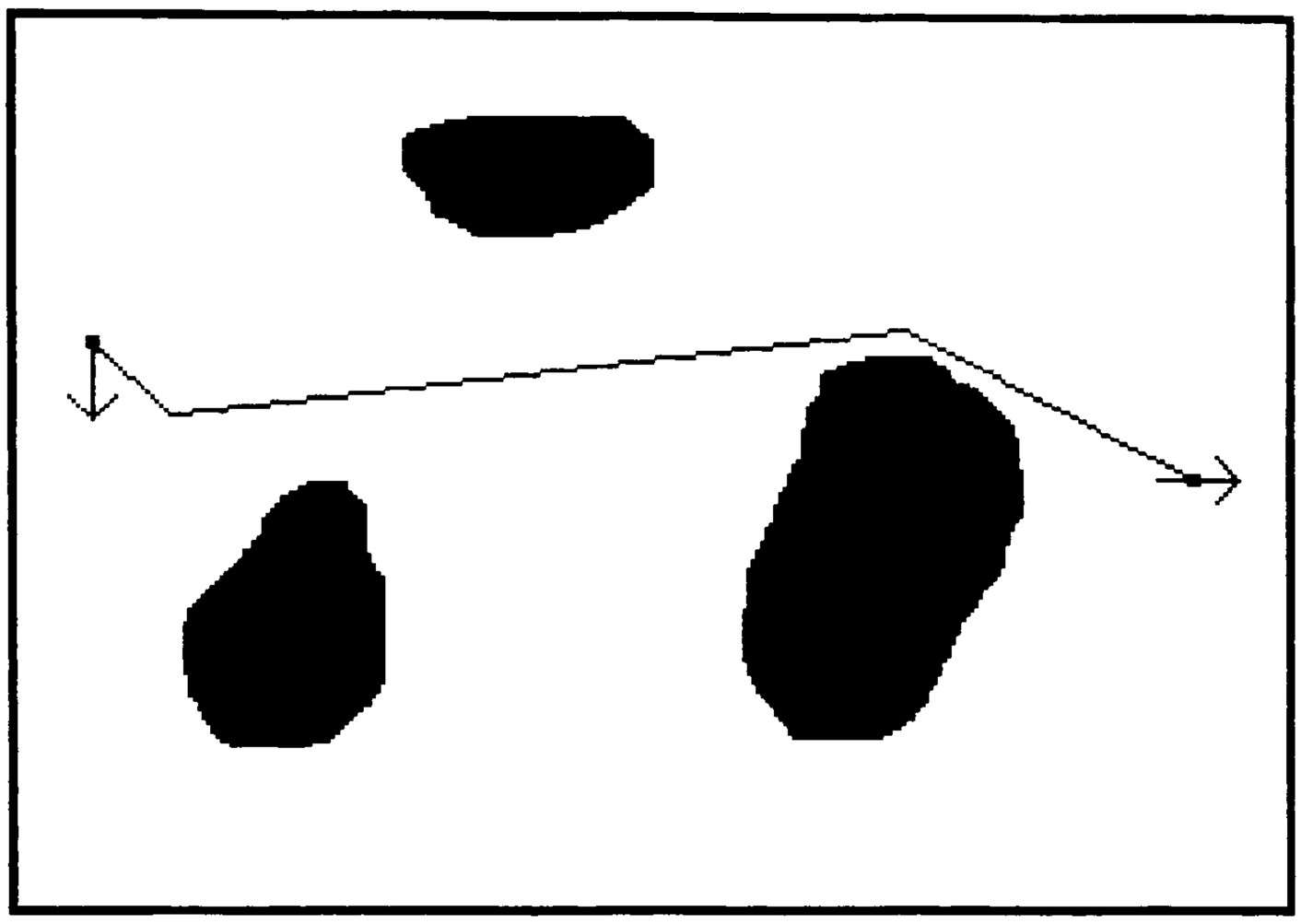


Figure 5. 10 100% time optimisation result

Figure 5.11 shows the answer for 50% Energy and 50% Time optimisation. This answer is very similar to the energy only optimisation. It was the case that the time for the time only optimisation was very similar to the time for the energy only optimisation, therefore, including even a small percentage of relative importance to energy, made the system find the optimal path using the lower of the two paths as it could be made smoother.

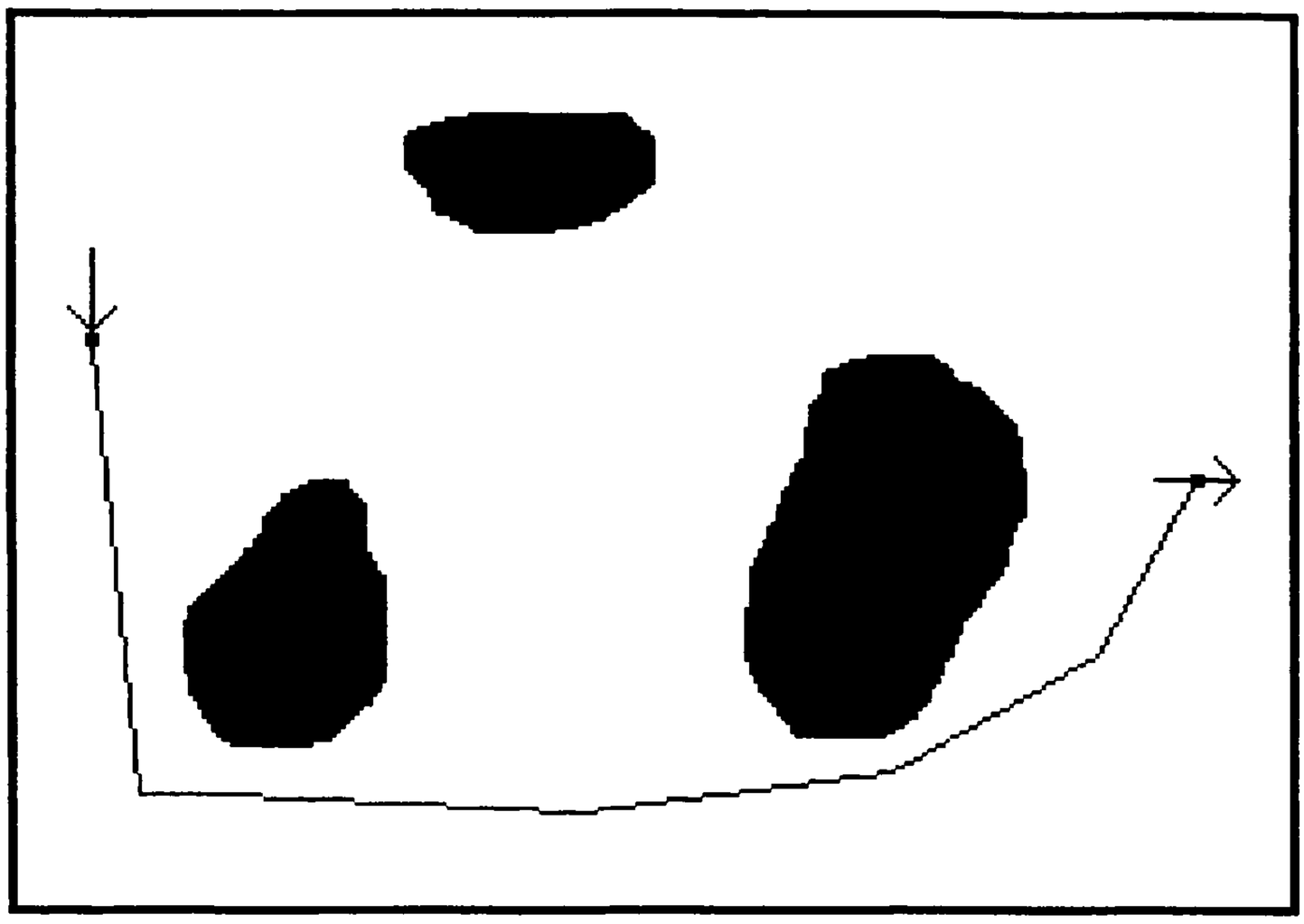


Figure 5. 11 50% Energy and 50% time optimisation result

Figure 5.12 shows the answer for a 100% risk optimisation. Note the background colours, which denote the risk areas, the darker the colour the higher the risk of the area, black is maximum risk (obstacle) and white is no risk (free space).

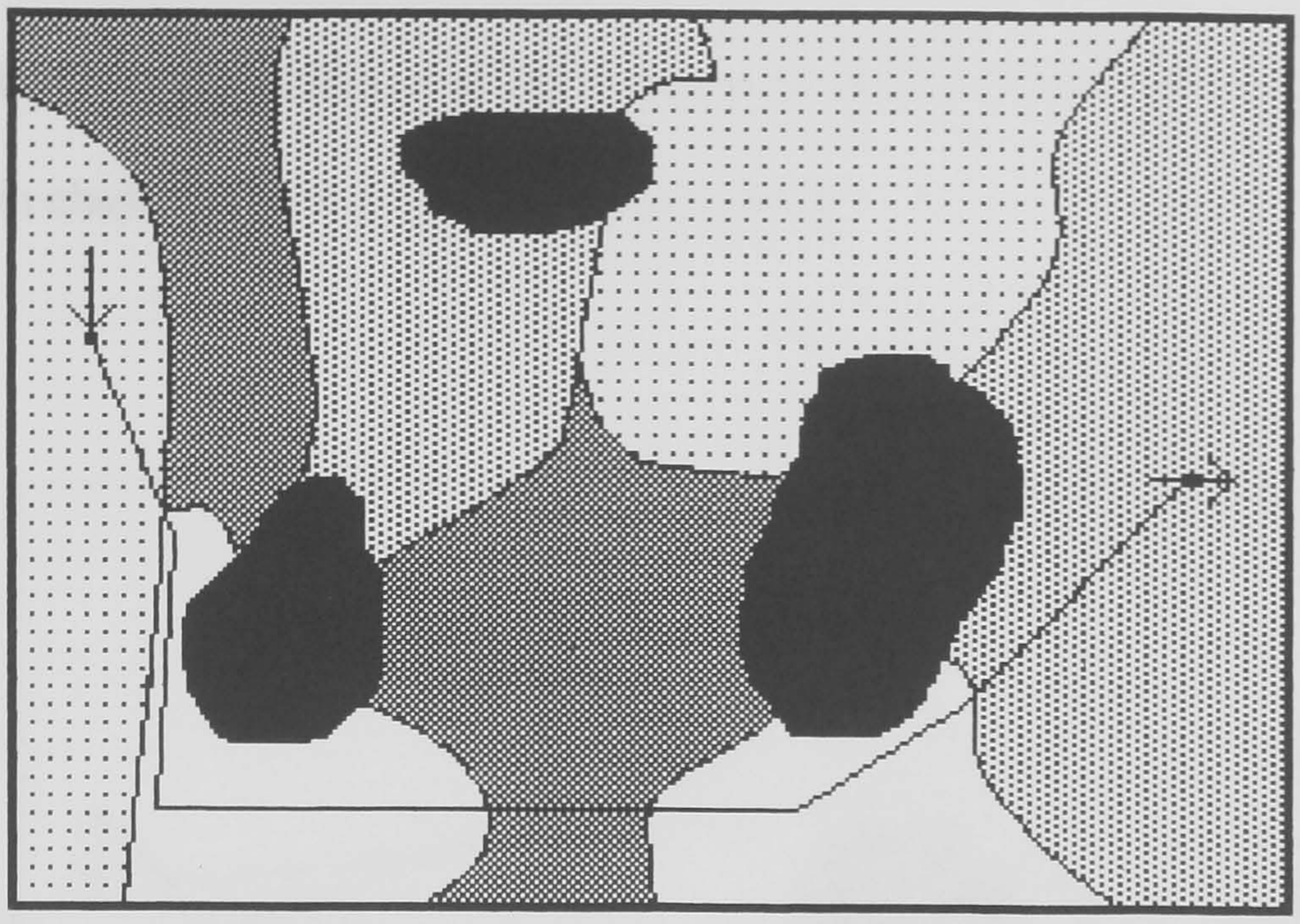


Figure 5. 12 100% risk optimisation result

Appendix E shows results for various combinations of criteria and orientations of the mobile robot, as described below each picture.

To prove that the paths found by the optimisation system were the optimal, many runs with the same problem and criteria were executed, and the standard deviation of the results was calculated. To illustrate this, figure 5.13 shows an exaggerated example of the difference between two results for the case of time optimisation. These two results have the same form but differ in the magnitude of their segments. In a perfect system that could find the absolute optimal, every execution of the process should ideally produce exactly the same answer. Calculating the standard deviation of the results around the mean result from a series of executions, gives an idea of the quality of the process and a measure of confidence that the results found are really an approximation of the global optimum. To

prove mathematically, in a completely formal way, that the results are the global optimal is too difficult (if not impossible) and out of the scope of this work. Later in the analysis of the results, in chapter 6, this will be discussed in detail.

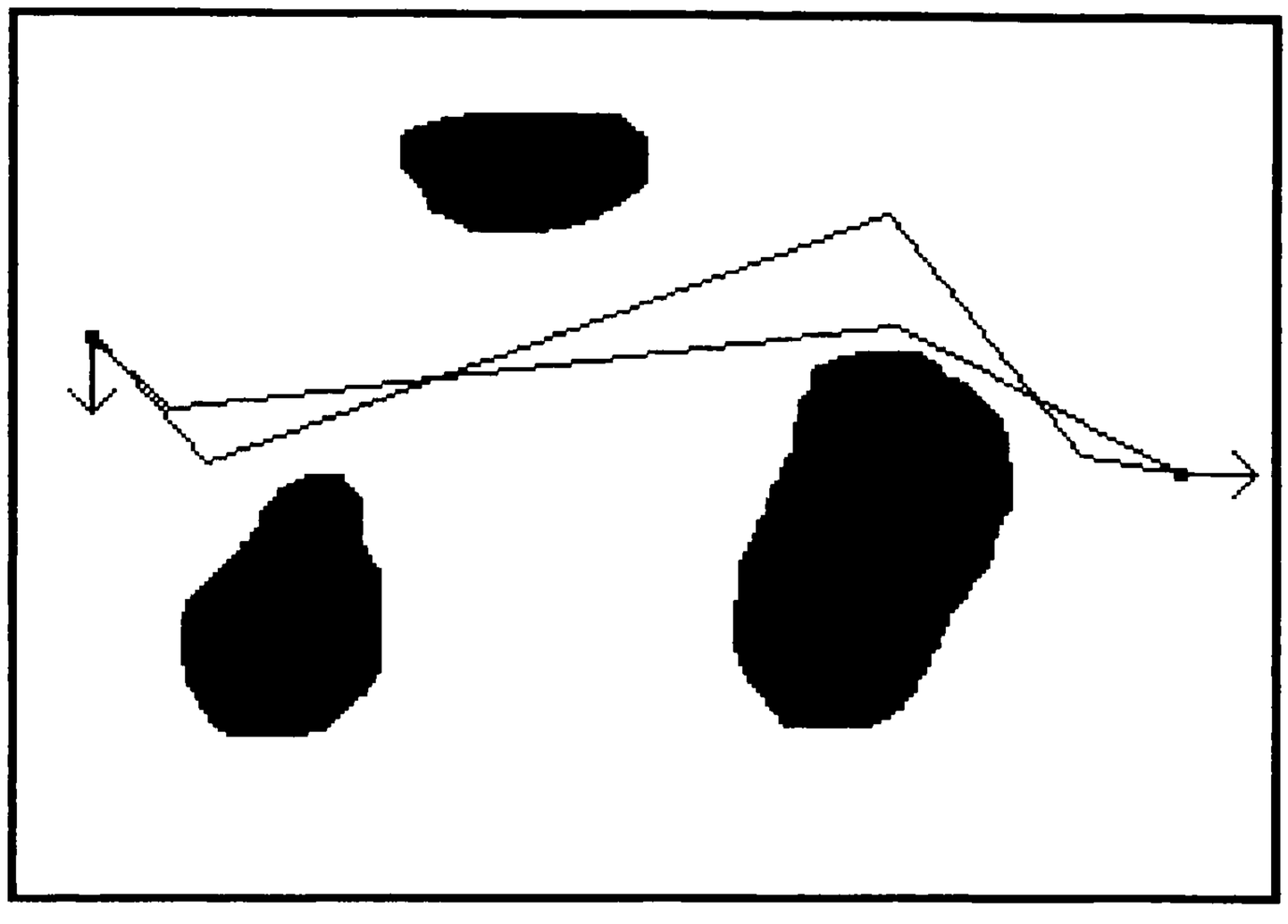


Figure 5. 13 Different results for the same problem

One interesting feature of many of the optimised paths found, was that for the time and energy criteria, the system frequently found the optimum path that was very similar to one of Rutar's alternatives. This may be due to the smoothness of Rutar's answers, and the fact that it does not minimise distance in the path generation stage. This feature will be discussed later in more detail in section 6.3.

Although the path *shown* in figures 5.8 to 5.12 is composed of straight line segments, it is important to remember, as explained in sections 5.2. and 5.3, that depending on the way in

which the robot executes the paths described in the Mitchi language using straight line segments, the "actual followed" path is going to be smoothed in most corners, and that the real execution of the path by the robot (the simulator in this case, which is supposed to be faithfully simulating the robot) is the one that is considered by the genetic algorithm and considered optimal. Therefore, these figures show only the straight-line segment description of the optimal path which when performed by the robot, results in a smooth path, if that is the case.

In the following chapter, an analysis of the optimisation process described here will be presented. These and additional results will be verified, including the genetic algorithm aspects that can be highlighted from these experiments.

Chapter 6

ANALYSIS OF RESULTS IN PATH OPTIMISATION

The genetic algorithm module in path optimisation executed efficiently (50 seconds) even on low performance machines, as compared with the genetic algorithm used in the development stage of "Rutar". The main reason for this is because of the type of problem that is being solved. With this low execution time, this optimisation system is useful in off-line real-time path planning. It can be used if a waiting time of about one minute is accepted at the beginning of a process. And if the execution of a path (the robot following it) takes longer than a minute, the next stage can be planned in the mean time and thus

executed immediately after the completion of it.

In this chapter, the performance of the genetic algorithm used for path optimisation is discussed.

In section 6.1 a discussion on the verification of the results is presented. Section 6.2 contains the analysis from the point of view of the optimisation criteria used.

The type of problem which is being solved makes an important difference to the way the answer is sought and this, in turn, makes an important difference to the computing time needed. In this thesis two very different problems were solved with Genetic Algorithms: in the first problem the genetic algorithm was used to find the rules of death for "Rutar", in the second problem it was used for path optimisation. The main differences that have the greatest influence on the computing time are explained in section 6.3.

Section 6.4 contains the analysis of the results taking into account the paths provided by Rutar, as a starting point for the optimisation. Finally, section 6.5 presents the analysis from the point of view of the fitness of the individuals, as a function of the combination of the optimisation criteria.

6.1. Verification of results and the computers performance

In this section it will be shown that the results provided by different computers, used for optimisations, show that the method is very efficient, but the availability of more computing resources (especially memory) dramatically increases the potential of the genetic algorithms.

The main difference between the execution of the optimisation genetic algorithm in the Unix and the MS-DOS system is the availability of memory and mathematical resolution. For this reason, in the Unix system a population of 200 individuals per generation was used, while in the MS-DOS based system only 20 individuals were used due to memory limitations. The mathematical resolution in the Unix system was higher (80 bits floating point versus 32 bits real numbers in the MS-DOS system).

For a typical execution, comparing the results obtained in the Unix system versus the PC (MS-DOS) system, the following was observed:

- When using any non-exhaustive search method (including Genetic Algorithms), there is a possibility of falling into a local minimum as explained in section 2.2. Due to the limited capacity of a personal computer under MS-DOS, the results indicate that the search is more likely to fall into a local minimum than in a Unix system. This is due to the fact that the number of individuals used for each generation of the genetic algorithm is limited to a lower quantity in the PC than in Unix systems. This higher

probability of falling into a local minimum can be seen in the following analysis of the results of many executions.

- Performing many optimisation processes with the same problem, and parameters, in a given system (computer) results in slightly different answers in each execution. This happens because of the random component of the genetic algorithms. This can also be seen in the following analysis of the results, where 20 executions of exactly the same problem were performed with each system.

As mentioned in section 5.4, one way to verify that the results produced by the optimisation system are *really optimal*, is to measure the standard deviation of the answers for many executions of the same problem. In an ideal system, that could find deterministically the absolute optimal result, and without mathematical approximations, every execution of the same problem should produce exactly the same answer. In this system, that has to use mathematical approximations (due to the limits in digital computers) as well as a probabilistic component introduced by the genetic algorithms, every execution yields a slightly different answer. By measuring the standard deviation of the answers, an idea of the reliability and repeatability of the system can be obtained.

To perform this analysis, 20 executions of exactly the same problem were done in each system (the MS-DOS based computer and the Unix based computer). The 100% time optimisation problem shown in figure 5.10 was repeated 20 times. Figure 6.1 and 6.2 show 4 overlapped (randomly selected out of the total 20) answers for the PC and Unix systems,

respectively. Showing more overlapped answers makes no sense since it loses clarity and only a single blotch can be seen, for the Unix system this is even less clear since the answers are more similar to each other.

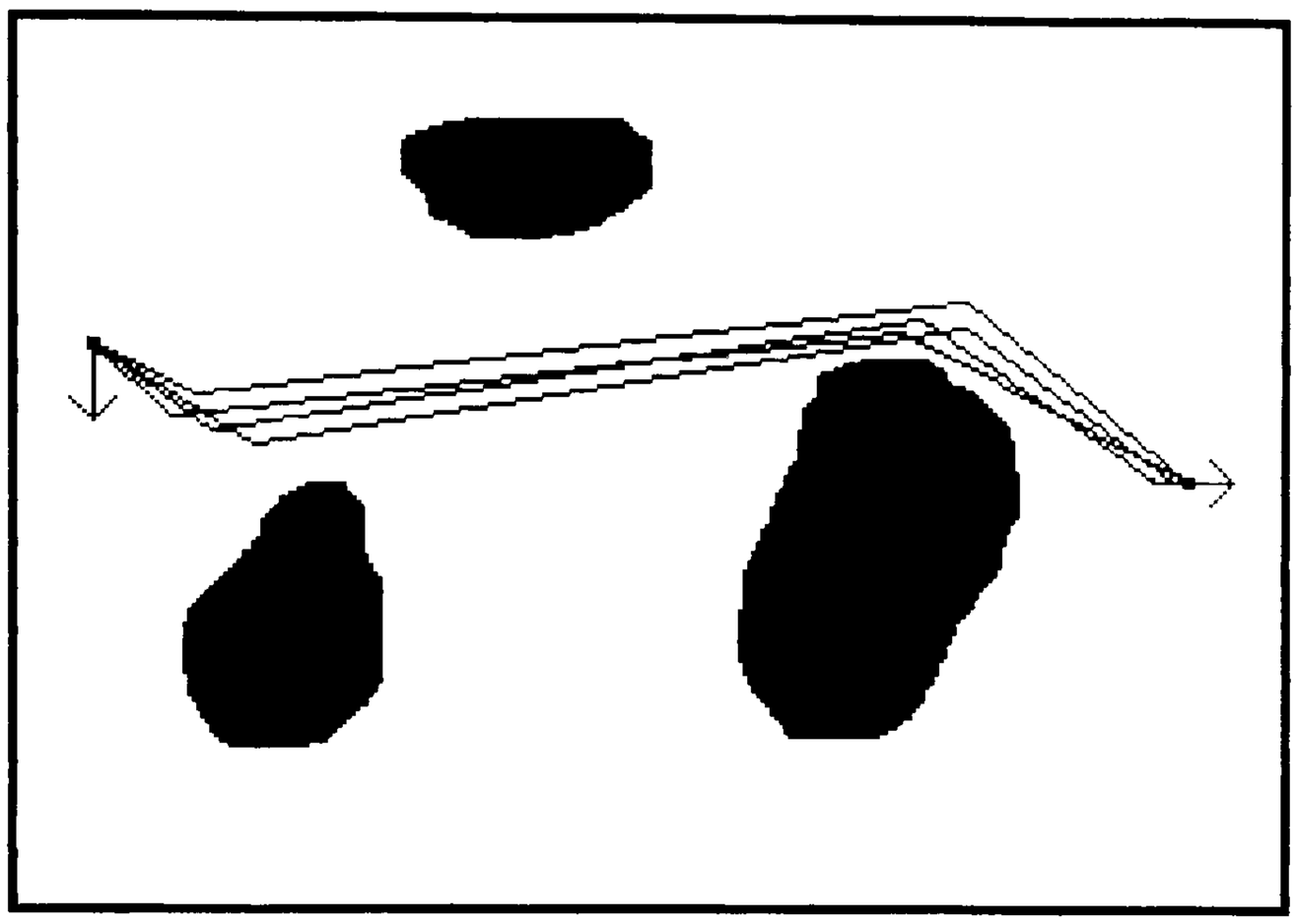


Figure 6. 1 Different answers for different executions (MS-DOS)

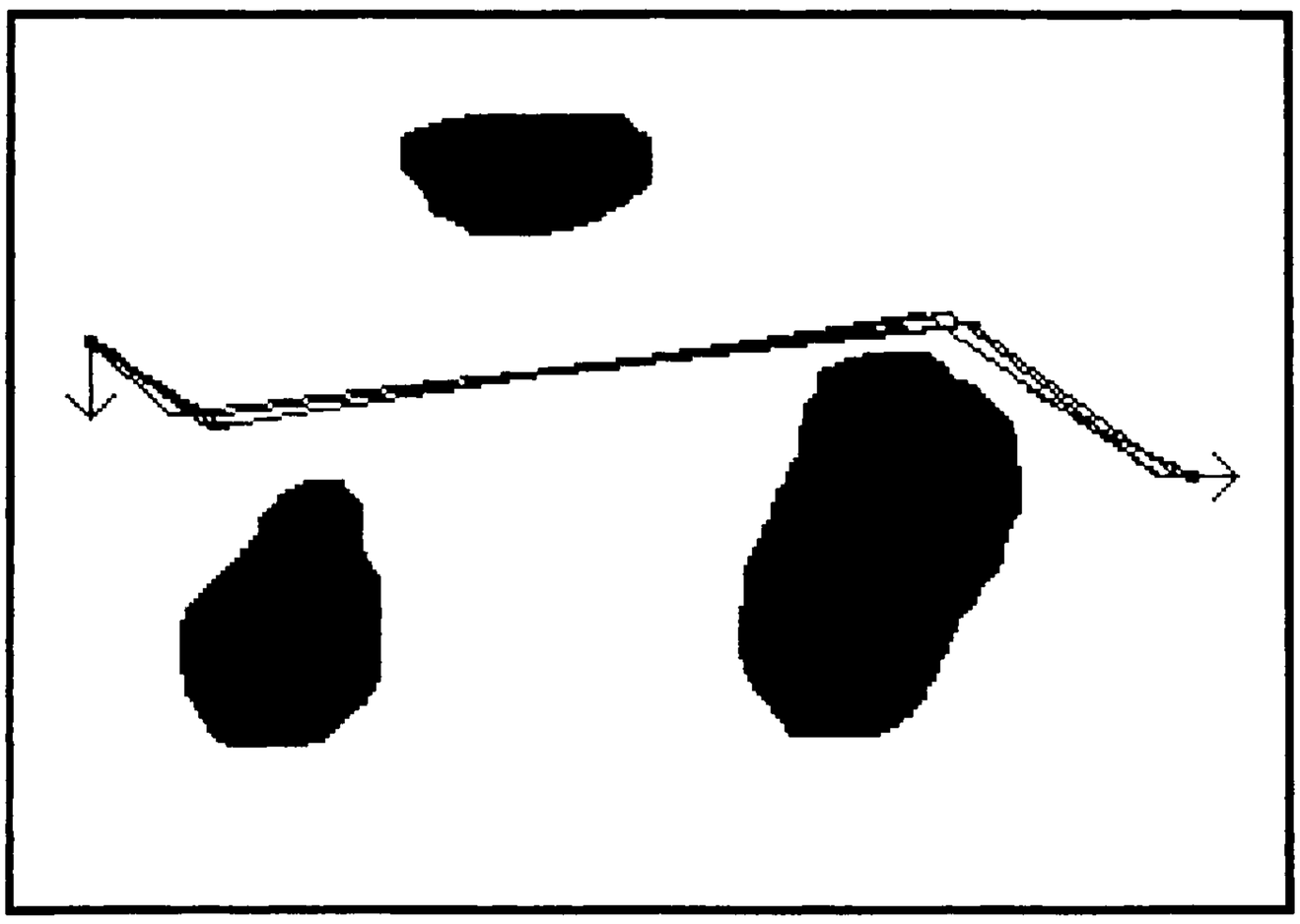


Figure 6. 2 Different answers for different executions (Unix)

Seen in a graphical form, the results for all the runs show that they present a slight deviation from the mean value. As shown in figure 6.3, the Unix system presents a much smaller deviation.

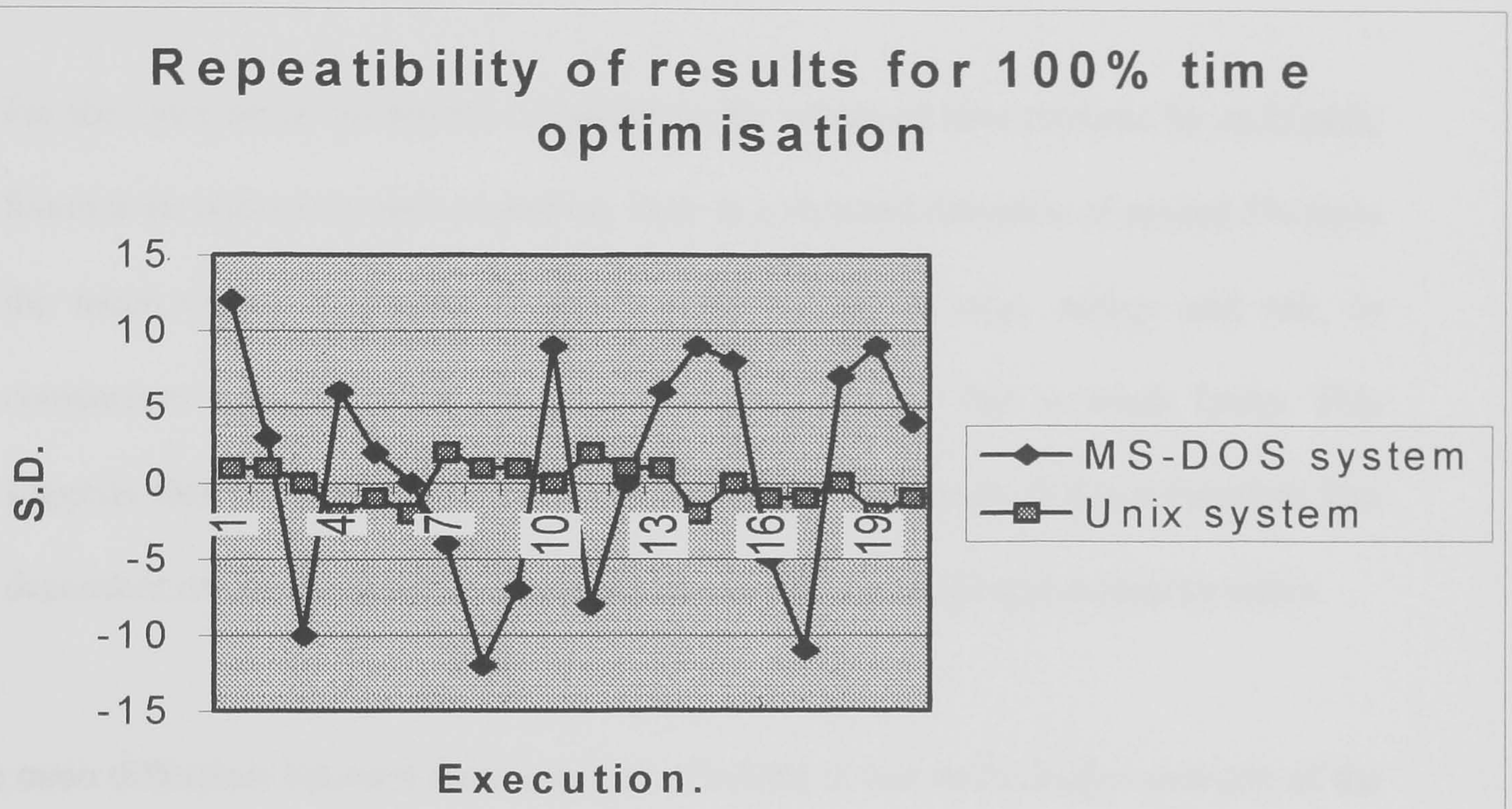


Figure 6. 3 Plot of different results for different executions

These results show that:

- For the MS-DOS based computer: comparing the values of time returned by each path that was found to be optimal in each execution, there is a standard deviation of approximately 12%. In Appendix D more examples of these tests are presented, and the standard deviation for the results of many tests performed using time, energy and

risk optimisation produced by the PCs, lies between 10% and 32% from the mean values. One reason for this variation is the difference in the initial random population, and another reason is that it may find a local minimum that is very near to the global minimum and it takes that path as the answer.

- For the Unix based workstation: comparing the values of time returned by each path, found to be optimal in each execution, there is a standard deviation of around 2% from the mean values. Appendix D shows more results for time, energy and risk. In comparison with the result variation of the PC system, this is much lower. This suggests that the answers are nearer to the global minimum, and are therefore less dependent on the initial random population and mathematical approximation errors.

The main difference between these two sets of results is due to the higher capacity of the operating system to handle memory, and thus use more individuals per generation. In the Unix system the memory capacity is virtually limited to the capacity of all the data storage systems in the machine such as actual RAM, local hard disk, and networked hard disks. In the MS-DOS based systems, the program was limited to use a *page* of memory from the actual RAM. In modern versions of personal computers where the memory limitations have been reduced, such as Windows 95 based compilers capable of handling the complete memory without 64K page limitations and handling virtual memory, both systems have subsequently been shown to perform similarly with standard deviations of approximately 2%. The last set of results presented in Appendix D, from executions in a Windows 95 based system, show this. This confirms the importance of memory availability in the

performance of genetic algorithms, which reflects as genetic variety, as explained in section 2.2. Figure 6.4 shows the performance of the system, in terms of S.D. of the results, according to the number of individuals used in the population. It is important to clarify that the behaviour of the S.D. of the results, as shown in figure 6.4, depends only on the number of individuals. It is independent of the system in which is executed (therefore 200 individuals will yield the same behaviour in a Unix system as in a Windows 95 system).

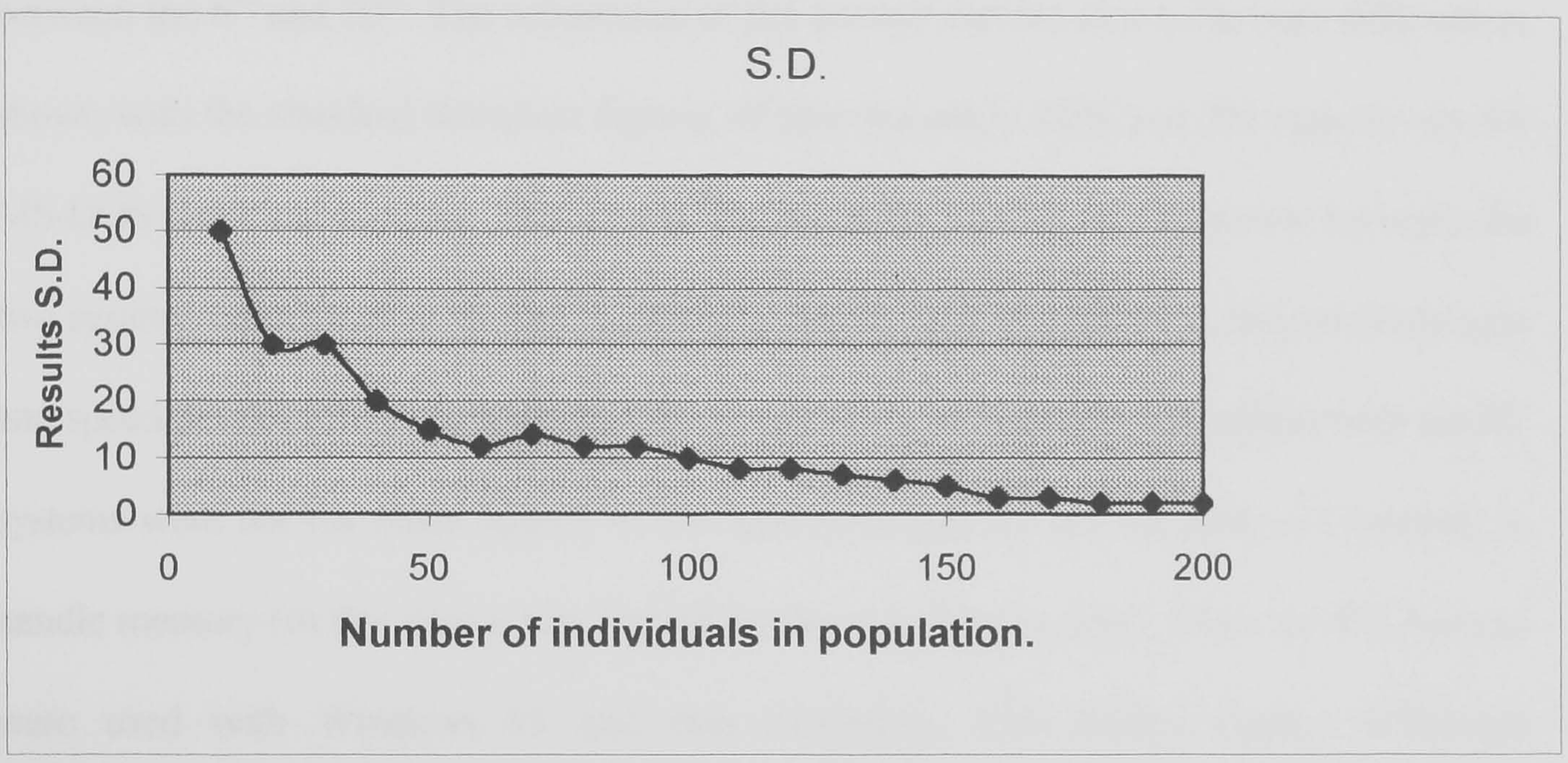


Figure 6. 4 Repeatability of the results according to the number of individuals in the population

The other factors of the computational aspects apart from memory and mathematical precision, such as speed, type of processors, software and compilers, etc., only significantly affects the speed at which the program runs, while they have no noticeable effect on the quality of the answer.

These differences show that the most important aspect in genetic algorithms, that prevent them from falling into local minima, is the random component, which in turn is greatly affected by the population size in the form of genetic variety. In the initial tests of this system, that were performed on 386 based machines for the MS-DOS version, and on DEC Alpha stations for the Unix based machines, the number of individuals were respectively 20 and 200 in each generation. For both systems the answer was found in any generation between the 6^{th} and 20^{th}. The robustness of the answer can be seen to be very different as shown with the standard deviation figures, of approximately 20% and 2% respectively for MS-DOS and Unix systems. Thus, the difference in the quality of the answer between the two systems relied mainly on their capacity to handle large populations; the processor type and speed did not affect the quality of the results. This means that the problem with the PC systems were not the clock speeds or the type of processor, but the system's capacity to handle memory (in this case it was limited by the operating system). Once the PC systems were used with Windows 95 and new compilers, this answer quality difference disappeared.

The main limitation imposed by a small number of individuals, low genetic variety, can be overcome by using a higher percentage of mutation than in a system with a larger population. But this can be done only up to a limit since mutation can also damage the few good individuals. Experience, as reported by many researchers [Davis, L.] [Koza, J] and as observed in other work by the author, says that mutation rates in excess of 10% is too high and lowers the "guided evolution" rate of genetic algorithms. Too much mutation increases

the raw randomness of the process, and this effect is more important as the number of individuals in the population decreases, therefore a problem is always present with small populations.

Execution time.

The execution times of the optimisation process were considered to be very low, even on the 33MHZ, 386 based PCs. An average execution would last about 1 minute on a PC, and less than 50 seconds on the Unix workstations. Of course we have to take into account the fact that the Unix system had to solve the problem with 10 times more individuals than the PC. In present day computers, such as a Pentium at 233MHz, the optimisation process takes less than 1 minute with the 200 individuals population.

As mentioned at the beginning of this chapter, this execution time of around a minute can be considered low for some applications. Although it may seem that Rutar's speed is under-utilised when the optimisation process takes so long compared to it, two reasons can be seen to contradict this:

- First, the lower the time for every part of the optimisation process, the more it helps in the overall speed.

- Second, in the recommendations for further work (in chapter 8) it will be shown that

Rutar's answers can be used for some very fast on-line path planning applications.

Thus, the results from the optimisation process, and the low execution times, show that the use of direct kinematics is a very efficient way of solving problems using a suitable search method such as genetic algorithms. And since for the solution of the direct kinematics an unambiguous answer always exists, and the results are measurable for any given robot, it is indeed much simpler to implement this problem in a computer, than the solution for the inverse kinematics. Conversely, inverse kinematics methods may prove very difficult and need particular implementations for every criterion that would need to be optimised.

This also applies to the modularity of the system. As will be seen in the chapter on recommendations for further work, this system is very versatile and to use it with other mobile robots only the simulator needs to be changed. Since the simulator is a completely independent and self-contained module, the simulator for the robot in which the system is going to be applied can be replaced efficiently.

6.2. Optimisation criteria results

Several points can be made about the type of results obtained from the optimisation process, depending on the criteria used (refer to the results shown in figures 5.8 to 5.11 and Appendix D). For example:

- For distance only optimisation this algorithm is clearly overworked since the distance optimisation criterion is fairly independent of the kinematics and dynamics of the mobile robot. Thus, calculating its direct kinematics and dynamics via the simulator is not efficient since only the geometry (size) of the robot and its steering capabilities need to be known. These two characteristics are used only to confirm if the path is feasible or not. Therefore, for distance only optimisation a more direct vectorial method could be used for an even faster result. Still, since the genetic algorithm executed in less than a minute, and this process is useful for other purposes, this algorithm can be used for distance optimisation, having the advantage that distance can be combined with the other criteria that can be solved with this algorithm (time, energy and risk).

- In energy and time optimisation this method is considered to be truly innovative. The optimisation for these criteria is achieved explicitly with no need for approximations. And it is not only time and energy that can be optimised independently, but they can also be optimised together in any proportions. The optimisation of these two criteria combined as a composite criterion can be performed without it being any more complicated than optimising only one of them.

- Although risk optimisation is also independent of the characteristics of the robot, this method is very efficient. Risk optimisation needs very little knowledge about the robot. The only information needed is the size of the robot to calculate where it is *stepping on* at any given moment (i.e. over which areas of risk is it travelling). Nevertheless, the

process involves looking for a combination of paths across a series of areas of risk that would minimise the average overall risk. Thus, this problem is not easy to solve by means of other mathematical methods while it is very suitable to a search method such as a genetic algorithm.

- The risk optimisation capability can be used for other purposes as well. Since the time and energy efficient paths tend to be wide curves away from obstacles, this may pose some problems in applications where the mobile robots are operating in an obstacle free space but it is still desirable for it to use the space in a cautious way. Therefore the risk criterion can be included in the optimisation criteria, and the areas that the robot had best avoid can be *tagged* as high-risk areas. In this way, although the robot is unlikely to collide with obstacles in these areas, the robot will be inclined to avoid these areas when this criterion is used as part of its path optimisation process. Examples of such areas could include regions where the surface material changes, and a robot repeatedly moving over them would wear them out faster than desired (grass or carpet covered surfaces around concrete paths for example). Or places where people might frequently be found and it is better to avoid them rather than sense (detect) the people and perform a last minute evasive action. So although the robot could cross these areas easily, it is better if it avoids them.

6.3. The Chromosomes Structure and Path Optimisation

A feature that shows how powerful genetic algorithms are, is the capacity to optimise paths even if the mobile robot (and its simulator) do not follow the exact path drawn by "Rutar", and manipulated by the genetic algorithm. As was explained in the previous chapter, if the interpretation of the path is properly simulated, the fact that the actual path that the mobile robot will follow is a variation of the path represented by straight segments, and not its exact boundaries and corners, it does not mean that the optimisation is being performed wrongly. Since the genetic algorithm uses the values given by the simulator, and the simulator produces the values that the robot would produce, the real path followed by the robot is being considered. Thus, the fitness given to a path is calculated on the basis of its actual execution by the robot and not by the graphical representation shown in the map. Since it is the fitness values that dictate the genetic operations to perform over the population, the genetic algorithm considers the real characteristics of the paths as seen by the robot and not by the user.

This means that, in energy and travel time terms, the genetic algorithm is optimising the representation of a path described in an arbitrary way. In other words, the relation between the description of the path and the actual path that the mobile robot will follow, depends entirely on the characteristics of the mobile robot used, and will change if the robot is changed. Moreover, it also depends on the speed at which the robot will try to follow the path. If this were to be described mathematically, it would mean that the optimisation would be performed by manipulating a path whose characteristics (fitness for the genetic

algorithm) are related in a highly non-linear way to its description. If this were to be optimised by mathematical methods utilising inverse kinematics and dynamics, it would be an extremely complex problem. But because of the effect of using direct kinematics and dynamics both in the process itself and in the simulator tool, the optimisation works just as efficiently, even if the robot distorts the paths in a highly non-linear way.

The chromosome structure that was used (as explained in section 5.2) also proved very efficient for the system, since it permitted a standardised input to the simulator to be made, and proved to be a fast way to perform the genetic operations on them. If the direct representation of "Rutar" paths had been used (the brick-road representation), the simulation and the genetic operations would have required more computer resources, so the optimisation process would have been slower. This is very interesting since it shows that for each stage of the process (generating and optimising a path) a different representation is required to get the higher efficiency for each tool. That is, in the generation stage the brick-road representation is much more efficient than using vectors, while in the optimisation stage the vector representation is the more efficient of the two.

6.4. "Rutar" and the Optimisation Process

Compared to the execution times of the genetic algorithm used to develop Rutar's rules of death (see section 3.2), the optimisation genetic algorithm executed very rapidly (3 - 10 hours compared to 50 – 60 seconds). It is important to clarify here that what is being

compared to the optimisation genetic algorithm is the genetic algorithm used to develop Rutar, not the execution time of Rutar itself (which was in the order of 200 ms, as presented in section 4.3).

An important aspect that produced this difference was the fact that the paths provided by "Rutar", and the populations generated from these paths, already have the main characteristics of near-optimal paths for the energy and time criteria, which are being smooth and, of course, feasible. These paths can be considered near optimal, because in many cases the optimal path produced by the optimisation process used the space toward the centre of gravity between the obstacles, as do the original path provided by "Rutar". Only when the optimisation is executed exclusively to minimise distance, are the optimal path results considerably nearer the obstacles as compared to the paths generated by "Rutar". Of course, in any case, Rutar's paths are not expected to be optimal (except by chance) since at the generation stage no robot characteristics are considered, nor are its orientation and origin and destination defined.

Another aspect that contributes to the high speed of execution of the optimisation process is the type of problem itself. In the case of path optimisation, small changes around a working (feasible) path are needed to find the minimum cost path. The length of the path is limited (by the optimisation criteria as explained in section 5.3) and the process itself tends to keep it as short as possible. This can be compared with problems like Rutar's rules of death, where there is no known limit to the length of the chromosomes, and there are no feasible answers before the final answer is found (i.e. either there is or there is not an

answer, no near-optimal answers exist). Also in the case of the development of Rutar, the genetic operations can result in all kinds of radical changes to the fitness of the individual. This can also be thought of in terms of the fact that, as in the case of the rules of death, the search has to be made by exploring the whole scope of an infinite search space. While in the optimisation process, the search is made by exploring only a small, limited region, around a known path that is always near the optimum that is being searched. Thus, the path optimisation problem is intrinsically efficiently performed by the genetic algorithm, and is therefore appropriate to produce fast results.

From the aspect of termination criteria, the optimisation process is more complicated than the rules of death. The genetic algorithm that found the rules of death for "Rutar" had a very clear goal. Some specific characteristics had to be met by the successful chromosome (answer), in other words, it was clear when an answer had been found. In the case of the optimisation process, there is not a known answer, that is, it is not known from the beginning what the minimum consumed energy (or time or risk or distance) should be, therefore it is difficult to know when the solution is found. It is even impossible to prove formally that the solution found corresponds to the real absolute minimum, unless another deterministic mathematical method could be used to prove this. That is why the standard deviation of the results produced by many executions was used (in section 5.4 and 6.1) as a means to verify that the optimal path was being found and not local minima.

To decide when to terminate the genetic algorithm, two criteria were used: one was a maximum allowed number of generations to run, and the other the fitness gradient curve.

The first one does not imply that an answer has been found, it just limits the process so that in no case will it go on executing forever. The second criterion tests the fitness of the individuals and decides that an answer has been found because nothing better is produced within the next generation. This does not formally guarantee that an answer has been found, but indicates that the search process is not improving the results any longer. Again, it can only be proved experimentally as was shown in section 6.1. By combining the standard deviation analysis with the termination criteria for no improvement between generations, it gives an indication that the optimal answer has been found.

From the discussions above, a typical characteristic of the genetic search method can be identified: this is that the time consumed by every part of the program can be very different depending on the configuration of the problem. In the case of the rules of death for "Rutar", most of the processing time was spent on the genetic manipulation of the chromosomes, while in the optimisation algorithm, most of the time was spent on the execution of the simulation process. In other words, for the development of Rutar, the main work for the genetic algorithm was to *try* a large number of individuals and manipulate them genetically. For path optimisation, the manipulation of chromosomes is much smaller and the main work consists of simulating them.

6.5. Fitness in Genetic Algorithms

The optimisation criteria used in the process are all in different forms. They are reviewed

as follows:

- Distance is dependent almost exclusively on the shape of the path and the size and steering capabilities of the mobile robot.

- Risk can be represented in a variety of ways depending on the application. In the case of the example problem used as the case study in this thesis (see risk figures in Appendix D), it depends entirely on which risk zones of the map the robot is passing through.

- Energy depends in a very complex way on the dynamic characteristics of the mobile robot, and the characteristics of the path that involve acceleration and deceleration.

- Travel time depends on the kinematics of the robot and its relation to the characteristics of the path.

The way in which travel time relates the kinematics of the mobile robot to the characteristics of the path is very different to the way in which energy consumed relates the dynamics of the robot to the shape of the path.

Apart from being related to different features of the robot, these different optimisation criteria also involve different search methods and different evaluation methods. Distance optimisation utilises an exclusively geometrical search method. Energy and time

optimisations need a mathematical search method and the risk utilises a minimisation method similar to the travelling salesman problem [Koza] [Goldberg].

Despite these profound differences between the various optimisation criteria used, the Genetic Algorithm approach provides the possibility of merging and relating all these aspects together and in a much simpler form than could be done mathematically.

The aspects that enable such an efficient mixture of different criteria is the fact that the Genetic Algorithm works based on a single fitness value, regardless of how it is defined. In this way, the different aspects are quantified, blended together and used as a single fitness value. The Genetic Algorithm then optimises the path taking into account the results, regardless of the form of each different assessment criterion.

This means that although the Genetic Algorithm is a very powerful tool that enables us to mix different types of highly non-linear elements of problems, its performance also depends very much on the configuration of the problem as defined by the designer. In particular, for the case in this thesis, the fitness is provided by means of a simulator, and the way of combining the different criteria is provided by the relative importance scheme.

Regarding the fitness measurement, the simulator is the critical part in the optimisation process. The genetic algorithm optimises the path according to the results given by the simulator regardless of the graphical shape of the path (as explained in sections 5.2 and 5.3). Thus the simulator is directly responsible for the quality of the results. At this point it

is important to clarify the point that using a simulator representing a real robot has no effect on the validity of the results, it would only validate the fidelity of the simulator, but not the optimisation process. Thus, to analyse the optimisation process it is more important to observe the effect of different aspects of the simulation regardless of which real or hypothetical robot it is simulating.

One interesting test that was performed was to use a simulator that represented a robot with a very high coefficient of friction in the drive system (e.g. its bearings). Figure 6.5 shows the result of performing the same optimisation process as the one performed in figure 5.9 (100% energy optimisation). The resultant path is in effect the same as for the minimum distance (shown in figure 5.8).

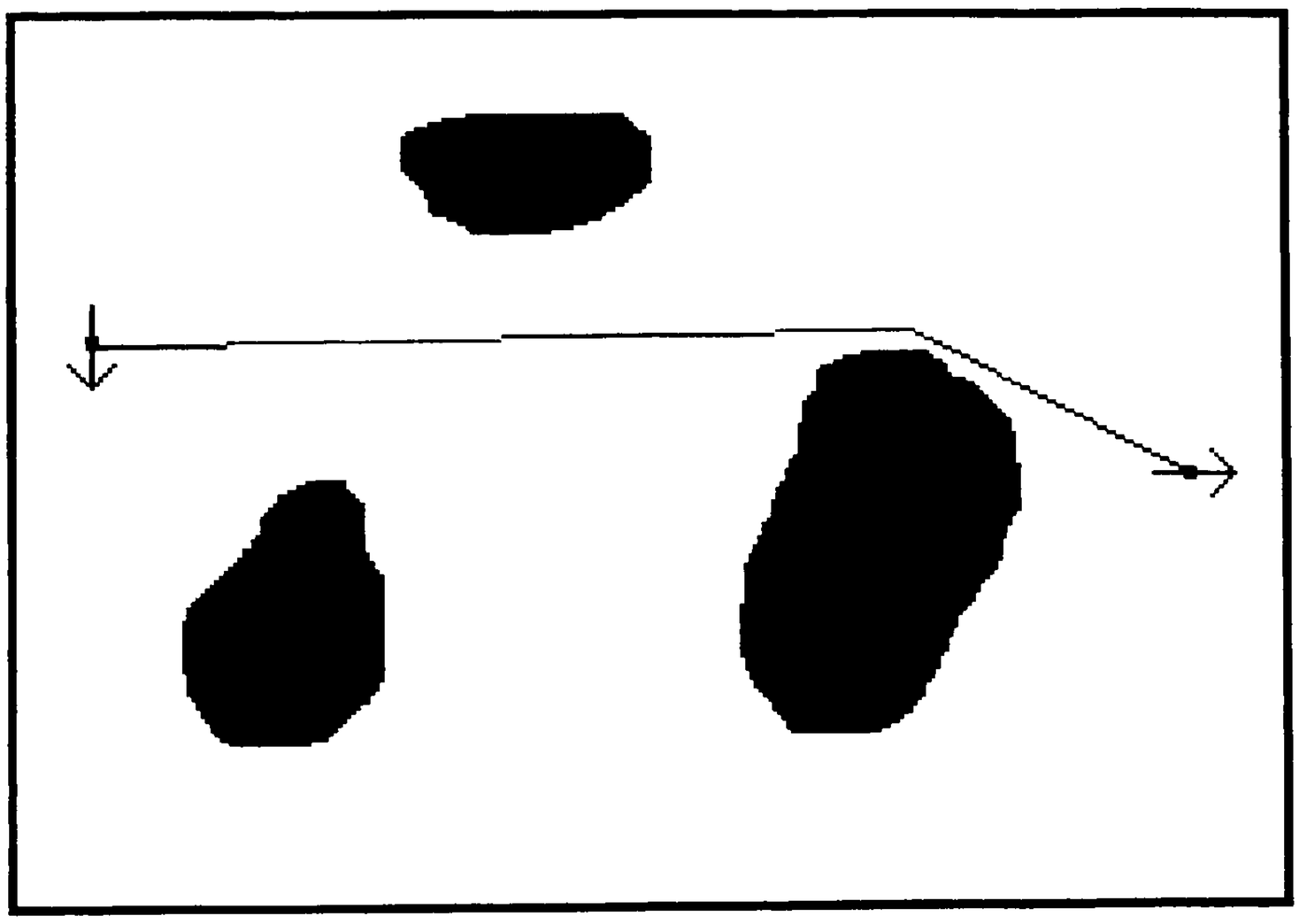


Figure 6. 5 100% Energy optimisation for a high friction robot.

In this case, provided by the simulation results, the path resulted in the same path as the minimum distance, since most of the energy is used exclusively in overcoming friction, therefore inertia has very little effect in the energy conservation/usage of the movement of the robot. In consequence, the minimum energy consumption is achieved by travelling the minimum distance because for every unit of distance travelled there is a directly proportional amount of energy used necessary to overcome friction.

This example shows that the genetic algorithm is in fact considering the characteristics of the robot for the path optimisation process. Whether or not the robot is being properly simulated is another matter, and depends on the quality of the design of the simulator (which is out of the scope of this work, since a simulator is just a standard module used as a tool).

Regarding the combination of criteria, the combination was achieved satisfactorily, but it is not possible to define formally how the combination is taking place. Because the optimisation criteria depend, in different ways, on the characteristics of the robot and the path, it is extremely difficult to define numerically an equivalence of their importance. This means that when the user enters the percentage of importance values for each of the optimisation criteria, the genetic algorithm uses these percentages to provide the relative importance between the normalised values of each of the criterion fitness results, but this is not the same as saying that the percentages are applied to the different criteria. In short, since an exact way of numerically relating the importance of each criterion has not been defined, the relative importance of each criterion as defined by the user is a subjective

measure. Thus, these relative importance percentages must be treated rather as fuzzy values than exact proportions, from the point of view of the user of the optimisation system. In the chapter that presents the recommendations for further work, a discussion on new soft-computing schemes is presented. There, an analysis of these aspects by means of other artificial intelligence tools is suggested.

Chapter 7

CONCLUSIONS

AND

RECOMMENDATIONS FOR

FURTHER WORK

In this chapter, the final conclusions are presented. After the conclusions, the following sections present some recommendations for further work that can be carried out. These recommendations are presented as guidelines on how to adapt, or enhance, the tools developed in this work, so that they can be used for on-line applications.

7.1. CONCLUSIONS.

The problem of path planning is one of the main considerations in the application of mobile robots. The ability to generate its own path, and to optimise it, gives the robot an extended functionality as an autonomous machine. In the research work that has been reported in this thesis, a novel system has been developed to generate and optimise paths in an efficient manner. A set of artificial intelligence techniques was used in a novel scheme to produce a set of tools for this purpose, of generating and optimising paths.

By using a two-stage system, generating and optimising the paths in two steps, a versatile and efficient scheme was achieved. The generation stage provides all the non-redundant paths, which are feasible and smooth on their own. The optimisation stage uses the non-redundant alternatives to produce an optimal path according to criteria defined by the user.

The generation stage.

The generation stage, embedded in a program called Rutar, was achieved in a novel way, by means of simulating a shrinking colony of cellular automata. The development of the behaviour rules for the cellular automata was performed using a genetic algorithm. This work showed four very interesting features of these two artificial intelligence tools, cellular

automata and genetic algorithms. And they are:

- Cellular automata are a very powerful tool that can produce complex and useful behaviour when used as a colony despite the very simple behaviour of each automaton. This makes it a very useful tool which executes very efficiently in digital computers to produce results, that otherwise would require much greater computer resources.

- To produce a complex and useful behaviour in a cellular automata colony, an appropriate set of rules has to be found. The difficulty in finding the desired set of rules is determined by the fact that it needs to be very special to produce a simple, yet very precise interaction between the automata, so that the overall behaviour of the colony will result in a macro-behaviour. That is, a behaviour that will produce a coherent result from the colony as a whole.

- Finding this *precise* set of rules of behaviour can be a very complex (or intricate) task. A genetic algorithm was used successfully for this purpose. From this search, a novel set of *rules of death* for the cellular automata was found that produces the desired results.

- Genetic algorithms are a very versatile search method that can be used for a vast range of applications and types of searches. The limitations on what can be searched for by a genetic algorithm are determined by the capacity of the user to configure the problem and design the assessment (fitness) criteria, as well as the termination criteria. In the

case of the path generation system development, the search was successful.

The optimisation stage.

Optimisation has been for a long time a very interesting subject for research. In the case of this work, a very complex optimisation task was performed. Path optimisation was performed in a novel way, under a set of criteria, some of which make it a challenging task to attempt, such as energy and time minimisation. The criteria used in this optimisation process (energy, time, risk and distance) were applied to a path between two points (origin and destination) which makes the problem very different to other time minimisation problems such as the travelling salesman example. In the case of this work, the characteristics of the robot make an important difference to the results of a path in terms of the criteria used, and there are (in theory) an infinite number of possible path variations that would produce different results, therefore the problem is very complex. It is also very complex to prove that the absolute best answer has been found.

Again, a versatile artificial intelligence tool was used to achieve the goal of developing a path optimisation system. With a genetic algorithm, a search for the optimum path was performed and, as shown by the results presented, it was successfully accomplished. This showed that the search configuration (chromosomes, population, assessment method, fitness and genetic operand management, and termination criteria) was performed correctly for this novel scheme of path optimisation.

A novel method for generating the initial population was developed to generate a large number individuals from a single alternative (provided by Rutar, in the form a non-redundant path). The method developed used the same mutation scheme that was used in the genetic algorithm operations, and was applied to copies of the original path; in this way, a large number of random individuals can be generated from a first feasible (though, not optimal) answer. Although this initial population was not entirely random, by applying random mutation to every copy made from the original path, the initial genetic variety for the algorithm was guaranteed.

One of the more interesting and novel characteristics of the optimisation system, developed in this work, is its capacity to optimise a path under a combination of the possible criteria. No research work had reported so far this capability. The optimisation using a combination of criteria was accomplished with no additional needs in terms of the problem's complexity or in computer resources.

The optimisation, and the capacity to combine criteria, were achieved by using a search scheme that used the results of a direct kinematics calculation for the robot. By using the direct kinematics solutions, all the mathematical components of the system were kept at their maximum simplicity, rather than using complex inverse kinematics mathematics.

Thus, the genetic algorithm was used successfully to perform a search in a direct way. The problem solved in this way, means that it was transformed from a very complex, inverse

mathematics problem (which additionally is highly non-linear), to a very repetitive direct mathematical problem. As many genetic algorithm applications have shown (including the one presented in the work) it is more efficient to perform highly repetitive simple calculations, than a few very complex problems.

In the case of the work presented so far, the relative importance of each criterion has been treated in a way such as *"energy is more important than time for this path"*, or *"risk and energy are important in this case while distance is not very important"*, etc. Some very extensive experimental and theoretical work is needed to look for a precise method to relate, numerically, the importance of each optimisation criterion. For this purpose, the most up-to-date applications of soft computing [Muscato G.] [Lin CT et. al.] can provide means for analysing these fuzzy and unclear relationships of this system. With these types of new approaches (similar in principle to the hierarchical approach that is presented further in section 7.2.1.1) where an artificial intelligence tool is applied to another one, the way in which a problem is solved could be interpreted. Understanding how the genetic algorithm finds the answers and combines the different criteria could help to develop mathematical models, which could provide means of quantifying the percentages of importance of each criteria.

Meanwhile, as in many applications a precise numerical correlation may not be needed, the system, as existing at this moment, can be useful after some experimental test (usage).

The genetic algorithms implemented.

The very different execution times for the genetic algorithms used in the development of the rules of death of Rutar and the optimisation system were discussed and analysed in chapter 6. The two algorithms had different problems to solve, and the search space was presented in a very different way for each one of them. The important outcome was that the execution times were suited for each type of application:

- For the genetic algorithm used to develop Rutar, the execution time was very long (several hours), but it was a one-time execution program. This means, that once the result was produced by the genetic algorithm (the *rules of death* for Rutar), the answer was used after that as a tool, and the algorithm was not executed any more.

- For the genetic algorithm used to optimise, the execution time is relatively low (less than a minute) which makes it useful for its application. This algorithm is the optimisation tool itself, therefore it has to be executed every time an optimisation process needs to be performed.

Execution times of the programs.

The work and results presented so far provide very encouraging evidence that even with low cost microcomputers, the execution times of the task involved in a path generation and

optimisation process can be made small enough to enable the robots to use them in some real-time applications.

With the execution times provided by the early systems when the trials for the work in this thesis began (286 based PCs and T800 transputers), the "less than a minute" execution times provided real-time capabilities for relatively slow mobile robots. For example, it is useful for mobile robots moving at speeds of less that 1 m/s, which is typical of the vast majority of today's robots that are limited by sensor range resolution coupled with safety constraints.

These execution times, now cut to less than half with present day computers (pentium based PCs and T820/T9000 transputers), can now be applied to faster mobile robots in industrial applications. And with computer processing speed increasing at a great rate, these techniques could be used in real-time in faster applications of other type of path planning robots.

Being able to provide real-time execution times means that these techniques can be applied in on-line optimisation processes and in on-line learning. In order to use the path generation and optimisation techniques presented in the previous chapters, in on-line optimisation or learning, some considerations need to be reviewed, and some additional support tools need to be considered. This will be presented in the recommendations for further work sections that follow.

7.2. RECOMMENDATIONS FOR FURTHER WORK.

ANALYSIS FOR POSSIBLE APPLICATIONS ON-LINE

An analysis was performed to provide a set of guidelines on how to use these techniques as part of an on-line optimisation, or learning, system; this analysis will be presented in the following sections. Additionally, an important tool was developed and tested by a team of researchers supervised by the author, as part of another project [Góngora M.A., 1995b and 1996a]. This tool is very useful in the on-line applications of the optimisation system presented in this thesis. This tool, called a *self simulator*, and the analysis performed to provide guidelines to use "Rutar" and the Optimisation algorithm are going to be presented in the following sections.

The guidelines presented here are a theoretical analysis that attempts to outline future work in the field of on-line optimisation in terms of how to use "Rutar" and the path optimisation genetic algorithm. This analysis (and the guidelines that result from it), is considered important information since it is the outcome of the experience gained by the author during the development and testing of the systems and results that have been presented in this thesis. The path generation and path optimisation tools developed in this research work are the core around which a much more complex and elaborate system can

be built for on-line path planning for mobile robots.

7.2.1. On-line application of "Rutar" and the path optimisation algorithm

With current personal computers, "Rutar" and the path optimisation algorithm developed in this work can be applied in on-line applications that can manage delays of up to a minute. This does not mean that for every path that a mobile robot has to plan, it has to wait for this length of time before starting. If a path can be planned while performing another part of the task such as following the previous segment, and provided that the robot's computer is capable of doing these two tasks simultaneously, there will be no delay in practice. Additionally, since Rutar's paths are smooth and relatively efficient in terms of energy and time (although not necessarily optimal), these paths could be used directly by an on-line system requiring very fast response time for dynamically changing environments. This can provide an excellent compromise between the need to keep re-planning the route or finding an optimal route.

This task planning using the tools described in this thesis can be achieved by enclosing the path generation, optimisation and execution in a hierarchical control scheme, which will now be considered.

7.2.1.1. Hierarchical control

On-line use of these programs implies the use of a higher level of control software (such as a mission control shell) that can control the execution of each task in the proper order providing them with the appropriate parameters. The higher order control software can be any automated control of a complete task, where path planning is an integral part of it, such as picking up a load from a work-cell and delivering it to a distant storage or processing point. If a mobile robot has to transport materials between work-cells, and in the mean time it needs to decide how to re-route its path depending on the status of a given work-cell, the complete mission would be subdivided into loading and unloading tasks, decision making tasks, and path planning tasks. The control software has to schedule each task according to the decision making task in conjunction with a pre-programmed objective. Figure 7.1 shows the structure of a mission control shell that could be used for a task involving path planning.

The control software has to decide, or be programmed by a human operator to decide, on the sequence in which to execute a complex task that includes path generation and optimisation. Once a path planning process is needed, the control software has to decide, or be programmed by the operator to decide, on which parameters to use for the execution of each part of the planning process.

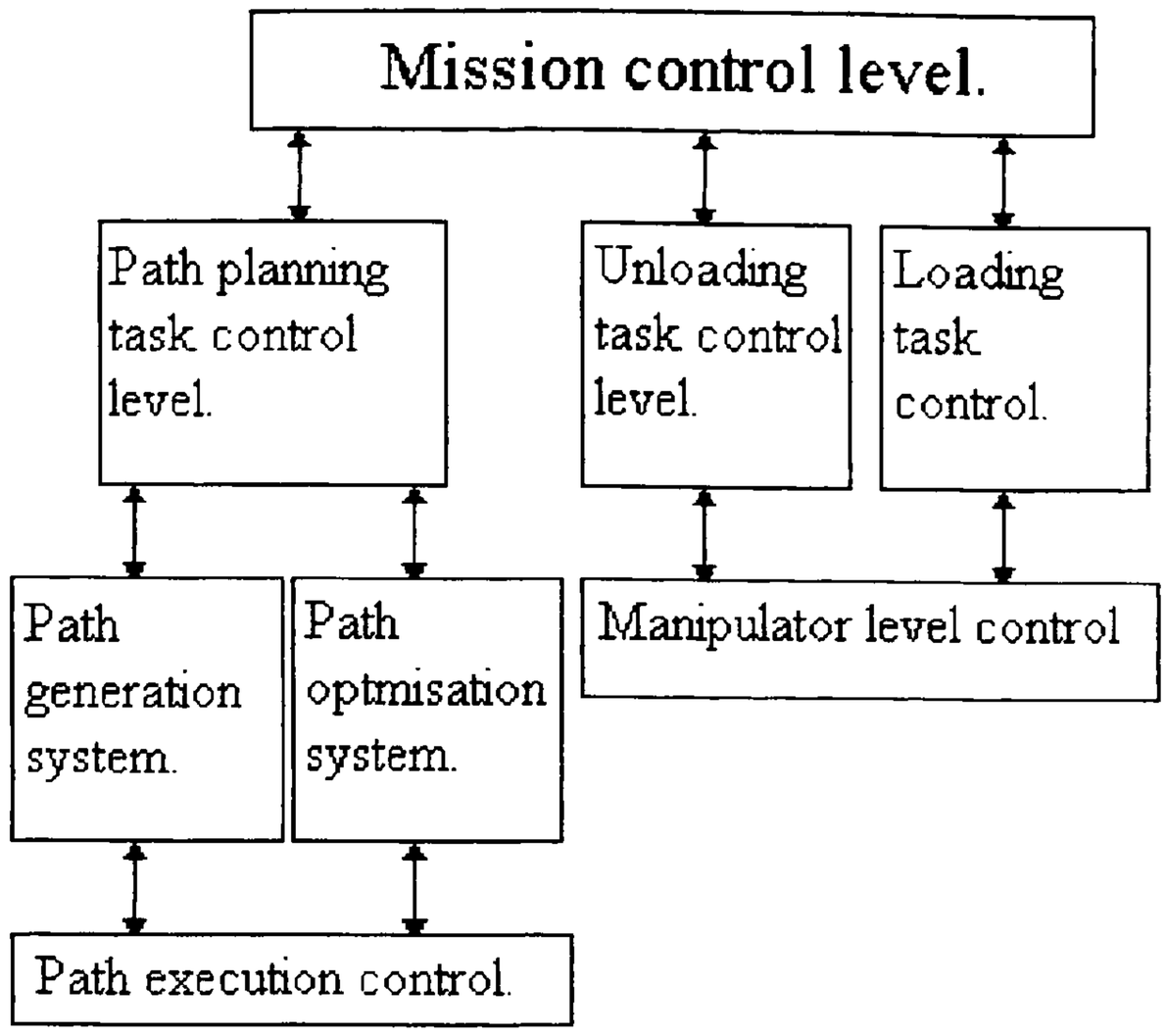


Figure 7. 1 Mission control shell

To use "Rutar" at its simplest, the control software needs to have the appropriate information to provide the correct parameters, which are:

- The maps need to be stored in memory, with information relating to when to use each map, i.e., for which of the path planning tasks it has to be used (if more than one path planning process is to take place in a more complex or composite task), and the anchor points to use. In highly automated systems, the maps can be provided by external sensors and associated recognition software (such as laser range finders or vision systems), and the anchor points could be extracted from the current position of the mobile robot and a detected, or defined, destination point to visit.

- The control software has to execute "Rutar" and to provide it with the map and the

anchor points needed for the path planning task.

- The control software has to receive the output from "Rutar", and save it to make it available to the next stage in the optimisation process.

Subsequently, the control software needs to execute the optimisation algorithm, which would entail the following:

- Execute the path extraction programs to generate the initial populations of mutated paths needed by the genetic algorithm. For this, it needs the information supplied by the user (human operator) on how many paths to extract from all the non-redundant possibilities generated by "Rutar", and on the initial population to generate. It can also use a general value, such as to consider all possible alternatives, or a percentage of them, so that no human intervention is needed during a long and complex mission.

- Execute the Genetic Algorithm with the population generated in the previous step. At this stage the information regarding the optimisation criteria, as well as the simulation parameters is needed. Again, these optimisation criteria can be stored from user defined values, or a higher level of automated mission control can define it, depending on the stage of the mission at which the process is taking place. The simulator's parameters, such as energy available and dynamic characteristics of the robot, can be taken from user defined data as well as from internal sensors of the robot (such as

battery charge gauges, etc.).

- Receive the result from the Genetic Algorithm, and save it, to then pass it over to the module that controls the mechanical system, when it is time to execute the path.

- At this stage, it is assumed that the simulator software is also stored in the robot's control computer and that the Genetic Algorithm optimisation process is enabled to execute it when it needs to evaluate the fitness.

Having considered these aspects, "Rutar" and the path optimisation algorithm can be used in a wide range of on-line applications. The control software can manage the scheduling of the path planning process, by executing the programs in advance while the previous task is executed, so that, if all tasks can be performed on schedule, the execution times will not produce a delay in the system. This of course, means that the robot's computer has to have multitasking capabilities. In the case of a timing problem, when the schedule is broken before being completed, the execution times will produce a delay in the start of the next new task, while re-planning the path. For this last case, it does not mean that the entire task schedule has to be redefined from the very beginning. For the purpose of using "Rutar" and the path optimisation algorithm on-line, a schedule determined in advance of the execution, being a little longer than the execution time, is enough to enable the use of these tools on-line with no delay. Therefore the schedule would have to be broken by an unpredictable event occurring a short time before the next task was due. This means that unpredictable events that can be detected some time in advance (the time equivalent of a

path planning execution) can be managed with no extra delays due to program execution times. If the application permits this condition, the system is useful for on-line purposes as it is.

7.2.1.2. Simulation aspects

The path optimisation process is based on the assumption that a suitable simulator is available to "predict" the kinematics and dynamic behaviour of the mobile robot. In the aspects discussed above, it could therefore be assumed, that the appropriate simulation program has to be available within the path planning computer (normally the robot's on-board computer for on-line applications) so that it can be executed in conjunction with the path optimisation algorithm.

If the robot's computer has a simulator of the robot itself, the term *Self-Simulator* has been coined for this, in this thesis. Although the term may not have much significance within the context of the simulator programs, the fact that it is a *Self-Simulator* is going to be of great importance in the application of the path optimisation algorithm on-line as will be explained in the following paragraphs.

A simulator is intended to "simulate" as nearly as possible the real system, so that trials for hypothetical situations can be performed without using the real system. The simulator's "quality" is determined by how similar its behaviour is as compared with the system it is

simulating. By having a *Self Simulator* in the situation that is proposed in this thesis, a very interesting aspect arises: the simulator can be assessed every (or nearly every) time it is used, since as soon as a path is optimised it is executed by the real system. This means that every time the robot follows a path, the exact outcome of the path can be measured by the robot's sensors, and compared to the simulated results, provided that they were saved when running the optimisation algorithm. An assessment of the simulator is not only important as the information "per se"; but additionally, if the simulator software has the appropriate structure, it can even be perfected on-line as the system works.

7.2.2. Self simulator

Since the simulator for the path optimisation system may have to be executed in the on-board computer of the robot, it is important to consider its structure in order to make it efficient in terms of CPU resource requirements. Additionally, as mentioned above, since the fact that a *Self Simulator* enables continuous assessment of its performance, a structure that can provide efficient assessment and even on-line modifications can be very useful.

For this purpose, a *Self-Simulator* was developed by the author in an Artificial Intelligence project in the Department of Electronics Engineering, Universidad Javeriana, Bogota D.C., Colombia [Gongora M.A., Obregon D., Ortega J.]. The author supervised this development, so that the correct structure was designed, making it a very useful tool for on-line applications of the results of the work presented in this thesis.

The *Self-Simulator* was developed to simulate the kinematics and dynamics of a mobile robot available in the Department of Electronics Engineering of the Universidad Javeriana. Appendix F shows the characteristics of the mobile robot and some details of the simulator developed.

To define the criteria used to design the structure of the *Self-Simulator*, an analysis of the work presented in this thesis was performed. This analysis will be presented in the following sections.

7.2.2.1. Structure of the Mobile Robot.

The robot's physical construction was analysed to provide the first aspects of the structure of the simulator.

The mobile robot has two main systems that affect its kinematics and dynamics: the electronic control circuitry, and the mechanical system (controlled by the electronics in a closed loop) including the motors and actuators.

The electronic control system, as can be seen in Appendix F, is based on two digital motor control I.C.s, a microprocessor, and a classical H bridge mosfet power driver for DC motors. From this system, all the digital electronics is completely defined and documented

and has been designed so as to facilitate being simulated in software.

The advantage of this digital electronics part of the robot is that from the source of the software used by the microprocessor, and the manufacturers data sheets provided for the I.C.s, the exact behaviour of the system can be determined, and this behaviour is completely stable and independent of ageing of the circuit, the components, etc. (two equivalent microprocessors will behave exactly the same despite the slight differences in manufacturing, while two equivalent transistors can have slightly different characteristics and they even change with time, temperature, etc.). For this reason, this part of the robot can be simulated by straight forward software programmed to calculate the same values as those *calculated* by the actual electronics hardware with the given digital inputs.

The power electronics part however, is greatly affected by the component's characteristics (Mosfets, resistors, etc.), and by the electrical characteristics of the DC motors. The mechanical system has a very non-linear behaviour that at the time of its design was not even completely defined or determined. Additionally, all moving parts are subject to wear with use, which will make the behaviour of the mechanical system change significantly over time. Thus, this part of the mobile robot has to be simulated by a technique that can work with highly non-linear systems, and it is this part of the simulator that can be assessed in terms of its performance, and if possible, be modified with time to make it more similar to reality, each time.

The *Self-Simulator* development project included an analysis of the different alternative

Artificial Intelligence techniques that could be used to design the simulator with the desired characteristics. As shown in Appendix F, two artificial neural networks modules were used to implement the power electronics and mechanical systems, and as explained before, a module was programmed to simulate (reproduce) the digital electronics part. The development of the *Self Simulator* was not exclusively the work of the author, but the concept for self simulation in the context of this thesis and the criteria of its design was entirely the authors own work.

7.2.2.2. Possible assessment method

Once the simulator is installed in the mobile robot, its simulated results can be compared with the actual path followed by the robot, to assess its performance. Since the digital electronics part is considered to be accurately simulated, the assessment is applied only to the Neural Networks module.

Since the simulator returns data in the form of Energy used, travel Time and Distance covered, these are the data sets that have to be collected from the mobile robot executing the path. Therefore, a data acquisition system needs to be included in the robot, to read the internal state sensors and save the data while the mobile robot is moving, and finally produce the total itemised data set of results at the end of the path. Each of these variables can then be compared to its corresponding predicted result saved from the simulation process.

In theory, if the results of the variables involved in the optimisation (Energy. time and distance) are equal for both the simulator and the real system, then the simulator is performing well. But it is possible (although very unlikely) that the simulator predicted a different path than that followed by the robot, but both produced identical results for the same variables. If the simulator remains unaltered, then the assumption that this possibility is very unlikely, can be accepted (in any case the resulting movement is that which is needed). But if the simulator is to be altered then this level of assessment still has an important disadvantage in the form of two main limitations:

- The internal state sensors of the mobile robot will provide (within their defined ranges) the information about what the mechanical system was "instructed to do", but will fail to provide information on other factors such as if the robot slipped or skidded, etc. If these factors have a significant effect on the path of the robot, external sensor information has also to be included, and integrated with the internal sensors, so that a more precise description of the actual path can be used as a pattern for comparison.

- The simulator does not provide the internal sensors either. So this additional feature of the simulator has to be included. This information, which will not be used in the fitness calculations for the genetic algorithm, needs to be considered in the correction stage.

The assessment resulting from comparison would be performed mainly on the optimisation variables. But when a correction is to be made, the actual path followed by the mobile

robot and the one predicted by the simulator have to be used to re-train the neural networks.

With the development of the *Self-Simulator* it was possible to show how the neural networks could be left in a continuous "supervised training mode" so that the assessment of the simulator's performance could be used to update the simulator, either to correct initial differences with the real system, or to adapt to new characteristics due to wear, replacement of bearings or tyres, etc. [Góngora, M.A. 1995b].

7.2.2.3. Modularity provided by the "Self Simulator".

As seen in Figure 5.5, the program structure of the path optimisation algorithm uses the simulator as a completely independent module. By having a *Self-Simulator* for each robot, this results in a very versatile system, since to use it in another mobile robot only the simulator needs to be changed. Since the simulator is a completely independent and self-contained module, the *Self Simulator* for each robot in which the system is going to be used can be replaced quickly and easily.

7.2.2.4. On-Line learning

With an integrated system, as shown in figure 7.1, an approach to on-line learning can be

produced. The structure of the tools developed in this thesis has provided an environment for further work in on-line optimisation and learning. The analysis presented in this chapter should help the future workers to develop an efficient on-line learning system using "Rutar", the optimisation genetic algorithm, and the suggested simulation techniques.

7.2.3. Trade-offs for On-line applications

As presented along with the description of the development and analysis of "Rutar" and the optimisation genetic algorithm, both "Rutar" and the Path Optimisation Algorithm use a number of parameters that determine how the data is going to be used. In on-line applications these parameters can be used to control the execution times so that the system can be more efficient in real-time.

Referring again to section 7.2.1.1., and remembering the versatility of "Rutar" with map resolution, it can be seen that in the cases where there is "free time" between the execution of tasks, high map resolutions and a high number of individuals can be used for the path generation and optimisation. In this way high quality, and accuracy, of planning can be achieved.

In the case of very late schedule changes, trade-offs can be considered. If the delay that re-planning will cause is acceptable, high quality planning can be used. But if the delay needs to be kept at a minimum, a lower resolution and lower number of individuals can be used,

so that the delay will be less, with of course, a loss in planning quality.

Although a trade-off is unavoidable, the important aspect is that all the tools studied and developed in this thesis provide the versatility necessary to make them useful in future on-line applications.

REFERENCES

Abe T.; Shibata T.; Nose M.; Tanie K; *Motion planning for a cutting task based on criteria of skilled operators*, Nippon Kikai Gakkai Ronbunshu, C Hen/Transactions of the Japan Society of Mechanical Engineers, Part C, Jan 1996, Vol 62, No 593, pp 215-222, JSME, Tokyo, Japan.

Anderson, J.A.; *An introduction to neural networks*, Cambridge Mass, London MIT press, 1995.

Asano T.; Giubas L. ; Hershberger J. ; Imai H. ; *Visibility polygon search and euclidean shortest path*, 26th Simposium on Foundations for Computer Science, Portland OR, Oct. 21-23, 1985, pp. 155-164

Boucher, A; *Parallel machines*; Minds and Machines, Nov 1997, Vol.7, No.4. pp.543-551

Cameron, R.D.; Dixon, A.H.; *Symbolic computing with LISP /Robert D.* Cameron ; Englewood Cliffs, NJ: Prentice-Hall.

Cao B.; Dodds G.I.; Irwin G.W.; *Approach to time-optimal, smooth and collision-free path planning in a two robot arm environment*, Robotica, Jan-Feb 1996, Vol 14, No pt 1, pp 61-70, Cambridge Univ. Press, New York, NY, USA.

Cho, HK;Lee, BH;Ko, MS ; *Time-optimal motion of a cooperating multiple robot system along a prescribed path*, IEEE Transactions on Systems, Man and Cybernetics, Dec 1995, Vol 25, No 12, pp 1648-1657, IEEE, Piscataway, NJ, USA

Cook, D.J.; Varnell, R.C.; *Maximizing the benefits of parallel search using machine learning*. Proceedings of the National Conference on Artificial Intelligence, 1997, pp.559-564

Davis L.; *Handbook of Genetic Algorithms*. Van Nostrand Reinhold,New York.

DeLaRosa F.; Laugier C.; Nágera J.; *Robust path planning in the plane*, IEEE Transactions on Robotics and Automation, April 1996, Vol 12, No 2, pp 347-351, IEEE, Piscataway, NJ, USA

Desaulniers, G;Soumis, F; *Efficient algorithm to find a shortest path for a car-like robot*, IEEE Transactions on Robotics and Automation, Dec 1995, Vol 11, No 6, pp 819-828, IEEE, Piscataway, NJ, USA

Elnagar, A;Basu, A; *Local path planning in dynamic environments with uncertainty*, Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, Las Vegas, NV, USA, Oct 2-5 1994, (Conf. code 42317), pp 183-190, IEEE, Piscataway, NJ, USA

Falkenauer E; *Genetic Algorithms and Grouping Problems*; Wiley; 1998, isbn 0-471-97150-2.

Fujimori A,; Nikiforuk P.N.; Gupta M.M.; *Adaptive navigation of Mobile Robots with obstacle avoidance*, IEEE Transactions on Robotics and Automation, August 1997, Vol 13, No 4, pp 596-599, IEEE, Piscataway, NJ, USA

Geng, Z., Jamshidi, M., Liebowitz, J.; *Design of self-learning controllers using expert system techniques.* ; Third Int Symp Intell Control., 1988, pp.551-558, IEEE, IEEE Service Center, Piscataway, NJ, USA.

Gill, M.A.C.; Zomaya, A.Y.; *Genetic algorithms for robot control*, Proceedings of the IEEE Conference on Evolutionary Computation, 1995, Vol.1, pp.462-466

Goldberg, David E.; *Genetic algorithms in search, optimization, and machine learning*, Reading, Mass., Wokingham, Addison-Wesley, c1989

Góngora M.A., 1993a, *"TOtOLL: Task Optimisation through On-Line Learning"*, PhD. Progress Report, Departament of Engineering, University of Warwick (Coventry, U.K.), April 1993

Góngora M.A., Goodhead T.C., Hines E.L., 1994a, *"High Reliability Path Generation For Automated Systems"*, Conference Proceedings of the IASTED International Conference On Reliability Engineering and its Applications, Honolulu, Hawaii, USA, August 15-17, 1994, pp. 43-47.

Góngora M.A., Goodhead T.C., Hines E.L., 1995a *"The Use of Wandering Intelligence in Mission Control for Mobile Robots"*, Conference Proceedings of the Fourteenth IASTED International Conference on Modelling, Identification, and Control ,Innsbruck (Igls), Austria, Feb 24-26, 1995, pp. 331-334.

Góngora M.A., Obregon D., Ortega J.; 1995b *"Internal Self-Simulator for a Mobile Robot"*, Conference Proceedings of the IASTED International Conference on Modelling and Simulation, Colombo, Sri Lanka, July 26-28, 1995.

Góngora-Florian, M.A., 1996a, *"Robust Path Optimization using Self Simulation"*,

IASTED International Conference on Modelling, Simulation and Optimization, Gold Coast, Australia, May 6-9, 1996.

Graham, P.; *ANSI Common Lisp*; Englewood Cliffs, N.J.; London: Prentice Hall, c1996, Prentice Hall series in artificial intelligence.

Haykin, S.; *Neural Networks*; Macmillan, 1994

Hwang Y.K.; Ahuja N.; *Gross motion planning - A survey*, ACM Computing Surveys, vol. 24 no. 3, pp. 219-291, Sept. 1992

IbarraZannatha J.M.; SossaAzuela J.H.; GonzalezHernandez H.; *New roadmap approach to automatic path planning for mobile robot navigation*, Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics. Part 3 (of 3), San Antonio, TX, USA, Oct 2-5 1994, (Conf. code 42882), Vol 3, pp 2803-2808,0884-3627

Janét J.A.; Luo R.C.; Kay M.G.; *Autonomous mobile robot global motion planning and geometric beacon collection using traversability vectors*, IEEE Transactions on Robotics and Automation, February 1997, Vol 13, No 1, pp 132-136, IEEE, Piscataway, NJ, USA

Kanayama Y., Yuta S.; *Vehicle Path Specification by a Sequence of Straight Lines.* In

IEEE Journal of Robotics and Automation. Vol. 4, No. 3. June 1988.

Kang D.; Hashimoto H.; Harashima F.; *Path generation for mobile robot navigation using genetic algorithm*, IECON Proceedings (Industrial Electronics Conference) 1995, Proceedings of the 1995 IEEE 21st International Conference on Industrial Electronics, Control, and Instrumentation. Part 1 (of 2), Orlando, FL, USA, Nov 6-10 1995, (Conf. code 44293), Vol 1, pp 167-172, IEEE, Los Alamitos, CA, USA

Katoh R.; Ichiyama O.; Yamamoto T.; Ohkawa F.; *Real-time path planning of space manipulator saving consumed energy*, Proceedings of the 20th International Conference on Industrial Electronics, Control and Instrumentation. Part 2 (of 3), Bologna, Italy, Sep 5-9, 1994, (Conf. code 42923), Vol 2, pp 1064-1067, IEEE, Los Alamitos, CA, USA.

Koditschek D.E.; *Robot planning and control via potential functions*, in Robotics Review, Khatib O., Graig J., LozanoPerez T., Eds., Cambridge MA, MIT press 1989, vol 1.

Kosko, Bart; *Neural Networks and Fuzzy Systems*; Prentice Hall, 1992

Koza, John R.; *Genetic programming, on the programming of computers by means of natural selection*, Cambridge, Mass., London, MIT, c1992

Koza, John R.; *Genetic programming 2, automatic discovery of reusable programs*,

Cambridge, Mass., London, MIT Press, 1994

Latombe C.J.; *Robot motion planning*, Boston MA, Kluwer 1991.

Lee D.T.; Drysdale R.L.; *Generalized Voroni Diagramas in the Plane*, SIAM Journal of Computing, vol. 10, no. 1, 1981, pp. 73-87.

Lee J.H.; Lee B.H.; Choi M.H.; Kim J.D.; Joo K.T.; Park H.; *Real time traffic control scheme for a multiple AGV system*, Proceedings of the 1995 IEEE International Conference on Robotics and Automation. Part 2 (of 3), Nagoya, Japan, May 21-27 1995, (Conf. code 43853), Vol 2, pp 1625-1630, IEEE, Piscataway, NJ, USA

Lin, CT; Lee, CSG; Neural Fuzzy Systems: *A neuro-fuzzy synergism to intelligent systems*, Prentice Hall, 1996, ISBN 0-13-235169-2.

Madan, S.; *Benchmark for artificial intelligence applications on parallel computers - BEAP*; IEEE WESCANEX Communications, Power, and Computing, 1997, pp.82-87

Marik V.; Stepankova O.; Trappl R. (Eds.); *Advanced topics in artificial intelligence*, International Summer School, Prague, Czechoslovakia, July 6-17, 1992, Berlin, London, Springer-Verlag, 1992

Martin P.; delPobil A.P.; *Application of artificial neural networks to the robot path*

planning problem, Proceedings of the 9th International Conference on Applications of Artificial Intelligence in Engineering, University Park, PA, USA, July 19-21 1994, (Conf. code 21476), pp 73-80.

MartinezAlfaro, H;Flugrad, DR ; *Collision-free path planning for mobile robots and or AGVs using simulated annealing*, Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics. Part 1 (of 3), San Antonio, TX, USA, Oct 2-5 1994, (Conf. code 42882), Vol 1, pp 270-275, IEEE, Piscataway, NJ, USA

Meng, Qc;Haman, Y; *Optimal dynamic control of a mobile robot by genetic algorithm with symmetric code-GASC*, Proceedings of the 1995 IEEE Conference on Control Applications, Albany, NY, USA, Sep 28-29 1995, (Conf. code 44033), pp 1146-1147, IEEE, Piscataway, NJ, USA

Moran, A;Hayase, M ; *Shortest-trajectory control of autonomous mobile robots using nonlinear observers*, Proceedings of the 34th SICE Annual Conference, Hokkaido, Japan, July 26-28, 1995, (Conf. code 44018), pp 1415-1418, Society of Instrument and Control Engineers (SICE), Tokyo, Japan

Muscato, G; Soft *Computing techniques for the control of walking robots*, Computing & Control Engineering Journal, August 1998, pp. 193-200.

Nebendahl, Dieter (ed.); *Sistemas Expertos*; Editorial Marcombo S.A. and Siemens

Aktiengesellschaft, 1988 (spanish translation of: Experten-Systeme, Siemens Aktiengesellschaft, ISBN 3-8009-1495-6).

Noborio, H; *Functional extension of the sufficient condition to design a family of sensor-based deadlock-free path-planning algorithms*, Proceedings of the 20th International Conference on Industrial Electronics, Control and Instrumentation. Part 2 (of 3), Bologna, Italy, Sep 5-9 1994, (Conf. code 42923), Vol 2, pp 1058-1063, IEEE, Los Alamitos, CA, USA

Payeur P.; Gosselin C.M.; Laurendeau D.; *Analysis of path-planning strategies in dynamic computer vision guided teleoperation*, Proceedings of SPIE - The International Society for Optical Engineering, Telemanipulator and Telepresence Technologies II, Philadelphia, PA, USA, Oct 25-26 95, (Conf. code 22418), pp 74-85.

Peitgen, Heinz-Otto; *Chaos and fractals, new frontiers of science*, Heinz-Otto Peitgen, Hartmut Jurgens, Dietmar Saupe, New York, London, Springer-Verlag, 1992

Rao, S.S.; Bhatti, P.K .; *Optimization in the design and control of robotic manipulators: A survey,* Applied Mechanics Reviews, April 1989, Vol.42, No.4, pp.117-128

Sharma R.; LaValle S.M.; Hutchinson S.; *Optimizing robot motion strategies for assembly with stochastic models of the assembly process*, IEEE Transactions on Robotics and

Automation, April 1996, Vol 12, No 2, pp 160-172, IEEE, Piscataway, NJ, USA

Shiller, Z;Chang, H ; *Trajectory preshaping for high-speed articulated systems*, Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME, Sep 1995, Vol 117, No 3, pp 304-310, ASME, New York, NY, USA

Song, K.T., Sun, W.Y.; *Robot control optimization using reinforcement learning.* ; Journal of Intelligent and Robotic Systems: Theory & Applications, Mar 1998, Vol.21, No.3, pp.221-238.

Stentz, A;Hebert, M ; *Complete navigation system for goal acquisition in unknown environments*, Autonomous Robots, 1995, Vol 2, No 2, pp 127-145, Kluwer Academic Publishers, Dordrecht, Netherlands.

Sullivan, J.C.W., Pipe, A.G.; *Evolutionary optimization approach to motor learning with first results of an application to robot manipulators.* ; Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 1997, Vol.5, pp.4406-4411.

Tanimoto,S.L.; *The elements of artificial intelligence :using common LISP* ; New York; Oxford: Computer Science Press, Principles of computer science series.

Tzionas P.; Tsalides P.; Thanailakis A.; *Cellular automata based minimum cost path*

estimation on binary maps, IEE Electron Letters, vol. 28, no.17, pp. 1653-1654, 1992.

Wang, Q., Zalzala, A.M.S.; *Genetic control of near time-optimal motion for an industrial robot arm* ; Proceedings - IEEE International Conference on Robotics and Automation, 1996, Vol.3, pp.2592-2597

Zalzola, A.M.S.; Morris, A.S. (editors); *Neural networks for Robotic control : theory and applications,* London, Ellis Horwood, 1996.

Appendix A

Development of Rutar

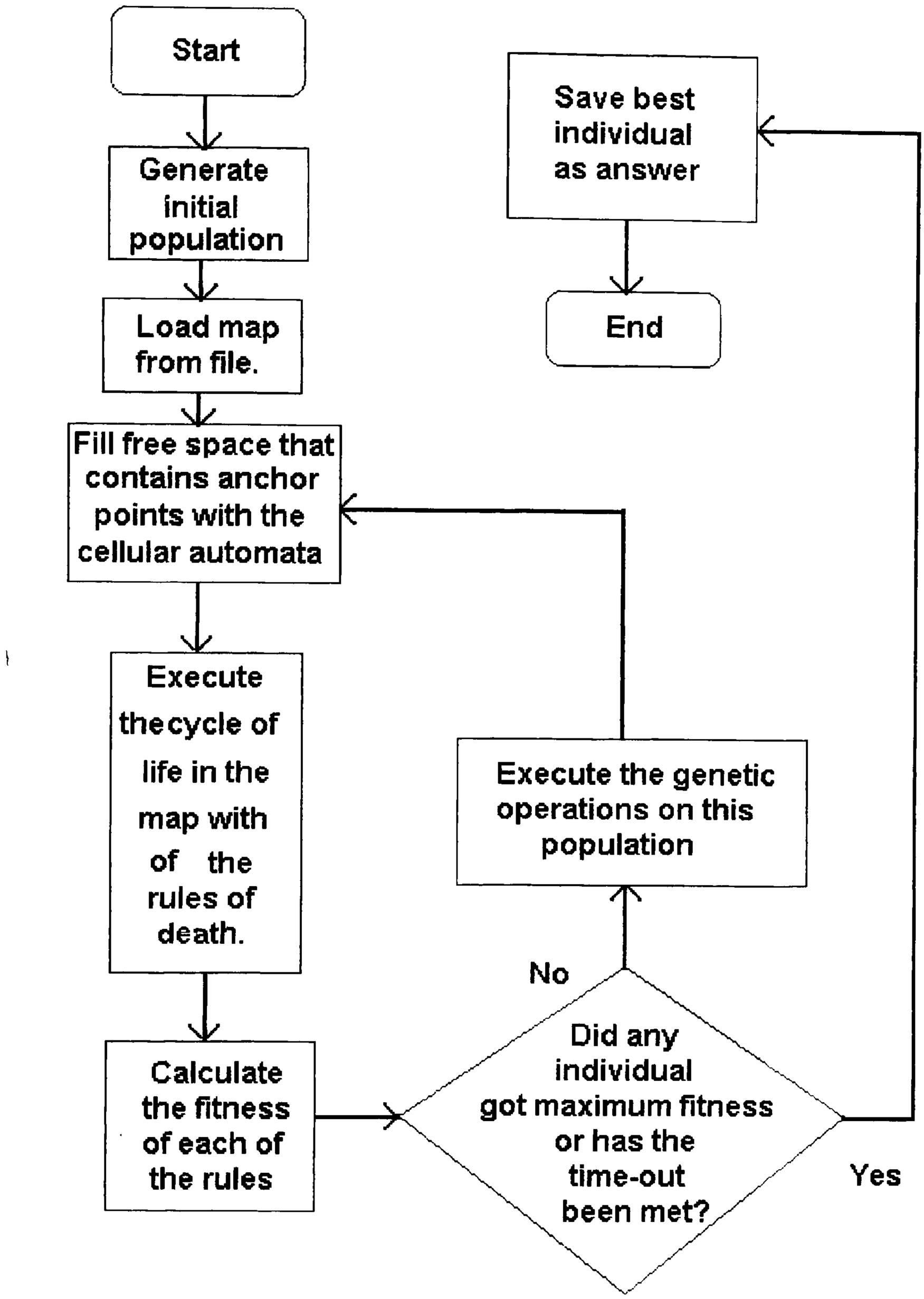


Figure A. 1 Flow diagram of the genetic algorithm used to find the cellular automata rules of behaviour

Some of the best individuals in the first T800 execution that finished from time-out.

These individuals were the result of an execution that terminated from time-out due to the failure to obtain minimum distance. The chromosomes have been arranged so that the similarities with the final best individual can be seen.

OR (OR (AND (OR (AND (AND (AND (AND ((8D), (5D)), (3A)), (2A)), (4P)), OR (AND (AND (AND (AND ((4A), (5A)), (3A)), (2A)), (6A)), OR (OR (AND (AND (AND ((4A), (8A)), (6A)), (3A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)), OR (3D,AND (AND (AND (AND ((1A), (5A)), (6A)), (7A)), (8A)),OR (AND (AND (AND (AND ((8A), (7A)), (6A)), (2A)), (1A)), OR (AND (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A))))))))), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (AND(AND (AND (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))), OR (OR (OR (OR (OR (OR (OR ((8P), (7P)), (6P)), (5P)), 4P), 3P), (2P)), (1P))), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A)), (2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)), (6A)), OR (AND (AND (AND (OR (OR (OR (OR ((1A), (2A)), (5A)), (4A)), (3A)), (6D)), (8D)), (7A)), (8A)))))))))

AND (OR (OR (OR (AND (AND (AND (1A), OR (AND (AND (AND (AND ((1A), (4A)), (3A)), (5A)), (6A)), OR (OR (AND (AND (AND ((6A), (5A)), (3A)), (4A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)), AND (AND (AND (AND (AND ((6A), (7A)), (8A)), (2A)), (1A)), OR (OR (AND (OR (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A))))))), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (AND(3P, (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((6A), (6A)), (7A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND (AND (AND (AND (AND (AND (AND ((7D), (8D)), (6D)), (5A)), (4D)), (3D)), (2D)), (1A), OR (AND (OR(AND

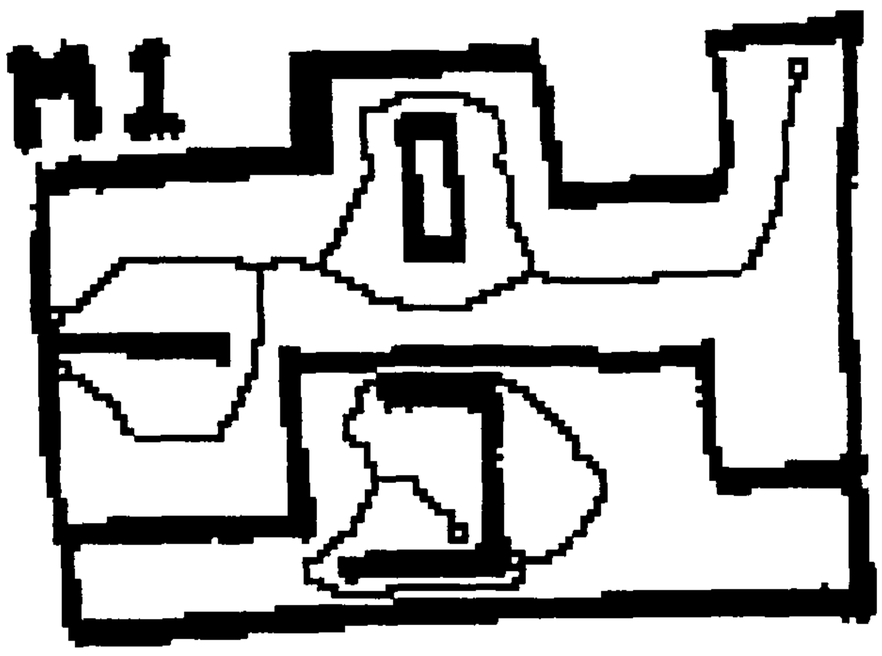
((7A), AND((AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (OR (AND (OR (OR (OR ((7A), (5A)), (3A)), (1A)), (6A)), (2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)), (6A)), OR (AND (AND (AND (OR (AND (OR (OR ((1A), (2A)), (5A)), (4A)), (8D)), (5D)), (3A)), (7A)), AND (AND (AND (AND (OR (AND (OR ((7A), (1A)), (6A)), (5A)), (8A)), (6A)), (4A)), (3A)))))))))

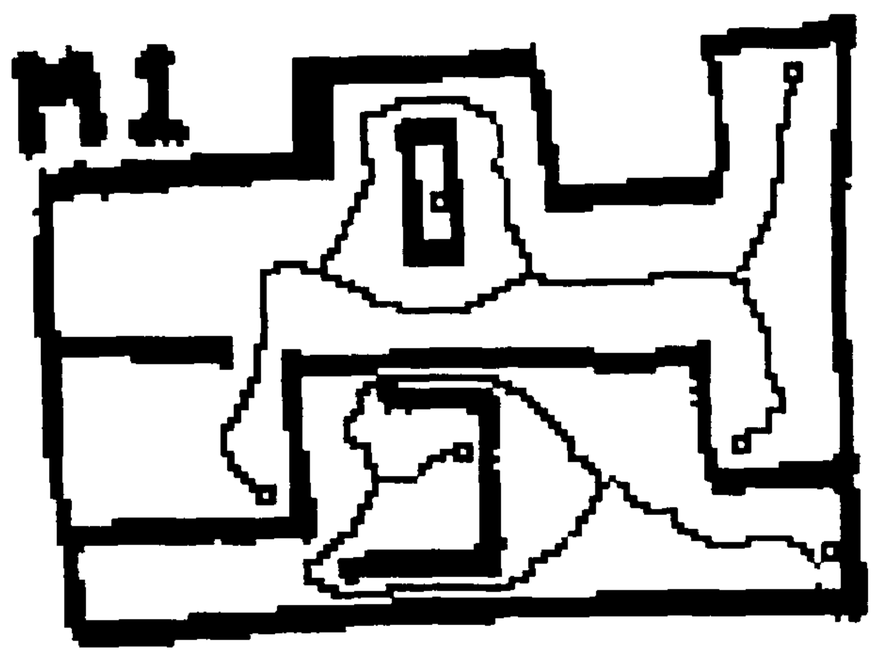
AND (AND (AND (OR (AND (AND (AND (AND ((2A), (5A)), (3A)), (4D)), (1A)), OR (AND (AND (AND (AND ((3A), (6A)), (5A)), (4A)), (2A)), OR (OR (AND (AND (AND ((4A), (5A)), (3A)), (6A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)), OR (, AND (AND (AND (AND ((3A), (1A)), (5A)), (8A)), (8A)), OR (AND (AND (AND (AND ((6A), (7A)), (8A)), (2A)), (1A)), OR (AND (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A)), OR (AND (AND (AND (AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (6A)), (2D)), (1D)), OR (AND (AND(AND (AND (AND (AND (AND ((7A), (8A)), (6A)), (5A)), (4D)), (3D)), (2D)), (1A), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))), OR (OR (OR (OR (OR (OR (OR ((8P), (7P)), (6P)), (5P)), 4P), 3P), (2P)), (1P))), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A)), (2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)), (6A)), OR (OR (OR (OR (OR (OR (OR (OR ((1A), (2A)), (5A)), (4A)), (3A)), (6D)), (8D)), (7A)), OR (OR (OR (OR (OR (OR (OR (OR ((7A), (1D)), (1A)), (2A)), (2A)), (8D)), (5D)), (7A))))))))))))))))

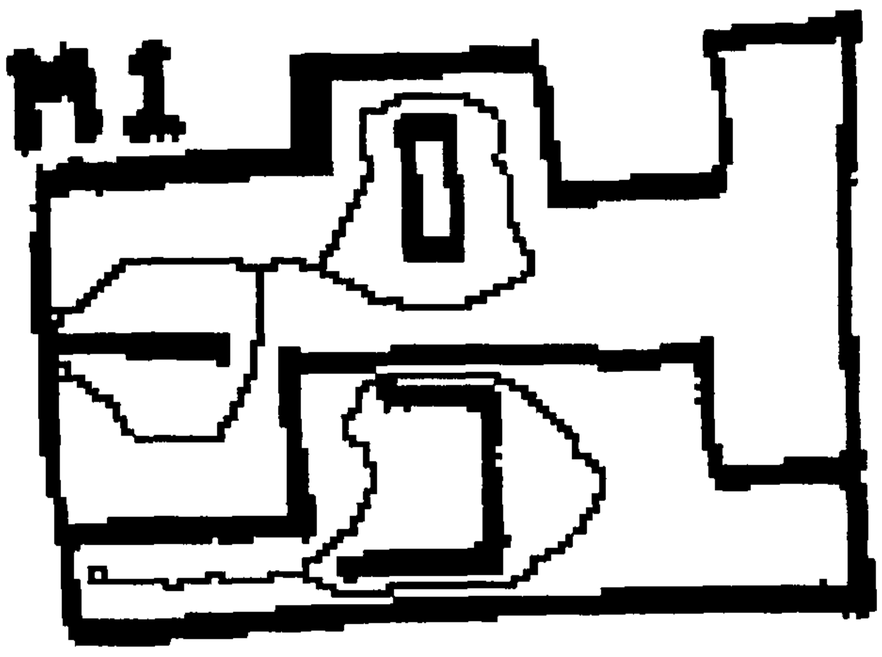
OR (OR (OR (OR (AND (AND (AND (AND ((5A), (4A)), (2A)), (2A)), (1A)), OR (AND (AND (AND (AND ((2A), (4A)), (5A)), 32A)), (6A)), OR (OR (AND (AND (AND ((4A), (5A)), (3A)), (6A)), (7A)), OR (AND (AND (AND (AND ((4A), (5A)), (6A)), (7A)), (8A)),OR (,AND (AND (AND (AND ((1A), (5A)), (6A)), (7A)), (8A))OR(AND (AND (AND (AND ((7A), (8A)), (3A)), (2A)), (1A)), AND (AND (AND (AND ((4A), (8A)), (3A)), (2A)), (1A)))))))), OR (AND (AND(AND (AND (AND (AND ((7D), (8A)), (6D)), (5D)), (4A)), (3D)), (2D)), (1D)), OR (AND (1A), OR (AND (AND(AND (AND (AND (AND (AND ((7D), (8D)), (6A)), (5D)), (4D)), (3D)), (2A)), (1D), (AND (AND(AND (AND (AND (AND (AND ((7A), (8D)), (6D)), (5D)), (4D)), (3A)), (2D)), (1D))))))), OR (AND (AND (AND (OR (OR (OR (OR

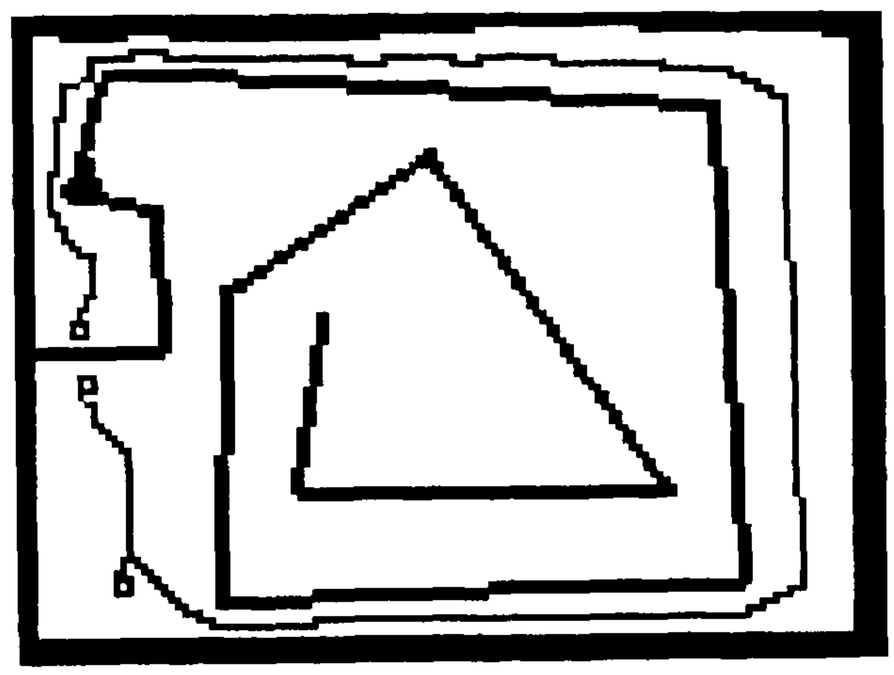
((7A), (6A)), (5A)), (4A)), (3A)), (2D)), (8D)), (1A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (3A)), (1A)), (4D)), (8D)), (2A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (6A)), (5A)), (7A)), (1A)), (2D)), (4D)), (3A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (5A)), (1A)), (3A)), (2D)), (6D)), (4A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (8A)), (1A)), (2A)), (3A)), (6D)), (4D)), (5A)), OR (AND (AND (AND (OR (OR (OR (OR ((7A), (1A)), (5A)), (2A)), (3A)), (4D)), (8D)), (6A)), (7P))))))))

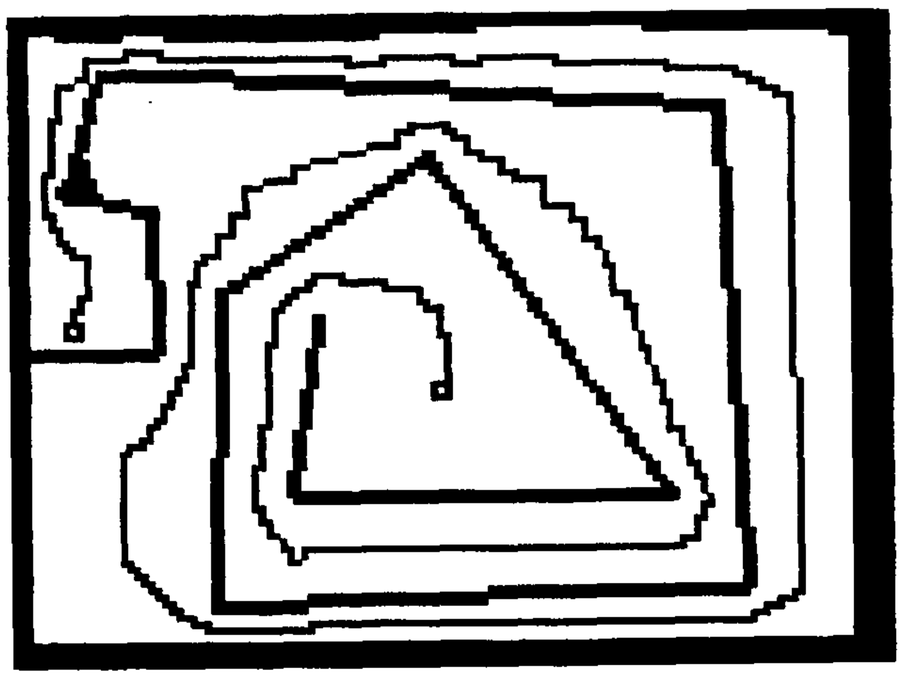
Some additional results from Rutar in different maps.

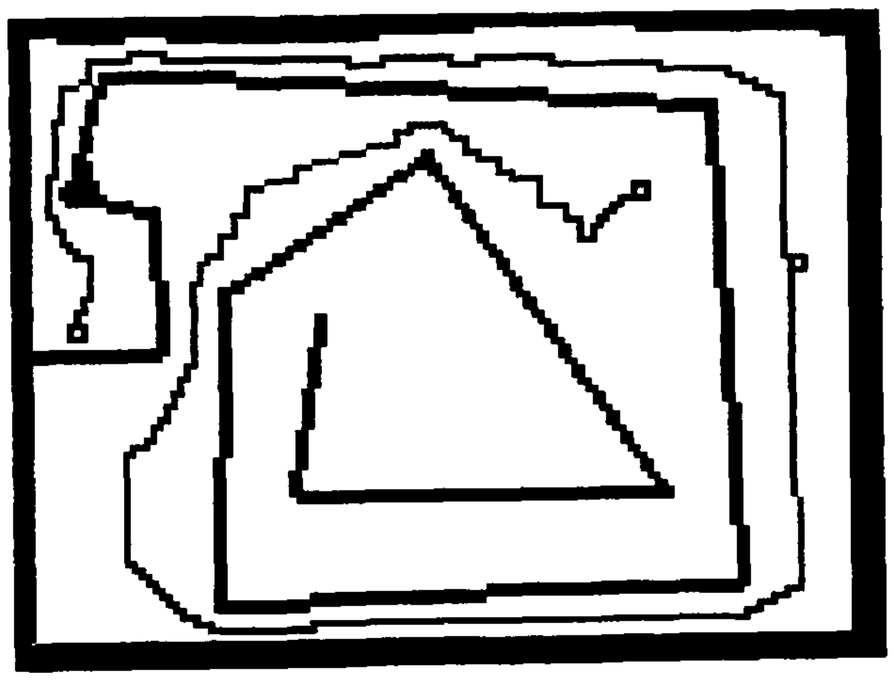


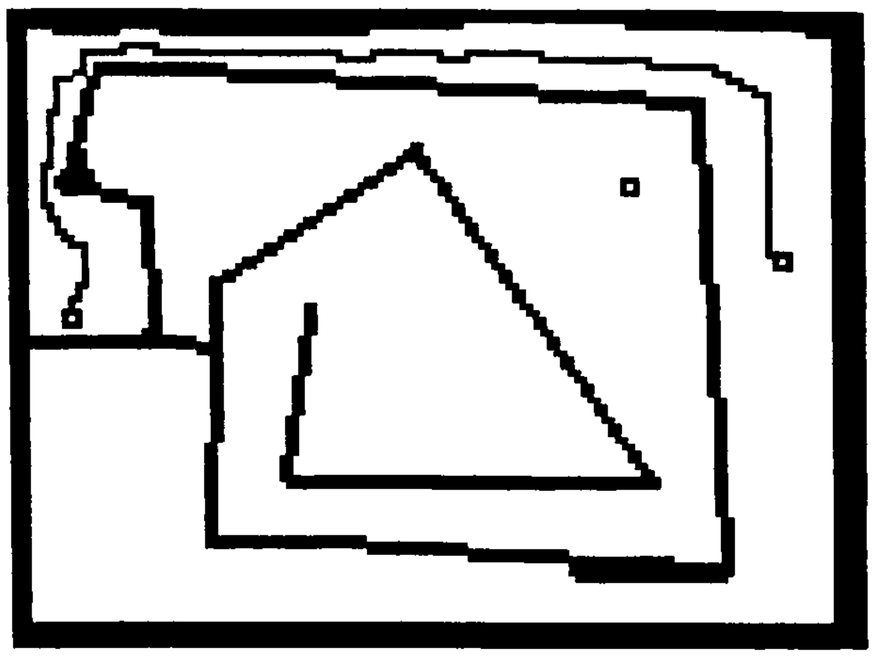


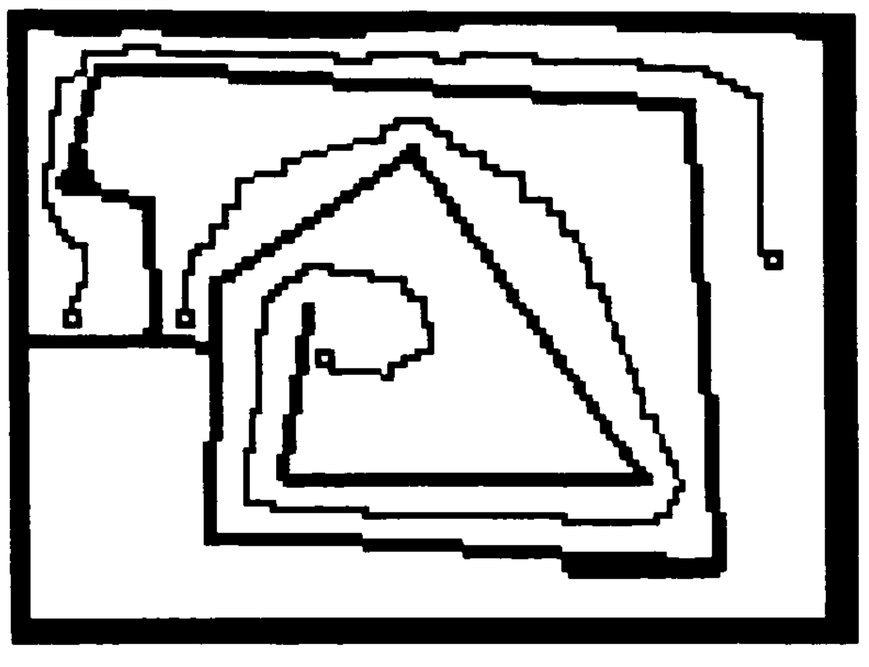


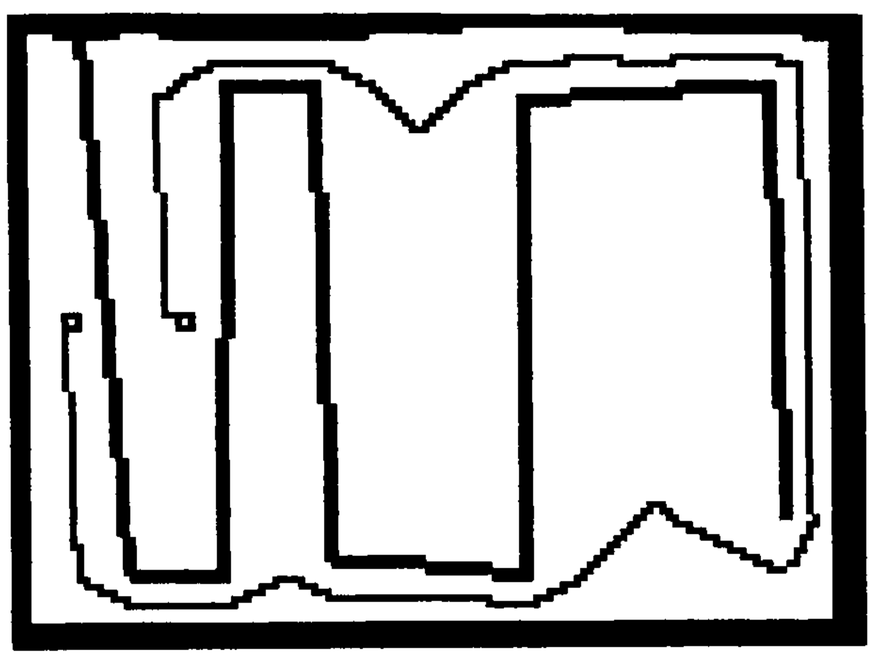


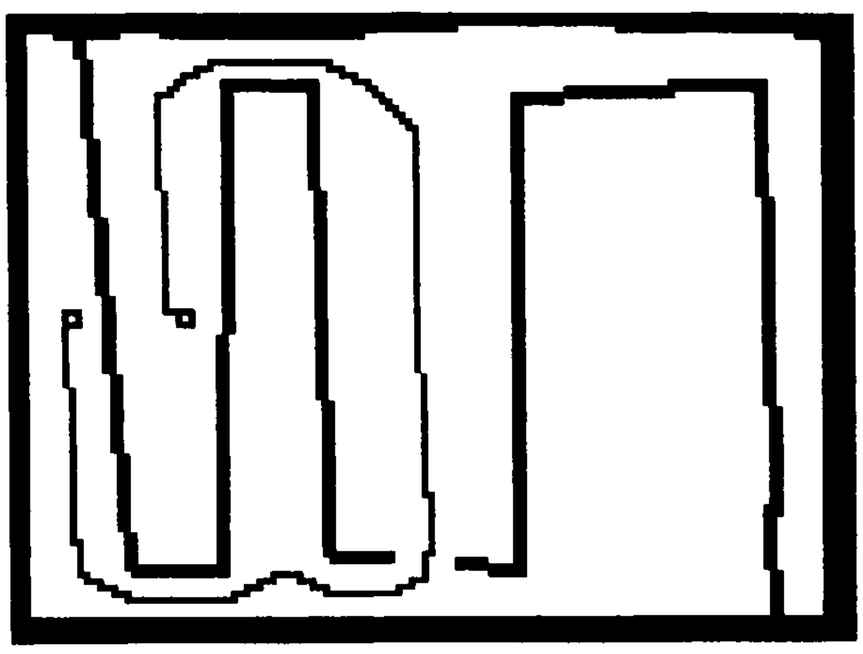


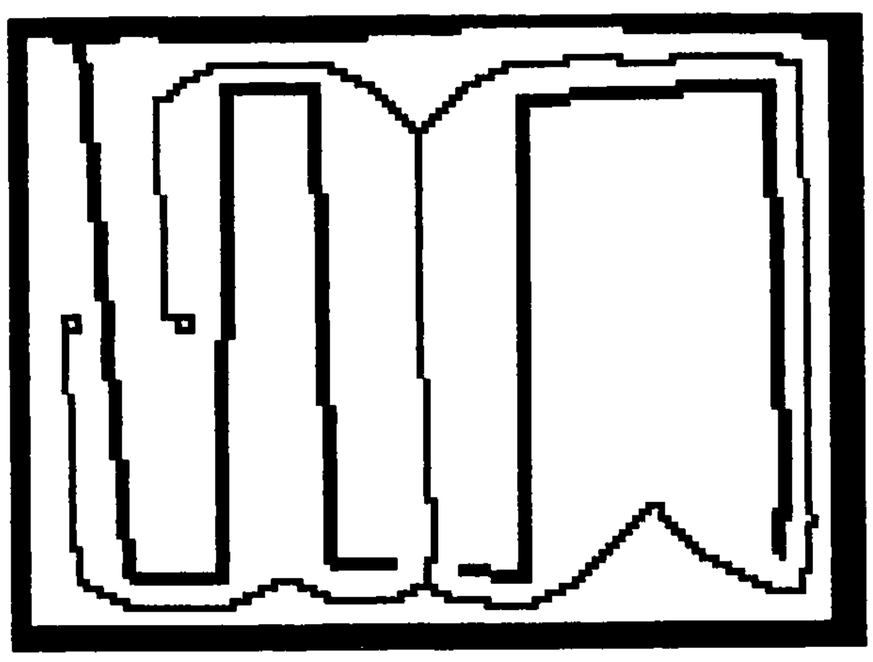


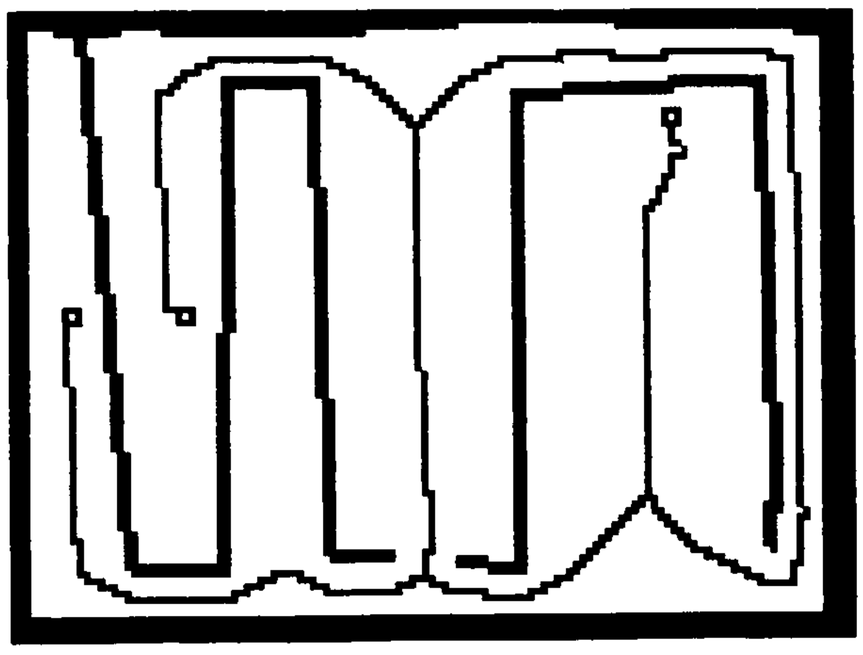


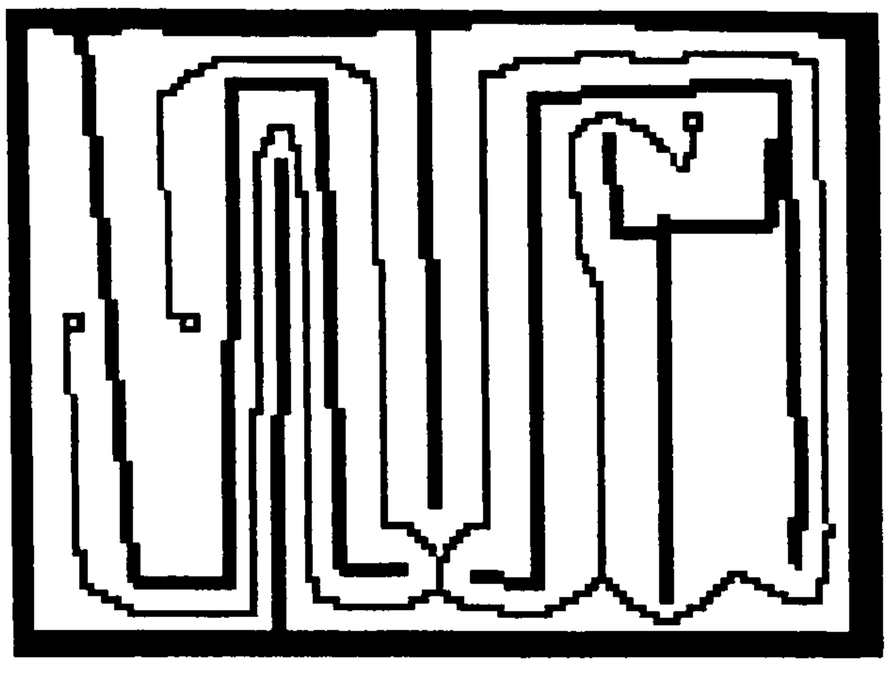


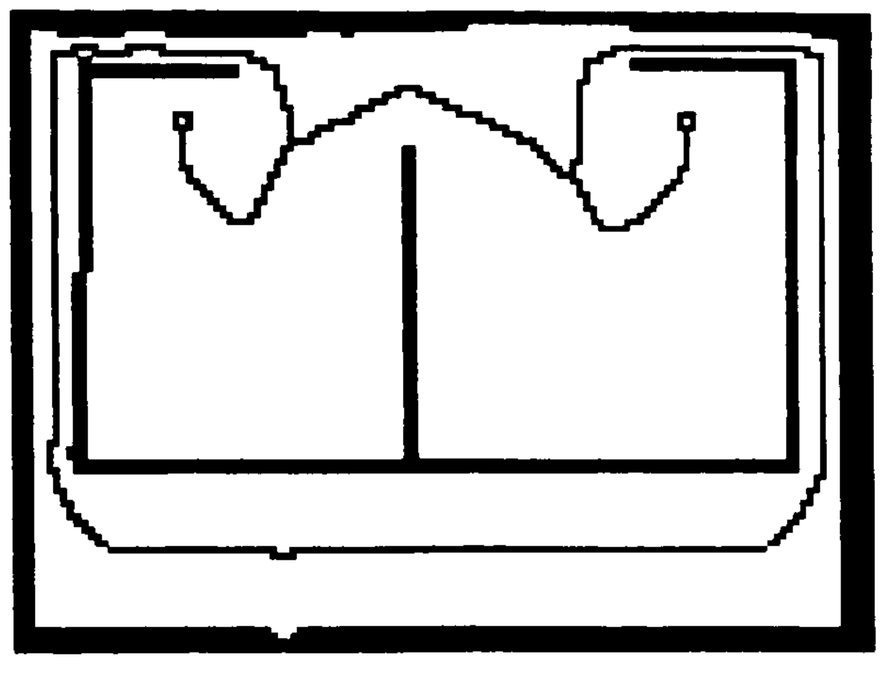


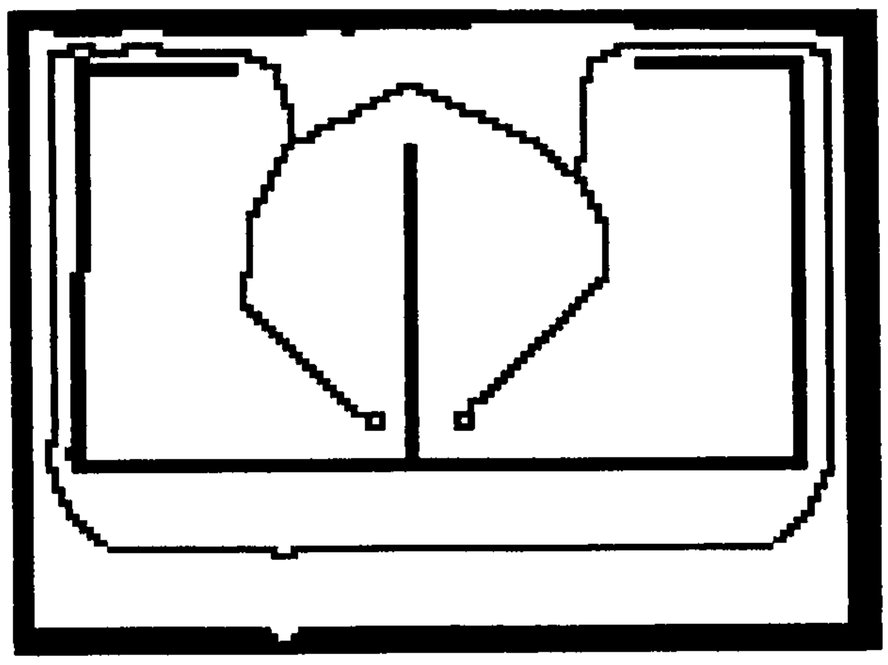


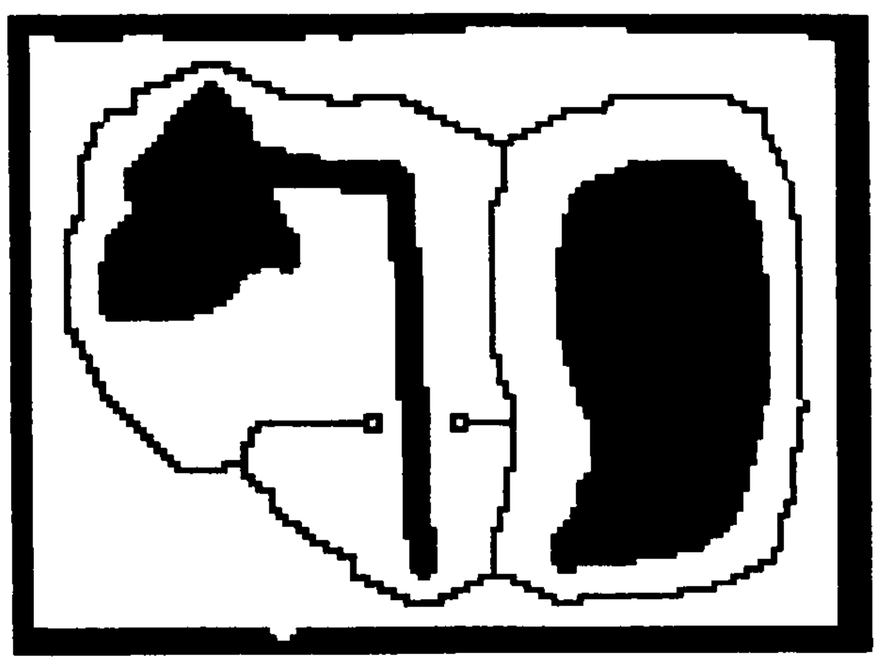


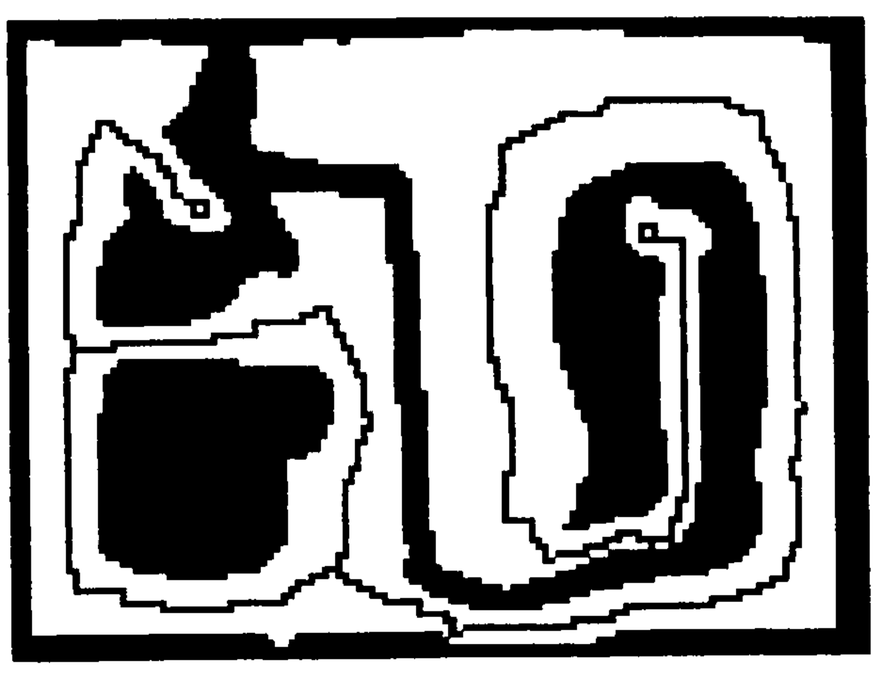












Appendix B

Rutar executable in disk

The disk contains two programs, which are versions of Rutar for MS-DOS:

Rutar.exe

Is a graphical slow motion version of Rutar. It executes in a DOS window (it can be double-clicked from windows for an automatic MS-DOS window to appear, or executed at the DOS prompt).

Rutar1.exe

Is the same version as above, with pauses every other cycle of life. Any key will continue the execution. Each time it pauses it saves the intermediate image in a file called *Temp01.rut* (this file can be viewed using the utility from the web site).

The program prompts for the input map: *Archivo del mapa ?:*
The input file must be the name of a valid map, such as the *mapxxx.rut* files in the disk. The filename must be written in full, i.e. including the extension (Rutar does not assumes a default extension).

It then prompts for the output file: *Archivo de salida ?:*
Any filename can be used, except the same input filename. Rutar will overwrite any file of the same name!

The program terminates with a key-press after it has finished the generation.

The disk contains some example maps (*mapxxx.rut*). More maps can be downloaded from the web site, as well as a Windows 95 utility to convert maps to and from

standard bitmap format so that customised maps can be used.

The web site address for Rutar is:

http://venus.javeriana.edu.co/~mgongora/Path/Path.html

Alternatively, if this direct address does not work, this page can be found in the main page of the Department of Engineering:

http://venus.javeriana.edu.co

following the links for Research Projects, Mario Gongora, Path Planning.

At this site the executables for Unix and non-graphical form for DOS can be found, as well as additional maps, results examples, the utility to convert the map and route files to and from standard windows bitmap (*.bmp) format and some more information.

Appendix C

Structure of the optimisation genetic algorithm

The genetic algorithm used to perform the optimisation utilises some input and outputs modules that are described in this appendix.

The input module prompts the user for the file name for the map and the position of the robot including its orientation. It also prompts for the maximum allowable travel time and maximum available energy. Finally it prompts for the percentage to use for each optimisation criteria (Time, Energy, Risk and Distance). A screen dump of the input module is shown bellow.

```
MODULO DE ENTRADA
-------------------------------------------------------------------------
  Nombre del archivo que contiene el mapa:

  Xm:      pixeles    Ym:      pixeles    Dp:       mm

  Xo:      pixeles           Xf:      pixeles
  Yo:      pixeles           Yf:      pixeles
  Ao:       grados           Af:       grados

-------------------------------------------------------------------------
  Tiempo máximo de recorrido :       segundos
  Energía disponible         :       julios

-------------------------------------------------------------------------
  Divisor para Mínimo Tiempo:        Divisor para Mínima Energía    :
  Divisor para Mínimo Riesgo:        Divisor para Mínima Trayectoria:
```

The parameters that define the robot's characteristics and the genetic algorithm termination criteria are prompted in another module. This module reads from the user the dimensions of the robot, some manoeuvrability and energy consumption characteristics. It also reads the genetic algorithm characteristics such as percentages of the genetic operators, number of individuals per generation, maximum number of

generations to execute, and minimum change in fitness considered to be an improvement. This characteristics remain constant (saved in a file) until changed by the user, so that many optimisations can be performed with the same set of data (while the input module is executed every time the optimisation process is executed). The screen dump of the prompt is shown bellow.

```
MODULO DE PARAMETROS ------------------------------------------
  Dimensiones físicas del robot
    Ancho:       cms   Radio frontal:       cms   Radio posterior:       cms
  Velocidades y aceleraciones del robot
    Velocidad angular máxima:     grados/s
    Velocidad frontal máxima:     cms/s    Factor de velocidad (K):
    Angulo critico 1        :     grados   Angulo critico 2        :     grados
    Tiempo de aceleración en giro:     seg.   y desaceleración:     seg.
    Tiempo de aceleración frontal:     seg.   y desaceleración:     seg.
  Energías relacionadas con el robot
    Energía a velocidad cte. frontal:      julios/s  y en giro:      julios/s
    Máximo pico de energía acelerando frontal:      j  y valor final:      j
    Máximo pico de energía desaceler. frontal:      j  y valor final:      j
    Máximo pico de energía acelerando en giro:      j  y valor final:      j
    Máximo pico de energía desaceler. en giro:      j  y valor final:      j
    Tau acelerando frontal:       seg.     y desacelerando:       seg.
    Tau acelerando en giro:       seg.     y desacelerando:       seg.
    Energía mínima permitida para el robot:      julios
  Avanzado
    # máximo de generaciones      :       Porcentaje mínimo de cambio:
    # de individuos por generación:       # de caminos a optimizar   :
    # pixeles de ensanch.:      Prob. de crossover:      Prob. de mutar:
```

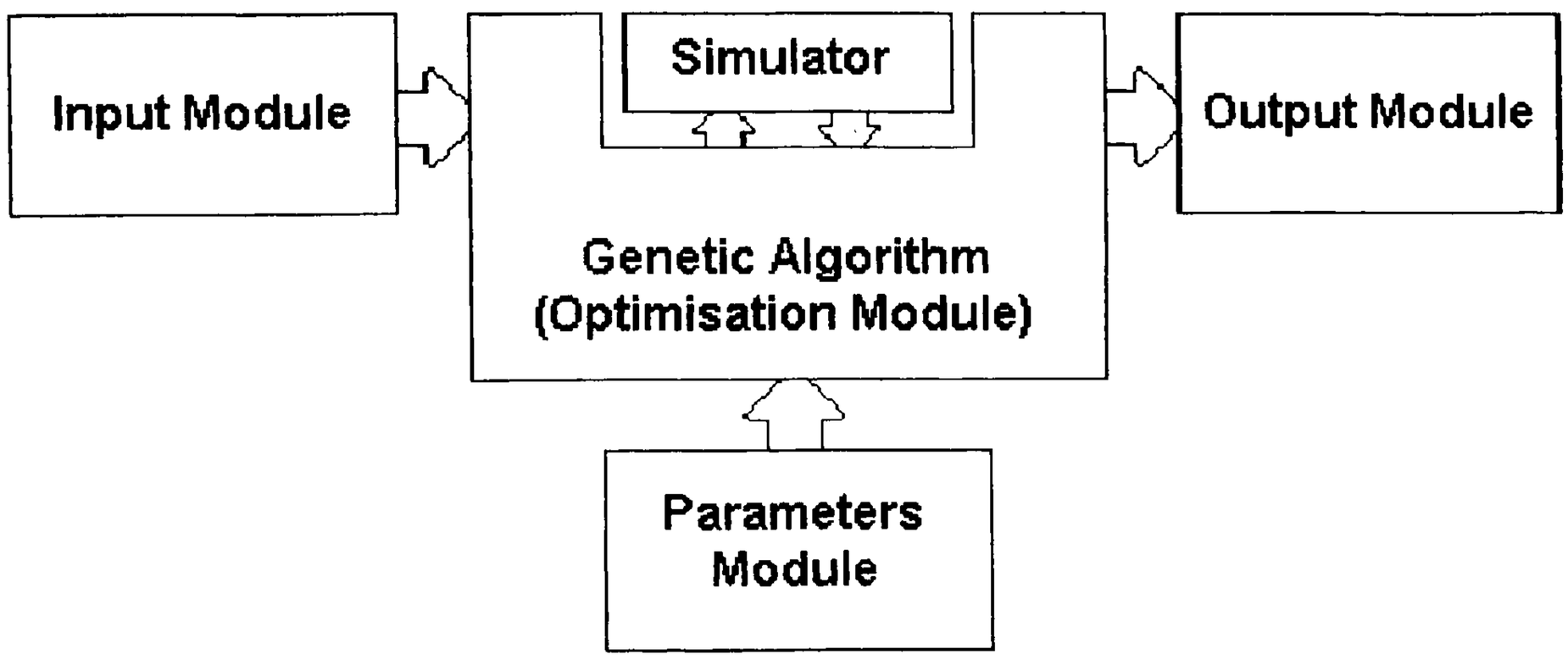
The output is provided by the system as a map (shown in Appendix E) saved in a file and the numerical data of the criteria, shown in the output module bellow. This module shows the energy consumed, time and risk taken, and distance travelled.

```
MODULO DE SALIDA
-------------------------------------------------------------------------
  La ruta hallada por el sistema entrega los siguientes resultados:

      Energía consumida : 330.850006 julios
      Energía restante  : 2669.139893 julios
      Tiempo requerido  : 13.51 segundos
      Longitud total    : 564.98999 cms
-------------------------------------------------------------------------
  Los valores de FITNESS de la ruta son:

      Según el criterio de Mínimo Tiempo     : 96.919998
      Según el criterio de Mínima Energía    : 87.269997
      Según el criterio de Mínimo Riesgo     : 59.509998
      Según el criterio de Mínima Trayectoria: 86.669998
      El Factor de Penalización es           : 1

      El FITNESS TOTAL es                    : 271.470001

-------------------------------------------------------------------------
< M-Ver Mapa >     < C-Ver Comandos >     < S-Salir >
```

The following diagram shows the structure of the previous modules in the optimisation algorithm.

Defines the data for each execution (criteria, maximum available energy, etc).

Presents the output data to the user for analysis, also saves the map containing the optimal path.



Defines general and not very volatile data for the optimisation process and the simulator.
For the simulator: Robot characteristics.
For the Genetic Algorithm: Termination criteria, number of individuals per generation, fitness evaluation method, genetic operands percentages, etc.

Appendix D

Simulator for optimisation tests

The optimisation system was tested using a simulator for a hypothetical robot. The simulator was designed to run fast in the computer and to simulate various typical characteristics of robots. It simulates a differentially steered three wheeled robot capable of spinning over a fixed point. Some basic characteristics of the robot can be defined in the parameters module so that variations can be tested in the optimisations processes.

This simulator is included as a <include.h> file. The simulator program is listed below.

```
/*
                                                Modulo del SIMULADOR

Este modulo esta conformado como un archivo que se incluye dentro de
el
modulo de optimizacion y es el destinado para la simulacion de la ruta
propuesta dentro del mapa actual.
Existen unas variables globales que dicho modulo requiere que
corresponden
a valores relacionados con el mapa, la ruta y las caracteristicas del
robot.
Estas variables son:
      gmapa[128][128]   = Arreglo que contiene el mapa
      gdp                              = Diametro de cada pixel en cms.
      gruta[50]                  = Arreglo de tipo punt que contiene los
puntos de la ruta
      gn                                       = Numero de puntos que
conforma la ruta (de 0 a gn)
      grf,gra             = Radio frontal y posterior del robot
      gvg,gtag,gtdg       = Velocidad m x. de giro, tiempo de de acel. y
desac. en giro
      gvl,gtal,gtdl       = Velocidad m x. frontal, tiempo de de acel. y
desac. frontal
      gk,gal1,gal2        = Factor de velocidad, angulo critico 1 y 2
      gevl                = Energia (en julios/segundo) a velocidad
m xima frontal
      geal1,geal2,gtual = Picos de energia inicial y final y tau
aceler. frontal
      gedl1,gedl2,gtudl = Picos de energia inicial y final y tau
desace. frontal
      gevg                = Energia (en julios/segundo) a velocidad
m xima en giro
      geag1,geag2,gtuag = Picos de energia inicial y final y tau
aceler. en giro
      gedg1,gedg2,gtudg = Picos de energia inicial y final y tau
desace. en giro
*/

/*INCLUDES*/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>


/*DECLARACION DE VARIABLES GLOBALES*/
struct punt {              /*estructura de los puntos de la ruta*/
      int x,y;                    /*coordenada del punto*/
      float a,av;                 /*angulo respecto a la horizontal y
variacion del angulo*/
};
unsigned char gmapa[128][128];      /*arreglo con el mapa//DE 50 A
128*/
struct punt gruta[50];                                  /*arreglo de
puntos de la ruta//30 X 50*/
int gn,gal1,gal2;                                       /*numero de puntos y
angulos criticos*/
float pi=3.1415926,gdp;                                 /*valor de pi y diametro de
cada pixel*/
float
grf,gra,gvg,gtag,gtdg,gvl,gtal,gtdl,gk,gevl,geal1,geal2,gtual,gedl1
```

```
      ,gedl2,gtudl,gevg,geag1,geag2,gtuag,gedg1,gedg2,gtudg;


/*DECLARACION DE FUNCIONES*/

float cuadratica (float a, float b, float c);
void evalua_giro (float gxag ,float gxdg ,float gav ,float * gtc,
float * gec);
void halla_Pi (int gxin, int gyin, int gxfi, int gyfi, float gm,
      int ginv    , int * gp1, int * gp2, int * gp3, int * gp4, int *
gp5);
void halla_G1G2 (int grp, float gap, float gdesfase, int * gg1, int *
gg2);
void halla_pixeles (int * gpc, int * gpco, int * gr1, int * gr2, int *
gr3);
void halla_giros (int * ggc, int * ggco);
void halla_tiempo_energia (float * gtc, float * gec);
void simulador (int * spc, int * spco, int * sr1, int * sr2
      ,int * sr3, int * sgc, int * sgco, float * stc, float * sec);


/*DEFINICION DE FUNCIONES*/

/*saca la raiz mas cercana a cero de la funcion cuadratica*/
float cuadratica (float a, float b, float c)
{
      float f,xa,xb,raiz;           /*variables auxiliares y de la
raiz*/

      f=(pow(b,2)-(4*a*c));   /*saca el radical*/
      if (f>0)                                    /*evalua si es mayor
que cero*/
      {                                                 /*la raiz es de
tipo real*/
            f=sqrt(f);
            xa=(-b+f)/(2*a);     /*evalua la primera raiz*/
            xb=(-b-f)/(2*a);        /*evalua la segunda raiz*/
      }
      else                                        /*calcula la parte
real de la raiz compleja*/
      {
            xa=xb=(-b/(2*a));       /*evalua las dos raices (reales)*/
      }
      raiz=xa;                                    /*asigna una de ellas
al resultado*/
      if (((xa<0)&&(xb>=0))||((xb>=0)&&(xb<xa))) raiz=xb;
      /*reemplaza por otra*/
      if ((xa<0)&&(xb<0)&&(xb>xa)) raiz=xb;     /*reemplaza por otra*/
      return raiz;                          /*devuelve la raiz*/
}

/*evalua el tiempo requerido y la energia consumida en un giro sobre
el
eje de giro del robot*/
void evalua_giro (float gxag ,float gxdg ,float gav ,float * gtc,
float * gec)
{

      float t2,t1;                                /*variables
auxiliares*/

      if (fabs(gav)>(gxag+gxdg))    /*evalua si el giro es mayor que
el necesario*/
      {                                           /*para
acelerar y desacelerar en giro*/
            *gtc = (*gtc + gtag + gtdg + (abs(gav)-(gxag+gxdg))/gvg);
      /*tiempo*/
```

```
                *gec = (*gec + geag2*gtag + gtuag*(geag1-geag2)*(1-exp(-
gtag/gtuag)));     /*energia1*/
                *gec = (*gec + gedg2*gtdg + gtudg*(gedg1-gedg2)*(1-exp(-
gtdg/gtudg)));     /*energia2*/
                *gec = (*gec + gevg*(abs(gav)-(gxag+gxdg))/gvg);
                      /*energia3*/
        }
        else                                                    /*si el giro es
menor que el necesario*/
        {
                t2 = sqrt(abs(gav)/(gvg*(1+gtag/gtdg)/(2*gtdg)));
        /*tiempo de aceleracion*/
                *gtc = (*gtc + t2 + t2*gtag/gtdg);
                /*tiempo*/
                *gec = (*gec + gedg2*t2 + gtudg*(gedg1-gedg2)*(1-exp(-
t2/gtudg)));       /*energia1*/
                t1 = t2*gtag/gtdg;
        /*tiempo de desaceleracion*/
                *gec = (*gec + geag2*t1 + gtuag*(geag1-geag2)*(1-exp(-
t1/gtuag)));      /*energia2*/
        }
}

/*halla el numero de pixeles entre dos puntos de la trayectoria, el #
de
pixeles que son obstaculo, # de pixeles de riesgo bajo, medio y alto*/
void halla_Pi (int gxin, int gyin, int gxfi, int gyfi, float gm,
        int ginv    , int * gp1, int * gp2, int * gp3, int * gp4, int *
gp5)
{
        int j,dx,xe,ye;
        /*variables auxiliares*/
        int pixel;
        /*valor del pixel*/

        *gp1=0;*gp2=0;*gp3=0;*gp4=0;*gp5=0; /*pone los valores en cero*/
        if (gxfi<gxin) j=-1;            /*evalua si la variable indep. se
incrementa*/
        else j=1;
        dx=0;
        /*inicializa el delta x*/
        do                                                      /*loop de hallar
pixeles entre los puntos*/
        {
                dx=dx+j;
        /*incrementa el delta x*/
                ye=gyin+ceil(dx*gm-0.5);                /*halla Y del
siguiente pixel*/
                xe=gxin+dx;
        /*halla X del siguiente pixel*/
                if (ginv==1) pixel=gmapa[ye][xe];/*asigna el pixel a la
variable pixel*/
                else pixel=gmapa[xe][ye];
                *gp1=*gp1+1;
        /*incrementa el #pixeles*/
                switch (pixel) {                                /*evalua el tipo de
pixel*/
                        case 1:     *gp2=*gp2+1;
        /*incrementa #pixeles obstaculo*/
                                        break;
                        case 2:     *gp3=*gp3+1;
        /*incrementa #pixeles riesgo bajo*/
                                        break;
                        case 3:     *gp4=*gp4+1;
        /*incrementa #pixeles riesgo medio*/
                                        break;
                        case 4:     *gp5=*gp5+1;
```

```
        /*incrementa #pixeles riesgo alto*/
                                        break;
                }
        }
        while (!((ye==gyfi)&&(xe==gxfi)));  /*evalua fin de loop*/
}

/*halla el numero de giros que una ruta tiene y cuantos de estos son
no validos*/
void halla_G1G2 (int grp, float gap, float gdesfase, int * gg1, int *
gg2)
{
        int i,inv,n,xin,yin,x1,y1,xfi,yfi,xx,p1,p2,p3,p4,p5;
        /*variables*/
        float m,deltay,deltax,gapp;
        /*variables*/

        i=-1;*gg1=0;*gg2=0;inv=0;                    /*inicializacion de
valores*/
        do
        /*loop que va por cada punto de la ruta*/
        {
                i++;
        /*incrementa el punto de la ruta*/
                gapp=gap;                                   /*asigna
el delta angulo de evaluacion*/
                if (gruta[i+1].a<gruta[i].a) gapp=-gapp;  /*lo invierte si
es necesario*/
                xin=gruta[i].x;yin=gruta[i].y;/*determina el punto de
giro*/
                x1=ceil(cos((gruta[i+1].a-gdesfase)*pi/180)*grp-0.5);
        /*final del*/
                y1=ceil(sin((gruta[i+1].a-gdesfase)*pi/180)*grp-0.5);
        /*giro*/
                x1=x1+xin;y1=y1+yin;                    /*determina el
punto final para el arco*/
                n=-1;
        /*inicializa el valor del delta angulo*/
                do
        /*loop que evalua cada radio del arco*/
                {
                        n++;inv=0;
        /*incrementa el delta angulo*/
                        xfi=xin+ceil(cos(((gruta[i].a-
gdesfase)*pi/180)+((n*gapp*pi)/180))*grp-0.5);
                        yfi=yin+ceil(sin(((gruta[i].a-
gdesfase)*pi/180)+((n*gapp*pi)/180))*grp-0.5);
                        if (xin!=xfi)                       /*prevee
una division por cero*/
                        {
                                deltay=yfi-yin;
                                deltax=xfi-xin;
                                m=(deltay/deltax);          /*calcula la
pendiente del vector*/
                        }
                        else m=20;                          /*si
deltax=0 da un valor mayor a 1*/
                        if (fabs(m)>1)                      /*si la
abs(pendiente) es menor a 1*/
                        {
                                if (xfi==xin) m=0;      /*determina m=0 si
deltax=0*/
                                else m=1/m;                 /*sino
invierte a m*/
                                xx=xin;xin=yin;yin=xx;  /*invierte las
coordenadas*/
                                xx=xfi;xfi=yfi;yfi=xx;
```

```
                        inv=1;
      /*asigna true a flag de inversion*/
                  }
                  halla_Pi (xin,yin,xfi,yfi,m,inv,&p1, &p2, &p3, &p4,
&p5); /*halla P2*/
                  if (inv==1)                                    /*si se
habia invertido, se vuelve*/
                  {
      /*a invertir*/
                        xx=xin;xin=yin;yin=xx;
                        xx=xfi;xfi=yfi;yfi=xx;
                  }
            }
            while ((p2==0)&&((xfi!=x1)||(yfi!=y1)));  /*evalua fin del
loop*/
            if (p2>0) *gg2=*gg2+1;                  /*si hay pixeles
obstaculo se cuenta*/
                  *gg1=*gg1+1;                                    /*se
incrementa el # de giros totales*/
      }
      while (i<gn-1);                                 /*evalua fin de
loop por puntos*/
}

/*halla el numero de pixeles que conforma la trayectoria y cuantos de
estos son obstaculo, de riesgo bajo, medio y alto*/
void halla_pixeles (int * gpc, int * gpco, int * gr1, int * gr2, int *
gr3)
{
      int i,xin,yin,xfi,yfi,inv,xx,p1,p2,p3,p4,p5;    /*variables*/
      float deltay,deltax,m;                  /*variables para la
pendiente*/

      i=0;                                    /*inicializa el indice del
punto*/
      *gpc=1;*gpco=0;
      /*inicializa #de pixeles y obst.*/
      *gr1=0;*gr2=0;*gr3=0;                           /*inicializa
riesgos*/
      do
            /*loop que evalua tramos entre todos*/
      {
            /*los puntos consecutivos de la ruta*/
            i++;inv=0;
      /*incrementa el indice del punto*/
            xin=gruta[i-1].x;yin=gruta[i-1].y;  /*toma la coordenada
inicial*/
            xfi=gruta[i].x;yfi=gruta[i].y;            /*toma la
coordenada final*/
            if (xin!=xfi)
      /*prevee una division por cero*/
            {
                  deltay=yfi-yin;
                  deltax=xfi-xin;
                  m=(deltay/deltax);                        /*calcula
la pendiente del vector*/
            }
            else m=20;                                    /*si
deltax=0 da un valor mayor a 1*/
            if (fabs(m)>1)                                /*si
la abs(pendiente) es mayor a 1*/
            {                                   /*se hace x=f(y)*/
                  if (xfi==xin) m=0;                /*determina m=0
si deltax=0*/
                  else m=1/m;
      /*sino invierte a m*/
                  xx=xin;xin=yin;yin=xx;                  /*invierte las
```

```
coordenadas*/
                    xx=xfi;xfi=yfi;yfi=xx;
                    inv=1;
      /*asigna true a flag de inversion*/
            }
            if (!((xin==xfi)&&(yin==yfi)))
            {
                  halla_Pi (xin,yin,xfi,yfi,m,inv,&p1, &p2, &p3, &p4,
&p5);/*halla pixeles entre 2 puntos*/
                  *gpc=*gpc+p1;*gpco=*gpco+p2;        /*incrementa los
valores segun el res.*/
                  *gr1=*gr1+p3;*gr2=*gr2+p4;*gr3=*gr3+p5;
            }
      }
      while (i<gn-1);                                          /*valida
fin del loop (todos los puntos)*/
}

/*halla el numero de giros que hay en una ruta y cuantos de ellos
son no posibles*/
void halla_giros (int * ggc, int * ggco)
{
      int rfp,rap,i,rp,g1,g2;
            /*variables*/
      float af,aa,ap,desfase,angulo,deltay,deltax;   /*variables*/

      rfp=ceil(grf/gdp);                              /*calcula #pixeles del
radio frontal*/
      af=(360/(2*pi*ceil(grf/gdp)));      /*calcula el delta angulo
para Rf*/
      rap=ceil(gra/gdp);                              /*calcula #pixeles del
radio posterior*/
      aa=(360/(2*pi*ceil(gra/gdp)));      /*calcula el delta angulo
para Ra*/
      *ggc=0;*ggco=0;                                       /*inicializa
#giros y #giros invalidos*/
      i=0;
      /*inicializa indice de puntos*/
      do
      /*loop para calcular los angulos*/
      {
            i++;
      /*incrementa indice*/
            deltay=(gruta[i].y-gruta[i-1].y);deltax=(gruta[i].x-
gruta[i-1].x);    /*deltas*/
            if (deltax==0)                                    /*prevee
division por cero*/
            {
                  if (deltay>0) angulo=90;
                  else angulo=-90;
            }
            else angulo=atan(deltay/deltax)*180/pi;
            if (deltax>=0) gruta[i].a=angulo;   /*determina el angulo
de cada punto*/
            else
            {
                  if (deltay>0) gruta[i].a=180+angulo;
                  else gruta[i].a=angulo-180;
      }
      angulo=(gruta[i].a-gruta[i-1].a);
      if (fabs(angulo)<=180) gruta[i-1].av=angulo;
/*variacion de angulos*/
      else
      {
            if (angulo<-180) gruta[i-1].av=(360+angulo);
            else gruta[i-1].av=(angulo-360);
      }
```

```
        }
        while (i<gn-1);
        /*fin de loop que halla angulos*/
        i=gn;                                           /*indice del ultimo
punto*/
        angulo=(gruta[i].a-gruta[i-1].a);
        if (fabs(angulo)<=180) gruta[i-1].av=angulo;    /*ultima
variacion de angulo*/
        else
        {
              if (angulo<-180) gruta[i-1].av=(360+angulo);
              else gruta[i-1].av=(angulo-360);
        }
        desfase=0;rp=rfp;ap=af;
        /*asigna valores de Rf y Af*/
        halla_G1G2 ( rp, ap, desfase, &g1, &g2);  /*evalua choques por
delante */
        *ggc=g1;*ggco=g2;
        /*asigna resultados*/
        desfase=180;rp=rap;ap=aa;
        /*asigna valores de Ra y Aa*/
        halla_G1G2 ( rp, ap, desfase, &g1, &g2);  /*evalua choques por
atras   */
        if (g2>*ggco) *ggco=g2;
        /*reemplaza #giros invalidos   */
}

/*halla el tiempo y la energi arequerida por el robot para seguir la
ruta actual*/
void halla_tiempo_energia (float * gtc, float * gec)
{
        int i,normal=0;
        float va,ve,xag,xdg,xe,xa,xb,t1,t2,a,b,c,ee,ef,htc=0,hec=0;

        i=0;va=0;*gtc=0;*gec=0;                         /*inicializa
variables*/
        xag=(gvg*gtag/2);xdg=(gvg*gtdg/2);  /*distancia en giro para
acel. y des.*/
        evalua_giro (xag,xdg,gruta[i].av,&htc,&hec);
        *gtc=*gtc+htc;*gec=*gec+hec;              /*energia y tiempo de
giro inicial*/
        do
        /*loop que evalua cada tramo de la ruta*/
        {
              i++;
        /*incrementa el indice de la ruta*/
              xe=(gdp*sqrt(pow((gruta[i].x-gruta[i-
1].x),2)+pow((gruta[i].y-gruta[i-1].y),2)));/*distancia a recorrer*/
              if (fabs(gruta[i].av)<gal1)/*evalua si variacion de angulo
< angulo1*/
              {
                    ve=gvl;                                 /*asigna
maxima velocidad*/
                    normal=1;                             /*flag de tramo
normal*/
              }
              else
              {
                    if
((fabs(gruta[i].av)>=gal1)&&(fabs(gruta[i].av)<gal2))
                    {
        /*evalua si angulo1 < angulo < angulo2*/
                          ve=gvl*gk*((180-fabs(gruta[i].av))/(180-
gal1));     /*velocidad < Vl*/
                          normal=1;                       /*flag de tramo
normal*/
                    }
```

```
                else normal=0;                                  /*flag de tramo
con frenada*/
            }
            if (normal==1)                                      /*tramo normal*/
            {
                if (va<=ve)                                     /*evalua si
tiene que acelerar*/
                {
                    t1=((ve-va)*gtal/gvl);  /*calcula el tiempo de
acelerada*/
                    xa=(va*t1 + (gvl*pow(t1,2))/(gtal*2));
      /*tramo acelerando*/
                    ee=(geal2*t1 + gtual*(geal1-geal2)*(1-exp(-
t1/gtual)));       /*energia*/
                }
                else                                                /*si
tiene que desacelerar*/
                {
                    t1=((va-ve)*gtdl/gvl);  /*tiempo de la
desacelerada */
                    xa=(va*t1 - (gvl*pow(t1,2))/(gtdl*2));
      /*tramo desacelerando*/
                    ee=(gedl2*t1 + gtudl*(gedl1-gedl2)*(1-exp(-
t1/gtudl)));       /*energia*/
                }
                if (xa<=xe)             /*evalua si tramo total es
mayor que la acelerada*/
                {
                    *gtc=*gtc + t1 + (xe-xa)/ve;
      /*suma tiempos*/
                    *gec=*gec + ee + gevl*(xe-xa)/gvl;  /*suma
energias*/
                }
                else                /*si tramo total es menor a la
acelerada*/
                {
                    if (va<=ve)       /*calcula si se acelera o
desacelera*/
                    {
                        a=(gvl/(gtal*2));b=va;c=-xe;  /*indices
de cuadratica*/
                        t1=cuadratica(a,b,c);
      /*halla tiempo*/
                        ve=va + (gvl*t1/gtal);
      /*halla velocidad*/
                        ee=(geal2*t1 + gtual*(geal1-geal2)*(1-
exp(-t1/gtual))); /*energia*/
                    }
                    else                                /*si se desacelera*/
                    {
                        a=(-gvl/(gtdl*2));b=va;c=-xe; /*indices
de cuadratica*/
                        t1=cuadratica(a,b,c);
      /*halla tiempo*/
                        ve=va - (gvl*t1/gtdl);
      /*halla velocidad*/
                        ee=(gedl2*t1 + gtudl*(gedl1-gedl2)*(1-
exp(-t1/gtudl))); /*energia*/
                    }
                    *gtc=*gtc + t1;
      /*suma tiempo*/
                    *gec=*gec + ee;
      /*suma energia */
                }
                va=ve;                                          /*actualiza valor de
la velocidad que lleva*/
            }
```

```
            else                                        /*si no es tramo
normal ( frena al final )*/
                {
                    t1=(gvl-va)*gtal/gvl;   /*tiempo de aceleracion a
Vl*/
                    xa=va*t1 + (gvl*pow(t1,2)/(gtal*2));xb=gvl*gtdl/2;
      /*tramo de acel. y des.*/
                    ee=(geal2*t1 + gtual*(geal1-geal2)*(1-exp(-
t1/gtual)));       /*energia acel.*/
                    ef=(gedl2*gtdl + gtudl*(gedl1-gedl2)*(1-exp(-
gtdl/gtudl)));     /*energia des.*/
                    if (xe>=(xa+xb))                /*si tramo total es
mayor a acel. y desa.*/
                    {
                        *gtc=*gtc + t1 + gtdl + (xe-(xa+xb))/gvl;
      /*suma tiempos*/
                        *gec=*gec + ee + ef;
                        /*suma energias*/
                        *gec=*gec + gevl*(xe-(xa+xb))/gvl;
                    }
                    else                                    /*si tramo
total es menor a acel. y desa.*/
                    {
                        t1=va*gtdl/gvl;          /*calcula tiempo de
frenada*/
                        xa=va*t1 - gvl*pow(t1,2)/(gtdl*2);  /*tramo de
frenada*/
                        ee=(gedl2*t1 + gtudl*(gedl1-gedl2)*(1-exp(-
t1/gtudl)));       /*energia*/
                        if (xe>xa)                      /*si tramo total
es mayor a frenada*/
                        {
                            a=(gvl/gtal)*(0.5 + gtdl/(2*gtal));
      /*indi.de cuadratica*/
                            b=va*(1+gtdl/gtal);
                            c=((pow(va,2)*gtdl)/(2*gvl)) - xe;
                            t1=cuadratica(a,b,c);
      /*tiempo de acelerada*/
                            t2=gtdl*((va/gvl) + (t1/gtal));
      /*tiempo de desacelerada*/
                            ee=(geal2*t1 + gtual*(geal1-geal2)*(1-
exp(-t1/gtual))); /*energias*/
                            ef=(gedl2*t2 + gtudl*(gedl1-gedl2)*(1-
exp(-t2/gtudl)));
                            *gtc=*gtc + t2;    /*suma tiempo2*/
                            *gec=*gec + ef;    /*suma energia*/
                        }
                        *gtc=*gtc + t1;            /*suma tiempo1*/
                        *gec=*gec + ee;            /*suma energia*/
                    }
                    htc=0;hec=0;                          /*evalua giro
sobre el eje*/
                    evalua_giro (xag,xdg,gruta[i].av,&htc,&hec);
                    *gtc=*gtc + htc;                  /*suma tiempo*/
                    *gec=*gec + hec;                  /*suma energia*/
                    va=0;
      /*actualiza velocidad que va a cero*/
                }
            }
            while (i<gn-1);          /*valida fin de loop de evaluacion de
tramos*/
}

/*simulador del robot segun la ruta a seguir y el entorno*/
void simulador (int * spc, int * spco, int * sr1, int * sr2, int * sr3
      , int * sgc, int * sgco, float * stc, float * sec)
{
```

```
      int bpc,bpco,br1,br2,br3,bgc,bgco;  /*variables que retornan los
valores*/
      float btc,bec;
      /*variables que retornan los valores*/

      halla_pixeles ( &bpc, &bpco, &br1, &br2, &br3); /*halla
pixeles*/
      halla_giros (&bgc, &bgco);
            /*halla giros*/
      halla_tiempo_energia (&btc,&bec);
                  /*halla tiempo y energia*/
      *spc=bpc;*spco=bpco;
      /*reemplaza valores para retornarlos*/
      *sr1=br1;*sr2=br2;*sr3=br3;
      *sgc=bgc;*sgco=bgco;
      *stc=btc;*sec=bec;
}
```

An example of the type of data that the final system would yield in Mitchi language is shown bellow:

```
G, 0, 0, 0;
P 1;
N 0.78540;
G 5, 10, 0, 0;
V 10;
G4, 5, 1, 1.57080;
P 1;
R;
V 5;
G4, 5, -1, -1.57080;
I 5,-3,3.14159;
G,2,-1,0;
Q;
U20;
P6;
```

The actual data that the simulator would predict for such a path is given in the following format:

```
/ G,0,0,0;
/ go: e=0.000000, xf=0.000000, yf=0.000000, teta=0.000000
/ P1;

/ stop: xf=1.000000
    0.0100,      0.0494,      0.0000
    0.0200,      0.0975,      0.0000
    0.0300,      0.1444,      0.0000
    0.0400,      0.1900,      0.0000
    0.0500,      0.2344,      0.0000
    0.0600,      0.2775,      0.0000
    0.0700,      0.3194,      0.0000
    0.0800,      0.3600,      0.0000
    0.0900,      0.3994,      0.0000
    0.1000,      0.4375,      0.0000
    0.1100,      0.4744,      0.0000
    0.1200,      0.5100,      0.0000
    0.1300,      0.5444,      0.0000
    0.1400,      0.5775,      0.0000
    0.1500,      0.6094,      0.0000
    0.1600,      0.6400,      0.0000
    0.1700,      0.6694,      0.0000
    0.1800,      0.6975,      0.0000
    0.1900,      0.7244,      0.0000
    0.2000,      0.7500,      0.0000
    0.2100,      0.7744,      0.0000
    0.2200,      0.7975,      0.0000
    0.2300,      0.8194,      0.0000
    0.2400,      0.8400,      0.0000
    0.2500,      0.8594,      0.0000
```

```
0.2600,     0.8775,     0.0000
0.2700,     0.8944,     0.0000
0.2800,     0.9100,     0.0000
0.2900,     0.9244,     0.0000
0.3000,     0.9375,     0.0000
0.3100,     0.9494,     0.0000
0.3200,     0.9600,     0.0000
0.3300,     0.9694,     0.0000
0.3400,     0.9775,     0.0000
0.3500,     0.9844,     0.0000
0.3600,     0.9900,     0.0000
0.3700,     0.9944,     0.0000
0.3800,     0.9975,     0.0000
0.3900,     0.9994,     0.0000
0.4000,     1.0000,     0.0000
/ N0.78540;

/ spin: teta=0.785400
/ G5,10,0,0;

/ go: e=5.000000, xf=10.000000, yf=0.000000, teta=0.000000
/ No puede andar: V = 0.000000
/ V10;
    0.4100,     1.0035,     0.0035
    0.4200,     1.0141,     0.0141
    0.4300,     1.0318,     0.0318
    0.4400,     1.0566,     0.0566
    0.4500,     1.0884,     0.0884
    0.4600,     1.1273,     0.1273
    0.4700,     1.1732,     0.1732
    0.4800,     1.2263,     0.2263
    0.4900,     1.2864,     0.2864
    0.5000,     1.3536,     0.3536
    0.5000,     1.3536,     0.3536
/ G4,5,1,1.57080;

/ go: e=4.000000, xf=5.000000, yf=1.000000, teta=1.570800
    0.5100,     1.4243,     0.4243
    0.5200,     1.4950,     0.4950
    0.5300,     1.5657,     0.5657
    0.5400,     1.6364,     0.6364
    0.5500,     1.7071,     0.7071
    0.5600,     1.7778,     0.7778
    0.5700,     1.8485,     0.8485
    0.5800,     1.9192,     0.9192
    0.5900,     1.9899,     0.9900
    0.6000,     2.0607,     1.0607
    0.6100,     2.1314,     1.1314
    0.6200,     2.2021,     1.2021
    0.6300,     2.2728,     1.2728
    0.6400,     2.3435,     1.3435
    0.6500,     2.4142,     1.4142
    0.6600,     2.4849,     1.4849
    0.6700,     2.5556,     1.5556
    0.6800,     2.6263,     1.6263
    0.6900,     2.6971,     1.6971
    0.7000,     2.7678,     1.7678
    0.7100,     2.8385,     1.8385
    0.7200,     2.9092,     1.9092
    0.7300,     2.9799,     1.9799
    0.7400,     3.0506,     2.0506
    0.7500,     3.1213,     2.1213
    0.7600,     3.1920,     2.1920
    0.7700,     3.2627,     2.2627
    0.7800,     3.3334,     2.3335
    0.7900,     3.4042,     2.4042
```

```
0.8000,      3.4749,      2.4749
0.8100,      3.5456,      2.5456
0.8200,      3.6163,      2.6163
0.8300,      3.6870,      2.6870
0.8400,      3.7577,      2.7577
0.8500,      3.8284,      2.8284
0.8600,      3.8991,      2.8991
0.8700,      3.9698,      2.9699
0.8800,      4.0406,      3.0406
0.8900,      4.1113,      3.1113
0.9000,      4.1820,      3.1820
0.9000,      4.1820,      3.1820
0.9000,      4.1820,      3.1820
0.9100,      4.2520,      3.2540
0.9200,      4.3181,      3.3328
0.9300,      4.3770,      3.4221
0.9400,      4.4260,      3.5233
0.9500,      4.4628,      3.6352
0.9600,      4.4859,      3.7554
0.9700,      4.4944,      3.8802
0.9800,      4.4880,      4.0052
0.9900,      4.4669,      4.1263
1.0000,      4.4320,      4.2396
1.0100,      4.3847,      4.3424
1.0200,      4.3270,      4.4334
1.0300,      4.2617,      4.5135
1.0400,      4.1920,      4.5861
1.0414,      4.1820,      4.5962
/ P1;

/ stop: xf=1.000000
1.0514,      4.1130,      4.6651
1.0614,      4.0476,      4.7305
1.0714,      3.9857,      4.7924
1.0814,      3.9274,      4.8508
1.0914,      3.8726,      4.9056
1.1014,      3.8213,      4.9568
1.1114,      3.7736,      5.0046
1.1214,      3.7294,      5.0487
1.1314,      3.6888,      5.0894
1.1414,      3.6516,      5.1265
1.1514,      3.6181,      5.1601
1.1614,      3.5880,      5.1902
1.1714,      3.5615,      5.2167
1.1814,      3.5385,      5.2397
1.1914,      3.5191,      5.2591
1.2014,      3.5031,      5.2750
1.2114,      3.4908,      5.2874
1.2214,      3.4819,      5.2962
1.2314,      3.4766,      5.3015
1.2414,      3.4749,      5.3033
/ R;

/ reverse.
/ V5;
1.2514,      3.4784,      5.2998
1.2614,      3.4890,      5.2892
1.2714,      3.5067,      5.2715
1.2814,      3.5314,      5.2467
1.2914,      3.5633,      5.2149
1.2914,      3.5633,      5.2149
/ G4,5,-1,-1.57080;

/ go: e=4.000000, xf=5.000000, yf=-1.000000, teta=-1.570800
1.3014,      3.5986,      5.1796
1.3114,      3.6340,      5.1442
1.3214,      3.6693,      5.1088
```

```
1.3314,     3.7047,     5.0735
1.3414,     3.7400,     5.0381
1.3514,     3.7754,     5.0028
1.3614,     3.8107,     4.9674
1.3714,     3.8461,     4.9321
1.3814,     3.8814,     4.8967
1.3914,     3.9168,     4.8614
1.4014,     3.9522,     4.8260
1.4114,     3.9875,     4.7907
1.4214,     4.0229,     4.7553
1.4314,     4.0582,     4.7199
1.4414,     4.0936,     4.6846
1.4514,     4.1289,     4.6492
1.4614,     4.1643,     4.6139
1.4714,     4.1996,     4.5785
1.4814,     4.2350,     4.5432
1.4914,     4.2704,     4.5078
1.5014,     4.3057,     4.4725
1.5114,     4.3411,     4.4371
1.5214,     4.3764,     4.4017
1.5314,     4.4118,     4.3664
1.5414,     4.4471,     4.3310
1.5514,     4.4825,     4.2957
1.5614,     4.5178,     4.2603
1.5714,     4.5532,     4.2250
1.5814,     4.5886,     4.1896
1.5914,     4.6239,     4.1543
1.6014,     4.6593,     4.1189
1.6114,     4.6946,     4.0835
1.6214,     4.7300,     4.0482
1.6314,     4.7653,     4.0128
1.6414,     4.8007,     3.9775
1.6514,     4.8360,     3.9421
1.6614,     4.8714,     3.9068
1.6714,     4.9068,     3.8714
1.6814,     4.9421,     3.8361
1.6914,     4.9775,     3.8007
1.7014,     5.0128,     3.7654
1.7114,     5.0482,     3.7300
1.7214,     5.0835,     3.6946
1.7314,     5.1189,     3.6593
1.7414,     5.1542,     3.6239
1.7514,     5.1896,     3.5886
1.7614,     5.2250,     3.5532
1.7714,     5.2603,     3.5179
1.7814,     5.2957,     3.4825
1.7914,     5.3310,     3.4472
1.8014,     5.3664,     3.4118
1.8114,     5.4017,     3.3764
1.8214,     5.4371,     3.3411
1.8314,     5.4724,     3.3057
1.8414,     5.5078,     3.2704
1.8514,     5.5432,     3.2350
1.8614,     5.5785,     3.1997
1.8714,     5.6139,     3.1643
1.8814,     5.6492,     3.1290
1.8914,     5.6846,     3.0936
1.9014,     5.7199,     3.0582
1.9114,     5.7553,     3.0229
1.9214,     5.7906,     2.9875
1.9314,     5.8260,     2.9522
1.9414,     5.8614,     2.9168
1.9514,     5.8967,     2.8815
1.9614,     5.9321,     2.8461
1.9714,     5.9674,     2.8108
1.9814,     6.0028,     2.7754
1.9914,     6.0381,     2.7400
```

```
2.0014,      6.0735,      2.7047
2.0114,      6.1088,      2.6693
2.0214,      6.1442,      2.6340
2.0314,      6.1796,      2.5986
2.0414,      6.2149,      2.5633
2.0514,      6.2503,      2.5279
2.0614,      6.2856,      2.4926
2.0714,      6.3210,      2.4572
2.0814,      6.3563,      2.4219
2.0914,      6.3917,      2.3865
2.0914,      6.3917,      2.3865
2.0914,      6.3917,      2.3865
2.1014,      6.4270,      2.3510
2.1114,      6.4617,      2.3145
2.1214,      6.4955,      2.2762
2.1314,      6.5279,      2.2357
2.1414,      6.5584,      2.1925
2.1514,      6.5868,      2.1463
2.1614,      6.6126,      2.0972
2.1714,      6.6357,      2.0452
2.1814,      6.6557,      1.9904
2.1914,      6.6725,      1.9332
2.2014,      6.6858,      1.8740
2.2114,      6.6956,      1.8131
2.2214,      6.7017,      1.7510
2.2314,      6.7041,      1.6883
2.2414,      6.7028,      1.6256
2.2514,      6.6977,      1.5633
2.2614,      6.6890,      1.5020
2.2714,      6.6766,      1.4422
2.2814,      6.6608,      1.3844
2.2914,      6.6417,      1.3289
2.3014,      6.6195,      1.2761
2.3114,      6.5944,      1.2261
2.3214,      6.5667,      1.1791
2.3314,      6.5367,      1.1351
2.3414,      6.5049,      1.0938
2.3514,      6.4714,      1.0550
2.3614,      6.4369,      1.0180
2.3714,      6.4017,      0.9823
2.3743,      6.3917,      0.9723
/ I5,-3,3.14159;

/ global_set: X=5.000000, Y=-3.000000, Alfa=3.141590
/ G,2,-1,0;

/ go: e=0.000000, xf=2.000000, yf=-1.000000, teta=0.000000
2.3743,      5.0000,     -3.0000
2.3843,      4.9500,     -2.9999
2.3943,      4.9002,     -2.9992
2.4043,      4.8506,     -2.9973
2.4143,      4.8015,     -2.9938
2.4243,      4.7528,     -2.9883
2.4343,      4.7046,     -2.9804
2.4443,      4.6571,     -2.9701
2.4543,      4.6101,     -2.9570
2.4643,      4.5639,     -2.9411
2.4743,      4.5183,     -2.9223
2.4843,      4.4734,     -2.9007
2.4943,      4.4292,     -2.8762
2.5043,      4.3856,     -2.8490
2.5143,      4.3427,     -2.8192
2.5243,      4.3003,     -2.7870
2.5343,      4.2584,     -2.7526
2.5443,      4.2170,     -2.7163
2.5543,      4.1760,     -2.6782
2.5643,      4.1353,     -2.6388
```

```
2.5743,      4.0948,     -2.5982
2.5843,      4.0546,     -2.5569
2.5943,      4.0145,     -2.5151
2.6043,      3.9744,     -2.4732
2.6143,      3.9342,     -2.4315
2.6243,      3.8939,     -2.3904
2.6343,      3.8534,     -2.3501
2.6443,      3.8127,     -2.3110
2.6543,      3.7715,     -2.2734
2.6643,      3.7300,     -2.2376
2.6743,      3.6880,     -2.2038
2.6843,      3.6454,     -2.1722
2.6943,      3.6023,     -2.1431
2.7043,      3.5585,     -2.1167
2.7143,      3.5141,     -2.0930
2.7243,      3.4690,     -2.0721
2.7343,      3.4233,     -2.0542
2.7443,      3.3769,     -2.0391
2.7543,      3.3298,     -2.0268
2.7643,      3.2820,     -2.0171
2.7743,      3.2337,     -2.0100
2.7843,      3.1849,     -2.0051
2.7943,      3.1356,     -2.0020
2.8043,      3.0859,     -2.0005
2.8143,      3.0361,     -2.0000
2.8215,      3.0000,     -2.0000
/ Q;

/ stop0: V=5.000000
2.8315,      2.9550,     -2.0000
2.8415,      2.9200,     -2.0000
2.8515,      2.8950,     -2.0000
2.8615,      2.8800,     -2.0000
2.8715,      2.8750,     -2.0000
2.8715,      2.8750,     -2.0000
/ U20;
2.8815,      2.8700,     -2.0000
2.8915,      2.8550,     -2.0000
2.9015,      2.8300,     -2.0000
2.9115,      2.7950,     -2.0000
2.9215,      2.7500,     -2.0000
2.9315,      2.6950,     -2.0000
2.9415,      2.6300,     -2.0000
2.9515,      2.5550,     -2.0000
2.9615,      2.4700,     -2.0000
2.9715,      2.3750,     -2.0000
2.9815,      2.2700,     -2.0000
2.9915,      2.1550,     -2.0000
3.0015,      2.0300,     -2.0000
3.0115,      1.8950,     -2.0000
3.0215,      1.7500,     -2.0000
3.0315,      1.5950,     -2.0000
3.0415,      1.4300,     -2.0000
3.0515,      1.2550,     -2.0000
3.0615,      1.0700,     -2.0000
3.0715,      0.8750,     -2.0000
/ P6;

/ stop: xf=6.000000
3.0815,      0.6767,     -2.0000
3.0915,      0.4817,     -2.0000
3.1015,      0.2900,     -2.0000
3.1115,      0.1017,     -2.0000
3.1215,     -0.0833,     -2.0000
3.1315,     -0.2650,     -2.0000
3.1415,     -0.4433,     -2.0000
3.1515,     -0.6183,     -2.0000
```

```
3.1615,    -0.7900,    -2.0000
3.1715,    -0.9583,    -2.0000
3.1815,    -1.1233,    -2.0000
3.1915,    -1.2850,    -2.0000
3.2015,    -1.4433,    -2.0000
3.2115,    -1.5983,    -2.0000
3.2215,    -1.7500,    -2.0000
3.2315,    -1.8983,    -2.0000
3.2415,    -2.0433,    -2.0000
3.2515,    -2.1850,    -2.0000
3.2615,    -2.3233,    -2.0000
3.2715,    -2.4583,    -2.0000
3.2815,    -2.5900,    -2.0000
3.2915,    -2.7183,    -2.0000
3.3015,    -2.8433,    -2.0000
3.3115,    -2.9650,    -2.0000
3.3215,    -3.0833,    -2.0000
3.3315,    -3.1983,    -2.0000
3.3415,    -3.3100,    -2.0000
3.3515,    -3.4183,    -2.0000
3.3615,    -3.5233,    -2.0000
3.3715,    -3.6250,    -2.0000
3.3815,    -3.7233,    -2.0000
3.3915,    -3.8183,    -2.0000
3.4015,    -3.9100,    -2.0000
3.4115,    -3.9983,    -2.0000
3.4215,    -4.0833,    -2.0000
3.4315,    -4.1650,    -2.0000
3.4415,    -4.2433,    -2.0000
3.4515,    -4.3183,    -2.0000
3.4615,    -4.3900,    -2.0000
3.4715,    -4.4583,    -2.0000
3.4815,    -4.5233,    -2.0000
3.4915,    -4.5850,    -2.0000
3.5015,    -4.6433,    -2.0000
3.5115,    -4.6983,    -2.0000
3.5215,    -4.7500,    -2.0000
3.5315,    -4.7983,    -2.0000
3.5415,    -4.8433,    -2.0000
3.5515,    -4.8850,    -2.0000
3.5615,    -4.9233,    -2.0000
3.5715,    -4.9583,    -2.0000
3.5815,    -4.9900,    -2.0000
3.5915,    -5.0183,    -2.0000
3.6015,    -5.0433,    -2.0000
3.6115,    -5.0650,    -2.0000
3.6215,    -5.0833,    -2.0000
3.6315,    -5.0983,    -2.0000
3.6415,    -5.1100,    -2.0000
3.6515,    -5.1183,    -2.0000
3.6615,    -5.1233,    -2.0000
3.6714,    -5.1250,    -2.0000
```

Appendix E

Optimisation results

These are figures showing various results from the tests made with the optimisation genetic algorithm. Each figure's caption indicates the criteria selected.

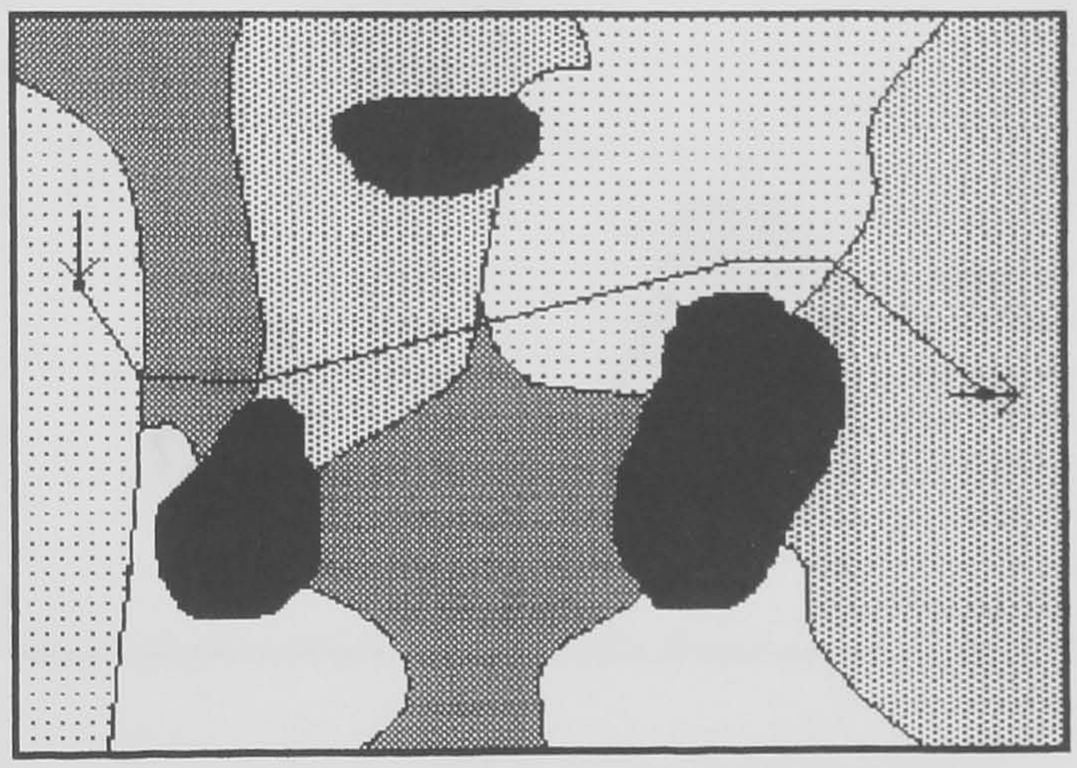


Figure E. 1 50% Time and 50% Risk optimisation

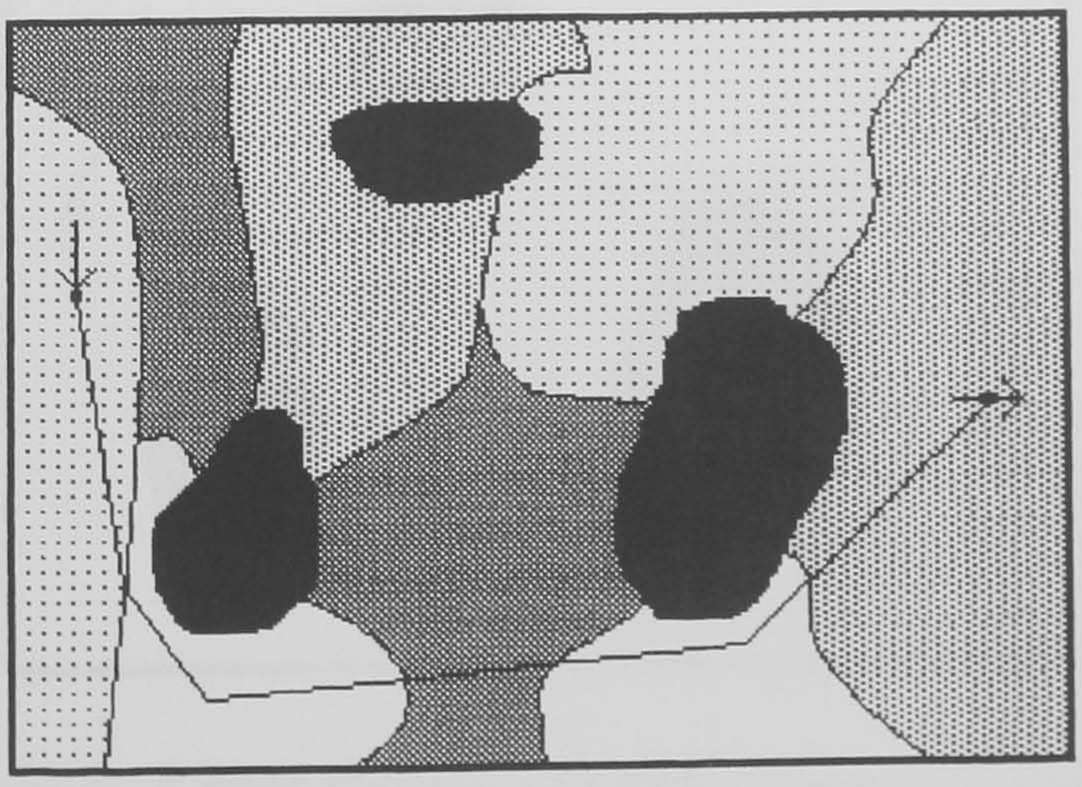


Figure E. 2 50% Energy and 50% Risk optimisation

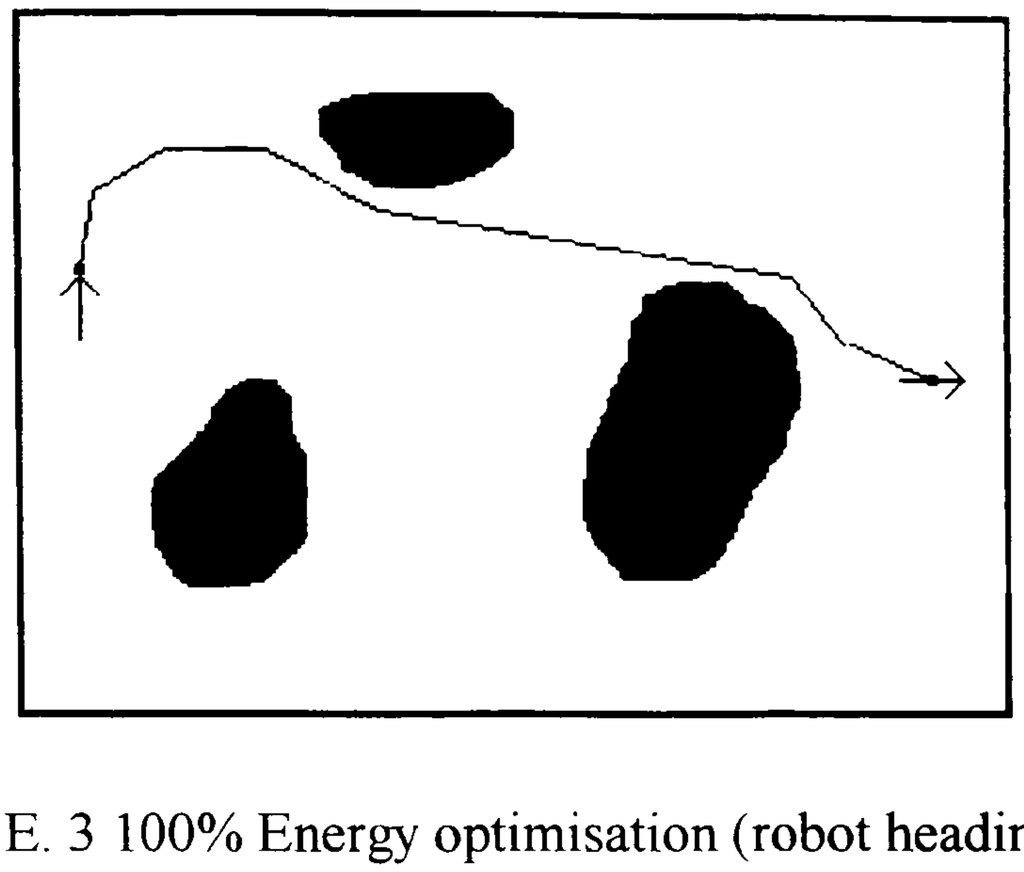


Figure E. 3 100% Energy optimisation (robot heading north)

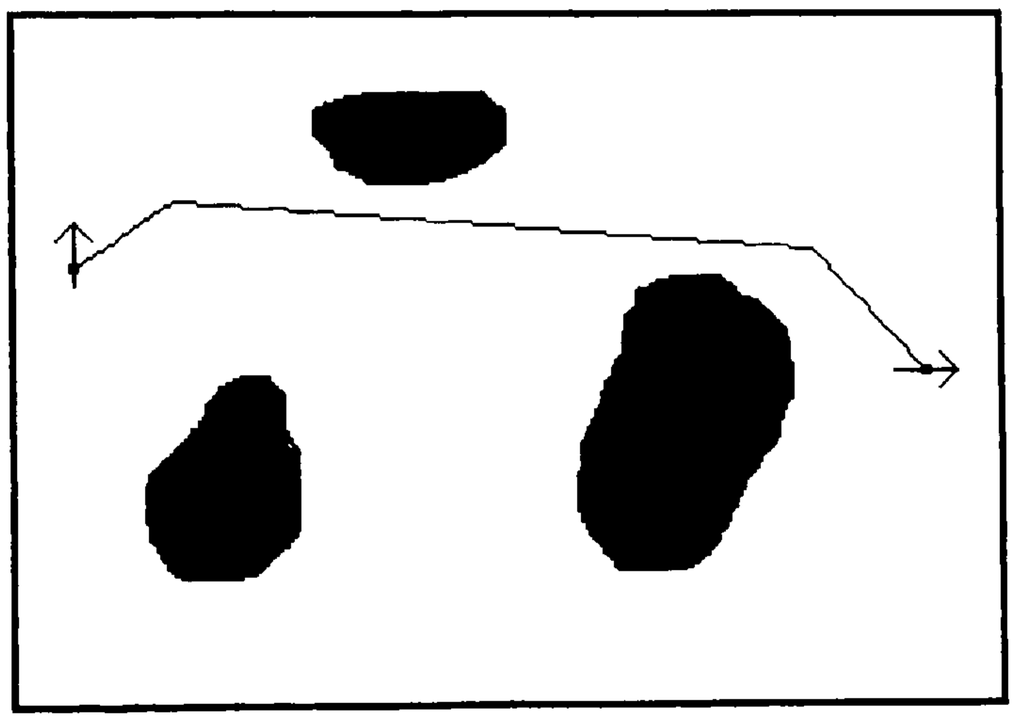


Figure E. 4 100% Time optimisation

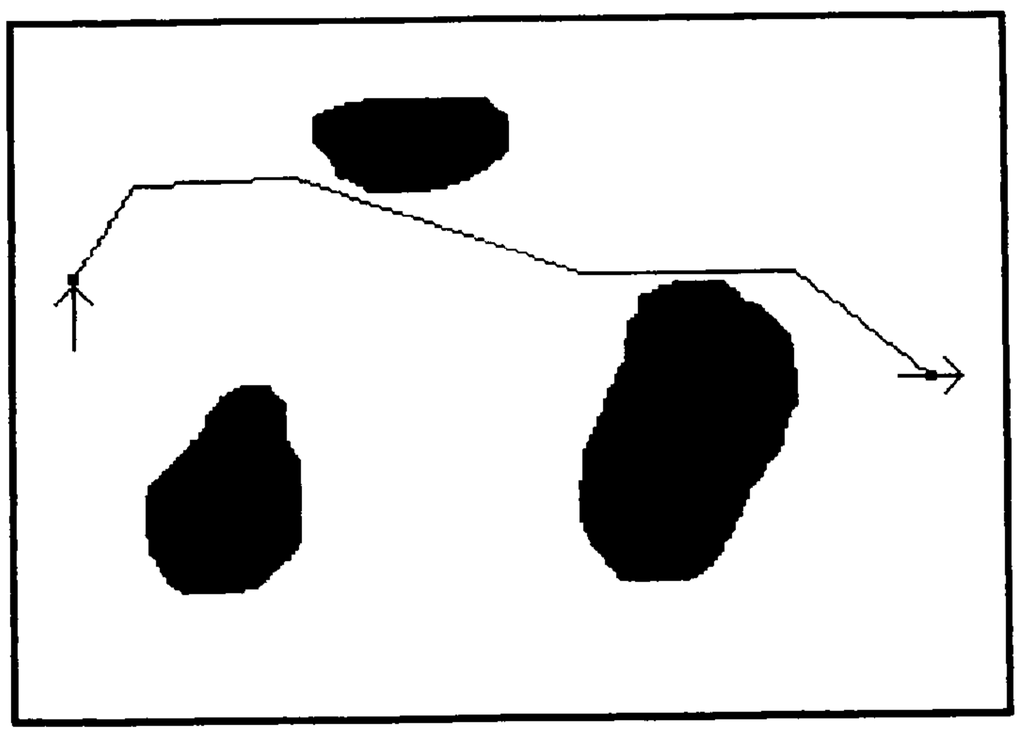


Figure E. 5 50% Energy and 50% Time optimisation

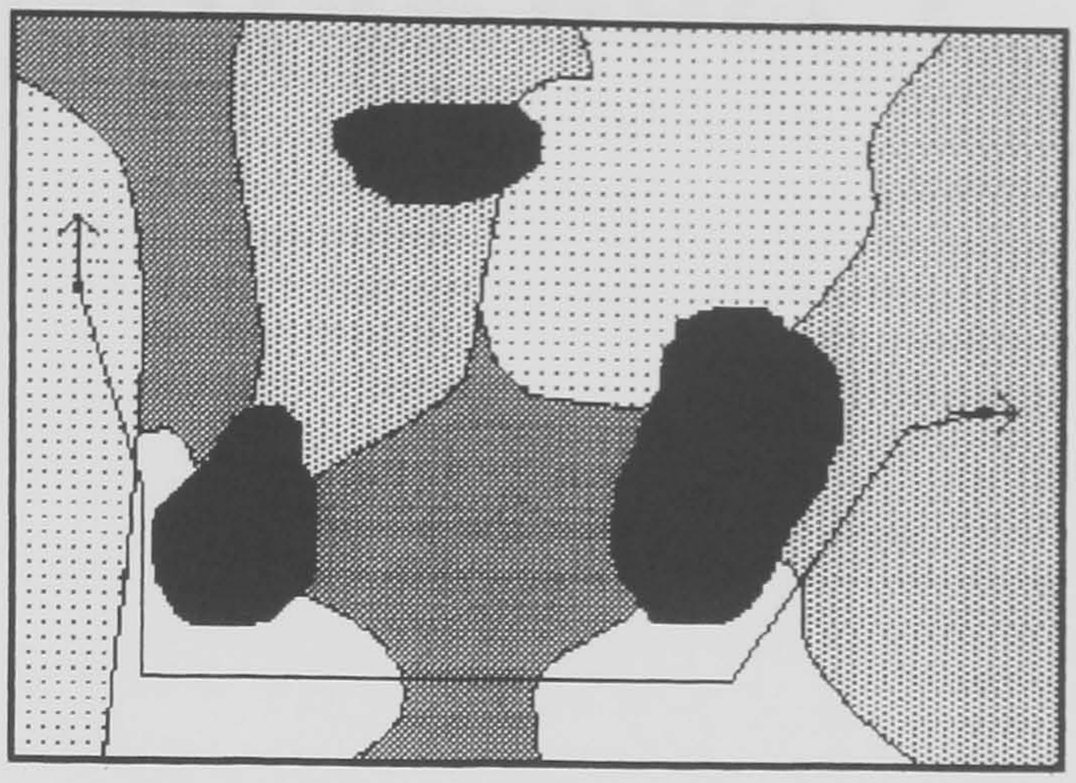


Figure E. 6 100% Risk optimisation

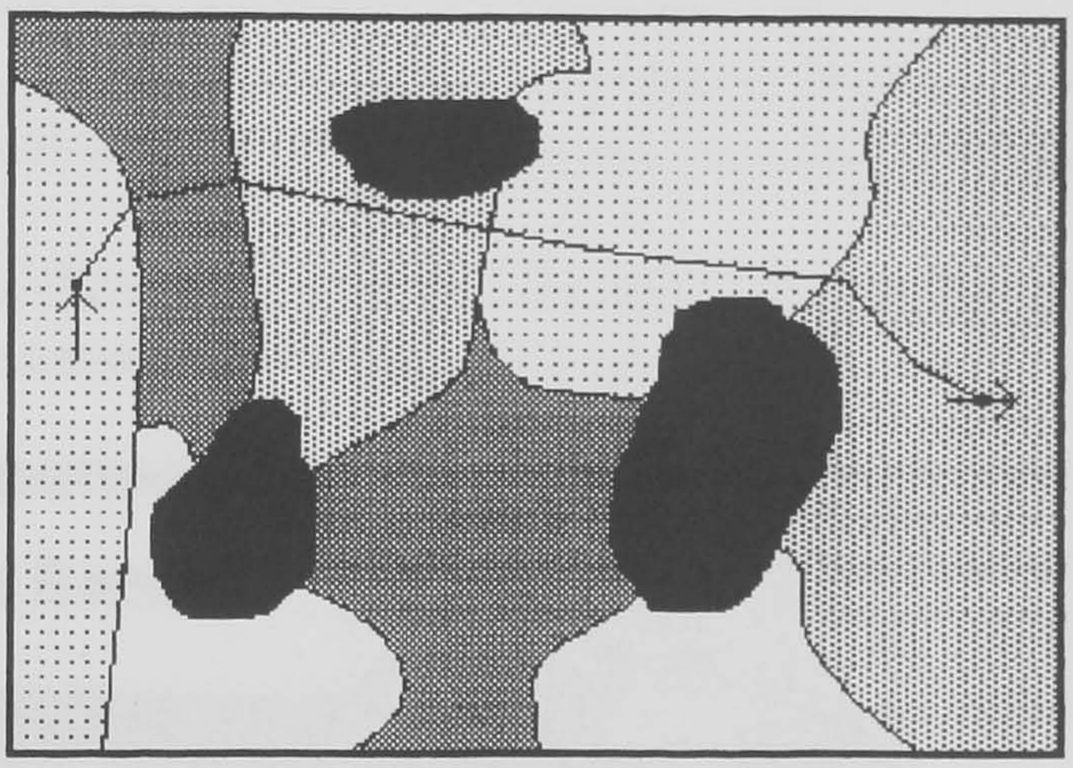


Figure E. 7 50% Energy 50% Risk optimisation

Appendix F

Self simulator

The self-simulator was designed for a mobile robot developed in the Department of Electronics Engineering of the Pontificia Universidad Javeriana under the supervision of the author. The robot was connected to external sensors and the reading of these sensors and the internal sensors of the robot were monitored to develop the simulator.

The following figures show the robot, the mechanical and electronic structure of it, and the connection to the simulator training system.

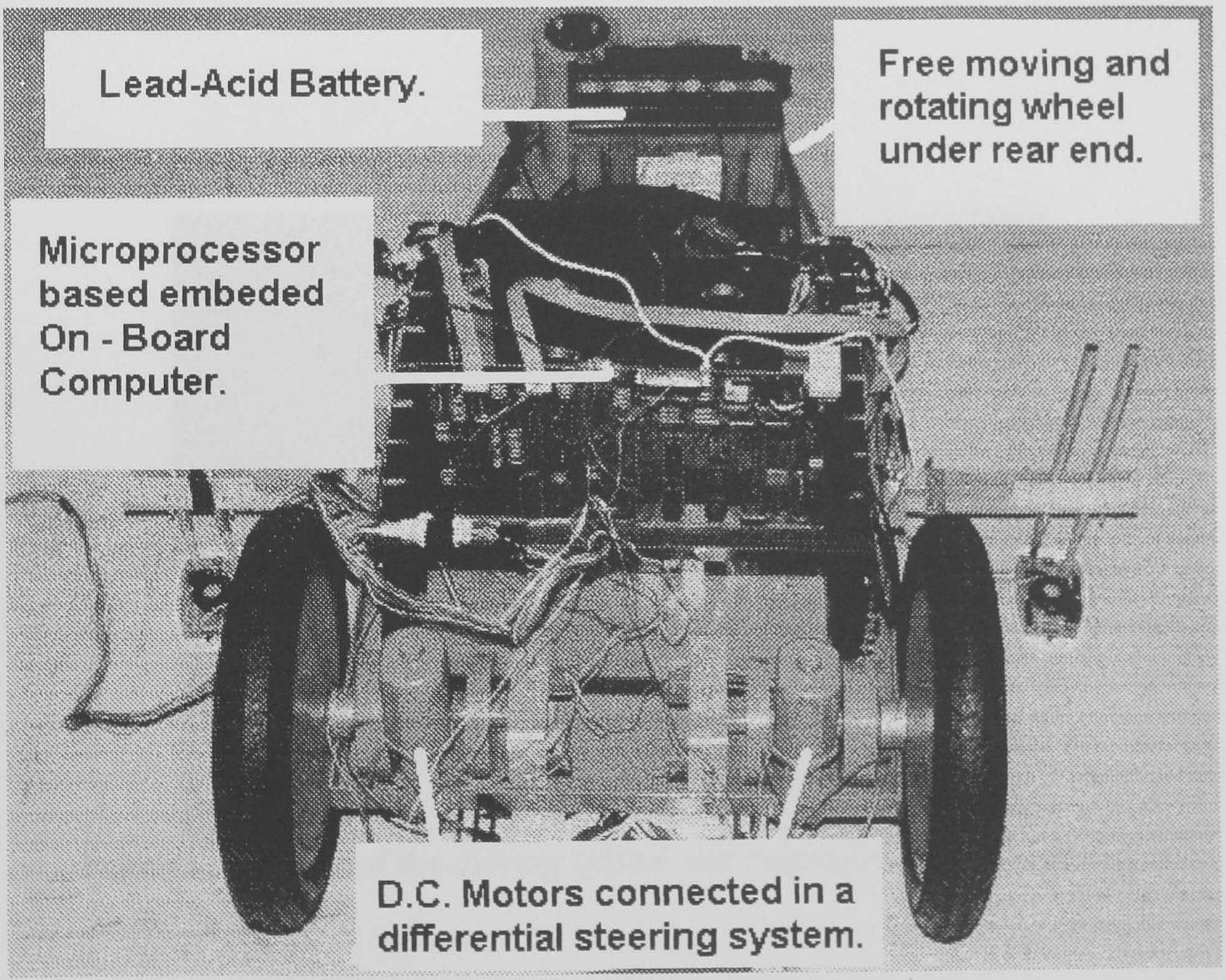


Figure F. 1 Mechanical structure of the robot

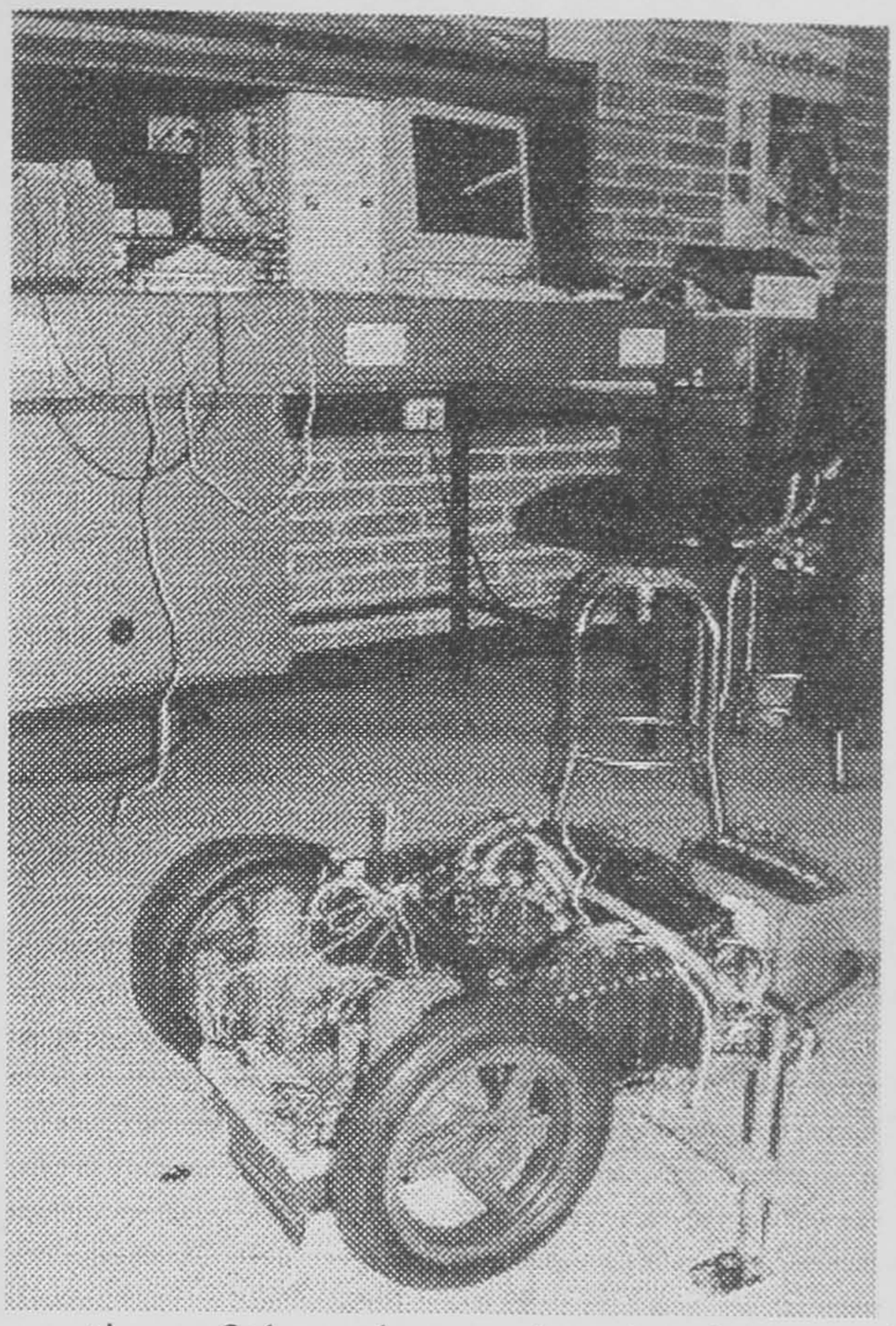


Figure F. 2 Connection of the robot to the simulator training computer

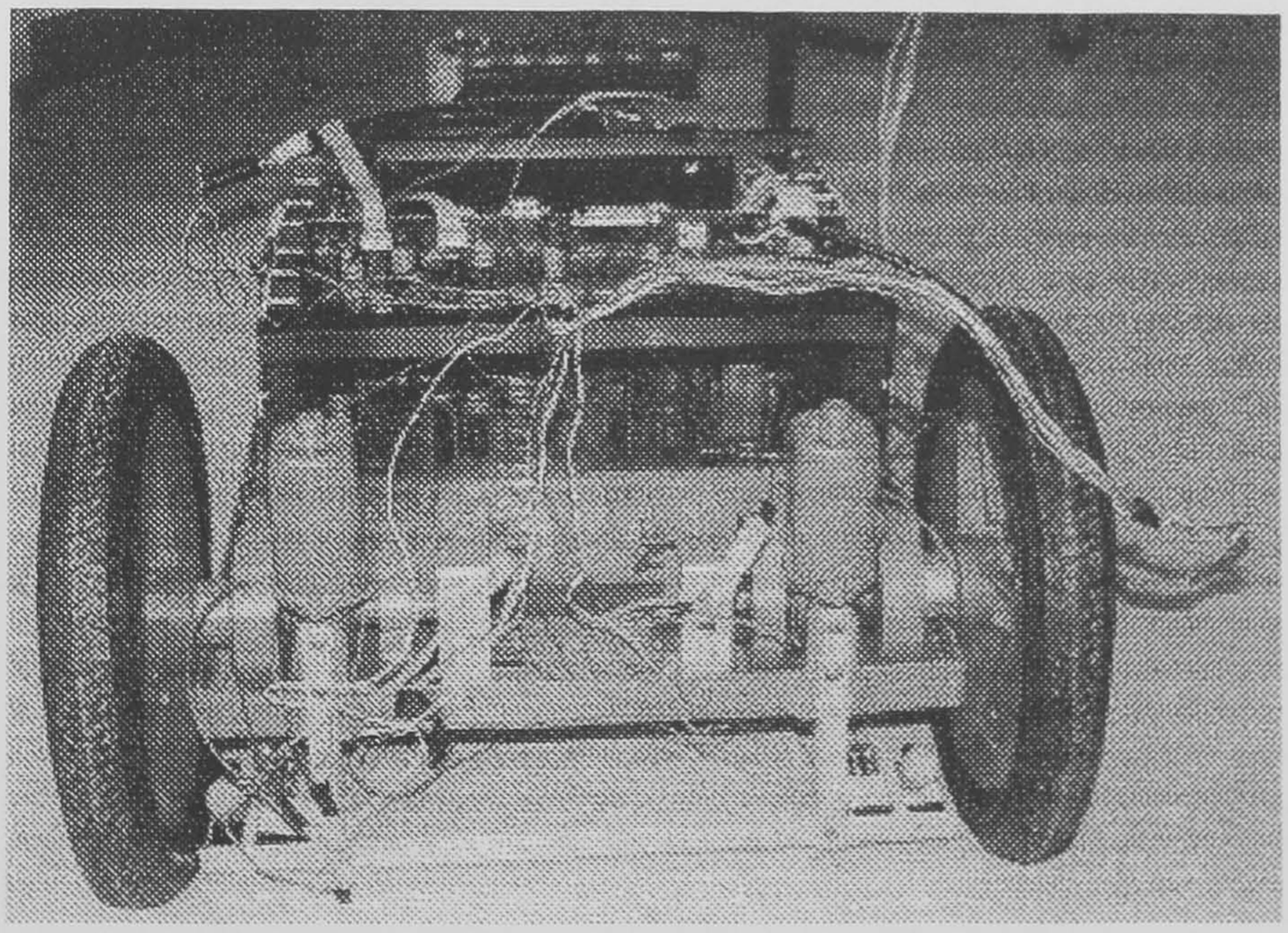


Figure F. 3 Detail of the driving system and internal sensor monitoring

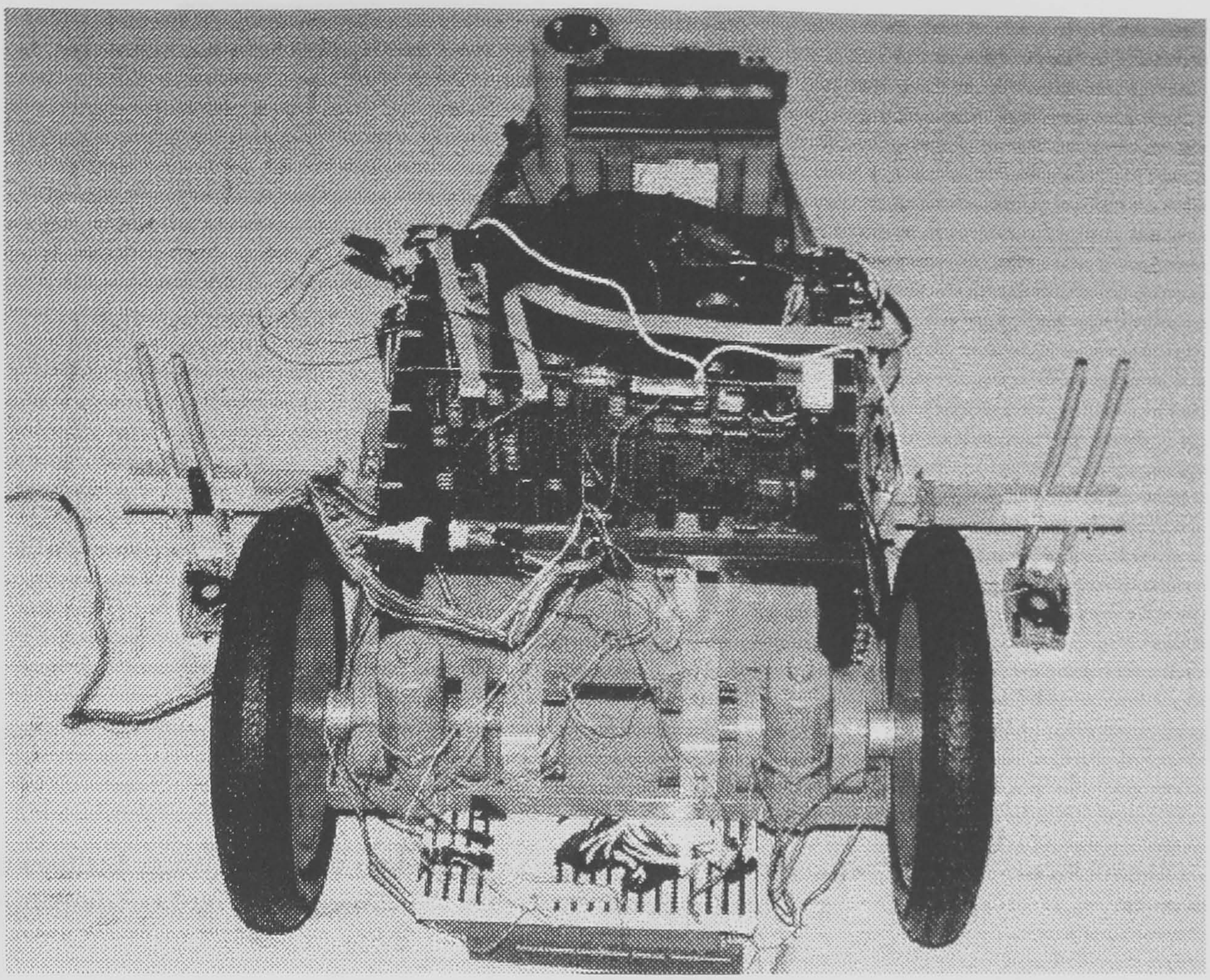


Figure F. 4 Conection to external sensors

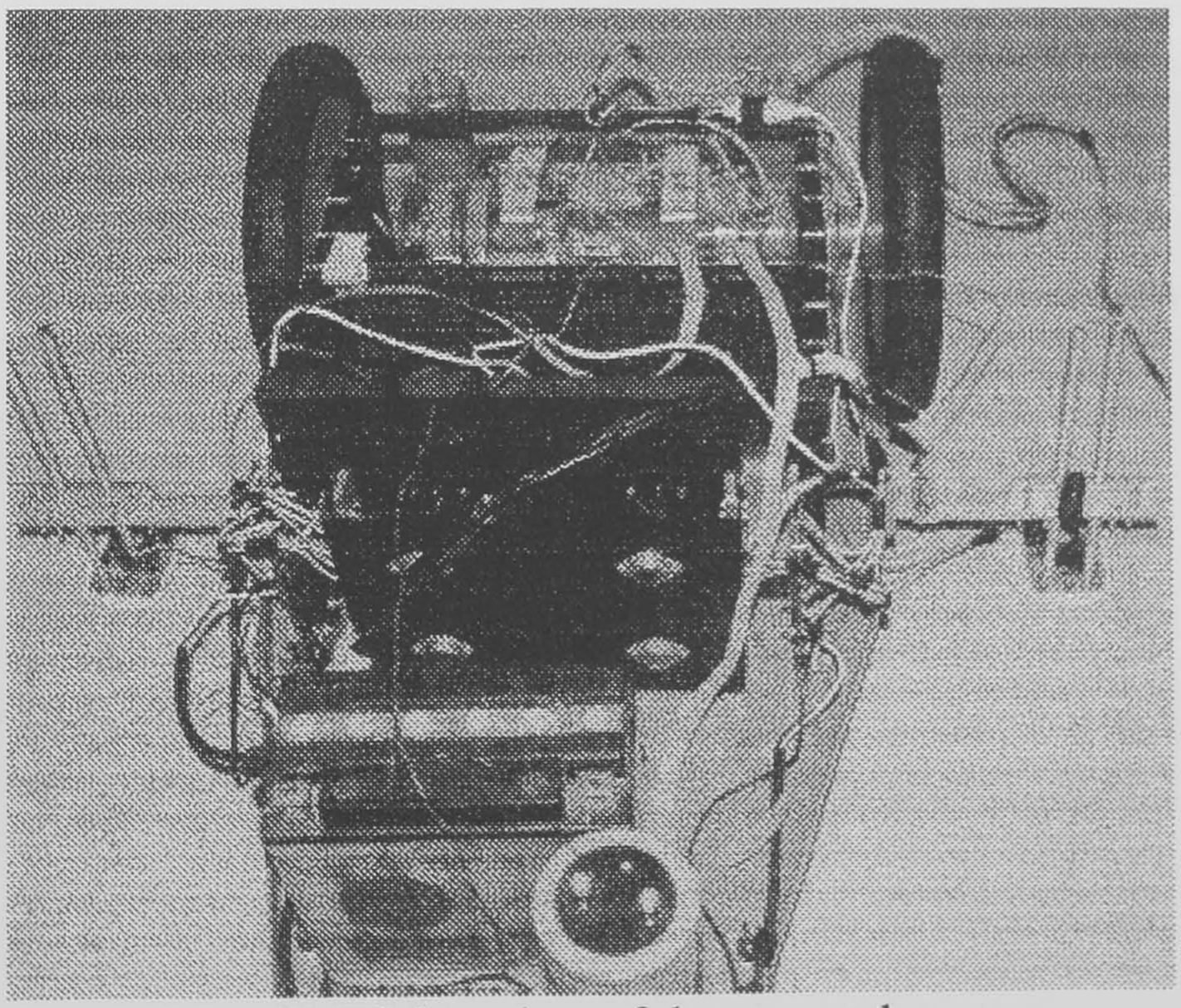


Figure F. 5 Other view of the external sensors

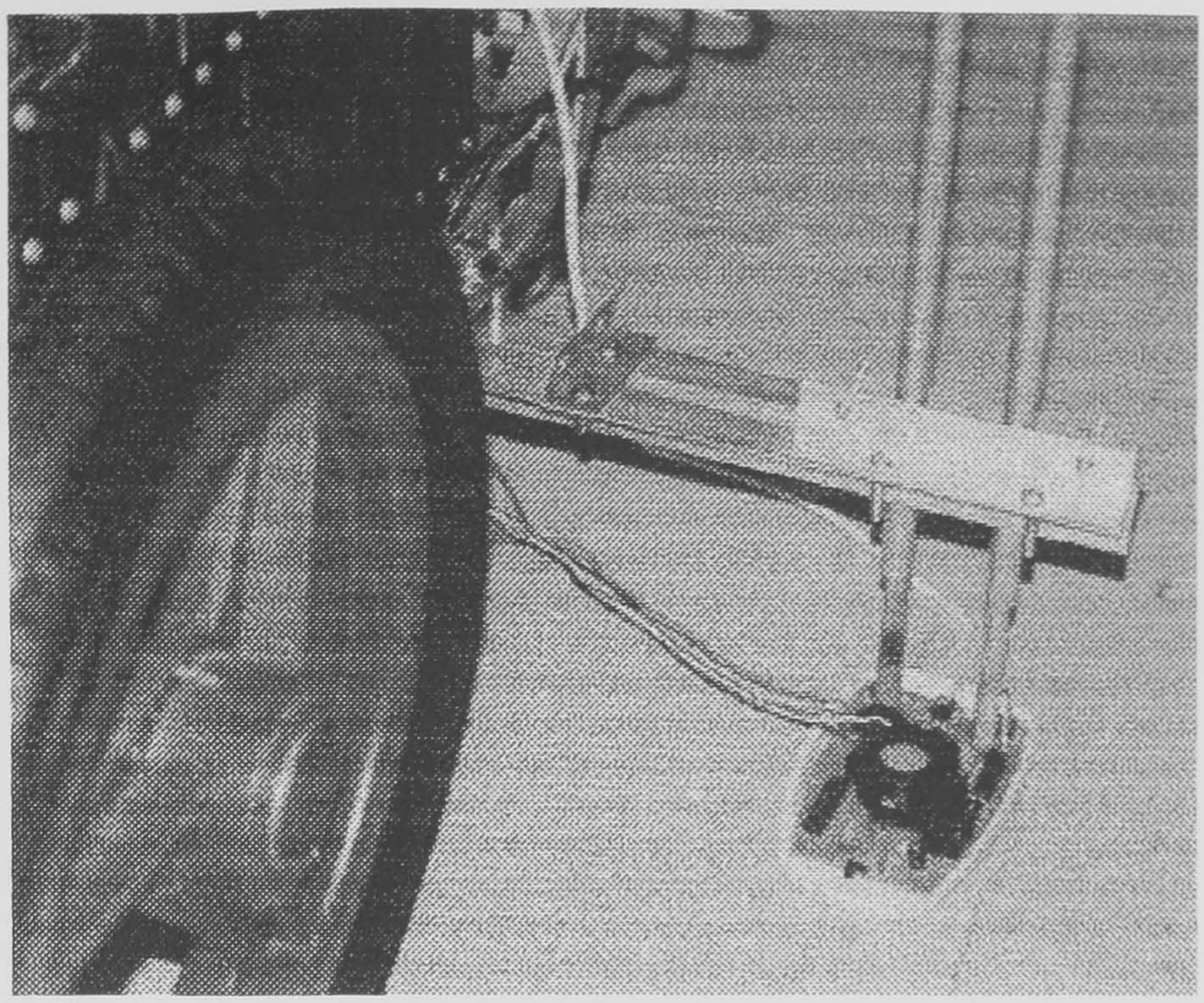


Figure F. 6 Detail of the external sensors. Computer mouse were used for this purpose.

The control system consists of a digital control computer (microprocessor, memory, peripherals and digital motor drivers), and the power driver.

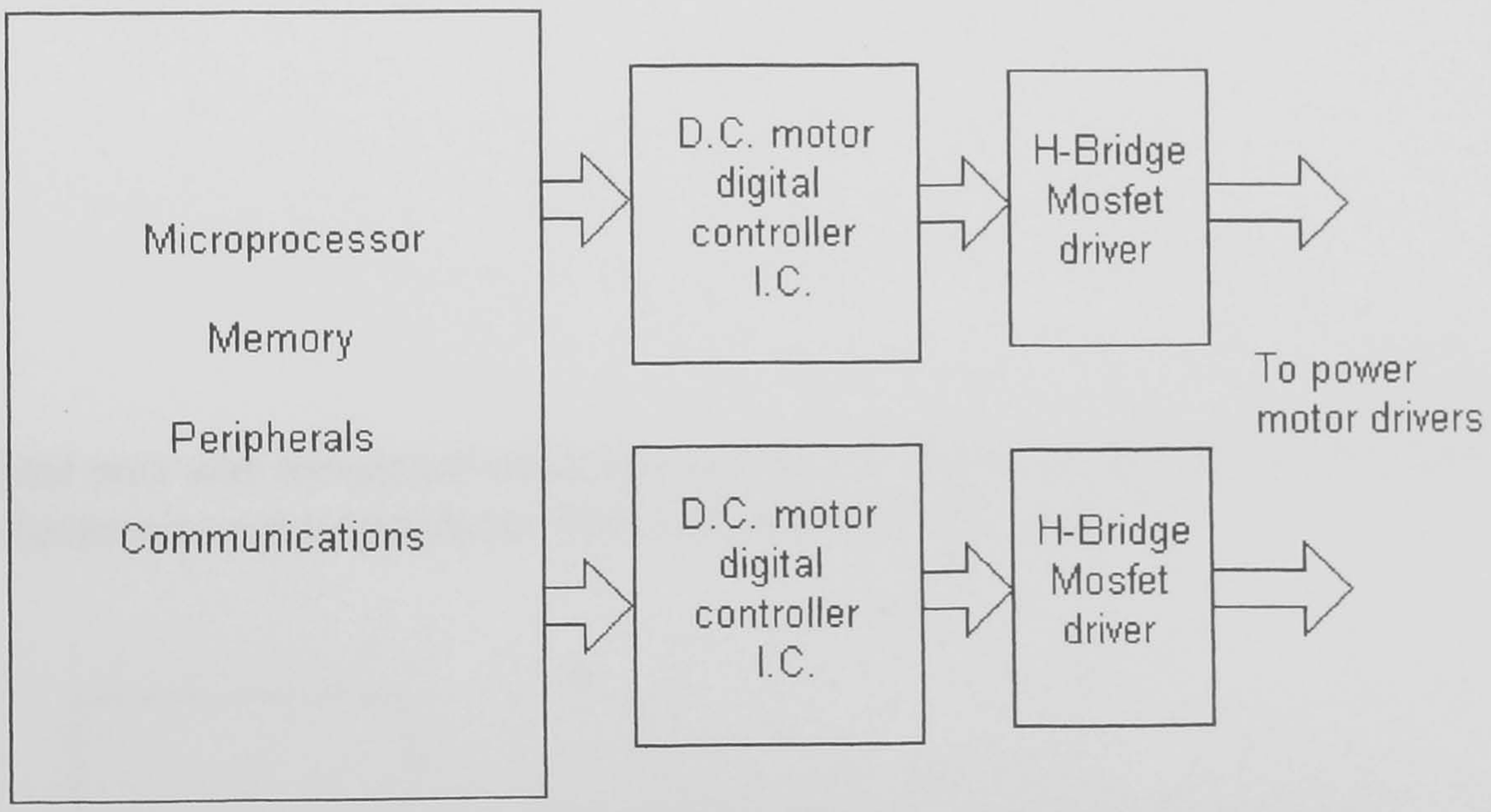


Figure F. 7 Electronics control system (digital module)

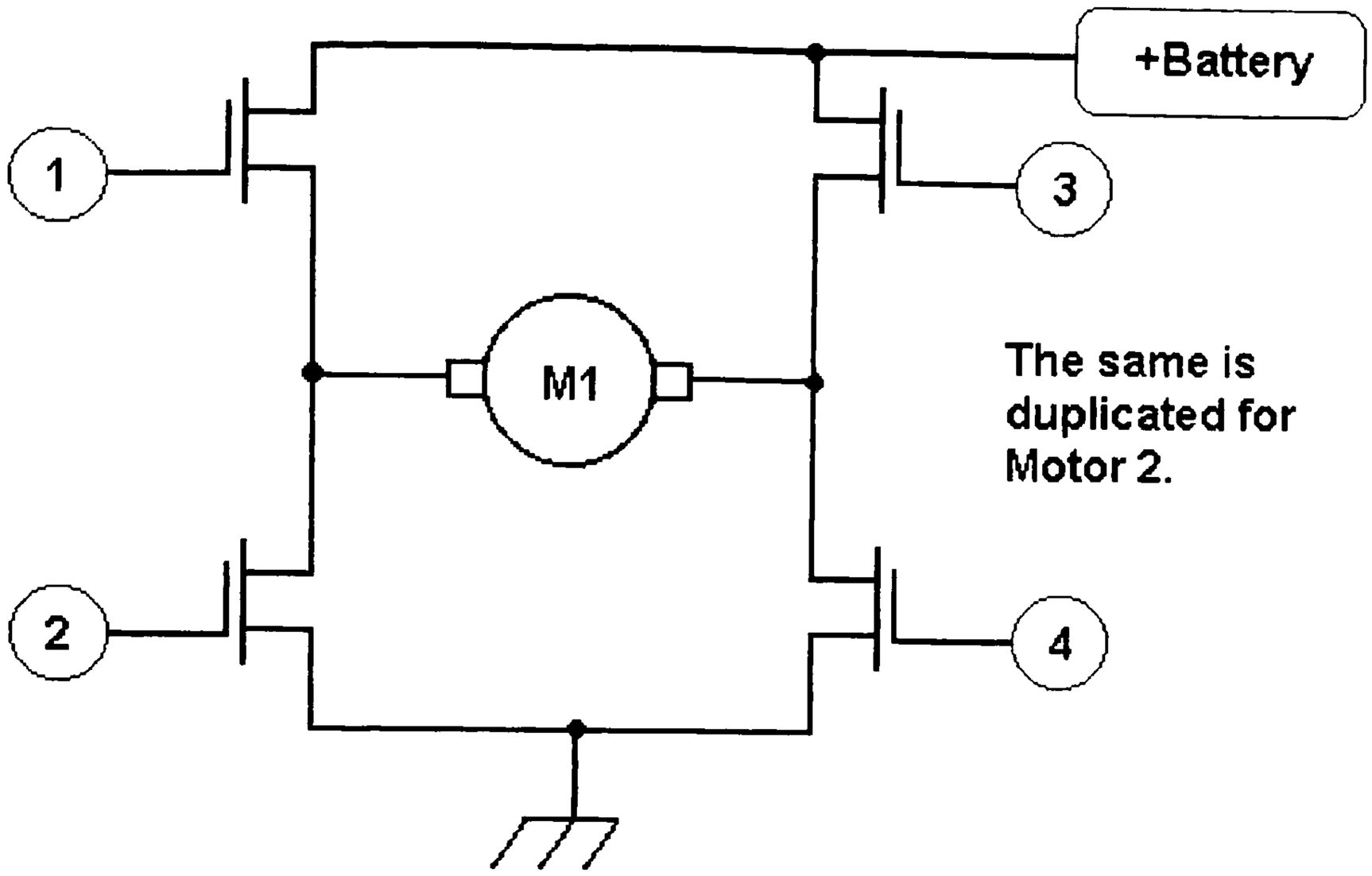


Figure F. 8 Power electronics module struture (analog part)

The digital part was simulated straight away by a C program. The mechanical and the power electronics were simulated with a neural network each.

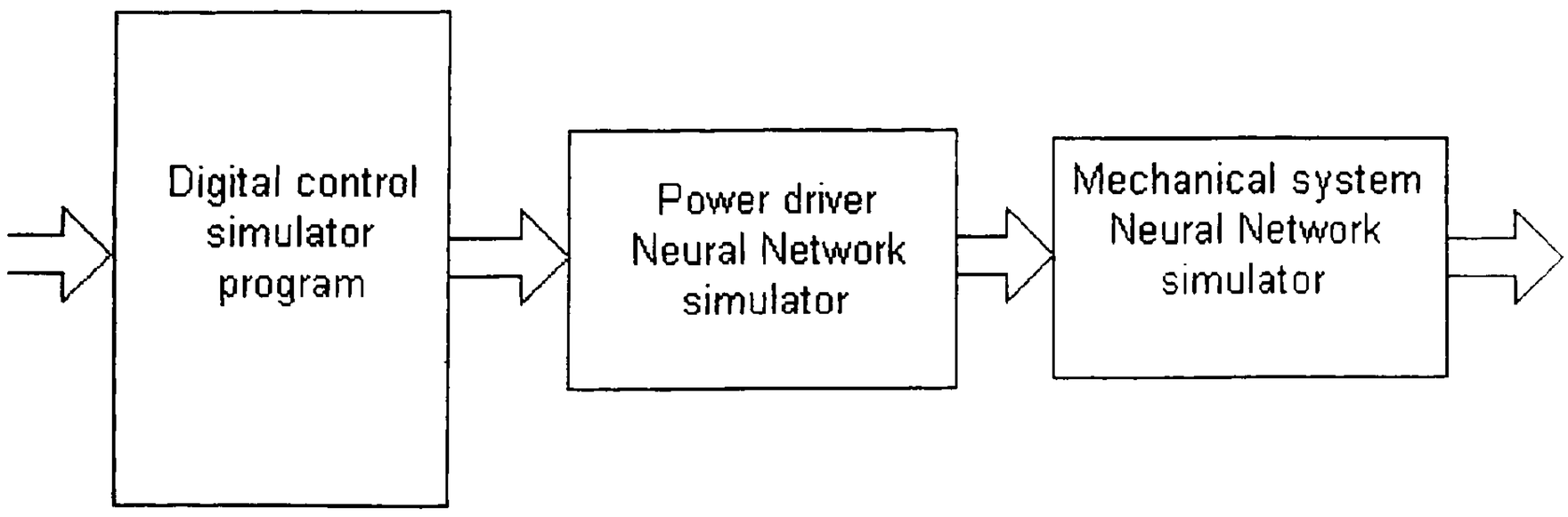


Figure F. 9 Struture of the self simulator

Once the simulator is being used inside the on-board computer of the robot, the digital control simulator is no longer needed since it is the same as the control computer program for motion control. The same outputs that the computer sends to the power driver are sent to the simulator.