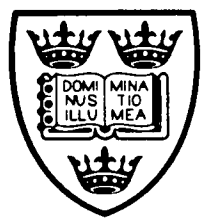
# Dynamic Planning and Real-time Control for a Mobile Robot

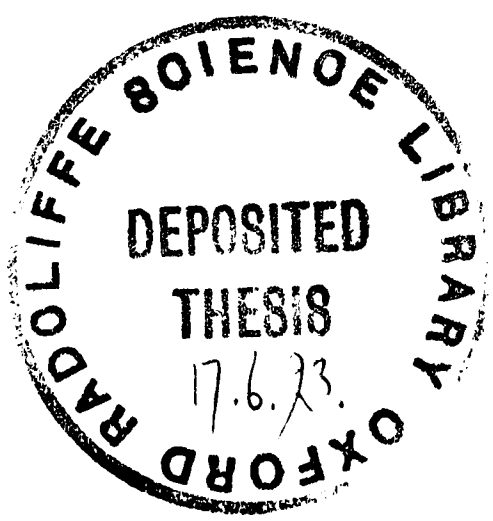
*by*

**Huosheng Hu**



Department of Engineering Science
University of Oxford

Michaelmas Term, 1992



This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfillment of the requirements for the degree of Doctor of Philosophy. The thesis is entirely my own work, and except where otherwise stated, describes my own research.

Huosheng Hu, Keble College, Oxford

Huosheng Hu
Keble College

Doctor of Philosophy
Michaelmas, 1992

# Dynamic Planning and Real-time Control for a Mobile Robot

## Abstract

A mobile robot becomes more intelligent as its control system is given more capabilities to respond to its environment autonomously. This thesis develops a distributed real-time control system for a mobile robot which is intended to operate autonomously in an industrial environment. It is a unified approach to real-time sensing, planning, and control based on a parallel processing architecture.

To be fully autonomous, a mobile robot must be able to sense its environment, build or update maps, plan and execute actions, and adapt its behaviour to environmental changes. The ability of a control system to support these complex tasks in real time is significantly affected by the organisation of information pathways within the architecture. After examining different architectures described in the literature, a transputer-based architecture is developed to maximise the parallel information flow from sensing to action to provide minimal delay in responding to a dynamically changing environment.

Taking account of uncertainty, planning an optimal path is difficult since the internal world model quickly becomes invalid. To find a solution, dynamic changes in the environment are classified as different types of obstacles that are assumed to appear randomly. Bayes' theorem is applied to build statistical models to estimate the mean number unexpected obstacles encountered. This provides a feasible way for the global path planner to update its internal world model dynamically based on available sensor information. A dynamic programming algorithm is used to plan an optimal path.

An array of sonar sensors is used to detect dynamic changes in the environment. To reduce uncertainty in noisy data, a probabilistic sensor model and rule-based heuristics are built. The collision avoidance problem is formulated using decision theory to achieve both collision-free and optimal solution. An optimal decision rule to avoid unexpected obstacles is calculated to minimise the Bayes risk in trading between a careful maneuver and an alternative path.

To control the motion of the robot to follow the planned path, a new guidance system is proposed to provide dynamic trajectory planning and optimal tracking capability to a mobile robot that is subject to nonholonomic kinematic constraints.

The success of this approach is demonstrated by the *Turtle* mobile robot which is able to interact intelligently with a dynamically changing environment.

# Acknowledgements

I am very grateful to Prof. Mike Brady for introducing me to the AGV research and providing excellent motivation and constant support. Many thanks to Mike Brady and Penny Probert for their constant encouragement, critical comments, and many insightful discussions at various stages of the work described in this thesis. I am also indebted to Hugh Durrant-Whyte for providing constant support and encouragement. Thanks to Stuart Russell for useful discussions during his stay at Oxford.

All the members of the Robotics Research Group have helped me in some way, and enriched my time at Oxford. In particular, I would like to thank Fenglei Du for his help with creating image sequence from AGV demonstration video. Thanks to Nicola Ferrier, Billur Barshan, Chris Pearson, and Michael Stevens for their help in proof-reading draft of this thesis. My thanks also go to Martin Adams, Gursel Alici, Jennet Batten, Paul Beardsley, Alan Brown, Yang Gao, Stewart Grime, Sarah Harrington, Henry Lau, Youfu Li, John Leonard, James Manyika, Bobby Rao, Paul Sharkey, Bill Triggs, Han Wang, and Wu Wen. My time at Oxford has also been enriched by the pleasant company of all my friends throughout the University, especially thanks to Gang Liu, Desong Chen, Anan He, Yougan Wang, and Gang Wei for friendship and useful conversations.

Thanks also to Pete Lindsey, Jon Tombs and Simon Turner for administering the computer system. Conversations with a stream of visitors from different countries to our AGV Lab have forced me to express my idea clearly and provided insights.

The research reported herein has benefited from collaboration with GEC Rugby, BP, and GCS Ltd, especially thanks to Malcolm Roberts, Russel Miles, John Potter, and Alistair Kinross for their help with the AGV. I would like to thank the SERC ACME Directorate who funded this work under grant GR/G37361.

Finally but by no means least, I would like to thank my wife for her love and my daughter for her understanding, as well as my parents and family for their support.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Mobile robots are a paradigm of the challenge of system integration in Robotics [Brady *et al.*, 1987]. This thesis describes the development of a real-time control system for a mobile robot which is expected to be able to navigate in a factory or a warehouse autonomously. A unified approach to real time sensing, planning, and control based on a parallel processing architecture is the central issue in the thesis. The success of the approach is demonstrated by the *Turtle* mobile robot which interacts intelligently with a dynamically changing environment.

## 1.1 Motivation

### 1.1.1 Mobile Robots in Industry

To date, mobile robots or autonomous guided vehicles (AGVs) that exist in industry to transport material or clean a warehouse are mainly wire-guided, typically by the burial of a signal-carrying wire, or by the painting of a visible line on the floor [Premi and Besant, 1983] [Tsumura, 1986] [Hollier, 1987]. As a result, they can only follow predetermined paths that form an unchanging network. Such AGVs need, and have, very limited sensing capabilities and in the presence of obstacles can only stop. Installation of these wire guides can be costly and disruptive, and modifications may be very difficult. Also, the cost and inflexibility of laying out the path inhibits use of the high density of paths which would maximise the flexibility and efficiency of their applications. These limitations have seriously hampered commercial application of AGVs.

Recently, a more flexible approach to the guidance of industrial mobile robots has been based on locating artificial beacons in the environment, such as retro-reflecting bar charts [Stevens *et al.*,

1983], light spots [Takeda *et al.*, 1986], laser and sonic beacons [Tsumura, 1986]. Rather than physical guidance, mobile robots are free to move around their environment as long as beacons are "in sight". Such mobile robots potentially have the ability for their paths to be modified dynamically and are termed *free ranging*. However, they still have no capability to avoid unexpected obstacles and can only stop and wait for the obstacle to be removed. Also, the mobile robots can only follow the path commanded by a central control computer and have no capability to make decisions locally in the face of unexpected events. A further problem with such a highly centralised control scheme is that if anything goes wrong with the control center, it may cause the whole system to stop. Also the bandwidth of the central control system is limited.

### 1.1.2 Towards fully Autonomous competence

The capabilities of a robot system are strongly dependent on the ability to observe and understand the environment in which it operates [Durrant-Whyte, 1988]. Since an industrial environment is complex and dynamic, it is impossible for a designer to foresee all of the circumstances that might be faced by a mobile robot in continuous, long-term interaction with such an environment. Therefore, any fully autonomous mobile robot must possess a considerable degree of autonomy, such as: robust navigation, obstacle avoidance, dynamic planning and environment learning. In other words, it should be able to adapt flexibly its behaviour to unexpected situations and dynamic changes in the environment without explicitly being told what to do at each moment. It is essential to build a decentralised control system in industrial applications to obtain high reliability and flexibility.

Considerable progress has been made recently in the design and implementation of experimental mobile robots that can cope with an unknown or dynamically changing environment [Giralt *et al.*, 1984] [Brooks, 1986][Harmon, 1987][Jarvis and Byrne, 1988][Borenstein and Koren, 1989]. They are characterised by rapid, real-time response to unexpected obstacles based on sensor-based control. It is essential for a mobile robot to work safely in industry. However, real time implementation of

path planning and environment learning are equally important for a mobile robot to do something useful, such as acquiring pallets, transporting materials and cleaning warehouses. But most path planning and environment learning algorithms today are intrinsically inefficient and unsuited to real-time operation. As a result, most mobile robots built to date are far from having the fully autonomous competence needed for reliable and useful industrial applications.

A crucial problem in building such an intelligent mobile robot is how to provide it with the capability to deal with uncertainty in the environment and with errors in the robot itself. A solution to this problem needs to address two basic issues: (i) the use of multiple sensors and data fusion algorithms to reduce uncertainty [Durrant-Whyte *et al.*, 1990]; and (ii) the design of a stochastic planning and control strategy to make decisions with imperfect information. Since the real world is so complicated and no sensor is perfect, the problem of how to handle uncertainty is still an unsolved challenge in mobile robotics. This is the motivation for the research presented in this thesis.

## 1.2 Objectives

The experimental work reported in this thesis is based on a small research version, namely *Turtle*, of commercially-available, free-range AGVs that can acquire pallets and clean warehouses [Brady *et al.*, 1990]. The robot vehicle has three wheels and the front wheel is driven and steered. It follows routes stored in a computer in the form of spatial world coordinates which are established by a landbase computer that models the pathways and places in a factory or warehouse, as shown in Figure 1.1. It is equipped with a rotating infrared laser scanner that reads bar charts attached to the factory wall. An accurate location information is obtained by integrating odometry data and scanning data. The commercial version of the Turtle has been successfully deployed in a number of industrial plants.

However, like every other mobile robot mentioned above, the Turtle mobile robot is limited in

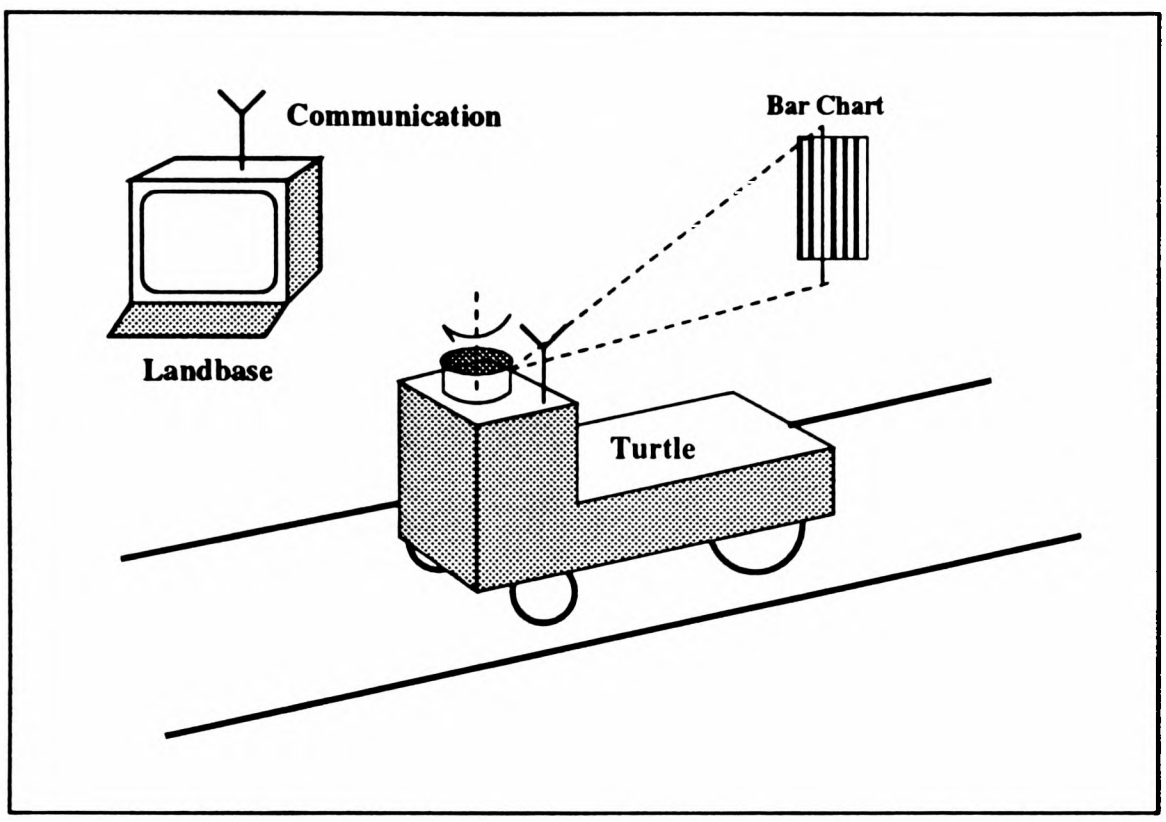



Figure 1.1: The *Turtle* mobile robot controlled by a landbase station via radio link

scope, for example:

- There is no on-board sensor to detect dynamic changes in its environment except for a plastic bumper that is used for emergency stop.
- There is no planning capability on board and it can only follow the path generated by the landbase.
- There are no decision-making capabilities to cope with uncertainty.
- There is not enough computer power on board to implement complex tasks towards fully autonomous competence.

The primary objective of the research in this thesis is to provide such a mobile robot with sufficient local intelligence that it is able to sense its environment, find an optimal path, avoid unexpected obstacles, and make decisions under uncertainty. For this reason, efforts have been made to integrate multiple sensors, dynamic planning and real-time control algorithms with a distributed transputer architecture to realise fast connection between sensing and action. This in turn poses a key issue in this thesis: system integration.

## 1.3 Proposed Approach

In this section, the approach taken in this thesis to achieve the objectives outlined above is briefly described as follows:

**Bayesian Approach to Planning with Uncertainty**

Mobile robots, in most industrial applications, operate in a dynamically changing environment. The changes in the environment are caused by people, objects, and other robots which are normally unknown *a priori*. In the face of this uncertainty, planning a collision-free path is obviously difficult since the internal world model becomes invalid quickly. Given that global optimality of the solution is desirable, we have to answer the following questions:

- What is an optimal path satisfying the given constraints on the mobile robot and its environment?
- How can we quantify the uncertainty to assign a merit to a path?
- How can we build and update internal world model dynamically?
- How can we plan an optimal path in real time?

To answer these questions leads us to investigate a probabilistic approach to the problem of planning with uncertainty. A key issue is to build the statistical models for encountering unexpected obstacles since the application of statistics to planning problem enables the uncertainty to be assigned a measure, and alternative paths to be compared quantitatively.

**Decision Theoretic Approach to Collision Avoidance**

To handle dynamic changes in the environment, multiple sonars are built and used for obstacle detection. The direct range return by a sonar sensor makes the sensor extremely easy to use. However, it has some inherent limitations, including poor directionality and specularity. These limitations give rise to uncertainty in the real world by providing incomplete observations and makes the interpretation of sonar data very difficult. Typically, in obstacle detection,

- Sonar may fail to report an obstacle that is actually in the way.
- Sonar may announce the presence of an obstacle when none occurred.

A probabilistic model of the uncertainty of observation data is obviously required for real-time interpretation. This model in turn tells about what is to be expected from these sensors. Based on imperfect sensory information, the collision avoidance algorithm should have the capabilities to provide solutions on:

- How to accommodate uncertainty in collision detection.
- How to choose an optimal action based on imperfect state information.

Given that the solution we are looking for is not only collision free but also optimal in terms of time cost, and constrained by the geometry of the road way and the vehicle kinematics. The collision avoidance problem is formulated using decision theory in this thesis.

**Optimal Tracking and Nonholonomic Constraints**

Given that the motion of the mobile robot should be limited within predefined pathways and places for safe operation in industry, the next questions to be examined are:

- How a smooth trajectory is dynamically generated for the mobile robot that is subject to nonholonomic constraints?
- How should the robot track the trajectory accurately and smoothly in the face of unmodelled disturbances?

The planned trajectory should be dynamic in the sense that it is easy to change according to the changes in the real world. Instead of a conventional PID controller, an optimal controller is investigated to track a planned trajectory in a predictive manner based on a quadratic cost function.

**Transputer Architecture to System Integration**

The complex tasks of a mobile robot are normally divided among different components in a control

architecture. In turn, how this is done reveals a great deal about how the architecture will function. In general, to achieve a real-time architecture for a mobile robot in real world applications, timing constraints are critical, and the connection between sensing and action should be fast. Building such a control architecture by integrating conventional computers faces problems of bandwidth, communication, flexibility, and ease of implementation. A transputer architecture offers the promise of higher performance than conventional computers. But the questions are:

- How it should be built to support planning and control tasks described above?
- How it should be structured to realise fast connection from sensing to action?

## 1.4 Methodology

To solve these problems, the following methodology has been adopted in this thesis:

**Separation of Estimation and Control**

In the face of uncertainty, the planning and control problem addressed in this thesis is stochastic in nature. Since an optimal solution (suboptimal in practice) is proposed, the problem becomes a stochastic optimal control problem. Generally speaking, this problem is considerably more complex than typical deterministic optimal control problems both from an analytic and a computational point of view. However, the *Separation theorem* tell us that the problem of seeking a linear control law for a linear dynamical system with Gaussian disturbance and measurement noise subject to a quadratic performance index (namely a LQG system) can be cast in terms of two separate problems: (i) deterministic optimal control; (ii) stochastic optimal estimation. It has been shown that these two problems can be solved separately to yield an optimal solution to the combined control problem [Bertsekas, 1976]. Although this separation property does not hold for nonlinear systems, in many cases, practical applications proceed after linearisation as if it did. This has been adopted in this thesis by designing controllers (planners) and state estimators separately and combining them to obtain a complete control system for dynamic planning and optimal control tasks. For instance,

the global planning algorithm in Chapter 3 adopts deterministic dynamic programming to find an optimal path based on a minimum cost criterion in which all random quantities are replaced by the expected values estimated by recursive models that are derived using Bayes' theorem.

**Complementarity of Planning and Control**

The key issue in this thesis is to integrate both the planning and control capabilities into a distributed control architecture. This requires clarification of the relation between planning and control. It is worth noticing here that the optimal planning problem in this thesis is viewed from the perspective of an optimal control problem and *vice versa*. This is because at a fundamental level, planning and control address the same problem: choosing actions over time to influence a process, based on some models of that process [Dean and Wellman, 1991]. For example, planning in this thesis is dynamic in the sense that it monitors sensory information to search for an optimal course of actions dynamically, which is the perspective of a control problem. On the other hand, prediction in optimal control strategy addressed in Chapter 5 can be viewed from the perspective of planning. This in turn leads to a better understanding of relation between planning and control to integrate a system.

**Distributed Decision Making**

Sensor-based control of the mobile robot we investigate in this thesis includes many different decision-making tasks such as interpreting sensor data, planning a global path, avoiding obstacles, and tracking a trajectory. In order for the mobile robot to respond rapidly to dynamic changes in its environment, all of these decision making tasks are distributed into different modules, which we call local intelligent control agents (LICA). Each LICA is assigned a certain responsibility such as path planning and obstacle avoidance. All LICAs in the system communicate with each other in order to achieve their own objectives and the objectives of the system as a whole.

**Construction of Simulation Model**

A simulation model of our AGV Laboratory and the mobile robot is built in the SunView Graphics environment that can run on any SUN3/4 workstation, as shown in Figure 1.2. The simulation

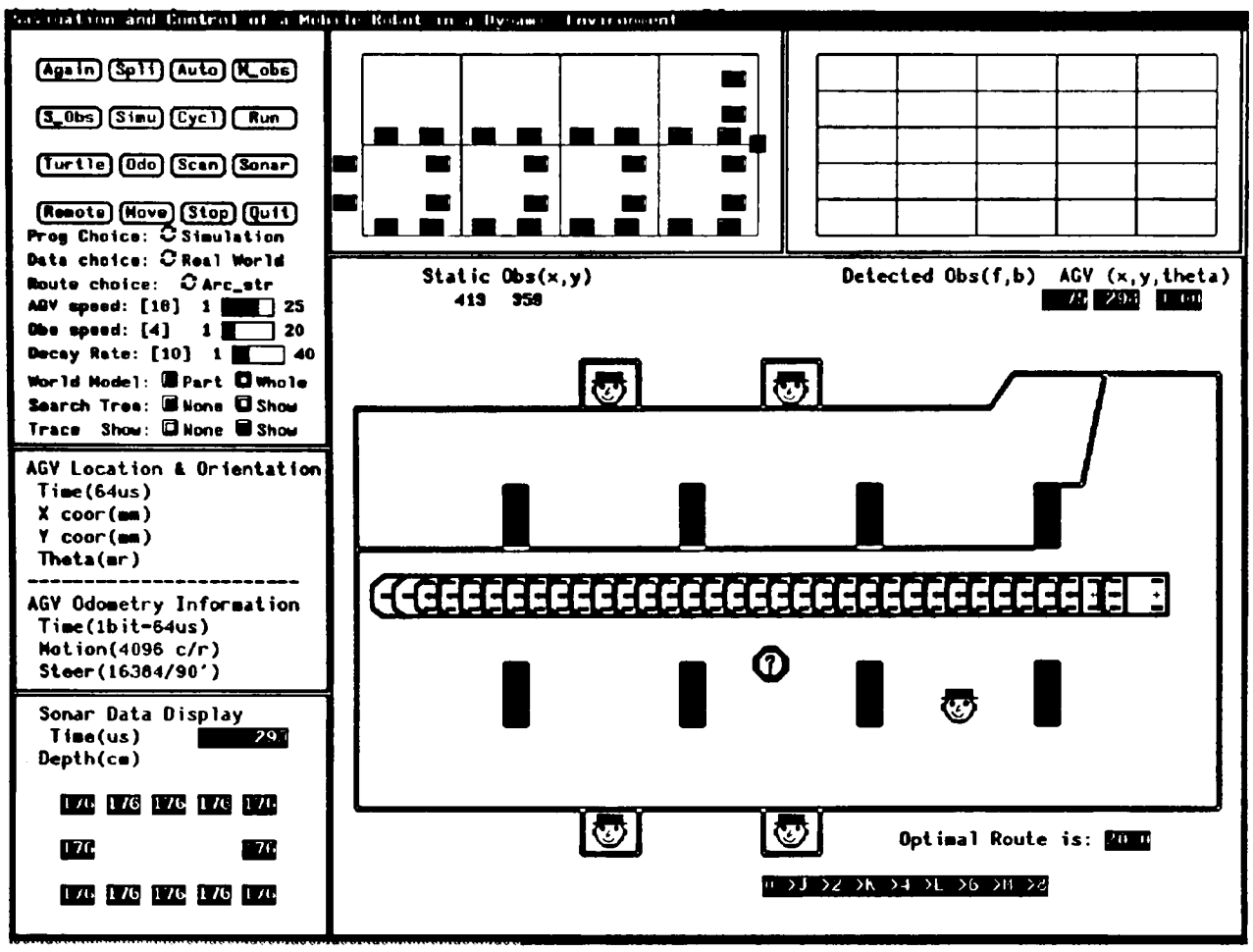



Figure 1.2: A simulation model run on a SUN workstation

package is written in ANSI C language. Different planning and control algorithms are easily tested and verified in this simulator. Since the simulator is able to communicate with on-board transputers via RS422 differential links, the verified algorithms can be directly down loaded to on-board transputers to run in real time. Also, implemented results and data can be collected and stored for further analysis. This obviously provides a convenient environment for the design of the whole control system.

## 1.5 Thesis Outline

In Chapter 2, we first review some design principles which are used currently in the mobile robotics community. Then, a distributed real-time architecture which merges some of the better features of both the hierarchical and subsumption approaches, is presented in detail. In our design, complex control tasks of the mobile robot are distributed among a set of behavioural layers that tightly couple

sensing and action, and which are also loosely coupled with each other. Each behavioural layer is provided with a significant amount of self-knowledge to allow it to perform a specific task. It has been implemented on state-of-art parallel processors: a transputer network, and in turn possesses many desirable properties: modularity, flexibility, and high bandwidth.

Chapter 3 presents a global path planning algorithm that we designed for our Turtle mobile robot to be able to operate in a dynamically changing environment. It is dynamic in the sense that the optimal paths are continuously generated when the mobile robot travels around. To cope with unexpected obstacles that are assumed to be randomly present, a probabilistic approach is adopted. The cost of handling obstacles in any portion of the path that the robot travels along is quantified as an uncertainty cost (expected value). Then, the planning algorithm takes these costs into account to search an optimal path accordingly using a dynamic programming algorithm. The environment model is updated continuously by available sensor data when the robot is moving so that it is always valid.

Chapter 4 investigates how the Turtle mobile robot can avoid unexpected obstacles in a dynamic environment. An array of 12 sonar sensors has been used to detect obstacles that are assumed static and random. Although under appropriate circumstances sonars are able to provide relatively accurate range information, they often fail to do so because of specularity and multiple reflection. The wide beam-width of a sonar sensor is useful to detect an obstacle at some distance. However, this causes difficulties to detect the size of the obstacle. Therefore, a probabilistic sensor model and rule-based heuristics are developed to interpret data produced by the three front sonars. A new collision avoidance approach based on Bayesian decision theory is conducted and its success is demonstrated both by simulation and on the mobile robot.

In Chapter 5, the discussion turns to the trajectory generation and tracking problem. This is natural since the path planned by the global and local path planners is a set of intermediate points that can not be used directly to control the motion of the robot. A precise geometric trajectory

with a velocity profile needs to be created, taking into account the robot kinematics. Like other car-like mobile robots, the Turtle robot is subject to Nonholonomic kinematic constraints. It has a lower bound on its turning angle and needs a continuous-curvature trajectory. After a brief review of different schemes that have recently been used, we propose a new guidance system consisting of a trajectory generator and a tracking controller. Continuous-curvature trajectories subject to constraints of the roadway can be dynamically generated. An optimal tracking controller is used to track these trajectories to minimise quadratic cost functions.

Finally, Chapter 6 concludes this thesis with a brief summary of what has being achieved so far and the agenda for future research. In addition, the original control system of the Turtle mobile robot is included in Appendix A. The review of the relevant literature is distributed throughout the thesis.

## 1.6 Main Contributions

Based on the approach described above, the thesis has presented theoretical and practical solutions to the problem of how a mobile robot can perform real-time planning and control tasks in the face of uncertainty. It has made a major advance in allowing the Turtle mobile robot to respond to environmental changes autonomously. In particular, the main contributions made in this thesis can be summarised as follows:

- A distributed real-time architecture is developed to support dynamic planning and real-time control tasks. It is a distributed community of sensing, planning, and action modules based on transputer technology. Its design maximises the amount of parallel information flow from sensing to action so as to provide minimal delay in responding to a constantly changing environment. A distributed map building is adopted to provide different information for different decision making processes in terms of time constraints.

- A new global path planning algorithm is developed to handle dynamical changes in the environment. It is a Bayesian approach to planning with uncertainty. It monitors dynamic changes in the environment and makes prediction of the new path, differing from a traditional path planner that is open-loop in nature. To quantify uncertainty, statistical models for encountering unexpected obstacles are built to assign a merit to a path. These models are recursively updated by available new observations when the robot moves around. This in turn provides a feasible way to find an optimal path and learn global knowledge.
- A probabilistic sensor model and rule-based heuristics are developed to interpret sonar data and reduce uncertainty. They provide a feasible way to achieve real-time performance of collision avoidance.
- The collision avoidance problem has been cast into the framework of decision theory. A recursive decision-making algorithm is given to provide a feasible way to perform collision avoidance with imperfect sensory data in real time. Both collision free path and minimum time cost are achieved.
- A new guidance system is proposed to provide dynamic trajectory planning and optimal tracking capability to a nonholonomic mobile robot that is expected to navigate in an environment with dynamical changes. The generalised predictive control strategy is adopted to achieve smooth tracking performance in the face of unexpected disturbances.

# Chapter 2

# System Design: Sensor-based Control

In order to achieve a high degree of autonomy and robustness, a mobile robot should be able to sense its surroundings, build or update maps, plan and execute actions, and adapt its behaviour to environmental changes. To implement these tasks demands a sensor-based control architecture to provide for unified action in real time. In this chapter, we describe the system design of a real-time, sensor-based control architecture for our mobile robot. A distributed design for sensing and control, as well as implementation on a network of transputers are presented.

## 2.1 Introduction

Sensor-based control provides the capabilities for a mobile robot to interact intelligently with a dynamic environment while performing specified tasks. To support such a sensor-based control strategy, many distributed architectures have been developed recently in mobile robotics, ranging from very specific ones for a particular mobile robot to very general ones that are intended to be suitable for a wide variety of applications. However, no specific architectural approach can be identified as best so far [Harmon, 1991]. Therefore, the design of a suitable, robust and reactive architecture remains a key challenge in robotics.

The Turtle mobile robot we investigate is controlled by an industrial programmable controller, the GEM80 [Rugby, 1986]. It has a tightly coupled multiprocessor architecture in which all processors share a common memory. It is also very hard to extend its capabilities by adding more computing power to perform sensing, planning, and control tasks intelligently and autonomously.

More details can be seen in appendix A.

In general, as the capability of a mobile robot increases, the number of sensors, motors and processing components that must be integrated and coordinated also increases. Therefore, a distributed computing architecture offers a number of advantages to aid in coping with the significant design and implementation complexity inherent in sophisticated mobile robot systems. It is often cheaper and more resilient than the alternative uniprocessor designs. Also, multiple, possibly redundant, processors provide the opportunity to take advantage of parallelism for improved throughput and fault tolerance. The modularisation necessary for a distributed implementation often simplifies the difficulties of complex hardware and software design as well as debugging. The most important aspect of a system design is the need to meet real-time constraints, which is considerably easier given the concurrency inherent to distributed computing solutions. Fortunately, transputers provide more efficient ways to build such distributed architectures. In this chapter, we present how control and sensing are distributed among a network of transputers.

In Section 2, we review previous approaches to control architectures of mobile robots. Then the distributed architecture designed for the Turtle mobile robot is given in Section 3. Section 4 describes why the INMOS transputer is selected to build a real-time architecture to implement complex planning and control tasks. Section 5 investigates how to map these tasks into a transputer network based on a modular approach. Section 6 describes a development environment that is set up for debug and data logging. Finally, brief conclusions are given in Section 7.

## 2.2 Previous Work

During the last two decades, the research community in mobile robotics has paid a lot of attention to the development of different control architectures. Two principle designs have been adopted. One is called the functional, or horizontal, decomposition; the other is the behavioural, or vertical, decomposition [Brooks, 1986].

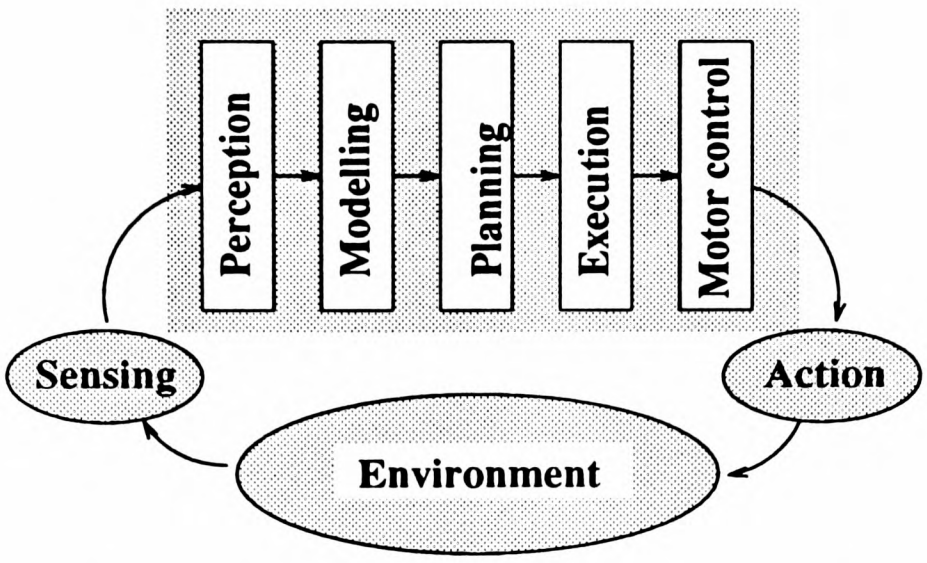



Figure 2.1: Horizontal decomposition of the control system

Functional decomposition is a classical top-down approach to building systems. In this approach, the entire control task of a mobile robot is divided into subtasks which are then implemented by separate modules. These functional modules form a chain through which information flows from the robot's environment, via the sensors, through the robot and back to the environment via actuators, closing the feedback loop, as shown in Figure 2.1.

Most previous mobile robots have been based on this approach. Both hierarchical and blackboard architectures have been used, such as the Shakey robot at SRI [Nilsson, 1984], the Hilare robot at LAAS [Giralt *et al.*, 1984],[Noriels and Chatila, 1989], the Alvin robot at HAIC [Daily and *et al*, 1988], the Ground Surveillance Robot at RII [Harmon, 1987], the CMU Rover [Elfes and Talukdar, 1983], the IMP robot [Crowley, 1985], and Navlab robot [Goto and Stentz, 1987]. The hierarchical architecture is a highly deterministic and allows for the coordination of problem solving strategies [Paul *et al.*, 1985]. It naturally decomposes a large complex system into small modules for relative ease of implementation. More details can be found in [Albus *et al.*, 1981], [Albus *et al.*, 1982], [Herman and Albus, 1988]. The blackboard architecture includes a central database, a pool of knowledge-intensive modules, and a system manager which synchronises the modules. It provides a very powerful mechanism through which to fuse various types of sensor data to realise

a loosely coupled computer network. However, both a hierarchical and a blackboard architecture have limitations. A pure hierarchy precludes interaction with low-level sensing and action processes at higher levels and tends to result in very structured decomposition of problems leading to overall longer processing times and poor use of sensory information [Orlando, 1984]. Its strictly structured decomposition makes it difficult to deal with uncertainty. This sequential structure requires an instance of each module to be built in order to run the robot at all. The blackboard structure has the problem that the shared database may cause bottlenecks in access, imposing a limit on the bandwidth of the whole system.

In contrast, behavioural decomposition is a bottom-up approach to building a system. A behaviour encapsulates the perception, explore, avoidance, planning, and task execution capabilities necessary to achieve one specific aspect of robot control. Therefore, each of them is capable of producing meaningful action which in turn can be composed to form levels of competence [Brooks, 1987a]. In other words, each of them realises an individual connection between some kind of sensor data and actuation. The system is built step by step from a very low level, say from locomotion, to obstacle avoidance, to wandering. Successive levels can be added incrementally in order to enhance the functionality of the robot. Figure 2.2 shows this decomposition scheme.

This design method has grown in popularity recently. Examples include Brooks' subsumption architecture [Brooks, 1987b], Anderson and Donath's emergent behaviour [Anderson and Donath, 1990], Kaelbling's intelligent reactive system [Kaelbling, 1987], and Arkin's schema based architecture [Arkin, 1989]. In the subsumption architecture, control is distributed among those task-achieving behaviours that operate asynchronously. Lower layers can subsume the operation of the higher ones when necessary, only one layer at a time actually controlling the robot. Since each layer is to achieve a single task, it requires only the limited information that is useful for its operation. It is claimed that the control system can respond rapidly to dynamic changes in the environment without the delays imposed by sensor fusion. But the implementation of higher layers

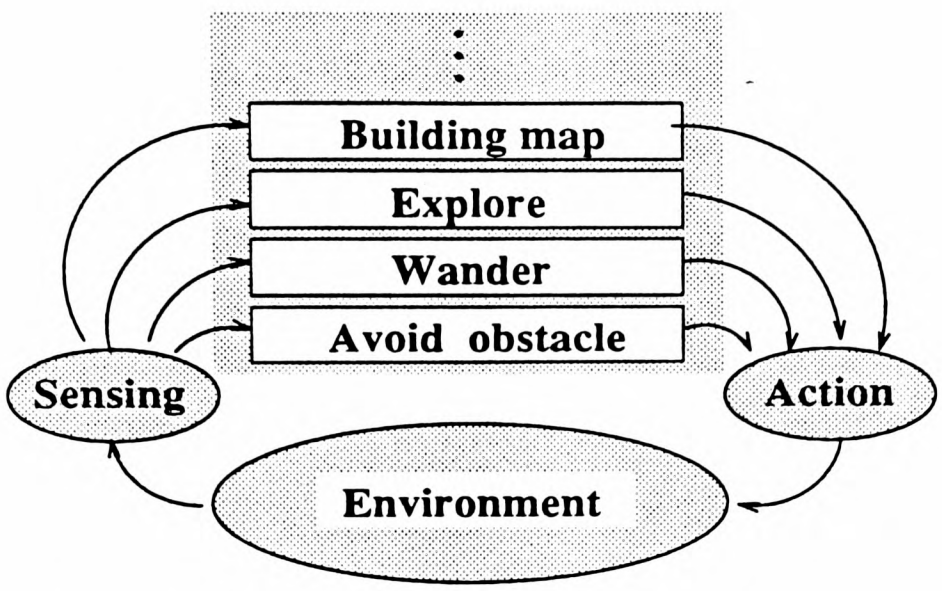



Figure 2.2: Vertical decomposition of the control system

of competence still poses a problem. More careful initial design in specifying the communications and modularity is required. Moreover, the higher levels often rely on the internal structure of lower levels, thus sacrificing modularity. In Kaelbling's architecture, control is broken down into levels that are similar to Brooks's. The key difference is that each level consists of a set of parallel decision modules whose output are fed to a mediation unit. Arkin develops motor and perceptual schemas each of which is an individual computing agent. His approach is strongly influenced by cognitive psychology and neurophysiological studies. Although it shares the viewpoint of the necessity of behavioural expression and the importance of the immediacy of sensory information, Arkin's approach differs from Brooks' subsumption architecture by avoiding layering entirely; instead setting up a dynamic network of schemas based on the current goals of the robot. Control of the robot is achieved through schemas which implement basic behaviours. To ensure a high degree of concurrency, schemas communicate through a blackboard. The simulation result has shown the feasibility of this approach, but it is necessary to find better ways for real time applications [Lawton *et al.*, 1990].

We believe that the control task associated with a mobile robot is so complicated that we cannot strictly follow one decomposition scheme while completely ignoring the other. In fact,

every approach has both benefits and limitations. A compromise, or combination, of both schemes may be a better choice to build more robust, flexible, and high bandwidth architectures for mobile robots. The control architecture we designed and implemented for the *Turtle* mobile robot is a hybrid structure which merges positive features of both the hierarchical and subsumption approaches. It is called a distributed real-time architecture(DRTA), and is based on state-of-art parallel processors: a transputer network [Hu *et al.*, 1991c]. A similar hybrid approach has recently been adopted in other mobile robot projects. For example, Badreddin proposes a *recursive nested behaviour control* structure in [Badreddin, 1991] which is traditional nested control where the behaviour levels implement feedback and feedforward compensations. Knieriemen and Puttkamer suggest an *orthogonal control* structure with cascaded control loops, combining aspects of hierarchical and behavioural structures in [Keieriemen and von Puttkamer, 1991]. Mitchell presents a Theo-Agent architecture which combines a stimulus-response subsystem for rapid reaction, with a search-based planner for handling unanticipated situations [Mitchell, 1990]. The key issue of hybrid approach is to merge positive features of the first two approaches.

## 2.3 Distributed Design

The control architecture of our mobile robot has been designed to satisfy the following criteria:

- Robustness – to cope with the inaccuracies and uncertainties of sensor data as well as with hardware errors like communication failures.
- Modularity – to distribute sensing and control among different modules for task-achieving behaviours.
- Expandability – to build up the entire system stepwise from low-level behaviours like obstacle avoidance to high-level behaviours like path planning.
- High bandwidth – to realise dynamic planning and reactive control in real time.

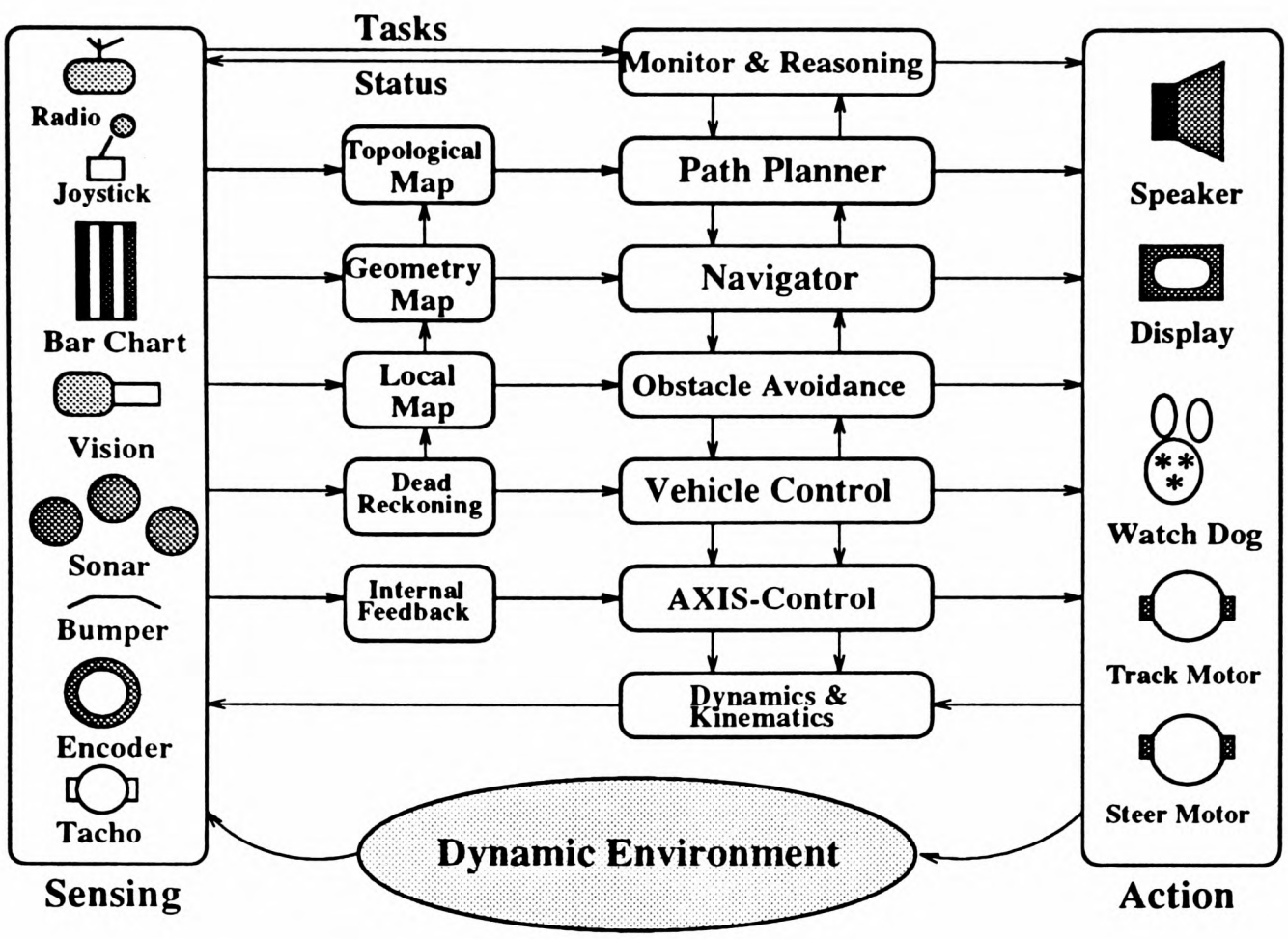



Figure 2.3: Block diagram of the control architecture

- High intelligence – to cope with uncertainty in a dynamically changing environment, including:
  - the ability to generate a solution in unexpected situations.
  - the ability to learn something new from its travels.
  - the ability to communicate with other robots and humans.

Figure 2.3 shows a block diagram of the control architecture. It consists of a distributed community of sensing, action and reasoning nodes. Each of them has sufficient expertise to achieve a specific subtask, following a hierarchical decomposition scheme. Few of them are composed of a single task-achieving behaviour as in a behavioural decomposition scheme. The key point is that the control task of the mobile robot is distributed among a set of behaviour experts that tightly couple sensing and action, and which are also loosely coupled to each other. In this way, particular sensor data can be used directly in a corresponding layered control task to form a task-achieving

behaviour. This differs from the functional decomposition in which the combination of subtasks is a single processing chain. To function effectively, each layer requires a significant amount of self-knowledge to allow it to decide what information it can supply to others; how best to recover from local errors; as well as what to do if no information is sent from other layers.

Basically, there are two subsystems in our design: a layered perception system and a layered control system. The layered perception system is used for active sensor control and distributed sensor fusion to support a layered control strategy. The layers process raw sensor data from internal sensors (tachometer, encoder, resolver) and external sensors (sonar, vision) to build up models in a bottom-up manner. All the sensing layers operate independently and are loosely coupled. Communication between them is used to realise sensor data fusion. The design of the layered controller is based primarily on the observation that different response times are demanded by different tasks. The lower levels perform simple, more general tasks such as *smooth path guidance* and *avoiding obstacles* for fast reactivity. The higher levels perform more complex, situation specific tasks such as *path planning* and *monitoring*. All layers are intended to operated in parallel.

Each layer of our DRTA consists of a control node and a sensing node. Each sensing node delivers a representation, which, in the context of a given task, causes a corresponding control node to generate commands. Here, we try to avoid a centralised and complicated model for a dynamic environment since it needs more time to compute, thereby reducing the system's response speed. Instead, we are employing several simple models, each tuned to a different range and resolution of situations for different tasks. In other words, this multi-model control architecture takes the real-time capability and the expected task-achieving behaviours into account. In more details:

- the top layer (Monitor & Reasoning) is used mainly for communication tasks which include receiving and interpreting mission tasks, $t_i \in \mathrm{T}$, from a planning center and reporting the operation status back to it. Also, it tells the operation status to humans via a speaker.

- The next layer (Path Planning) works on a topological map to determine the subgoals to reach an assigned task. At this level of resolution, the world representation consists of a network of distinctive places such as rooms, corridors, junctions, etc. It forms a predefined graph with a cost function on each arc based on prior information, which is then updated according to sensor data. An optimal path, $\mathbf{n}_i \in \mathcal{N}$ between the start node $\mathbf{n}_s$ and the goal $\mathbf{n}_g$ can be planned based on this graph.
- The task of the third layer (Navigator) is mainly to update the robot position. A rotating infrared laser scanner situated on the top of the vehicle is used to read retroreflective bar charts that are attached to the factory wall at known locations. The position measuring algorithm is capable of simultaneously accepting data provided by a laser scanner and by odometry. Since both set of data, the odometry data $(\alpha, \omega)$ and the measured angle $\lambda$ to each bar chart, are noisy, a Kalman filter is used to estimate the position $(x, y)$ and orientation $\theta$ of the robot in real time.
- The fourth layer (Obstacle Avoidance) uses sonar data to build a local map by linear approximations of range and azimuth information, and provides the capability to avoid obstacles which are unknown to (or ignored by) the Navigator.
- The vehicle control layer below obstacle avoidance plans a continuous-curvature trajectory and maintains the continuity of motion along this trajectory to achieve smooth guidance and stability of overall system.

Traditional methods in control stress the use of feedback which has been naturally adopted in our layered design, and described in Figure 2.4 in the way of a control system design. Note that the planning problem is viewed from the perspective of a control problem in the design. To implement this layered control strategy, we describe a real time computer architecture in the next section.

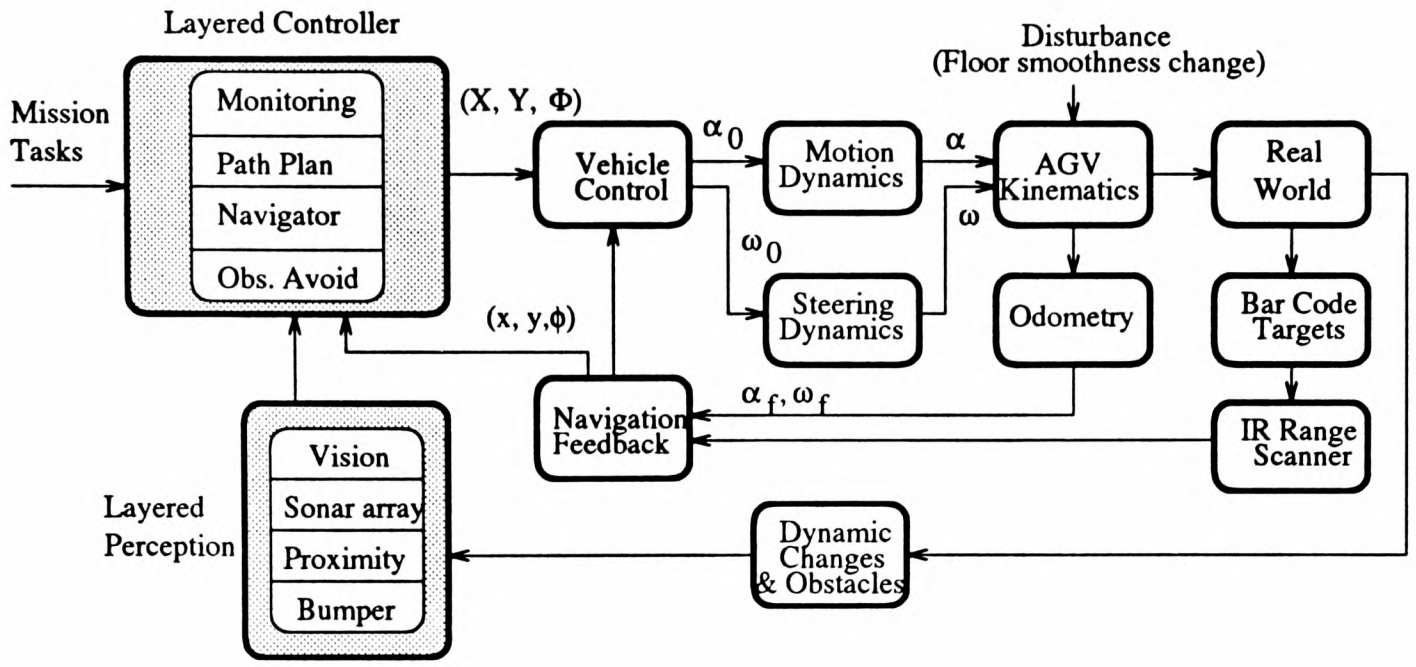



Figure 2.4: Control strategy for the *Turtle* mobile robot

## 2.4 Real-time computer architecture

The key feature of the DRTA we have described is that all the layers in the system should operate in parallel and be loosely coupled to each other. This naturally demands a multiprocessor computer architecture since it is unlikely that all the robot's tasks, such as navigation, map building, planning, and control can be carried out concurrently by one processor. This multiprocessor architecture should be capable of real time, on board processing that is modular, reliable, and expandable. Also, low power consumption and the use of commercially available products are important features. However, multiprocessor control of a mobile robot is complicated and poses problems that don't exist in an uniprocessor system. Typically, the following factors must be taken into account in the design for a particular application:

- the degree of coupling.
- the configuration of processor interconnection.
- the method of deadlock detection and protection.

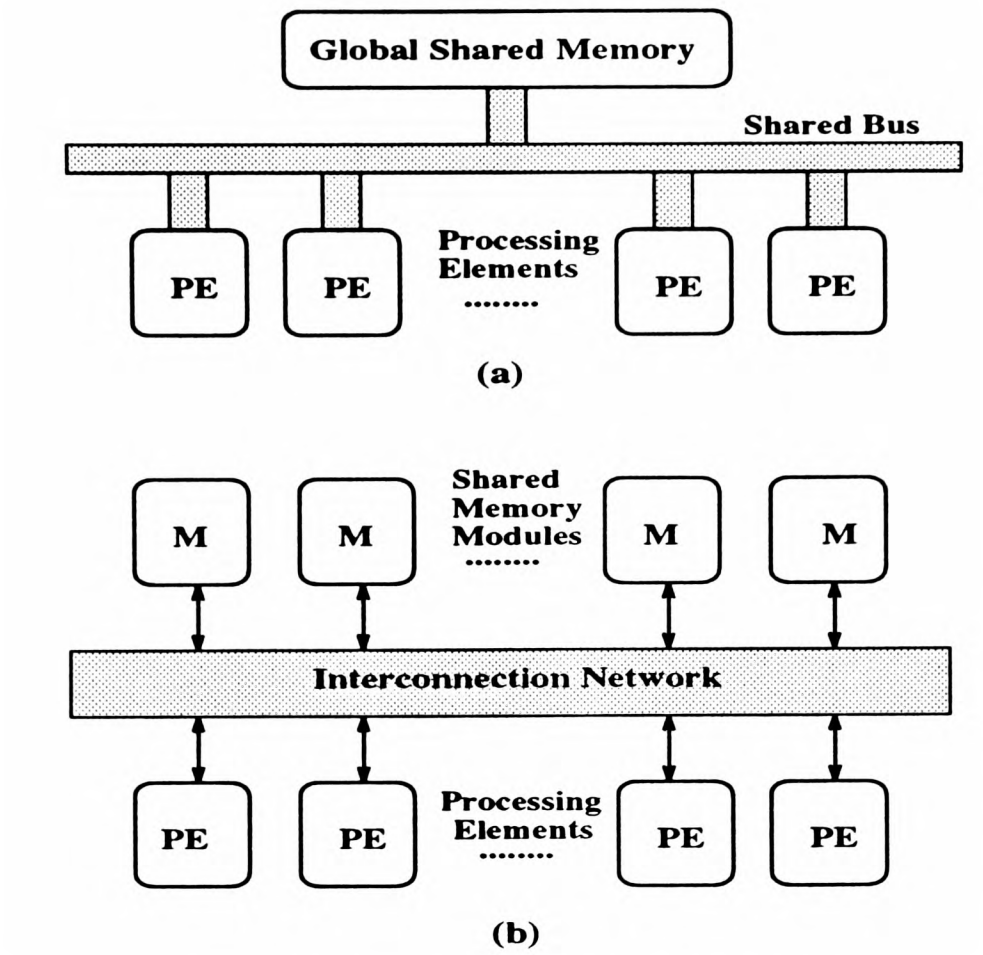



Figure 2.5: Tightly coupled multiprocessor

### 2.4.1 Conventional Approach

Multiprocessors are general-purpose, asynchronous parallel machines with multiple instruction-streams and multiple data streams [Flynn, 1972],[Hwang and Briggs, 1984]. They can be classified as being *tightly coupled* or *loosely coupled.* Processors in a tightly coupled multiprocessor communicate through a shared memory, even though they may still have their own private memory. In such an architecture, there are two ways to access the shared memory: via a shared bus (Figure 2.5(a)) or via an interconnection network such as a crossbar (Figure 2.5(b)).

In contrast, loosely coupled multiprocessors have no globally shared memory. Instead, processors communicate by sending and receiving messages, which typically use interconnection networks with direct, point-to-point connections between processors, as shown in Figure 2.6. In this architecture, communication between processors can be time-multiplexed through a common bus; switched through a limited number of channels, as in a crossbar scheme; or done through dedicated links.

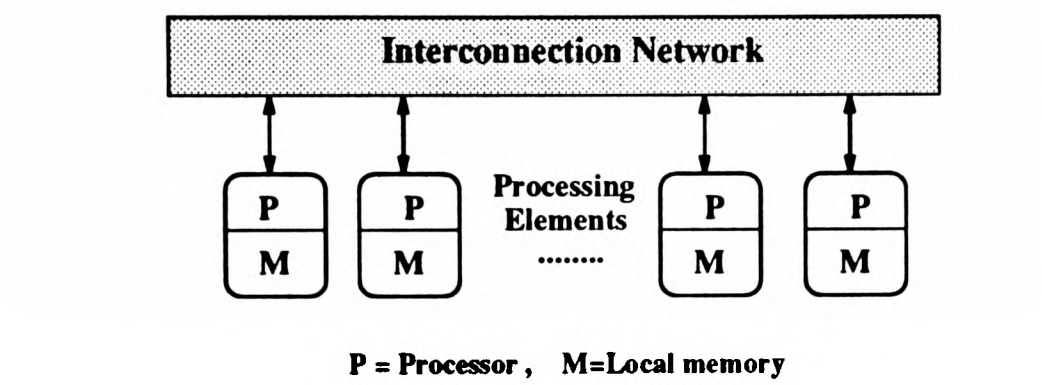



Figure 2.6: Loosely coupled multiprocessor

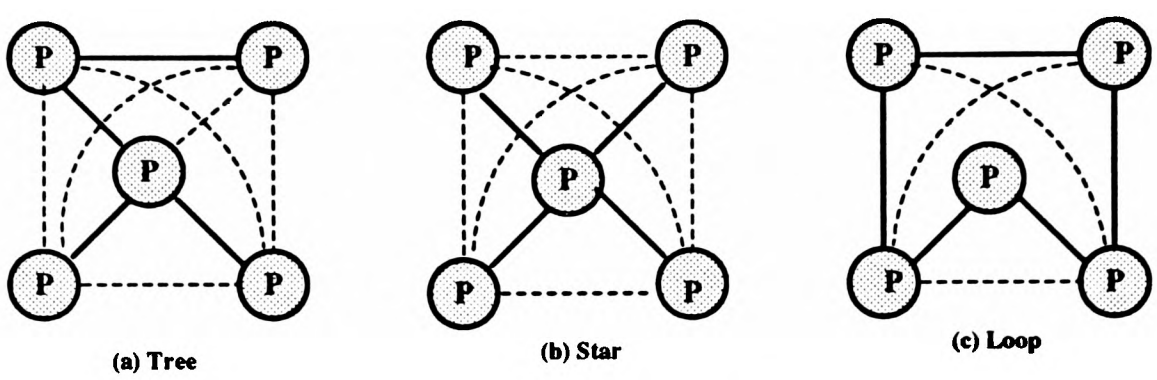



Figure 2.7: Different configurations with a fully connected system

This is also called a *distributed architecture*. Klein and Wahawisan proposed a loosely coupled system for the control of the OSU Hexapod vehicle, using an inexpensive, high density, eight-channel parallel line unit [Klein and Wahawisan, 1982]. They implement different configurations (tree, star, and loop) with a fully connected system, as shown in Figure 2.7.

To build these multiprocessor systems using conventional processors is very difficult both in terms of the implementation of the hardware and the software. In the hardware implementation, communication between processors has to use shared memory, a common bus or any other communication facility since conventional processors follow the *Von Neumann* architecture that is sequential in nature (the instructions and data share a common path and are fetched into a central processing unit (CPU) one by one). The lack of any special communication unit on-chip for communication with other processors makes the implementation complicated and the configuration inflexible. It also leads to some serious limitations such as bus contention, which arises when more than a pair of

processors request communication in a shared bus system, or bottlenecks in access to the globally shared memory, or saturation of its bandwidth as the size of the system increases.

As regards software implementation, a major obstacle is the lack of convenient parallel programming languages. The languages currently available on multiprocessors are essentially existing sequential languages (C, Fortran, Pascal) with a few extensions for parallelism, such as Concurrent C [Cmelik *et al.*, 1986], Parallel Fortran [Allen and Kennedy, 1982], and Concurrent Pascal [Hansen, 1975]. As a result, difficulties exist for fully exploiting the potential parallelism provided by the hardware. This leads to systems that are highly obscure and unportable. Also, parallel hardware may not be able to implement efficiently certain constructs of the original sequential languages. For this reason, the trend appears to be towards languages with explicit parallelism and partition, such as *Ada and Multilisp* which offer portability but still force the users to decompose explicitly the program into tasks and to control their synchronisation and communication [Sarkar, 1989]. Another major problem in concurrent programming is *deadlock*, in which all affected process are suspended indefinitely waiting for one another. It is extremely unlikely that compilers will be intelligent enough to detect deadlock. It is thus necessary for designers to demonstrate that the possibility of deadlock has been removed from their software [Burns, 1988]. This is a tremendous burden [Sarkar, 1989].

### 2.4.2 Transputer Approach

Considering the complex tasks and real time requirements of a mobile robot as well as ease of implementation, we have therefore developed a novel computer architectures for real-time sensing and control using a new generation of processors and languages that differ from the conventional ones described above. *Transputers* and the *occam* language provide a feasible way to achieve this [Inmos, 1989a].

**Transputer Technology**

The transputer is a single-chip microprocessor, containing all the major components of a conven-

tional SISD (single instruction single data) machine, sequential Von Neumann computer. It also has a on-chip memory and communication system built in. All of them are connected via a 32-bit internal bus. The bus also connects to the external memory interface, enabling additional local memory to be used. Moreover, the floating point transputers have an on-chip floating point unit with small size and high performance [Inmos, 1989b].

There are three main important features in a transputer: (i) it has a micorcoded process scheduler which enables any number of concurrent processes to be executed together and shared the processor time. The scheduler ensures that inactive processes (waiting to input, output, or timing) do not consume any processor time and active processes held on a list are executed one by one. Whenever a process is unable to proceed, its instruction pointer is saved in its workspace and the next process is taken from the list. (ii) communication between processes is achieved by means of channels which are point-to-point, synchronised and unbuffered. A channel between two processes executing on the same transputer is implemented by a single word in memory. In contrast, a channel between two processes executing on different transputers is implemented by point-to-point, on-chip links operating at a high speed (10 or 20 Mbits/s). (iii) A multiprocessor architecture built using transputers does not need external communication facilities. Instead, it can easily be built by connecting transputers through links, as shown in Figure 2.8(a). It can also be simply interfaced to external peripheral via the specially designed Link Adaptors, shown in Figure 2.8(b).

**Occam Language**

Occam is well known for its influence on the design of the transputer [Inmos, 1988]. As one of the first languages capable of automatic synchronised communication in concurrent programming [Pountain and May, 1988], occam provides explicit parallelism, partitioning and scheduling for parallel processing on transputer [Sarkar, 1989]. It is based on the concepts founded by David May in EPL (Experimental Programming Language) and Tony Hoare in CSP (Communicating Sequential Processing) [Hoare, 1978; Hoare, 1985], and is produced by INMOS [Inmos, 1988].

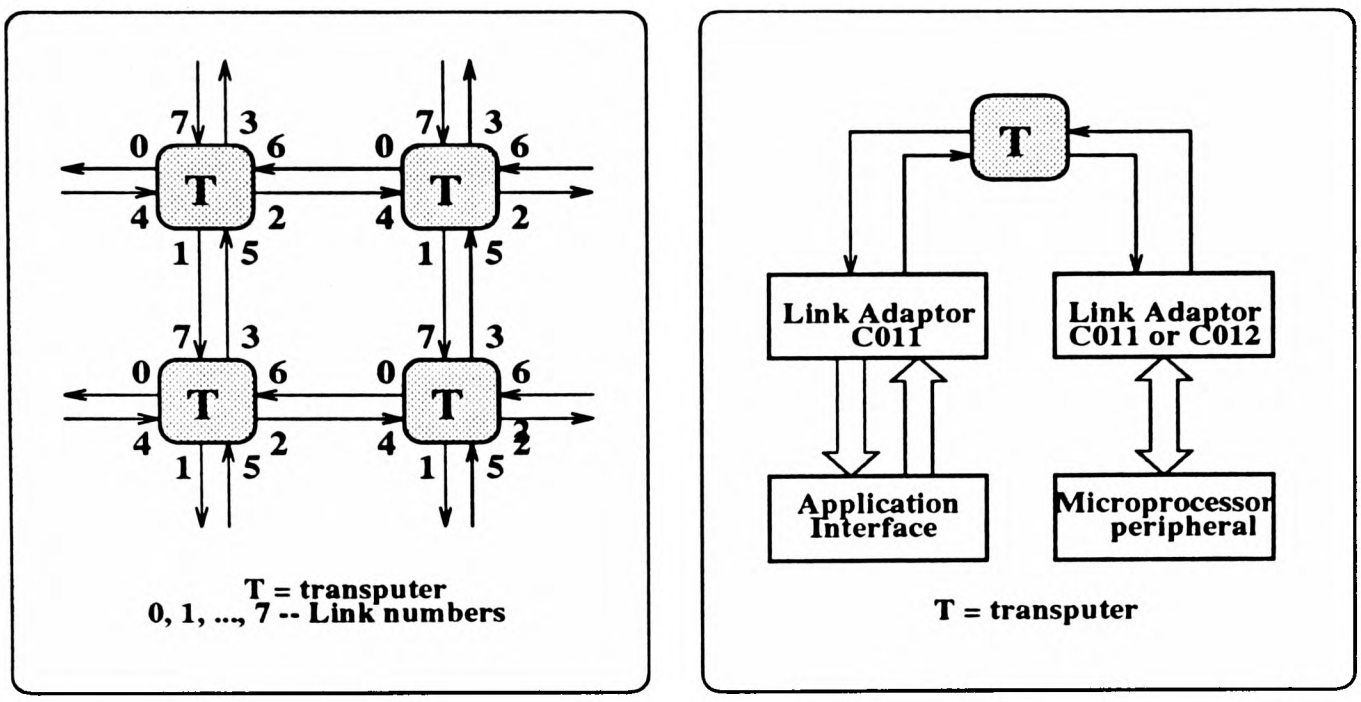



Figure 2.8: A Transputer network and peripheral interfacing

CSP is a formalism that can be used to prove the absence of potential deadlock. It is also an abstract generalised model of concurrency based on the idea of independently executing processes exchanging data with each other via one way connections called channels. The realisation of CSP in occam is so elegant that all the necessary synchronisation and control primitives for concurrency are closely coupled with the transputer instruction set. In other words, communication between different processes in occam is built into the language itself and automatically synchronised by channels. A channel can pass data between two occam processes that are running concurrently either on the same transputer or on different ones, which is synchronised in the sense that the input process will wait automatically for data from the output process, and vice versa [Jones and Goldsmith, 1988], [Burns, 1988].

Based on transputers and the occam language, we built a real-time computer architecture for our mobile robot to implement real-time sensing, planning and control tasks. We will give more details on how the complex tasks of a mobile robot are mapped onto a transputer network in the next section.

## 2.5 Mapping onto a Network of Transputers

The fundamental idea of our DRTA design is to distribute the complex tasks of a mobile robot into different sensing and control modules, and then combine them into several layered control loops to form different task-achieving behaviours in a bottom-up manner. In this section, we further address how to implement this idea on a network of transputers which will satisfy the performance requirements that were described in section 2.3.

### 2.5.1 Modular Approach

A modular design is the natural way to build a complex control system in terms of flexibility, expandability, robustness and the ease of implementation. Following this idea, we propose a transputer-based module, called a locally intelligent control agent (LICA), as a building block to implement our distributed sensing and control architecture [Hu *et al.*, 1991c].

Each LICA is composed of a motherboard and a few functional components: a standard TRAM (TRAnsputer Module), an interface, and a driver, as shown in Figure 2.9(a). Note that the number of TRAMs, and the size of the interface and driver, in each LICA is completely flexible, depending on which task this LICA is to perform. The board size of LICA is also flexible, according to the need of different applications.

There are three kinds of sizes of LICA board in the system we design: a standard single Eurocard, a standard double Eurocard and the commercially available PERIMOS MB8-TIO, as shown in Figure 2.12. The PERIMOS MB8-TIO is shown in Figure 2.10. Note that there are six TRAM modules on board, which are a RS232 TRAM(SIZE 2), a flash RAM(SIZE 2), two computing T800 TRAMs, a RS422 TRAM, and a EPROM TRAM in the order from left from right.

In general, each LICA is an expert in its own localised function and contains its own localised knowledge source. Figure 2.9(b) shows its logical function. An interface board has one or two C011 link adaptors to provide connections between the Inmos high speed link (10 or 20 Mbits/s)

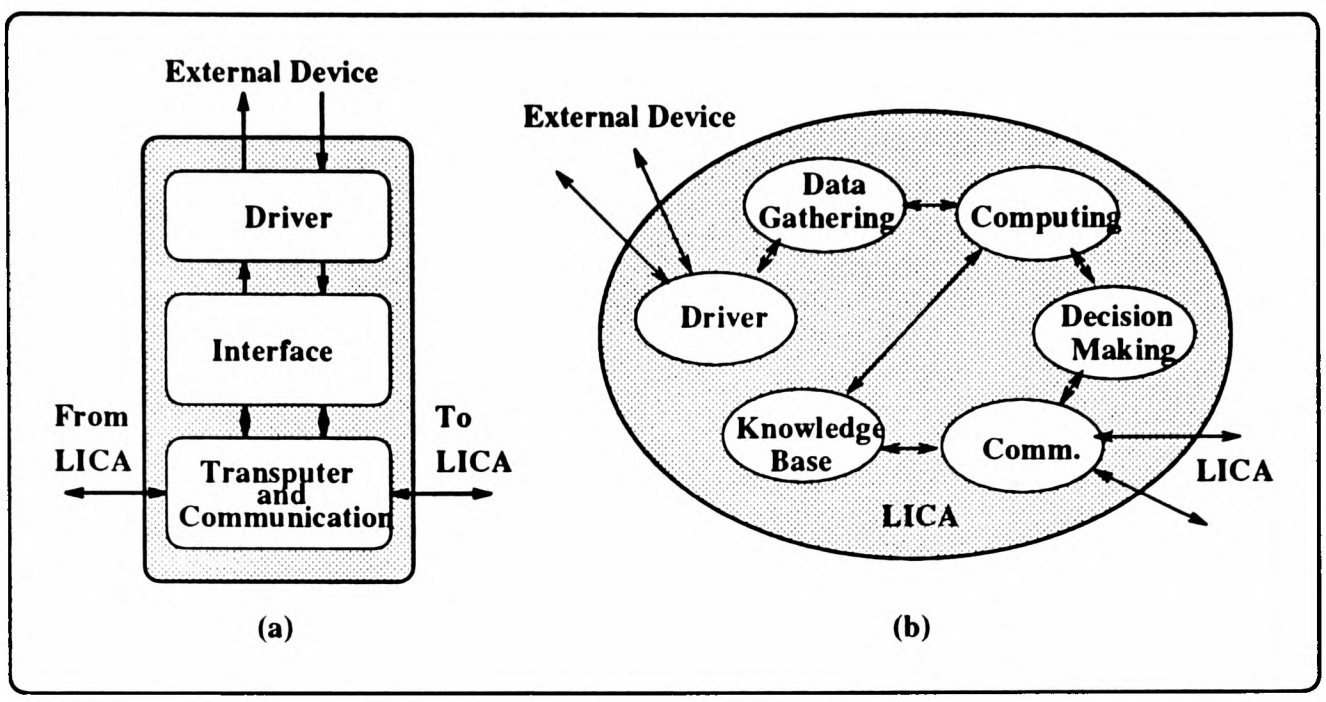



Figure 2.9: Block diagram of a Locally Intelligent Control Agent (LICA)

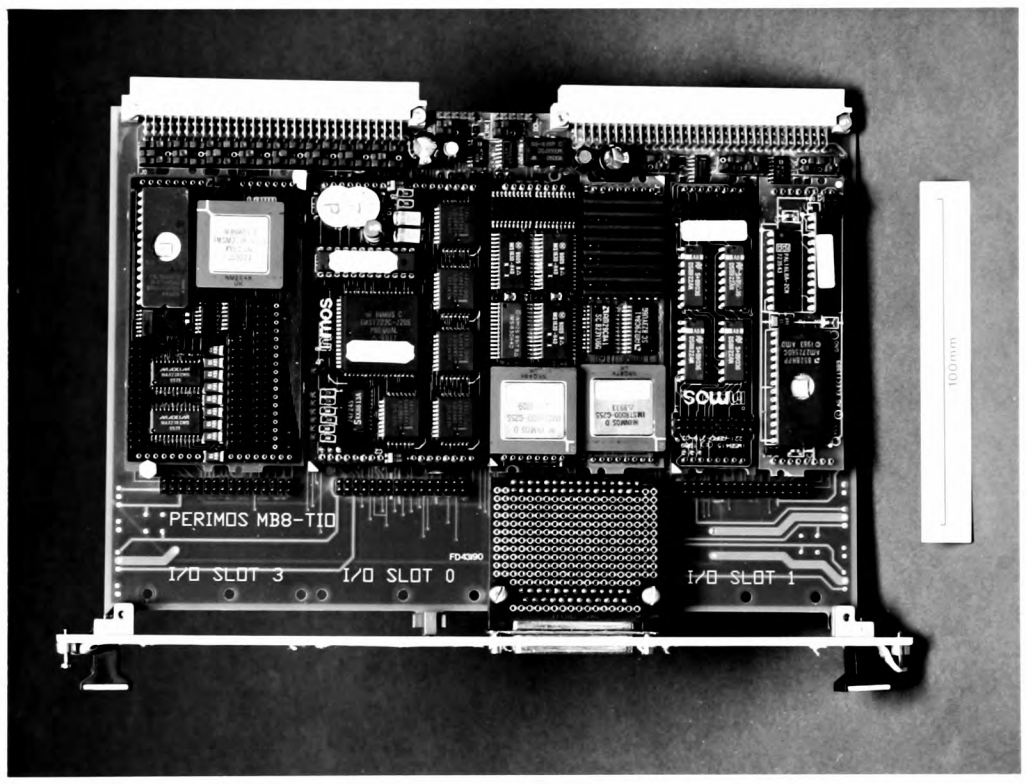



Figure 2.10: Prototype of a Locally Intelligent Control Agent (LICA)

and an external device. The function of the interface board depends heavily upon which kind of external device is connected. We have built a range of different interface modules, such as A/D, D/A, digital I/O, RS232, counter/timer or any combination of these modules, as shown in Figure 2.12. A driver module is mainly used to increase the driving capability or for isolation in a noisy environment. We have designed different driver modules for our use, such as optical isolation, RS422, motor driving or any combination of them.

### 2.5.2 Hardware Mapping

The LICA design provides an effective way to build a distributed control system for our mobile robot. Figure 2.11 shows that the complex control system of the *Turtle* robot can be composed of five LICAs. Each LICA connects sensing to action to achieve a level of competence for the *Turtle* robot to interact intelligently with the dynamic environment. The new controller we built includes five LICAs, as shown in Figure 2.12 (19-inch Parallax rack).

As a example, Figure 2.13(a) shows a LICA configuration for an obstacle avoidance behaviour. Here, only one transputer TRAM (size 1) has been used to fire sonars, control motors, process sonar data, build a local map, and avoid unexpected obstacles. More transputer TRAMs can be added easily if necessary. The interface board has two C011 link adaptors, connected to counters for sonar timing, digital I/O for sonar firing, and a PWM circuit for servo driving. Figure 2.13(b) shows the occam processes and program (PAR means all processes inside are concurrently operating). **comm.proc()** procedure is in charge of communication with other LICAs. **sonar.proc()** handles sonar firing and data processing. **motor.proc()** drives the servo motors to direct the sonar towards interesting objects. **map.proc()** builds a local map to add unexpected obstacles. **avoid.proc()** works out a collision-free path when obstacles are encountered. All of them communicate with one another via software channels, $c1, ..., c4$ and $d1, d2, ..., d6$, rather than via global variables, as in conventional computer languages. Figure 2.14 shows the LICA board we built to implement a level of obstacle avoidance competence described above.

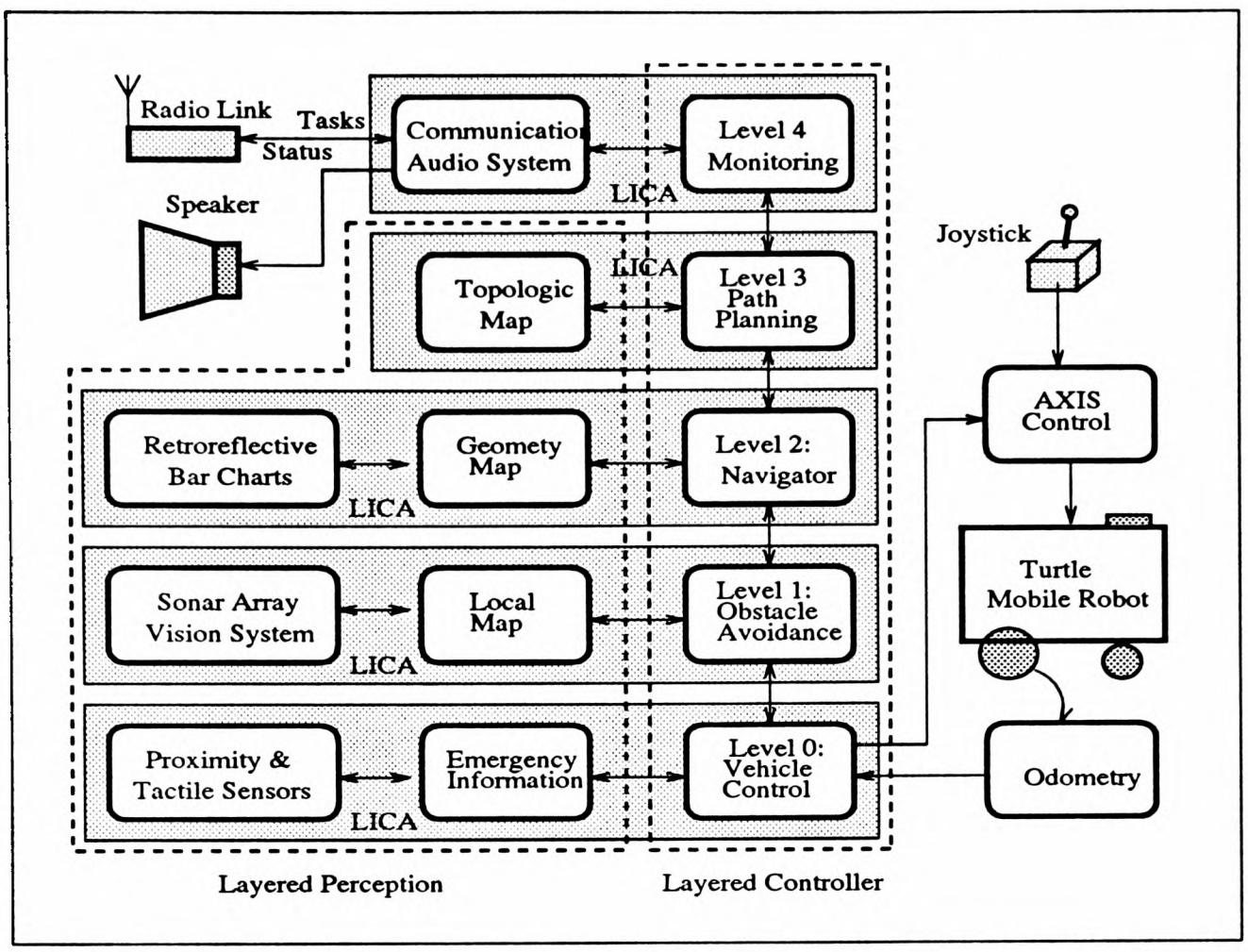



Figure 2.11: A Transputer architecture for the *Turtle* mobile robot

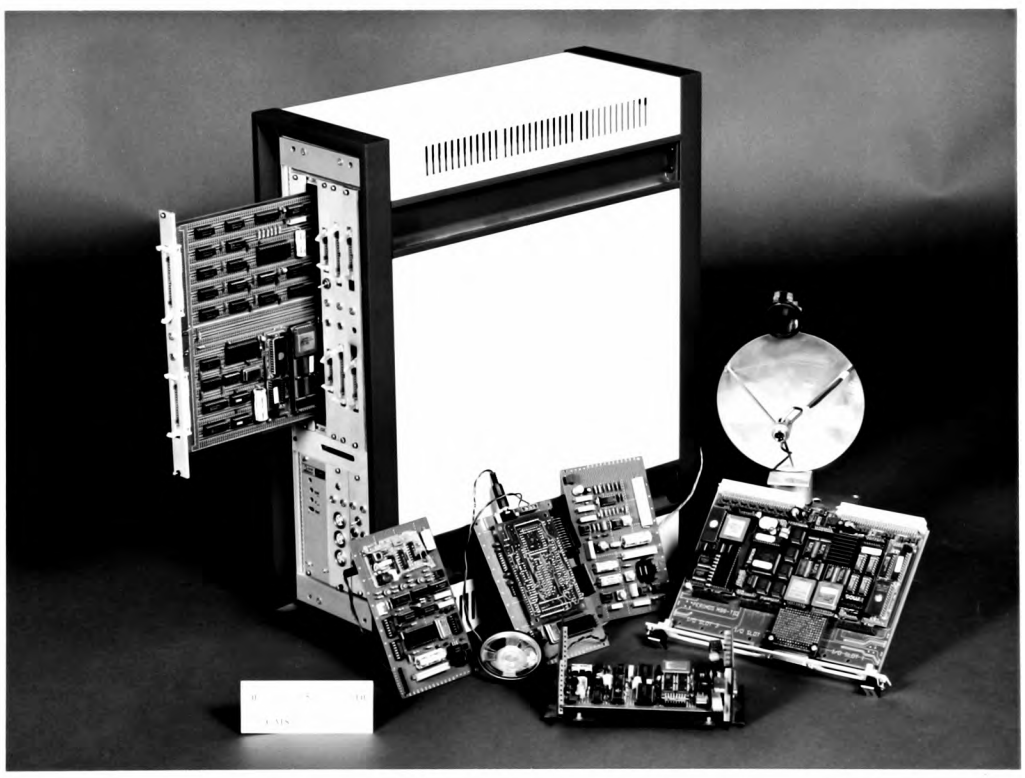
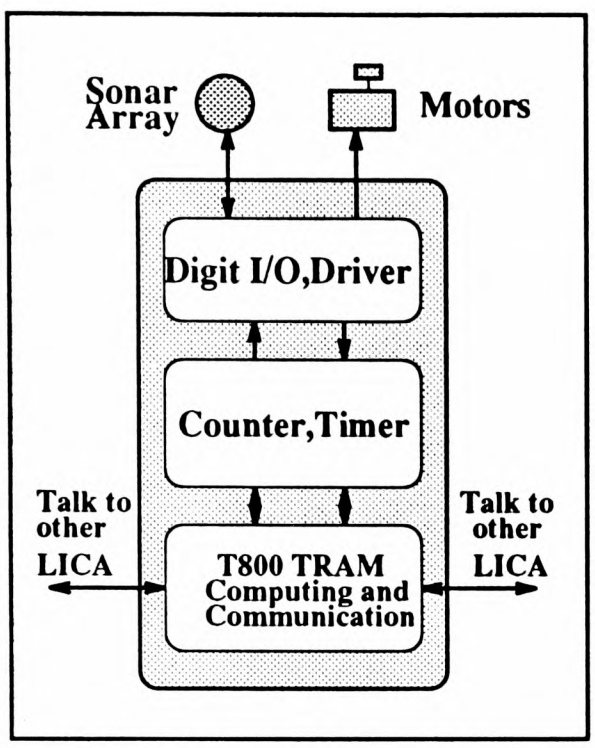


Figure 2.12: New controller consisting of five LICAs




(a)

```
PROC local.planner.proc(CHAN OF ANY in1,
        out1, in2, out2, s.in, s.out, m.in, m.out)
    ... initialization
    ... sonar.sampling
    ... motor.driver
    ... map.building
    ... communication
    ... avoid.control
    PAR
        sonar.proc (s.in, s.out, d1, c1)
        motor.proc (m.in, m.out, d2, c2)
        map.proc (c1, c2, c3, c4, d3, d4)
        avoid.proc (c3, c4, d3, d4, d5, d6)
        comm.proc (in1, out1, in2, out2, d5, d6)
  :
```

(b)

Figure 2.13: Block diagram and software of a LICA for a level of competence

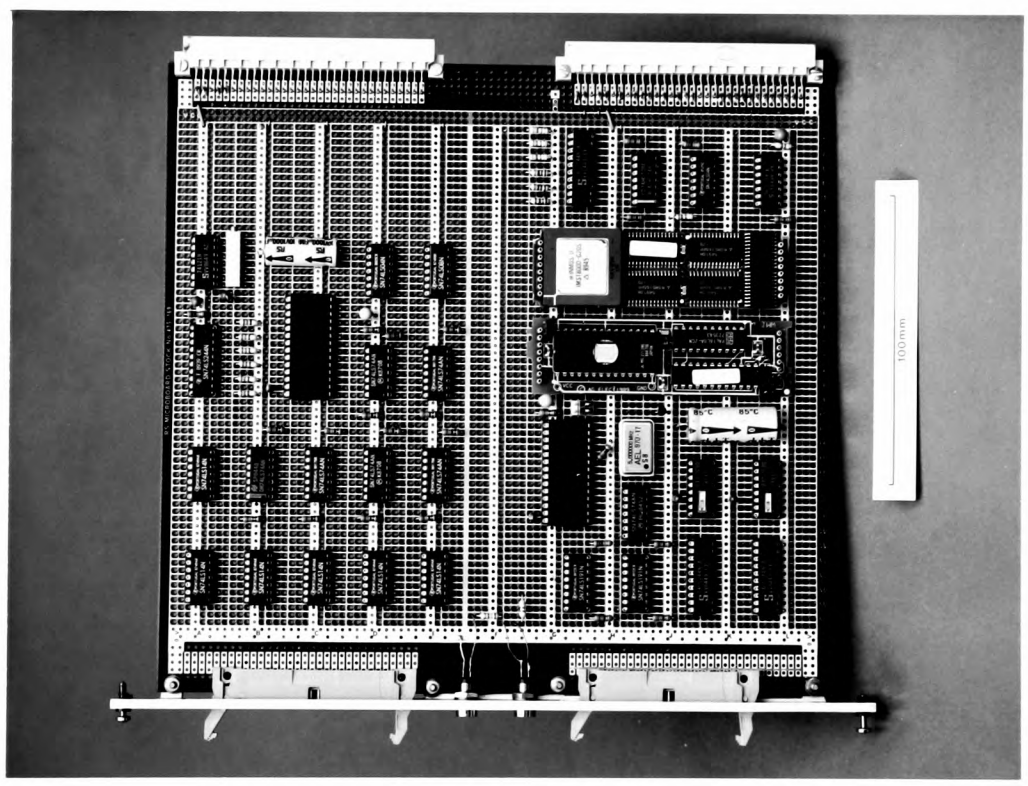



Figure 2.14: A LICA board for the obstacle avoidance competence

Other task-achieving behaviours such as path planning and navigator are built in a similar way. Also, more behaviours can easily be added using LICAs. In this way, the LICA design enables us to achieve flexibility, expandability, and ease of implementation.

### 2.5.3 Software Mapping

Occam is a block structured concurrent language, programs consisting of a network of communicating processes. It is based on the idea of concurrency and communication. It gives us the ability to map our application onto a yet-to-be determined number of transputers, maintaining all the potential for exploiting the parallelism that exists in the application. Figure 2.15 presents the mapping of sensing and control tasks onto a network of communicating processes in the implementation of the DRTA architecture, and Figure 2.16 shows the configuration of a network of communicating processes onto 5 T800 transputers.

To date, only five LICAs are used to build the layered control architecture. When more transputers are needed, the processes currently in a transputer could be mapped easily onto separated transputers in the same LICA. The only change we have to make is that the software channels need to be mapped to the corresponding transputer links.

### 2.5.4 Achieving accurate timing

To be in the right place at the right time, the robot system must be able to determine, and compensate for, system latencies [Andersson, 1986]. The information related by a sensor that an object is three meters from the robot is soon invalidated if the robot or the object is moving. Therefore, any sensor reading in a dynamic environment must be time stamped to define the only time at which the information is accurate. Implementing this strategy requires that the robot controller be able to measure precisely an event's time of occurrence. A transputer has a timer (a clock which 'ticks' every microsecond) that is readable by software (a *read timer* instruction). Therefore, time stamping the sensor data is easy to achieve. Synchronisation between transputers is enforced by software

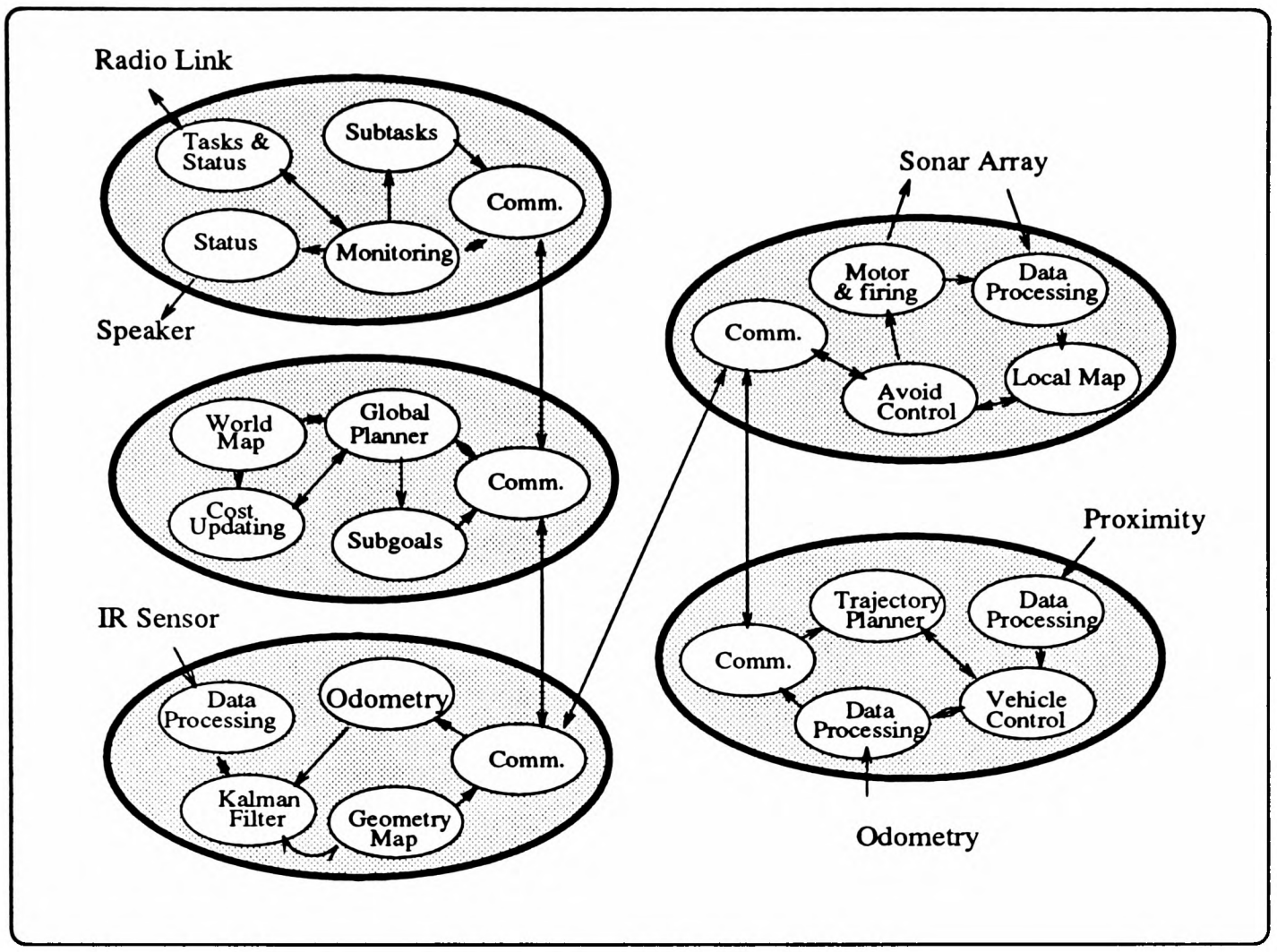



Figure 2.15: Software structure for DRTA

```
... SC monitoring.proc
... SC global.planner.proc
... SC navigator.proc
... SC local.planner.proc
... SC motion.control.proc
... initialization
PLACED PAR
  PROCESSOR 1 T8
    ... channel definition
    monitoring.proc(app1.in, app1.out, radio.in, radio.out, speak.in, speak.out)
  PROCESSOR 2 T8
    ... channel definition
    global.planner.proc(app1.in, app1.out, app2.in, app2.out)
  PROCESSOR 3 T8
    ... channel definition
    navigator.proc(app2.in,app2.out,app3.in,app3.out,ir.in,ir.out)
  PROCESSOR 4 T8
    ... channel definition
    local.planner.proc(app3.in,app3.out,app4.in,app4.out,sonar.in,sonar.out,motor.out)
  PROCESSOR 5 T8
    ... channel definition
    motion.control.proc(app4.in,app4.out,odom.in,track.out,proxi.in,steer.out)
```

Figure 2.16: Software configuration for DRTA

initially when the occam code is down loaded.

### 2.5.5 Asynchronous communication

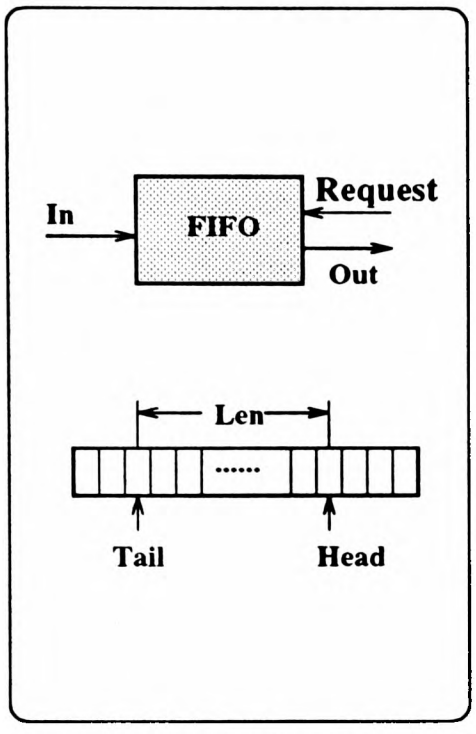
Each LICA incorporates a task-achieving behaviour. It monitors its input data from the sensors and other LICAs, processes that data, and outputs its final decision. Communication between LICAs are loosely coupled, using asynchronous communication. To implement this scheme, we introduce two buffer processes: a FIFO and a memoryless buffer.

A FIFO buffer process is used to collect and transfer data between two LICAs which perform different tasks asynchronously. Figure 2.17 shows its block diagram and corresponding occam program. The size of the FIFO buffer can be chosen according to the applications. The process monitors the input and the request channels to response accordingly. Two pointers, *Head* and *Tail* are used to direct data flow. The guard *Len* is used to avoid the problems of buffer overflow or reading a empty buffer.

A memoryless buffer process is used to transfer the data that is currently updated. For instance, the global planner dynamically generates alternative paths for the local planner to read them when the mobile robot moves from one place to another. If no obstacles are encountered, these alternative paths are ignored by the local planner. Therefore, the global planner needs a memoryless buffer to keep the alternative path data for a while. Figure 2.18 presents the functional diagram and corresponding occam program of the memoryless buffer process we used. This process will take values from the controller and make the most recent data available to be read by other LICAs. It is guaranteed that the other LICAs will always get an up-to-date value. Also, this provides an efficient way to avoid deadlocks.

## 2.6 Debugging and Data Logging

In order to develop a woking transputer-based system, we must have a way to debug it. Figure 2.19 shows a development environment for the research described in this thesis. Niche transputer

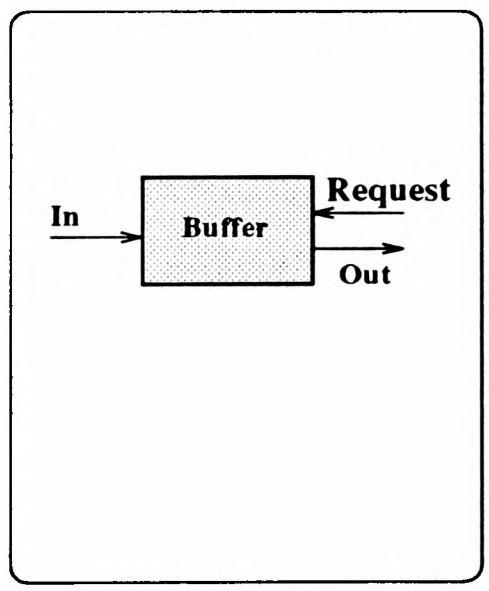



(a) Block Diagram

```
PROC fifo.proc(CHAN OF INT In, Out, Request)
  VAL INT buf.size IS 32:
  [buf.size]INT Buffer:
  INT Req, Len, Head, Tail:
  SEQ
    Len := 0
    Head := 0
    Tail := 0
    WHILE TRUE
      ALT
        Len < buf.size & In ? Buffer[Head]
          SEQ
            Len := Len + 1
            Head := (Head + 1) REM buf.size
        Len > 0 & Request ? Req
          SEQ
            Out ! Buffer[Tail]
            Len := Len -1
            Tail := (Tail + 1) REM buf.size
:
```

(b) Occam Process

Figure 2.17: A FIFO buffer for asynchronous communication




(a) Block Diagram

```
PROC buffer(CHAN OF INT In, Out, Request)
  VAL INT buf.size IS 32:
  [buf.size]INT Buffer:
  INT Len1, Len2:
  WHILE TRUE
    ALT
      In ? Len1
        SEQ
          In ? [Buffer FROM 0 FOR Len1]
          Ready := TRUE
      Ready & Request ? Len2
        SEQ
          Out ! [Buffer FROM 0 FOR Len2]
          Ready := FALSE
:
```

(b) Occam Process

Figure 2.18: A memoryless buffer for asynchronous communication

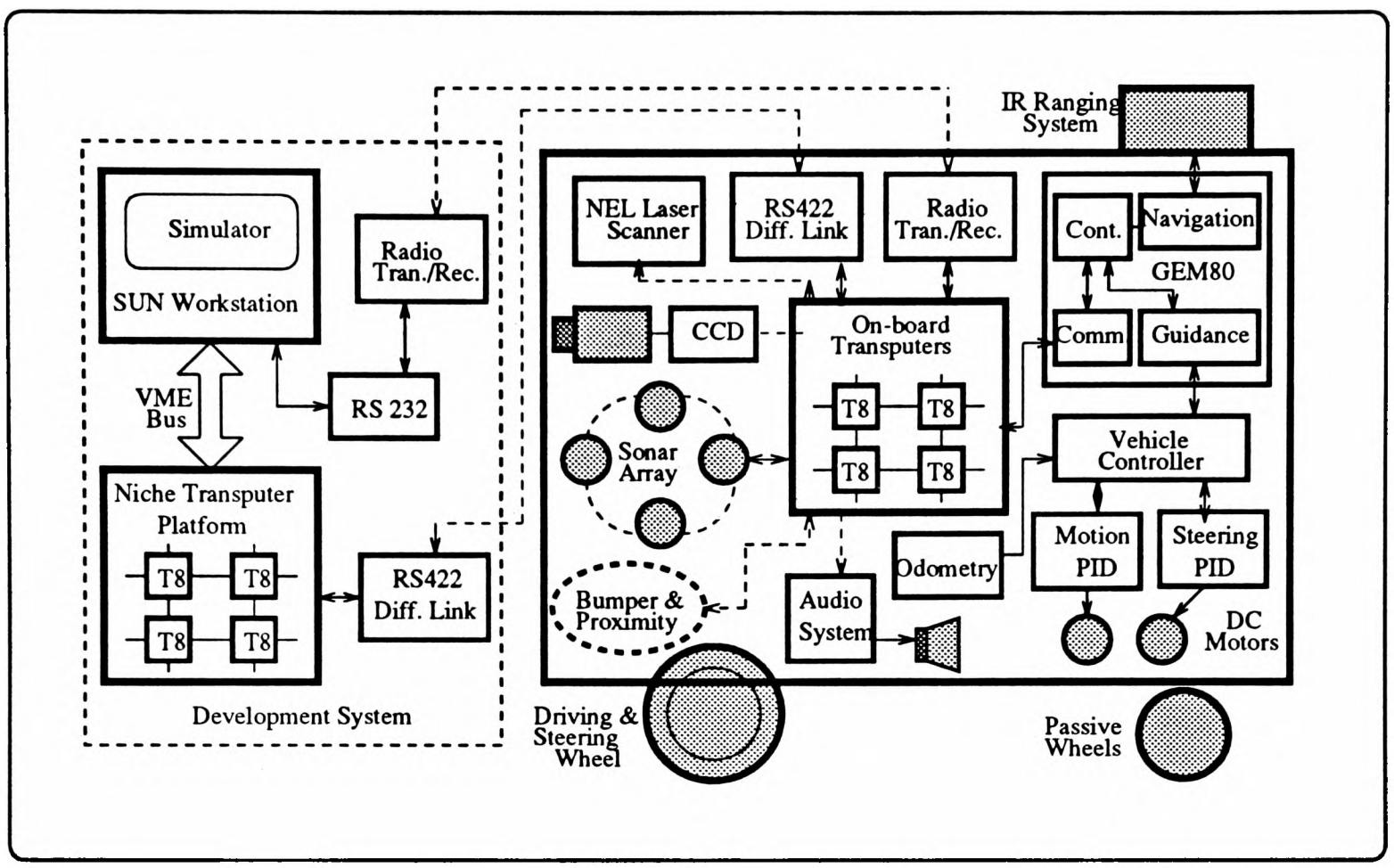



Figure 2.19: Real-time debugger

platform located in a SUN workstation has 4 sites each of which can accommodate 8 transputers. We can choose any one of the 4 sites as root to communicate with on-board transputers via a pair of RS422 differential links through which the verified planning and control algorithms can be downloaded from the SUN workstation to the on-board control system for further testing. Note that RS422 differential links are not connected when the robot is moving. A simulator, as shown in Figure 1.2, running on the SUN workstation is able to control the motion of the Turtle robot and log any data that is useful for analysis via radio links. In this way, the Turtle can travel around in our AGV Lab freely without any umbilical.

## 2.7 Conclusions

The system design of mobile robots poses a key challenge to the robotics community. The system design in this chapter is heavily influenced by the need for real-time sensing, control and decision-making in order for the *Turtle* mobile robot to interact intelligently with a dynamic environment. In general,

- The architecture we build for the *Turtle* is a distributed community of sensing, planning, and acting modules. It is considered as a distributed problem solving network, integrating high-level decision making with low-level control and sensor interpretation to provide for navigation, real-time obstacle avoidance, and exploration in a dynamically changing environment.
- The system is based on locally intelligent control agents (LICAs), each with its own processing unit and local memory, which together do not require any common buses or any central communication facility. It has point to point synchronised communication links which provide no contention for the communication mechanism, regardless of the number of transputers in the system. There is no global shared memory. The communication bandwidth does not saturate as the size of the system increases.
- The system has many desirable properties including modularity of sensing and control devices, flexibility to the addition of more sensing and control nodes, and high bandwidth.

# Chapter 3

# Dynamic Path Planning With Uncertainty

Given the control system designed in the last chapter, we are now in a position to address a fundamental issue related to path planning with uncertainty for the Turtle mobile robot. We assume that prior information about the environment is available, but that unexpected obstacles may appear randomly. To handle the uncertainty, we adopt a Bayesian approach to build statistical models for encountering unexpected obstacles, which are based on assumed prior distributions and updated dynamically by sensor data. These statistical models provide quantified uncertainty costs which are used to assign merit scores to each portion of a path. When the robot searches for an optimal path, it takes the uncertainty cost into account in making a decision. Based on prior information and sensor data, we show that the proposed method allows the robot vehicle to cope with unexpected obstacles, plan an optimal path and learn global knowledge from its travels in real time.

## 3.1 Introduction

To improve the performance of mobile robots in a factory environment or a warehouse naturally demands an intelligent planning system. This planning system should provide a mobile robot with the capabilities to handle multiple goals, find an optimal path, and respond appropriately to unexpected events. It should be dynamic in the sense that the planned path should be easily modifiable according to available sensor data in order for the mobile robot to interact with its surroundings more intelligently. Constructing such a planning system is obviously difficult and relies on knowing how to deal with uncertainty.

The planning task of a mobile robot can be decomposed into four distinct subtasks: mission planning, global path planning, local path planning and trajectory planning. In general, *mission planning* is the process of determining the requirements and constraints for the global tasks and obtaining an optimal schedule for multiple goals. *Global path planning* is to find a collision-free path (a set of intermediate points) for a mobile robot to follow from the start position to the goal position. In contrast, *local path planning* generates relatively local detours around sensed obstacles while following a globally planned path. Trajectory planning is to generate a nominal motion plan consisting of a geometrical path and a velocity profile along it in terms of the individual kinematics of the robot. This chapter addresses mainly the problem of global path planning. Local path planning and trajectory planning will be addressed in the following chapters. Mission planning is not studied in this thesis.

Changes in the robot's environment may be due to the motion of the robot, the appearance and disappearance of objects, and to object motion. If the changes are predictable, they can be taken into account when the robot plans its optimal path. But, if the world changes unpredictably, the robot has to plan and replan a collision-free path dynamically. This is the problem of planning under uncertainty. In such a case, a robot has to rely on its sensors to detect unexpected events and then adapt its path accordingly. Further, to plan an optimal path, uncertain events, such as unexpected obstacles, should be quantified as costs in order to make judgement feasible. Fortunately, statistics and Bayesian decision theory provide the general tools for representing and reasoning with uncertainty.

We begin with a literature review of previous approaches on global path planning in Section 2. The planning strategy we adopt is presented in Section 3. Section 4 presents the factory-like robot environment, and its topological map, which the Oxford AGV inhabits. To search for an optimal path, global path planning based on cost functions is presented in Section 5. A statistical model for counting unexpected obstacles is proposed in Section 6. Section 7 describes how the subgoals

are dynamically generated during traversals. Implementation results are presented in Section 8. Finally, the conclusions are summarised.

## 3.2 Previous Work

Global path planning has attracted much attention in recent years by the robotics community. Many different approaches have been proposed, based on different representations of the robot's environment. To simplify the problem, some path planning methods use a static world model to deal with a well-known environment and known obstacles. Other approaches are adopted to handle a dynamic environment and find a path through a dynamic representation of the robot's environment that is built incrementally.

Global path planning for a mobile robot in a known environment with known static objects has been studied extensively. Graph searching and potential field methods are the two main approaches used to solve the path-finding problem. The main feature of both methods is that the entire environment is modeled geometrically before the motion takes place. In the graph searching approach, a graph is created that shows the free spaces and forbidden spaces in the robot's environment. A path is then generated by piecing together the free spaces or by tracing around the forbidden area. Such approaches include *Configuration Space* [Lozano-Pérez, 1983], *Generalised Cones* [Brooks, 1983], the *Vertex Graph Method* [1977], the *Voronoi diagram* [Takahashi and Schiling, 1989], *Intersecting Convex Shapes* [Singh and Wagh, 1987], and the *Quadtree Representation* [Kambhampati and Davis, 1986]. In contrast, the potential field approach uses a scalar function to describe both objects and free space. The main advantage of a potential field approach to path planning is that it offers a relatively fast and effective way to solve for safe paths around obstacles. More specifically, the negative gradient of the potential field is precisely the direction in which to move to avoid obstacles. Previous researchers who have applied potential fields to global path planning include Newman and Hogan [1987], Warren [1989], Hague, Brady, and Cameron [1990], and Hwang and Ahuja [1992].

Note that the potential field method is also used in local path planning which will be described in the next chapter.

However, the robot's environment is not always static. Recently, a number of path planning algorithms have been reported that allow the robot to cope with moving objects whose shape, speed, and direction are assumed known a priori [Fujimura and Samet, 1989], [Kant and Zucker, 1986], [Pan and Luo, 1990]. Note that most of these approaches are designed primarily to solve the path-finding problem. When an optimal path is concerned, search algorithms such as a $A^*$ search, hill climbing, and dynamic programming can be used to find a globally optimal solution, in both the graph searching and potential field approaches.

More generally, globally planning a path for a mobile robot in a dynamic or unknown environment poses a difficult problem. The main difficulty is how to model the dynamic, uncertain environment in a manner that makes it possible to provide a solution for an optimal path constrained by the real world. Most approaches use active on-board sensors for the acquisition of the information about the robot's surroundings. The environment models are then built up dynamically and the path determined as the robot moves around. Various grid representations of the environment have been proposed, such as the *Occupancy Grid* by Moravec and Elfes [1985], [1987], the *State Graph* by Faverjon and Tournassoud [1987], and the *Identical cubes* by Jun and Shin [1988]. In these approaches, probabilistic models are built based on grid cells and updated dynamically using sensor data. A path is then found by minimising the probability of encountering obstacles. However, such methods involve enormous computation to set up different grids to map the environment, forcing the robot to deal with huge amounts of data. Moreover, if the kinematics and dynamics of a non-holonomic mobile robot are taken into account, the method is difficult to implement. Since there is not complete information about the robot's environment during planning, the methods achieve a suboptimal solution in most cases.

In this chapter, we propose a probabilistic approach to address the problem of path planning

with uncertainty for mobile robots. Our approach is different from the approaches mentioned above. Instead of using an uncertainty grid, we use a topological graph to represent the free space of the robot's environment, weighted by scalar cost functions. The statistical models are built to quantify uncertainty, forming uncertainty costs for unexpected events. These uncertain costs are updated by available sensor data when the robot moves around. An optimal path is then found from the topological graph using cost functions and dynamic programming.

## 3.3 Overview of the Approach

We assume that the mobile robot is provided with a two-dimensional geometrical model which is a projection of the factory layout and populated by machine tools, workstations, stores, and free space. However, there are unknown objects such as furniture, pallets, boxes, *etc.* Their position may change during mobile robot missions. In addition, people and other robots may move around. No prior information about them is available. On-board sensors are used to detect the unexpected events within a limited range and accuracy in real time. To plan a collision-free path in this partially-unknown environment is obviously difficult. Given that global optimality of the solution is desirable, we have to answer the following questions:

- What is an optimal path satisfying the given constraints on the mobile robot and its environment?
- How is the uncertainty quantified in a dynamically changing industrial environment?
- If the robot encounters an unexpected obstacle during a portion of a path, which decision should be made next time: try again or give up this path?

To answer these questions leads us to investigate a probabilistic approach to the problem and to look for solutions that are optimal in an average sense. Based on the *Separation theorem* [Bertsekas, 1976], the original planning problem is separated into two basic problems in our design, i.e. stochastic optimal estimation and deterministic optimal planning, for simplification. In the

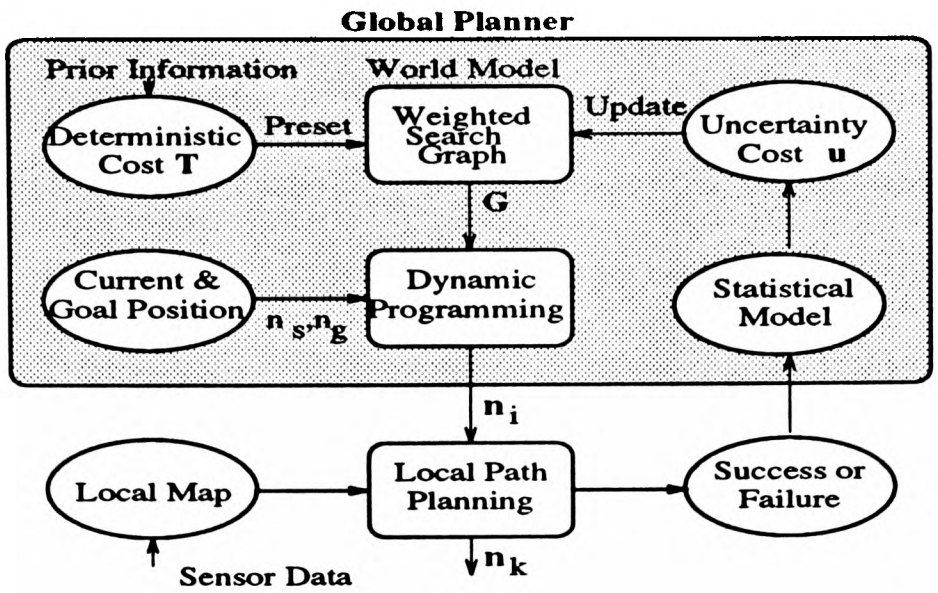



Figure 3.1: Flow chart of the proposed path planning strategy

estimation, the basic idea is: the unknown changes of the environment (classified into different types of obstacles) are represented by random variables with unknown statistical properties. These unknown properties are parameters which define the probability distribution functions of the random variables, such as mean or variance. These unknown parameters are learned by Bayesian estimation recursively from the outcomes of random variables, i.e. the observations of the environment provided by on-board sensors, during the traversals of the mobile robot. Then the global path planner modifies its internal world model dynamically on the basis of the estimated properties and searches for an optimal path using a deterministic dynamic programming algorithm.

More precisely, the method we propose can be divided into three steps: (i) we construct a topological search graph G (defined in the next section) to model the robot's environment. In this graph, nodes $n_i$ represents the centers of topological spaces and arcs are the connection paths between any two nodes. Both of them are formulated to represent the static part of the two-dimensional environment in terms of free space. Each arc has an associated cost function, including a deterministic cost $T$ proportional to each path length and width. (ii) we develop a statistical model to quantify the uncertainty cost $u$ which is also used to weight each arc. (iii) deterministic dynamic programming is used to find a path that minimises the total cost between the current position and

the goal. Figure 3.1 shows a flow chart of our approach.

At the global level, no geometric information is stored and processed. Instead, a search graph G with scaled variable cost functions $\sum_{i=1}^{N} \mathbf{C}_i$ are used to find a minimum cost path, a set of intermediate points (nodes) $\sum_{i=s}^{g} \mathbf{n}_i \in \mathbf{G}$ from the start position to the goal. This thus eliminates the (expensive) transformation of obstacles into a *Configuration Space* representation that is often used, and makes real-time path planning feasible. Whenever an obstacle is detected, the local planner generates a collision-free detour path, $\mathbf{n}_k = (x_k, y_k)$ (k=0, 1, 2, ...), between subgoals $\mathbf{n}_i$, based on a local map and available sensor data, subject to the constraints imposed by the robot's kinematics and the predefined road boundaries. It has four main tasks:

- Maintain a local map using information abstracted from available sensor data.
- Check the clearance along the robot's path and avoid unexpected obstacles.
- Make decisions whether to sidestep or backtrack in the face of unexpected obstacles.
- Provide motion results to the global planner to update the uncertainty cost.

In general, the planning algorithm we propose is to operate in a closed-loop manner. It monitors the implementation of a planned path (success or failure), updates the internal world model using the statistical models defined below, and makes a prediction for an optimal path. More details will be presented in the following sections.

## 3.4 Environment Model and Weighted Graph

### 3.4.1 Motivation

Global path planning needs a map to tell the robot where and how to move from one position to another. Traditional spatial representation methods for a known environment, and corresponding approaches to the map-building problem in an unknown environment, are based on the accumulation of accurate geometrical descriptions of the environment. We have mentioned some of these methods in Section 3.2. Some of them perform reasonably well where environments are small enough to

observe most of the important features from the robot's position [Leonard and Durrant-Whyte, 1992]. However, it is often difficult to build this accurate map in a large-scale space because of low mechanical accuracy and sensory errors. Instead, some researchers use various types of topological models or graph models to represent the connectivity of the robot's environment. For instance, Chatila and Laumond [1985] build a topological model from the geometric description and then derive a semantic model, e.g. identifying rooms and corridors, from the topological map. Turchan and Wong [1985] use an attributed graph to model the world, in which line segments and their attributes become vertices, and related adjacent line segments are represented by arcs. Oommen *et al.* [1987] use a visibility graph, where vertices represent observable or actually visited meaningful points in the environment, and arcs show the connectivity of vertices for travel. Kuipers and Byun [1988] conduct a similar qualitative approach to the environment model, in which distinctive places and paths are defined by their properties at the control level and serve as the nodes and arcs of the topological model.

The industrial environment may include different buildings, stores, floors, rooms and work-stations. To model and update this large-scale space using a purely metrically accurate map is complicated, and is often brittle in the face of low mechanical accuracy and sensory errors. Motivated by qualitative approaches, we propose a topological model to represent the robot's environment for real time implementation. It has a hierarchy structure in terms of buildings, floors, and rooms. It is abstracted from the geometrical features of the environment, and used for global path planning. Differing from the topological model mentioned above, it includes variable cost functions. The following sections describe how to build this model. Note that only the model of rooms is constructed in this thesis. However, the method can be directly used to build similar models for floors and buildings in a large-scale factory in a hierarchical structure.

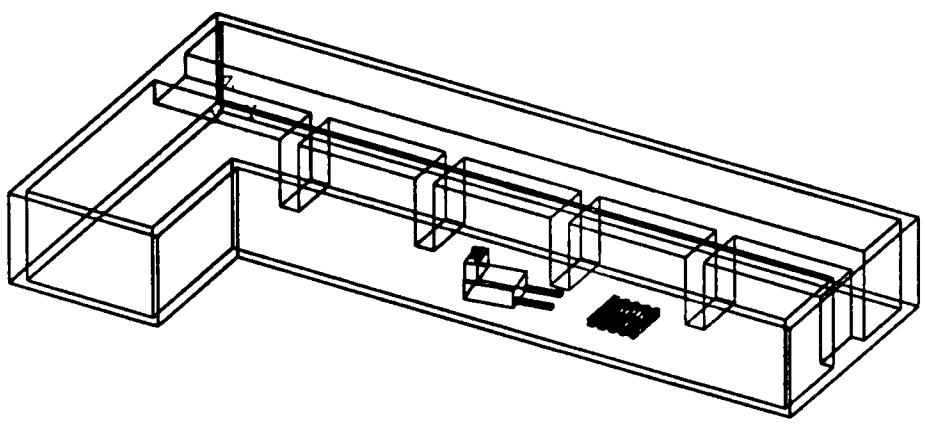


Figure 3.2: The Oxford AGV laboratory

### 3.4.2 Model of AGV Laboratory

We have a factory-like test site, shown in Figure 3.2. It is 15m long and 3.8m wide, including four pillars. We assume that the geometrical features of this environment are known *a priori*, but people, tables, chairs, boxes, *etc.* form unknown obstacles which may be in any position.

The model of the environment we build consists of two parts: the *known* part and the *unknown* part. For the known part, we decompose free space into connecting quadrangles. Then, we construct a topological model that contains information about how the different parts of the world are interconnected. This information is typically invariant for the environment and allows the robot to plan a feasible path and verify its location as it travels. The model is a simple non-directed graph consisting of a network of nodes classified as corridors, corners and junctions. It is formulated by taking into account the topological and geometrical features of the environment. Figure 3.3 shows a topological model of our AGV Lab generated from its geometrical description. Note that since places, junctions, corners, and corridors are unlikely to change, the topological model is a deterministic model.

The unknown part of the environment includes any dynamic changes in the world. In the absence of *a priori* knowledge about such changes the uncertainty model is built statistically and in turn updated dynamically as the robot moves around, using its on-board sensors. More details are presented in Section 3.6.

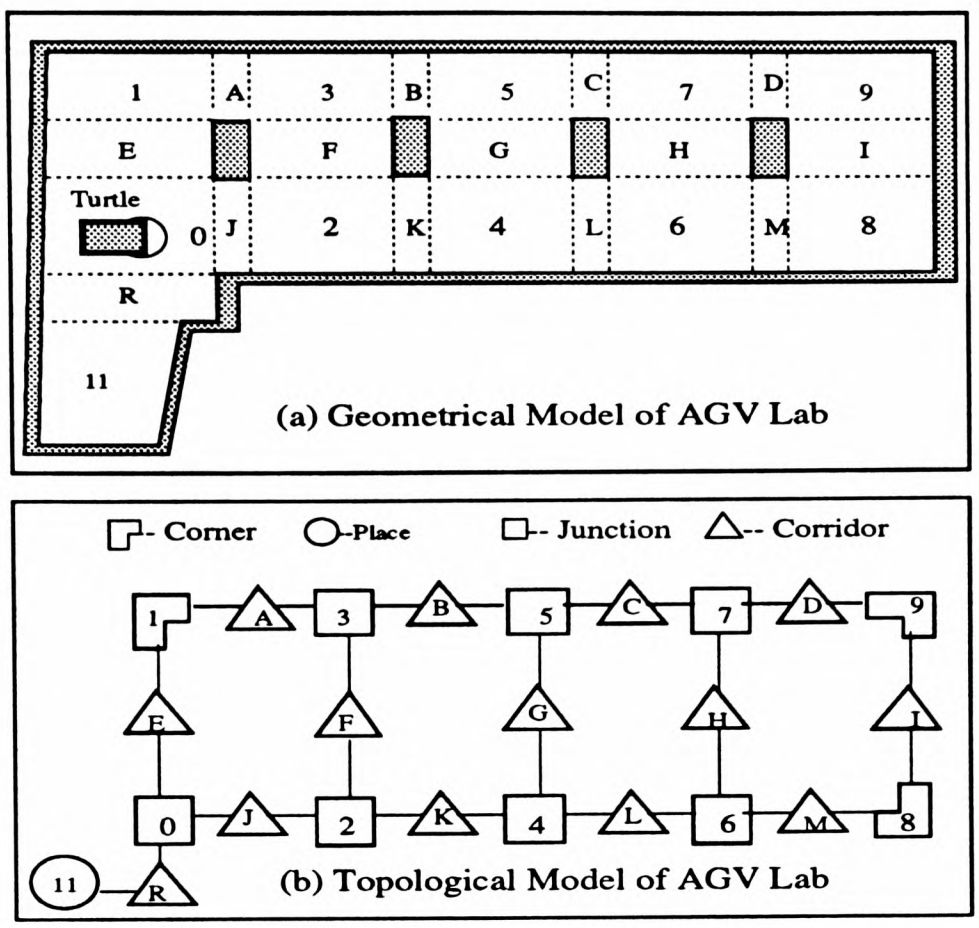



Figure 3.3: Geometrical Model and Topological Model

### 3.4.3 A Weighted Graph

To represent this topological model, we give the following definition.

**Definition 1** *Let* $\mathbf{N} = \{n_1, n_2, ..., n_p\}$ *be a set of nodes in a graph. Let* $\mathbf{A}$ *denote a set of arcs* $\sum a_{ij}$, $i, j = n_1, n_2, ..., n_p$, *joining the nodes in the graph. For each arc* $a_{ij} \in \mathbf{A}$, *let* $C_{ij}$ *be the cost function of moving from nodes* $n_i$ *to* $n_j$. *Assume that* $0 \leq C_{ij} < \infty$ *and* $C_{ii} = 0$ *for all* $i, j$. *Then a weighted graph* $\mathbf{G}$ *is defined as:*

$$\mathbf{G} = (\mathbf{N}, \mathbf{A}, \mathbf{C}) \tag{3.1}$$

Note that the cost function associated with each arc integrates information from a deterministic model $T_{ij}$ and an uncertainty model $u_{ij}$ (defined later). $T_{ij}$ is constant for static part of the environment and $u_{ij}$ varies according to dynamic changes in the environment. In fact, $\mathbf{G}$ is a variable cost graph to be used to plan an optimal path dynamically.

Table 3.1: Adjacency Matrix Representation

| | $\mathbf{n_0}$ | $\mathbf{n_1}$ | $\mathbf{n_2}$ | ... | $\mathbf{n_9}$ | $\mathbf{n_{11}}$ | $\mathbf{n_a}$ | $\mathbf{n_b}$ | $\mathbf{n_c}$ | ... | $\mathbf{n_m}$ | $\mathbf{n_r}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{n_0}$ | 0 | $C_{01}$ | $C_{02}$ | ... | $C_{09}$ | $C_{011}$ | $C_{0a}$ | $C_{0b}$ | $C_{0c}$ | ... | $C_{0m}$ | $C_{0r}$ |
| $\mathbf{n_1}$ | $C_{10}$ | 0 | $C_{12}$ | ... | $C_{19}$ | $C_{111}$ | $C_{1a}$ | $C_{1b}$ | $C_{1c}$ | ... | $C_{1m}$ | $C_{1r}$ |
| $\mathbf{n_2}$ | $C_{20}$ | $C_{21}$ | 0 | ... | $C_{29}$ | $C_{211}$ | $C_{2a}$ | $C_{2b}$ | $C_{2c}$ | ... | $C_{2m}$ | $C_{2r}$ |
| : | : | : | : | ... | : | : | : | : | : | ... | : | : |
| $\mathbf{n_9}$ | $C_{90}$ | $C_{91}$ | $C_{92}$ | ... | 0 | $C_{911}$ | $C_{9a}$ | $C_{9b}$ | $C_{9c}$ | ... | $C_{9m}$ | $C_{9r}$ |
| $\mathbf{n_{11}}$ | $C_{110}$ | $C_{111}$ | $C_{112}$ | ... | $C_{119}$ | 0 | $C_{11a}$ | $C_{11b}$ | $C_{11c}$ | ... | $C_{11m}$ | $C_{11r}$ |
| $\mathbf{n_a}$ | $C_{a0}$ | $C_{a1}$ | $C_{a2}$ | ... | $C_{a9}$ | $C_{a11}$ | 0 | $C_{ab}$ | $C_{ac}$ | ... | $C_{am}$ | $C_{ar}$ |
| $\mathbf{n_b}$ | $C_{b0}$ | $C_{b1}$ | $C_{b2}$ | ... | $C_{b9}$ | $C_{b11}$ | $C_{ba}$ | 0 | $C_{bc}$ | ... | $C_{bm}$ | $C_{br}$ |
| $\mathbf{n_c}$ | $C_{c0}$ | $C_{c1}$ | $C_{c2}$ | ... | $C_{c9}$ | $C_{c11}$ | $C_{ca}$ | $C_{cb}$ | 0 | ... | $C_{cm}$ | $C_{cr}$ |
| : | : | : | : | ... | : | : | : | : | : | ... | : | : |
| $\mathbf{n_m}$ | $C_{m0}$ | $C_{m1}$ | $C_{m2}$ | ... | $C_{m9}$ | $C_{m11}$ | $C_{ma}$ | $C_{mb}$ | $C_{mc}$ | ... | 0 | $C_{mr}$ |
| $\mathbf{n_r}$ | $C_{r0}$ | $C_{r1}$ | $C_{r2}$ | ... | $C_{r9}$ | $C_{r11}$ | $C_{ra}$ | $C_{rb}$ | $C_{rc}$ | ... | $C_{rm}$ | 0 |

In order to process the weighted graph during global path planning, we first need to decide how to represent it within the computer. Considering the global path planning to be performed, we choose an adjacency matrix [Sedgewick, 1988] to represent the weights of the search graph. It is a symmetric matrix, shown in Table 3.1.

The elements, $C_{ij}$, of the matrix are the cost functions between any two nodes: 0 for itself, otherwise $\geq 0$. Note that $C_{ij} = C_{ji}$ since $G$ is non-directed graph. A question is how to plan an optimal path on a global level using this graph. The next section addresses this problem based on cost functions and dynamic programming.

## 3.5 Global Path Planner

The topological world model we construct is designed primarily for fast execution, efficient storage and abstraction of geometrical information about the real world. It is mainly used for the global path planner to determine a path or a set of intermediate points for the robot to follow from a start position located in one place to a destination located in another place without collision. To determine an optimal path, we use travel time estimate as a metric rather than distance based on the

following reasons:

- A minimum distance path may cause the robot to decelerate sharply at corners or stop to pivot, and may also not be safe as it passes too near to obstacles.
- The robot velocities are likely to be piecewise constant and depends on the different situations of path width and straightness, as well as maneuvering with unexpected obstacles, which makes distance metric difficult to be used.
- Reducing the time required to transfer a part from one location to another is economically attractive since even a small decrease in each traversal, multiplied by the number of cycles in the lifetime of the transportation, could imply substantial savings in cost.

### 3.5.1 Road Representation

Global path planning needs a good representation of free space to decide where to move. For a factory or warehouse, it is possible to use a direct representation, showing free motion space as distinct roads or motion channels bounded by walls, machine tools, workstations, stores, and delivery bays [Brooks, 1983]. The overall operational area for a mobile robot is enclosed within the road or channel boundary. This greatly reduces the complexity of searching for an optimal path and makes real-time path planning feasible.

Figure 3.4 shows the central lines of the roadways defined for our mobile robot. The road sides consist of walls and pillars that are at a safe distance. As can be seen, these predefined roadways are made up of both straight-line and curve segments, and can be regarded as generalised Voronoi edges. In fact, the roadways can be computed automatically by a Voronoi-like algorithm [Hague and Cameron, 1990].

### 3.5.2 Cost Function

The path optimisation problem can be seen as the selection of the path along which the cost function is minimised. Therefore, the global path planner requires a cost function to assign merit to a

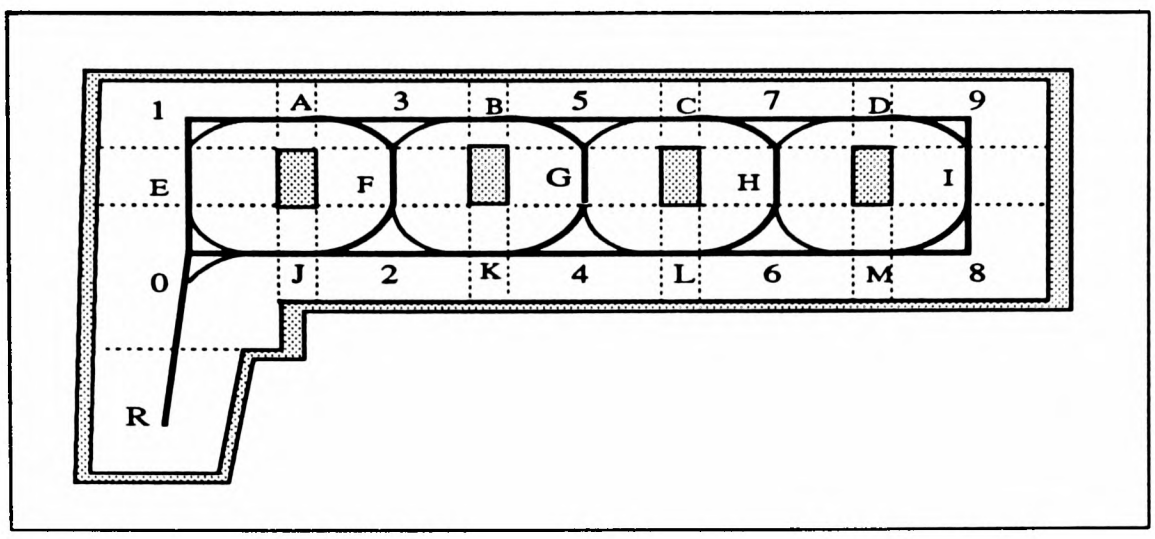



Figure 3.4: A road representation

path to measure its goodness. To take into account the probability of encountering unexpected obstacles during path planning, we define the cost function of any portion path to consist of two parts: the deterministic cost (time spent without obstacles), the uncertainty cost (time spent avoiding obstacles).

**Definition 2** *Let $T_{ij}$ denote the deterministic cost and $u_{ij}$ the uncertainty cost $u_{ij}$. $u_{ij}$ is estimated by the statistical models and available sensor data described in Section 3.6. $u_{ij}$ is 0 for no obstacles, otherwise strictly positive. The total cost $C_{ij}$ associated with a portion of path (arc or straight line) between node $n_i$ to node $n_j$ is then:*

$$C_{ij} = T_{ij} + u_{ij} \tag{3.2}$$

The deterministic cost $T_{ij}$ depends on path parameters such as length, width, and straightness. $T_{ij}$ holds the same value for both directions of a path portion, i.e. $T_{ij} = T_{ji}$. It is calculated off-line before planning, and doesn't change for a specific environment. To motivate the choice of a deterministic cost function, Figure 3.5 shows two typical paths between two nodes: a straight line and a curve path.

**Definition 3** *Let $L_{ij}$ be the path length between nodes $n_i$ and $n_j$, and $S_{ij}$ the piecewise constant speed corresponding to different portions of the path. $K_w$ denotes a weight factor inversely proportional to the path width, and $K_\eta$ is a weight factor inversely proportional to the path*

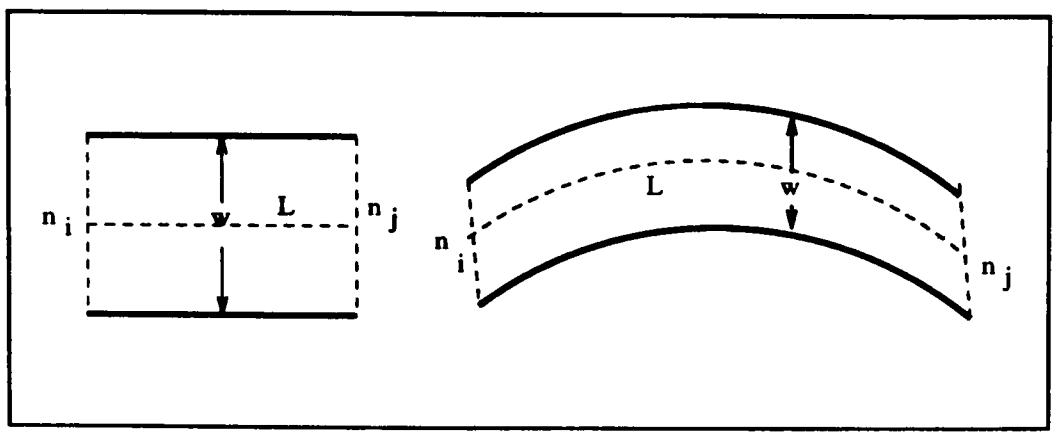



Figure 3.5: Two Typical Paths

*straightness. The deterministic cost $T_{ij}$ spent by the robot when travelling from node $n_i(x_i, y_i)$ to node $n_j(x_j, y_j)$ without obstacles is defined as:*

$$T_{ij} = K_w K_\eta \frac{L_{ij}}{S_{ij}} = K_w K_\eta \frac{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{S_{ij}} \tag{3.3}$$

The definition of uncertainty cost $u_{ij}$ is given in Section 3.6. It is worthwhile to notice that each cost weight $C_{ij}$ can be calculated initially based on prior information and then updated on the basis of sensor data. Its change only depends on uncertainty cost $u_{ij}$.

The cost function defined above is used to weight each arc of the topological model to form a non-directed search graph. Figure 3.6 shows a non-directed search graph with costs on its edges.

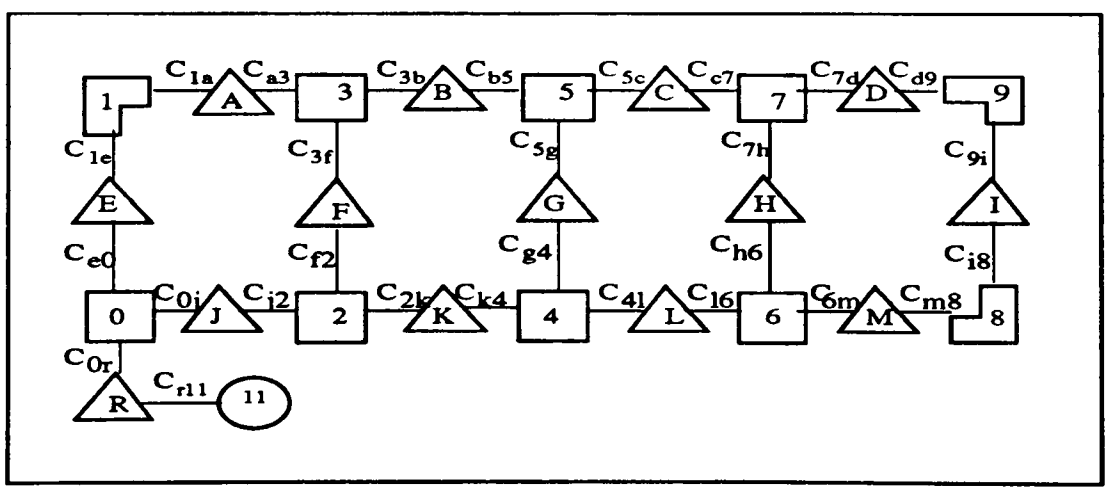



Figure 3.6: A weighted search graph

Based on this weighted search graph, the global planning algorithm generates an optimal path by minimising the total cost from the start to the goal position based on dynamic programming

technique which we describe in the next section.

### 3.5.3 Dynamic Programming Algorithm

The optimisation method known as dynamic programming was developed by Bellman [Bellman, 1957], and arose out of the study of multistage decision processes. It has been successfully applied to a class of problems which require an optimal trajectory or route. A usual objective in such problems is to find the route between two points which will minimise a cost function, such as the travel time or the distance.

Fundamental to the theory of dynamic programming is the principle of optimality which can be stated as follows:

- An optimal policy has the property that, whatever the initial state and the initial decision may be, the remaining decisions must constitute an optimal policy with respect to the state resulting from the first decision.

In terms of state transitions of a mobile robot, we may express the principle of optimality in the following way. Given that a mobile robot is initially at state $\mathbf{x}(0)$, and an optimal path is formed by successive N-stage transitions to specified states $\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(N)$. From any intermediate state $\mathbf{x}(k)$ onwards, remaining path is optimal for the $(N - K)$-stage subproblem initiated at state $\mathbf{x}(k)$.

This principle, illustrated in Figure 3.7, allows us to build up solutions by progressing backward in time.

Let us now consider the optimal planning problem for a mobile robot system defined by a general form:

$$\mathbf{x}(k+1) = \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k)] \tag{3.4}$$

where $\mathbf{x}(k)$ is the current state of the robot, $\mathbf{x}(k+1)$ is the next state of the robot, and $\mathbf{u}(k)$ is the decision based on the current state. The transition function $\mathbf{f}[\cdot]$ relates a state and a decision at a given stage $k$ to the succeeding stage $k+1$.

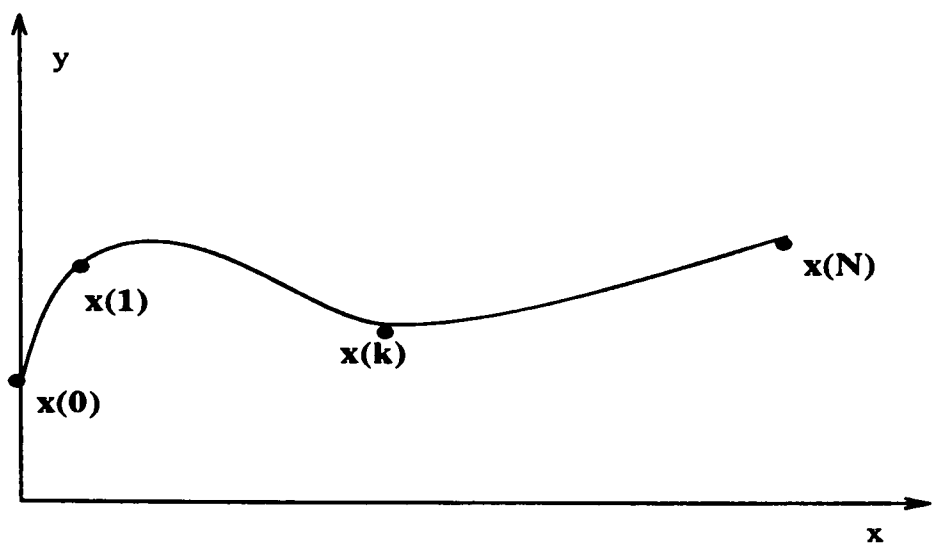



Figure 3.7: Principle of optimality. Note that the optimal solution to the original problem is the entire path from $\mathbf{x}(0)$ to $\mathbf{x}(N)$ and the optimal solution to the subproblem initiated at $\mathbf{x}(k)$ is the final part of the original path.

To apply the dynamic programming idea, the principle of optimality is captured in mathematical terms by introducing the concept of the cost or penalty function which is associated with the state transition of the mobile robot and expressed as a scaled function of the state $\mathbf{x}(k)$ and the decision $\mathbf{u}(k)$:

$$Cost = C[\mathbf{x}(k), \mathbf{u}(k)] \tag{3.5}$$

Then the optimal planning problem can be expressed as follows: given an initial state $\mathbf{x}_s$ and the goal state $\mathbf{x}_g$, make a sequence of $N$ decisions

$$\pi = \{\mathbf{u}(0), \mathbf{u}(1), ..., \mathbf{u}(N-1)\} \tag{3.6}$$

in such a way as to minimise the accumulated cost expressed by the summation

$$J = C[\mathbf{x}(N)] + \sum_{k=0}^{N-1} C[\mathbf{x}(k), \mathbf{u}(k)] \tag{3.7}$$

subject to the following constraints

$$\mathbf{x}(k+1) = \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k)] \tag{3.8}$$

where

$$\mathbf{x}(0) = \mathbf{x}_s \tag{3.9}$$

$$\mathbf{x}(N) = \mathbf{x}_g \tag{3.10}$$

$$\mathbf{u}(k) \in \mathbf{U} \tag{3.11}$$

$$\mathbf{x}(k) \in \mathbf{X} \tag{3.12}$$

here $\mathbf{U}$ is defined to be the admissible decision space and $\mathbf{X}$ is the state space (or nodes in a search graph).

The problem of minimising equation 3.7 over all stages subject to the constraints can be simplified using the principle of optimality. In other words, the calculation involved in the dynamic programming algorithm can be reduced to the following recurrence relation which involves the calculation of an optimal value function

$$V(\mathbf{x}, k) = \min_{\mathbf{u}(k) \in U} [C[\mathbf{x}, \mathbf{u}] + V(\mathbf{f}[\mathbf{x}, \mathbf{u}], k+1)] \tag{3.13}$$

This effectively states that the optimal value for a sequence of $k$ stages is expressed in terms of its value for preceding $k+1$ and its value at stage $k$. It can be iteratively evaluated backward to yield the solution, starting with the condition

$$V(\mathbf{x}, N) = C(\mathbf{x}(N)) \tag{3.14}$$

Consider now the application of the above algorithm in the path planning problem, shown in Figure 3.8. The problem may be stated as follows: make a sequence of decisions $\mathbf{u} = \{u(0), u(1), u(2)\}$ to find an optimal path from $\mathbf{n}_0$ to the goal $\mathbf{n}_g$ to minimise the total cost

$$J = C[\mathbf{n}_g] + \sum_{k=0}^{2} C[\mathbf{x}(k), \mathbf{u}(k)] \tag{3.15}$$

Starting at $k = 3$ with a specified cost $C(3)$. In this subproblem the initial point is already the terminal point. No decision is needed and we have

$$V(\mathbf{n}_g, 3) = C(3) = 0 \tag{3.16}$$

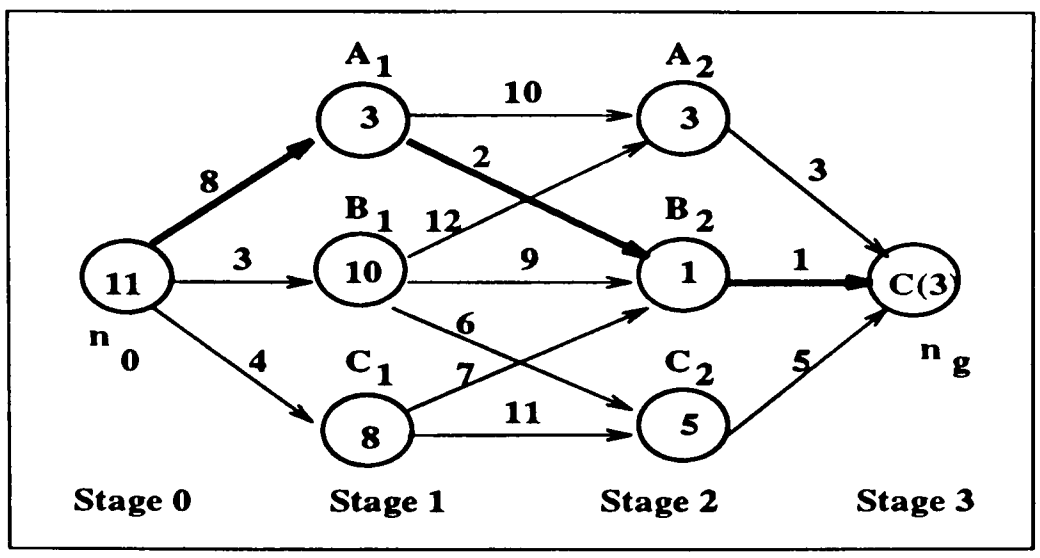



Figure 3.8: Example 1 of a route problem

To determine the other cost functions, we work backward one step at a time using the principle of optimality.

*Stage 2:*

$$V(A,2) = 3 + V(\mathrm{n}_g,3) = 3 \tag{3.17}$$

$$V(B,2) = 1 + V(\mathrm{n}_g,3) = 1 \tag{3.18}$$

$$V(C,2) = 5 + V(\mathrm{n}_g,3) = 5 \tag{3.19}$$

*Stage 1:*

$$\begin{aligned} V(A,1) &= \min[10 + V(A,2), 2 + V(B,2)] \\ &= \min[13,3] = 3 \end{aligned} \tag{3.20}$$

$$\begin{aligned} V(B,1) &= \min[12 + V(A,2), 9 + V(B,2), 6 + V(C,2)] \\ &= \min[15,10,11] = 10 \end{aligned} \tag{3.21}$$

$$\begin{aligned} V(C,1) &= \min[7 + V(B,2), 11 + V(C,2)] \\ &= \min[8,16] = 8 \end{aligned} \tag{3.22}$$

*Stage 0:*

$$\begin{aligned} V(\mathrm{n}_0,0) &= \min[8 + V(A,1), 3 + V(B,1), 4 + V(C,1)] \\ &= \min[11,13,12] = 11 \end{aligned} \tag{3.23}$$

The values of $V(\mathbf{x}, k), k = 0, 1, 2$, that are calculated above are written into the circles in Figure 3.8, which are the minimum total costs from the state $\mathbf{x} = \{A, B, C\}$ to the terminal point $\mathbf{n}_g$. We now have enough information to find the optimal path which is $\mathbf{n}_0 \rightarrow A_1 \rightarrow B_2 \rightarrow \mathbf{n}_g$, as shown by the heavy lines in Figure 3.8. It is worth noting from Figure 3.8 that the difference between the encircled numbers at the two ends of any portion $(i, j)$ along the optimal path is equal to the cost incurred in traversing that portion path, i.e.

$$V(i, k) - V(j, k + 1) = C_{ij} \tag{3.24}$$

Note that the dynamic programming algorithm described above is deterministic in a sense that the cost functions are perfectly known *a priori*. In the next section, we will show how it can be used in a stochastic planning problem. In our case, the cost functions are not known perfectly since dynamical changes in a real world.

### 3.5.4 Path Generation

Due to the random disturbance in a real world, the planning problem we deal with is stochastic in nature and the cost function associated with each arc in the weighted graph is not precisely known. Instead, the uncertainty part of cost functions in Equation 3.2 are estimated by means of statistical models which we define later. In this case, the optimal planning of a mobile robot system becomes: given an initial state $\mathbf{x}_s$ and the goal state $\mathbf{x}_g$, an optimal path is found by chosen a sequence of admissible decisions

$$\pi = \{\mathbf{u}(0), \mathbf{u}(1), ..., \mathbf{u}(N - 1)\} \tag{3.25}$$

in such a way as to minimise the accumulated cost (expectation) expressed by the summation

$$J = E\{C[\mathbf{x}(N)] + \sum_{k=0}^{N-1} C[\mathbf{x}(k), \mathbf{u}(k), \mathbf{w}(k)]\} \tag{3.26}$$

subject to the system equation constraints

$$\mathbf{x}(k + 1) = \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k), \mathbf{w}(k)] \tag{3.27}$$

where $k = 0, 1, ..., N-1$, $\mathbf{x}(k)$ is the current state of the robot, $\mathbf{x}(k+1)$ is the next state of the robot, $\mathbf{u}(k)$ is the decision based on the current state, and $\mathbf{w}(k)$ represents uncertainty in system state. The transition function $\mathbf{f}[\cdot]$ relates a state and a decision at a given stage $k$ to the succeeding stage $k+1$.

Note that the notation $E\{\cdot\}$ is used for the expectation operation. It is motivated by the certainty equivalence principle where a given stochastic system is replaced by a corresponding deterministic system by substituting expected values for random variables.

As a result, the recursive equation 3.13 becomes

$$E[V(\mathbf{x}, k)] = \min_{\mathbf{u}(k) \in U} [E[C(\mathbf{x}, \mathbf{u})] + E[V(\mathbf{f}[\mathbf{x}, \mathbf{u}], k+1)]] \tag{3.28}$$

However, the above recursive equation is only suitable for the applications where the nodes are grouped into stages such that each arc leads from a node in one stage, $k$, to a node in the next stage, $k+1$, as shown in Figure 3.8. In our case, the nodes can not be grouped in the same way, see Figure 3.6. Therefore, an equivalent recursive relation can be given as follows

**Definition 4** *Let $i$ be the current state (or node), $i \in \mathbf{N}$, $x_k$ the state adjacent to $i$ which is reached by the decision-making process at state $i$, $x_k \in \mathbf{N}$, and $j$ the final state. $\mathbf{E}(C_{ix_k})$ denotes the expected cost to move from $i$ to $x_k$. $\mathbf{E}(C_{x_kj})$ is the expected minimum cost to reach the final state $j$ from $x_k$. $\mathbf{E}(C_{ix_kj})$, or $\mathbf{E}(C_{ij})$ for simplification, is the expected minimum cost to go from $i$ to $j$ via $x_k$. The optimal decision $u^*$ at state $i$ to reach the goal $j$ is the decision that leads to the lower bound*

$$\mathbf{E}(C_{ij}) = \min_{x_k \in \mathbf{N}} \{\mathbf{E}(C_{ix^k}) + \mathbf{E}(C_{x_kj})\} \tag{3.29}$$

As a matter of fact, Equation 3.29 is the application of the principle of optimality. It can be rewritten as the following equation:

$$\mathbf{E}(C_{ij}) = \min_{x_k \in \mathbf{N}} \mathbf{E}\{C_{ix_1j}, C_{ix_2j}, ..., C_{ix_kj}, ...\} \tag{3.30}$$

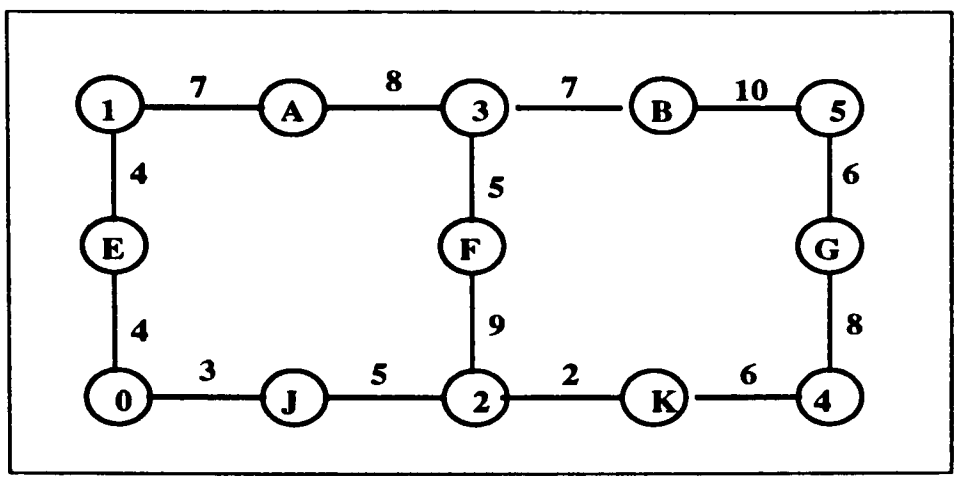



Figure 3.9: Example 2 of a route problem

where $x_k = i$ for direct connection between $i$ and $j$, and $x_k \neq j$.

It is clear from the above equation that to search an optimal path from node $i$ to node $j$ through intermediate nodes, the *Dynamic Programming* process ignores all those paths from node $i$ to any intermediate nodes except for the one with the minimum cost. The solution is obtained by recursive implementation of Equation 3.29 or Equation 3.30 from the goal position backwards to the start position.

Let us now consider a planning problem shown in Figure 3.9 which is a part of Figure 3.6. It should be noticed that the time cost associated with each arc is assumed to be the mean value.

In this problem, assume that the mobile robot searchs a path to minimise the cost of reaching the destination $G$ from its current location, node 1. The decision is the choice of the node to be reached. From Figure 3.9 we can see, the expected minimum cost from node 3 to the destination $G$ is

$$\begin{aligned} \mathbf{E}(C_{3G}) &= \min\{\mathbf{E}(C_{3BG}), \mathbf{E}(C_{3FG})\} \\ &= \min\{23, 30\} = 23 \end{aligned} \tag{3.31}$$

and the optimal decision at node 3 is to go to node B. Similarly, the expected minimum cost from node 2 to the destination $G$ is

$$\mathbf{E}(C_{2G}) = \min\{\mathbf{E}(C_{2KG}), \mathbf{E}(C_{2FG})\}$$

$$= \min\{16, 37\} = 16 \tag{3.32}$$

and the optimal decision at node 2 is to go to node K. Consider that the the current position of the robot is at node 1, the expected minimum cost to reach the destination $G$ is

$$\begin{aligned} \mathbf{E}(C_{1G}) &= \min\{\mathbf{E}(C_{12G}), \mathbf{E}(C_{13G})\} \\ &= \min\{32, 38\} = 32 \end{aligned} \tag{3.33}$$

and the optimal decision at node 1 is to go to node 2. Therefore, the optimal solution is

$$1 \rightarrow E \rightarrow 0 \rightarrow J \rightarrow 2 \rightarrow K \rightarrow 4 \rightarrow G \tag{3.34}$$

The dynamic planning algorithm that we use is based on recursion relation defined above. In other words, an optimal path is generated by minimising the total cost from the goal to the start position based on the weighted search graph shown in Figure 3.6. Since the uncertainty of encountering unexpected obstacles is taken into account in 3.29, the resulting path will be the path of both the minimum cost and the minimum chance of collision. In the next section, we describe more details about coping with uncertainty.

## 3.6 A Probabilistic Approach

In a dynamically changing environment, it is desirable to provide a mobile robot with the ability to estimate the likely success of a path using data based on previous traversals. The obstacle information provided by sensors is used not only for obstacle avoidance and path replanning, but also for future decision-making. In this way, the uncertainty and the risk associated with a decision can be reduced. In this section, we show how statistical models are built for unexpected obstacles and how Bayes' theorem is used to enable the robot to learn.

### 3.6.1 Assumptions

Assume that an obstacle can appear at a random location in the robot's environment. That is, the presence of unexpected obstacles is a discrete random event, say described by $W$. For a predefined

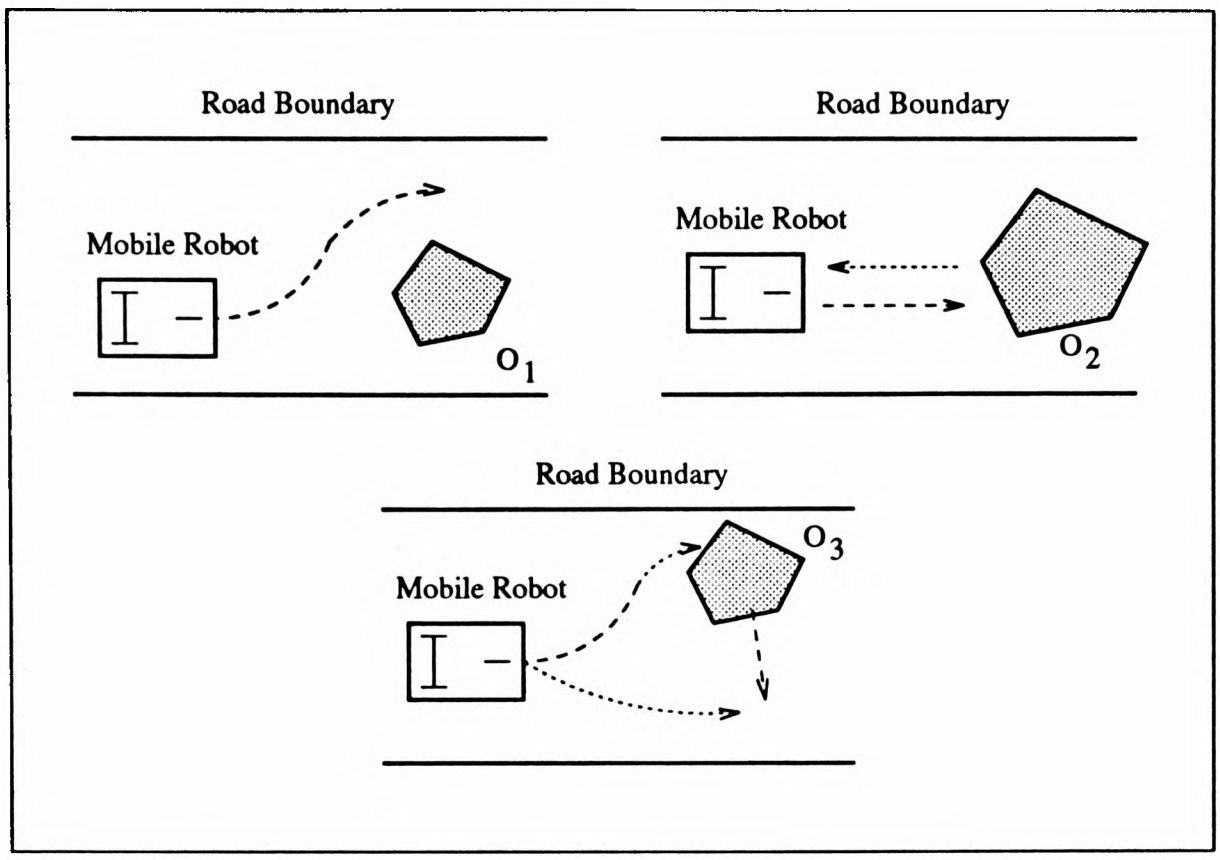



Figure 3.10: Random static and moving obstacles

path, unexpected obstacles can be classified into the following three situations:

- $O_1$ — a randomly-positioned static obstacle that partially blocks the robot's path.
- $O_2$ — a randomly-positioned static obstacle that totally blocks the robot's path.
- $O_3$ — a randomly-moving obstacle that crosses the robot's path.

These different situations have different effects on the robot's motion, shown in Figure 3.10. $O_1$ causes the robot to detour around the detected obstacle, but the robot continues its forward motion. $O_2$ forces the robot to stop and pay the extra cost to backtrack. However, $O_3$ forces the robot to change its speed, to speed up or slow down, as well as its steering angle in order to avoid a collision. In fact, $O_1$ and $O_3$ can be dealt with in the same way since the path is still passable.

**Definition 5** *Let $m$ be the type of encountered obstacles (1 for obstacle $O_1$; 2 for obstacle $O_2$, 3 for obstacle $O_3$), and $\hat{t}_{ij}^m$ the average time spent handling instances of obstacles of type $O_1$, $O_2$ or $O_3$ encountered during previous traversals of the arc. $\hat{v}_{ij}^m$ denotes the mean number of instances of*

*an $O_1$, $O_2$, or $O_3$ encountered so far along this arc. Then, the uncertainty cost, $u_{ij}$, between the nodes $i$ and $j$ is defined as:*

$$u_{ij} = \sum_{m=1}^{3} u_{ij}^m = \sum_{m=1}^{3} \hat{t}_{ij}^m * \hat{v}_{ij}^m = \hat{t}_{ij}^1 * \hat{v}_{ij}^1 + \hat{t}_{ij}^2 * \hat{v}_{ij}^2 + \hat{t}_{ij}^3 * \hat{v}_{ij}^3 \tag{3.35}$$

We assume a prior time cost for each type of obstacle and a prior expectation of encountering unexpected obstacles $O_1$, $O_2$, and $O_3$. Then, they are updated dynamically according to the available sensory information. In this way, dynamic changes in the environment are quantified as a cost to be taken into account during planning.

### 3.6.2 Estimate of Average Time Cost

In [Hu *et al.*, 1991a], in an initial exploration of these ideas, we simply apply constant gain to the uncertainty costs. This depends heavily upon prior knowledge about the obstacles' sizes. Here, we refine the approach and choose a more realistic method to estimate the average time for handling a $O_1$, $O_2$, or $O_3$. It relies on sensor data rather than prior knowledge. Suppose the uncertainty cost continues to be measured during the robot's motion and we wish to continuously update our estimate of $\hat{t}_{ij}^m$ with new observations, $z^m$, at each time, $k+1$, when any obstacle is encountered. We adopt the sample mean to estimate the average time $\hat{t}_{ij}^m$, which has the following recursive format:

$$\hat{t}_{ij}^m(k+1) = \frac{1}{k+1} \sum_{h=1}^{k+1} z_h^m = \hat{t}_{ij}^m(k) + \frac{1}{k+1}[z^m(k+1) - \hat{t}_{ij}^m(k)] \tag{3.36}$$

where $m$ is the type of encountered obstacles defined in Definition 5, and $z^m = \{z^1, z^2, z^3\}$ is the time costs measured to handle an obstacle $O_1$, $O_2$, or $O_3$ respectively.

### 3.6.3 Bayesian Estimation

The changing character of the environment is classified into three types of obstacles, assumed as discrete random events $W$ in Section 3.6.1. Suppose that each random event has a certain type of probability distribution, but one of the real-valued parameters of the probability distribution

function is unknown and random. Then, Bayesian estimation is employed to estimate the unknown parameter, for the following reasons:

- The *a priori* information about the unknown parameter may be utilised to achieve more efficiency.
- The estimation is made recursively so that more accurate estimation is obtained as more observations are available.

The Bayesian approach has been adopted in various optimisation problems involving unknown constants or stochastic processes with imperfectly known statistical properties [Bellman and Kalaba, 1960], [Lin and Yau, 1967]. Typically, this approach assumes an *a priori* distribution of the unknown parameter of the system on some parameter space. This is updated by the Bayes rule to obtain a posterior distributions on that parameter space if and when new information on the unknown parameter becomes available. The key idea is that of using the measurement to update the knowledge of the parameter being estimated. It has sometimes been termed a *Bayesian learning* process.

Let $a$ be the unknown real-valued parameter which defines the probability distribution function of the random event $W$, $\pi(a)$ the *a priori* probability density function (PDF). Based on an observation $z$, Bayes theorem describes how to compute the posterior PDF over the unknown parameter $a$ by

$$\pi(a|z) = \frac{p(z|a)\pi(a)}{\sum_a p(z|a)\pi(a)} = \frac{p(z|a)\pi(a)}{p(z)} \tag{3.37}$$

where $p(z|a)$ is known as the *likelihood function* (or the measurement density) and is used as a measure of how likely a parameter value is, given the observation that has been made. The prior PDF $\pi(a)$ is mapped into the posterior PDF $\pi(a|z)$ by the ratio of the measurement density $p(z|a)$ to the marginal density $p(z)$. In other words, the posterior PDF consists of information from both the subjective prior PDF and the objective measurement density and expresses our degree of belief in the location of the unknown parameter after observation is made available.

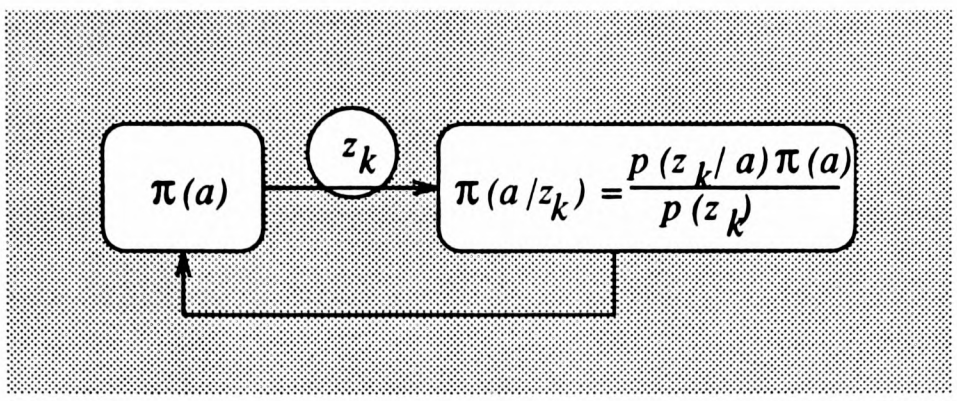



Figure 3.11: The data $z_k$ used to map the prior density into the posterior density recursively. Note that the prior density of the current stage is the posterior density at the previous stage.

This updating process can be iterated over time using observations $z_k = \{z(1), z(2), ..., z(k)\}$, as shown in Figure 3.11. Hence, Equation 3.37 becomes a recursive form as follows

$$\pi(a|z_k) = \frac{p(z_k|a)\pi(a)}{\sum_a p(z_k|a)\pi(a)} \tag{3.38}$$

In Bayesian estimation of a real-valued parameter $a$, the quality of the estimate $\hat{a}$ is measured by the real-valued loss function $L[a, \hat{a}]$. A typical loss would be the quadratic function which is easier to deal with

$$L[a, \hat{a}] = (a - \hat{a})^2 \tag{3.39}$$

The posterior expected loss, i.e. the conditional Bayes risk, is then

$$R_B(a, \hat{a}) = \int (a - \hat{a})^2 \pi(a|z_k) da \tag{3.40}$$

The value of $\hat{a}$ which minimises this can be found by expanding the quadratic expression, differentiating with respect to $\hat{a}$, and setting to zero as follows:

$$\frac{d}{d\hat{a}}[\int (a^2 - 2a\hat{a} + \hat{a}^2)\pi(a|z_k) da] = 0$$

$$-2 \int a\pi(a|z_k) da + 2\hat{a} = 0$$

Solving for $\hat{a}$ gives the following result

**Result 1** *The Bayes estimator under quadratic loss is the mean of the posterior PDF of a given* $z_k$

$$\hat{a}_B(z_k) = \int a\pi(a|z_k) da = E[a|z_k] \tag{3.41}$$

As a result, Bayes estimation under quadratic loss reduces to the computation of the mean of the posterior PDF $\pi(a|z_k)$. Since the Bayes risk under quadratic loss is in fact mean-squared error, Bayes estimator, which is the conditional mean of $a$ given $z_k$, minimises mean-squared error. It can be shown that the Bayes estimator under quadratic loss is an unbiased estimator of $a$ as follows

$$E[\hat{a}_B(z_k)] = \int \hat{a}_B(z_k)p(z_k)dz_k = \int [\int a\pi(a|z_k)da]p(z_k)dz_k = E[a] \qquad (3.42)$$

The Bayes estimator under quadratic loss is also the minimum variance unbiased estimator of the random parameter $a$ [Scharf, 1990]. The proof of the convergence of the Bayesian estimation and uniqueness can be found in [Aoki, 1967]. We are now in a position to solve the problem of how an unknown parameter should be estimated from available data using Bayes estimation described above so that the estimated parameter can be fit the probability model for encountering unexpected obstacles.

### 3.6.4 Statistical Models for Unexpected Obstacles

The application of statistics to the planning problem enables the uncertainty to be assigned a measure, and alternative paths to be compared quantitatively. In this section, we address how to use Bayesian estimation described above to build the statistical models to estimate the mean number, $\hat{v}_{ij}^m$, of the unexpected obstacles $O_1$, $O_2$ and $O_3$ encountered.

**Mean of Encountering $O_1$**

Assume that the presence of an unexpected $O_1$ is a discrete random event described by $W^1$. $O_1$ may appear at any time along any partial path $(i,j)$. The numbers of unexpected $O_1$ in any part of different path portions are assumed independent and do not favour one epoch of time over another. Naturally, a *Poisson* distribution is chosen to model the probability distribution of $W^1$. It has been used by traffic engineers as a model for traffic control [Rice, 1988].

$$p(W^1 = k|\lambda_1) = \frac{\lambda_1^k}{k!}e^{-\lambda_1} \quad (k = 0,1,2,...) \qquad (3.43)$$

where the parameter $\lambda_1 > 0$ is initially unknown.

To estimate the parameter $\lambda_1$, we use Bayes' estimator described above and assume that $\lambda_1$ has a conjugate prior density function subject to the *Gamma* distribution, i.e. $\lambda_1 \sim \Gamma(\alpha_1, \beta_1)$:

$$\pi(\lambda_1) = \Gamma(\alpha_1, \beta_1) = \begin{cases} \frac{\beta_1^{\alpha_1}}{\Gamma(\alpha_1)} \lambda_1^{\alpha_1 - 1} e^{-\beta_1 \lambda_1} & (\lambda_1 > 0) \\ 0 & (\lambda_1 \leq 0) \end{cases} \tag{3.44}$$

where: $\alpha_1 > 0$, and $\beta_1 > 0$.

The *Gamma* density function depends on two parameters, $\alpha_1$ and $\beta_1$, providing a fairly flexible class for modelling nonnegative random variables. It maps to our application as follows: $\alpha_1$ is interpreted as the number of obstacles encountered on this portion of the path during $\beta_1$ traversals. The choice of the conjugate form for $\lambda_1$ is mostly for computational convenience, since the posterior can be evaluated very simply, i.e. having a analytic solution. Of course, other forms of prior distribution can be used, but almost surely cause the difficulty of computation. That is, numerical methods of integration would have to be used to evaluate the posterior [Rice, 1988].

Suppose that $r_1$ obstacles $O_1$ are encountered the first time that the path is traversed. Then the posterior density can be calculated using Bayes' theorem as follows:

$$\pi(\lambda_1 | W^1 = r_1) = \frac{p(W^1 = r_1 | \lambda_1)\pi(\lambda_1)}{\int_0^{+\infty} p(W^1 = r_1 | \lambda_1)\pi(\lambda_1) d\lambda_1} \tag{3.45}$$

where $\lambda_1$ is the parameter being estimated, $W^1 = r_1$ is the current evidence, $\pi(\lambda_1)$ is the *a priori* probability distribution of the parameter, and $p(W^1 = r_1 | \lambda_1)$ is the probability that the evidence would be present given that the path is traversed. $\pi(\lambda_1 | W^1)$ is what we need for decision-making, namely the conditional probability that the path is in the first traversal in light of the evidence.

From Equations 3.43 and 3.44, we have

$$\begin{aligned} \pi(\lambda_1 | W^1 = r_1) &= \frac{\frac{\lambda_1^{r_1}}{r_1!} e^{-\lambda_1} \frac{\beta_1^{\alpha_1}}{\Gamma(\alpha_1)} \lambda_1^{\alpha_1 - 1} e^{-\beta_1 \lambda_1}}{\int_0^{+\infty} \frac{\lambda_1^{r_1}}{r_1!} e^{-\lambda_1} \frac{\beta_1^{\alpha_1}}{\Gamma(\alpha_1)} \lambda_1^{\alpha_1 - 1} e^{-\beta_1 \lambda_1} d\lambda_1} \\ &= \frac{\lambda_1^{\alpha_1 + r_1 - 1} e^{-(\beta_1 + 1)\lambda_1}}{\int_0^{+\infty} \lambda_1^{\alpha_1 + r_1 - 1} e^{-(\beta_1 + 1)\lambda_1} d\lambda_1} \\ &= \frac{(\beta_1 + 1)^{\alpha_1 + r_1}}{\Gamma(\alpha_1 + r_1)} \lambda_1^{(\alpha_1 + r_1) - 1} e^{-(\beta_1 + 1)\lambda_1} \\ &= \Gamma(\alpha_1 + r_1, \beta_1 + 1) \end{aligned} \tag{3.46}$$

where $r_1 = 0, 1, 2, ....$ The posterior density for $\lambda_1$ is also *Gamma* but with parameters $\alpha_1 + r_1$ and $\beta_1 + 1$.

This updating process can be iterated over time using independent observations when the path is traversed repeatedly. We identify the path state at stage $k$ as $(\alpha_1^k, \beta_1^k)$. By iterating Equation 3.45 recursively, the posterior density at the $k$th stage can be denoted as:

$$\pi(\lambda_1|\alpha_1^k, \beta_1^k) = \Gamma(\alpha_1^k, \beta_1^k) = \Gamma(\alpha_1 + r_{1k}, \beta_1 + k) \tag{3.47}$$

where $r_{1k}$ is the number of obstacles that has been encountered during $k$ traversals of the path.

The mean and variance of the posterior *Gamma* density are

$$\hat{\lambda}_1 = E[\lambda_1|\alpha_1^k, \beta_1^k] = \frac{\alpha_1 + r_{1k}}{\beta_1 + k} \tag{3.48}$$

$$\sigma^2 = E[(\lambda_1 - \hat{\lambda}_1)^2|\alpha_1^k, \beta_1^k] = \frac{\alpha_1 + r_{1k}}{(\beta_1 + k)^2} \tag{3.49}$$

where $\alpha_1$ and $\beta_1$ are *a priori* information. $r_{1k}$ is the outcome of $W^1$ during $k$ traversals.

It is seen that the variance of $\lambda_1$ monotonically decreases as the number of observations $k$ increases. Hence, the value of $\lambda_1$ can be estimated with more certainty as more observations are made.

Finally we have the mean of the random variable $W^1$:

$$\hat{v}_{ij}^1 = E[W^1] = \hat{\lambda}_1 = \frac{\alpha_1 + r_{1k}}{\beta_1 + k} \tag{3.50}$$

Based on Equation 3.50, the expected number, $\hat{v}_{ij}^1$, of unexpected obstacles $O_1$ encountered along the partial path between nodes $i$ and $j$ is continuously updated, i.e. increased or decreased, by new observations $r_{1k} \geq 0$ when the robot travels along this portion of the path.

**Mean of Encountering $O_2$**

Suppose that the presence of an unexpected $O_2$ is a discrete random event described by $W^2$. The numbers of unexpected $O_2$ in any part of path portions $(i, j)$ are assumed independent and do not favour one epoch of time over another. Since $O_2$ is defined as a randomly-positioned static obstacle

that totally blocks the robot's path in Section 3.6.1, $W^2$ has two possible outcomes, namely 1 for its presence and 0 for its absence, with the probability $p$ for $W^2 = 1$, which we can use the *Bernoulli* distribution to model

$$f(W_2 = k) = p^k(1-p)^{1-k} \quad (0 \leq p \leq 1) \tag{3.51}$$

where $k$ is zero if no obstacle is present and one if there is an obstacle. The constant parameter $p$ is initially unknown and is to be estimated.

Assume that $p$ is an unknown random variable. Again, the Bayes estimator is used to estimate it from the observation of the outcomes of $W^2$. We assume that $p$ has a conjugate prior density which follows the $Beta$ distribution with parameters $(\alpha_2, \beta_2)$, i.e. $p \sim \mathcal{B}(\alpha_2, \beta_2)$:

$$\pi(p) = \begin{cases} \frac{\Gamma(\alpha_2+\beta_2)}{\Gamma(\alpha_2)\Gamma(\beta_2)} p^{\alpha_2-1}(1-p)^{\beta_2-1} & (0 \leq p \leq 1) \\ 0 & (otherwise) \end{cases} \tag{3.52}$$

where $\alpha_2$ and $\beta_2$ are positive parameters, with $\alpha_2$ interpreted as the number of the unexpected $O_2$ present, i.e. $W^2 = 1$, and $\beta_2$ as the number of unexpected $O_2$ absent, i.e. $W^2 = 0$.

Suppose that the first traversal along a portion of path is successful, then the posterior density is calculated using Bayes' theorem:

$$\begin{aligned} \pi(p|W^2=0) &= \frac{f(W^2=0|p)\pi(p)}{\int_0^1 f(W^2=0|p)\pi(p)dp} \\ &= \frac{(1-p)\frac{\Gamma(\alpha_2+\beta_2)}{\Gamma(\alpha_2)\Gamma(\beta_2)}p^{\alpha_2-1}(1-p)^{\beta_2-1}}{\int_0^1 (1-p)\frac{\Gamma(\alpha_2+\beta_2)}{\Gamma(\alpha_2)\Gamma(\beta_2)}p^{\alpha_2-1}(1-p)^{\beta_2-1}dp} \\ &= \frac{p^{\alpha_2-1}(1-p)^{\beta_2}}{\int_0^1 p^{\alpha_2-1}(1-p)^{\beta_2}dp} \\ &= \frac{\Gamma(\alpha_2+\beta_2+1)}{\Gamma(\alpha_2)\Gamma(\beta_2+1)}p^{\alpha_2-1}(1-p)^{(\beta_2+1)-1} \\ &= \mathcal{B}(\alpha_2, \beta_2+1) \end{aligned} \tag{3.53}$$

However, if the first traversal failed, the posterior density is calculated as follows:

$$\pi(p|W^2=1) = \frac{f(W^2=1|p)\pi(p)}{\int_0^1 f(W^2=1|p)\pi(p)dp}$$

$$= \frac{p^{\alpha_2}(1-p)^{\beta_2-1}}{\int_0^1 p^{\alpha_2}(1-p)^{\beta_2-1}dp}$$
$$= \mathcal{B}(\alpha_2+1,\beta_2) \tag{3.54}$$

Obviously, this updating process can be iterated over time using independent observations when the path is traversed. We can identify the state at stage $k$ as $(\alpha_2^k, \beta_2^k)$. So the posterior density of $p$ at the $k$th stage can be described as:

$$\pi(p|\alpha_2^k,\beta_2^k) = \mathcal{B}(\alpha_2^k,\beta_2^k) = \mathcal{B}(\alpha_2+h_1,\beta_2+h_2) \tag{3.55}$$

where: $h_1$ — the number of present $O_2$.

$h_2$ — the number of absent $O_2$.

$k = h_1 + h_2$ — the total traveling time.

The mean and variance of the posterior *Beta* density are

$$\hat{p} = E[p|\alpha_2^k,\beta_2^k] = \frac{\alpha_2+h_1}{\alpha_2+\beta_2+k} \tag{3.56}$$

$$\sigma^2 = E[(p-\hat{p})^2|\alpha_2^k,\beta_2^k] = \frac{(\alpha_2+h_1)(\beta_2+h_2)}{(\alpha_2+\beta_2+k)^2(\alpha_2+\beta_2+k+1)} \tag{3.57}$$

It is seen that the variance of $p$ monotonically decreases as the number of observations $k$ increases. Hence, the value of $p$ can be estimated with more certainty as more observations are made.

Finally we have the mean of the random event $W^2$:

$$\hat{v}_{ij}^2 = E[W^2] = \hat{p} = \frac{\alpha_2+h_1}{\alpha_2+\beta_2+k} \tag{3.58}$$

Based on Equation 3.58, the expected number, $\hat{v}_{ij}^2$, of unexpected obstacles $O_2$ encountered along the partial path between nodes $i$ and $j$ is continuously updated, i.e. increased or decreased, by new observation $W^2 = 0$ or $W^2 = 1$ each time the robot travels along this portion of the path.

**Mean of Encountering $O_3$**

Assume that the appearance of $O_3$ that crossed randomly in any portion of path is also a discrete

random event described by $W^3$. The events $W^3$ are supposed to be independent in any part of path portions. Again, the *Poisson* distribution is chosen to describe the probability distribution of $W^3$:

$$p(W^3 = k|\lambda_3) = \frac{\lambda_3^k}{k!}e^{-\lambda_3} \quad (k = 0, 1, 2, ...) \tag{3.59}$$

where the parameter $\lambda_3 > 0$ is initially unknown and is to be estimated.

We assume again that $\lambda_3$ has a conjugate prior density function subject to the *Gamma* distribution, i.e. $\lambda_3 \sim \Gamma(\alpha_3, \beta_3)$:

$$\pi(\lambda_3) = \Gamma(\alpha_3, \beta_3) = \begin{cases} \frac{\beta_3^{\alpha_3}}{\Gamma(\alpha_3)}\lambda_3^{\alpha_3-1}e^{-\beta_3\lambda_3} & (\lambda_3 > 0) \\ 0 & (\lambda_3 \leq 0) \end{cases} \tag{3.60}$$

where $\alpha_3 > 0$ and $\beta_3 > 0$. We interpret $\alpha_3$ as the number of obstacles encountered on any portion of the path during $\beta_3$ traversals.

Using Bayes' theorem, the posterior density of $\lambda_3$ after $k$ traversals of the path is:

$$\pi(\lambda_3|\alpha_3^k, \beta_3^k) = \Gamma(\alpha_3^k, \beta_3^k) = \Gamma(\alpha_3 + r_{2k}, \beta_3 + k) \tag{3.61}$$

where $r_{2k}$ is the number of obstacles encountered during $k$ traversals of the path.

The mean and variance of the posterior *Gamma* density are

$$\hat{\lambda}_3 = E[\lambda_3|\alpha_3^k, \beta_3^k] = \frac{\alpha_3 + r_{2k}}{\beta_3 + k} \tag{3.62}$$

$$\sigma^2 = E[(\lambda_3 - \hat{\lambda}_3)^2|\alpha_3^k, \beta_3^k] = \frac{\alpha_3 + r_{2k}}{(\beta_3 + k)^2} \tag{3.63}$$

It is seen that the variance of $\lambda_3$ monotonically decreases as the number of observations $k$ increases. Hence, the value of $\lambda_3$ can be estimated with more certainty as more observations are made.

Finally we have the mean of the random variable $W^3$:

$$\hat{v}_{ij}^3 = E[W^3] = \hat{\lambda}_3 = \frac{\alpha_3 + r_{2k}}{\beta_3 + k} \tag{3.64}$$

Based on Equation 3.64, the expected number, $\hat{v}_{ij}^3$, of unexpected obstacles $O_3$ encountered along the partial path between nodes $i$ and $j$ is continuously updated, i.e. increased or decreased, by new observations $r_{2k} \geq 0$ when the robot travels along this portion of the path.

### 3.6.5 Uncertainty Cost

Based on the statistical models, $\hat{v}_{ij}^1$ and $\hat{v}_{ij}^2$, $\hat{v}_{ij}^3$ and the average time cost, $\hat{t}_{ij}^1$, $\hat{t}_{ij}^2$, $\hat{t}_{ij}^2$, the uncertainty costs $u_{ij}^1$, $u_{ij}^2$ and $u_{ij}^3$ for avoiding unexpected obstacles $O_1$ and $O_2$ can be calculated using the following equations:

$$u_{ij}^1 = \hat{t}_{ij}^1 \frac{\alpha_1 + r_{1k}}{\beta_1 + k} \tag{3.65}$$

$$u_{ij}^2 = \hat{t}_{ij}^2 \frac{\alpha_2 + h_1}{\alpha_2 + \beta_2 + k} \tag{3.66}$$

$$u_{ij}^3 = \hat{t}_{ij}^3 \frac{\alpha_3 + r_{2k}}{\beta_3 + k} \tag{3.67}$$

Then, the total uncertainty cost, $u_{ij}$, for a portion of path between nodes $i$ and $j$ can be obtained using Equation (3.35). Figure 3.12 shows how $u_{ij}^1$ is updated during path executions. Curve 1 shows that the robot encounters $O_1$ obstacles on every traversal. Curve 2 shows that the $O_1$ obstacles are never encountered by the robot during its traversals. Curve 3 shows that obstacles $O_1$ appear after the first move, then disappear after the second move, and so on. A similar figure can be drawn for $u_{ij}^2$ and $u_{ij}^3$.

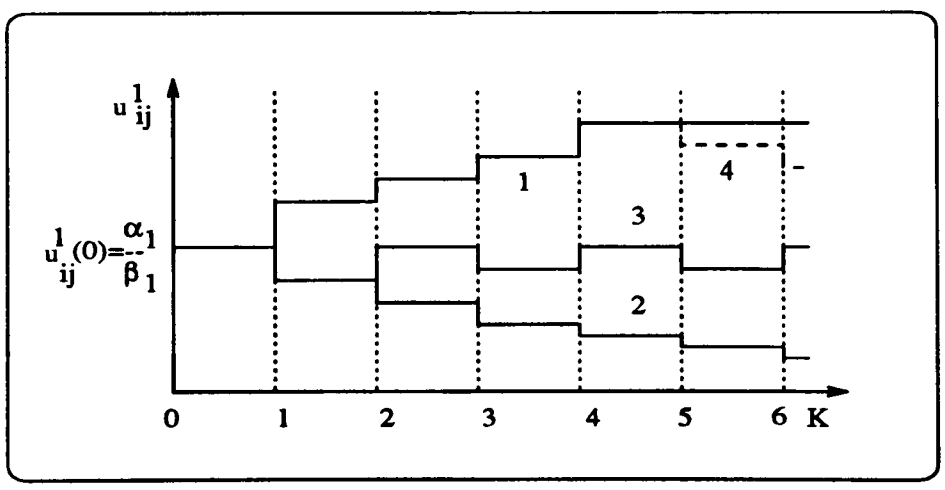



Figure 3.12: A graph for uncertainty cost $u_{ij}^1$

Now the cost $C_{ij}$ in the Equation (3.2) can be preset using the prior information $\alpha_1$, $\beta_1$, $\alpha_2$, $\beta_2$, $\alpha_3$, $\beta_3$, and updated using the observation information $r_{1k}$,$r_{2k}$, $k$, $h_1$, $h_2$ as the robot moves around. Then, we can use Equation (3.29) to search for an optimal path.

### 3.6.6 Modelling Persistence and Change

Since we have already assumed, in Section 3.3, that dynamic changes in the environment are measured by the on-board sensors during the motion, and as there are no other sensors to monitor the whole environment all the time, reasoning about dynamic changes in the environment requires a prediction of how long the uncertainty cost for each portion of the path, once estimated, will continue to remain true if there is no new information available. For example, curve 1 in Figure 3.12 keeps the same value after k=4, since this portion of path is no longer chosen and no further information about $O_1$ is available. This value will persist until the robot travels along this portion of the path again. But when (and why) should the robot return to this path? It is likely that the robot will choose this path when the costs of other paths to the goal increase to values which are no longer less than one travelling along this path. This is the situation studied in [Hu *et al.*, 1991a]. Here, we consider the problem of modelling persistence and change.

We propose a heuristic method to solve this problem by assuming that the uncertainty cost will decay exponentially over time when no new information is available. In other words, we employ a survivor function, $e^{-\mu t}$, to capture the tendency of the established uncertainty cost to become false [Dean and Wellman, 1991]. Then, Equations (3.65) and (3.67) become:

$$u_{ij}^1 = e^{-\mu(\Delta-1)}\hat{t}_{ij}^1 \frac{\alpha_1 + r_{1k}}{\beta_1 + k} \tag{3.68}$$

$$u_{ij}^2 = e^{-\mu(\Delta-1)}\hat{t}_{ij}^2 \frac{\alpha_2 + h_1}{\alpha_2 + \beta_2 + k} \tag{3.69}$$

$$u_{ij}^3 = e^{-\mu(\Delta-1)}\hat{t}_{ij}^3 \frac{\alpha_3 + r_{2k}}{\beta_3 + k} \tag{3.70}$$

where: $\mu$ — an exponential rate determined by experiments.

$\Delta$ — the time duration between two updating instances.

By adding a survivor function into the equations calculating the uncertainty costs, the $u_{ij}^1$, $u_{ij}^2$, and $u_{ij}^3$ no longer persist even though there is no new information available. As we see in Figure 3.12, the dashed line 4 shows the exponential decay in the uncertainty cost curve 1. Therefore, in

a few steps, the uncertainty cost between nodes $i$ and $j$ will decrease to zero. That means that the robot will once more try this path.

## 3.7 Dynamic Generation of Subgoals

In traditional planning systems, global path planning and local path planning are sequential processes. In other words, the local planner is idle when the global planner is busy, and vice versa. This causes a delay for the whole system, since the local planner, whenever a preplanned path is blocked, triggers the global planner and then has to wait for an alternative path. Concurrent processing of both planners is crucial for solving this delay problem. Concurrency has been implemented in our system design, as shown in Figures 2.15 and 2.16.

In our design, the global planner generates alternative subgoals dynamically when the robot is travelling along an preplanned optimal path. In other words, it is always assuming that the next node of the preplanned path may be blocked by unexpected obstacles. If it is true, the local planner can backtrack along these subgoals without delay. However, if nothing happens when the robot is approaching the next node, the alternative subgoals provided by the global planner will be ignored.

Figure 3.13 shows the global planning algorithm in OCCAM. **comm.proc()** is in charge of communication with the monitoring level to receive tasks and the navigator to obtain the current position of the robot and information of path execution, as shown in Figure 2.15. **update.graph.proc()** updates the weighted search graph based on available sensor data. In contrast, **dynamic.planning.proc()** receives tasks and the current position of the robot from **comm.proc()**, and plans an optimal path and the alternatives dynamically. Three procedures operate simultaneously.

**dynamic.planning.proc()** is further presented in Figure 3.14. As can be seen, the dynamic planning algorithm is active throughout the time the robot is moving from the current node to a goal, controlled by the local planner. Since unexpected obstacles may appear any time before

```
PROC global.planner.proc (CHAN OF ANY in1, out1, in2, out2)
    ... declarations of variables and internal channels
    ... SC update.graph.proc
    ... SC dynamic.planning.proc
    ... SC comm.proc
    SEQ
        ... initialisation
        WHILE TRUE
            PAR
                comm.proc (in1, out1, in2, out2, b1, b2, b3, b4)
                update.graph.proc (b1, b2, b5, b6)
                dynamic.planning.proc (b3, b4, b5, b6)
:
```

Figure 3.13: A OCCAM procedure for the global planner

```
PROC dynamic.planning.proc (CHAN OF ANY c1, c2, c3, c4)
    ... declarations of variables and internal channels
    ... get.current.node
    ... get.goal.node
    ... find.optimal.path
    SEQ
        ... initialization
        get.goal (c1, c2, goal.node)
        get.current.node (c1, c2, goal.node)
        find.optimal.path (current.node, goal.node, p.lst, p.len, c1, c2, c3, c4)
        last.node := current.node
        WHILE loop
            SEQ
                get.current.node (c1, c2, current.node)
                CASE current.node
                    goal.node
                        loop := FALSE
                    last.node
                        SKIP
                    ELSE
                        find.optimal.path (current.node, goal.node, p.lst, p.len, c1, c2, c3, c4)
:
```

Figure 3.14: A OCCAM procedure for the dynamic planning algorithm

reaching a goal, the global planner generates an alternative path which is a suboptimal solution. More precisely, subgoals that form a new path to reach the goal position are dynamically generated by the global planner when the robot is approaching next node of the current path. Both costs corresponding to travelling along the current path and an alternative path are sent to the local planner together. Therefore, if the robot encounters an unexpected obstacle on the way, the local planner will choose an action, either generating a detour path or choosing an alternative path to minimise the expected loss. More detail will be given in the next chapter.

## 3.8 Experimental Results

The strategy described above is implemented both in simulation and using real data. This section briefly presents the experimental results.

### 3.8.1 Simulation

We implemented the proposed path planning algorithm on a SUN workstation, using a interactive, graphics-based toolkit, SunView [Sun, 1988]. An environment model is set up, based on the geometrical description of the AGV laboratory, as shown in Figure 3.15. Simulated sonar sensors are used to report unexpected obstacles, though for simplicity we have not taken into account the problems caused by specularities with a real sonar sensor. The global path planning algorithm is written in the C language. Table 3.2 shows the initial conditions and the robot's input parameters used in the simulation.

Figure 3.16 shows the mobile robot is moving to the goal position along an optimal path planned by the global planning algorithm. Sonar sensors detect an obstacle $O_1$ which is unmapped. The mobile robot is controlled by the local planner to sidestep it. The robot successfully reaches the goal and reports the partial blockage between nodes $n_4$ and $n_k$ to the global planner to increase the corresponding cost function $u^1_{4k}$ accordingly. This information is taken into account in the next planning stage. Note that the costs for the other portions of the path are also updated by decreasing

Table 3.2: Simulation Input Data

| Description | Symbol | Preset Value |
|---|---|---|
| Prior Information | $\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3$ | 1, 4, 1, 4, 1, 4 |
| Initial data | $r_{1k}, r_{2k}, h_1$ | 0, 0, 0 |
| Average time(sec) | $t^1, t^2, t^3$ | 4, 20, 4 |
| Sample interval(sec) | $K$ | 0.5 |
| Decay rate | $\mu$ | 1 |
| Initial position(pixel) | $X_0, Y_0$ | 708, 298 |
| Gaol position(pixel) | $X_g, Y_g$ | 84, 298 |
| Vehicle speed(pixel/s) | $S^1$(straight), $S^2$(turn) | 18, 12 |

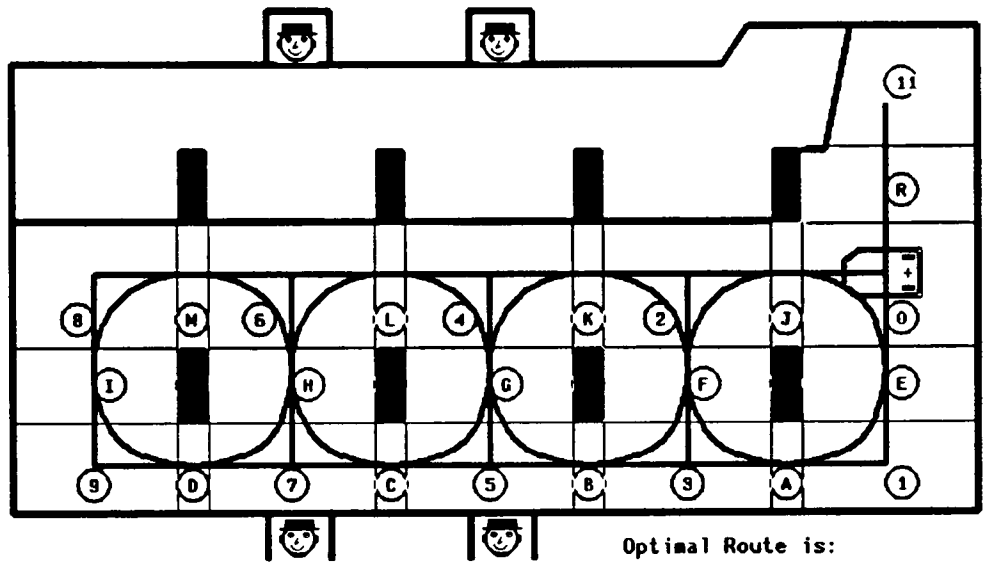



Figure 3.15: The model and partition of Oxford AGV Laboratory. Note that the partition of its free space and the central line of the defined roadway are presented

uncertainty costs.

Figure 3.17 shows a situation in which there are two unexpected obstacles on the way. Based on prior information and previous traversals, the robot computed the optimal path, a straight line, and moved towards the goal. When the sensors report an $O_2$ obstacle on the way between nodes $n_4$ and $n_k$, the robot has to backtrack along an alternative path planned dynamically by the global path planner. The global planner updates the cost function of the graph $G$, $C_{ij}$, when the robot moves around in the environment. The robot travels along the new path until it encounters another $O_2$ obstacle. Once again, the robot backtracks along the alternative one and eventually reaches the

goal.

Figure 3.18 shows that an optimal path is planned based on information gathered from previous traversals. Since the costs $u_{4k}$ and $u_{7c}$ are increased greatly during the last traversal, both portions are not chosen next time. In this way, the robot is able to adapt to changes in its environment, rather than repeatedly trying the original straight line path all the time. Note that the weighted search graph $G$ is updated dynamically when the information is available. The uncertainty costs $u_{4k}$ and $u_{7c}$ decrease exponentially during subsequent traversals even though those arcs are not chosen and no new sensor data is available. This is the problem of modelling persistence and change discussed above.

Figure 3.19 shows how the cost function of the arc joining $node4$ and $nodeK$ which changes during the following 25 traversals. The decay rate $\mu$ for graph 1 is chosen as 0.025. After 16 traversals, the cost $C_{4k}$ in graph 1 has decreased to the level that is no longer larger than choosing other arcs. Therefore, the robot once again chooses this arc, ending with backtracking again but learning new information. In contrast, the cost $C_{4k}$ in graph 2 decreases very quickly, and the arc is retried more often since the decay rate is chosen to be 0.1. In a similar way, Figure 3.20 shows the exponential decay of cost function $C_{7c}$.

### 3.8.2 Real-time Implementation

The strategy is implemented by the *Turtle* mobile robot in the Oxford AGV Laboratory in real time, shown in Figure 3.21. The global planner generates optimal paths based on prior information and the available sensor data. The local planner(avoiding level) uses an array of twelve sonar sensors as a *Safety Curtain* to detect and handle unexpected obstacles, and sends updating information to the global planner(upper level) to update the internal world model dynamically. In this way, the robot learns global knowledge about its environment.

Once the global planner generates an optimal path, say the straight line between the current position and the goal shown in Figure 3.22, it is sent to the AGV's GEM80 controller. In turn,

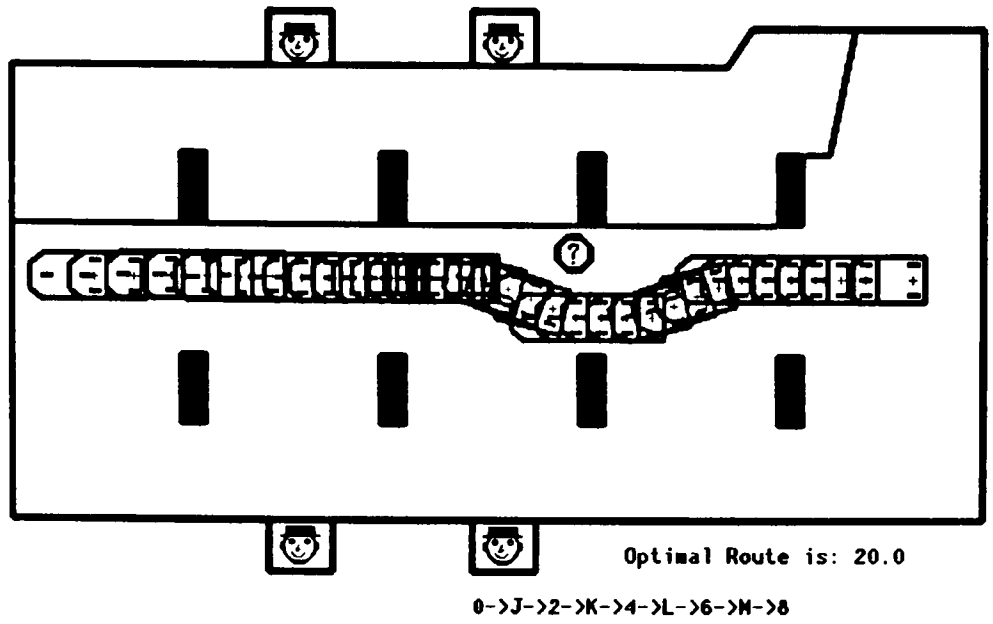



Figure 3.16: Avoiding type $O_1$ obstacle

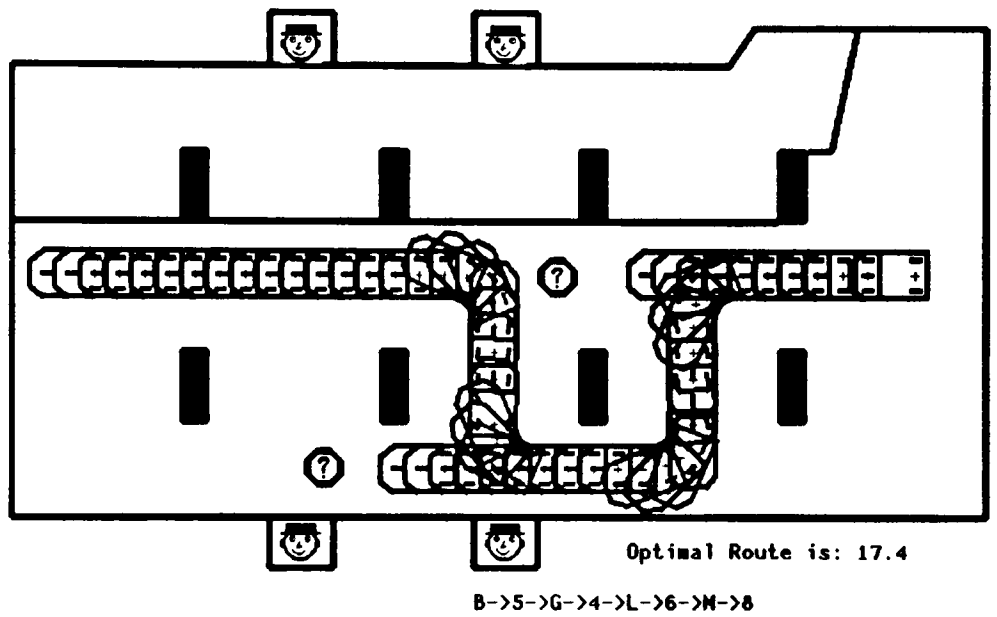



Figure 3.17: Avoiding type $O_2$ obstacles with two backtracking

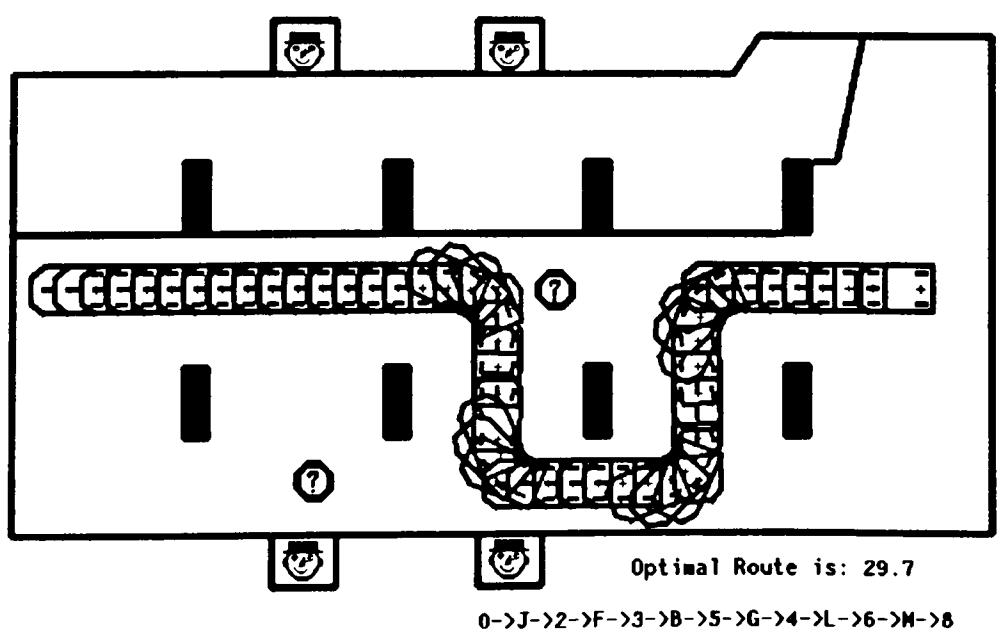



Figure 3.18: The robot travels along an optimal path based on information gathered from previous traversals

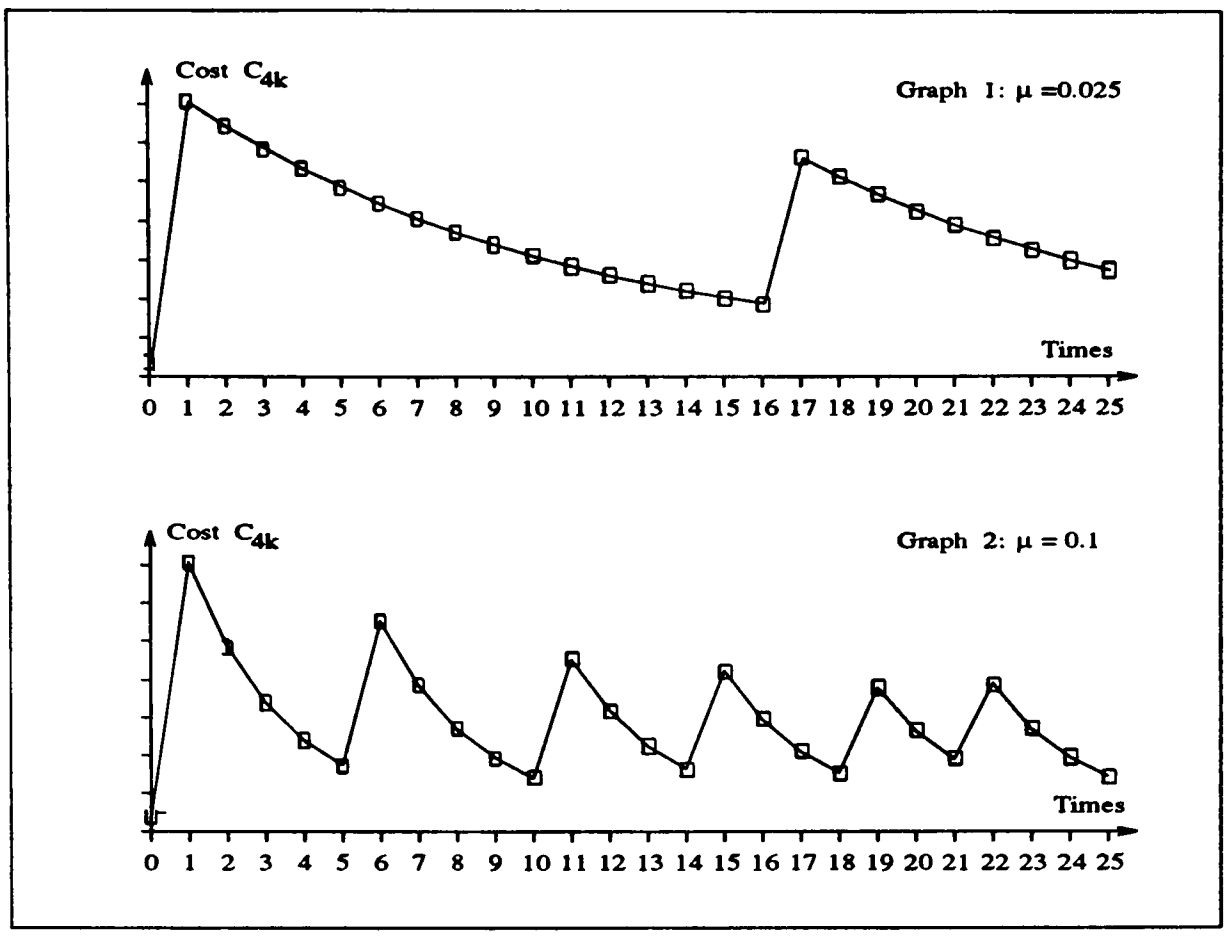



Figure 3.19: Exponential decay of cost function $C_{4k}$ associated with the portion of the path between $Node4$ and $NodeK$

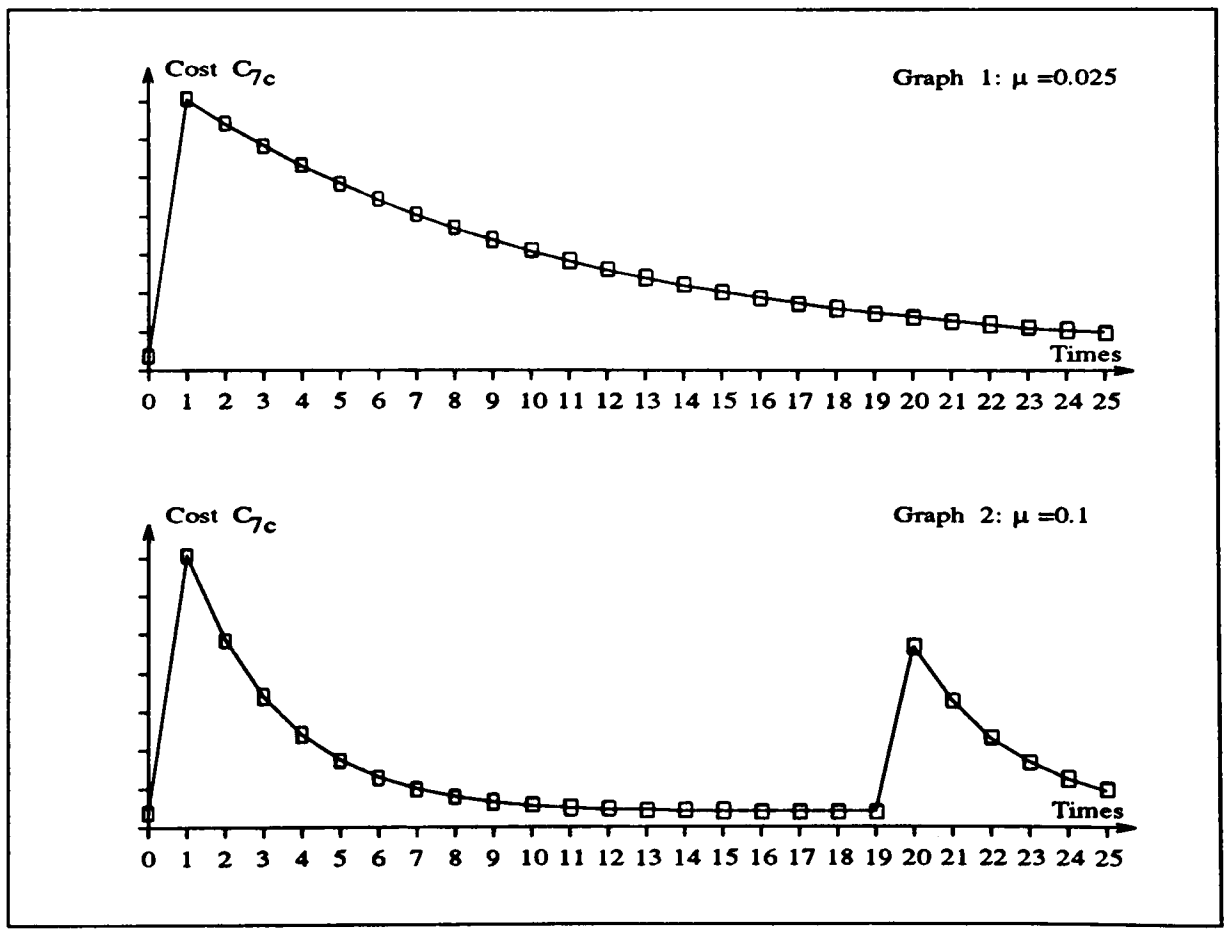



Figure 3.20: Exponential decay of cost function $C_{7c}$ associated with the portion of the path between $Node7$ and $NodeC$

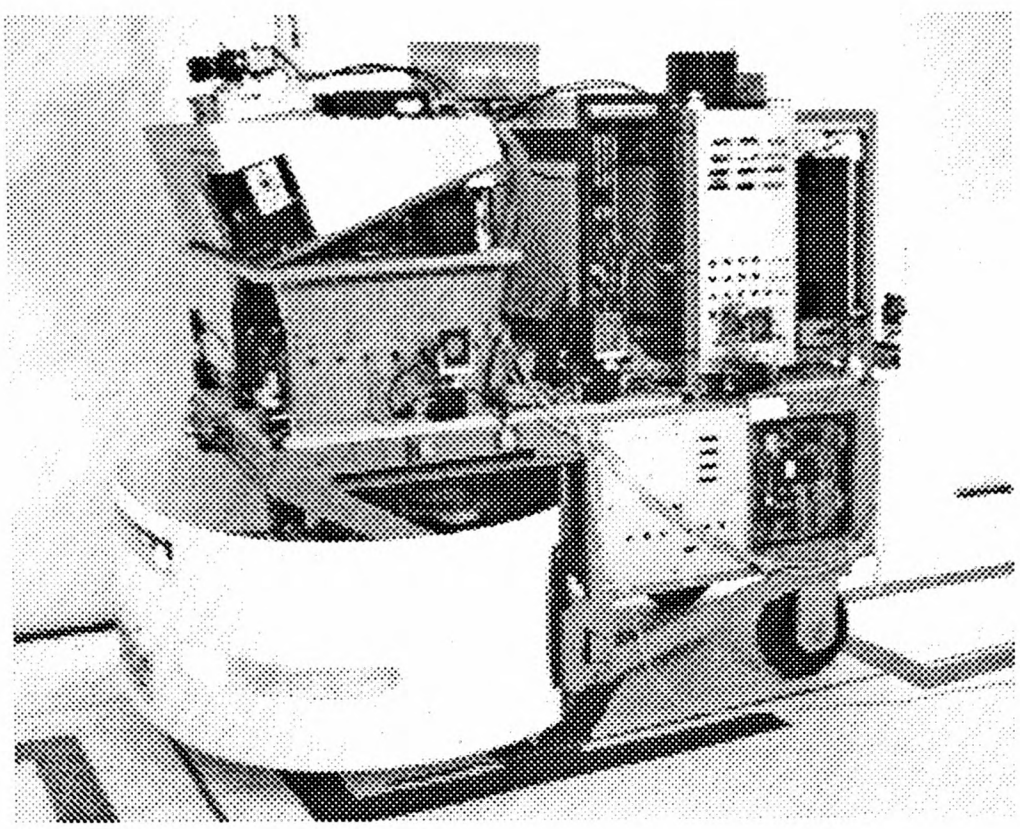


Figure 3.21: The *Turtle* mobile robot in Oxford AGV Laboratory

the *Turtle* robot will move along this path until the sonar sensors detect unexpected obstacles. If the planned path is completely blocked by an obstacle such as $O_2$ shown in Figure 3.22, the local planner requests an alternative path from the global planner. Then the Turtle backtracks along the alternative path to reach the goal. At the same time, the global planner increases the cost of the that portion of the path for the next planning stage. Note that costs for the other portions of the path that the robot travels along are decreased by the statistical models we built. If more obstacles are found, path search will continue in the same way until all paths have been attempted. If none is successful the mobile robot returns to the start node and reports failure.

Based on the statistical models we have proposed, the mobile robot is able to learn global knowledge from path execution. Therefore, if the mobile robot frequently encounters obstacles which block the way completely in a portion of the path, the cost of this partial path will quickly increase so that this partial path is removed from planning for some time. As its cost function decays exponentially, this portion of the path will be a candidate path later and the robot will try it again. Figure 3.23 shows that the robot chooses an arc path rather than the straight one because

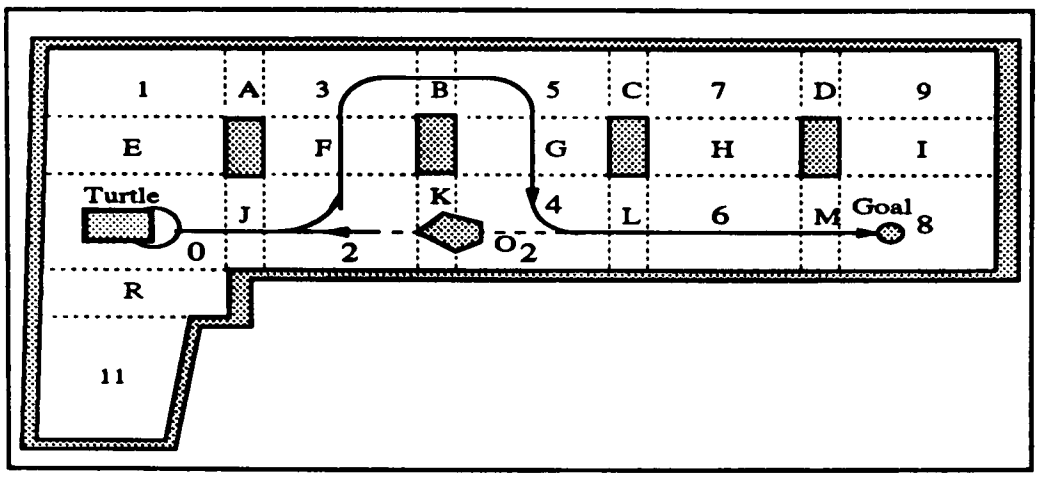



Figure 3.22: The *Turtle* chooses an alternative path to avoid collision with $O_2$ at node K. Note that the optimal path is straight line (dashed line).

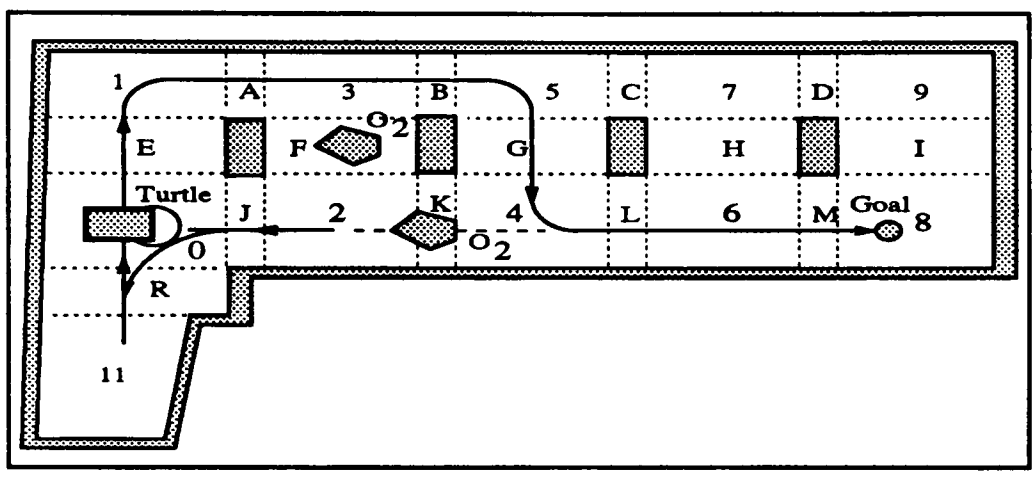


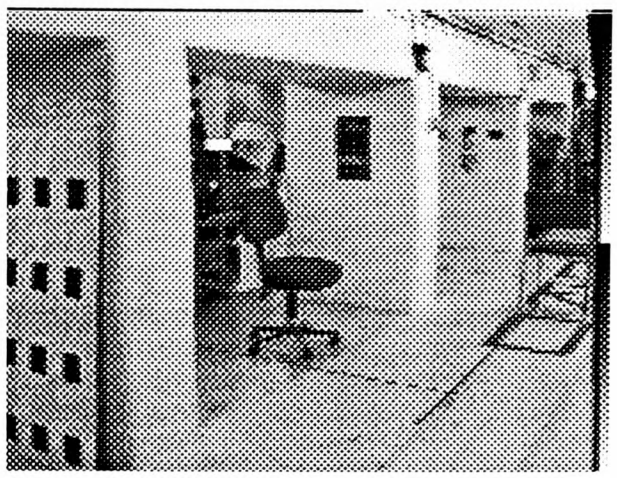

Figure 3.23: The *Turtle* backtracks along an alternative path to avoid two unexpected obstacles

obstacles have appeared frequently during past traversals. But the robot will try the straight line path from time to time, depending upon the exponential decay rate.

Figure 3.24 presents a sequence of 8 images to show that the Turtle mobile robot moves along the optimal path (straight line) towards the goal that is the end of the corridor. The sonar sensors detect an unexpected obstacle, i.e. a person who is standing in the way. The robot backtracks along the alternative path. This corresponds to the situation shown in Figure 3.22. In contrast, the image sequence in Figure 3.25 shows that when the Turtle travels along the optimal path (straight line) towards the goal, it encounters two obstacles which block the way. The Turtle has to choose another alternative path to reach the goal, which corresponds to the situation shown in Figure 3.23.

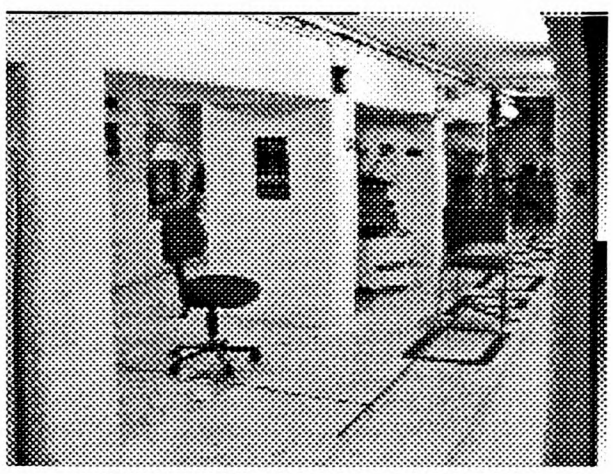

Figure 3.24: The *Turtle* mobile robot avoids a person who is standing on the way

1

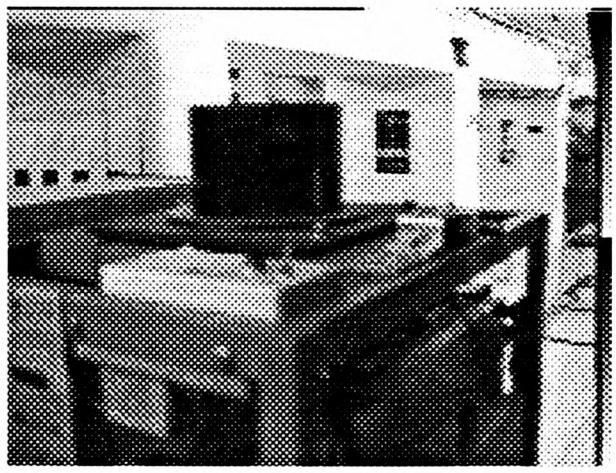


5

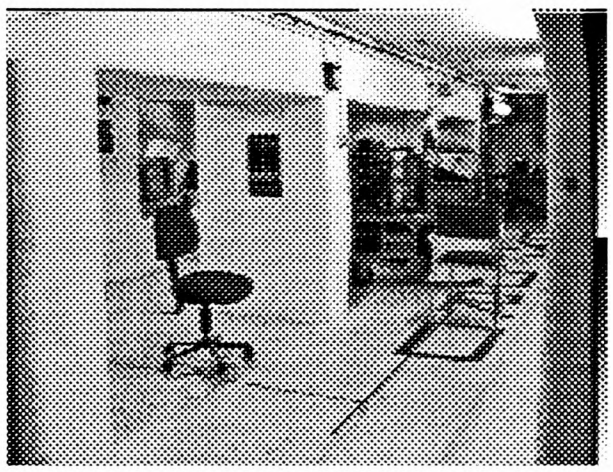
2



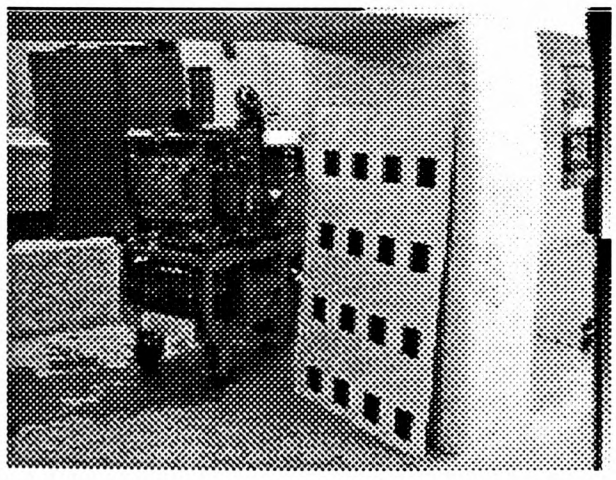
6



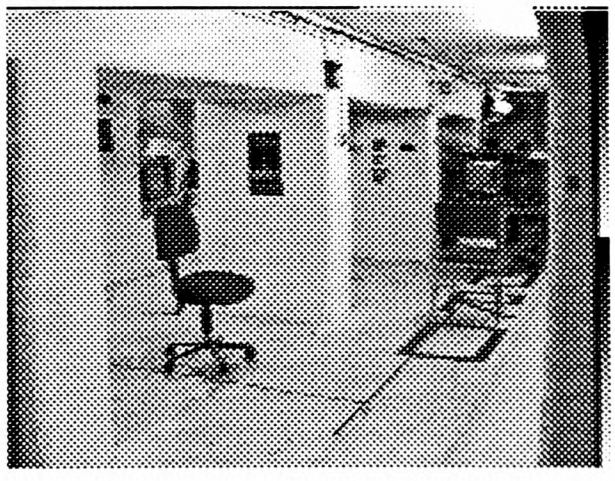
3



7



4



8

Figure 3.25: The *Turtle* mobile robot avoids two unexpected obstacles

## 3.9 Conclusions

This chapter presents a probabilistic approach to real-time global path planning for a mobile robot. This approach has been formulated to handle the uncertainty which arises in a dynamically changing environment. The proposed global planning algorithm differs from many previous approaches in a sense that:

- It operates in a closed-loop manner which is the perspective of a feedback control problem. It monitors the implementation of a planned path (either success or failure), updates the internal world model, and makes a prediction for an optimal path dynamically.
- No geometrical information is dealt with at this level. The free space and dynamic changes in the robot's environment is represented by a topological model, forming a weighted search graph. The weights in the search graph are composed of a deterministic cost (a static part of the environment) and an uncertainty cost (a dynamic part of the environment).
- The unknown changes of the environment are classified into three types of obstacles which are assumed to be random variables with unknown statistical parameters. The statistical models are built to quantise uncertainty and assign a merit to a path. The Bayesian approach is used to estimate the unknown parameters based on the sampled data during the traversals of the robot.
- The uncertainty cost for each portion of path is dynamically updated when the robot passes through. However, it will decay exponentially over time using a survivor function when no new information is available.
- An optimal path is planned using dynamic programming and the variable cost graph. Based on prior information and sensor data, the mobile robot is able to avoid unexpected obstacles and learn global knowledge from its traversals.

# Chapter 4

# Real-time Obstacle Avoidance

The capability of real-time obstacle avoidance is essential for the safe operation of mobile robots in a dynamically changing environment. This chapter investigates how a nonholonomic mobile robot can handle unexpected static obstacles while following an optimal path planned by the global path planner. To find an optimal solution of the problem, the obstacle avoidance problem is formulated as a decision theoretic approach. The optimal decision rule we seek is to minimise the Bayes risk by trading between a deliberative maneuver and alternatives. Real-time implementation is emphasised here to provide a framework for real world applications.

## 4.1 Introduction

The problem of obstacle avoidance is one of the key issues to be addressed to achieve successful applications of mobile robots in a real world. Research on this problem can be classified as one of two approaches: *model-based* and *sensor-based.* In general, the model-based approach considers obstacle avoidance in a global way. It uses built-in models to describe known obstacles completely in order to generate a collision-free path. It is normally performed at the global path planning level, which we described in the last chapter. In contrast, the sensor-based approach aims to detect and avoid unknown obstacles in real-time. It is formulated as a local path planning problem in the sense that the local detours around sensed obstacles are generated while the robot follows a globally planned path. This chapter addresses this issue. Many methods are reported to achieve real time obstacle avoidance for mobile robots and manipulators, of which the artificial potential field is the most popular approach. This method includes the *Force Field* [Brooks, 1986], [Arkin, 1989], the

*Virtual Force Field* and *Vector Field Histogram* [Borenstein and Koren, 1989], [Borenstein and Koren, 1990b], and the *Goal Relocation Approach* [Adams *et al.*, 1990].

In the artificial potential field approach, the obstacles to be avoided exert repulsive forces $\mathbf{F}_o = \sum_i^N \mathbf{F}_{oi}$ ($N$ is number of obstacles) onto the mobile robot, while the goal applies an attractive force $\mathbf{F}_g$. The sum of all forces, the resultant force $\mathbf{F}_{att} = \mathbf{F}_g + \mathbf{F}_o$, determines the subsequent direction and speed of travel so that the mobile robot reaches the goal without colliding with obstacles. It can be shown that if the potential of the regions within obstacles is set to infinity, collision will never occur. The method gets its name from the fact that the force $\mathbf{F}$ is usually derived from a scalar potential field $U$, i.e. $\mathbf{F} = -\nabla U$. Khatib [1986] first used the potential fields for real-time obstacle avoidance by robot arms (or mobile robots). But this approach requires *a priori* and complete information about the location of the obstacles in the environment. Brooks [1986] adopts the force field method in an experimental robot equipped with a ring of twelve sonars and treats each sonar reading as a repulsive vector $\mathbf{F}_{sonar}$. If the magnitude of the sum of the repulsive forces, $\mathbf{F}_o(x) = \sum_{i=1}^{12} \mathbf{F}^i_{sonar}(x)$, exceeds a certain threshold, the robot stops, turns into the direction of the resultant force vector, and moves on. The *Virtual force field* proposed by Borenstein and Koren [1989] allows continuous, smooth, and relatively fast motion of the controlled vehicle among unexpected obstacles(up to 0.78m/s). It is a combination of a two-dimensional Cartesian certainty grid, named the histogram grid, to represent obstacles, and a potential field method for local path planning.

The potential field method is an attractive approach to the collision avoidance problem because of its simplicity and efficient implementation. However, the complexity of tasks that can be implemented using this approach is limited. In a cluttered environment, the robot may become trapped because of the occurrence of local minima in the resultant potential field. This occurs when the attractive forces of the goal are balanced by the repulsive forces due to the presence of obstacles, a problem particularly likely to occur in environments that include concave obstacles.

Other problems include path oscillations when travelling through a narrow passage or corridor, and the inability to pass easily among densely spaced obstacles [Koren and Borenstein, 1991]. To overcome these problems, Borenstein and Koren [1990b] proposed another obstacle avoidance method called the *Vector field histogram*, which consists of a two-dimensional Cartesian *Histogram grid* (at world-model level), updated continuously with range data, and a one-dimensional *Polar histogram* (at intermediate data-level) that represents the polar obstacle density around the robot. Then a sector with low obstacle density which is close to the goal direction is selected as the robot's direction of motion. Zhao and BeMent [1990] developed a heuristic search-based algorithm for trap recovery. In their approach, a local obstacle avoidance algorithm, used in conjunction with the high level grid-based map, allows the robot to avoid collisions. Whenever a potential trap is identified, the trap recovery algorithm generates intermediate points a short distance away that are used as temporary goals to guide the robot out of the trap situation, based on the low level node-based map.

However, most of the potential field methods mentioned above are in fact difficult to implement directly on car-like mobile robots whose kinematic constraints are nonholonomic. The main reason for this is that these mobile robots have a lower bound on the turning radius and are unable to follow an arbitrary path or change direction instantaneously. For instance, if the direction of the resultant force $F_{att}$ applied to a car-like mobile robot is perpendicular to the current orientation, the robot has either to turn sharply or to stop to pivot. This is obviously not desired or allowed in practical applications. Moreover, limitations on robot motion within roadways imposes additional problems which are not taken into account by potential field methods.

We propose a new sensor-based obstacle avoidance strategy which aims to achieve both collision free motion and minimum costs in the next section. In section 3, we describe how the probabilistic sensor models are built to reduce the uncertainty in sonar data. The formulation of loss function and the recursive Bayes decision rule are given in section 4. The simulation and experimental results are presented in section 5. Finally, a brief summary is given in section 6.

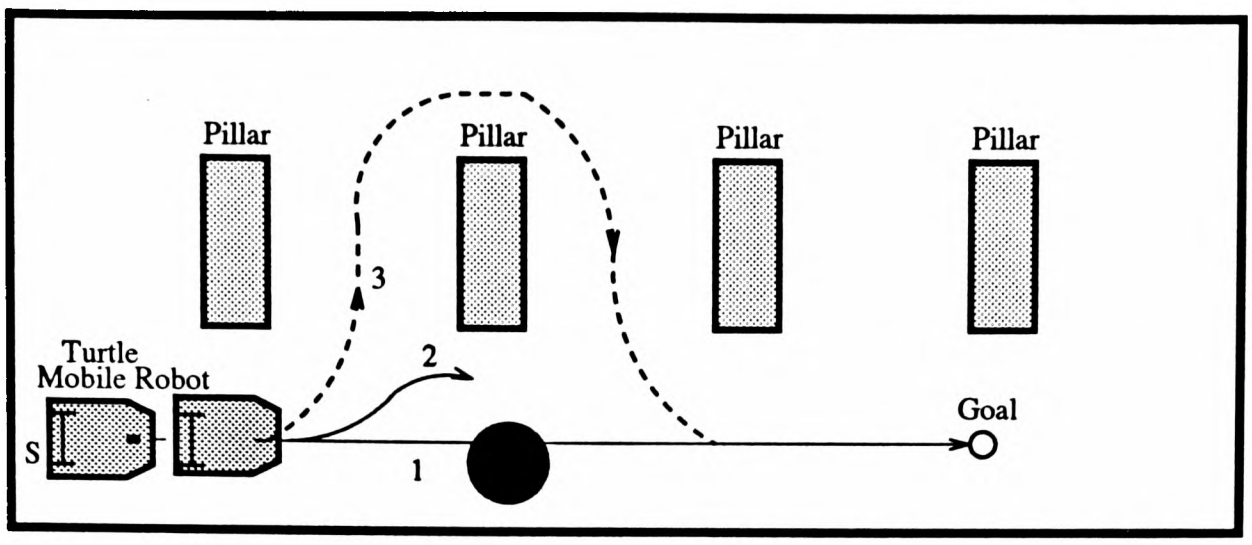



Figure 4.1: *Turtle* encounters an unexpected obstacle

## 4.2 Problem Formulation

Let us suppose that a mobile robot is commanded to travel along an optimal path generated by the global path planning algorithm, shown as path 1 in Figure 4.1. However, an unexpected obstacle appears along its path and there arises the possibility of a collision. Since the sensors have limitation on their range and accuracy, they may not be able to tell exactly whether the gap between the obstacle and the pillar is wide enough for a sidestep maneuver when it is some distance away. Therefore, a decision problem arises as to whether the robot should continue its motion along path 2 to make further observations to maneuver (at a certain cost) or instead to follow along path 3 and incur a certain loss. If the cost for doing extra sensing is less than the cost for taking the alternative path, it may be worth persevering along path 2. But the robot may eventually find the gap is impassable, incurring an overall cost greater than following path 3.

To solve the problem described above, we face two basic difficulties:

- How to deal with uncertainty in collision detection.
- How to choose an optimal action based on imperfect state information.

This leads us to pursue a decision theoretic approach to build a probabilistic model for the noisy sensors and to seek a decision rule for an *avoiding* control task in the face of uncertainty. All decision theoretic methods are essentially probabilistic in nature and require a common structure

such as the state of nature, prior information, likelihood functions and utility or loss [Ferguson, 1967] [Berger, 1985], to be incorporated into the decision-making process, which we introduce in the next section.

### 4.2.1 Decision Theoretic Approach

Decision theory is a useful formalism for making decisions in the face of uncertainty. In general, the following terminology is used in statistical decision theory:

- the state of nature (or environment), $\theta \in \Theta$.
- prior probability distribution on $\theta$, $p(\theta) \in \Lambda$.
- the decision to be made (or action), $a \in \mathcal{A}$.
- the lost (cost) function on $\theta$ and $a$, $L(\theta, a) \in \Theta \times \mathcal{A}$.
- the noisy measurement (data), described by $f(z|\theta)$, $z \in \mathcal{Z}$.
- a decision rule mapping $\mathcal{Z}$ into $\mathcal{A}$, $d(z) = a$, $d \in \mathcal{D}$.

Note that $\Theta$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{Z}$ is the sample space, and $\mathcal{D}$ is the decision space.

The fundamental problem of decision theory can be stated quite simply. Given $(\Theta, \mathcal{D}, L)$ and the noise measurements $z \in \mathcal{Z}$ whose distribution $f(z|\theta)$ depends on $\theta \in \Theta$, what decision rule $d(z)$ should the decision maker choose to minimise the *risk function*, or the *expected loss*, defined by

$$\begin{aligned} R(\theta, d) &= \mathbf{E}_\theta[L(\theta, d(\mathcal{Z}))] \\ &= \int_{\mathcal{Z}} L(\theta, d(z)) f(z|\theta) dz, \text{(for the continuous case)} \qquad (4.1) \\ &= \sum_{\mathcal{Z}} L(\theta, d(z_i)) f(z_i|\theta), \text{(for the discrete case)} \qquad (4.2) \end{aligned}$$

where the notation $\mathbf{E}[\cdot]$ is used for the expectation operation which is taken with respect to the probability distribution of $\mathcal{Z}$, which depends on $\theta$.

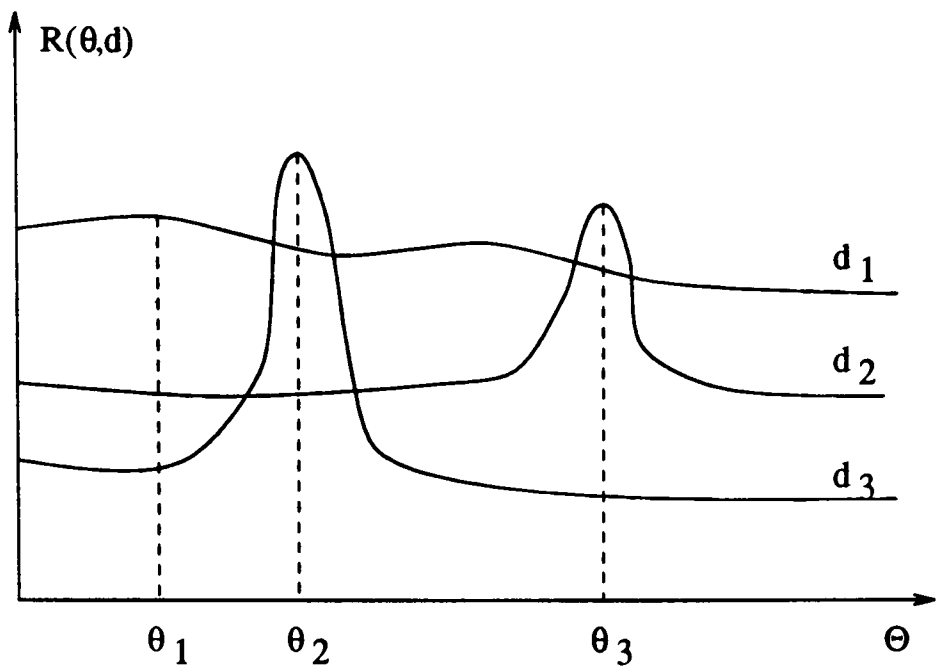



Figure 4.2: The risk function for three decision rules

The decision maker is concerned with taking the best action in the face of uncertainty. Thus the risk function $R$ serves as a criterion to compare different decision functions. It depends on both the true state of nature, $\theta$, and the decision function, $d$. Since the true state of nature $\theta$ is not exactly known during the decision-making process, the difficulty in a decision problem naturally arises.

It is unlikely that a decision rule has the smallest risk no matter what the true state of nature is. Instead, for each fixed state of nature there may be a best action for the decision maker to take. However, this best action will differ, in general, for different states of nature, so that no one action can be presumed best over all. For instance, there are three rules in Figure 4.2. Rule $d_3$ is superior(has lower risk) for the majority of states, and has worst-case risk at the state $\theta_2$ and its vicinity. In contrast, rule $d_1$ has greater risk for the majority of states, and has lower risk at the states $\theta_2$ and $\theta_3$, as well as their vicinity. As a matter of fact, none of them is a best rule in terms of all states of nature. The best action is really depends on what the true state of nature is and how accurate information on $\theta$ the sensors can provide. In the face of both uncertainties, decision theory is to find a method to determine a better rule. There are two basic methods that have been used to choose an optimal action, namely (i) the *Minimax* rule; (ii) the *Bayes* rule.

**The minimax approach**

The minimax approach is to find a decision rule that will minimise the worst-case risk. It is mainly used in the field of game theory to outwit an intelligent adversary. It proceeds as follows. First, for a given decision function $d$, consider the worst that the risk $R$ could be:

$$\sup_{\theta \in \Theta}[R(\theta, d)] \tag{4.3}$$

Then choose a decision function, $\hat{d}$, that minimises this maximum risk:

$$\hat{d} = \inf_{d \in \mathcal{D}} \{\sup_{\theta \in \Theta}[R(\theta, d)]\} \tag{4.4}$$

where sup and inf are least up bound and greatest low bound respectively.

Such a decision rule, if exists, is called a *minimax rule*. It is independent of the probability distribution of the state of nature. In other words, it avoids making any assumption about the probability distribution. The weakness of the minimax approach is intuitively apparent. It is a very conservative procedure that places all its emphasis on guarding against the worst possible case. For instance, by applying the minimax rule, the decision maker would select rule $d_1$ since its worst-case risk $R(\theta_1, d_1)$ is lower than the worst-case risk $R(\theta_3, d_2)$ for rule $d_2$ and $R(\theta_2, d_3)$ for rule $d_3$, see Figure 4.2. In fact, $d_1$ is not a best choice at all for the majority of states.

**The Bayesian approach**

The Bayesian approach is to find a rule that minimises the *expected risk* using a prior distribution of the state of nature, $\pi(\theta)$. The expected risk, or *Bayes' risk*, is defined as

$$\begin{aligned} B(\pi(\theta), d) &= \mathbf{E}^{\pi(\theta)}[R(\Theta, d(\mathcal{Z}))] \\ &= \int_{\Theta}\int_{\mathcal{Z}} L(\theta, d) f(z|\theta)\pi(\theta) dz d\theta, \text{ (for the continuous case)} \qquad (4.5) \\ &= \sum_{\Theta}\sum_{\mathcal{Z}} L(\theta_i, d) f(z_j|\theta_i)\pi(\theta_i), \text{ (for the discrete case)} \qquad (4.6) \end{aligned}$$

Here the expectation is taken with respect to the probability distribution of both $\Theta$ and $\mathcal{Z}$. Note that Bayes's risk $B(\pi(\theta), d)$ is a scalar function of $d$. Then, the *Bayes rule* is a decision function

which minimises the Bayes risk, i.e.

$$\hat{d} = arg\min_{\mathcal{D}} B(\pi(\theta), d) \tag{4.7}$$

By applying the Bayes rule, the decision maker would choose rule $d_3$ if the prior probability distribution of the state, $\pi(\theta)$, is not concentrated around its peak, see Figure 4.2. On the other hand, rule $d_2$ would be a better choice if $\pi(\theta)$ is highly concentrated between $\theta_1$ and $\theta_2$.

By taking more measurements, $\mathbf{z} = [z_1, z_2, ..., z_k]^\mathsf{T}$, on $\theta$, the decision maker could update his knowledge about $\theta$ and obtain the conditional distribution of $\theta$, $\pi(\theta|\mathbf{z})$, using Bayes theorem. We define the posterior risk of an action, $a = d(\mathbf{z})$, as the *Bayesian expected loss* to be a real-valued function on $\mathcal{Z} \times \mathcal{A}$:

$$\begin{aligned} B(\pi(\theta|z), a) &= \mathbf{E}^{\pi(\theta|z)}[L(\theta, a)] \\ &= \int_{\Theta} L(\theta, a)\pi(\theta|\mathbf{z})d\theta, \text{ (if } \theta \text{ is continuous)} \qquad (4.8) \\ &= \sum_{\Theta} L(\theta_i, a)\pi(\theta_i|\mathbf{z}), \text{ (if } \theta \text{ is discrete)} \qquad (4.9) \end{aligned}$$

Thus the Bayes rule is an action that minimises the expected loss $B(\pi(\theta|\mathbf{z}), a)$, i.e.

$$\hat{a} = arg\min_{\mathcal{A}} B(\pi(\theta|\mathbf{z}), a) \tag{4.10}$$

Note that for a given decision rule $d$, Equation 4.6 defines the risk over the joint distribution of measurements and states, $f(\mathbf{z}|\theta_i)\pi(\theta_i)$, which could be quite time consuming for a large number of possible values of $\mathbf{z}$. In contrast, Equation 4.9 defines the risk over the conditional distribution of states, $\pi(\theta_i|\mathbf{z})$, on the measurement $z$ which is currently made available. As a result, Equation 4.9 can be iterated recursively to achieve real-time performance. It is worth to notice that two formulations are equivalent except for the execution time.

There are different opinions on the Bayesian approach. Some made criticism on the grounds that the models of the prior distribution are unrealistic. Others questioned how prior distributions can be determined in practice. Another response, however, was to show that Bayesian strategies are robust

to the choice of prior distribution [Berger, 1985]. Bayesian decision theory has been successfully applied to robot sensing, computer vision, and control. For instance, Durrant-Whyte [1987] used it to develop a unified approach to combining and transforming geometric models. Hager [1988] applied both Minimax and Bayesian decision theory to a general class of sensor fusion problem. Dickson [1991] used Bayesian approach to build probabilistic models to image segmentation and grouping problem.

### 4.2.2 Application to Obstacle Avoidance

In order for Bayesian decision theory to be applied to the obstacle avoidance problem, we first address how to cast the avoidance problem into the decision-making framework in this section. Then an optimal decision rule to avoid unexpected obstacles is calculated by minimising the expected loss in trading between a deliberative maneuver and the alternative path in the fellowing sections.

We consider the appearance of the unexpected static obstacles to be a random event. Let $\Theta$ denote the state of the path to be a random variable consisting of two states:

$$\Theta = (\theta_1, \theta_2) = (passable, impassable) \tag{4.11}$$

The sensors take $k$ measurements that comprise an observation vector $\mathbf{z}$:

$$\mathbf{z} = [z_1, z_2, z_3, ..., z_k]^{\mathsf{T}} \tag{4.12}$$

The measurement $z_k$ is the true state of path $\theta_k$ corrupted by random noise $w$

$$z_k = \theta_k + w \tag{4.13}$$

The observation $\mathbf{z}$ gives some indication of the value of the state $\theta$, and enables us to reduce uncertainty in the decision. The probabilistic information in $\mathbf{z}$ about $\theta$ is described by a conditional probability density function $p(\mathbf{z}|\theta)$ of the observation vector $\mathbf{z}$, namely the *likelihood function* $l_k(\theta)$:

$$l_k(\theta) = p(\mathbf{z}|\theta) \tag{4.14}$$

$l_k(\theta)$ contains all the relevant information from the observation $\mathbf{z}$ required to make inferences about the true state $\theta$ to make a decision [Berger, 1985]. In general, the likelihood function is obtained from either empirical or analytic models.

Given that the mobile robot should stay on the path in any case, strategies to avoid unexpected static obstacles include: (i) *maneuver*, if there is a large enough gap, (ii) *shyaway*, when an obstacle is too close, (iii) *backtrack*, if necessary, and (iv) *stop*, in an emergency. Note that both *shyaway* and *stop* are for emergency use only. Therefore, we define the action space by:

$$\mathcal{A} = (a_1, a_2) = (maneuver, backtrack) \tag{4.15}$$

The robot chooses an action $a$ from a set $\mathcal{A}$ of all possible actions based on the observation $\mathbf{z}$ of the current state of the path $\theta$. This is a decision process mapping the sample space onto the action space $\mathcal{A}$, denoted by

$$a = d(\mathbf{z}) \tag{4.16}$$

In general, the decision made by the robot is concerned with the best possible action in the face of uncertain information $\mathbf{z}$. Therefore, a way of evaluating a decision or action is required. We define a *lossfunction* $L(\theta, a)$ which gives a measure of the loss incurred in taking the action $a$ when the state is $\theta$. It is often chosen subjectively in order to reflect the merit of the chosen action for the given $\theta$. For example, when the state $\theta$ is impassable and the robot chooses backtrack action $a_2$, then the incurred loss $L(\theta_2, a_2)$ can be defined as

$$L(\theta_2, a_2) = 0 \tag{4.17}$$

The most desirable action for the robot is that which minimises $L(\theta, a)$ when the state of nature is $\theta$. We assume throughout this chapter that the loss function is bounded below; then, without loss of generality, we assume that the loss function we use is nonnegative

$$L(\theta, a) \geq 0 \tag{4.18}$$

The way the action is chosen by the robot is called the decision rule. A Bayes decision rule is pursued here to minimise the expected loss over the possible actions. It attempts to make maximum use of any prior information about the state $\theta$, termed $p(\theta)$ [Cameron and Durrant-Whyte, 1990]. Although such information is often regarded as subjective and is seldom precise, it contains any knowledge that is available. Given such a prior distribution $p(\theta)$, we can calculate the *Bayes risk* for each action $a$ being chosen

$$B(p(\theta), a) = E^{p(\theta)}[L(\theta, a)] = \sum_{\Theta} L(\theta, a)p(\theta) \tag{4.19}$$

The Bayes risk is the average of the expected loss (also called the *risk function*) with respect to the prior distribution of $\theta$ for each action $a$. Note that it depends on both the true state of nature, $\theta$, and the action, $a$. The Bayes decision rule is the selection of an action $a$ that minimises the Bayes risk. More precisely, the Bayes rule chosen by the robot is the action $\hat{a}$ which minimises the Bayes risk

$$\hat{a} = arg\min_{A} B(p(\theta), a) \tag{4.20}$$

Until now, we have cast the problem of the choice of optimal collision avoidance within the general framework of Bayesian decision theory [Ferguson, 1967], [Rice, 1988]. In the rest of this chapter, we describe further how we actually implement the approach, focusing on the following aspects:

- modelling sensor uncertainty.
- formulation of the loss function for a deliberative maneuver and backtrack.
- recursive implementation of Bayes decision rule.

## 4.3 Obstacle Detection

To avoid collisions, a robot should be able to detect obstacles. Range sensors are usually used to measure the distance between the robot and obstacles. The most popular range sensors include

ultrasonic and laser. In addition, one or more CCD cameras can be used to detect distance from an image sequence, though this demands large computational power. Tactile sensors can be used to "feel" obstacles to stop in emergency. Among these sensors, the ultrasonic sensor has been widely used since it is inexpensive, easy to use, and immune to lighting interference. In this section, we analyse some sonar sensor models used by other researchers and then introduce our methodology using an array of 12 sonar sensors for real-time obstacle detection.

### 4.3.1 Sensor Model and Data Interpretation

The standard Polaroid sonar ranging system [Corporation, 1984] employs a single acoustic transducer that acts as both a transmitter and receiver. It transmits 56 cycles of a 49.4 kHz square wave in a 1.13ms pulse duration. After the transmitted pulse encounters an object, an echo may be detected by the same transducer acting as a receiver. The first echo whose amplitude exceeds the preset threshold level determines the time-of-flight, denoted by $t_0$. Then a range measurement $r_0$ is obtained by the following equation:

$$r_0 = ct_0/2 \tag{4.21}$$

where $c$ is the speed of sound in air, equal to 343m/s at room temperature(20 degrees). This limits the speed at which range readings can be obtained. It would take, for example, about 60ms for the echo to be return from an object at a range of 10 meters.

The time-of-flight measurement of the Polaroid sonar sensor is able to provide comparatively accurate range information from $r_{0\min} = 0.27m$ to $r_{0\max} = 10m$. However, the angular resolution $\alpha_s$ is rather poor due to the wide beam-width. We denote a sonar reading as a vector $\mathbf{r}_s = (r_s, \alpha_s)$. The azimuth $\alpha_s$ is typically an ambiguous value in $[-\vartheta_s, \vartheta_s]$, where $\vartheta_s$ is the half beam-width of the transducer, which can be calculated from [Kuc and Viard, 1991]

$$\vartheta_s = \sin^{-1}(0.61\lambda/a) = \sin^{-1}(0.61c/af) \tag{4.22}$$

where $\lambda = c/f$ is the wavelength of the acoustic signal. $f$ is the resonant frequency of the transducer

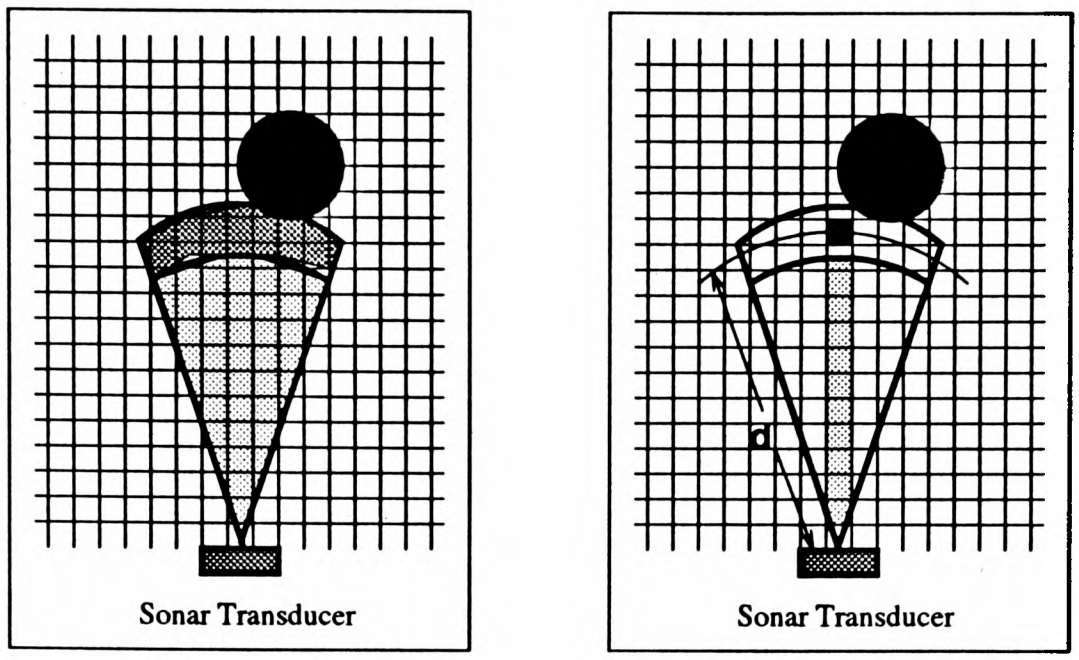
and $c$ is the speed of sound in air. $a$ is the radius of the transducer. For the Polaroid transducer we use, $f = 49.4kHz$, $a \simeq 20mm$, $c = 343m/s$, and $\vartheta_s \simeq 12°$. This figure, however, may need to be found exactly by calibration for each sonar sensor being used in applications.

The direct measurement of range by the sonar sensor makes it extremely easy to use. However, it has some inherent limitations, including: (1) poor directionality caused by wide beam-width; (2) specular reflections away from the sensor; (3) frequent misreadings from multiple reflections (4) crosstalk with neighbouring sonar sensors. These disadvantages may be overcome by integrating it with other sensors, such as infrared sensors [Flynn, 1988]. On the other hand, it is possible to build a suitable sonar sensor model and experimental model to give accurate interpreting of the sonar readings. A number of researchers have explored this direction, which we briefly review here.

Grid Model

The certainty grid [Moravec and Elfes, 1985] [Elfes, 1987] and the occupancy grid [Elfes, 1990] are probabilistic approaches to interpret sonar readings. They explicitly represent free, occupied and unknown regions in the robot's two-dimensional world space, digitised as small square grid cells. They operate in two stages. First, a sensor range measurement $r$ is interpreted using a probabilistic sensor model which is defined by a probability density function of the form $p(r|s(c))$, where $r$ is the range reading and $s(c)$ is the state of a cell $c$. Then, a large number of grid cells corresponding to the projection of the sonar beam (two-dimensional) are updated using a Bayesian estimation model, as shown in Figure 4.3(a). The light shaded cells inside the cone of the beam are updated as *empty*, and the heavily shaded cells are updated as *occupied*. This updating procedure is computationally intensive.

Borenstein and Koren [1990a] developed a different grid-based representation called the *Histogram grid* in which each cell holds a certainty value that represents the confidence of the algorithm in the existence of an obstacle at that location. Instead of using a two-dimensional sensor model like the occupancy grid, they use a one-dimensional sensor model. In other words, for each sensor



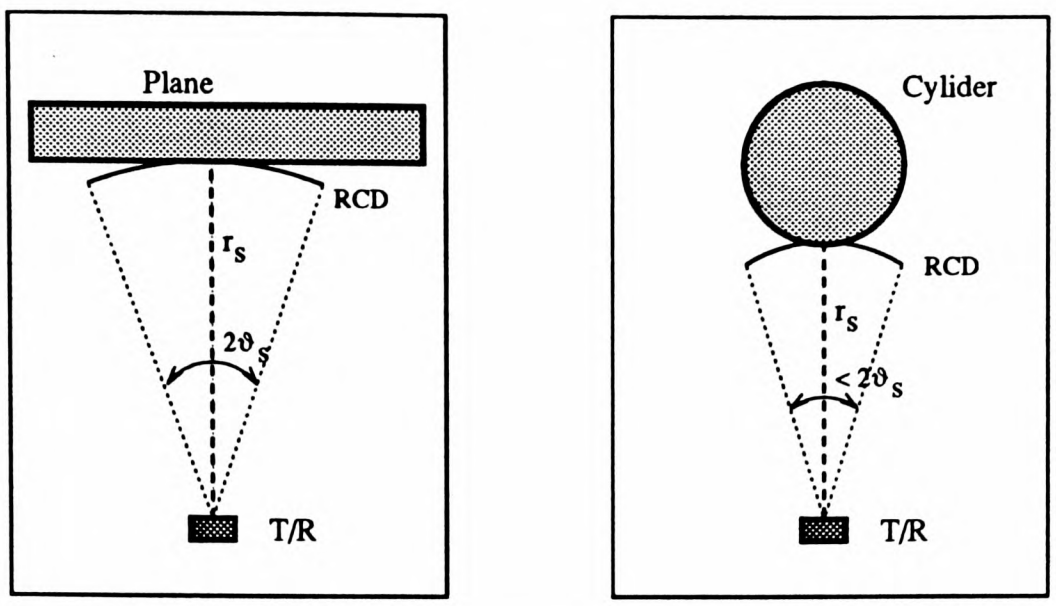

(a) Occupancy Grid (b) Histogram Grid

Figure 4.3: The Occupancy Grid and the Histogram Grid

reading, only the cells lying on the acoustic axis of the sonar beam are updated, as shown in Figure 4.3(b). This trivial sensor model is based on the assumption that an object located near the acoustic axis (center of the cone) is more likely to produce an echo than an object further away from the acoustic axis. Note that the heavily shaded cell corresponding to the measured range *d* has increased certainty value, and the light shaded cells are decremented. This updating procedure is rapid and performed when the vehicle is moving, creating a histogram pseudo-probability distribution, in which high certainty values in cells represent the expected location of the obstacles. It achieves surprisingly good results [Borenstein and Koren, 1989], [Borenstein and Koren, 1990b]. To solve the crosstalk problem caused by fast firing of multiple sonars, a method called *Error eliminating rapid ultrasonic firing* is introduced in [Borenstein and Koren, 1992].

**RCD Model**

The RCD (*Regions of Constant Depth*) model provides another feasible way to interpret sonar data for map-building tasks. It is based on the fact that all environment features appear as specular reflectors to an in-air sonar sensor resulting in range-bearing scans composed of regions in which the measured range is constant [Kuc and Viard, 1991] [Leonard and Durrant-Whyte, 1992]. Thousands




(a) A RCD for Plane Target (b) A RCD for Cylider Target

Figure 4.4: The RCD models for Plane and Cylinder

of data scans associated with each type of environmental feature, such as a wall, corner, cylinder and edge, are observed and analyzed to obtain different RCD models [Leonard and Durrant-Whyte, 1992].

The width of RCDs depends on the specific threshold of the sonar sensor, the smoothness of the feature surface, and the materials of the objects being detected. For a normal indoor environment, it has been found that the widths of the observed RCDs from a wall and corner are approximately equal to the total beam width $2\vartheta_s$, while the width of the RCD observed from an edge is smaller because of the smaller amplitude echo [Kuc and Viard, 1991]. In contrast, the RCD observed from a cylinder has a width that is proportional to the radius of the cylinder. The maximum value equal to $2\vartheta_s$. Figure 4.4 shows one RCD for a plane and another for cylinder, where the width of the RCD for a cylinder is smaller than the total beam width $2\vartheta_s$ because of the diffuse reflection of the acoustic wave [Barshan, 1991].

These RCD models can be used in practice to validate and match sampled sonar data for recognition of the corresponding environment features. Leonard and Durrant-Whyte call such features "General Beacons" and suggest how they can be used to localise the robot itself and for map-building in an unknown environment, based on Kalman filtering [Leonard and Durrant-Whyte,

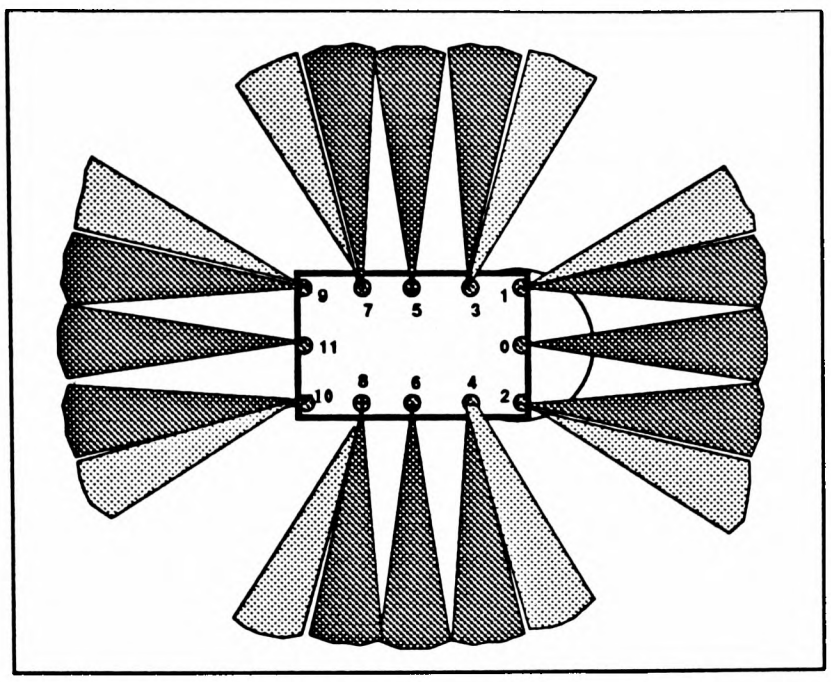



Figure 4.5: The configuration of a array of 12 sonars

1990]. However, using this method to process data and abstract RCD is time consuming, the robot has to operate in a "move-stop-scan" manner. Recently, the use of a pair of differential sonars, namely *tracking sonar* [Manyika *et al.*, 1991], provides a more attractive approach to interpret sonar readings in real-time, based on the RCD model and sensor management.

### 4.3.2 Sensor Configuration

Although both the grid model and the RCD model described above assume unknown environments, they can also be used in a known environment subject to dynamic changes. However, both of them emphasise how to interpret sonar readings for dynamic map-building and localisation. In our case, there is an *a priori* environment map and the robot knows where it is. What we need is a method to use sonar sensors to detect unexpected random obstacles whose positions are not recorded in any map. For the use of the global planner, described in the last chapter, we need to record how many and which kind of obstacles the robot has met during its traversal to update the uncertainty cost of each portion of path.

To achieve real-time obstacle avoidance, we form a "sonar bumper" consisting of an array of 12 sonars around the robot, shown in Figure 4.5. Each of them is driven by a Futaba servo motor

and can be rotated up to 170 degrees. The rotation speed is about 100 degrees per second. The light shaded cone depicts one sonar beam rotated about one beam width from the original direction shown as the heavily shaded cone. This provides greater flexibility than the more usual fixed sonar array used by people such as [Moravec and Elfes, 1985], [Chatila and Laumond, 1985], [Brooks, 1986], [Walter, 1987], [Crowley, 1989b], [Lang *et al.*, 1989], [Borenstein and Koren, 1990b].

The advantage of using a sonar array is the increase in sampling rate. However to avoid crosstalk (the sound wave from one transducer influencing others), the 12 sonars can not be simply operated simultaneously. Instead, we adopt a group sampling strategy. The 12 sonars are spatially separated by 30cm, and also decoupled in time in the sense that only one group of four sonars, one per side, is sampled at the same time. The other two groups are waiting or rotating. The maximum range of sonar sensors is set to be $R_{s\max} = 4.2m$. Any return range that is more than $R_{s\max}$ will be ignored so that the multiple reflection problem can be effectively reduced. To read all 12 sonar sensors in three groups takes about 120ms.

### 4.3.3 Local Map and Collision Zone

We introduce a local map to record the sensory information provided by the 12 sonar sensors, based on the robot local frame. The map is composed of a collision zone and a prediction zone, as shown in Figure 4.6. The collision zone is a two-dimensional circular area with radius $R_0$(proportional to the robot speed), located at the physical center of the vehicle. It is divided into four subzones: front, back, left, and right. When the sonar sensors see an unexpected obstacle in one of the subzones along which the robot is currently moving, a control action such as *stop* or *shyaway* will be chosen immediately to avoid collision. In contrast, the prediction zone is a rectangular space(measuring $4m \times 2.7m$) immediately ahead of the robot, in the direction which the robot is approaching. It is used mainly to predict potential collisions based on noisy sensor data, and to choose control actions such as $a_1$(maneuver) or $a_2$(backtrack).

Since sonar sensor readings typically cover a large space, it is necessary to provide a mapping

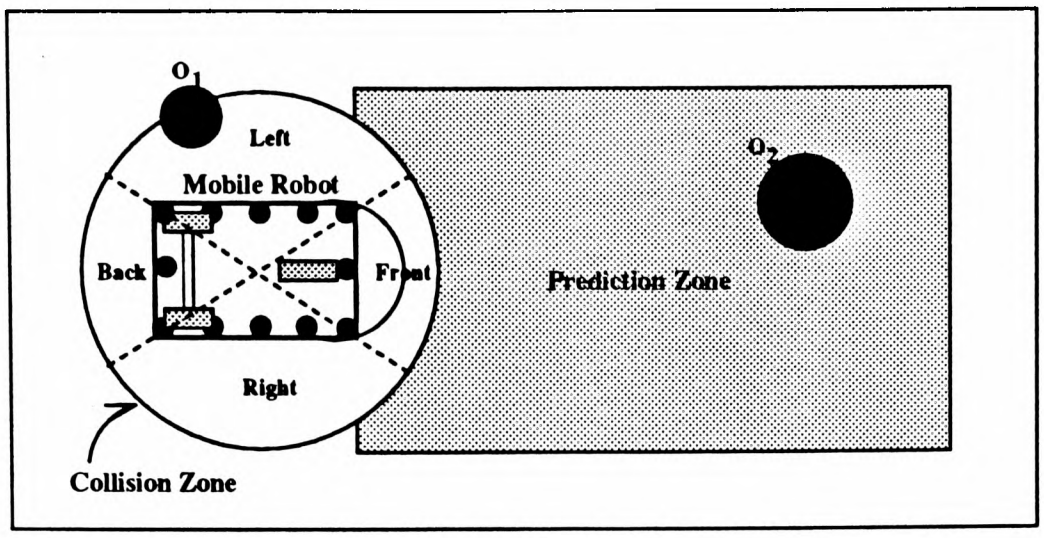



Figure 4.6: The local map consisting of collision and prediction zone

from the real sensor space to the abstract sensor space. In other words, the real sensor data is abstracted into: (i) short range, which falls into the collision zone and prediction zone and means that there is an obstacle close by, and (ii) long range, which means there probably is no obstacle. In this way, the robot is able to react to unexpected obstacles very quickly. The abstracted short range data is mapped into a local map and further processed for local obstacle avoidance in real time. For instance, the obstacle $O_1$ in Figure 4.6 is mapped into the left subzone, but no action is taken since the robot is moving forward. Instead, the robot reacts to the obstacle $O_2$ in the prediction zone and chooses a suitable action to avoid it.

### 4.3.4 Rule-based Interpretation of sonar data

Confronted by an unexpected obstacle in the prediction zone, the robot has to use its three front sonars to estimate the width of the gap, to judge whether it is passable. The observed measurements require interpretation to obtain reliable results. Since the wide beam width of sonar sensors make accurate measurements of the clearance of the path difficult, we adopt a simple heuristic rule for interpreting the three sonar readings. It is based on the superimposition of their beam wavefronts at some distance, as shown in Figure 4.7, where the robot is moving into a portion of path constrained by the road boundary.

From Figure 4.7, we can see that three beams superimpose at ranges greater than $0.7m$. It is

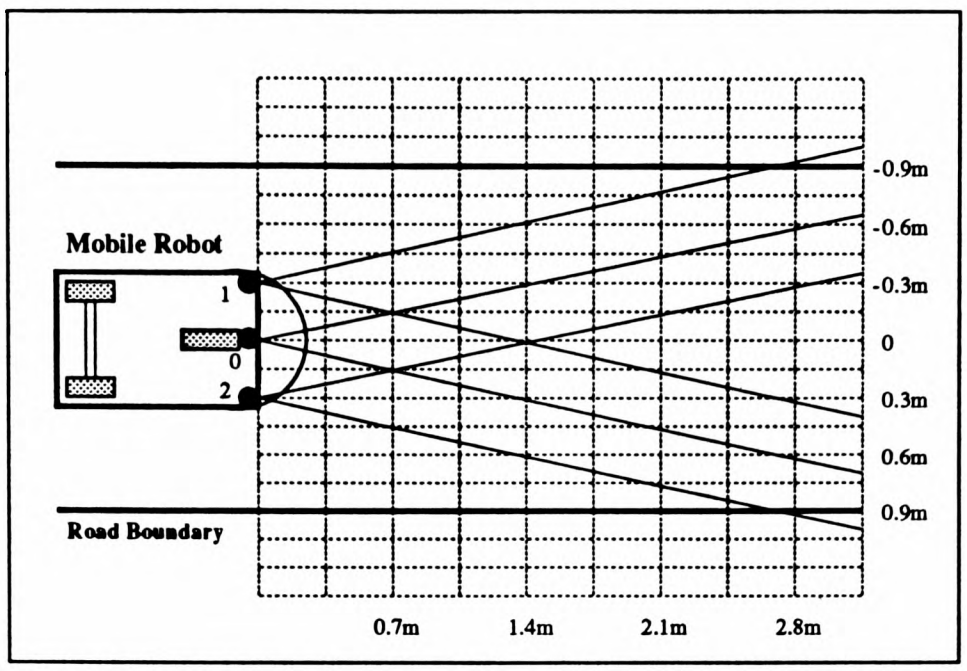



Figure 4.7: The superimposition of the wavefronts of three sonars at some distance.

possible to estimate the gap width with the resolution $0.3m$ within $2.1m$. By rotating three sonars, the resolution of measuring gap width can be increased. Here, we only consider the situation that three sonars are not rotated since this is the worst case to test our obstacle avoidance strategy proposed in the next section.

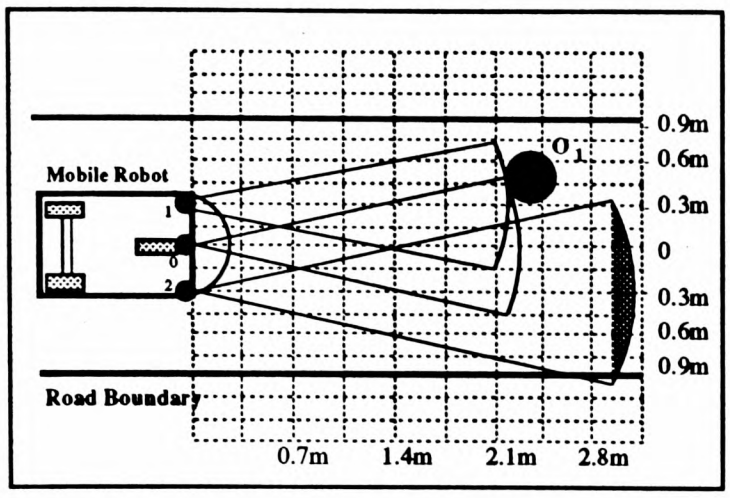
We choose the range $R_{dec} = 2.1m$ as the decision distance for the robot in terms of the robot's speed $0.5m/s$, the road width $1.8m$, and the resolution that can be provided by the three front sonars. The decision distance $R_{dec}$ increases up to $4m$ (the length of the prediction zone) as $W_r$ increase, and decreases as the road getting narrow. However, the robot's speed provides a low bound for the decrease of $R_{dec}$.

The heuristic interpreting rules can be simply described as:

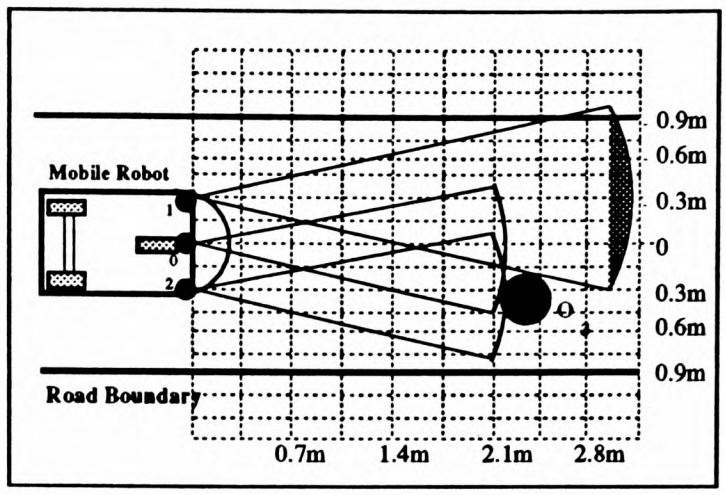
**Rule 1**: If all of three sonars see clearance, the predetermined path is passable and no action should be taken.

**Rule 2**: If the middle sonar sees nothing in the cone of its acoustic beam, then the predetermined path is passable even though the two side sonars may see obstacles, as shown Figures 4.8(d).

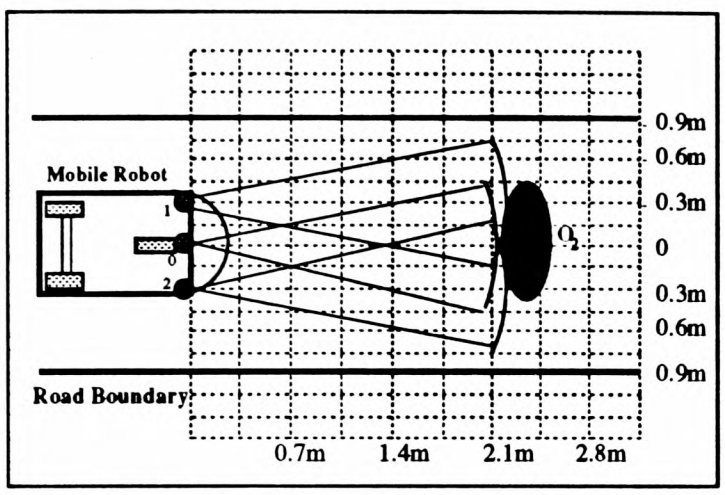
**Rule 3**: If any one of the side sonars sees clearance in the prediction zone and the



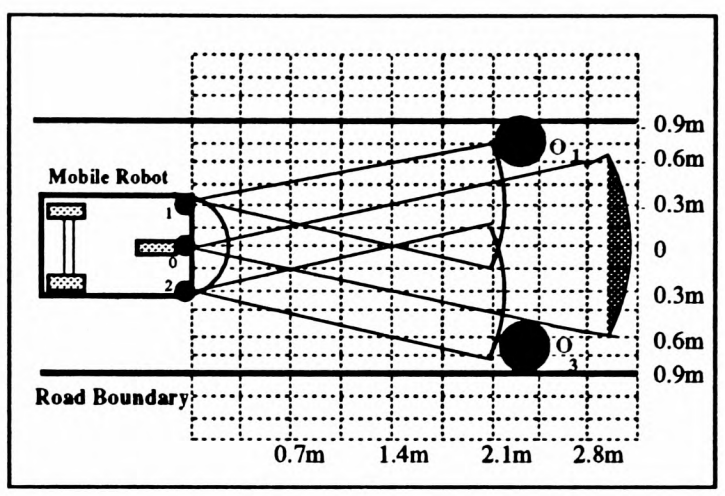

(a) An Obstacle Appears at the Left Side




(b) An Obstacle Appears at the Right Side




(c) An Obstacle Appears at the Middle




(d) The Obstacles Appear at the both Sides

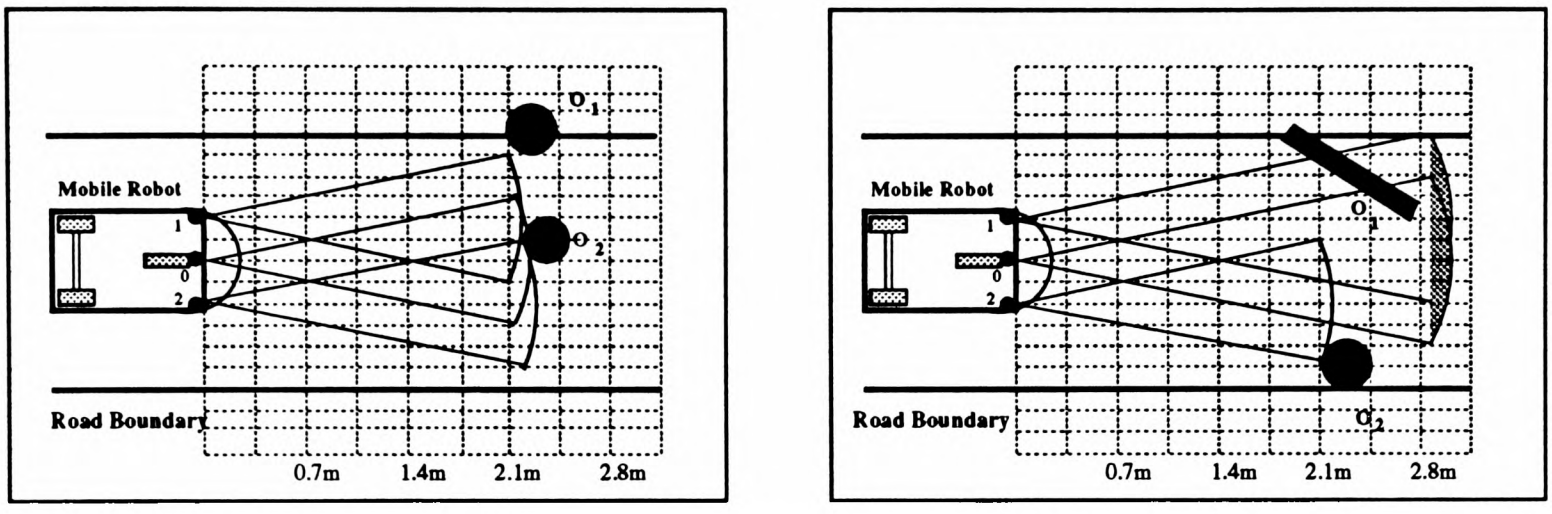
Figure 4.8: Interpretation of three front sonar data. Note that the light shaded circles meant that nothing is found in this part of prediction zone.

road width $W_r$ is at least twice the robot width, the gap is considered wide enough for a maneuver although the other two sonars may see an obstacle, as shown in Figures 4.8(a) and 4.8(b).

**Rule 4**: If all three sonars see something, then the path is announced to be a blockage, as shown in Figure 4.8(c).

This rule-based interpretation of the three sonar data is obviously very simple. However, there are two problems that we have to solve so that it can be used in practice. The first is that the feasible path may be eliminated by the broad beam width of the sonars. For instance, Figure 4.9(a) shows that the right side gap is actually passable even though three sonars see the obstacle $O_2$. We name

this situation as a 'miss' measurement since rule 4 concludes that the path is impassable. The second problem is caused by specular reflection. Figure 4.9(b) presents a situation in which the obstacle $O_1$ can not be detected by the sonars since its surface is not perpendicular to the acoustic axis of the sonar beam. As a result, an impassable path is interpreted as passable. We call this situation a 'false' measurement.




(a) Three Sonars see an Obstacle

(b) The Obstacle 1 is not Detected

Figure 4.9: Problems caused by the wide beam width and specular reflection

To solve the 'miss' and 'false' interpretations, it is necessary to use a probabilistic measure for the rules to represent knowledge about sensor errors. The probability measure is performed by 2000 random trials. The main obstacles we consider are furniture, boxes, and people. We assume that the size of the obstacles (as seen from the sensors), $O_s$, is a random variable with uniform distribution between 10cm and 60cm. The main features of the obstacles include the following: corners, edges, cylinders, planes, and coarse irregular surfaces. The appearance of these features is also assumed to be a random variable that is uniformly distributed. We introduce the following terminology:

- $N_t$ is the total number of trials. Each measurement is one.
- $n_1 \in N_t$ is the number of the times the state is $\theta_1$ (passable).
- $n_2 \in N_t$ is the number of the times the state is $\theta_2$ (impassable).
- $\mathbf{z}_{1c}$ is the correct measurement given that the state is $\theta_1$, and $n_{1c}$ is the number of times the event $\mathbf{z}_{1c} \in n_1$ occurs.

- $\mathbf{z}_{1m}$ is the miss measurement given that the state is $\theta_1$, and $n_{1m}$ is the number of times the event $\mathbf{z}_{1m} \in n_1$ occurs.
- $\mathbf{z}_{2c}$ is the correct measurement given that the state is $\theta_2$, and $n_{2c}$ is the number of times the event $\mathbf{z}_{2c} \in n_2$ occurs.
- $\mathbf{z}_{2f}$ is the false measurement given that the state is $\theta_2$, and $n_{2f}$ is the number of times the event $\mathbf{z}_{2f} \in n_2$ occurs.

From the above definitions, we have

$$N_t = \sum_{i=1}^{2} n_i = n_{1c} + n_{1m} + n_{2c} + n_{2f} \tag{4.23}$$

and the conditional probabilities of the four events $\mathbf{z}_{1c}$, $\mathbf{z}_{1m}$, $\mathbf{z}_{2c}$, and $\mathbf{z}_{2f}$ are defined as

$$p(\mathbf{z}_{1c}|\theta_1) = n_{1c}/n_1 \quad \text{(for the event } \mathbf{z}_{1c}) \tag{4.24}$$

$$p(\mathbf{z}_{1m}|\theta_1) = n_{1m}/n_1 \quad \text{(for the event } \mathbf{z}_{1m}) \tag{4.25}$$

$$p(\mathbf{z}_{2c}|\theta_2) = n_{2c}/n_2 \quad \text{(for the event } \mathbf{z}_{2c}) \tag{4.26}$$

$$p(\mathbf{z}_{2f}|\theta_2) = n_{2f}/n_2 \quad \text{(for the event } \mathbf{z}_{2f}) \tag{4.27}$$

It is obvious that the following conditions hold:

$$p(\mathbf{z}_{1c}|\theta_1) = 1 - p(\mathbf{z}_{1m}|\theta_1) \tag{4.28}$$

$$p(\mathbf{z}_{2c}|\theta_2) = 1 - p(\mathbf{z}_{2f}|\theta_2) \tag{4.29}$$

Table 4.1 presents the results from such a trial. The probability of miss measurements is reduced as the robot approaches the obstacle closely. In contrast, the probability of a false measurement increases when the obstacle is close to the robot. This is because there is no reflection from the surface itself as it is specular. However at a distance the sonars see a return from the edges, whereas at close range the edges are outside the beam width and the sonars see nothing. From this point of view, the sonar wide beam width is good for obstacle detection at some distance.

Table 4.1: Conditional Probability of Sonar Reading Given the State $\theta$

| Range (meter) | Correct $p(\mathbf{z}_{1c}\|\theta_1)$ | Miss $p(\mathbf{z}_{1m}\|\theta_1)$ | Correct $p(\mathbf{z}_{2c}\|\theta_2)$ | False $p(\mathbf{z}_{2f}\|\theta_2)$ |
|---|---|---|---|---|
| 1.40 | 0.82 | 0.18 | 0.72 | 0.28 |
| 1.75 | 0.73 | 0.27 | 0.75 | 0.25 |
| 2.10 | 0.65 | 0.35 | 0.79 | 0.21 |
| 2.45 | 0.57 | 0.43 | 0.83 | 0.17 |
| 2.80 | 0.49 | 0.51 | 0.86 | 0.14 |
| 3.15 | 0.41 | 0.59 | 0.89 | 0.11 |
| 3.50 | 0.33 | 0.67 | 0.91 | 0.08 |

## 4.4 Obstacle Avoidance Algorithm

In the last section, we presented our strategy to deal with uncertainty in obstacle detection using sonar sensors. We are now in a position to consider how the robot chooses an optimal action to avoid collision based on inaccurate sensory information. First the loss function is defined to set up a criterion for decision making. Then we explain the actual implementation of the decision theoretic approach in real-time obstacle avoidance.

### 4.4.1 Formulation of Loss Function

As already described in Section 4.2, a loss function $L(\theta, a)$ is a real value which is assigned as the outcome of action $a$ when the state is $\theta$. In our case, both the state space $\Theta$ and the action space $\mathcal{A}$ have two components. Therefore, we have four loss functions to be defined: $L(\theta_1, a_1)$, $L(\theta_1, a_2)$, $L(\theta_2, a_1)$, and $L(\theta_2, a_2)$.

$L(\theta_1, a_1)$ is the loss incurred by taking the action $a_1$ (maneuver) given that the path is passable, $\theta = \theta_1$. We simply define it to be zero since the robot lost nothing. Similarly, we have $L(\theta_2, a_2) = 0$, which means that the robot does not suffer any loss when it chooses to backtrack given that the state is impassable, $\theta = \theta_2$. (Recall that the robot does not know the true state $\theta$ when it makes its decision. Instead, the decision made by the robot is based on a probabilistic interpretation of the

sonar data.)

Both $L(\theta_1, a_2)$ and $L(\theta_2, a_1)$ are not equal to zero. $L(\theta_1, a_2)$ is defined in terms of the loss incurred by *backtrack* when the state of the path is in fact passable. It can be calculated by

$$L(\theta_1, a_2) = C_a - C_c \tag{4.30}$$

where $C_a$ is the cost of travelling along the alternative path from the current position to the goal, and $C_c$ is the cost of travelling along the predefined path. Both of them are sent from the global planner dynamically.

The loss $L(\theta_2, a_1)$ is incurred by the *maneuver* action when the true state of the path is impassable, and is defined by

$$L(\theta_2, a_1) = C_o + C_m \tag{4.31}$$

where $C_o$ is the cost needed to take the extra observation, and $C_m$ is the cost incurred by travelling from the decision position to the position where the extra sensing is taken.

The loss function is therefore represented in the following *loss matrix*:

Table 4.2: Definition of Loss Function

| Action | $\theta_1$ | $\theta_2$ |
|---|---|---|
| $a_1$ | 0 | $C_o + C_m$ |
| $a_2$ | $C_a - C_c$ | 0 |

### 4.4.2 Recursive Bayes Decision Rule

In Section 4.2, we formulated the problem of obstacle avoidance within the framework of decision theory. We seek the Bayes decision rule to minimise the expected loss incurred by taking action $a$ based on both the sensory information $\mathbf{z}$ and prior information, $p(\theta)$, about the state $\theta$.

When the robot travels around in a factory to transport parts and components, its sensors continuously sample its surroundings. The sample, $\mathbf{z}$, provides new information about the state of

the path, $\theta$. The new sample information, $\mathbf{z}$, can be combined with the prior information, $p(\theta)$, about the state $\theta$ by Bayes' theorem. This gives the posterior distribution of $\theta$ to be

$$p(\theta|\mathbf{z}) = \frac{p(\mathbf{z}|\theta)p(\theta)}{\sum p(\mathbf{z}|\theta)p(\theta)} \tag{4.32}$$

Given such a posterior distribution, we can compute the posterior expected loss of an action $a$ as follows:

$$B(p(\theta|\mathbf{z}), a) = E^{p(\theta|\mathbf{z})}[L(\Theta, a)] = \sum_{\Theta} L(\theta, a)p(\theta|\mathbf{z}) \tag{4.33}$$

Note that the above expectation is taken with respect to the posterior distribution of the state $\theta$. The Bayes decision rule chosen by the robot now is the action $\hat{a}$ that minimises the posterior expected loss

$$\hat{a} = arg\min_{A} B(p(\theta|\mathbf{z}), a) \tag{4.34}$$

This makes possible a recursive decision-making algorithm for finding the Bayes rule when sensor data is available. We can describe it as follows:

**Algorithm 1** *Obstacle Avoidance*

**Step 1**: *Wait for the current observation* $\mathbf{z}$ *to be provided by the heuristic rules for interpreting sonar data.*

**Step 2**: *Calculate the posterior distribution,* $p(\theta|\mathbf{z})$, *of the state.*

**Step 3**: *For each action* $a$, *calculate the Bayes risk* $\sum_{\Theta} L(\theta, a)p(\theta|\mathbf{z})$.

**Step 4**: *Find the Bayes rule: that is the action* $\hat{a}$ *with a minimum Bayes risk.*

**Step 5**: *Return to* **Step 1** *and continue until the robot reaches the goal.*

Figure 4.10 is a flow chart of the obstacle avoidance algorithm. We can clearly see how the decision rule finding algorithm is incorporated. As soon as the three front sonars see an unexpected obstacle in the prediction zone, the Bayes rule is selected to minimise the expected loss so that the optimal control action can be taken to avoid collisions. The implementation results are sent back to the global path planner to update the uncertainty cost which we have defined in the last chapter.

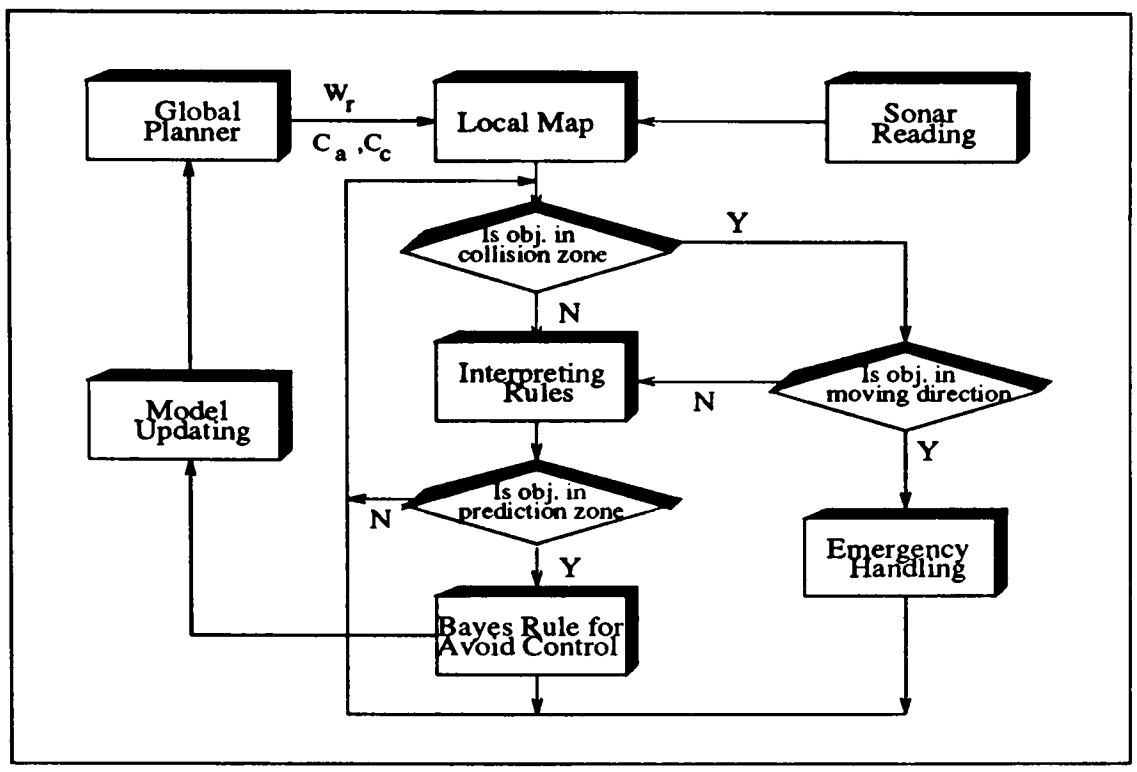



Figure 4.10: Flow chart of implementation of obstacle avoidance algorithm. Note that $W_r$ is the road width of the current portion of path. $C_a$ and $C_c$ are the costs of travelling along the alternative path and the predefined one respectively.

## 4.5 Experimental Results

The proposed algorithm has been implemented both by simulation on a SUN workstation and in reality on the *Turtle* mobile robot in our AGV Laboratory. For convenience to following description, Figure 4.11 redraws the model of the AGV Laboratory, the partition of its free space, and the central line of the defined roadway. Here, we give three examples.

### 4.5.1 Example 1

The mobile robot is commanded to move from the start position, *node 0*, to the goal, *node 8*. The optimal path planned by the global path planner is a straight line from *node 0*, via the intermediate nodes: *node J*, *node 2*, *node K*, *node 4*, *node L*, *node 6*, and *node M*, to *node 8*, which has a cost $C_c = 20s$ in terms of the path length $12m$ and the robot's speed $0.6m/s$.

When the robot moves along this optimal path, an unexpected obstacle appears on the way, see Figure 4.12. The heuristic rule 3 concludes that the state of the path is passable, i.e. $\theta = \theta_1$, since

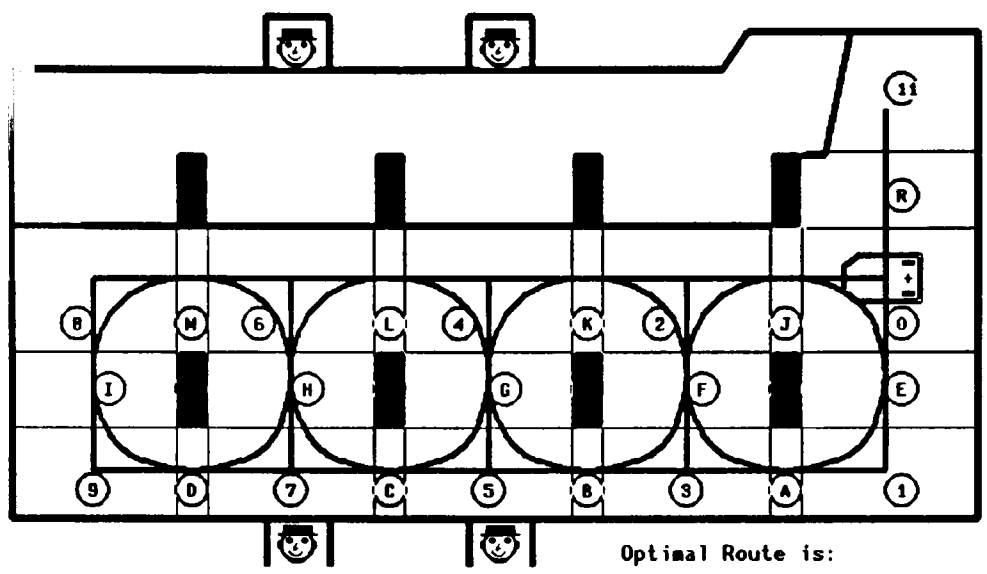



Figure 4.11: The model and partition of Oxford AGV Laboratory

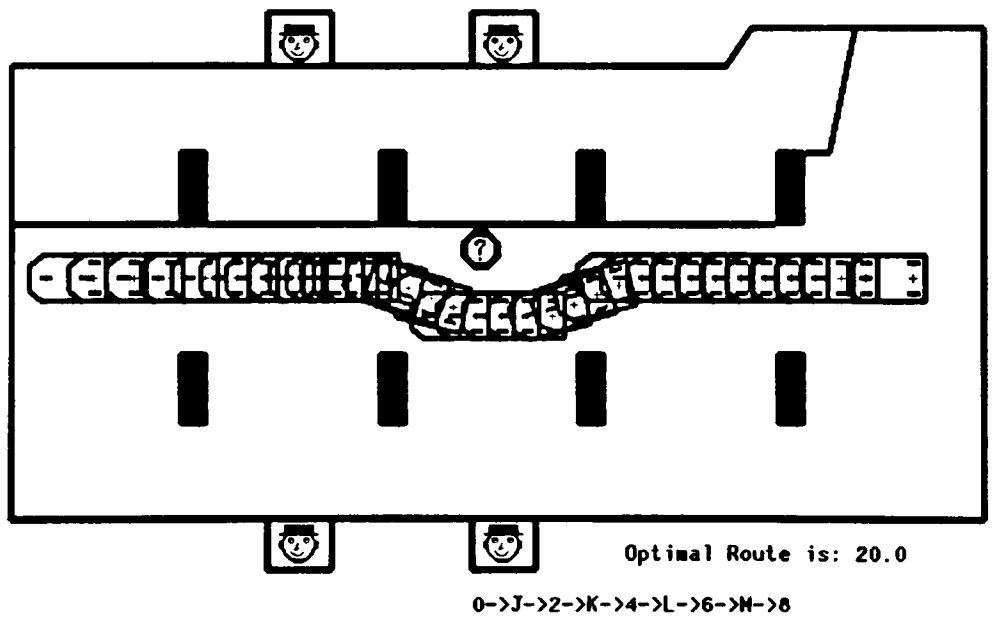



Figure 4.12: An action *maneuver* is chosen to minimise the expected loss

the left sonar(sonar 1) sees clearance. A new subgoal located at the middle of the gap near the obstacle is given. Which action should the robot take: *maneuver* or *backtrack*? We show how the robot finds the Bayes rule to give the answer, based on the proposed strategy.

Recall that the interpretation is made at a range $2.1m$ of sonar readings. From Table 4.1, the confidence we have on the current observation $\theta = \theta_1$ is:

$$p(\mathbf{z}|\theta_1) = 0.65, \quad p(\mathbf{z}|\theta_2) = 0.21$$

where $p(\mathbf{z}|\theta_1)$ is the measure of how "likely" a state of nature is $\theta_1$ given the observation $\mathbf{z}$, and $p(\mathbf{z}|\theta_2)$ is a measure of how "likely" the state is $\theta_2$ given the observation $\mathbf{z}$.

The best backtrack path has a cost $C_a = 27.2s$ and the cost from current position to the goal is $14s$, which are generated by the global path planner dynamically. We can calculate the loss $L(\theta_1, a_2) = 13.2s$ using equation 4.30. The loss $L(\theta_2, a_1)$ is incurred by doing extra sensing at a closed range and we assume that it has the constant value: $L(\theta_2, a_1) = 8s$. The loss matrix is given by Table 4.3.

Table 4.3: Loss Matrix 1

| Action | $\theta_1$ | $\theta_2$ |
|---|---|---|
| $a_1$ | 0 | 8 |
| $a_2$ | 13.2 | 0 |

Suppose that we have a priori distribution of the state $\Theta$ for each portion of path:

$$p(\theta_1) = 0.7, \quad p(\theta_2) = 0.3$$

Now we first calculate the posterior distribution:

$$\begin{aligned} p(\theta_1|\mathbf{z}) &= \frac{p(\mathbf{z}|\theta_1)p(\theta_1)}{\sum_{i=1}^{2} p(\mathbf{z}|\theta_i)p(\theta_i)} \\ &= \frac{0.65 \times 0.7}{0.65 \times 0.7 + 0.21 \times 0.3} = 0.88 \\ p(\theta_2|\mathbf{z}) &= \frac{p(\mathbf{z}|\theta_2)p(\theta_2)}{\sum_{i=1}^{2} p(\mathbf{z}|\theta_i)p(\theta_i)} \\ &= \frac{0.21 \times 0.3}{0.65 \times 0.7 + 0.21 \times 0.3} = 0.12 \end{aligned}$$

We next calculate the posterior expected loss for $a_1$ and $a_2$:

$$\begin{aligned} B(p(\theta|\mathbf{z}), a_1) &= L(\theta_1, a_1)p(\theta_1|\mathbf{z}) + L(\theta_2, a_1)p(\theta_2|\mathbf{z}) \\ &= 0 * 0.88 + 8 * 0.12 = 0.96 \\ B(p(\theta|\mathbf{z}), a_2) &= L(\theta_1, a_2)p(\theta_1|\mathbf{z}) + L(\theta_2, a_2)p(\theta_2|\mathbf{z}) \\ &= 13.2 * 0.88 + 0 * 0.12 = 11.62 \end{aligned}$$

Comparing the two, we see that $a_1$ has the smaller expected loss and is thus the Bayes rule. Therefore, the robot chooses the *maneuver* action to sidestep the obstacle, shown in Figure 4.12. From above derivation, we can easily see that the decision made by the robot is based on three factors: (1) the prior information about the state is heavily weighted on $p(\theta_1)$; (2) the information provided by the sensors is in favour that the state $\theta$ is likely passable since $p(\mathbf{z}|\theta_1) > p(\mathbf{z}|\theta_2)$; (3) the loss incurred by taking the action $a_2$ is larger than the loss incurred by taking action $a_1$.

### 4.5.2 Example 2

The mobile robot again moves from the start position to the goal along the straight line path which has a total cost $C_c = 21.4s$ generated by the global path planner. This cost is slightly larger than in the last example because of the increase in the uncertainty cost caused by the obstacle that partially blocked the path.

Since there is an unexpected obstacle that appears in the middle of the path, the sonars see it when the robot is approaching. When the obstacle is located at a range of $2.1m$ , the heuristic interpretation of the sonar data indicates that the path is impassable, $\theta = \theta_2$. Should the robot backtrack along the alternative path? We now see the answer provided by the Bayes rule. From Table 4.1, we have the confidence in this observation as follows:

$$p(\mathbf{z}|\theta_1) = 0.35, \quad p(\mathbf{z}|\theta_2) = 0.79$$

where $p(\mathbf{z}|\theta_1)$ is the probability that the current measurement $\mathbf{z}$ is a *miss* measurement, and $p(\mathbf{z}|\theta_2)$ is the probability that the current measurement $\mathbf{z}$ is the correct measurement.

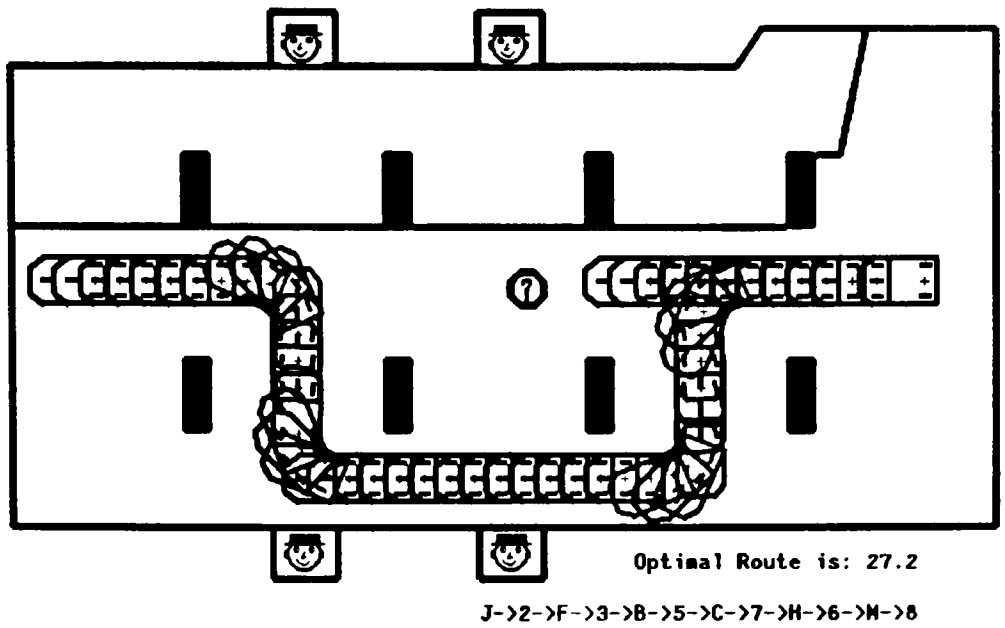



Figure 4.13: An action *backtrack* which has minimum expected loss

The cost of the alternative path from the current position to the goal is the same as in the last example, $C_a = 27.2$. However, the cost from the current position to the goal is 15.4. Therefore, we can calculate the loss $L(\theta_1, a_2) = C_a - C_c = 11.8s$ using equation 4.30. The current loss matrix is shown in Table 4.4.

Table 4.4: Loss Matrix 2

| Action | $\theta_1$ | $\theta_2$ |
|---|---|---|
| $a_1$ | 0 | 8 |
| $a_2$ | 11.8 | 0 |

Following from example 1 the prior information of the state $\theta$ for current decision-making is

$$p(\theta_1) = 0.88, \quad p(\theta_2) = 0.12$$

Then the posterior distribution of the state $\theta$ is calculated as follows:

$$\begin{aligned} p(\theta_1|\mathbf{z}) &= \frac{p(\mathbf{z}|\theta_1)p(\theta_1)}{\sum_{i=1}^{2} p(\mathbf{z}|\theta_i)p(\theta_i)} \\ &= \frac{0.35 \times 0.88}{0.35 \times 0.88 + 0.79 \times 0.12} = 0.76 \\ p(\theta_2|\mathbf{z}) &= \frac{p(\mathbf{z}|\theta_2)p(\theta_2)}{\sum_{i=1}^{2} p(\mathbf{z}|\theta_i)p(\theta_i)} \end{aligned}$$

$$= \frac{0.79 \times 0.12}{0.35 \times 0.88 + 0.79 \times 0.12} = 0.24$$

We next calculate the posterior expected loss for $a_1$ and $a_2$:

$$\begin{aligned} B(p(\theta|\mathbf{z}), a_1) &= L(\theta_1, a_1)p(\theta_1|\mathbf{z}) + L(\theta_2, a_1)p(\theta_2|\mathbf{z}) \\ &= 0 * 0.76 + 8 * 0.24 = 1.92 \\ B(p(\theta|\mathbf{z}), a_2) &= L(\theta_1, a_2)p(\theta_1|\mathbf{z}) + L(\theta_2, a_2)p(\theta_2|\mathbf{z}) \\ &= 11.8 * 0.76 + 0 * 0.24 = 8.97 \end{aligned}$$

Comparing the two, we see that $a_1$ has the smaller expected loss and is thus the Bayes rule. Therefore, the robot chooses the *maneuver* action. When the robot takes extra observations at a close range, it eventually finds that the gap is impassable. The robot has to backtrack along the alternative path, as shown in Figure 4.13. This can be intuitively explained for three reasons: (1) the probability that the path is passable, $p(\theta_1)$, is much higher than the probability that the path is impassable, $p(\theta_2)$; (2) the loss incurred by taking the alternative path, $L(\theta_1, a_2)$, is higher than the loss incurred by the deliberative maneuver, $L(\theta_1, a_1)$; (3) The sonar information is not certain. Note that although the robot has to backtrack and incurs a certain loss in this traversal, it however learns something new about the state $\theta$. The blockage information is sent to the global path planner to increase the uncertainty cost for this portion of the path to take into account when planning an global optimal path next time.

### 4.5.3 Example 3

The mobile robot is again moving from the start position to the goal along the straight line path, which is generated by the global path planner with the total cost $C_c = 29.7s$ rather that the original one since the uncertainty cost for this portion of the path was increased after encountering the blockage in the last example.

Once again, an unexpected obstacle appears in the middle of the path. When the robot is

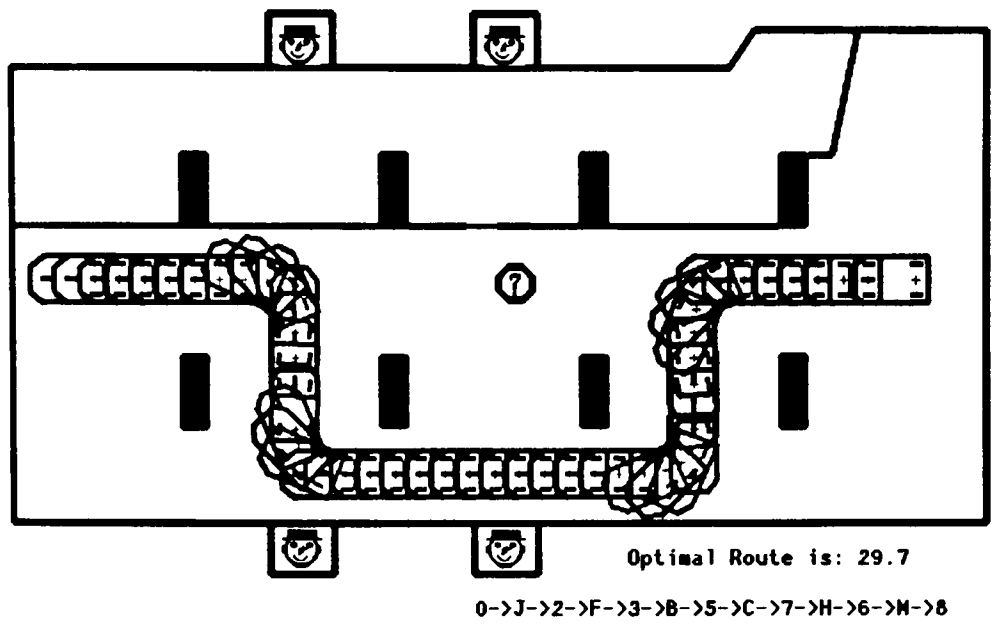



Figure 4.14: An action *backtrack* is chosen directly based on the posterior distribution of the state $\theta$

approaching it, and is at a distance $2.1m$, the sonar data is interpreted by the heuristic rule 4, which indicates that the path is impassable, $\theta = \theta_2$. From Table 4.1, the confidence in this observation is:

$$p(\mathbf{z}|\theta_1) = 0.35, \quad p(\mathbf{z}|\theta_2) = 0.79$$

where $p(\mathbf{z}|\theta_1)$ is the probability that the current measurement is *miss*, and $p(\mathbf{z}|\theta_2)$ is the probability that the current measurement is correct.

The cost of the alternative path is the same as in the last example: $C_a = 27.2$. However, the cost from the current position to the goal along the predefined path is $C_c = 23.4$. Using equation 4.30, we have the loss $L(\theta_1, a_2) = C_a - C_c = 3.8s$. Then, the current loss matrix is shown in Table 4.5.

Table 4.5: Loss Matrix 3

| Action | $\theta_1$ | $\theta_2$ |
|---|---|---|
| $a_1$ | 0 | 8 |
| $a_2$ | 3.8 | 0 |

Note that the prior information of the state $\theta$ for current decision-making is

$$p(\theta_1) = 0.76, \quad p(\theta_2) = 0.24$$

Then the posterior distribution of the state $\theta$ is calculated as follows:

$$\begin{aligned} p(\theta_1|\mathbf{z}) &= \frac{p(\mathbf{z}|\theta_1)p(\theta_1)}{\sum_{i=1}^{2} p(\mathbf{z}|\theta_i)p(\theta_i)} \\ &= \frac{0.35 \times 0.76}{0.35 \times 0.76 + 0.79 \times 0.24} = 0.58 \\ p(\theta_2|\mathbf{z}) &= \frac{p(\mathbf{z}|\theta_2)p(\theta_2)}{\sum_{i=1}^{2} p(\mathbf{z}|\theta_i)p(\theta_i)} \\ &= \frac{0.79 \times 0.24}{0.35 \times 0.76 + 0.79 \times 0.24} = 0.42 \end{aligned}$$

We next calculate the posterior expected loss for $a_1$ and $a_2$:

$$\begin{aligned} B(p(\theta|\mathbf{z}), a_1) &= L(\theta_1, a_1)p(\theta_1|\mathbf{z}) + L(\theta_2, a_1)p(\theta_2|\mathbf{z}) \\ &= 0 * 0.58 + 8 * 0.42 = 3.36 \\ B(p(\theta|\mathbf{z}), a_2) &= L(\theta_1, a_2)p(\theta_1|\mathbf{z}) + L(\theta_2, a_2)p(\theta_2|\mathbf{z}) \\ &= 3.8 * 0.58 + 0 * 0.42 = 2.20 \end{aligned}$$

Comparing the two, $a_2$ has smaller expected loss and is thus the Bayes rule. Therefore, the robot chooses to backtrack, as shown in Figure 4.14. This is because the expected loss incurred by taking *maneuver* is higher than the expected loss incurred by taking *backtrack*.

Following three examples presented above, it is clear that the Bayes rule we seek makes maximum use of the *a priori* information about the state $\theta$, i.e. $p(\theta)$. If the prior distribution $p(\theta)$ were changed sufficiently, the Bayes rule would change. However, in our application, this prior information for current decision-making is actually a previous posterior distribution which is combined with all knowledge that is obtained so far by previous traversals. It will more accurately reflect the true state $\theta$ as the robot moves around in its environment continuously. In other words, the uncertain state $\theta$ becomes less uncertain when more sensory information about it is made available.

### 4.5.4 Real-time Implementation

Figures 4.15, 4.16, and 4.17 show the results implemented by the mobile robot in our AGV Laboratory. The image sequences in the three figures demonstrate how the robot handles an

unexpected obstacle, a chair in this case, by taking either a *maneuver* or an alternative path towards the goal position that has been commanded. The robot speed is 0.6m/s. There is no stop during maneuvering.

Figure 4.15 consists of a sequence of eight frames, from frame 1 to frame 8. It clearly shows that the mobile robot takes an action to sidestep a chair which partially blocks the path. This corresponds to Example 1 described above.

Figure 4.16 shows that the mobile robot detects an unexpected chair when it moves towards the goal which is at the end of the corridor. After taking extra observations at close range, the robot found the path is impassable. Then it takes the alternative path and reaches the goal position eventually. This corresponds to Example 2.

Figure 4.17 presents the implementation of Example 3 that we have given. The sonar sensors see an chair on the way at a range $2.1m$. Then the robot makes its decision to backtrack according to optimal decision rule finding algorithm and previous traversals. There is no extra observation which is taken by further approaching the obstacle.

Finally, Figure 4.18 presents a sequence of 8 images which show that the Turtle is moving back to the docking position, and encounters a person who is moving towards it. Its sensor data is mapped into the collision zone in the local map, and the emergency handling module (see Figure 4.10) is active and controls the robot to move away from him continuously until he gives up.

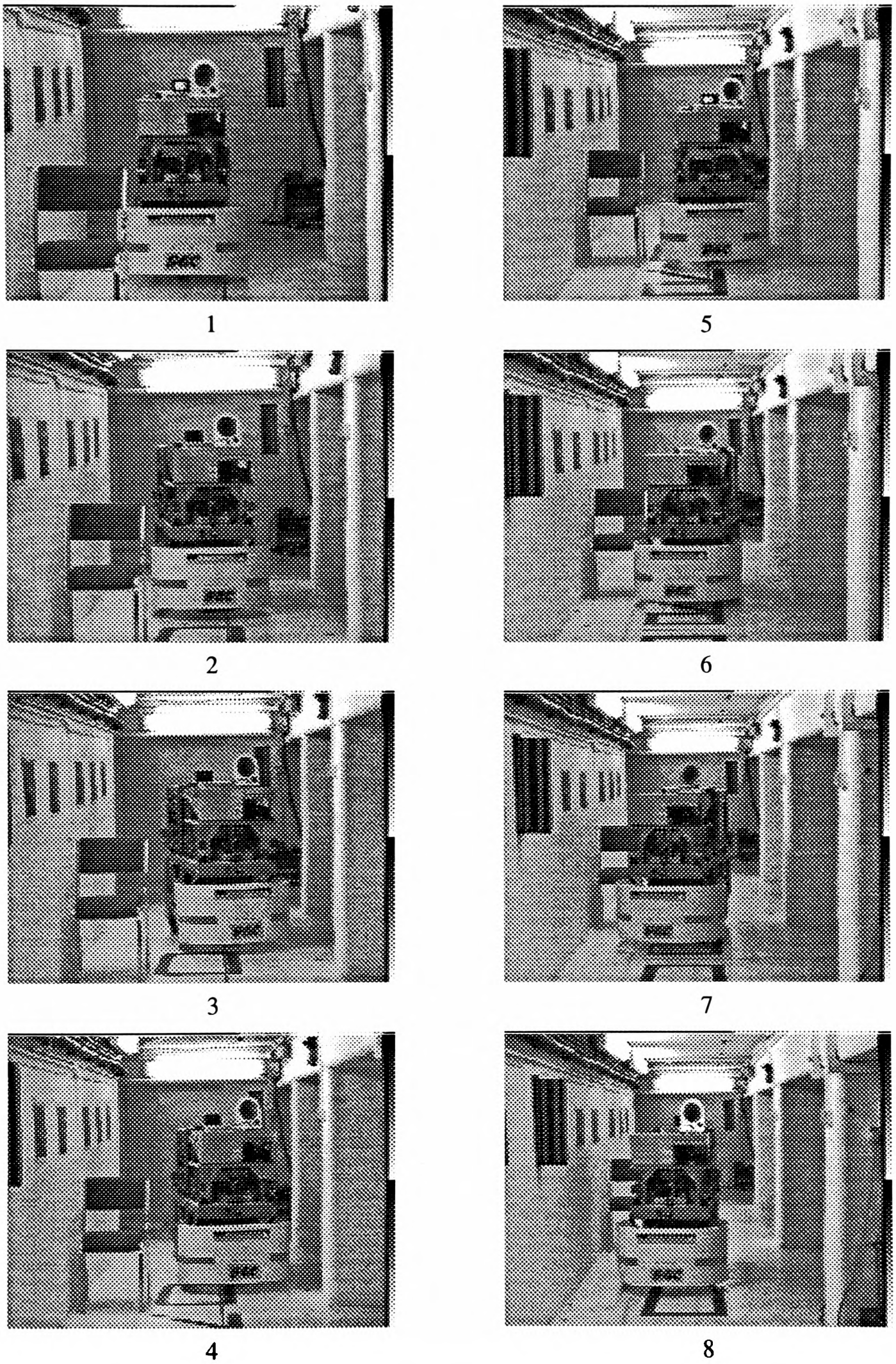



Figure 4.15: The *Turtle* mobile robot sidesteps to avoid an obstacle

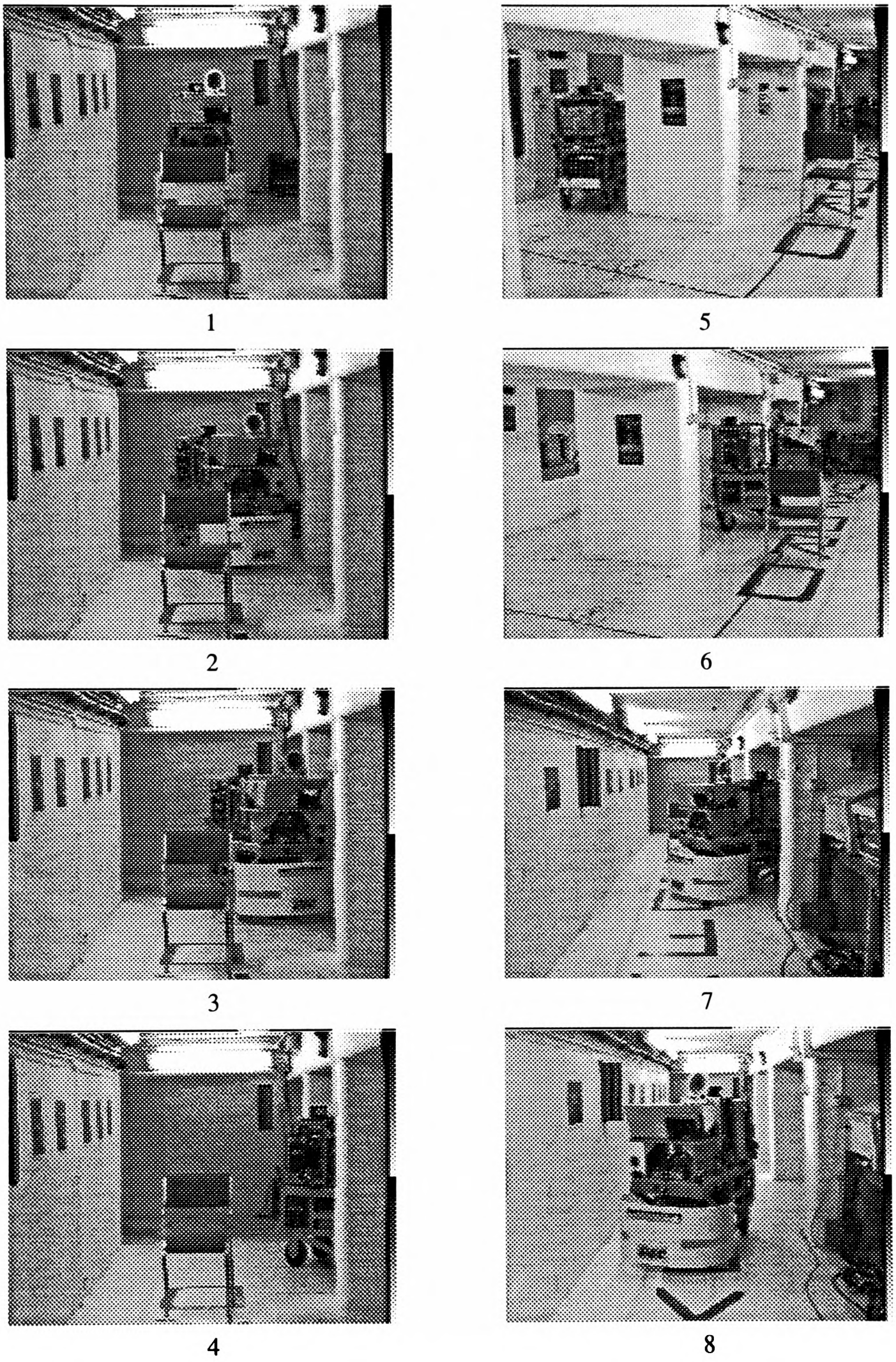



Figure 4.16: The *Turtle* mobile robot avoids an unexpected obstacle

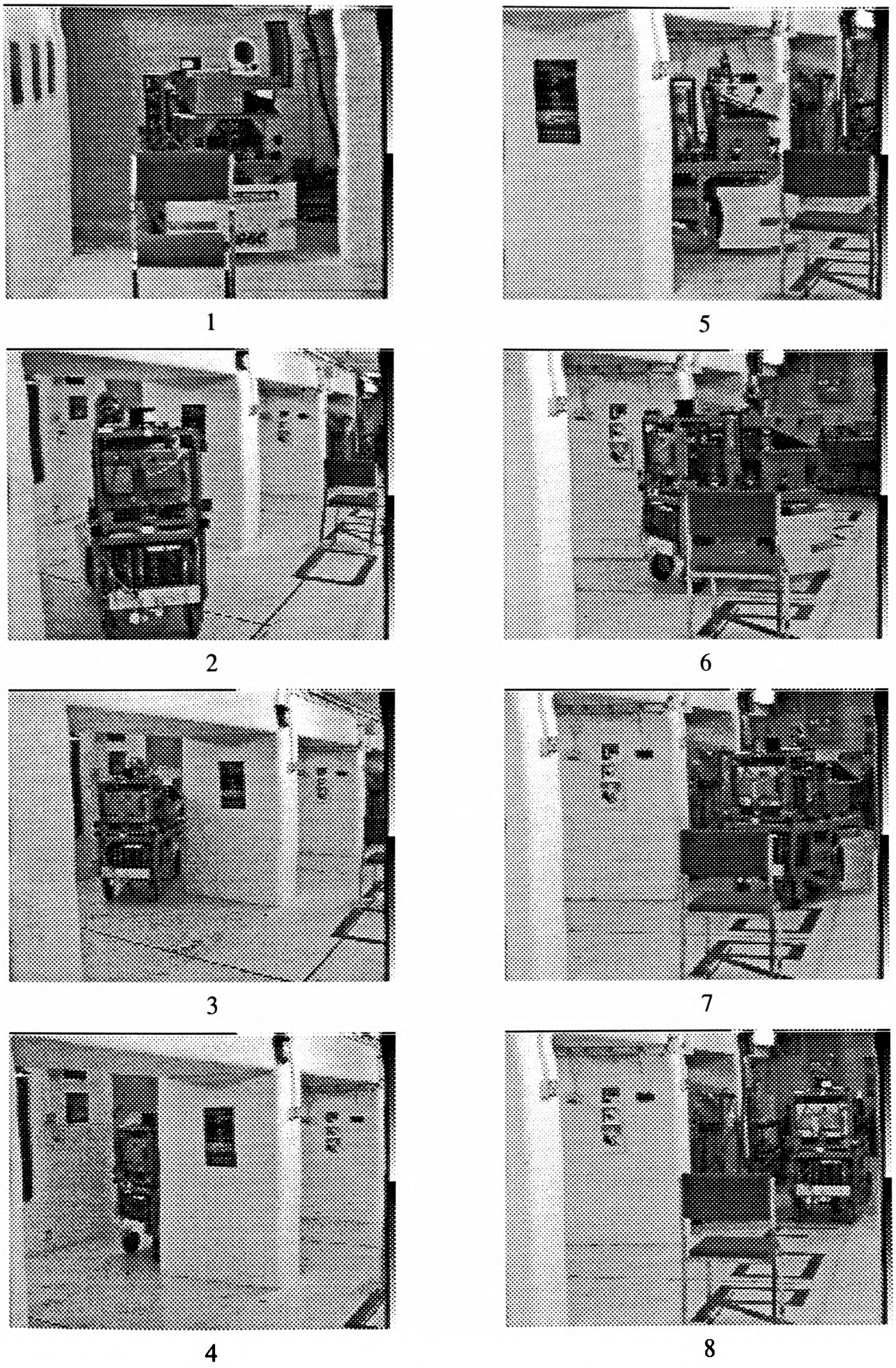



Figure 4.17: The *Turtle* mobile robot avoids an unexpected obstacles


Figure 4.18: The *Turtle* mobile robot shyaways from a person who is moving towards

## 4.6 Conclusions

In this chapter, we formulated the problem of obstacle avoidance using decision theory. It differs from the conventional approach in the sense that the solution we are looking for is not only collision free but also optimal in terms of time cost, taking into account the constraints imposed by the roadway and the vehicle kinematics. The major contribution is the treatment of uncertainties introduced by both the sensing process and the unexpected static obstacles. We give a brief summary here:

- A array of 12 sonars is adopted to detect unexpected obstacles that appear randomly in a dynamically changing environment. The sonar data is abstracted as long range, which means no obstacle close by, and short range which is mapped into the local map for collision avoidance.
- A local map consisting of a collision zone and a prediction zone is proposed to record the abstracted sonar readings. The collision zone is divided to four subzones. The avoiding control only pays attention on the subzones that lie along the motion direction of the robot.
- A simple heuristic interpreting rule for sonar readings lying in the prediction zone is proposed. To solve the problem caused by wide beam width, a probabilistic model is developed to reduce uncertainty in rule-based interpretation.
- We have cast the collision avoidance problem into the framework of decision theoretic approach. A recursive decision-making algorithm is given to provide a feasible way to perform path planning locally in the face of uncertainty.
- The optimal solution of avoiding unexpected static obstacles is obtained by trading between deliberative maneuver and the alternative according to Bayes decision rule. The results of the implementation are sent back to the global path planner to plan an global optimal path next time.

# Chapter 5

# Trajectory Planning and Optimal Tracking

Planning collision-free paths for a mobile robot moving in a dynamically changing roadlike industrial environment has been discussed in earlier chapters. However, such planned paths consist of a set of intermediate points that can not be used directly to control the motion of the robot. A precise geometric trajectory with a velocity profile needs to be created, taking into account the robot kinematics. This chapter introduces a unified approach to trajectory planning and tracking for a three-wheeled mobile robot subject to non-holonomic constraints. We will show (i) how a smooth trajectory is generated that takes into account the constraints from the dynamic environment and the robot kinematics; and (ii) how a proposed motion controller works to provide an optimal tracking capability. The tracking performance of the proposed guidance system is analysed by simulation.

## 5.1 Introduction

Trajectory generation and tracking for a mobile robot is a fundamental, relatively low level, planning and control task. It connects high level planners (global and local) to low level locomotion control, in order to implement globally optimal or collision-free paths. It is normally implemented by a two-layered system composed of a *trajectory planner* and a *motion controller*. The trajectory planner generates a nominal motion plan, consisting of a geometrical path and a velocity profile along this path to satisfy kinematic and dynamic constraints of the robot. In contrast, the motion controller guides the mobile robot to execute this nominal path and to track it with minimum deviation. Many different methods have been proposed and implemented, based on a variety of robot platforms:

wheeled, tracked or legged.

A car-like mobile robot built on a wheeled platform with one or two front steering wheels has a minimum turning radius. It is unable to follow an arbitrary path or change its direction instantaneously. This constraint is a limit on the path geometry, namely non-holonomic. Typically, as in the case of our Turtle robot, the two rear wheels are not steerable and the requirement that they should rotate without slipping on the floor introduces a constraint on the vehicle motion: the velocity of the center of the rear axle must be collinear with the orientation axis of the vehicle base. Therefore, a trajectory that is generated for the Turtle is required to satisfy such a path constraint. Otherwise, the planned trajectory cannot be followed at any speed and a new trajectory must be regenerated. Since the Turtle is expected to travel within a roadway, the trajectory generation becomes more complicated. From this point of view, many established trajectory planning methods prove inadequate for our Turtle mobile robot. The problem is simplified as the well-known *piano mover* approach in which the trajectory planning problem for any mobile is turned into trajectory planning for a point in the configuration space, since an arbitrary path in the admissible configuration space does not necessarily correspond to a feasible trajectory for the real robot [Laumond, 1991].

We begin with a literature review of previous research on non-holonomic trajectory planning and tracking in the next section. A unified approach to trajectory generation and tracking is proposed in Section 3. Section 4 presents a set of different trajectory modules which can be used to generate continuous curvature trajectories subject to constraints from the roadway, the robot kinematics, and the steering mechanism. An optimal controller is designed in Section 5 to provide smooth tracking capability in a predictive manner. The performance of the proposed trajectory generation and tracking strategy is analysed by simulation in Section 6. Finally, a brief summary is given in Section 7.

## 5.2 Previous Work

Non-holonomic trajectory planning has been studied recently, and some important theoretical and practical results are already produced. Laumond [1986] first attacked the problem and gives the condition for the existence of a trajectory for such a mobile. To reduce the expense of backing up maneuvers, he proposed a method for finding collision-free trajectories without backing up [Laumond, 1987]. Fraichard [1991] proposed a smooth trajectory planning algorithm for a car-like robot in a structured world. The generated trajectories are composed of straight segments and circular arcs to satisfy the constraints by natural lanes. Graettinger and Krogh [June 1989] described a method for evaluating the feasibility of trajectories generated by path-planning systems for wheeled mobile robots. Their approach presented useful conclusions that when path constraints are violated the path must be modified to achieve feasibility, while violations of kinematic constraints (velocities and accelerations) and dynamic constraints (torque input and frictional force) can be eliminated by time scaling so that the same path is followed at a slower speed. Other contributions to the problem include a heuristic motion planning algorithm by Tournassoud and Jehl [1988], a new motion planner based on a mixed global/local approach by Laumond *et al.* [1990], and a maneuvering trajectory generation method by Pin and Vasseur [1990].

To track a given smooth trajectory, feedback control is widely used in mobile robots, based on prior information about the environment and information from odometry and other location sensors. The *Turtle* mobile robot in the Oxford AGV Laboratory adopted a method in which the reference point sequence is stored dynamically in memory as a position reference file. During each cycle of locomotion control (80ms), the robot's current position(feedback) and the future position in the memory are compared to determine the next steering angle and transition by a proportional controller [Hu, 1990]. A similar approach can be found in [Tsumura *et al.*, 1981]. Kanayama [1988] first proposed a method using straight line segments for the robot's locomotion instead of a sequence of points. He suggested representing three degrees of freedom in the robot's position by

a posture $\mathbf{p} = [x, y, \theta]^\top$ and the use of the error posture for vehicle control, combining this with a PID control algorithm to determine linear and rotational velocity rules in [1988]. Nelson and Cox [1988] proposed an error-feedback controller for path tracking. In their method, a linear function of path error is adopted in control rules for steering and linear velocity. Crowley [1989a] developed a three layered locomotion controller to provide asynchronous independent control of forward displacement and orientation. Iida and Yuta [1990] proposed a feedback controller including a feedforward compensator based on inverse dynamics of the vehicle for trajectory tracking.

Conventional PID controllers are widely used for trajectory tracking in non-holonomic mobile robots, based on feedback of the position error and orientation error, with little analysis imposed by the constraints of the kinematics and dynamics of the vehicle. However, more analysis has been made recently for the calculation of control parameters and the performance of the control system. Kanayama *et al.* [1990] formulated the tracking problem in a systematic way. By introducing a Liapunov function, they determined a controller structure to stabilise the system to minimise the position and orientation errors for a non-holonomic vehicle. Based on a dynamic model of the robot with a front steering wheel, Hemami *et al.* [1990] introduced a nonlinear controller which includes the angular velocity of the vehicle in the feedback to obtain good performance in path tracking. More recently, the synthesis of an optimal control law for a three-wheeled front steering vehicle is given in [Hemami *et al.*, 1992]. It is based on a linearised plane motion dynamic model of the vehicle and the minimisation of a quadratic performance index incorporating the offset and the orientation errors of the vehicle in path tracking. Sordalen and Wit [1992] proposed an exponentially stable controller for a non-holonomic mobile robot to track a sequence of fixed points consisting of positions and orientations. It yields stabilisation about an arbitrary point in the state space. Samson [1992] recently revisits the path following problem for a nonholonomic mobile robot and proposes a nonlinear angular velocity control law which approximates locally optimal linear feedbacks and ensures convergence to the desired path, independently of the robot's advancement velocity.

Trajectory planning and tracking remains an important topic for applications of non-holonomic mobile robots in real world. In this chapter, we formulate the problem of planning and controlling the motion of the non-holonomic mobile robot as an optimal control problem. An optimal motion controller is sought to determine the vehicle's linear and rotational velocities such that an objective function in position and orientation errors is minimised.

## 5.3 Problem Formulation

A brief introduction to the problem, and a few preliminary definitions and assumptions are given in this section.

### 5.3.1 Vehicle Kinematics

The mobile robot we consider is a three-wheeled vehicle, shown in Figure 5.1. It is driven and steered by two DC motors which provide torques acting on the front wheel. The two rear wheels are passive 'idlers'. We assume that the robot moves on a horizontal plane, satisfying the conditions of rolling without slipping and ignoring gravitational forces.

To describe the motion of the robot, we introduce a fixed global reference frame $\mathcal{F}_0 = (X0Y)$. Then, we define a robot local frame $\mathcal{F}_1 = (X_1 0_1 Y_1)$ whose origin is the guidepoint located at the middle of the axle of the two rear wheels (the center of rotation of the vehicle). Both are Cartesian coordinate systems. The following notation is used in deriving the kinematics of the vehicle:

- $0_1$: the guidepoint that is used to follow the reference path generated by the global path planner.
- $R_f$: the radius of the front, steered wheel.
- $B$: the distance between $0_1$ and the steered wheel, namely the wheelbase.
- $\alpha$: the steering angle with respect to the robot central axis $O_1 X_1$
- $\omega$: the front wheel rotational velocity about its horizontal axis.

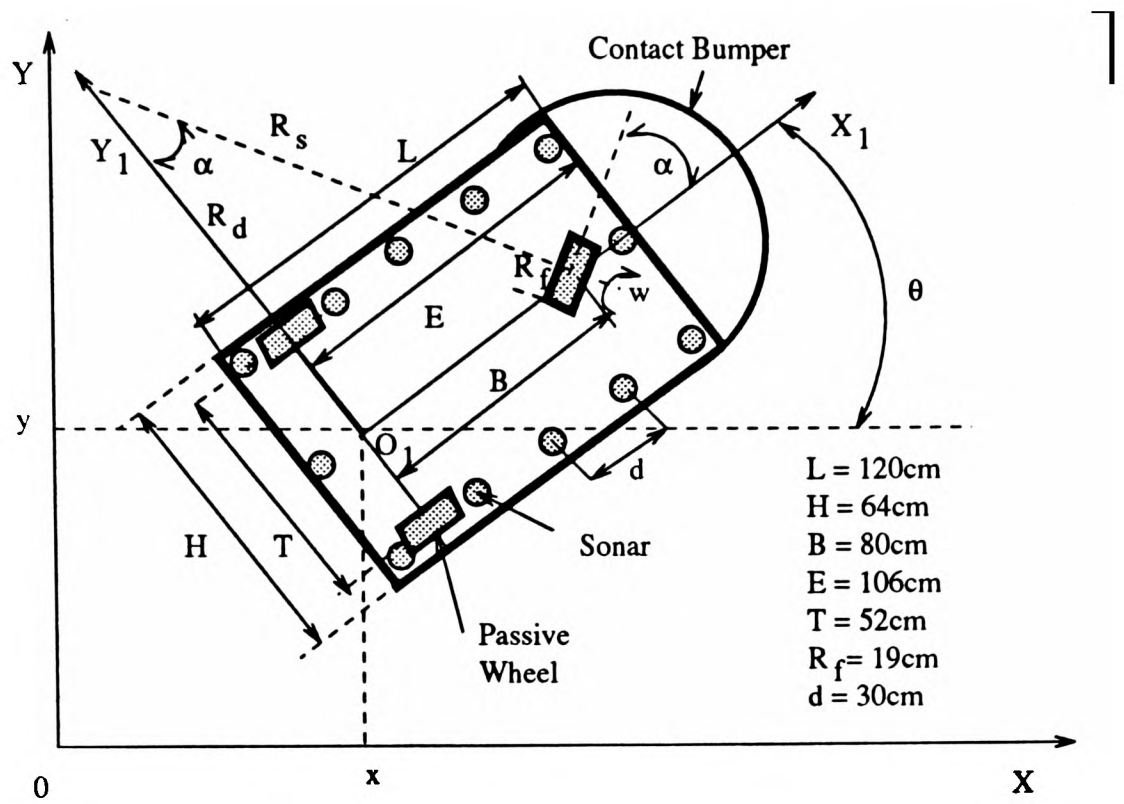



Figure 5.1: The *Turtle* mobile robot Coordinates and parameters

- $\theta$: the robot's orientation with respect to the frame $\mathcal{F}_0$, considered counterclockwise from its $X$ axis.
- $x, y$: the coordinates of the guidepoint $0_1$ in the frame $\mathcal{F}_0$.

In the global reference frame $\mathcal{F}_0$ the robot possesses three degrees of freedom in its position which are represented by the state vector $\mathbf{x} = [x, y, \theta]^T$. However, it has two degrees of freedom for locomotion. Let $v(t)$ be the linear velocity along the x-axis of the local frame $\mathcal{F}_1$ and $\mu(t)$ the rotational velocity which is normal to the x-axis of $\mathcal{F}_1$. The vehicle's kinematics are defined by the Jacobian matrix $J$:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & \frac{1}{B} \end{bmatrix} \begin{bmatrix} v \\ \mu \end{bmatrix} = J\mathcal{U} \tag{5.1}$$

Note that $J$ is non-square since the vehicle is non-holonomic.

This nonlinear kinematic equation describes the motion of the robot, based on the guidepoint $0_1$ and the global frame $\mathcal{F}_0$. In the case where the front wheel is driven and steered, the physical control variables are the rotational velocity $w$ and the steering angle $\alpha$. Therefore, we have the

following equations:

$$\mathcal{U} = \begin{bmatrix} v \\ \mu \end{bmatrix} = \begin{bmatrix} R_f \omega \cos\alpha \\ R_f \omega \sin\alpha \end{bmatrix} = \mathbf{f}(\alpha, \omega) \tag{5.2}$$

where $|\omega| \leq \omega_{max}$, $|\alpha| \leq \alpha_{max}$.

From the first two rows of Equation 5.1, we can derive one independent equation of constraint in terms of velocity:

$$\dot{x} \sin\theta - \dot{y} \cos\theta = 0 \tag{5.3}$$

This equation is not integrable. It is a constraint on the velocity of the robot but does not affect the dimension of the space of configurations, i.e. the vehicle can have any orientation at any position. On the other hand, for a given configuration $(x, y, \theta)$, the space of achievable velocities has only two dimensions. This is therefore a non-holonomic constraint [Laumond, 1986].

Another characteristic of the vehicle is that it has a minimum turning angle, which imposes a constraint on the vehicle's motion. Let $R_d$ denote the radius of curvature of the trajectory of the guidepoint $0_1$ and $R_s$ the radius of curvature of the front wheel trajectory according to the steering angle $\alpha$, as shown in Figure 5.1. Given the constraint on the steering angle $\alpha_{\max}$, the minimum turn radius, $R_{d\min}$ of the vehicle is

$$R_{d\min} = \frac{B}{\tan\alpha_{\max}} \tag{5.4}$$

In contrast, the minimum radius of curvature of the front wheel trajectory is

$$R_{s\min} = \frac{B}{\sin\alpha_{\max}} \tag{5.5}$$

Clearly, the robot performs pivot turning only when $\alpha_{\max} = \pi/2$.

### 5.3.2 Trajectory Generation

In Chapter 3 and 4, we addressed the problem of global and local path planning. As defined there, a globally planned path is a set of intermediate points which is used to move the mobile robot from its current topological space $\mathbf{n}_0$ to the goal $\mathbf{n}_f$, based on a minimum time cost criterion. When an

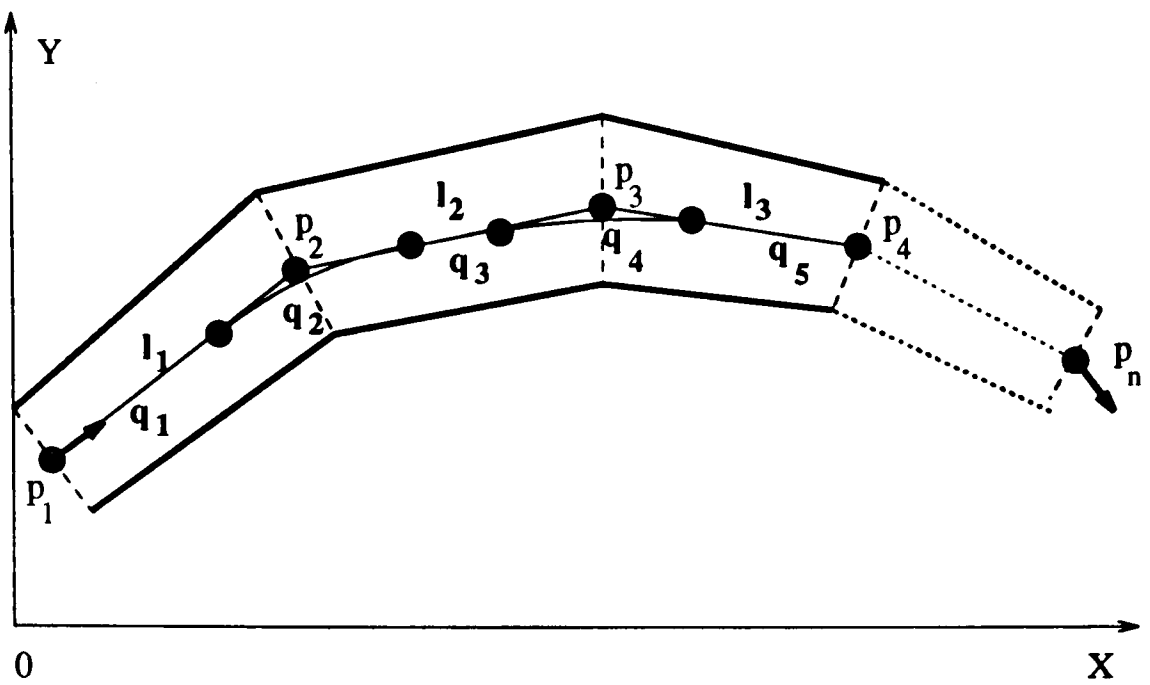



Figure 5.2: Trajectory generation

unexpected obstacle is detected, the local path planner either generates new intermediate points to detour around it or to shy away from it. In this way, the robot paths can easily be generated and dynamically modified. However, to travel along such a planned path, any specific mobile robot needs a precise geometric trajectory to be created.

Recall that a globally planned path can be represented as a sequence

$$\mathcal{F} = \{p_1, p_2, ..., p_n\} \tag{5.6}$$

where each point, $p_i = (x_i, y_i, d_i), i = 1, 2, ...n$, $(x_i, y_i)$ is the node $\mathbf{n}_i$ defined in the global frame $\mathcal{F}_0$, and $d_i$ is an associated path width as shown by the dashed lines in Figure 5.2.

Given the sequence $\mathcal{P}$, the purpose of the trajectory planner is to generate a trajectory $\mathcal{L}$ which is (i) collision-free, (ii) remains in the lane defined by $\mathcal{P}$, and (iii) is executable by the robot subject its kinematic constraints. The trajectory planner we have designed operates in two steps. First, the path $\mathcal{P}$ is preprocessed as a sequence of line segment vectors each of which connects consecutive path points, $p_i$ and $p_{i+1}$:

$$\mathcal{L} = \{\mathbf{l}_1, \mathbf{l}_2, ..., \mathbf{l}_{n-1}\} \tag{5.7}$$

where each element vector, $\mathbf{l}_i$, consists of a desired velocity $v_i$, a maximum position deviation (or

offset) $\epsilon_i$ and a maximum orientation deviation $\eta$ each of which is associated with the line segment $l_i$ by the global path planners, modified by the local path planner if the robot encounters obstacles during its motion, described as follows:

$$\mathbf{l}_i = [p_i, p_{i+1}, v_i, \epsilon_i, \eta_i]^\top \tag{5.8}$$

Secondly, the trajectory planner generates a smooth trajectory composed of straight lines and arcs that are obtained by fitting splines or polynomials (see the next section) between those line segments that have different slopes, and taking into account the constraints on the vehicle kinematics and the widths of the roadways, as shown in Figure 5.2.

**Definition 6** *Let C denote a smooth trajectory and* $\mathbf{q}_i$ *a trajectory segment vector.* $m$ *is the number of trajectory segment vectors and is equal to or larger than the number of path segments,* $n - 1$. *Each* $\mathbf{q}_i$ *has a segment type c, a start position* $p_i$, *an end position* $p_{i+1}$, *a radius of curvature* $R_d$, *a maximum velocity* $v$, *and maximum position deviation* $\epsilon$, *as well as maximum orientation error* $\eta$. *Then a trajectory is defined as*

$$\mathcal{C} = \{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_m\} \tag{5.9}$$

*where*

$$\mathbf{q}_i = [c_i, x_i, y_i, x_{i+1}, y_{i+1}, R_d, v_i, \epsilon_i, \eta_i]^\top \tag{5.10}$$

The segment type $c_i$ will be defined in Section 5.4.

In this way, the globally planned path $\mathcal{P}$ is decomposed into a sequence of trajectory segment vectors $\mathbf{q}_i$ which we define as non maneuver trajectories, for the robot to move along. When the robot reaches the goal node, it may need some maneuver trajectories to dock precisely to a given orientation to implement useful tasks. The next section addresses how to generate both trajectories.

### 5.3.3 Path Tracking and Error State Vector

From a sequence of trajectory segment vectors 5.9, the trajectory planner further generates a sequence of incremental reference state vectors for the motion controller to track at each control

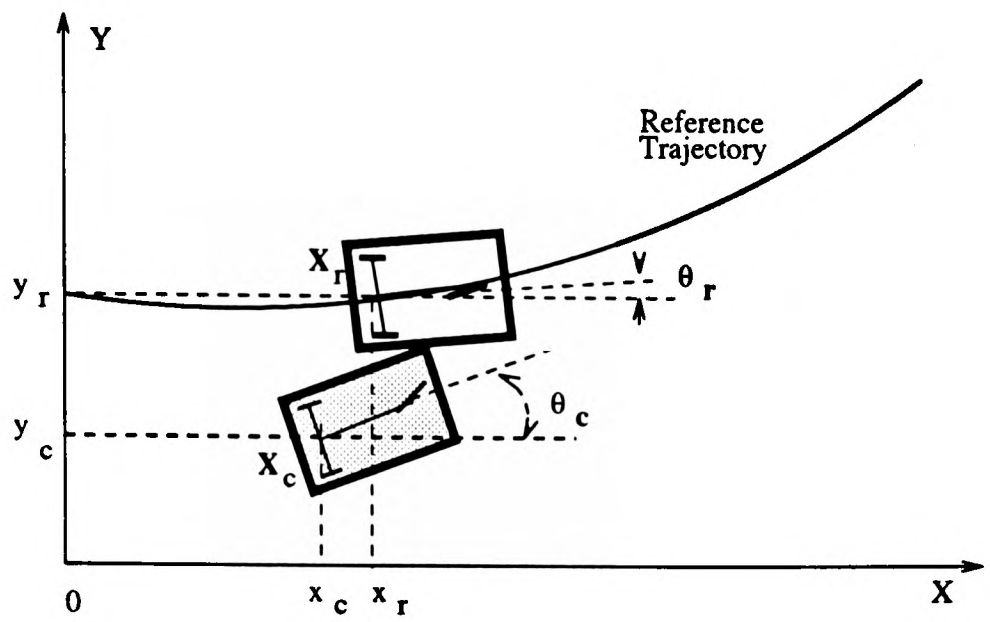



Figure 5.3: Reference and current position

cycle time (80ms in our case). However, the robot may deviate from the desired reference state because of tracking errors and disturbances from the floor, as shown in Figure 5.3. An error state vector is introduced to represent the difference between the reference vector and the current state of the vehicle.

**Definition 7** *Let* $\mathbf{x}_r = [x_r, y_r, \theta_r]^\top$ *be a reference state vector.* $\mathbf{x}_c = [x_c, y_c, \theta_c]^\top$ *denotes the current state vector of the robot, measured by odometry and external sensors. An error state vector* $\mathbf{x}_e = [x_e, y_e, \theta_e]^\top$, *representing the difference between* $\mathbf{x}_r$ *and* $\mathbf{x}_c$, *is defined as*

$$\mathbf{x}_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} x_c - x_r \\ y_c - y_r \\ \theta_c - \theta_r \end{bmatrix} = (\mathbf{x}_r - \mathbf{x}_c) \tag{5.11}$$

Note that the transformation of the error vector $\mathbf{x}_e$ into the robot local frame $\mathcal{F}_1$ is realised using the following equation

$$\mathbf{x}_e^1 = \begin{bmatrix} x_e^1 \\ y_e^1 \\ \theta_e^1 \end{bmatrix} = \begin{bmatrix} \cos\theta_c & \sin\theta_c & 0 \\ -\sin\theta_c & \cos\theta_c & 0 \\ 0 & 0 & 1 \end{bmatrix} (\mathbf{x}_r - \mathbf{x}_c) \tag{5.12}$$

To track the reference trajectory, the motion controller is responsible for reducing the error state vector to a tolerance range, $|\mathbf{x}_e| \leq \chi$, where $\chi = [\epsilon, \epsilon, \eta]^\top$ is application specific.

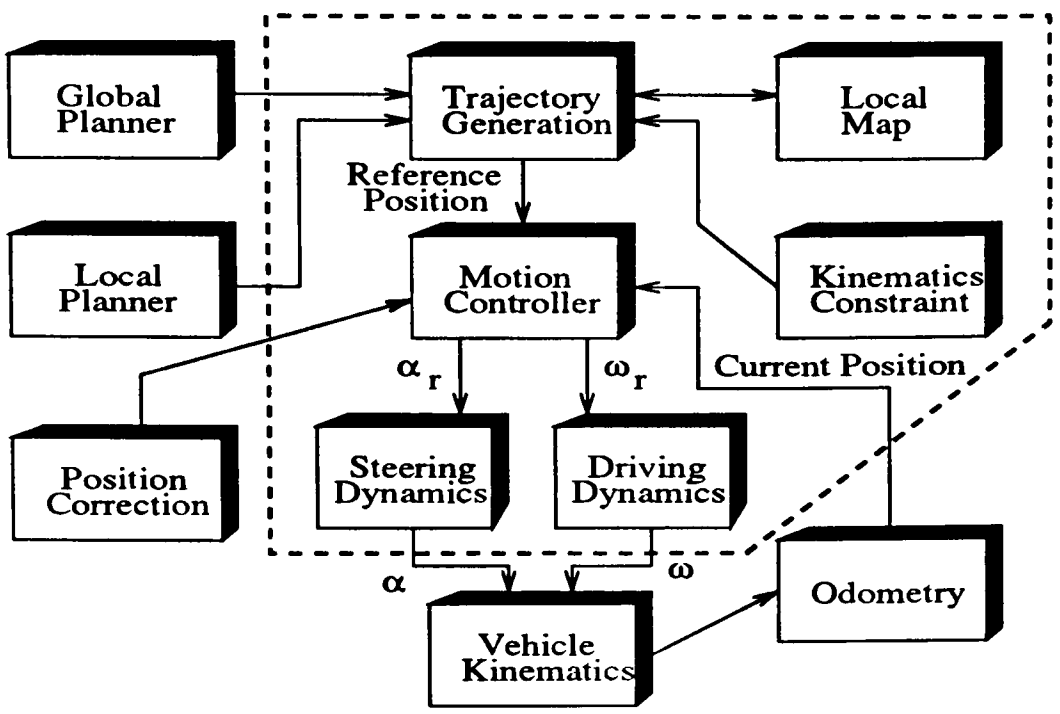



Figure 5.4: Trajectory planner and motion controller

### 5.3.4 Unified Approach

We propose a new guidance system for trajectory generation and trajectory tracking, taking into account constraints from the dynamic environment and the robot kinematics. Figure 5.4 shows the architecture of the proposed system. It consists of two layers: a trajectory generator and a motion controller. This is the same as a conventional guidance system. However, conventional two-layered guidance systems normally include only position information from odometry and, perhaps, from other external location sensors. The system we propose is provided in addition with three new types of information: (1) a local map in the vicinity of the vehicle, (2) a geometrical and kinematic model of the vehicle, and (3) a set of behavioural modules. The information (2) and (3) constitutes the a priori knowledge. More detail is given in the next two sections. In contrast, information (1) is dynamically updated by whatever external sensors are available.

The reason we include information about dynamic changes in the vicinity of the robot in the guidance system is that the robot needs information about its surroundings to perform maneuvers such as docking, pivot turning, and escaping from a deadend or road blockage. All such maneuvers may require the robot to approach very close to its environment. At such moments, obstacle

detection needs a criterion that is different from the one used by the local path planner. There are two ways to solve this problem. One is to provide the local path planner with different criteria for obstacle detection. Another is for the guidance system itself to judge collisions during the maneuver. This is exactly the situation when one is parking a car in a narrow gap. If the driver is provided with enough information about the vicinity of his car by external sensors, there is no doubt that he can do it very well by himself. However, if one person maneuvers the car to obey the commands of another person outside, misunderstanding and slow responses may cause a collision. Therefore, we build obstacle detection and an avoiding capability into the guidance system so that it can make judgements for itself during maneuvering.

## 5.4 Trajectory Planning

This section presents a method for generating smooth trajectories, taking into account the constraints from the changing world and from the robot itself. Restricting the motion of the robot for safety to designated lanes (roadways) in a factory environment, we also take into account roadway boundaries during trajectory planning. Most trajectory planning methods for a non-holonomic mobile robot concentrate on how to generate a smooth trajectory without roadway boundaries. As a result some of these methods may not be directly applicable to the constraint space of a roadway. Moreover, the robot must perform parking and docking in a limited space, and so a maneuvering capability is needed. Therefore, the trajectory planning method we propose is intelligent in the sense that it consists of many different planning modules and is capable of choosing one automatically according to different task and constraint conditions.

### 5.4.1 Different Trajectory Modules

The path given by (5.7) is a non-maneuver smooth trajectory for a non-holonomic vehicle along which the robot can move without having to stop, pivot turn or reverse. Three kinds of non-maneuver trajectories are considered: (1) a line segment trajectory since the path (5.7) consists of

a sequence of line segments; (2) a curve enables smooth transition between two successive line segment trajectories in different directions; (3) a lane change trajectory that is required to avoid an unexpected obstacle that partially blocks the path.

### Straight line

Straight line paths are frequently used by mobile robots that move along the corridors of a factory. It is also needed for a long trajectory. It is very simple and straightforward to generate. Let $(x_1, y_1)$ be the start point and $(x_2, y_2)$ the end point. The straight line trajectory between two points is defined as follows:

$$\mathbf{q} = [1, x_1, y_1, x_2, y_2, k_s, v_1, \epsilon, \eta]^\top \tag{5.13}$$

where the driving speed $v_1$ and the maximum position and orientation errors $(\epsilon, \eta)$ are defined by the global and local path planners. Note that the type of trajectory for the straight line is given by $c = 1$.

The reference steering angle and drive speed for straight line segments are

$$\alpha_r = 0, \quad \omega_r = v_1 / R_f \tag{5.14}$$

where $R_f$ is the radius of the front wheel of the robot.

### Arc turn

An *arc turn* is a curve that produces a smooth transition between lines intersecting at an arbitrary turn angle and which is symmetric with respect to the line that bisects the angle. It is one of the most useful trajectories for a mobile robot that travels along a network of roads. Fitting a circular-arc for an arc turn is geometrically convenient, and in certain cases is optimal for constructing minimum length paths [Dubins, 1957]. It is also a good choice for a non-holonomic vehicle to turn at a junction subject to constraints from the road boundary. However, circular arc turns inevitably involve a steering discontinuity at the line-arc-line transition points since line segments have zero curvature and circular arcs have a constant non-zero curvature. Figure 5.5(a) shows that the center

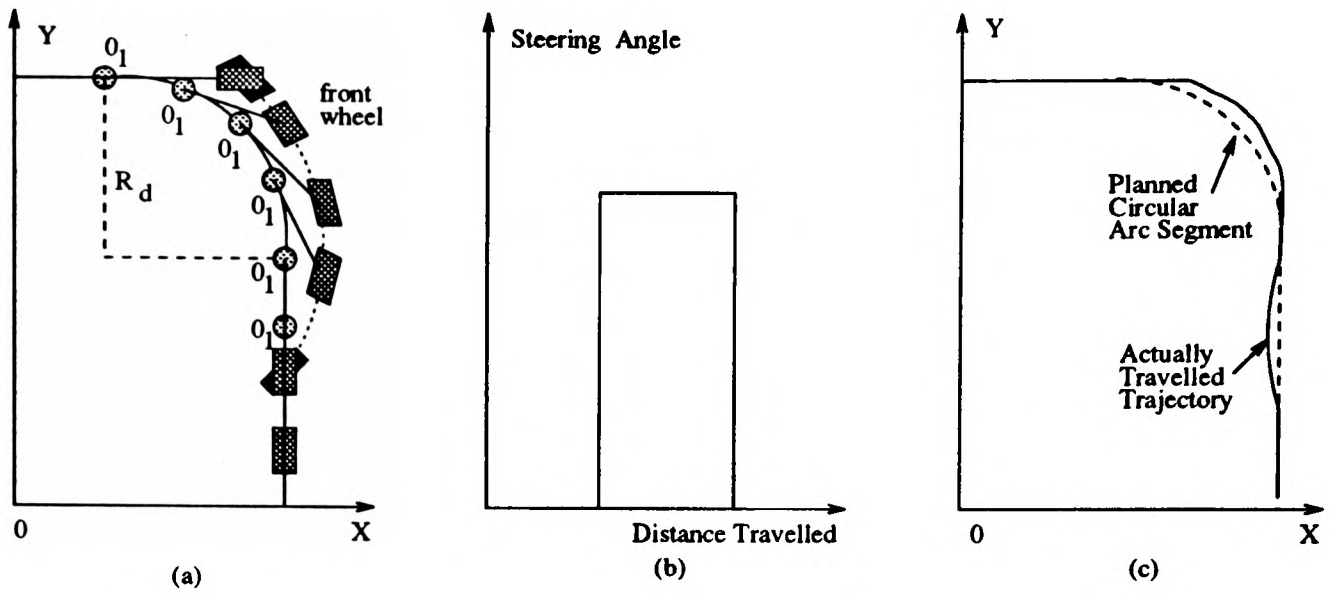



Figure 5.5: Tracking a line-arc-line trajectory

of rotation of the robot $0_1$ is tracking a line-arc-line trajectory. Instantaneous changes of steering angle at the transition points are needed to track the trajectory, as shown in Figure 5.5(b). However, since the steering angle can not be changed instantaneously in practice, errors in the position and heading of the robot increase at each transition point, as shown in 5.5(c). One inefficient way to eliminate the errors would be to stop the vehicle at the transition points to turn the steering wheel to the desired angle. This obviously wastes time.

Instead of circular arc turns, we would like to have a smooth curve with continuous curvature that does not deviate too far in shape from a circular arc, yet satisfies the position, heading, and curvature constraints imposed at the start and end points, as well as at the road boundary.

*Polar Spline Path*

The polar spline path proposed by Nelson [1989a] is reasonably close to a circular arc, and can produce continuous-curvature arc turns of arbitrary angle. It is called a single polar-polynomial (SPP) curve, and has a closed-form expression:

$$r(\phi) = R(1 + \frac{\phi^2}{2} - \frac{\phi^3}{\Phi} + \frac{\phi^4}{2\Phi^2}) \tag{5.15}$$

where $R$ is the radius of the circular arc connecting the two line segments, and $\Phi$ is the total angle of the turn. Here $r(\phi)$ is the radius of curvature of the path, parameterised by $\phi \in [0, \Phi]$. It is

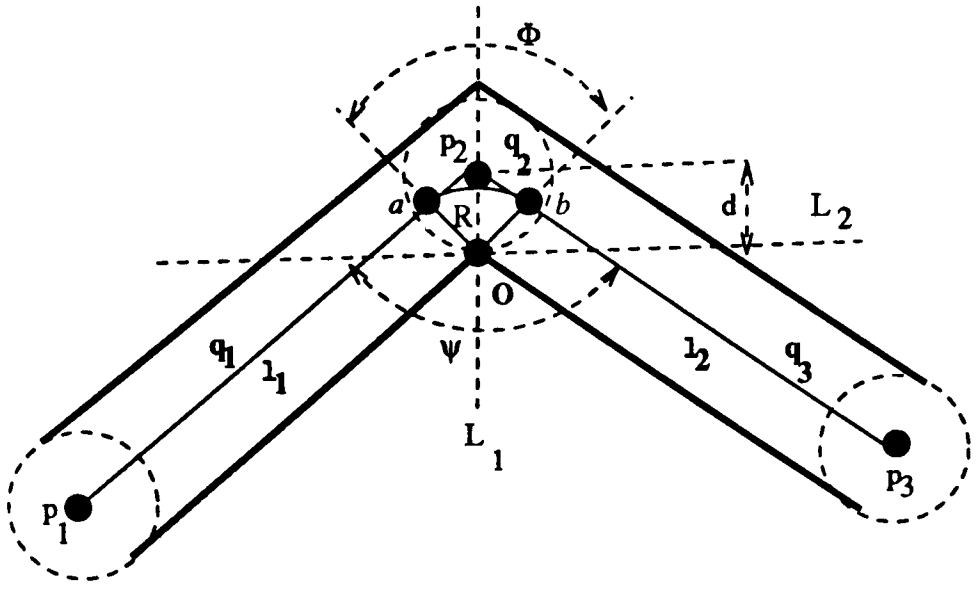



Figure 5.6: A arc turn trajectory with roadway constraints

required to satisfy $r(0) = r(\Phi) = R$.

The SPP matches the boundary conditions of the two lines it joins:

$$\begin{aligned} r = R, \quad \dot{r} = 0, \quad \kappa = 0, \quad \text{at } \phi = 0 \\ r = R, \quad \dot{r} = 0, \quad \kappa = 0, \quad \text{at } \phi = \Phi \end{aligned} \tag{5.16}$$

where $\dot{r} = dr/d\phi$. $\kappa$ is the curvature of the SPP, given by

$$\kappa = \frac{|r^2 + 2\dot{r}^2 - r\ddot{r}|}{(r^2 + \dot{r}^2)^{3/2}} \tag{5.17}$$

To apply the SPP to the arc turn of a mobile robot which travels along the roadway, we define an algorithm here to choose the value of $R$ and $\Phi$ subject to the constraints of the roadway boundary. It can also be used for other continuous-curvature arc turns discussed later. Figure 5.6 shows how we take into account roadway constraints to choose both $R$ and $\Phi$.

**Algorithm 2** *Continuous-curvature Arc Turns*

1. *A path in Figure 5.6 consists of two line segments $l_1$ and $l_2$ joining at $p_2$ that enclose an angle $\psi$.*
2. *The bisecting line $L_1$ is found by halving the angle $\psi$.*
3. *Line $L_2$ is perpendicular to $L_1$, at distance d from point $p_2$ (half road width).*

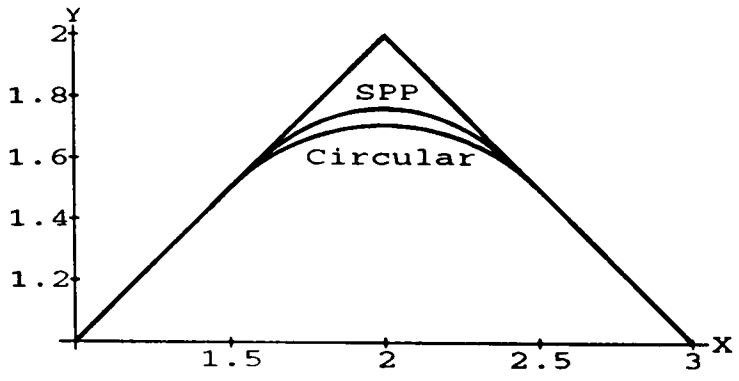



Figure 5.7: A circular arc and a SPP

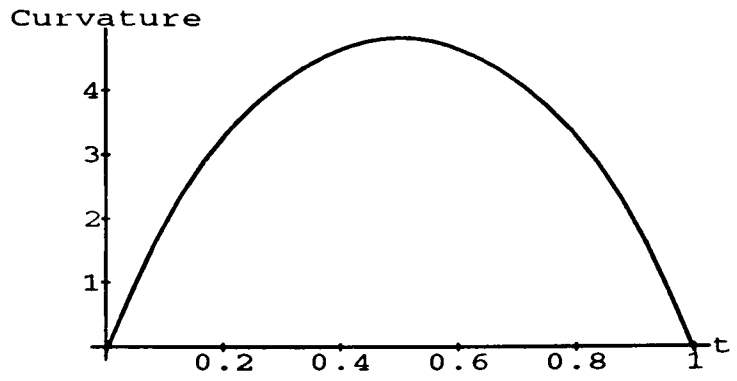



Figure 5.8: The SPP curvature

4. *R is then obtained by drawing two lines perpendicular to* $l_1$ *and* $l_2$ *with two cross points* a *and* b *which are the start and end points of the arc turn, depending upon the direction of motion.*

5. $\Phi$ *is calculated by* $\Phi = \pi - \psi$.

6. *The* $(x, y)$ *coordinates of the polar spline are finally obtained by* $x = r\cos\phi$ *and* $y = r\sin\phi$.

For instance, we choose two perpendicular lines with slopes $k_1 = 1$ and $k_2 = -1$, as shown in Figure 5.7. Using the above algorithm, we obtain

$$R = \sqrt{2}/2, \quad \Phi = \pi/2, \quad a(x_1, y_1) = (1.5, 1.5), \quad b(x_2, y_2) = (2.5, 1.5). \tag{5.18}$$

Then the SPP arc to join the lines is given by the equations

$$x = 2 + R(1 + \frac{\pi^2}{8}t^2 - \frac{\pi^2}{4}t^3 + \frac{\pi^2}{8}t^4)\cos(\frac{\pi}{4} + \frac{\pi}{2}t) \tag{5.19}$$

$$y = 1 + R(1 + \frac{\pi^2}{8}t^2 - \frac{\pi^2}{4}t^3 + \frac{\pi^2}{8}t^4)\sin(\frac{\pi}{4} + \frac{\pi}{2}t) \tag{5.20}$$

while the circular arc is given by

$$x = 2 + R\cos(\frac{\pi}{4} + \frac{\pi t}{2}) \tag{5.21}$$

$$y = 1 + R\sin(\frac{\pi}{4} + \frac{\pi t}{2}) \tag{5.22}$$

where $0 \leq t \leq 1$. We can see that the SPP is very close to the circular line, but has continuous curvature as shown in Figure 5.8.

Let $(x_1, y_1)$ be the start point of the arc turn and $(x_2, y_2)$ the end point. Then the SPP trajectory between these two points is defined as follows:

$$\mathbf{q} = [2, x_1, y_1, x_2, y_2, \frac{1}{\kappa}, v_2, \epsilon, \eta]^\top \tag{5.23}$$

where the driving speed $v_2$ and the maximum position and orientation deviation $(\epsilon, \eta)$ are defined by the global and local path planners.The type of SPP segment is defined as $c = 2$.

The reference steering angle and drive speed for a SPP segment are

$$\alpha_r = tan^{-1}(B\kappa), \quad \omega_r = v_2/(R_f \cos\alpha_r) \tag{5.24}$$

where $R_f$ is the radius of the front wheel of the robot.

*Cubic Parametric Spline*

In some cases, a transition associated with an arc turn which goes around a sharp corner may cut the corner on the inside. Therefore, it is desired that a smooth curve should be generated to deviate from a circular arc towards the intersecting point of both lines it joins. For this reason, a cubic parametric spline (CPS) can be defined to replace the SPP. It has the general expression:

$$x = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{5.25}$$

$$y = b_0 + b_1 t + b_2 t^2 + b_3 t^3 \tag{5.26}$$

Let $(x_1, y_1)$ be the start point of the arc and $(x_2, y_2)$ the end point (coordinates of points $a$ and $b$ in Figure 5.6). The slopes of both lines $l_1$ and $l_2$ are $k_1$ and $k_2$ respectively. For the desired continuous-curvature replacement for the circular arc, the position, slope, and curvature constraints on the CPS are:

$$\begin{aligned} x = x_1, \quad y = y_1, \quad \dot{y} = k_1, \quad \kappa = 0, \quad \text{at } t = 0 \\ x = x_2, \quad y = y_2, \quad \dot{y} = k_2, \quad \kappa = 0, \quad \text{at } t = 1 \end{aligned} \tag{5.27}$$

where $\kappa$ is the curvature as follows:

$$\kappa = \frac{|\dot{x}\,\ddot{y} - \ddot{x}\,\dot{y}|}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \tag{5.28}$$

The 8 coefficients in Equations 5.25 and 5.26 are chosen to satisfy the constraints listed above. Then, the cubic parametric spline for the arc turns can be expressed in terms of parameter $t$ as

$$x = x_1 + 3\xi t - 3\xi t^2 + (x_2 - x_1)t^3 \tag{5.29}$$

$$y = y_1 + 3\xi k_1 t - 3\xi k_1 t^2 + (y_2 - y_1)t^3 \tag{5.30}$$

where $\xi = [(x_2 - x_1)k_2 - (y_2 - y_1)]/(k_2 - k_1)$.

For instance, Figure 5.9 shows a CPS segment and a circular segment for the arc turn. Compared with the SPP in Figure 5.7, we can see that the CPS is very close to the intersection point of the two perpendicular lines given by the equations:

$$y = x, \quad y = -x + 4 \tag{5.31}$$

Using algorithm 2, we can easily obtain the start and end points of the arc turn:

$$x_1 = 1.5, \quad y_1 = 1.5, \quad x_2 = 2.5, \quad y_2 = 1.5 \tag{5.32}$$

Then CPS is closer to the intersection point of the two lines, given by the equations

$$x = 1.5 + 1.5t - 1.5t^2 + t^3 \tag{5.33}$$

$$y = 1.5 + 1.5t - 1.5t^2 \tag{5.34}$$

where $0 \leq t \leq 1$.

Figure 5.10 shows its continuous curvature:

$$\kappa = \frac{8t(1-t)}{3[(1-2t+2t^2)^2 + (1-2t)^2]^{3/2}} \tag{5.35}$$

where $(0 \leq t \leq 1)$.

Finally, let $(x_1, y_1)$ be the start point of the arc turn and $(x_2, y_2)$ the end point. Then the cubic parametric spline trajectory between these two points is defined as follows:

$$\mathbf{q} = [3, x_1, y_1, x_2, y_2, \frac{1}{\kappa}, v_3, \epsilon, \eta]^\mathsf{T} \tag{5.36}$$

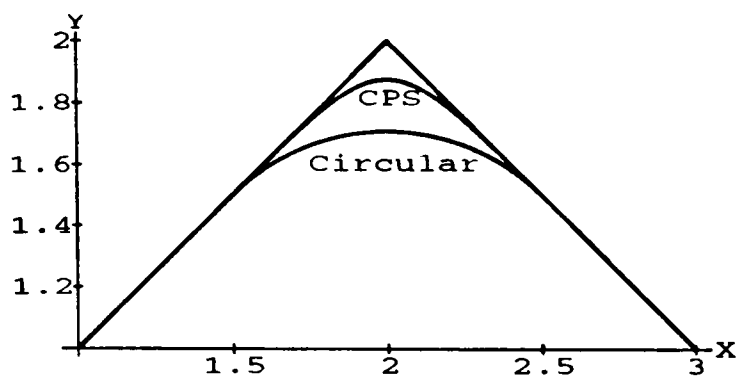



Figure 5.9: A CPS and a circular arc

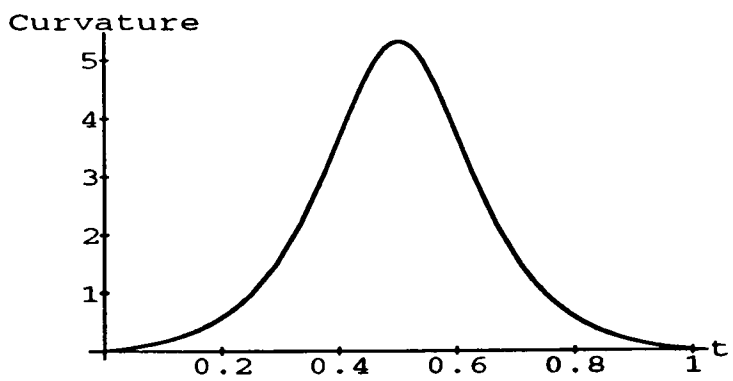



Figure 5.10: The CPS curvature

where the driving speed $v_3$ and the maximum position and orientation errors $(\epsilon, \eta)$ are defined by the global and local path planners. The type of CPS segments is defined to be $c = 3$.

The reference steering angle and drive speed for a CPS segment are

$$\alpha_r = tan^{-1}(B\kappa), \quad \omega_r = v_3/(R_f \cos \alpha_r) \tag{5.37}$$

where $R_f$ is the radius of the front wheel of the robot.

**Lane Change**

In a dynamic environment, the robot may encounter unexpected obstacles which partially block the path that the robot moves along. However, when there is enough space to sidestep, a lane change trajectory is demanded, as shown in Figure 5.11. Recall that three intermediate points $p_1$, $p_2$, and $p_3$ are chosen by the local path planner to avoid the unexpected obstacle (using observations from available sensors).

The lane change curve needed to achieve this typically consists of an arc-line-arc sequence. It can be constructed using two circular arc segments and one line segment, but again the problem of curvature discontinuities arises. Alternatively the path can be achieved by: (i) two smoothly joined cubic splines such as SPP or CPS, which is easily implemented; or (ii) a single quintic polynomial (SQP) segment proposed in [Nelson, 1989b]. SQP becomes attractive since it has a closed-form expression that guarantees precise matching of the two-point boundary conditions. However, as Brady [1982] already indicated, the disadvantage of a polynomial trajectory is its numerical accuracy

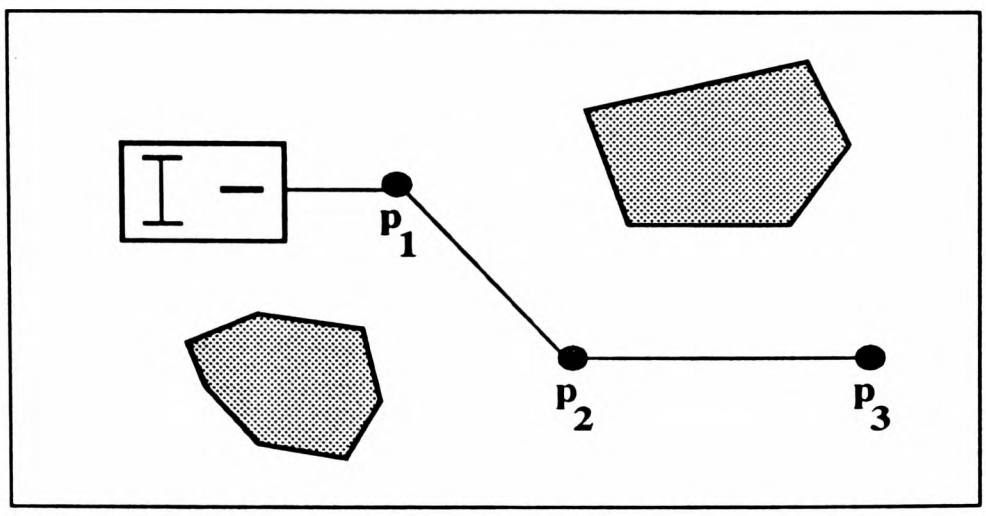



Figure 5.11: A lane change trajectory to avoid obstacle

to which a polynomial can be computed decreases as the degree of the polynomial increases. Any high order polynominal is poorly conditioned and has unfortunate tendency to overshoot and wander. That is why splines were invented and became popular.

Now let us see how the SQP generates the complete maneuver of lane change with a single segment. SQP has only three nonzero coefficients and can be written in the form

$$y(x) = y_t[10(\frac{x}{x_t})^3 - 15(\frac{x}{x_t})^4 + 6(\frac{x}{x_t})^5] \tag{5.38}$$

subject to constraints of the position, slope, and curvature of both terminal points:

$$\begin{aligned} y = 0, \quad \dot{y} = 0, \quad \kappa = 0, \quad \text{at } x = 0 \\ y = y_t, \quad \dot{y} = 0, \quad \kappa = 0, \quad \text{at } x = x_t \end{aligned} \tag{5.39}$$

where $\kappa$ is the curvature of the SQP, given by

$$\kappa = \frac{|\ddot{y}2|}{(1+\dot{y}^2)^{3/2}} \tag{5.40}$$

Assume for example that $x_t = 5$ and $y_t = 1$. As shown in Figure 5.12, the SQP curve becomes

$$y(x) = 10(\frac{x}{5})^3 - 15(\frac{x}{5})^4 + 6(\frac{x}{5})^5 \tag{5.41}$$

Figure 5.13 shows that the SQP provides a continuous curvature

$$\kappa = \frac{12(x_\mu - 3x_\mu^2 + 2x_\mu^3)}{5[1 + 36(x_\mu^2 - 2x_\mu^3 + x_\mu^4)^2]^{3/2}} \tag{5.42}$$

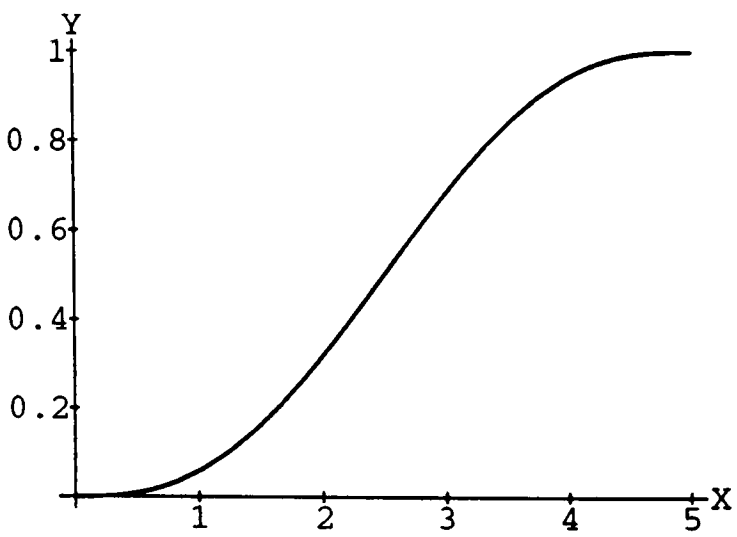



Figure 5.12: A SQP segment

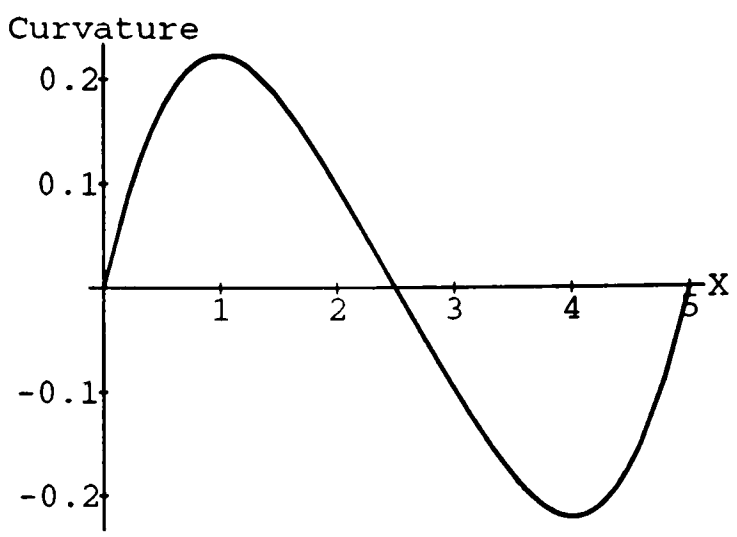



Figure 5.13: Its curvature function

where $x_\mu = x/x_t$. Note that the maximum curvature and curvature-rate for the function shown in Figure 5.13 increase directly with the lane-change slope ratio $y_e/x_e$. This ratio must be chosen sufficiently low so that the resulting continuous steering function does not violate the peak steering and steering-rate constraints for a particular mobile robot design.

Now let $(x_1, y_1)$ be the start point of the SQP and $(x_2, y_2)$ the end point. Then a lane change trajectory between these two points is defined as follows:

$$\mathbf{q} = [4, x_1, y_1, x_2, y_2, \frac{1}{\kappa}, v_4, \epsilon, , \eta]^\top \tag{5.43}$$

where the driving speed $v_4$ and the maximum position and orientation deviations $(\epsilon, \eta)$ are defined by global and local path planner. The type of lane change segments is defined to be $c = 4$.

The reference steering angle and drive speed for a lane change segment are

$$\alpha_r = tan^{-1}(B\kappa), \quad \omega_r = v_4/(R_f \cos\alpha_r) \tag{5.44}$$

where $R_f$ is the radius of the front wheel of the robot.

### 5.4.2 Docking at Given Configuration

Picking up a pallet with a forklift or docking at a workstation requires the mobile robot to be moved to a given position, with a given orientation. Errors in the position and orientation of the pallet to be picked up may necessitate a maneuver. For instance, Figure 5.14 shows the mobile robot

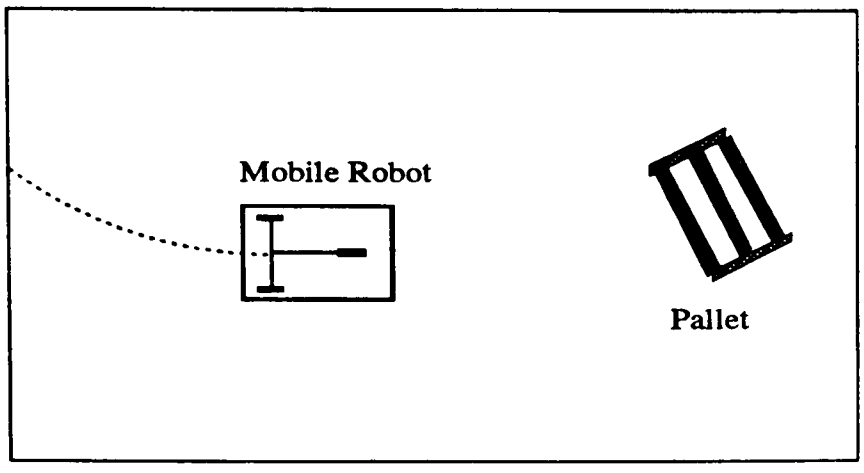



Figure 5.14: Picking up a pallet

arriving at its destination planned by the global path planner to pick up a pallet which is inaccurately positioned. A repositioning is obviously required.

We propose a heuristic algorithm to enable the robot to carry out the necessary maneuvers. Here, we focus on how the robot may reposition itself within a constrained environment. The same principle can be applied to docking maneuvers. Let $\mathbf{x}_c = [x_c, y_c, \theta_c]^\top$ be the current state vector of the robot, and $\mathbf{x}_d = [x_d, y_d, \theta_d]^\top$ the desired docking state vector. Let $L$ denote the total vehicle length and $S$ be defined as a safe gap between the forklift and the pallet edge. Assume that the position and orientation of the pallet can be observed by on-board sensors, and that a pivot turn of the robot is allowed at a reasonably low speed. The algorithm we propose involves two operations: one translation and one rotation, divided into four steps (refer to Figure 5.15):

**Algorithm 3** *Docking Maneuver*

1. *Draw a straight line $L_s$ along the pallet edge to be faced by the forklift, and a perpendicular line $L_d$ to $L_s$ intersecting it at the midpoint of the pallet edge.*
2. *Fix the desired final position $G(x_d, y_d)$ which is located on the line $L_d$ at a distance $D_d = L + S$ to the pallet edge, and then choose a turn point $T$ also on the line $L_d$ to satisfy the conditions: $D_t \geq D_d$ and $D_s \geq L/2$. If $T$ is found,* **then** *continue;* **otherwise**, *announce failure.*
3. *Draw a straight dashed line $L_0$ along the $X$ axis of the robot local frame $\mathcal{F}_1$ to intersect with*

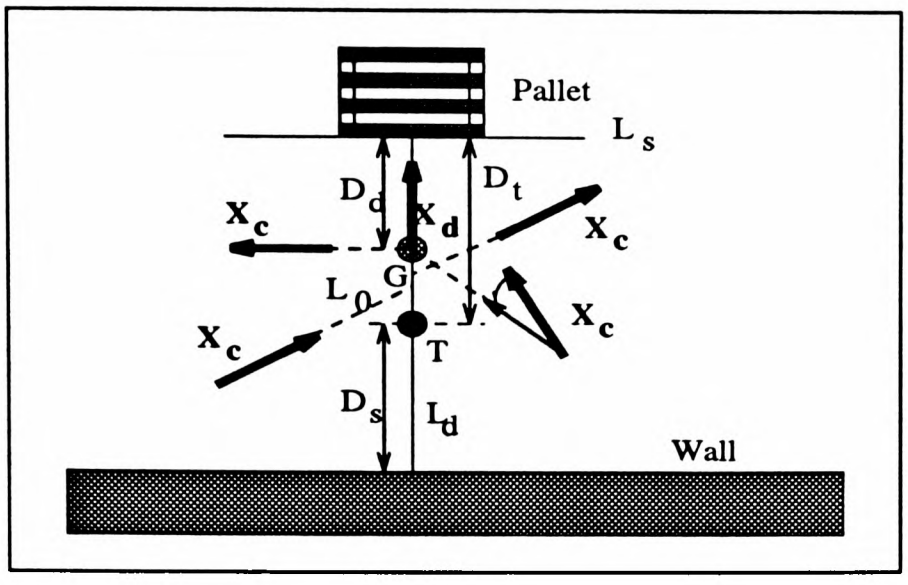



Figure 5.15: Repositioning to pick up a pallet

*the line* $\mathbf{x}_d$*, shown in Figure 5.15.* **If** *the intersecting point* $x_t$ *is between* $G$ *and* $T$*,* **then** *do nothing;* **otherwise**, *do pivot turn until* $x_t$ *is between* $G$ *and* $T$*.*

4. *Move the robot along the straight dashed line* $L_0$*, forward or backward, until it reaches* $x_t$ *and then perform a pivot turn to bring it collinear with the line* $L_d$*.* **If** $x_t$ *is at the final position* $(x_d, y_d)$ *within tolerance error,* **then** *do nothing;* **otherwise**, *the robot moves forward until it reaches* $G$*. Announce success.*

The reference steering angle and drive speed for a pivot turn are

$$\alpha_r = \pi/2, \quad \omega_r = v_p/R_f \tag{5.45}$$

where $R_f$ is the radius of the front wheel of the robot, and $v_p$ is the maximum speed for a pivot turn. The sign of $v_p$ represents the direction of turning.

## 5.5 Trajectory Tracking

In the last section, we considered how to generate a continuous-curvature trajectory, taking into account the constraints from the road boundary and the robot's kinematics. However, the generated trajectory is only useful when the low level motion controller can ensure good tracking and local recovery from deviations caused by unmodeled perturbations. This automatic recovery feature is

essential to industrial applications. In this section, we first describe a motion control architecture for tracking the reference trajectory. Secondly, the problems in the original motion controller is described. Thirdly, an optimal tracking strategy is proposed to minimise a quadratic performance index in the position and orientation errors.

### 5.5.1 Control Strategy

We adopt a feedback control principle to design a tracking control system for a mobile robot to track the desired trajectory. The following requirements are taken into consideration in the design:

- The robot should follow the drive path smoothly. There should be no overshoots in the path when it turns from one segment to another. When it follows the path, there should be no oscillations.
- The drive speed should be increased when the steer angle is close to zero, and reduced when the vehicle turns so that any risk of slipping under normal driving conditions is minimised.
- The change of the control inputs from one sampling instant to another should lie within certain limits since larger changes may cause slippage of the driving wheel and may also result in a poor estimate of the robot's position and heading.
- The reference path specification and current position estimation (combining dead reckoning and external location sensor) are given separately.

Figure 5.16 shows a block diagram of the proposed tracking control system. The input to the system is the reference trajectory $\mathbf{x}_r(k)$, including reference steering angle $\alpha_r(k)$ and velocity $\omega_r(k)$, which are provided by the trajectory planner. The output of the system is the current trajectory $\mathbf{x}(k)$ of the vehicle. The error state vector $\mathbf{x}_e(k) = \mathbf{x}_r(k) - \mathbf{x}_c(k)$ is input to the motion controller, where $\mathbf{x}_c(k)$ is the feedback state vector estimated by Kalman filter. The motion controller generates motion demands consisting of the steering angle $\alpha(k)$ and $\omega(k)$ to converge the error position vector $\mathbf{x}_e(k)$ to zero. These motion commands are used to generate the input control variables to

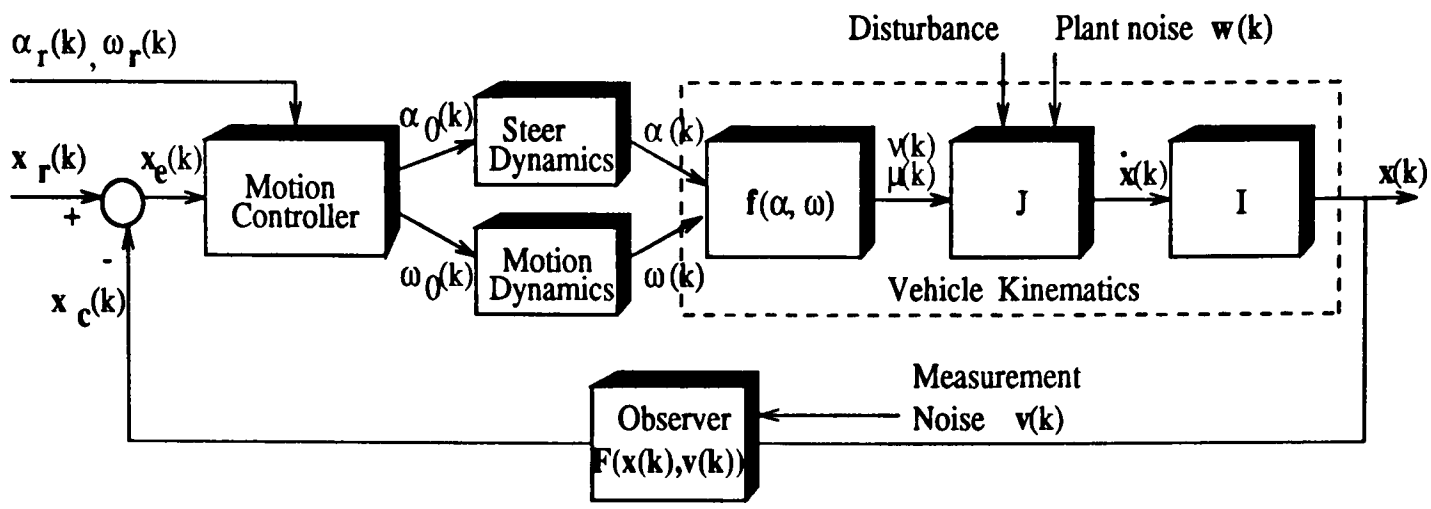



Figure 5.16: Block diagram of the feedback controller. Note that $I$ means integration.

the vehicle, $\mathcal{U}(k) = [v(k), \mu(k)]^{\top}$, according to Equation 5.2. The derivative of the current state vector, $\dot{\mathbf{x}}\ (k)$, is obtained by the Jacobian matrix in Equation 5.1. Finally, the current trajectory of the vehicle is integrated over the sample interval $T$.

In the feedback loop, $\mathbf{F}(\mathbf{x}(k), \mathbf{v}(k))$ describes an odometry estimate corrected by the Turtle's bar chart location system. Front wheel odometry is used in our mobile robot to estimate the trajectory of the front wheel every $10ms$ by the previous estimate plus the incremental changes of the position and orientation at each sampling duration [Steer, 1989]:

$$\begin{bmatrix} x_s(k) \\ y_s(k) \\ \theta_s(k) \end{bmatrix} = \begin{bmatrix} x_s(k-1) \\ y_s(k-1) \\ \theta_s(k-1) \end{bmatrix} + \begin{bmatrix} \Delta x_s(k) \\ \Delta y_s(k) \\ \Delta \theta_s(k) \end{bmatrix} \tag{5.46}$$

where the incremental changes is calculated by the following equation:

$$\begin{bmatrix} \Delta x_s(k) \\ \Delta y_s(k) \\ \Delta \theta_s(k) \end{bmatrix} = \begin{bmatrix} \Delta x & -\Delta y & 0 \\ \Delta y & \Delta x & 0 \\ 0 & 0 & R_f \omega(k)/B \end{bmatrix} \begin{bmatrix} \cos(\theta_s(k-1) + \alpha(k)) \\ \sin(\theta_s(k-1) + \alpha(k)) \\ \sin \alpha(k) \end{bmatrix} \tag{5.47}$$

and $\Delta x$ and $\Delta y$ are calculated by

$$\Delta x = R_s \sin \Delta \theta_s(k) \tag{5.48}$$

$$\Delta y = R_s(1 - \cos \Delta \theta_s(k)) \tag{5.49}$$

and $R_s$ is the steer radius, shown in Figure 5.1, defined by

$$R_s = B/\sin\alpha(k)$$

where $B$ is the wheelbase of the vehicle.

Finally, the trajectory of the guidepoint $0_1$ of the robot is obtained by

$$\begin{bmatrix} x_c(k) \\ y_c(k) \\ \theta_c(k) \end{bmatrix} = \begin{bmatrix} x_s(k) \\ y_s(k) \\ \theta_s(k) \end{bmatrix} - B \begin{bmatrix} \cos\theta_s(k) \\ \sin\theta_s(k) \\ 0 \end{bmatrix} \tag{5.50}$$

Errors in the odometry estimate may be caused by wheel slip and disturbances from the floor. Since these increase over time, an optimal recursive estimator, the Kalman filter, is used to make corrections as long as bar charts in known locations can be read. It is assumed that observations are corrupted by white Gaussian noise with zero mean and variance $\mathbf{Q}(k)$, i.e. $\mathbf{v}(k) \sim N(\mathbf{0}, \mathbf{Q}(k))$. More details about laser location system is given in Appendix A.

### 5.5.2 The problems in original motion controller

The problem of trajectory tracking is formulated so that the robot is given a reference trajectory to follow which is expressed as a set of the desired state vectors $\mathbf{x}_r(k) = [x_r, y_r, \theta_r]^\mathsf{T}$ at discrete time $k = 0, 1, ..., N$, as shown in Figure 5.17. The robot is allowed to deviate from the desired path by a certain tolerance $\chi$, which means that at any time $k$ the robot position must be at a distance at most $\epsilon$ from the point $(x_r(k), y_r(k))$ and its orientation must differ from $\theta_r(k)$ by at most $\eta$.

The robot we study has a three dimensional state vector $\mathbf{x}(k) = [x(k), y(k), \theta(k)]^\mathsf{T}$, but only a two dimensional control vector $\mathbf{u}(k) = [\alpha(k), \omega(k)]^\mathsf{T}$. To track a reference trajectory with minimum deviation in the position and orientation, the two control variables are not independent. Rather, they are constrained by the vehicle kinematic equations 5.1 and 5.2. Tracking the desired position and orientation at each path point depends on both control variables. However, the original motion controller in the Turtle mobile robot used two independent control loops with the two control

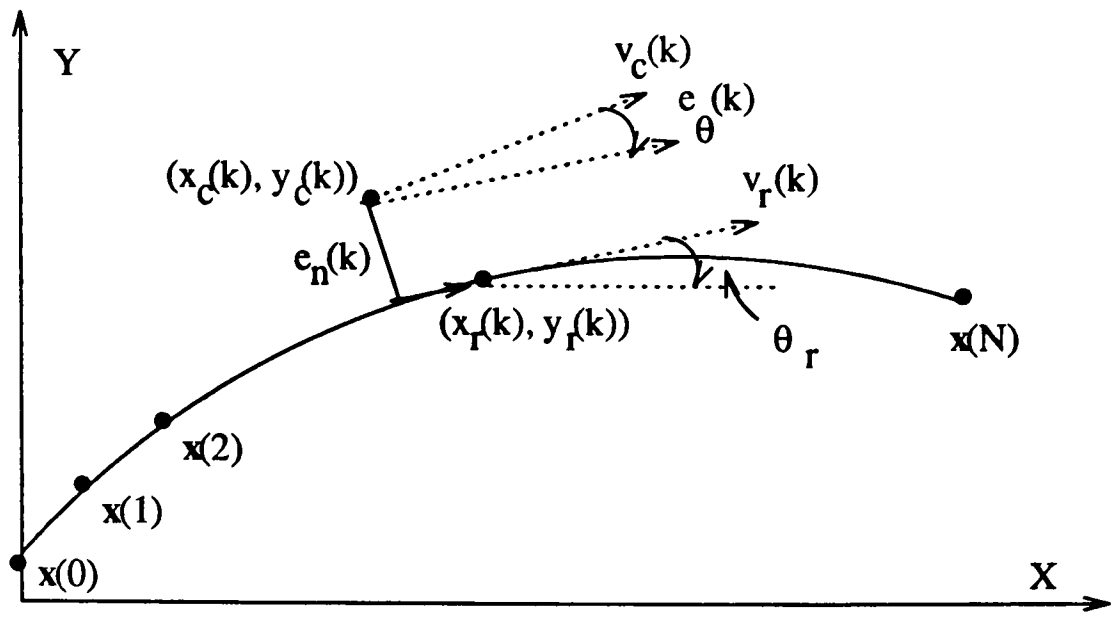



Figure 5.17: A reference trajectory and error components used for path tracking

variables fully decoupled for ease of implementation. It is based on two proportional controllers as follows

$$\alpha(k) = \alpha_r(k) + k_\theta e_\theta(k) - k_n e_n(k) \tag{5.51}$$

$$\omega(k) = \omega_r(k) + k_v e_v(k) \tag{5.52}$$

where $k_\theta$, $k_n$, and $k_v$ are proportional control parameters. $e_v$, $e_\theta$, and $e_n$ are velocity error, heading error, and normal position error respectively, as shown in Figure 5.17.

It is obvious that we have

$$\begin{aligned} e_\theta(k) &= \theta_r(k) - \theta_c(k) \\ e_n(k) &= -(x_r - x_c)\sin\theta_r + (y_r - y_c)\cos\theta_r \\ e_v(k) &= v_r(k) - v_c(k) \end{aligned}$$

Note that when the Turtle is following the desired reference path without error, the command steering and speed are equal to the reference values calculated by the trajectory planner, i.e. $\alpha(k) = \alpha_r(k)$ and $\omega(k) = \omega_r(k)$.

There are three problems with such a design. First, although the vehicle heading $\dot{\theta} = (R/B)\omega \sin\alpha$,

which has a discrete form as follows

$$\theta(k+1) = \theta(k) + (R/B)\omega(k)\sin\alpha(k), \tag{5.53}$$

depends on both control variables $(\alpha(k), \omega(k))$, only the steering control variable $\alpha(k)$ is used in the original design, see Equation 5.52, to reduce the heading error $e_\theta(k)$. That is, the effect of the velocity control variable $\omega(k)$ on the vehicle orientation is ignored. This makes it very hard to achieve high tracking performance. Secondly, to track the reference trajectory, the gains $k_\theta$, $k_n$ and $k_v$ are chosen to optimise the overall path tracking performance of the robot. To reduce tracking error, high gains may be assigned. However, a compromise between accuracy and stability is often necessary since high gains may cause the system unstable. Thirdly, a steady state error can not be eliminated by a simple proportional controller.

### 5.5.3 An optimal controller for tracking

To solve the problems existed in the original motion controller, we propose an optimal controller for trajectory tracking. Although the trajectory tracking problem we attack is stochastic in nature, we formulate it as a deterministic problem based on the *Certainty equivalence principle*. All random quantities in the cost functional replaced by their expected values which are estimated by a Kalman filter. Then a quadratic cost functional is chosen here to calculate the optimal control law since it is often a reasonable choice in many applications. Even in cases where the quadratic cost function is not entirely justified, it is still used since it leads to an elegant analytical solution that can often be implemented with relative ease in engineering applications [Bertsekas, 1976].

#### Single-step Prediction

To keep the state of the robot close to a reference trajectory with minimum variance at each discrete time $k$, let us formulate the optimal trajectory tracking problem as follows: given the transition equation of the robot system defined by

$$\mathbf{x}(k+1) = \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k), \mathbf{w}(k)] \tag{5.54}$$

choosing a allowable control, $\mathbf{u}(k)$, at each discrete time $k$ to lead

$$\min_{\mathbf{u}(k)} E\{I(\mathbf{x}(k), \mathbf{u}(k), \mathbf{w}(k))\} \tag{5.55}$$

subject to the constraints 5.54 and

$$\begin{aligned} |\omega(k)| \leq \omega_{\max}, \quad |\alpha(k)| \leq \alpha_{\max} \\ |\dot{\omega}(k)| \leq \dot{\omega}_{\max}, \quad |\dot{\alpha}(k)| \leq \dot{\alpha}_{\max} \end{aligned} \tag{5.56}$$

where $\dot{\omega}_{\max}$ is the limit of maximum rotation acceleration and $\dot{\alpha}_{\max}$ is the limit of maximum steering speed, which are determined to satisfy the condition of rolling without slipping.

As a natural choice for trajectory tracking, the following quadratic cost function is considered

$$I = E\{||\hat{\mathbf{x}}(k) - \mathbf{x}_r(k)||^2_{\mathbf{S}_x} + ||\mathbf{u}(k) - \mathbf{u}_r(k)||^2_{\mathbf{S}_u}\} \tag{5.57}$$

where

- $\mathbf{S}_x$ is a weighting matrix ($3 \times 3$ positive definite symmetric), reflecting the relative importance of maintaining position and orientation deviations at small values.
- $\mathbf{S}_u$ is a weighting matrix ($2 \times 2$ positive definite symmetric), penalising the control effort.
- $\hat{\mathbf{x}}(k) = [\hat{x}(k), \hat{y}(k), \hat{\theta}(k)]^\top$ is a predicted state vector.
- $\mathbf{u}_r(k) = [\alpha_r(k), \omega_r(k)]^\top$ is the desired control vector feedforward by the trajectory planner.
- $\mathbf{u}(k) = [\alpha(k), \omega(k)]^\top$ is the control vector to be decided.

This cost function reflects the desire to drive the next state $\mathbf{x}(k)$ of the robot close to the desired state $\mathbf{x}_r(k)$ generated by the trajectory planner without excessive expenditure of control effort. It also reflects that positive and negative values of the state errors and the control errors are of equal importance in terms of tracking the desired trajectory.

Note that the single-step prediction strategy means that the state $\hat{\mathbf{x}}$ is predicted using the following predictor model:

$$\hat{\mathbf{x}}(k+1) = \mathbf{x}_c(k) + \Delta\hat{\mathbf{x}}(k) \tag{5.58}$$

where $\mathbf{x}_c(k)$ is estimated by Kalman filter algorithm. $\Delta\hat{\mathbf{x}}(k)$ is the incremental change of the robot state made by the current control $\mathbf{u}(k)$.

This control strategy can be seen as the *Minimum Variance Control* in which one step ahead prediction of the robot state based on the system model is used. It provides a solution for applications in which the trajectory planner is able to generate a smooth continuous-curvature trajectory with a smooth velocity profile along it. It also requires that the system delay caused by the vehicle dynamics and sensor measurements is known *a priori* and is relatively small, compared with the cycle time of the controller. When the dead-time of the system is more than one sample-interval, the solution provided by this strategy at each sample-interval is no longer optimal. Moreover, the control variables $\omega(k)$ and $\alpha(k)$ selected are very sensitive to errors caused by unmodelled disturbances since the optimal decision on the control variables tends to minimise the deviations in the position and orientation within one sample-interval without any predictive capability for the system delay. As a result, the control inputs $\alpha(k)$ and $\omega(k)$ jump at each disturbance being measured. This in turn causes the robot to accelerate and decelerate unacceptably often, which rules out in industrial applications where the mobile robot carries a load.

**Multiple-step Prediction**

To overcome these drawbacks, we adopt a multiple-step prediction strategy to search for an optimal combination of the two control variables $(\alpha(k), \omega(k))$, taking into account the system rise time and possible delay. The basic idea is that the current control variables are chosen to minimise the cost function over several steps in the future so that the path tracking of the robot is smooth and stable. It is motivated by the Generalised Predictive Control (GPC) algorithm developed by Clarke and his colleagues [1987]. Although the GPC algorithm assumes a linear system model, it has been used to the nonlinear systems [Kotzev *et al.*, 1992].

The predictive control method is based on the minimisation of the quadratic cost function given

by:

$$\mathcal{I} = E\{\sum_{i=N_1}^{N_2} ||\hat{\mathbf{x}}_e(k+i)||^2_{\mathbf{S}_x} + \sum_{i=1}^{NU} ||\Delta\mathbf{u}(k+i-1)||^2_{\mathbf{S}_u}\} \tag{5.59}$$

where $\hat{\mathbf{x}}_e(k+i)$ is the predicted system error,

$$\hat{\mathbf{x}}_e(k+i) = \hat{\mathbf{x}}(k+i) - \mathbf{x}_r(k+i), \tag{5.60}$$

and $\Delta\mathbf{u}(k+i-1)$ is a set of control variables,

$$\Delta\mathbf{u}(k+i-1) = \begin{bmatrix} \Delta\alpha(k+i-1) \\ \Delta\omega(k+i-1) \end{bmatrix} = \begin{bmatrix} \alpha(k+i) - \alpha(k+i-1) \\ \omega(k+i) - \omega(k+i-1) \end{bmatrix} \tag{5.61}$$

and

- $N_1$ and $N_2$ are the minimum and maximum output costing horizon respectively.
- $NU$ is called the control horizon.
- $\mathbf{x}_r(k+i)$ is a set of trajectory reference points generated by the trajectory planner.
- $\hat{\mathbf{x}}(k+i)$ is the predicted state vector over a range up to the prespecified prediction horizon $N_2 - N_1$.
- The positive definite weighting matrices $\mathbf{S}_x$ and $\mathbf{S}_u$ make it possible to penalise more or less position and orientation errors and the control effort.

The cost function 5.59 is conditioned on data up to time $k$, assuming no future measurements are available. In a sense this implies that the set of control signals are applied open-loop in the sequel. At each sampling instant, an optimal control sequence is calculated, but only the first one is applied to the system. This process will be repeated at the next sampling instant to form a *receding horizon* optimisation procedure.

In general $N_2$ is chosen to encompass all the responses which are significantly affected by the current control. In practice, it is more typically set to approximate the rise-time of the system. $N_1$ is selected according to the dead-time of the robot system, and is often taken as 1. However, if we know *a priori* that the dead-time of the vehicle is at least $k_d$ sample-intervals, then $N_1$ can be

chosen as $k_d$ or more to minimise computations. The control horizon $NU$ is an important design parameter. For a simple, open-loop stable system though with possible dead-time like the mobile robots, setting $NU = 1$ gives reasonable performance [Clarke *et al.*, 1987]. If $NU = 1$ only one control increment $\Delta\mathbf{u}(k) = \mathbf{u}(k) - \mathbf{u}(k-1)$ is calculated, after which the controls $\mathbf{u}(k+i)$ are all taken to be equal to $\mathbf{u}(k)$, i.e.

$$\Delta\mathbf{u}(k+i-1) = 0 \quad (i > NU) \tag{5.62}$$

To generate a set of the predicted state vectors, we adopt a prediction model as follows:

$$\hat{\mathbf{x}}(k+1) = \mathbf{x}_c(k) + \Delta\hat{\mathbf{x}}(k-k_d) \tag{5.63}$$

where $k_d$ is the system dead-time which is either known *a priori* or estimated on-line.

From Equation 5.63, we have

$$\begin{aligned}
\hat{\mathbf{x}}(k+2) &= \hat{\mathbf{x}}(k+1) + \Delta\hat{\mathbf{x}}(k+1-k_d) \\
&= \mathbf{x}_c(k) + \Delta\hat{\mathbf{x}}(k-k_d) + \Delta\hat{\mathbf{x}}(k+1-k_d) \\
&= \mathbf{x}_c(k) + \sum_{j=0}^{1}\Delta\hat{\mathbf{x}}(k+j-k_d)
\end{aligned} \tag{5.64}$$

Obviously, a recursive predictor model can be adopted

$$\mathbf{x}(k+i) = \mathbf{x}_c(k) + \sum_{j=0}^{i-1}\Delta\hat{\mathbf{x}}(k+j-k_d) \tag{5.65}$$

By choosing $NU = 1$, and $k_d = 1$, the cost function 5.59 becomes

$$\mathcal{I} = E\{\sum_{i=N_1}^{N_2} ||\hat{\mathbf{x}}(k+i) - \mathbf{x}_r(k+i)||^2_{\mathbf{S}_x} + ||\Delta\mathbf{u}(k)||^2_{\mathbf{S}_u}\} \tag{5.66}$$

Using the predictor model 5.65, we further write Equation 5.66 as

$$\mathcal{I} = E\{\sum_{i=N_1}^{N_2} ||\mathbf{x}_c(k) + \sum_{j=0}^{i-1}\Delta\hat{\mathbf{x}}(k+j-1) - \mathbf{x}_r(k+i)||^2_{\mathbf{S}_x} + ||\Delta\mathbf{u}(k)||^2_{\mathbf{S}_u}\} \tag{5.67}$$

Now the optimal tracking with multiple-step prediction is performed by choosing $\mathrm{u}(k)$ at each sampling instant to lead

$$\min_{\mathbf{u}(k)} E\{\sum_{i=N_1}^{N_2} ||\mathbf{x}_c(k) + \sum_{j=0}^{i-1} \Delta\hat{\mathbf{x}}(k+j-1) - \mathbf{x}_r(k+i)||^2_{\mathbf{S}_x} + ||\Delta\mathbf{u}(k)||^2_{\mathbf{S}_u}\} \tag{5.68}$$

subject to the system equation constraints

$$\mathbf{x}(k+1) = \mathbf{f}[\mathbf{x}(k), \mathbf{u}(k), \mathbf{w}(k)] \tag{5.69}$$

and the control constraints

$$\begin{aligned} |\omega(k)| \leq \omega_{\max}, \quad |\alpha(k)| \leq \alpha_{\max} \\ |\dot{\omega}(k)| \leq \dot{\omega}_{\max}, \quad |\dot{\alpha}(k)| \leq \dot{\alpha}_{\max} \end{aligned} \tag{5.70}$$

It is obvious that the objective of the proposed control strategy is to drive the future robot state $\hat{\mathbf{x}}(k+i)$ close to the reference trajectory set-point $\mathbf{x}_r(k+i)$ in a predictive manner. This is done using a receding-horizon approach at each sample-instant $k$. Due to the minimisation of a multi-step cost function, the controller is able to achieve smooth tracking performance under any unmodelled disturbances.

### 5.5.4 Numerical solution

Numerical solution of the optimal tracking problem above is a complex task which raises a number of questions. These include:

1. Determination of a good initial approximate solution that is feasible.
2. Choice of the descent algorithm.
3. Treatment of the inequality constraints and nonlinear system equation.
4. Compromise of the optimal solution and the real-time performance.

As expected in most nonlinear optimisation problems, the initial guess determines which local minimum the algorithm converges to and how quickly. We propose a heuristic recursive search algorithm for real time implementation based on the following definitions.

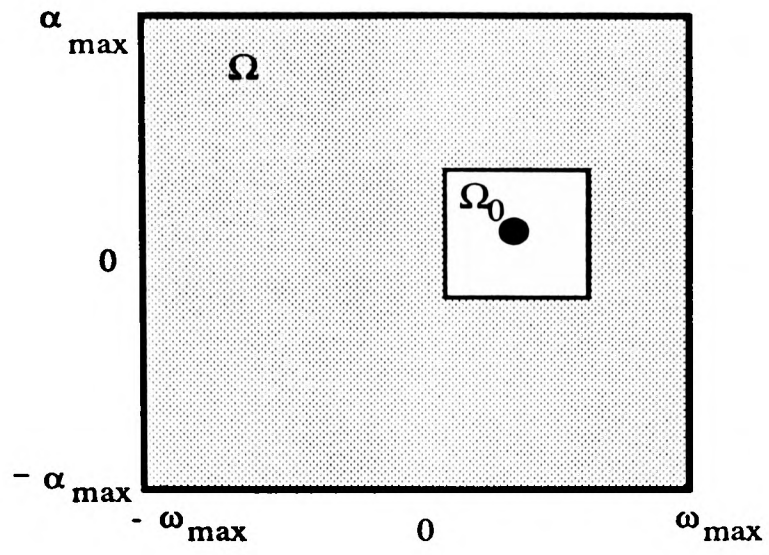



Figure 5.18: Search space for control variables $\omega$ and $\alpha$

**Definition 8** *Let $\Omega$ be a two-dimensional search space for admissible control inputs, in which $\alpha_i$ and $\omega_j$ denote the set of possible discrete values for the continuous control variables $\alpha$ and $\omega$ respectively. Assume that the values of $\alpha_i$ and $\omega_j$ are equally spaced with $\Delta\omega = \omega_{max}/I_1$ and $\Delta\alpha = 2\alpha_{max}/J_1$ in $\Omega$, where $I_1$ and $J_1$ are the numbers of $\omega_i$ and $\alpha_j$ contained in $\Omega$. Then the control variables $\alpha_i$ and $\omega_j$ to be decided are subject to the constraints:*

$$\begin{aligned} |\omega_i| &\leq \omega_{\max}, \quad i = 1, 2, ..., I_1 \\ |\alpha_j| &\leq \alpha_{\max}, \quad j = 1, 2, ..., J_1 \end{aligned} \tag{5.71}$$

To conduct an exhaustive search in $\Omega$ is time consuming. However, by taking into account $\dot{\omega}_{\max}$, $\dot{\alpha}_{\max}$, and the locomotion control cycle time $T = 80ms$, the search space can be greatly reduced. For this reason, a search window is defined.

**Definition 9** *Let $\Omega_0$ be a two-dimensional search window constrained in the search space $\Omega$, with the dimensions $I_0 \times J_0$, where $I_0 = \dot{\omega}_{\max} T$ and $J_0 = \dot{\alpha}_{\max} T$. The center of the search window $\Omega_0$ is located at reference control variables $\mathbf{u}(k) = [\alpha_r(k), \omega_r(k)]$ which is generated by the trajectory planning algorithm. The candidate control values $\alpha_i$ and $\omega_j$ are defined as follows, subject to the constraint 5.71*

$$\begin{aligned} \omega_i &= \omega_r \pm i \times \Delta\omega \\ \alpha_j &= \alpha_r \pm j \times \Delta\alpha \end{aligned} \tag{5.72}$$

where $i = 1, 2, ..., (I_0/\Delta\omega)$ and $j = 1, 2, ..., (J_0/\Delta\alpha)$.

Figure 5.18 describes the search space $\Omega$ and the search window $\Omega_0$ defined above. To find the control vector $(\alpha_i, \omega_j)$ in $\Omega_0$ that minimises the cost function $I$, the heuristic search algorithm is

**Algorithm 4** *Heuristic Search of Suboptimal Solution*

- Step 1*: initialisation.*

  1. *Maximum search processing time, $T_{sm}$, limited by control cycle time.*
  2. *Maximum tolerance, $I_t$, of the position and heading errors.*
  3. *Set the search step lengths $\Delta\omega = 2\omega_{\max}/I_1$ and $\Delta\alpha = 2\alpha_{\max}/J_1$.*
  4. *Set $i = -1$, $j = -1$, $h = 1$, $U_{ij} = [\alpha_r + h \times i \times \Delta\alpha, \omega_r + h \times j \times \Delta\omega]^\top$*
  5. *Set search time $t_s = 0$ and the objective function $I_{t1} > I_t$.*

- Step 2*: Termination check.*

  1. *If $t_s > T_{sm}$ or $I_{ij} < I_0$, then output the control vector $\mathbf{u} = U1$ and stop.*
  2. *Otherwise, continue.*

- Step 3*: Continuation.*

  1. *If $h \leq 1$, then $i = i + 1$, $j = j + 1$, otherwise $h = -1$.*
  2. *$U_{ij} = [\alpha_r + h \times i \times \Delta\alpha, \omega_r + h \times j \times \Delta\omega]^\top$*
  3. *Calculate $I_{ij}(U_{ij})$ using Equation 5.57 fo r single-step prediction or Equation 5.67 for multiple-step prediction.*
  4. *If $I_{ij} < I_{t1}$, then $I_{t1} = I_{ij}$ and $U1 = U_{ij}$.*
  5. *$h = h + 2$, goto* Step 2.

It should be noticed that the heuristic search algorithm we propose aims to find a suboptimal solution with fast processing speed. The sensitivity of the algorithm with respect to the discretisation of $\alpha$ and $\omega$ as well as motion inaccuracies, measurement errors, and further disturbances is expected to be low. For instance, if the robot rotation speed $\omega$ which has been ordered by the control strategy

at time $k$, is not performed accurately, then, in the next time step $k+1$, the new measurement $\mathbf{x}_c(k+1)$ will take account of the real robot position so that the control strategy will respond adequately and so forth. In other words, the feedback law avoids an accumulation of tracking errors on the desired robot trajectory.

### 5.5.5 Analytical solution

Analytical solution of the optimal tracking strategy described above is more attractive to the real-time application. Therefore, the analytical solution is further derivated in this section.

Since the kinematic equation of the robot is nonlinear, we use intermediate control variables $\Delta\mathbf{x}(k)$ to replace the true control variables $\Delta\mathbf{u}(k)$ in the objective function 5.59 which is to be minimised. Then, the corresponding objective function is:

$$\mathcal{I} = E\{\sum_{i=N_1}^{N_2} ||\hat{\mathbf{x}}(k+i) - \mathbf{x}_r(k+i)||^2_{\mathbf{S}_x} + \sum_{i=1}^{NU} ||\Delta\mathbf{x}(k+i-1)||^2_{\mathbf{S}_u}\} \tag{5.73}$$

This linearised objective function is then used to obtain the analytical solution of the optimal tracking. Recall that the key assumptions of GPC for minimising this objective function are

$$\begin{gathered} \Delta\hat{\mathbf{x}}(k+i-1) = 0 \\ N_1 \geq 1 \\ NU < N_2 \end{gathered} \tag{5.74}$$

where $i > NU$. Here, we chose:

$$N_1 = 1, \quad N_2 = 3, \quad NU = 1, \quad K_d = 1 \tag{5.75}$$

Using the recursive predictor 5.65, the objective function 5.73 becomes

$$\begin{bmatrix} \hat{\mathbf{x}}(k+1) \\ \hat{\mathbf{x}}(k+2) \\ \hat{\mathbf{x}}(k+3) \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix} \mathbf{x}_c(k) + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix} \Delta\hat{\mathbf{x}}(k) + \begin{bmatrix} \mathbf{I} \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix} \Delta\mathbf{x}(k-1) \tag{5.76}$$

It can be rewritten as

$$\hat{\mathbf{X}}(k) = \mathbf{A}\mathbf{x}_c(k) + \mathbf{B}\Delta\hat{\mathbf{x}}(k) + \mathbf{C}\Delta\mathbf{x}(k-1) \tag{5.77}$$

Thus the objective function 5.73 becomes

$$\mathcal{I} = ||\hat{\mathbf{X}}(k) - \mathbf{X}_r(k)||^2_{\mathcal{S}_x} + ||\Delta\hat{\mathbf{x}}(k)||^2_{\mathbf{S}_u} \tag{5.78}$$

where:

$$\mathbf{X}_r(k) = [\mathbf{x}_r(k+1), \mathbf{x}_r(k+2), \mathbf{x}_r(k+3)]^\top \tag{5.79}$$

$$\mathcal{S}_x = diag[\mathbf{S}_x, \mathbf{S}_x, \mathbf{S}_x] \tag{5.80}$$

To obtain optimal control law, we take

$$\frac{\partial \mathcal{I}}{\partial \Delta\hat{\mathbf{x}}(k)} = 0 \tag{5.81}$$

that is

$$[\hat{\mathbf{X}}(k) - \mathbf{X}_r(k)]^\top \mathcal{S}_x \frac{\partial \hat{\mathbf{X}}(k)}{\partial \Delta\hat{\mathbf{x}}(k)} + [\Delta\hat{\mathbf{x}}(k)]^\top \mathbf{S}_u = 0 \tag{5.82}$$

From equation 5.77, we have

$$\frac{\partial \hat{\mathbf{X}}(k)}{\partial \Delta\hat{\mathbf{x}}(k)} = \mathbf{B} \tag{5.83}$$

therefore, equation 5.82 becomes

$$\mathbf{B}^\top \mathcal{S}_x[\hat{\mathbf{X}}(k) - \mathbf{X}_r(k)] + \mathbf{S}_u^\top \Delta\hat{\mathbf{x}}(k) = 0 \tag{5.84}$$

$$\mathbf{B}^\top \mathcal{S}_x[\mathbf{A}\mathbf{x}_c(k) + \mathbf{B}\Delta\hat{\mathbf{x}}(k) + \mathbf{C}\Delta\mathbf{x}(k-1) - \mathbf{X}_r(k)] + \mathbf{S}_u\Delta\hat{\mathbf{x}}(k) = 0 \tag{5.85}$$

$$[\mathbf{B}^\top \mathcal{S}_x\mathbf{B} + \mathbf{S}_u]\Delta\hat{\mathbf{x}}(k) = \mathbf{B}^\top \mathcal{S}_x[\mathbf{X}_r(k) - (\mathbf{A}\mathbf{x}_c(k) + \mathbf{C}\Delta\mathbf{x}(k-1))] \tag{5.86}$$

Finally, we obtain the intermediate control increment as follows

$$\Delta\hat{\mathbf{x}}(k) = [\mathbf{B}^\top \mathcal{S}_x\mathbf{B} + \mathbf{S}_u]^\top \mathbf{B}^\top \mathcal{S}_x[\hat{\mathbf{X}}_r(k) - (\mathbf{A}\mathbf{x}_c(k) + \mathbf{C}\Delta\mathbf{x}(k-1))] \tag{5.87}$$

where the matrix $[\mathbf{B}^\top \mathcal{S}_x\mathbf{B} + \mathbf{S}_u]^\top \mathbf{B}^\top \mathcal{S}_x$ is the gain which can be adjusted in the applications.

The optimal control law $\mathbf{u}$ can be then obtained by calculating the inverse kinematics of the robot which can be derivated from the Equations 5.1 or 5.46:

$$\mathbf{u}(k) = \mathbf{h}(\Delta\hat{\mathbf{x}}(k)) \tag{5.88}$$

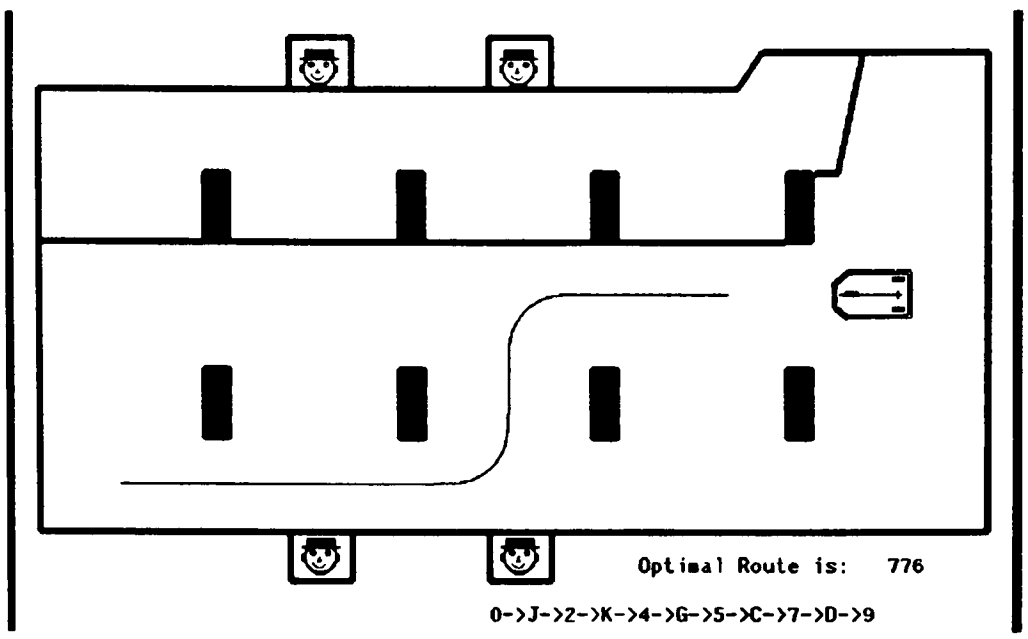



Figure 5.19: Simulation environment based on Sunview graphics

Finally, the optimal control vector is given by

$$\hat{\mathbf{u}}(k) = \mathbf{u}(k-1) + \Delta\hat{\mathbf{u}}(k) \tag{5.89}$$

## 5.6 Simulation

The proposed trajectory planning and control strategy has been simulated in a Sunview graphics environment, as shown in Figure 5.19, to evaluate its performance. The parameters used in the simulation are given in Table 5.1, where $T = 80ms$ is the cycle time for locomotion control.

Figure 5.20 shows the system structure which has been used in the simulation. Recall that the motor control systems for steering and motion are closed-loop systems based on conventional PI controllers. The parameters of both controllers have been chosen to satisfy the required stability and response speed. Considering that the robot moves in a relative slow speed (up to 2m/s), the dynamics of the robot is not very crucial for the analysis of the trajectory tracking strategy. For simplicity, it is assumed that both systems are first order with time constants $k_\alpha$ for steering and $k_\omega$ for motion, dependent on the vehicle weight and load. Initially, $k_\alpha = 0.05s$ $k_\omega = 0.05s$ have been chosen in the simulation. A random disturbance is considered to test the robustness of the tracking algorithm. Similarly, the weight matrices $\mathbf{S}_x$ and $\mathbf{S}_u$ in Equation 5.57 have been chosen as follows,

Table 5.1: Simulation Input Data

| Description | Symbol | Preset Value |
|---|---|---|
| AGV model size(pixel) | $L,H,B,R_f,E$ | 60,32,40,9,50 |
| Optimal linear speed | $\mu_1$(pixel/T) | 6 |
| Optimal SPP speed | $\mu_2$(pixel/T) | 4 |
| Optimal CPS speed | $\mu_3$(pixel/T) | 4 |
| Maximum control input | $\omega$(pixel/T),$\alpha$(radian) | [0,10], [-1.57,1.57] |
| Maximum acceleration | $\dot{\omega}$ (pixel/$T^2$) | 5.0 |
| Maximum steering speed | $\dot{\alpha}$(radian/T) | 0.05 |
| Initial position & Orientation | $X_0$(pixel), $Y_0$(pixel), $\theta_0$(radian) | 708, 298, $\pi$ |
| Goal position & Orientation | $X_g$(pixel), $Y_g$(pixel), $\theta_g$(radian) | 84, 450, $\pi$ |

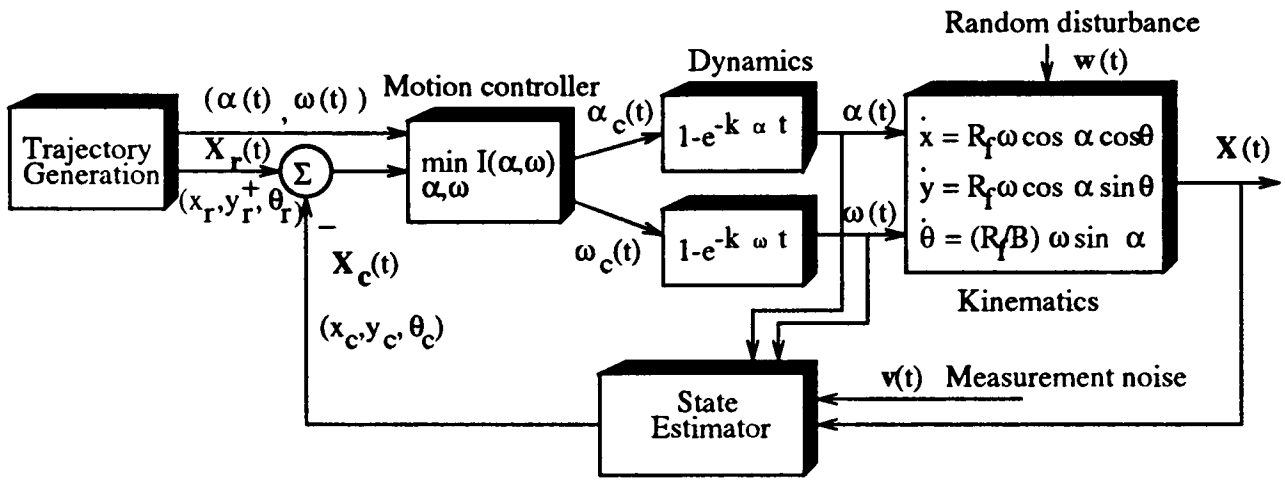



Figure 5.20: System structure for simulation

adjusted accordingly in the simulation and the real applications.

$$\mathbf{S}_x = diag\,[2,2,20], \quad \mathbf{S}_u = diag\,[1,2] \tag{5.90}$$

### 5.6.1 One-step Prediction

Based on the one-step prediction strategy, the simulation of trajectory tracking has been conducted by line segment, SPP and CPS curves. We first examine the performance of the proposed strategy by tracking a smooth trajectory consisting of line segment and SPP curves. Figures 5.21 and 5.22 present the results from the simulator. Graph 1 in Figure 5.21 shows the reference rotation speed $\omega_r$ and steering angle $\alpha_r$ as well as the orientation $\theta_r$ generated by the trajectory planner. In contrast,

Graph 2 in Figure 5.21 gives the control variables $(\alpha_c, \omega_c)$ output by the optimal motion controller in terms of minimising the cost function, as well as the orientation tracking results. Graph 1 of Figure 5.22 shows the reference trajectory (solid line) and the trajectory tracking results (marked as St. Andrew's crosses). Clearly the tracking follows the reference closely even though a random disturbance is included. The envelope of the simulated mobile robot is shown in Graph 2 of Figure 5.22.

Figure 5.23 gives the cost function surface recorded in one step search for suboptimal control variables $(\alpha_c, \omega_c)$. There is only one global minimum and no discontinuity or local minimum exists. Therefore, the suboptimal solution is easily calculated by the proposed heuristic search method within the specified time constraint.

Figure 5.24 shows the minimum values of the cost function $I$ obtained by the recursive search algorithm at each time step. Clearly, the errors in both the position and orientation of the tracking trajectory are very small, satisfying the predetermined error tolerance.

Similarly, Figures 5.25 and 5.26 present the tracking results for the reference trajectory consisting of the line segment and CPS curves generated by the trajectory planner. The results give good convergence between the planned trajectory and the actual trajectory travelled by the robot.

### 5.6.2 Multiple-step Prediction

In Figures 5.21 and 5.25, it can be clearly seen that the simulated vehicle speed $\omega$ jumps at some time steps. The main reason is that in the one-step prediction, the error between the current state $\mathbf{x}_c(k)$ and the future state $\mathbf{x}_r(k)$ is intended to be reduced in one sample interval. In many industrial applications, the jump of the vehicle speed may damage the components or goods that are carried by the robot. Moreover, a sudden change in the vehicle speed may lead to instability.

Consider now the simulation results resulting from the multiple-step prediction strategy. Choos-

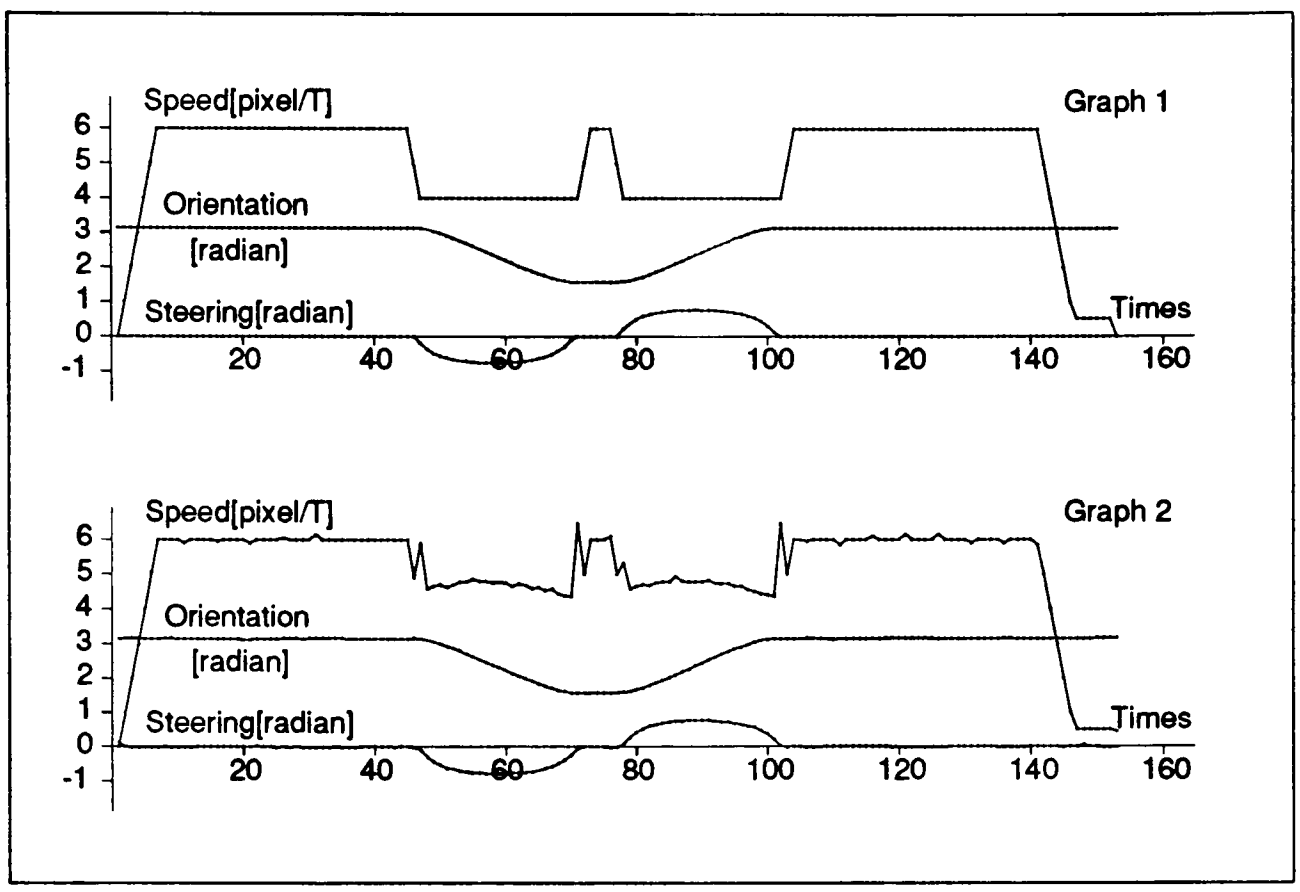



Figure 5.21: Optimal tracking of Line Segments and SPP with single-step prediction. Note that Graph 1 gives reference inputs and Graph 2 is tracking results

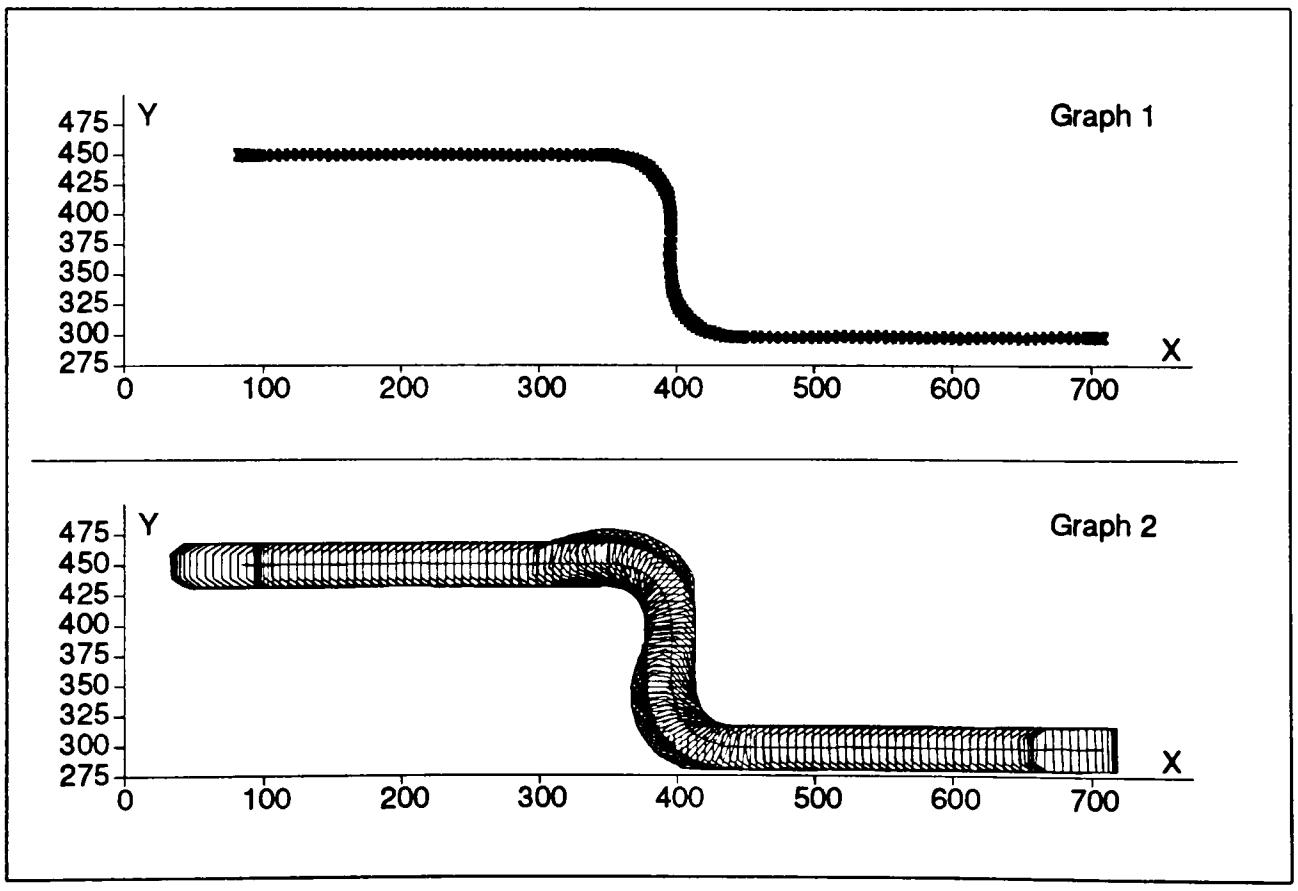



Figure 5.22: Optimal tracking of Line Segments and SPP with single-step prediction. Note that Graph 1 presents a reference trajectory and Graph 2 gives actual trajectory the robot travelled

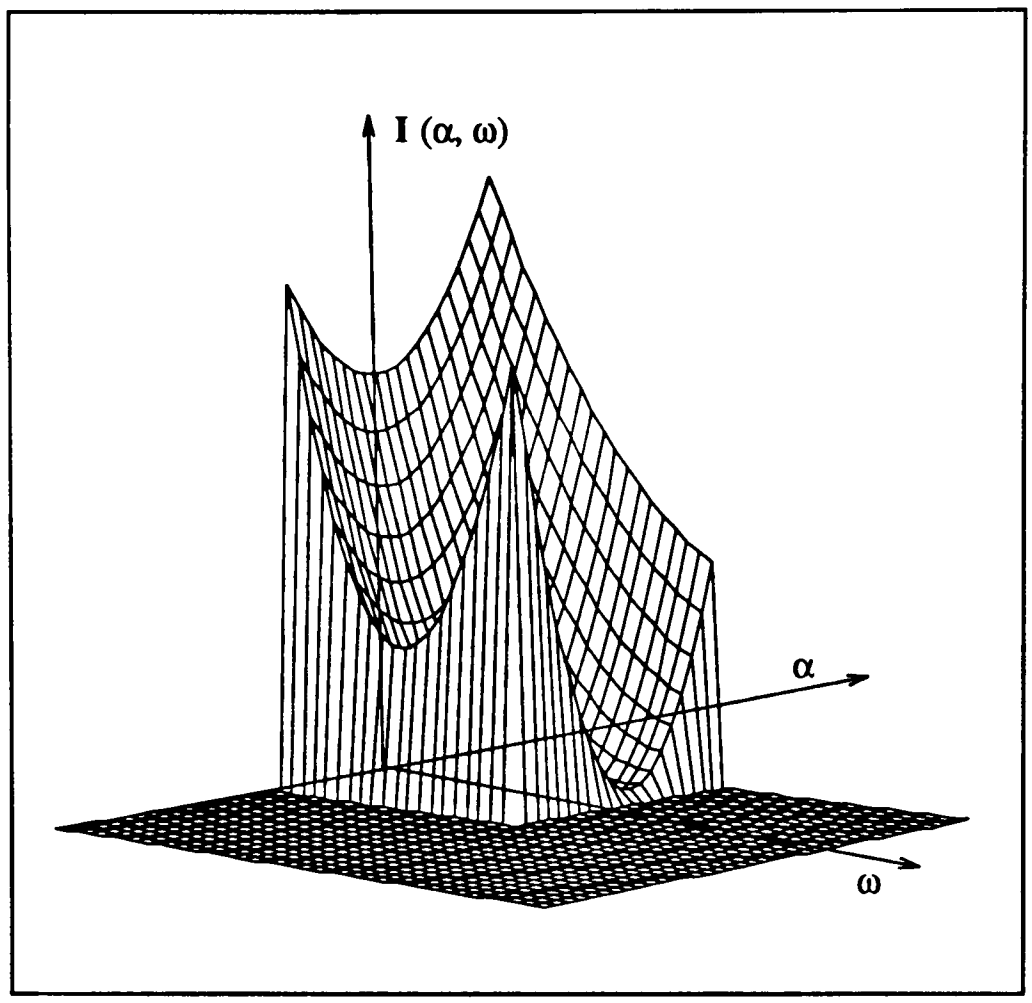



Figure 5.23: Cost Function Surface

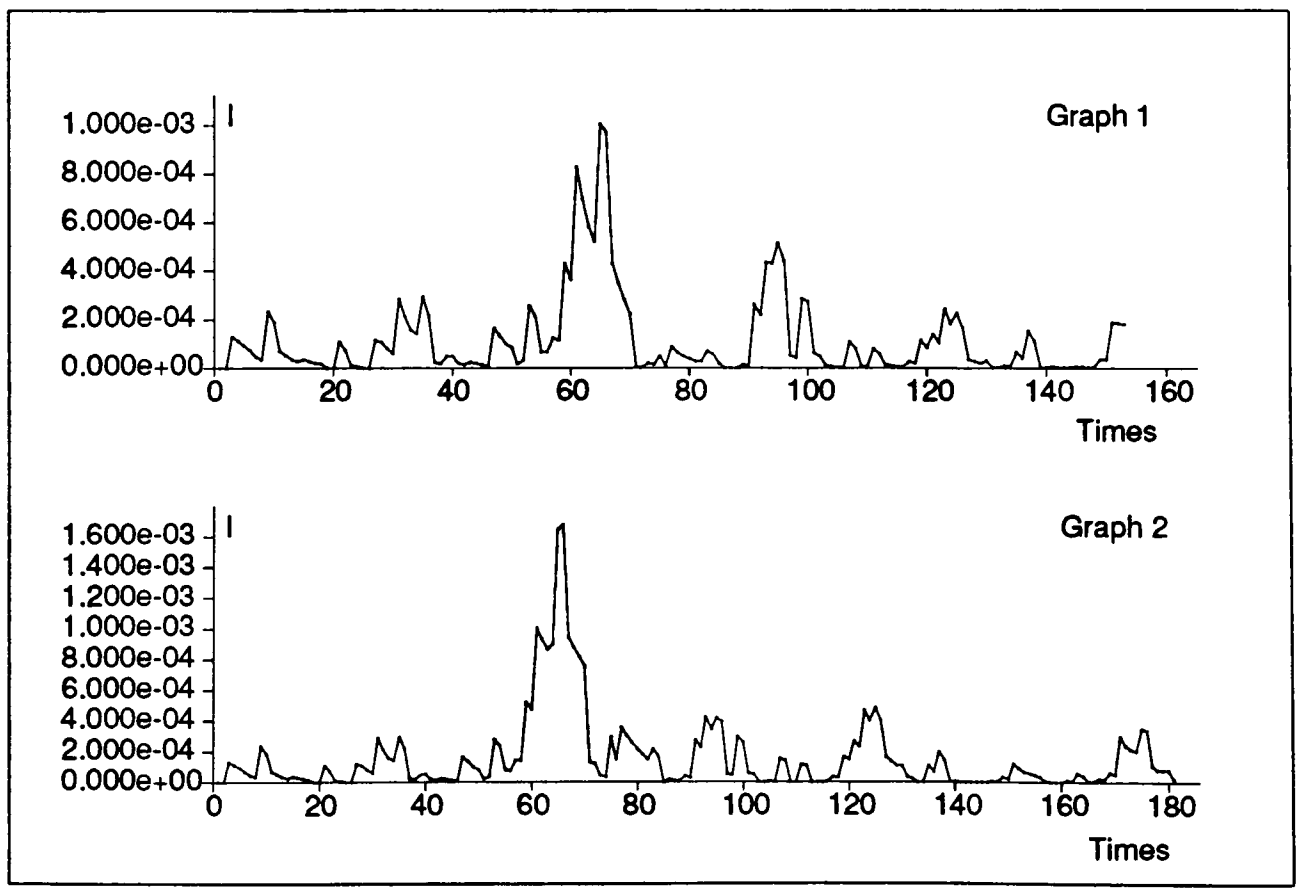



Figure 5.24: Plot of Minimum Value of Cost Function *vs* Time Steps for optimal tracking with single-step prediction

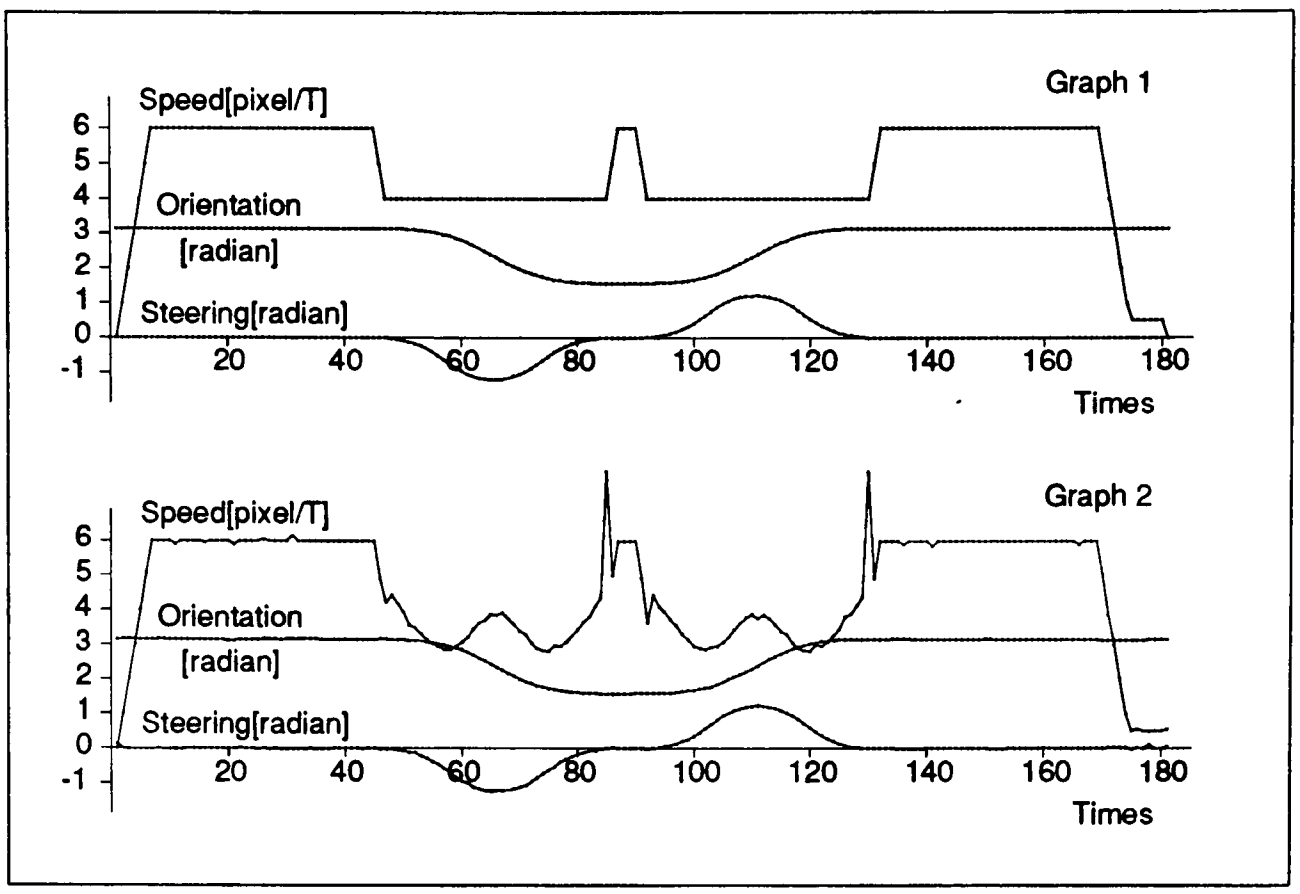



Figure 5.25: Optimal tracking of Line Segments and CPS with single-step prediction. Note that Graph 1 gives reference inputs and Graph 2 is tracking results

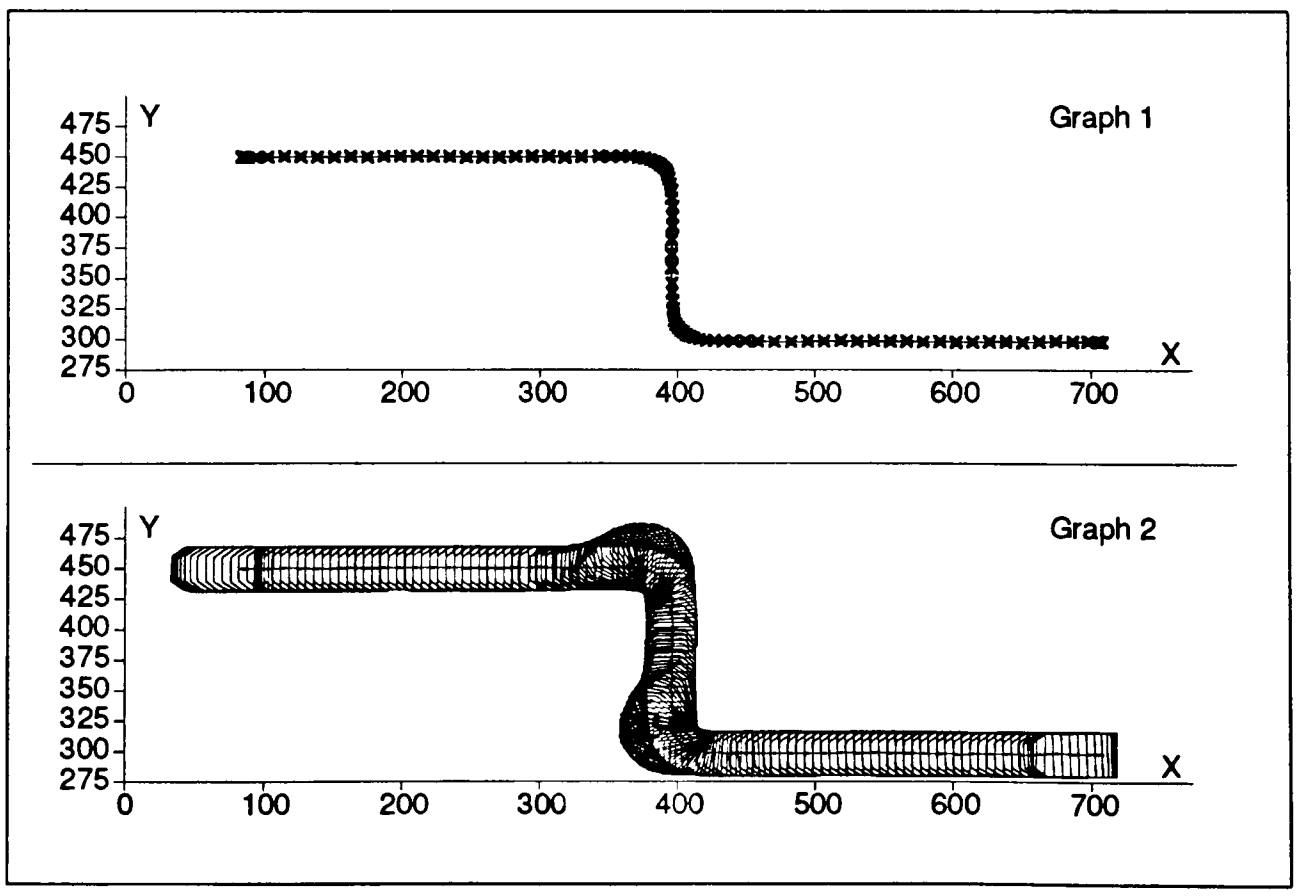



Figure 5.26: Optimal tracking of Line Segments and CPS with single-step prediction. Note that Graph 1 presents a reference trajectory and Graph 2 gives actual trajectory the robot travelled

ing $N_1 = 1$ and $N_2 = 2$, the cost function 5.67 becomes

$$\mathcal{I} = \sum_{i=1}^{2} ||\mathbf{x}_c(k) + \sum_{j=0}^{i-1} \Delta\hat{\mathbf{x}}(k+j-1) - \mathbf{x}_r(k+i)||^2_{\mathbf{S}_x} + ||\Delta\mathbf{u}(k)||^2_{\mathbf{S}_u} \tag{5.91}$$

The control variables $\mathbf{u}(k)$ are chosen at each sampling-instant to minimise the above cost function summed by two step predictions. Figure 5.27 shows the reference speed $\omega$, the reference steering $\alpha$, and the reference orientation $\theta$ in Graph 1 and the actual tracking results in Graph 2 for the straight line and SPP segments. It is clear that the vehicle speed avoids the several big jumps that happened in one-step prediction, seen in Figure 5.21. Similar tracking results for the straight line and CSP segments are shown in Figure 5.28, and again shows improvement, compared with Figure 5.25. Of course, this tracking strategy is in fact a compromise between the smooth change of control variables and the state errors to be minimised at each sampling-instant. In other words, it is optimal in two predictive steps rather than in one. Figure 5.30 gives the plots of the minimised values of the cost function 5.91 in terms of SPP turning in Graph 1 and CSP turning in Graph2. They are much bigger than the plots in Figure 5.24, which mainly caused by the lateral error along the moving direction, but still in the range of tolerance. Figure 5.29 presents the tracking results of the robot. Graph 1 shows that the robot travels along a continuous-curvature trajectory consisting of line segments and two SPP curves. Similarly, the results of tracking line segments and two CPS curves is presented. The transitions of the mobile robot are very smooth.

### 5.6.3 Closed form Solution

According to the analytical solution of the optimal tracking strategy 5.87, we choose the weighting matrices

$$\mathbf{S}_x = diag\,[1.0, 1.0, 0.5], \quad \mathbf{S}_u = diag\,[1.25, 1.25, 0.5] \tag{5.92}$$

Simulation results are given by Figures 5.31 and 5.32. The vehicle speeds are much smooth than the results given above since three-step predictions are adopted here. Moreover, the execution time is much fast than the numerical solution, which makes real-time application feasible.

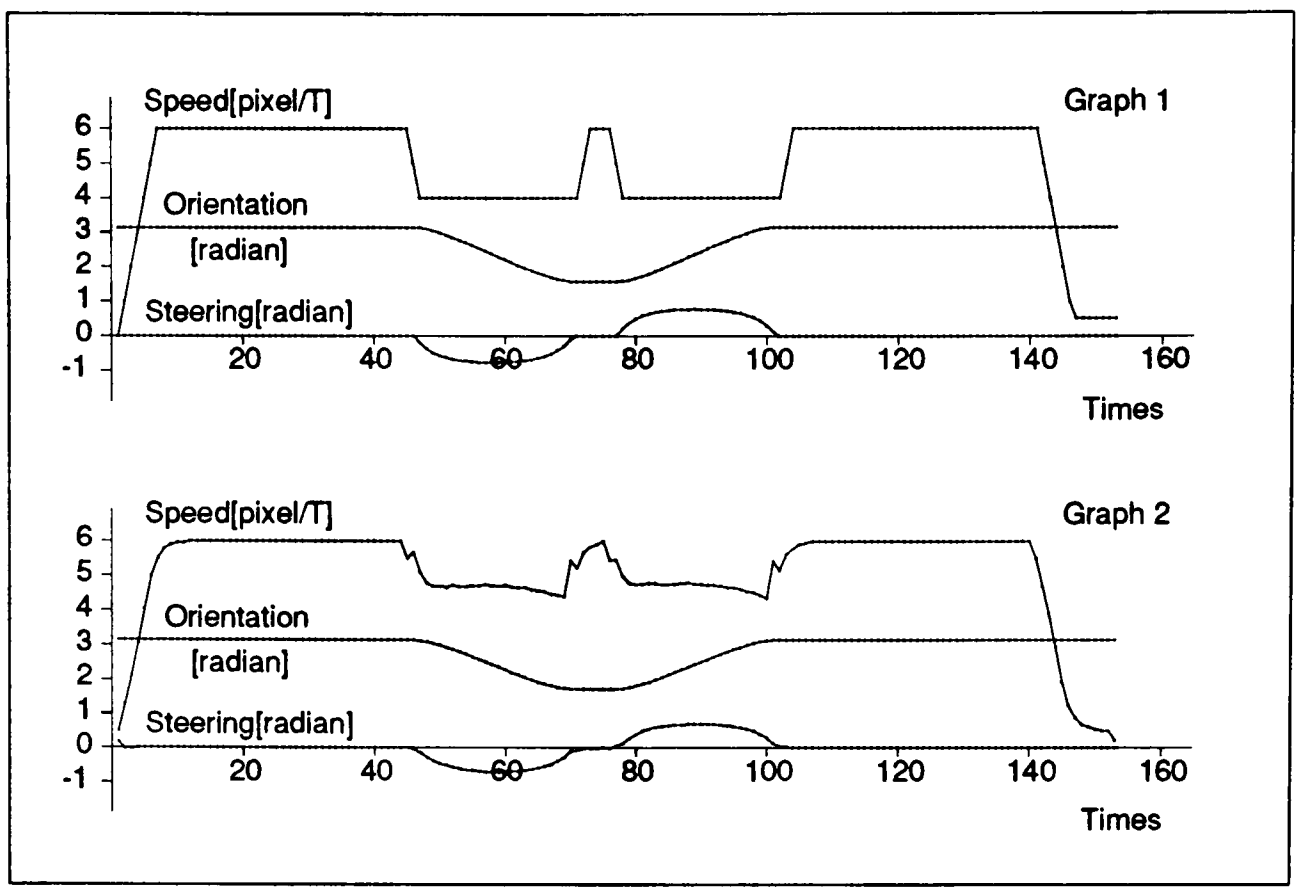



Figure 5.27: Optimal tracking with two-step prediction of SPP. Note that Graph 1 presents reference inputs and Graph 2 gives the tracking results

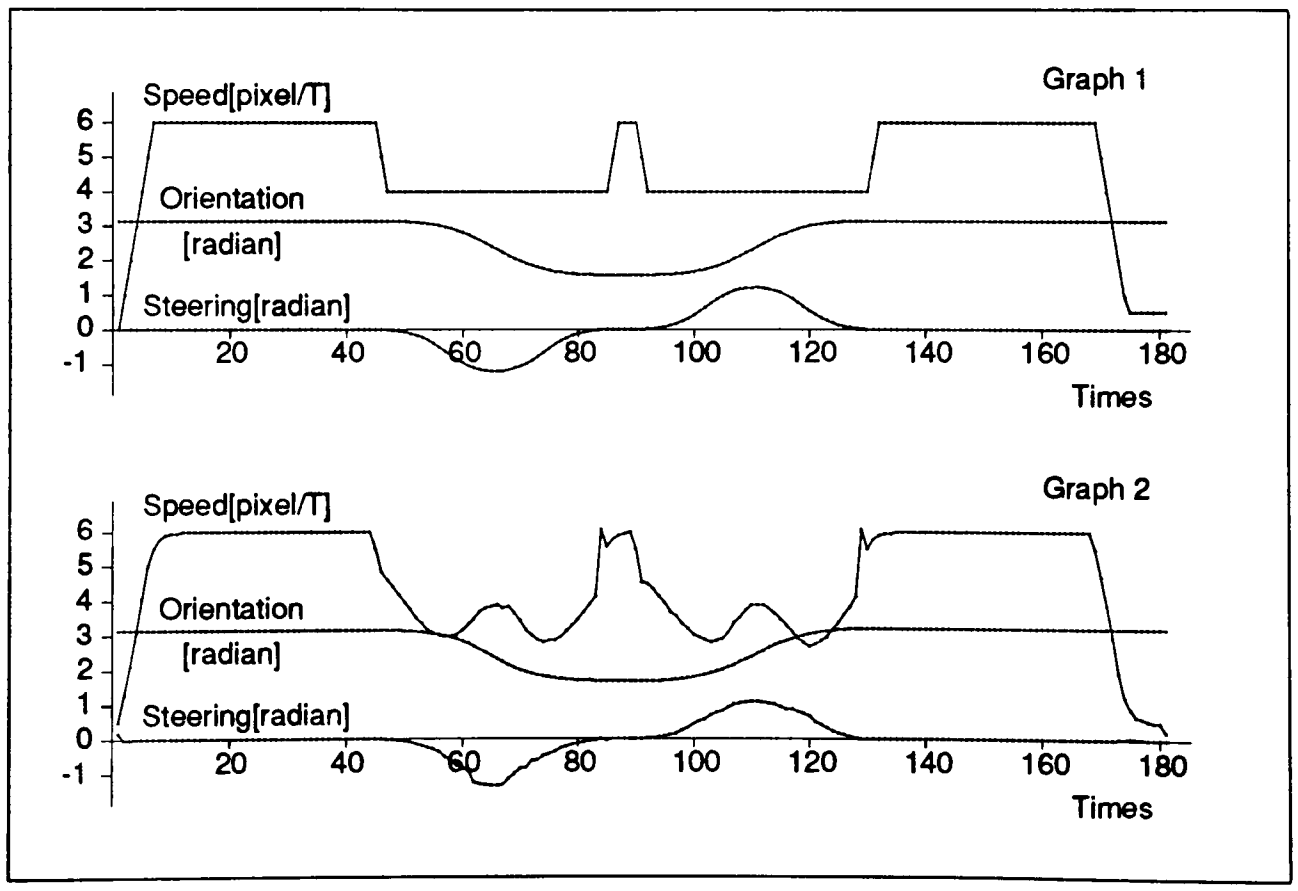



Figure 5.28: Optimal tracking with two-step prediction of CPS. Note that Graph 1 presents reference inputs and Graph 2 gives the tracking results

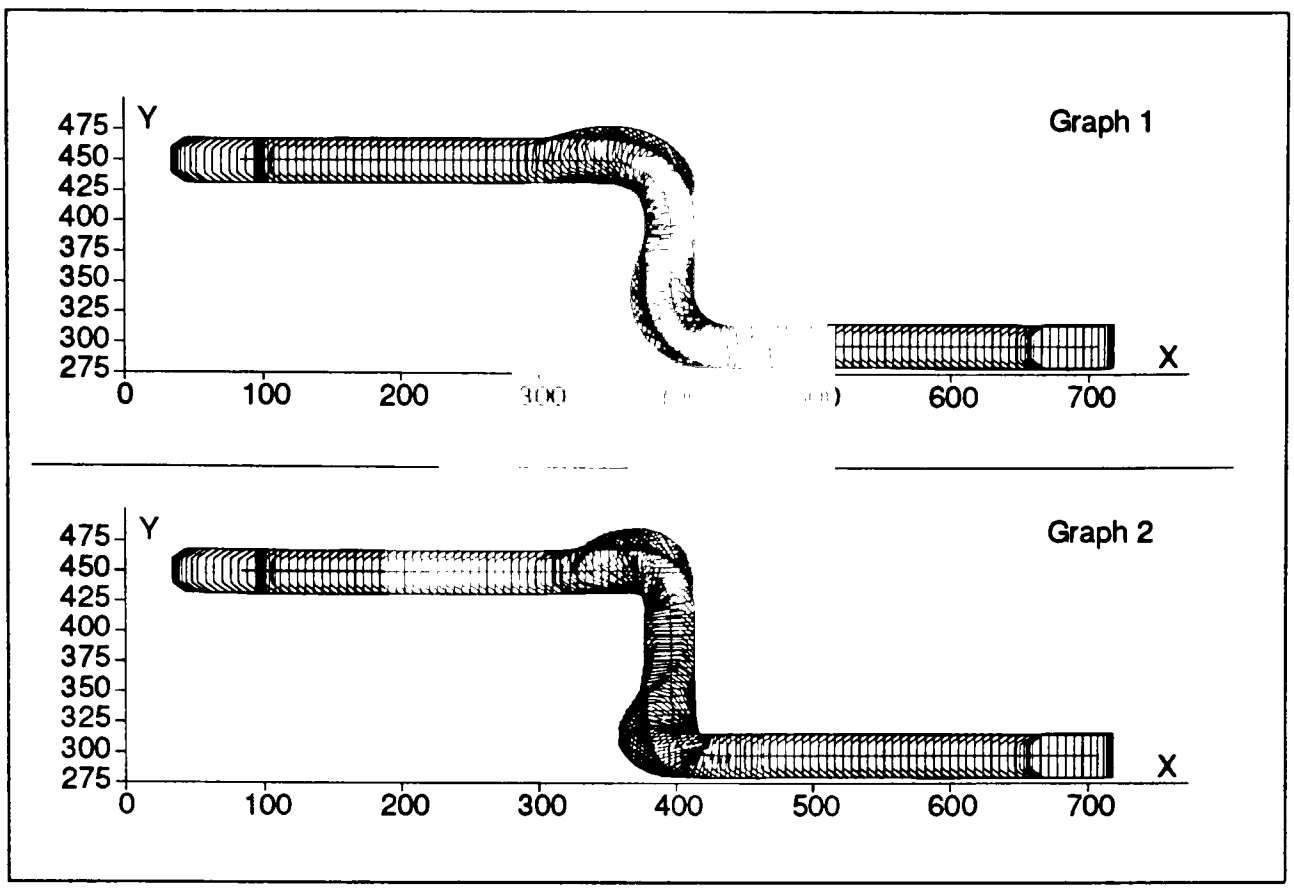



Figure 5.29: Optimal tracking with two-step prediction of the robot trajectories. Note that Graph 1 presents results of tracking SPP and line segments and Graph 2 gives the results of tracking CPS and line segments

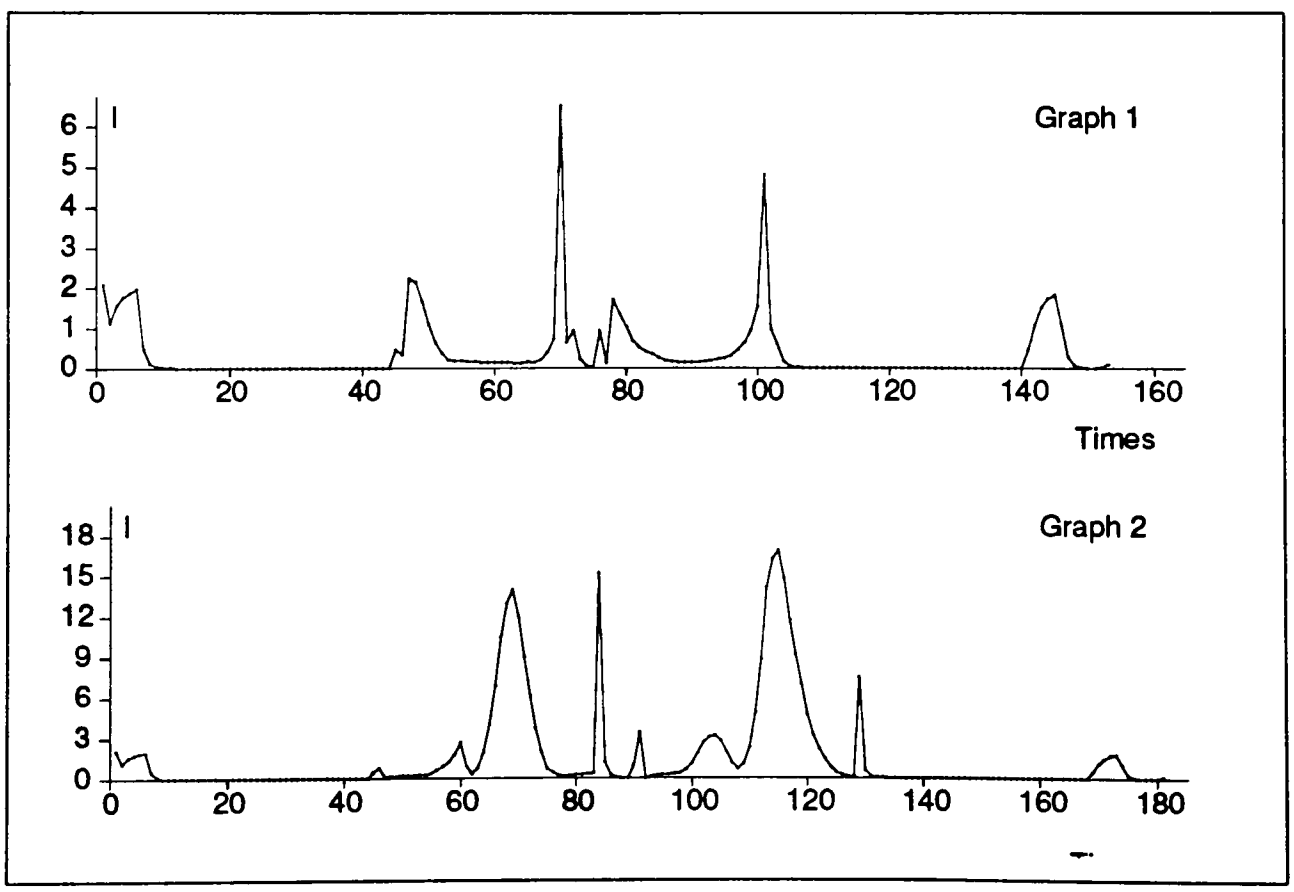



Figure 5.30: Plot the results of minimising objective function *vs* time steps. Note that Graph 1 gives the results of tracking SPP and line segments, and Graph 2 shows the results of tracking CPS and line segments

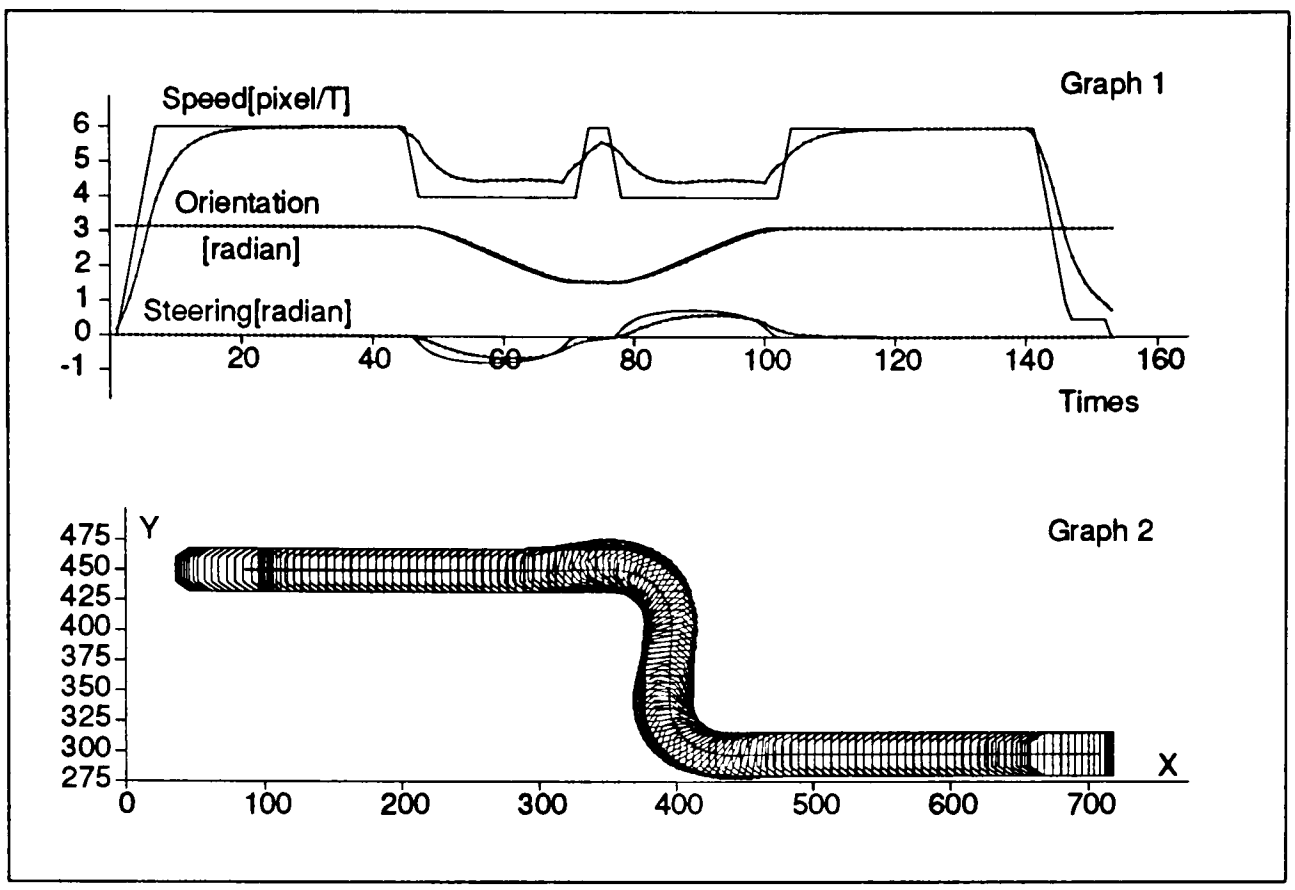



Figure 5.31: Analytical solution of optimal tracking SPP. Note that Graph 1 presents reference(solide line) and feedback(doted line) of $\alpha, \omega, \theta$ and Graph 2 gives the tracking results

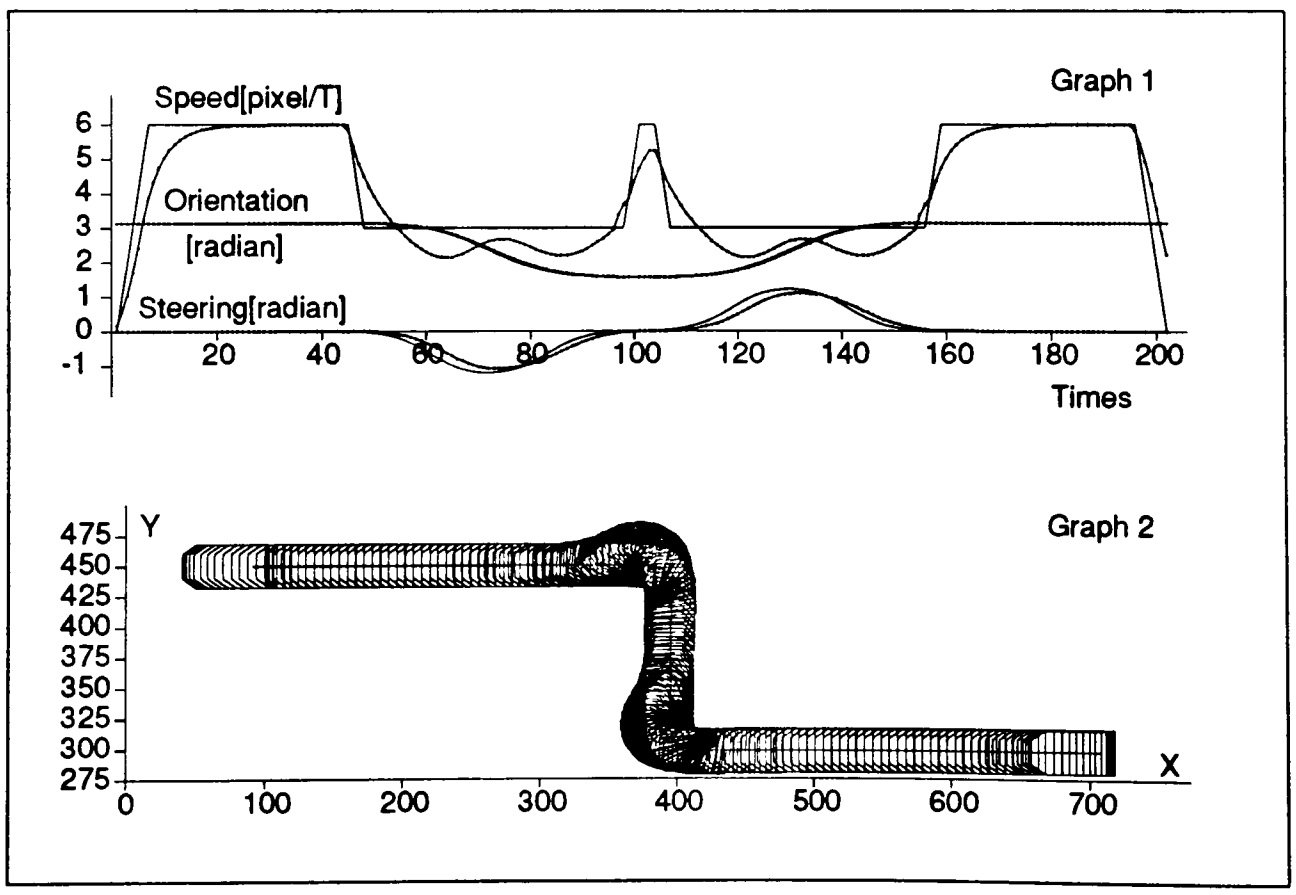



Figure 5.32: Analytical solution of optimal tracking CPS. Note that Graph 1 presents reference(solide line) and feedback(doted line) of $\alpha, \omega, \theta$ and Graph 2 gives the tracking results

## 5.7 Conclusions

In this chapter, a new guidance system for trajectory generation and trajectory tracking is proposed, taking into account the constraints from the changing environment and the robot kinematics. It enables a non-holonomic mobile robot to follow intermediate path points which are dynamically generated both by the global path planner and the local path planner. In general,

- The proposed guidance system consists of two layers: a trajectory generator and a motion controller. Continuous-curvature trajectories are dynamically generated for the mobile robot which is constrained to a roadlike motions and subject to non-holonomic constraints.
- The problem of controlling the motion of the non-holonomic mobile robot is formulated as an optimal control problem. Optimality is based on minimising a quadratic cost function consisting of the squared errors in the position and orientation of the robot. To obtain smooth tracking performance in the face of unmodelled disturbances, the generalised predictive control is adopted.
- A heuristic algorithm is proposed to find suitable control variables to perform optimal tracking. The algorithm starts by using the candidate control variables $(\alpha_r, \omega_r)$ given by the trajectory planner as an initial guess to the solution. Then it iteratively changes the candidates to decrease the value of the cost function $\mathcal{I}$ until either the desired accuracy or the predetermined time limit.
- The closed-form solution of the proposed optimal tracking strategy is also derivated by introducing an intermediate control variable. The control law naturally contains an integrator. Its gain can be adjusted during the applications.
- Simulation results show that the proposed control strategy provides high performance for tracking smooth trajectories that are dynamically generated.

# Chapter 6

# Conclusions and Future Research

Research in mobile robotics provides a rich environment for the testing and development of advanced concepts in a variety of areas such as navigation, sensor integration, real-world modelling, planning, control, and artificial intelligence. It has attracted much interest recently and will continue to pose many new challenges in real world applications. Research in this thesis provides a better understanding of the fundamental issues related to sensing, planning, and control for a mobile robot. A general framework to integrate different decision making capabilities for a mobile robot to handle uncertainties in the real world and sensors being used is presented. In this chapter, achievements so far and future extensions are summarised.

## 6.1 Achievements so far

This research has made significant progress towards the fully autonomous competence of mobile robots in industrial applications. The results demonstrate the success of the proposed approach. In general, both achievements and understanding can be given as follows:

1. **Distributed real-time architecture**

   An autonomous mobile robot must be constantly involved in the processing of large amounts of sensory data in order to produce meaningful actions. The ability of a control architecture to support this immense processing task in a timely manner is significantly affected by the organisation of information pathways within the architecture. The transputer architecture developed in this thesis has been explicitly designed to maximise the amount of parallel

information flow from sensing to action so as to provide minimal delay in responding to a constantly changing environment. Our own experience with such a system has led us to a better understanding of how action may be derived from the results of multiple independent decision-making processes and how plans may be used to guide these processes.

2. **Probabilistic sensor model**

   To handle dynamic changes in the environment, multiple sonar sensors have been used in this research. However, data produced by these sonar sensors can not tell the whole story about the real world because of sensor errors. Advanced data fusion techniques can be used to reduce errors to a limited range, but may result in high computation cost. Alternatively, in the case of obstacle avoidance where time is critical and data accuracy is not very strict, we build a simple sensor model (probabilities conditioned on the state of the world) to express what is to be expected from these sonars by analytic experiments. Such a model is then used to interpret sensor data and to achieve real-time performance for collision avoidance.

3. **Dynamic planning with uncertainty**

   Traditionally, robot planning has been treated as a purely predictive process where complete knowledge about the environment is assumed *a priori*. Recovering from error and dealing with unexpected events are usually left to the execution stage. No monitoring process is included. As a result, traditional planning systems are open-loop in nature and are unable to handle unmodelled disturbances in the real world. In contrast, the dynamic planning algorithm proposed in this thesis can be considered as a closed-loop planner, as seen from the point of view of feedback control. It includes two processes: monitoring and prediction. More precisely, it monitors dynamic changes in the real world (estimate) and makes prediction for an optimal path (planning). Statistical models have been built to estimate the state of path by quantifying uncertainty. They are updated dynamically and have an exponentially decay memory. As a result, the planning algorithm we develop allows the Turtle robot to respond

its environmental changes intelligently.

4. **Decision making to avoid collision**

   Decision-making under uncertainty is an important issue for mobile robots which navigate in a real world since no sensor is perfect. As humans, we know that very few decisions are made under perfect conditions. In this thesis, the collision avoidance problem has been cast into the general framework of a decision making process to achieve both optimal and collision-free paths. Based on prior information and new observation, trade-off is made between careful maneuvers and alternative paths so that Bayes risk is minimised.

5. **Distributed map building**

   Map building in this thesis has been distributed into different decision making layers to achieve specific goals. Four maps are being built, namely a topological map (a weighted search graph) for global path planing, a local map (collision zone and prediction zone) for collision avoidance, a geometric map for localisation, and another local map (in the vicinity of the robot) for trajectory tracking. They are each tuned to a different range of situations. As a result, they are easy to implement and are updated very fast.

6. **Optimal tracking algorithm**

   A new guidance system is proposed in the thesis to generate a continuous-curvature trajectory dynamically and track it accurately. A quadratic cost function is adopted to achieve optimal performance in terms of minimum variance in the position and orientation of the robot. The generalised predictive control strategy is used to obtain smooth tracking performance in the face of unmodelled disturbance.

## 6.2 Future Research

We conclude by identifying some directions for future research as follows:

- **Integrate Different Sensors**

  It can be seen from this thesis that the major difficulty in robot perception is to find out what the sensor data tells about the real world. In other words, the problem is to interpret the sensor readings more precisely. However, no matter how good a sensor is and how efficient data processing algorithms are, uncertainty in sensor data can not be eliminated totally. The use of multiple, physically different sensors can help to overcome the shortcomings of each individual sensing device since information can be provided concerning environment features that cannot be perceived using each sensing device alone. Also the fusion of redundant data from multiple sensors can reduce uncertainty especially if an individual sensor fails. There are many different sensors and navigation techniques that are widely used in mobile robots, which include ranging sensors, stereo vision, triangulation, laser range, tactile sensing, inertial navigation. One of natural extensions of this research is to integrate more sensors such as the stereo vision head that is currently being built in our AGV Laboratory [Du and Brady, 1992] to provide information about the size of obstacles which sonars can not resolve.

- **Avoiding Moving Obstacles**

  The problem of avoiding moving obstacles poses a challenge for the successful application of the mobile robots in a real world. To avoid moving obstacles such as people, other robots, and independent objects is substantially harder than the problem of avoiding unexpected static obstacles. This is because in the static case, once the path is found, the robot's speed along it does not matter for collisions. However, to guarantee that there will be no collisions with a moving obstacle, the robot's speed must also be determined according to the velocities and trajectories of moving obstacles that are known *a priori* or made available by the sensors. Moreover, the constraints imposed by the maximum velocity and acceleration of the robot are crucial in the avoiding strategy. Based on a Kalman fileter algorithm, we have conducted the preliminary study in [Hu *et al.*, 1991b] by simulation and the further investigation will be

done in the next stage.

- **Multiple Tasks and its Scheduling**

  Research carried out in this thesis only addressed how a mobile robot achieves an optimal solution for one specified task each time. However, a mobile robot used in industry may often have multiple tasks with different priorities, and some of them may conflict. Its control system must be responsive to high priority tasks, while still serving necessary low priority tasks. For the same priority tasks, careful scheduling is needed to achieve an optimal solution such as minimum time or energy. Therefore, it is a natural extension that the mobile robots are provided with more local intelligence to manage multiple tasks. It is obvious that we really want to optimise is total time in terms of multiple tasks. Sequential decision theory can be used to provide mobile robots with a scheduling capability to handle multiple tasks.

- **Multiple Mobile Robots and Decentralised Control**

  In industry, it is likely that there is more than one mobile robot working in the same environment. How to cooperate with each other is a natural problem that needs to be solved. In a centralised control system, cooperation among different robots is normally done by a control center. This is obviously a major bottleneck for the whole system. It would be more efficient to provide sufficient intelligence to each mobile robot to enable it to manage cooperation with other robots by itself. Exactly like humans have devised for the highway system, a traffic law may be a solution for this situation. Each robot will be provided with the knowledge in traffic law and know clearly what it should do in each situation such as joining a main road, turning left or right. Cooperation is a key issue in building a decentralised control system for multiple mobile robots. Then the environmental learning capability based on statistical model presented in this thesis can be used with some modifications by mobile robots to plan an optimal path subject to constraints imposed by traffic law.

# Appendix A

# The Oxford AGV: Turtle

As the first mobile robot project at Oxford, the *Turtle* provided by GEC Fast has lived in our AGV Laboratory since August 1988. It is a research version of the commercially-available, free-range AGV designed for transporting materials around factories. The work described in this thesis was developed originally in the context of the *Turtle* mobile robot. In this appendix, we briefly introduce the original control system of our *Turtle* mobile robot, including the computer architecture, control strategy, landbase system, laser location system, and the guidance system. Further details can be found in [Rugby, 1987].

## A.1 Landbase and the AGV

The *Turtle* mobile robot system is remotely controlled by a landbase control station (LCS) via a communication system, shown in figure A.1.

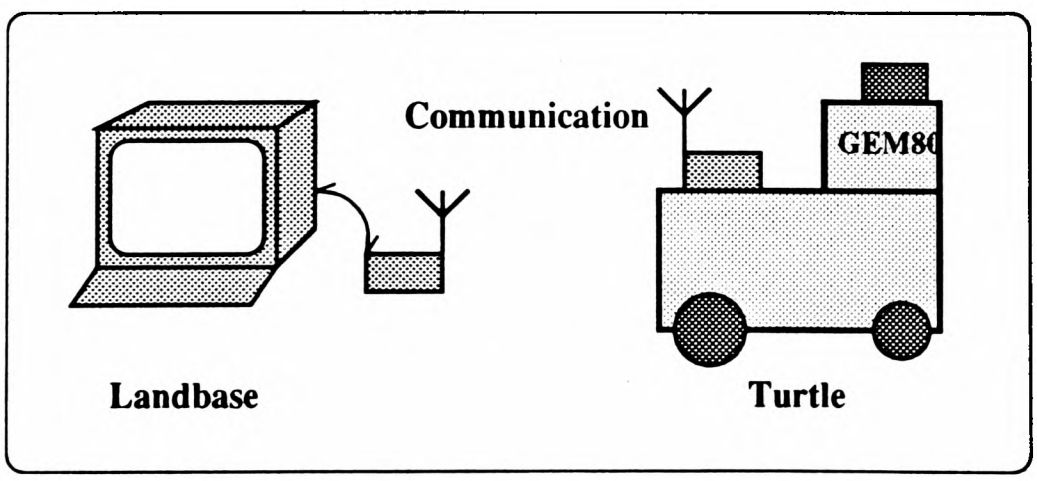



Figure A.1: Landbase and the AGV

The landbase, a microcomputer system, performs the following functions:

- Entry and display of allowable motion routes with detail of speed limit zones and width restrictions, including aisles, docking points, junctions, user zones, and paths.
- Entry of retroreflective barcode chart positions for laser guidance.
- Scheduling in multiple AGVs installations; rescheduling in fault conditions.
- Display of AGV positions, headings, operational parameters and alarms.
- Replan of an alternative path when encountering unexpected obstacles.
- Operation of a multidrop serial link protocol, called an Extended Simple Protocol (ESP), for communication with the robots.

Continuous bi-directional communications between the landbase and the *Turtle* is essential to allow the landbase to dynamically control the robot motion and allow the *Turtle* to report the status of its operation.

## A.2 On-board GEM80 Controller

The *Turtle* is controlled by an industrial programmable controller, namely the GEM80 [Rugby, 1986]. The GEM80 is supplied complete with power supply, microprocessors(Intel 8085, 8088, 8086), memory and space for 8 small I/O modules which can be configured to suit the industrial applications. It has a tightly coupled multiprocessor architecture in which all processors share a common memory. It is programmed by a special language, a ladder diagram, which offers a development environment for different applications. The GEM80 is a cycle controller. Its cycle time can be changed according to different applications. For the *Turtle*, it has been chosen at 80 ms. Figure A.2 shows its configuration for the control of the *Turtle* and a cycle of the operation.

The control task of the *Turtle* is functionally decomposed into different function modules, shown in figure A.3. It is a hierarchical architecture, combining three main functions: message handling, system management, task control. For more detail, see [Rugby, 1988].

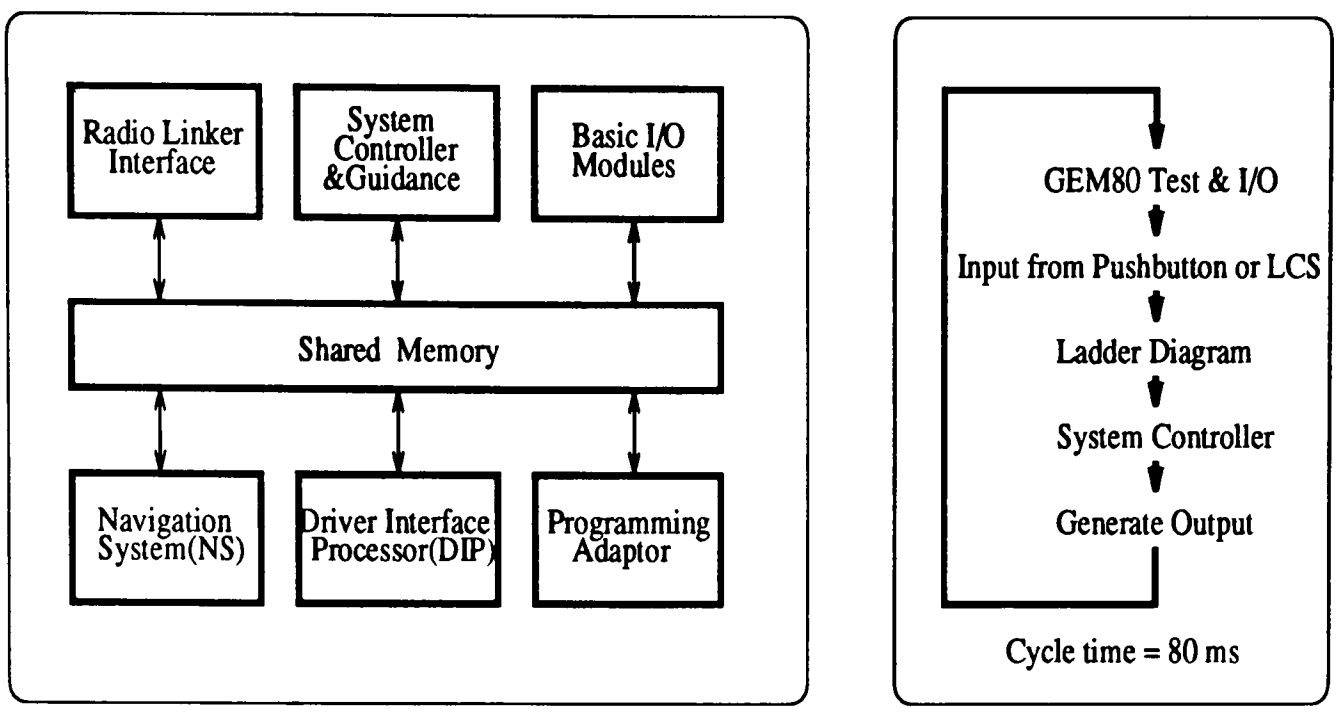



Figure A.2: The GEM80 Controller for the *Turtle* Mobile Robot

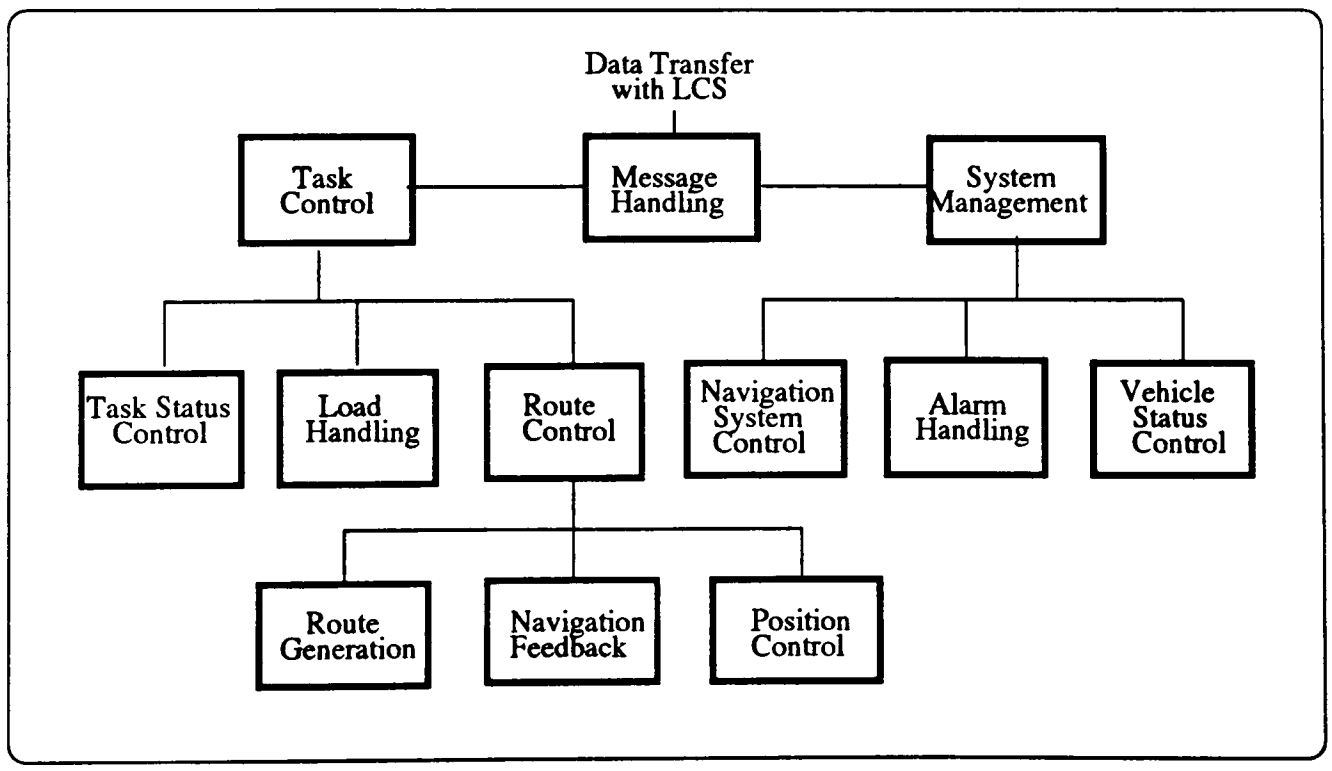



Figure A.3: Block Diagram of the Modular Structure

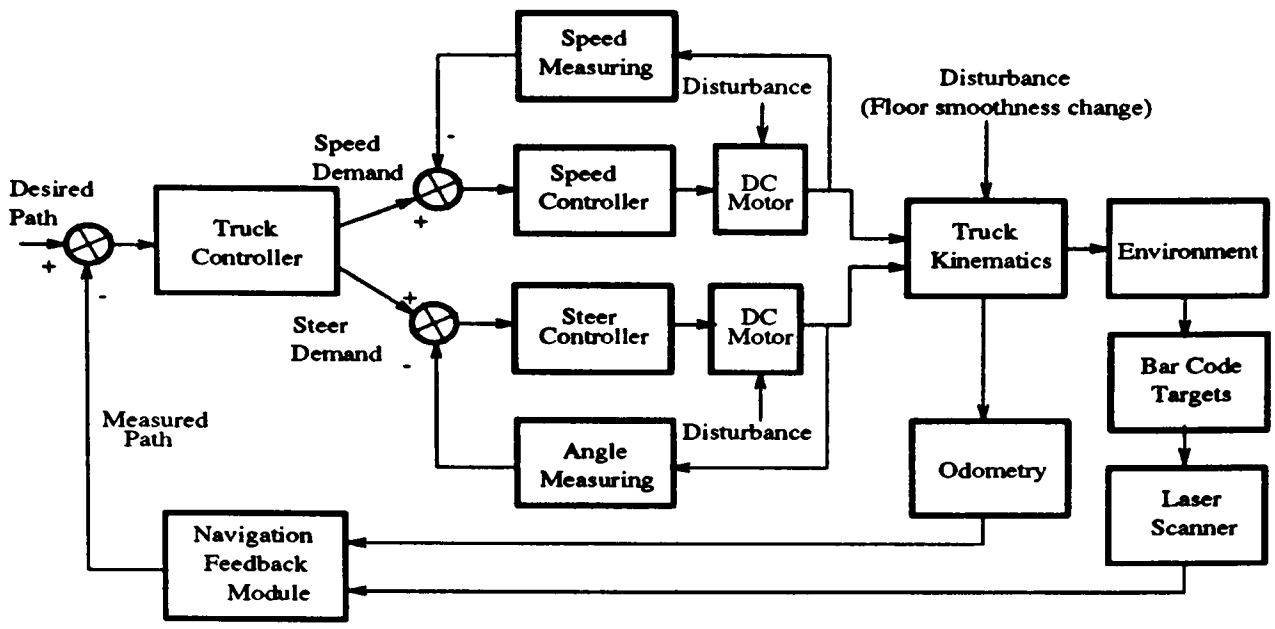



Figure A.4: Two feedback loops for *Turtle* mobile robot

## A.3 Control Strategy

The control of the *Turtle* is based upon two feedback loops as shown in Figure A.4. The inner loop consists of two parallel feedback loops for motion and steering control. Two traditional analogue PID controllers are employed so that the stable speed and steering angle can be obtained whenever disturbances appear. The outer loop based on GEM80 is used for position control of the robot. The odometry and bar charts information are feedback to a navigation system inside the GEM80. A guidance system is to track the position demand to follow the desired path issued from the landbase or preset in the memory. Based on error-feedback control principle, both loops automatically respond to unforeseen system change or to unanticipated disturbances. This control system has no capabilities regarding obstacle avoidance and path planning. If an obstacle is placed in its way, the *Turtle* can only use its physical plastical bumper to stop and waits for the obstacle to be removed.

## A.4 Laser Location System

The position of the *Turtle* is determined from data provided by a laser scanner situated on the top of the vehicle and by odometry. Light from a HeNe laser diode (class 3b) is scanned in azimuth at a constant speed of rotation at two revolutions per second. Passive retroreflective targets, in the

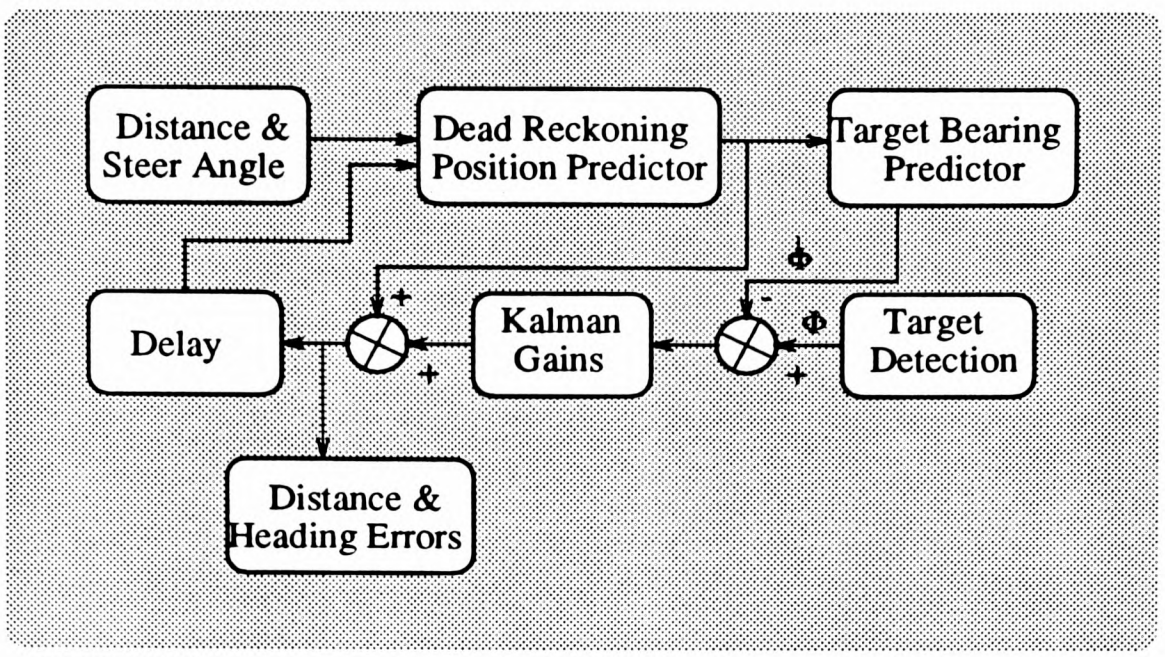



Figure A.5: Block Diagram of the Position Measuring Algorithm

form of bar codes, are positioned around the working area; the reflected light from these targets is received by a photodetector in the scanner unit. Each target is a unique 5 bit code and the absolute position of each target is known.

The position measuring algorithm is capable of simultaneously accepting data from different sensor devices, adding a weighting function to this data and producing new position predictions. Figure A.5 shows its block diagram. Since both sets of data, the odometry data $(\alpha, \beta)$ and the measured angle $\lambda$ to each bar chart, are noisy, a Kalman filter is used to estimate the position $(x, y)$ and orientation $\theta$ of the robot.

## A.5 Guidance System

We describe more details of the guidance system in this section. Figure A.6 shows the block diagram. The message handling process receives one task and its corresponding route description file (up to 59 steps) each time from the landbase in a landbase mode or from a P-table (pre-stored in memory by the user's ladder program) in a pushbutton mode. A new task will not be accepted by the system until the present task has been completed. Then, it uses a curve fitting algorithm to generate a smooth trajectory (curves and straight lines) which in turn is divided into small

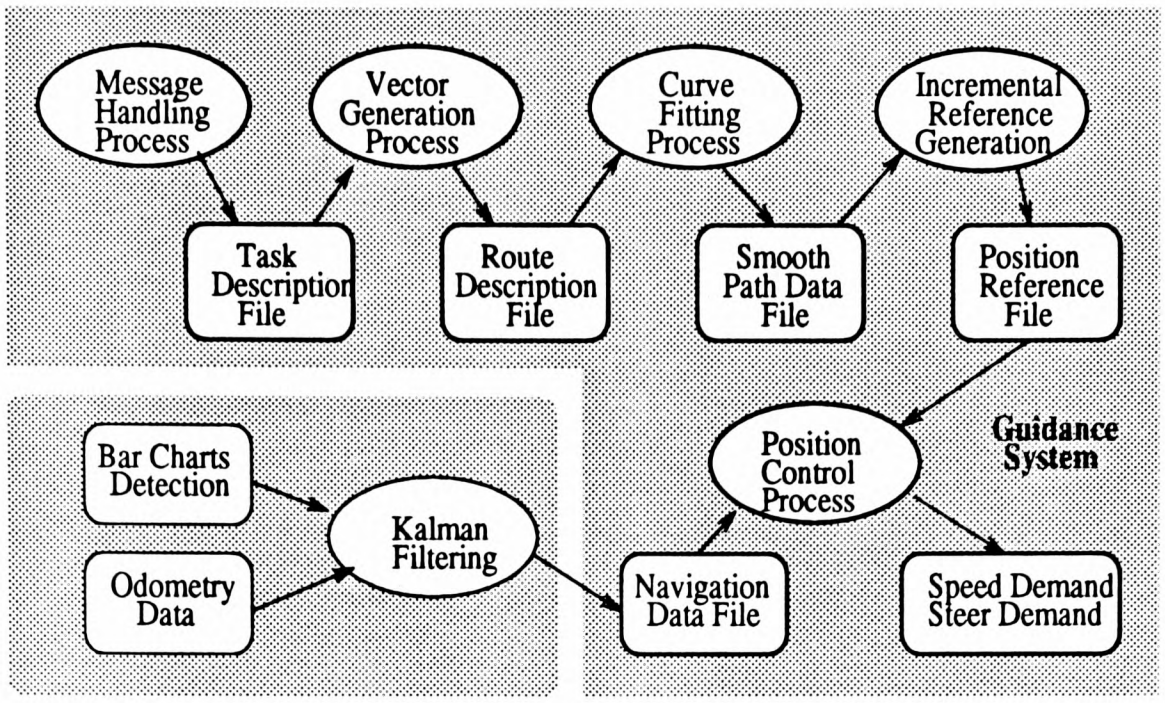



Figure A.6: Block Diagram of the Guidance System

incremental reference vectors, described by pathwidth, position, heading and speed. These vectors are then stored into a position reference file (PRF) and used sequentially as the reference positions for the position control. Finally, the position control process calculates the position error between the reference position and the feedback position in the navigation data file, and send the speed and steer demand to the vehicle controller to control the robot to follow the smooth fitted route at all points and minimise the deviation from it.

## A.6 Overview of the Control System for *Turtle*

Figure A.7 presents a block diagram of the configuration of the *Turtle* control system. It works as follows:

- A rotating infrared laser sensor reads the bar charts on the wall and sends the measured code information back to the SIP for further processing.
- A feedback module consists of three processors (SIP, NS, DIP). The SIP processes data from the laser sensor and transmits target code and angle information (in two words) to the navigation system (NS) immediately when a valid target is seen. The DIP, a modified version

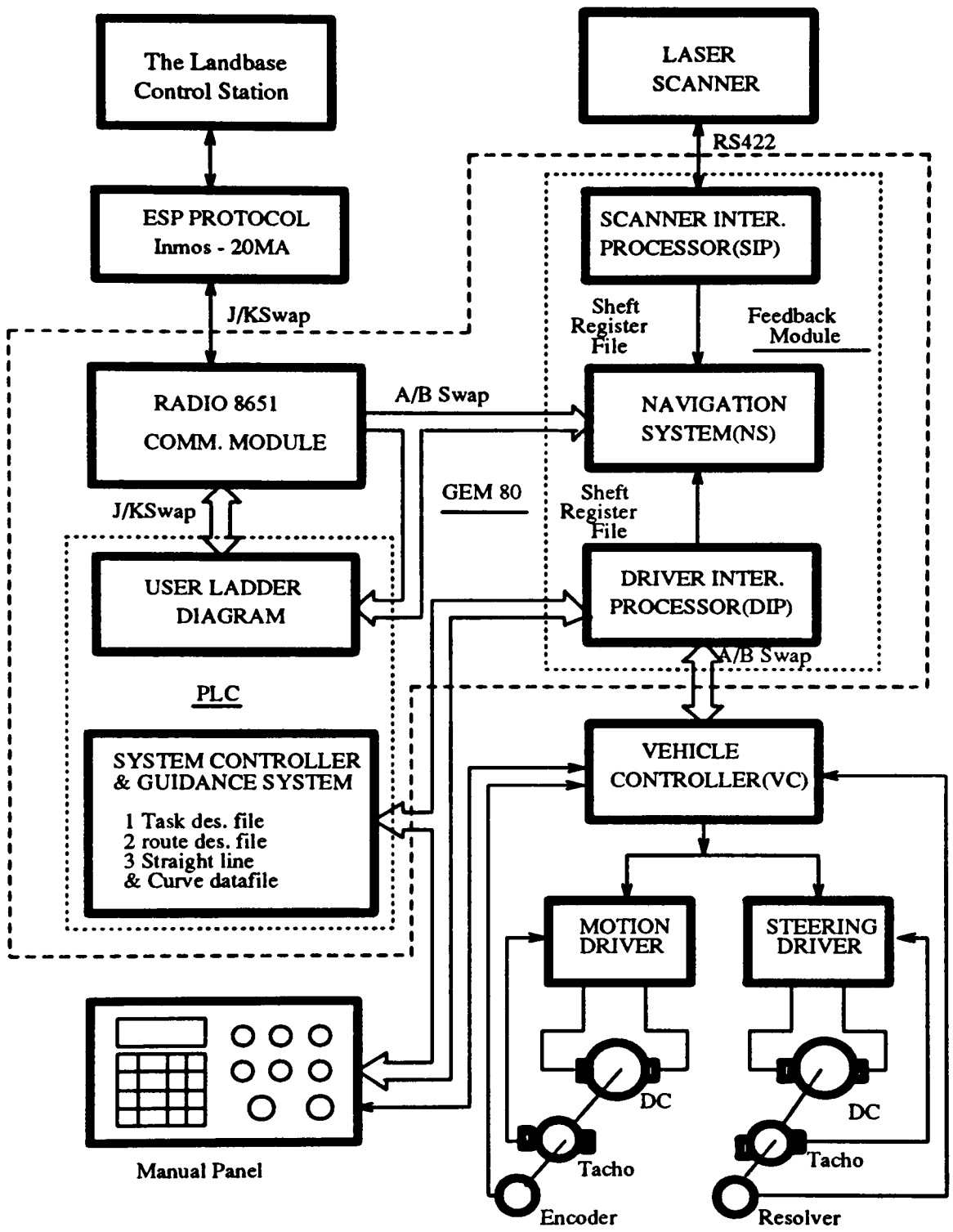



Figure A.7: Block Diagram of Control System Configuration

of the GEM80 basic I/O, collects odometry data and sends it to the NS. Using both data, the NS is able to run a Kalman filter to locate the AGV position and store the results in a navigation data file.

- The user's ladder diagram handles the input from a pushbutton panel, communicates with the landbase, and transfers data among the guidance system, the navigation system, and the vehicle controller.
- The guidance system handles only one task and its corresponding route description file at each time. It uses a curve fitting algorithm to generate a smooth trajectory as a position demand. It then compares the position feedback information in a navigation data file to generate the speed and steer demands which is sent to the vehicle controller via the DIP.
- The vehicle controller (VC) based on Intel 8088 processor receives the motion demands, either from the DIP in a automatic mode or from a joystick in a manual mode, to control the drive speed and steer angle of the front wheel. Also, it sends back the odometry information to the navigation system via the DIP.
- The conventional analogue PID controllers are used in two drive units to drive two DC motors and maintain the stable speed and steering angle.

# Bibliography


[Adams *et al.*, 1990] M. Adams, H. Hu, and P.J. Probert. Towards a real-time architecture for obstacle avoidance and path planning in mobile robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 584–589, Cincinatti, USA, 1990.

[Albus *et al.*, 1981] J.R. Albus, A.J. Barbera, and R.N. Nagel. Theory and practice of hierarchical control. In *Proc. of the 23th IEEE Computer Society Int. Conf.*, pages 18–27, New York, September 1981.

[Albus *et al.*, 1982] J.R. Albus, C.R. Mclean, C.R. Barbera, and M.L. Fitzgerald. An architecture for real-time sensory-interactive control of robots in a manufacturing facility. In *Proc. of the 4th IFAC/IFIP Symposium in Information Control Problems in Manufacturing Technology*, pages 81–90, Gaithersburg, 26-28 October 1982.

[Allen and Kennedy, 1982] J.R. Allen and K. Kennedy. Pfc: A porgram to convert fortran to parallel form. In *Proc. of the IBM Conference on Parallel Computers and Scientific Computations*, March 1982.

[Anderson and Donath, 1990] T.L. Anderson and Max Donath. Autonomous robots and emergent behavior: a set of primitive behaviors for mobile robot control. In *IEEE Int. Workshop on Intelligent Robots and Systems*, Tsuchiura, Ibaraki, Japan, 1990.

[Andersson, 1986] R.L. Andersson. Living in a dynamic world. In *Proc. of the ACM-IEEE Fall Joint Computer Conference*, pages 97–104, 1986.

[Aoki, 1967] M. Aoki. *Optimization of stochastic system*, volume 32. Academic Press Inc., New York, USA, 1967.

[Arkin, 1989] R.C. Arkin. Motor schema-based mobile robot navigation. *Int. J. Robotics Research*, 8(4):92–112, 1989.

[Badreddin, 1991] E. Badreddin. A recursive control structure for mobile robots. In G. Schmidt, editor, *Information Processing in Autonomous Mobile Robots*, pages 171–185. Springer-Verlag, 1991.

[Barshan, 1991] B. Barshan. *A Sonar-based Mobile Robot for Bar-like Prey Capture*. PhD thesis, Yale University, December 1991.

[Bellman and Kalaba, 1960] R. Bellman and R. Kalaba. Dynamic programming and adaptive processes: mathematical formulation. *IRE Trans. on Automatic Control*, AC-5:5–10, 1960.

[Bellman, 1957] R. Bellman. *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[Berger, 1985] J.O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Verlag New York Inc., USA, 1985.

[Bertsekas, 1976] D.P. Bertsekas. *Dynamic Programming and Statistic Control*. Academic Press, 1976.

[Borenstein and Koren, 1989] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Trans. Systems Man and Cybernetics*, 19(5):1179–1186, 1989.

[Borenstein and Koren, 1990a] J. Borenstein and Y. Koren. Real-time map-building for fast mobile robots obstacle avoidance. In *Proc. SPIE*, pages 74–81, 1990.

[Borenstein and Koren, 1990b] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environment. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 572–577, 1990.

[Borenstein and Koren, 1992] J. Borenstein and Y. Koren. Noise rejection for altrasonaic sensors in mobile robot applications. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1727–1732, Nice, France, May 1992.

[Brady *et al.*, 1987] J.M. Brady, S. Cameron, H. Durrant-Whyte, M. Fleck, D. Forsyth, A. Noble, and I. Page. Progress towards a system that can acquire pallets and clean warehouses. In *Fourth Int. Symp. Robotics Research*, Santa Cruz, August 1987.

[Brady *et al.*, 1990] J.M. Brady, H. Durrant-White, H. Hu, J.J. Leonard, P.J. Probert, and B.S.Y. Rao. Sensor-based control of AGVs. *IEE Journal of Computing and Control Engineering*, pages 64–70, March 1990.

[Brady, 1982] J.M. Brady. Trajectory planning. *Robot Motion – Planning and Control*, pages 221–243, 1982.

[Brooks, 1983] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Trans. Systems Man and Cybernetics*, 13:190–197, 1983.

[Brooks, 1986] R.A Brooks. A robust layered control system for a mobile robot. *IEEE J. Robotics and Automation*, 2:14, 1986.

[Brooks, 1987a] Rodney A. Brooks. Autonomous mobile robots. In W.E.L. Grimson and R.S. Patil, editors, *AI in the 1980s and Beyond*. The M.I.T Press, 1987.

[Brooks, 1987b] Rodney A. Brooks. A hardware retargetable distributed layered architecture for mobile robot control. In *Proc. IEEE Int. Conf. Robotics and Automation*, page 106, 1987.

[Burns, 1988] Alan Burns. *Programming in OCCAM 2*. Addison-Wesley Publishing Company, 1988.

[Cameron and Durrant-Whyte, 1990] A. Cameron and H. Durrant-Whyte. A bayesian approach to optimal sensor placement. *Int. J. Robotics Research*, 9(5):70–88, 1990.

[Chatila and Laumond, 1985] R. Chatila and J.P. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, page 138, 1985.

[Clarke *et al.*, 1987] D.W. Clarke, C. Mohtadi, and P.S. Tuffs. Generalized predictive control – part i. the basic algorithm. *Automatica*, 23(2):137–148, 1987.

[Cmelik *et al.*, 1986] R.F. Cmelik, N.H. Gehani, M. Plotnick, and W.D. Roome. Concurrent c project. Technical report, A collection of reports on concurrent C, 1986.

[Corporation, 1984] Polaroid Corporation. *Ultrasonic Ranging System.* Cambridge, Mass., 1984.

[Crowley, 1985] J.L. Crowley. Navigation for an intelligent mobile robot. *Proc. IEEE Int. Conf. Robotics and Automation*, 1:31–41, 1985.

[Crowley, 1989a] J. L. Crowley. Asynchronous control of orientation and displacement in a robot vehicle. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1277–1282, 1989.

[Crowley, 1989b] James L. Crowley. World modelling and position estimation for a mobile robot using ultrasonic ranging. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 674–680, 1989.

[Daily and *et al*, 1988] M. Daily and *et al.* Autonomous cross-country navigation with the alv. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 718–726, 1988.

[Dean and Wellman, 1991] T.L. Dean and M.P. Wellman. *Planning and Control.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Dickson, 1991] W. Dickson. *Image Structure and Model-based Vision.* PhD thesis, University of Oxford, 1991.

[Du and Brady, 1992] F. Du and M. Brady. Real-time gaze control for a robot head. In *Int. Conf. on Automation, Robotics, and Computer Vision*, Singapore, 15-18 September 1992.

[Dubins, 1957] L.E. Dubins. On curvers of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[Durrant-Whyte *et al.*, 1990] H.F. Durrant-Whyte, B.S.Y. Rao, and H. Hu. Towards a fully decentralized architecture for multi-sensor data fusion. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1331–1336, 1990.

[Durrant-Whyte, 1987] H.F. Durrant-Whyte. *Integration, Coordination, and Control of Multi-Sensor Robot Systems.* Kluwer Academic Press, Boston, MA., 1987.

[Durrant-Whyte, 1988] H.F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE J. Robotics and Automation*, 4(1):23–31, 1988.

[Elfes and Talukdar, 1983] A. Elfes and S.N. Talukdar. A distributed control system for the cmu rover. In *Int. Joint Conf. Artificial Intelligence*, page 830, 1983.

[Elfes, 1987] A. Elfes. Sonar-based real-world mapping and navigation. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 249–265, 1987.

[Elfes, 1990] A. Elfes. Ocucupancy grids: A stochastic spatial representation for active robot perception. In *Proc. of the 6th Conference on Uncertainty in AI*, July 1990.

[Faverjon and Tournassoud, 1987] B. Faverjon and P. Tournassoud. The mixed approach for motion planning: Learning global strategies from a local planner. In *Int. Joint Conf. Artificial Intelligence*, pages 1131–1135, 1987.

[Ferguson, 1967] T.S. Ferguson. *Mathematical Statistics—A Decision Theoretic Approach.* Academic Press, Inc, 1967.

[Flynn, 1972] M.J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. on Computers*, C-21(9):948–960, 1972.

[Flynn, 1988] A.M. Flynn. Redundant sensors for mobile robot navigation. *Int. J. Robotics Research*, 7(5), 1988.

[Fraichard, 1991] Th. Fraichard. Smooth trajectory planning for a car in a structure world. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 318–323, 1991.

[Fujimura and Samet, 1989] K. Fujimura and H. Samet. A hierarchical strategy for path planning among moving objects. *IEEE Trans. Robotics and Automation*, 5(1), 1989.

[Giralt *et al.*, 1984] G. Giralt, R. Chatila, and M. Vaisset. An integrated navigation and motion control system for autonomous multisensory mobile robots. In *First Int. Symp. Robotics Research*, page 191, 1984.

[Goto and Stentz, 1987] Y. Goto and A. Stentz. The c.m.u system for mobile robot navigation. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 99–105, 1987.

[Graettinger and Krogh, June 1989] T.J. Graettinger and B.H. Krogh. Evaluation and time-scaling of trajectories for wheeled mobile robots. *Transactions of the American Society of Mechanical Engineering*, 111:222–231, June 1989.

[Hager, 1988] G. Hager. *Active Reduction of Uncertainty in Multi-sensor Systems.* PhD thesis, University of Pennsylvania, 1988.

[Hague and Cameron, 1990] T. Hague and S. Cameron. Motion planning for the oxford agv. In *IEEE Int. Workshop on Intelligent Motion Control*, pages 277–282, Istanbul, 1990.

[Hague *et al.*, 1990] T. Hague, M. Brady, and S. Cameron. Using moments to plan paths for the oxford agv. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 210–215, 1990.

[Hansen, 1975] P.B. Hansen. The programming language concurrent pascal. *IEEE Transactions on Software Engineering*, SE-1(2):199–206, June 1975.

[Harmon, 1987] S.Y. Harmon. The ground surveillance robot (gsr): an autonomous vehicle designed to transit unknown terrain. *IEEE J. Robotics and Automation*, pages 266–279, 1987.

[Harmon, 1991] S.Y. Harmon. Architectures for real-time intelligent control systems: Concepts, approaches, methodologies. In *Record of the Working Group at the Workshop on Architectures for Intelligent Control Systems*, Marriott Key Bridge Hotel, Arlington, VA, August 1991.

[Hemami *et al.*, 1990] A. Hemami, M.G. Mehrabi, and R.M.H. Cheng. A new control strategy for tracking in mobile robots and agv's. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1122–1127, 1990.

[Hemami *et al.*, 1992] A. Hemami, M.G. Mehrabi, and R.M.H. Cheng. Synthesis of an optimal control law for path tracking in mobile robots. *Automatica*, 28(2):383–387, 1992.

[Herman and Albus, 1988] M. Herman and J.R. Albus. Overview of the multiple autonomous underwater vehicles (mauv) project. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 618–620, Philadelphia, 24-29 April 1988.

[Hoare, 1978] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hoare, 1985] C.A.R. Hoare. *Communication Sequential Processes.* Prentice Hall, London, England, 1985.

[Hollier, 1987] R.H. Hollier. *Automated Guided Vehicles.* IFS, London, 1987.

[Hu *et al.*, 1991a] H. Hu, M. Brady, and P. Probert. Coping with uncertainty in control and planning for a mobile robot. In *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, Osaka, Japan, 1991.

[Hu *et al.*, 1991b] H. Hu, M. Brady, and P. Probert. Navigation and control of a mobile robot among moving obstacles. In *IEEE the 30th Int. Conf. on Decision and Control*, Brighton, U.K., 1991.

[Hu *et al.*, 1991c] H. Hu, P. Probert, and M. Brady. Transputer architecture for real-time control of a mobile robot. In *Int. Conf. on Applications of Transputers 91*, Glasgow, U.K., 1991.

[Hu, 1990] H. Hu. Distributed architecture for sensing and control of a mobile robot. Technical Report OUEL 1836/90, Dept. of Engineering Science, University of Oxford, 1990.

[Hwang and Ahuja, 1992] Y.K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Trans. Robotics and Automation*, 8(1):23–32, February 1992.

[Hwang and Briggs, 1984] K. Hwang and F.A. Briggs. *Computer architecture and parallel processing.* McGraw-Hill, New York, 1984.

[Iida and Yuta, 1990] S. Iida and S. Yuta. Control of a vehicle subsystem for an autonomous mobile robot with power wheeled steering. In *IEEE Int. Workshop on Intelligent Motion Control*, pages 859–866, Istanbul, Turkey, 1990.

[Inmos, 1988] Limited Inmos. *OCCAM 2 Reference Manual.* Prentice Hall International(UK) Limited, Hertfordshire, U.K., 1988.

[Inmos, 1989a] Limited Inmos. *The Transputer Applications Notebook– Systems and Performance.* Redwood Burn Ltd., Trowbridge, U.K., 1989.

[Inmos, 1989b] Limited Inmos. *The Transputer Databook*. Redwood Burn Ltd., Trowbridge, U.K., 1989.

[Jarvis and Byrne, 1988] R.A. Jarvis and J.C. Byrne. An automated guided vehicle with map building and path finding capabilities. *Fourth Int. Symp. Robotics Research*, pages 498–504, 1988.

[Jones and Goldsmith, 1988] G. Jones and M. Goldsmith. *Programming in Occam 2*. Prentice Hall International(UK) Limited, Hertfordshire, U.K., 1988.

[Jun and Shin, 1988] S. Jun and K.G. Shin. A probablistc approach to collision-free robot path planning. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 220–224, 1988.

[Kaelbling, 1987] L.P. Kaelbling. An architecture for intelligent reactive systems. In M.P. Georgeff and A.L. Lansky, editors, *Reasoning about actions and plans*, pages 395–410. Morgan-Kaufmann, 1987.

[Kambhampati and Davis, 1986] S. Kambhampati and L.S. Davis. Multiresolution path planning for mobile robot. *IEEE J. Robotics and Automation*, RA-2(3):135–145, 1986.

[Kanayama and Yuta, 1988] Y. Kanayama and S. Yuta. Vehicle path specification by a sequence of straight lines. *IEEE J. Robotics and Automation*, 4(3):265–276, 1988.

[Kanayama *et al.*, 1988] Y. Kanayama, A. Nilipour, and C.A. Lelm. A locomotion control method for autonomous vehicles. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1315–1317, 1988.

[Kanayama *et al.*, 1990] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. A stable tracking control method for an autonomous mobile robot. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 384–389, 1990.

[Kant and Zucker, 1986] K. Kant and S.W. Zucker. Toward efficient planning: The path-velocity decomposition. *Int. J. Robotics Research*, pages 72–89, 1986.

[Keieriemen and von Puttkamer, 1991] T. Keieriemen and E. von Puttkamer. Real-time control in an autonomous mobile robot. In G. Schmidt, editor, *Information Processing in Autonomous Mobile Robots*, pages 187–200. Springer-Verlag, 1991.

[Khatib, 1986] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Int. J. Robotics Research*, volume RR5:1, pages 90–98, 1986.

[Klein and Wahawisan, 1982] C.A. Klein and W. Wahawisan. Use of a multiprocessor for control of a robotic system. *Int. J. Robotics Research*, 1(2):45–59, 1982.

[Koren and Borenstein, 1991] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1398–1404, Sacramento, California, 1991.

[Kotzev *et al.*, 1992] A. Kotzev, D.B. Cherchas, P.D. Lawrence, and N. Sepehri. Generalised predictive control of a robotic manipulator with hydraulic actuator. *Robotica*, 10:447–459, 1992.

[Kuc and Viard, 1991] R. Kuc and V.B. Viard. A physically based navigation strategy for sonar-guided vehicles. *Int. J. Robotics Research*, 10(2):75–87, April 1991.

[Kuipers and Byun, 1988] B.J. Kuipers and Y.T. Byun. A robust, qualitative method for robot spatial learning. In *Proc. of AAAI-88*, pages 774–779, 1988.

[Lang *et al.*, 1989] S.Y. Lang, L.W. Korba, and A.K. Wong. Characterizing and modelling a sonar sensor. In *Proc. of SPIE*, pages 291–304, 1989.

[Laumond *et al.*, 1990] J-P. Laumond, M. Taix, and P. Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *Proc. of Int. Workshop on Intelligent Robots and Systems*, pages 765–773, Japan, 1990.

[Laumond, 1986] J-P. Laumond. Feasible trajectories for mobile robots with kinematic and environment constraints. In *Proc. of Int. Conf. on Intelligent Autonomous Systems*, Amsterdam, Netherland, 1986.

[Laumond, 1987] J-P. Laumond. Finding collision-free smooth trajectories for a non-holonomic mobile robot. In *Proc. of Int. Joint Conf. on Artificial Intelligence*, pages 1120–1123, Milan, Italy, 1987.

[Laumond, 1991] J-P. Laumond. Controllability of a multibody mobile robot. In *Proc. of 5th Int. Conf. on Advanced Robotics*, pages 1033–1038, Pisa, Italy, 19-22 June 1991.

[Lawton *et al.*, 1990] D.T. Lawton, R.C. Arkin, and J.M. Cameron. Qualitative spatial understanding and reactive control for autonomous robots. In *IROS 90*, pages 709–714, Japan, 1990.

[Leonard and Durrant-Whyte, 1990] J.J. Leonard and H.F. Durrant-Whyte. Application of multi-target tracking to sonar-based mobile robot navigation. In *Proc. the 29th IEEE Conference on Decision and Control*, pages 3118–3123, Hawaii, USA, 1990.

[Leonard and Durrant-Whyte, 1992] J.J. Leonard and H.F. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation.* Kluwer Academic Publishers, 1992.

[Lin and Yau, 1967] T. Lin and S.S. Yau. Bayesian approach to the optimisation of adaptive systems. *IEEE Trans. Systems Science and Cybernetics*, SSC-3(2):77–85, 1967.

[Lozano-Pérez, 1983] T. Lozano-Pérez. Spatial planning–a configuration space approach. *IEEE Transactions on Computers*, pages 108–120, February 1983.

[Manyika *et al.*, 1991] J.M. Manyika, I.M. Treherne, and H.F. Durrant-Whyte. A modular architecture for decentralised sensor data fusion: A sonar-based sensing node. In *Proc. of the 2nd Int. Workshop on Sensor Fusion and Environmental Modelling*, pages 1–15, Oxford, UK, 2-5, September 1991.

[Mitchell, 1990] T.M. Mitchell. Becoming increasingly reactive. In *Proc. 8th National Conf. on Artificial Intelligence*, pages 1051–1058, 1990.

[Moravec and Elfes, 1985] H.P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 116–121, 1985.

[Nelson and Cox, 1988] W.L. Nelson and I.J. Cox. Local path control for an autonomous vehicle. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1504–1510, 1988.

[Nelson, 1989a] W.L. Nelson. Contimuous steering-function control of robot carts. *IEEE Trans. Industrial Electronics*, 36(3):330–337, 1989.

[Nelson, 1989b] W.L. Nelson. Continuous-curvature paths for autonomous vehicles. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1260–1264, 1989.

[Newman and Hogan, 1987] W.S. Newman and N. Hogan. High speed robot control and obstacle avoidance using dynamic potential functions. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 14–24, 1987.

[Nilsson, 1984] N. Nilsson. Shakey the robot. Technical Report 323, SRI, April 1984.

[Noriels and Chatila, 1989] F.R. Noriels and R.G. Chatila. Control of mobile robot actions. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 701–707, 1989.

[Oommen and *et al.*, 1987] B.J. Oommen and *et al.* Robot navigation in unknown terrains using learned visibility graphs. part 1: The disjoint convex obstacle case. *IEEE J. Robotics and Automation*, RA-3(6):672–681, 1987.

[Orlando, 1984] N.E. Orlando. An intelligent robotics control scheme. In *American Control Conference*, page 204, 1984.

[Pan and Luo, 1990] T.J. Pan and Ren C. Luo. Motion planning for mobile robots in a dynamic environment with moving obstacles. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 578–583, 1990.

[Paul *et al.*, 1985] R.P. Paul, H.F. Durrant-Whyte, and Max Mintz. A robust distributed robot control system. In *Third Int. Symp. Robotics Research*, France, 1985.

[Pin and Vasseur, 1990] F.G. Pin and H.A. Vasseur. Autonomous trajectory generation for mobile robots with non-holonomic and steering angle constraints. In *IEEE Int. Workshop on Intelligent Motion Control*, pages 295–299, Istanbul, 1990.

[Pountain and May, 1988] Dick Pountain and David May. *A Tutorial Introduction to Occam Programming*. BSP Professional Books, 1988.

[Premi and Besant, 1983] S. Premi and C. Besant. A review of various vehicle guidance techniques that can be used by mobile robots or agvs. In *Proc. 2nd Int. Conf. on Automated Guided Vehicle Systems*, pages 195–209, Stuttgart, Germany, 1983.

[Rice, 1988] J.A. Rice. *Mathematical Statistics and Data Analysis*. Wadsworth and Brooks/Cole Advanced Books and Software, Pacific Grove, California, 1988.

[Rugby, 1986] GEC Rugby. Lcs2 road system user guide. Technical Report T299 Issue3 12.85, GEC Industrial Controls Ltd., 1986.

[Rugby, 1987] GEC Rugby. The oxford university agv. Technical report, GEC Electrical Projects Ltd, Factory Automation Division, 1987.

[Rugby, 1988] GEC Rugby. The gec agv control system. Technical Report MTR Issue B, GEC Electrical Projects Ltd, Factory Automation Division, 1988.

[Samson, 1992] C. Samson. Path following and time-vering feedback stabilization of a wheeled mobile robot. In *Proc. Int. Conf. on Automation, Robotics, and Computer Vision*, Singapore, 16-18 September 1992.

[Sarkar, 1989] V. Sarkar. *Partitioning and scheduling parallel programs for multiprocessors.* Pitman publishing, 1989.

[Scharf, 1990] L.L. Scharf. *Statistical Signal Processing–Detection, Estimation, and Time Series Analysis.* Addison-Wesley Publishing Company, 1990.

[Sedgewick, 1988] R. Sedgewick. *Algorithm (The 2nd Edition).* Addision-Wesley Publishing Company, 1988.

[Singh and Wagh, 1987] J. S. Singh and M.D. Wagh. Robot path planning using intersecting convex shapes: Analysis and simulation. *IEEE J. Robotics and Automation*, RA-3(2):101–108, April 1987.

[Sordalen and de Wit, 1992] O.J. Sordalen and C.C. de Wit. Exponential control law for a mobile robot: extension to path following. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2158–2163, 1992.

[Steer, 1989] Barry Steer. Trajectory planning for a mobile robot. *Int. J. Robotics Research*, pages 3–14, 1989.

[Stevens *et al.*, 1983] P. Stevens, M. Robins, and M. Roberts. Truck location using retroreflective strips and triangulation with laser equipment(turtle). In *Proc. 2nd European conf. on automated manufacturing*, Birmingham,UK, 1983.

[Sun, 1988] Microsystem Sun. Sunview 1: Programmer's guide. Technical report, Digital Equipment Corporation, 1988.

[Takahashi and Schiling, 1989] O. Takahashi and R.J. Schiling. Motion planning in a plane using generalized voronoi diagram. *IEEE Trans. Robotics and Automation*, 5(2):143–150, 1989.

[Takeda *et al.*, 1986] T. Takeda, A. Kato, T. Suzuki, and M. Hosoi. Automated vehicle guidance using spotmark. In *Proc. IEEE Int. Conf. Robotics and Automation*, page 1346, 1986.

[Thompson, 1977] A.M. Thompson. The navigation system of the JPL robot. In *Proc. 5th IJCAI*, 1977.

[Tournassoud and Jehl, 1988] P. Tournassoud and O. Jehl. Motion planning for a mobile robot with a kinematic constraint. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1785–1790, 1988.

[Tsumura *et al.*, 1981] T. Tsumura, T. Fujiwara, and M. Hashimoto. An experimental system for automatic guidance of robot vehicle, following the route stored in memory. In *Proc. 11th Int. Symposium on Idustrial Robots*, pages 187–193, October 1981.

[Tsumura, 1986] T. Tsumura. Survey of automated guided vehicle in japanese factory. In *Proc. IEEE Int. Conf. Robotics and Automation*, page 1329, 1986.

[Turchan and Wong, 1985] M.P. Turchan and A.K.C. Wong. Low level learning for a mobile robot: environment model aquisition. In *Proc. of the 2nd Int. Conf. on AI application*, Miami, Florida, 1985.

[Walter, 1987] S.A. Walter. The sonar ring: Obstacle detection for a mobile robot. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1574–1579, 1987.

[Warren, 1989] C. W. Warren. Global path planning using artificial potential fields. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 316–610, 1989.

[Zhao and BeMent, 1990] Y. Zhao and S.L. BeMent. A heuristic search approach for mobile robot trap recovery. In *in Proc. SPIE*, pages 122–130, 1990.