DESIGN OF AN OBSTACLE AVOIDANCE SYSTEM FOR AUTOMATED GUIDED VEHICLES

Stephen Lockwood

A thesis submitted to the University of Huddersfield in Partial fulfillment of the requirements for the degree of Doctor of Philosophy.

October 1992

The University of Huddersfield in collaboration with the Holset Engineering Company Limited.

CONTENTS

ACKNOWLEDGEMENTS

I would like to express my gratitude to all in the School of Engineering at the University of Huddersfield and AMECAS for their help and friendship throughout the research project.

Thanks in particular to Dr Bruce Mehrdadi and Dr Jeff Chandler (Directors of Studies) for their technical input and editorial assistance in the preparation of the thesis. Their relentless encouragement, support and sense of humour have been invaluable. Special thanks are also extended to Dr Mike Freeman (Supervisor) for his technical advice and for proof reading the thesis. His feedback has been most helpful and constructive.

I would also like to express my gratitude to all the Technicians in the Engineering Systems Division for their practical help during the development of the hardware for the research.

Finally I would like to thank my partner Jacinta. I owe her much for her help in preparing the thesis, and more for her unwavering love and support.

ABSTRACT

Most Industrial Automated Guided Vehicles (AGVs) follow fixed guide paths embedded in the floor or bonded to the floor surface. Whilst reliable in their basic operation, these AGV systems fail if unexpected obstacles are placed in the vehicle path. This can be a problem particularly in semi-automated factories where men and AGVs share the same environment.

The performance of line-guided AGVs may therefore be enhanced with a capability to avoid unexpected obstructions in the guide path. The research described in this thesis addresses some fundamental problems associated with obstacle avoidance for automated vehicles.

A new obstacle avoidance system has been designed which operates by detecting obstacles as they disturb a light pattern projected onto the floor ahead of the AGV. A CCD camera mounted under the front of the vehicle senses obstacles as they emerge into the projection area and reflect the light pattern.

Projected light patterns have been used as an aid to static image analysis in the fields of Computer Aided Design and Engineering. This research extends these ideas in a real-time mobile application. A novel light coding system has been designed which simplifies the image analysis task and allows a low-cost embedded microcontroller to carry out the image processing, code recognition and obstacle avoidance planning functions.

An AGV simulation package has been developed as a design tool for obstacle avoidance algorithms. This enables potential strategies to be developed in a high level language and tested via a Graphical User Interface. The algorithms designed using the simulation package were successfully translated to assembler language and implemented on the embedded system. An experimental automated vehicle has been designed and built as a test bed for the research and the complete obstacle avoidance system was evaluated in the Flexible Manufacturing laboratory at the University of Huddersfield.

LIST OF FIGURES

1 INTRODUCTION

1.1 Automated Guided Vehicles and Factory Automation

The first Automated Guided Vehicles (AGVs) were developed in the 1950s by Barrett Electronics of the USA for use in warehouses[1]. These were automated trucks towing trains of carts, guided by a wire system similar to that still used in many installations today.

In 1990, the worldwide AGV market was worth about £2500 million having grown at an annual rate of 12-15% in the previous decade[2]. This growth rate has led to the appearance of AGVs in many guises ranging in complexity from basic units similar to the original Barrett vehicles, to highly 'intelligent' legged automatons designed for such tasks as extra-terrestrial exploration and hazardous environment inspection. In general, the cost of automated vehicles is reflected in their performance. Complex and expensive vehicles are used in applications where cost is secondary to safety or research, whereas more basic types are employed in warehouses, offices and factories where commercial viability is essential.

Only recently have automated vehicles been used widely in the manufacturing industry. This is partly due to advancing computer technology and its falling cost, and also, partly as a result of changing trends in industrial administration. In particular the adoption of management philosophies such as Just In Time (JIT) coupled with the concept of Flexible Manufacturing Systems (FMS) provide ideal environments for AGVs[3].

The benefits brought about by automated vehicle systems include reduced labour costs since drivers are not required, reduced paper work in the form of dockets and

requisitions and the fact that work need not be interrupted by rest periods. A further major advantage of automated vehicles as opposed to traditional forms of factory transport such as conveyors and railways, is that they allow more efficient use of the available factory space. This is made possible by the fact that AGVs use unobtrusive or hidden guide-paths that are only apparent when vehicles are actually present. At other times, the thoroughfares can be used by people and other transport. Conveyors and railways on the other hand, require exclusive routes constructed from intrusive steel work and transport equipment.

1.2 Research Objectives

The research presented in this thesis has been carried out in collaboration with AMECAS Ltd., (Advanced Manufacturing Equipment Control and Automation Systems), a trading division of the Holset Engineering Company Ltd., Huddersfield. This division was formed as a small consultancy following the award winning in-house implementation of a Flexible Manufacturing System at the Holset turbo-charger manufacturing plant in Huddersfield.

The subject of the research was highlighted at an AMECAS Automated Guided Vehicle installation in Doncaster, England. This factory is approximately 1/4 mile long and several hundred metres wide. The AGV system is a wire-guided network of pallet transporters operating along side manual trucks.

As mentioned in the previous section, a major benefit of AGVs is that they can share space with people. However, as in the case of the Doncaster installation, a problem occurs where automatic vehicles and similar manually operated vehicles are doing the same job in the same work space. AGVs can not operate with the same level of flexibility as manual truck drivers and require that their guide path be completely unimpeded. In practice, objects such as untidily stacked pallets or bins are inevitably

placed in the AGV guide-path, which in turn leads to automated vehicles becoming stranded. In a factory the size of the Doncaster installation, this is a severe problem as the time taken to find and travel to stranded vehicles to clear obstructions can be significant. This affects work schedules and hence disrupts production planning.

The problem of obstacle avoidance is associated with the guidance and navigation of automated vehicles and two main solutions are possible:

- either completely replace the existing AGV system with a new one that has the capability to avoid unexpected obstacles,
- or use an obstacle avoidance system that can be retrofitted to existing vehicles.

As yet, no AGV system that can avoid unexpected obstacles is commercially available. Even if such a system did exist, organisations would be unlikely to be persuaded to reinvest heavily in a completely new installation. Rather, they would prefer to upgrade existing vehicles with 'add on' obstacle avoidance units.

Much research has been carried out in the field of obstacle avoidance in attempts to provide commercially viable systems. This is based on a variety of sensor systems and techniques including ultrasound ranging systems, CCD camera systems, laser range finders and combinations of these. The research presented in this thesis is concerned specifically with the rigorous requirements of the manufacturing industry and with the high demands that such harsh operating environments place on any practicable design. For example, a suitable system should not use moving parts since these would be subject to the eventual ingress of dirt and wear resulting in a degradation of performance. This factor rules out the use of delicate mechanical sensors such as rotating scanners and moving cameras etc.

Cost is a crucial factor affecting the acceptance of products by competitive industry. Hence any retrofitting obstacle avoidance system must be low-cost. Not only in terms

of its relation to the value of the host vehicle, but also in its operation and maintenance. This implies that the hardware must be relatively simple to install and configure, preferably without the need for special calibration equipment.

A further constraint on the design of a retrofitting obstacle avoidance system is its physical size. Although commercial automated vehicles come in many shapes and sizes, an add-on system must be compact and light-weight enough to be relatively unobtrusive when installed on the host vehicle. This excludes the use of relatively large standard computers and suggests that a design based on compact, single-chip microcontrollers is desirable.

Many automated vehicle systems described in the literature require detailed on-board 'knowledge' of the factory layout in order to perform obstacle avoidance tasks. These tend to be application-specific and are not suited to general use. Although obstacle avoidance is often treated as a completely separate issue to that of obstacle detection, this research seeks to combine both detection and avoidance into a practical system that is suitable for general applications.

The objectives discussed in the previous paragraphs have been translated into an innovative obstacle avoidance system. This is based on a novel light pattern projection system and a charged Coupled Device (CCD) imaging system. The light pattern is projected onto the floor ahead of the automated vehicle and is not normally visible to the CCD camera mounted under the front of the chassis. However, if an obstacle emerges, the projected light pattern becomes visible to the camera, and the obstruction is detected. The system then controls the vehicle drives to circumnavigate the obstruction and rejoin the original guide path.

The complete obstacle avoidance system has been appraised in the Flexible Manufacturing Laboratory at the University of Huddersfield. Extensive tests have been carried out to confirm the design specifications in terms of real-time operation, the

smallest detectable object, and the diverse range of obstructions that can be detected. Several examples of the system detecting obstacles typical of those likely to be found in factories are included. The performance of the obstacle avoidance system is assessed in terms of the accuracy with which it returns to the original guide path. The accuracy with which it repeats a route given equivalent conditions and its ability to avoid multiple obstacles are also presented

2 BACKGROUND TO THE RESEARCH

2.1 Summary

This chapter explores the background to the problem of obstacle avoidance. A brief overview of the guidance methods available for Automated Guided Vehicles (AGVs) and mobile robots is presented. Options for solving the obstacle avoidance problem are discussed and a solution is identified for further research. Relevant work by other researchers is reviewed which highlights the key difficulties associated with detecting and negotiating unexpected obstacles. This review confirms that further research work is necessary towards the design of a low-cost, reliable obstacle avoidance system for industrial automated guided vehicles.

2.2 An Introduction to the Guidance Methods Available for Automated Vehicles

AGVs are guided using two basic systems:

- Line following. This is currently the most common method and has been available since the birth of AGV technology
- Free ranging. Advanced systems which are only just starting to be offered commercially.

2.2.1 Line Following Techniques

Line following is the most common method of automated vehicle guidance[4], characterised by the fact that AGVs follow paths physically marked on the ground. Either passive or active techniques can be used, depending on the application and the operating environment[5].

2.2.1.1 Passive Line Guidance

Passive systems are common in relatively clean environments where AGV guide lines are not vulnerable to hard wear and dirt. Various methods are used for marking and detecting AGV guide paths depending on the particular design. An excellent review of these and other methods of line guidance has been presented by Premi and Besant[6].

White line following systems use sensors consisting of infrared transmitters and receivers to detect the presence and position of white or brightly coloured lines painted or taped onto a dark floor, (or dark lines on light coloured floors). The position of a line in relation to the sensors is used to derive control signals for the AGV drive motors[7].

'Littons Patented Optical System'[1] uses a fluorescent compound to mark AGV guide lines, which is invisible to the human eye under normal circumstances. Automated vehicles are equipped with sensing heads consisting of ultra-violet lamps and optical detectors which irradiate the chemical compound and detect its position. Control actions are derived from the sensing heads in a similar fashion to white line following systems.

Metal line systems differ from optical methods in that a metal adhesive tape is used to mark AGV guide lines. This has the advantage that it can be concealed under carpets or other floor coverings. Sensors on board AGVs (similar to metal detectors) are used

to detect the metal tape and derive control signals for the vehicle drives.

2.2.1.2 Active Line Guidance

Active line guided systems are by far the most commonly used in factories, particularly where the environment is too hostile for systems requiring paint or tape on the floor surface. In active systems, AGV guide paths are marked by embedding large wire loops 1 or 2cm below the floor surface which are driven by AC signals. Automated vehicles equipped with pick-up coils sense the signals, the amplitude of which give a measure of the proximity of the embedded wire and are used to derive steering control. Each embedded guide loop is driven by a different frequency (typically in the range 1-15KHz) allowing AGVs to navigate between regions by using band pass filters to select the required loop frequency and reject all others.

2.2.1.3 Other Line Following AGV Guidance Methods

Other methods of AGV guidance which broadly fall into the category of passive line following have been devised. In particular, Tsukagoshi, Miura and Yamauchi of Japan have described a guidance system which uses ferrite tiles to construct lanes that cross each other in a lattice arrangement[8]. AGVs follow the ferrite lanes using magnetic sensors.

A variation on passive line following has been designed for a clean-room inspection robot[9,10,11]. This uses spot reflectors embedded in the floor at discrete intervals which are illuminated using a powerful infrared lamp. Reflections from the spot marks are detected using an infrared CCD camera. Steering control is derived from the position of the spot reflections in the CCD image.

In general, passive line guided systems are not robust enough for harsh factory

environments because paint or foil tape can easily become worn away, and white lines or spot reflectors can be obscured by dirt and debris. Also, white line following systems depend on a high contrast between the line and the surrounding floor. However, active embedded wire loops, are not subject to wear and debris and as a result, have been installed widely in the manufacturing industries.

2.2.2 Free Ranging AGV Navigation

Free-ranging automated guided vehicles do not follow physical guide lines. Most systems are based on odometry[12] (or dead reckoning) where vehicle position and heading are derived from incremental encoders coupled to the AGV road wheels and steering gear. Examples of this type of system can be seen in references[13,14,15]. The main disadvantage of AGV navigation using odometry is its inaccuracy[12,16]. If the wheel diameters are not accurately matched, the floor not perfectly flat, or the wheels pick up debris, errors accumulate in proportion to the distance travelled and can become unacceptably large. This error may be reduced by combining odometry with other methods of navigation. This approach has been adopted by Stephens Robins and Roberts in their 'TURTLE' system which uses scanning lasers and optical sensors to detect reflectors strategically positioned in known locations around the AGV environment[17]. The angular bearing of the reflectors is used to triangulate the absolute position of the vehicle, and this information is used to correct the accumulated odometry error. Similar triangulation systems have also been designed which use infrared beacons and detectors[18] and ultrasonic transponders and targets[19].

An alternative method of dead reckoning navigation based on inertial techniques has been considered by Eaton-Kenway Incorporated of the USA[20]. However, the cost of mechanical or optical gyro sensors is high and as Tsumura points out[21], the technique suffers from accumulative position errors in the same way as odometry.

Systems which do not use any form of dead reckoning guidance are based on ultrasonic ranging[22] and imaging techniques[23,24,25] which attempt to build range measurement maps of the AGV environment for navigation. Other systems employ optical systems to achieve the same ends and also to recognise previously described objects in the AGV environment[26,27].

Other work on free-ranging AGV navigation systems has tended towards the integration of various sensing techniques in attempts to achieve high reliability and accuracy[28,29,30].

2.3 Review of Obstacle Avoidance Research

An obvious solution to the problem of obstacle avoidance would be to prevent objects being placed in the path of automated vehicles. However, this undermines a major benefit of AGV systems which is the sharing of common thoroughfares. In an integrated environment, obstacles could only be totally prohibited from the working environment by making the guide path exclusive to AGVs.

Alternatively, the whole wire guidance system could be replaced by a free-ranging scheme with the means to avoid unexpected obstacles. Although considerable research effort continues to be devoted to the design of free-ranging AGV systems the problems which must be overcome to make such systems commercially viable are manifold. These include high cost, high complexity, low speed of operation and low reliability. Only one company has succeeded in commercially marketing a free-ranging AGV system (based on the previously mentioned TURTLE laser range finding system[17,31]), and that does not overcome the crucial problem of obstacle avoidance.

2.3.1 Architectures For Obstacle Avoidance

Most current research on obstacle avoidance uses subsumption architecture[32] which stems from work of biological origin[33]. Subsumption architecture is a hierarchical control structure with various priority levels. Each level of control has a set of conditions which once met, triggers its action. A characteristic of the architecture is that when a level of control is activated, all lower levels are inhibited. Figure 2.3.1.1 shows an example of an AGV hierarchical control structure incorporating three levels:

1. hardware safety devices
2. obstacle avoidance
3. navigation.

Level 3 is a general navigation control system which can be inhibited by an obstacle avoidance control system should an unexpected obstacle be detected (level 2). Level 2 can in turn be inhibited by the hard wired safety bumper (level 1) in the event of an object being physically touched by the Automated Guided Vehicle.

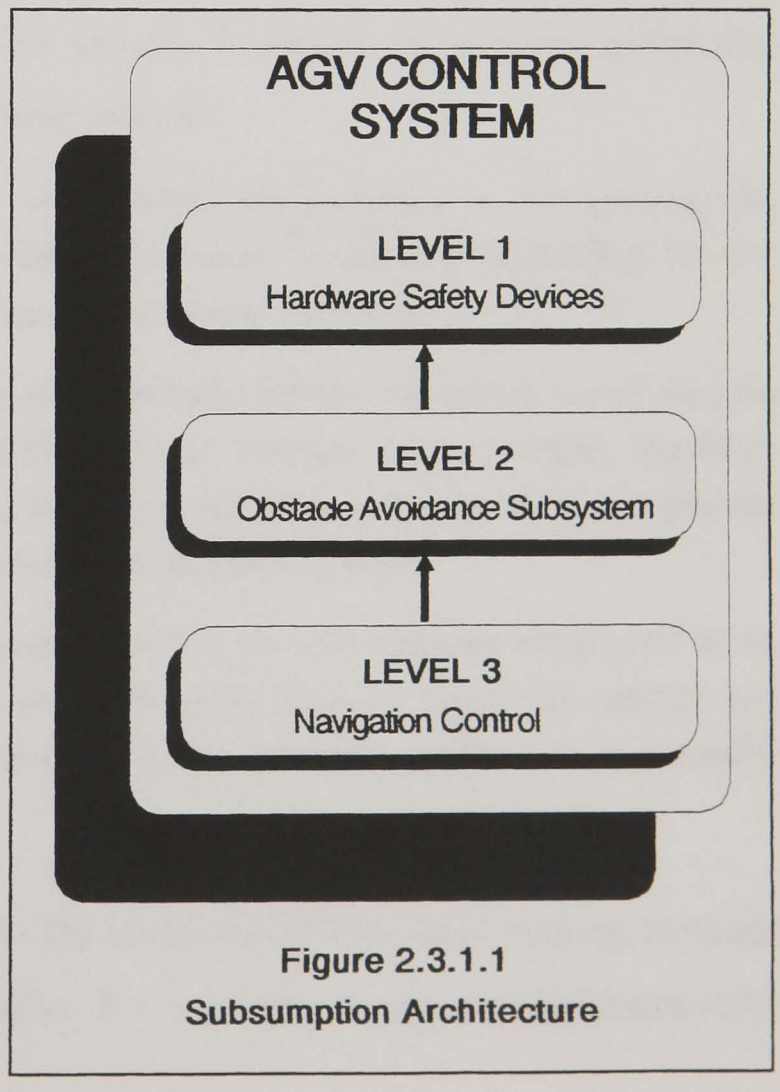


Figure 2.3.1.1
Subsumption Architecture

2.3.2 Sensors and Systems Used for Obstacle Avoidance

Work on obstacle avoidance is almost exclusive to free ranging AGV systems since, to avoid unexpected obstacles, the automated vehicle must deviate from its planned path, circumnavigate the obstruction and

either rejoin, or revise the original path. In order to achieve this, the AGV must have some notion of where it is and where it is allowed to go on a new trajectory. This knowledge is usually derived from a software model (or map) of the environment with which the AGV compares its current perceived position[34,35,36].

Although much work has been carried out on collision free path planning[37,38,39], a large emphasis is now placed on sensing as the key problem[34,36] and in particular achieving reliable obstacle detection in real-time.

The sensors used in obstacle detection are either acoustic or optical and in some cases both[40].

2.3.2.1 Ultrasonic Systems

The relatively modest information processing requirements of most ultrasonic ranging systems can be met in real-time but their success is limited by inherent difficulties associated with acoustics[34,35,40,41,42,43,44]. These include:

- Poor directionality which limits the accuracy in determining the spatial position of edges to about 10-50 cm depending on the distance of the target object from the sensor.
- Inaccuracies in distance measurements can easily occur due to ultrasonic noise from external sources (for example, machine tools, neighbouring sensors, hand tools being used or dropped on the floor etc.) and also by multi-path echoes.
- Specular reflections occur when smooth surfaces are placed at an angle to the sound source[41]. This leads to either the surface not being detected at all or at best, appearing smaller than it actually is.

Systems have been designed to overcome the shortfalls of ultrasonic ranging methods by integrating them with other technologies. For example, Evans and Krishnamurthy

et al[40] combine ultrasonic techniques with optical methods to navigate a 'health care service' robot. An integrated system has been described by Hollingham[45], which uses twelve combined SONAR and infrared sensing heads distributed around the periphery of an automated vehicle. Each Transputer based sensing head can be rotated by a small stepper motor in order to scan its locality. A further transputer is used for overall control of the system. Although these sensors may be useful in some automated environments, their cost and delicate nature makes them unsuitable for most factory based AGVs.

2.3.2.2 Optical Systems

Optical sensors used in obstacle avoidance systems range from laser range finders to CCD cameras and in general, present higher demands in terms of information processing because of their higher resolution. A system which uses both a colour video camera and a laser range finder[46] has been used to guide an Autonomous Land Vehicle (ALV) along outdoor roadways[47]. The system uses the colour camera to detect road edges and the laser range finder to detect objects within the road edges. Image processing is carried out by a powerful host computer and two dedicated digital image processors. Range measurements are processed by a programmable systolic array (the so-called 'warp machine')[48]. Although this system is able to operate in real time, (the warp machine has an array of 10 cells, each with a processor operating at 10 million floating point operations per second), the sheer size and cost of the hardware makes it impractical in commercial industrial applications.

An obstacle avoidance system using stereo CCD video cameras has been described in a multi-level architecture for vision based navigation[49]. This system employs a powerful multiprocessor computer work station and fast image processors to compare the stereo disparity from the CCD camera images with that of a previously computed stereo disparity map of the workspace floor. Although this obstacle avoidance system

operates in real time, the large multiprocessor work station is not carried on board the mobile robot and the predetermined stereo disparity map of the ground floor is only suitable for laboratory environments. For example, the system would fail if a new pattern was introduced on the floor due to shadows, or spilt liquid, etc.

An alternative obstacle detection system has been described by Takeno and Hachiyama[50], which proposes a new technology for processing stereo images called the 'Laminated Difference Method'. However, this system does not avoid obstacles, it only detects them and under controlled test conditions the system required four seconds to process one pair of images. This performance is too slow for use on industrial Automated Guided Vehicles.

Stereo Obstacle detection systems require precise relative positioning of the cameras in order to accurately calculate the position of features in the stereo image pair. An obstacle avoidance system which uses a single, less critically positioned camera has been described by Takeuchi, Enemoto and Nagai of Japan[51]. In this system a monochrome CCD camera is mounted approximately 1 metre from the ground on top of a mobile robot and focused on the floor ahead of the vehicle at an angle of 30 degrees from vertical. The obstacle detection system works on the principal that when the scene in front of the mobile robot is illuminated (by two powerful lamps), changes in grey level occur in the CCD image due to the boundaries between the plain floor and potential obstacles. The CCD image is processed to determine the positions of such grey level changes and a ''fuzzy'' controller is implemented to execute collision free motion[52]. This system will operate successfully in controlled light conditions on matt, uniformly coloured and textured surfaces. However, since the image processing system takes no account of the true three dimensional nature of the robot environment, it cannot distinguish between true obstacles and flat features on the floor such as pieces of paper, changes in colour, or spilt liquids. Also, the system is easily deceived by reflections from surrounding objects and glare from overhead lights.

2.3.3 The Need for a New Obstacle Avoidance System

A cost-effective approach to the obstacle avoidance problem is to design a modular obstacle avoidance sub-system auxiliary to existing AGVs which does not require modification to existing guidance networks. Enhanced performance of the overall system would be achieved by enabling AGVs to negotiate unexpected obstacles.

AGV users who have invested heavily in line guided navigation systems are unlikely to be persuaded to reinvest in free ranging systems to solve the problem of obstacle avoidance. However, the prospect of improving the efficiency of an existing system by installing a low cost modular obstacle avoidance unit is attractive. Existing conventionally guided AGV systems may not have the ''intelligence'' to avoid obstacles, but they are reliable if it is accepted that they will fail due to unexpected obstructions. A low cost, modular obstacle avoidance system would not therefore degrade the operation of an existing system; but it would enhance it by allowing AGVs to avoid such objects in the guide path. A completely new, free-ranging system has a high risk of for ''teething'' problems and may prove less reliable during normal, obstacle free navigation.

The review of obstacle avoidance systems presented in this chapter has not revealed a low cost and reliable system which is suitable for integrating into the control hierarchy of existing AGV navigation systems.

For these reasons, the work presented in this thesis builds on that carried out by Lockwood, Mehrdadi and Chandler[53], and is aimed at the design of a low cost, modular system which can be retrofitted to existing conventional AGVs.

The single camera system described in the review[51,52] has provided a valuable preface to the study of obstacle avoidance and using it as a starting point, this research is based on the following hypotheses:

- The geometry of the optical system can be redesigned to enable effective discrimination between true three dimensional objects and two dimensional disturbances on the floor.
- Illumination coding techniques can be developed to simplify the image processing task and enable reliable obstacle detection.
- A low cost, self-contained system can be realised by employing single chip microcomputers for image processing and steering control.

3 DESIGN OF THE LOW COST OBSTACLE DETECTION SYSTEM

3.1 Summary

Light patterns have been used in applications such as metrology, Computer Aided Engineering (CAE) and Computer Aided Design (CAD) to increase the information content in optical systems. However, so called structured light is normally used in static image analysis such as off-line inspection and 3-D surface measurement.

This chapter reviews relevant research works based on projected light patterns and introduces the concept of extending such systems for mobile use. Sections 3.3, 3.4 and chapters 4,5 and 6 discuss the design of a novel obstacle detection system which is low cost, simple to configure, self-contained and capable of operating in real-time.

3.2 The Application of Structured Light for Measurement and Detection

Lighting with known geometrical properties that are used to obtain information in illuminated scenes is referred to as 'structured' light[54]. A 3-D machine perception system which uses a standard slide projector and binary coded slide mask has been described[55]. The projector illuminates objects with a coded pattern (rather like a 'chess board') and uses a CCD camera and image processing system to determine the range of uniquely coded groups of light points. The surface of objects can be modelled from these range data[56]. Two important features of this system are:

- The system has potential for use in mobile applications because

it requires only a single 'snap shot' image to completely analyse a scene. This overcomes a problem called 'smudging' caused by successive frame integration which can occur in mobile systems that need multiple images[55].

- The hardware is low-cost since a standard slide projector and single camera are used (ie. range measurement can be achieved with a single camera as opposed to triangulation with a stereo pair).

However, a major disadvantage of the 3-D machine perception system is the number of computations required. As the authors point out, real-time operation could only be achieved if the computations were speeded up by a factor of at least 300. This makes the current system unsuitable for automated guided vehicles.

A 3-D measurement system has been designed using a similar projector but with a uniform dot-grid mask rather than a binary code[57]. This system uses a CCD camera which moves laterally on a sliding carriage. Images are acquired with the camera in two positions and range measurements are derived from the relationship of the projected dots in each image (rather like triangulating with stereo cameras). This system has the disadvantages that it uses moving parts which will eventually need service, and the image acquisition and processing procedures are too slow for mobile applications. For example, if the moving camera was mounted on a vehicle which was itself moving, the relationship between a pair of successive images would be extremely complicated unless the precise relative motion of the vehicle was taken into account (the 'smudging' problem described above).

Alternative light sources can be used to generate structured light. A system using coherent laser light has been designed which acquires range and surface information from two photographs of the same object with different laser light patterns[58]. Surface information is extracted by projecting a vertical grid of equidistant lines onto a subject and analysing the image distortion. The system has the disadvantages that multiple

images are required and image processing is carried out off line. Moreover, in monochrome laser light systems, surfaces with a narrow colour band will only reflect certain colours of light[59]. This problem does not occur when 'white' light, produced for example by tungsten-halogen light sources is used because its spectrum contains components of many colours. Detectable levels of white light are therefore reflected from a wide range of surfaces.

The following design work draws on the research carried out for the machine perception system described earlier[55,56]. A novel light pattern has been developed which allows the image processing task to be simplified in order to increase the speed of operation. The light generation and projection system is based on a white light source and adopts a single 'snap-shot' approach suitable for mobile systems.

3.3 Method of Light Generation and Projection

A standard slide projector is used to generate white light illumination. This allows experiments to be easily carried out using projection masks housed in a standard 35mm photographic slide format. Figure 3.3.1 shows a diagram of the lens system. The geometry of the light projector and CCD camera (described in the next chapter) are illustrated in figure 3.3.2. The aim of the system is to project a light pattern onto the floor ahead of the automated guided vehicle, and detect the distortion of the light pattern caused by objects emerging

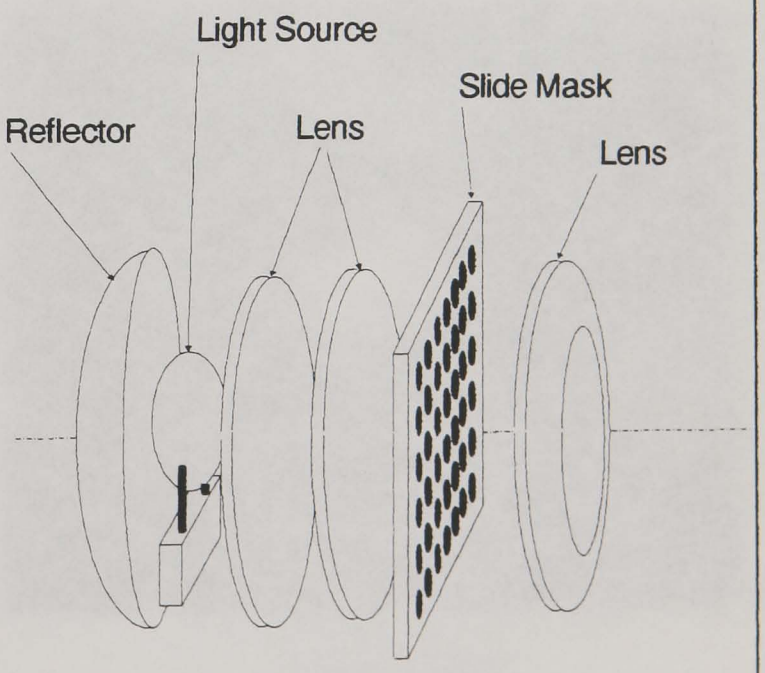


Figure 3.3.1
Projector Lens System

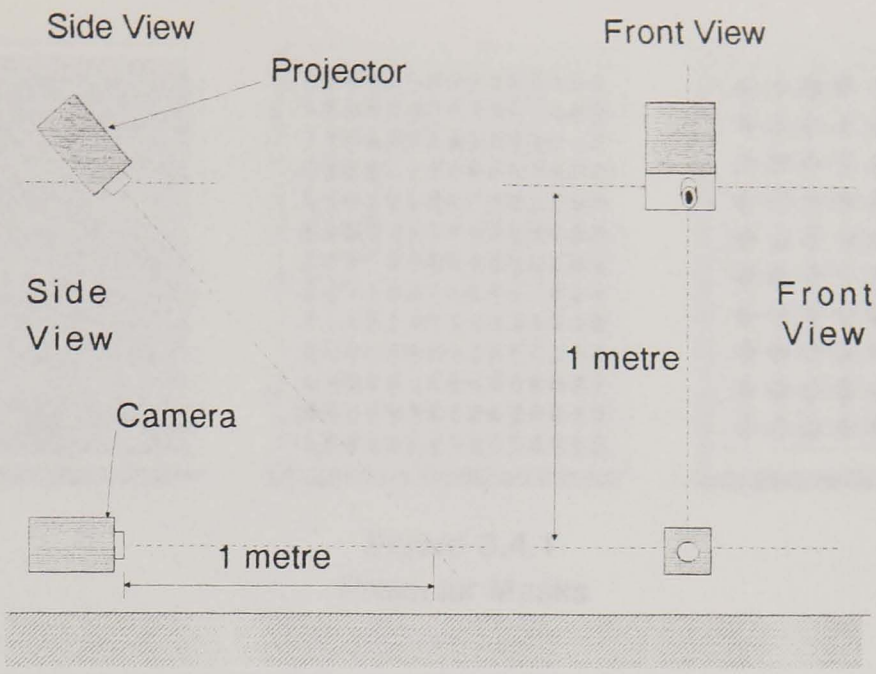


Figure 3.3.2
Projector - Camera Geometry

into the projection area (figure 3.3.3). The type of light pattern and the amount of information required to reliably detect obstacles determines the speed of operation.

3.4 Design for a Novel Light Coding System

Various projection masks were used in experiments, including those illustrated in figure 3.4.1. The presence of the pattern, for example the dot grid shown in figure 3.4.2 indicates the position of an obstacle. In order to detect the pattern by computer, images

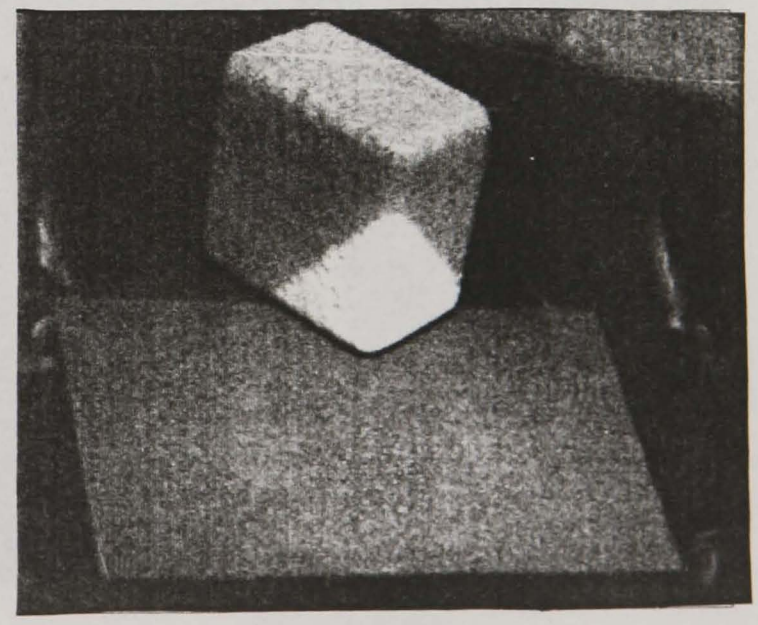


Figure 3.3.3
Projected Light Distortion

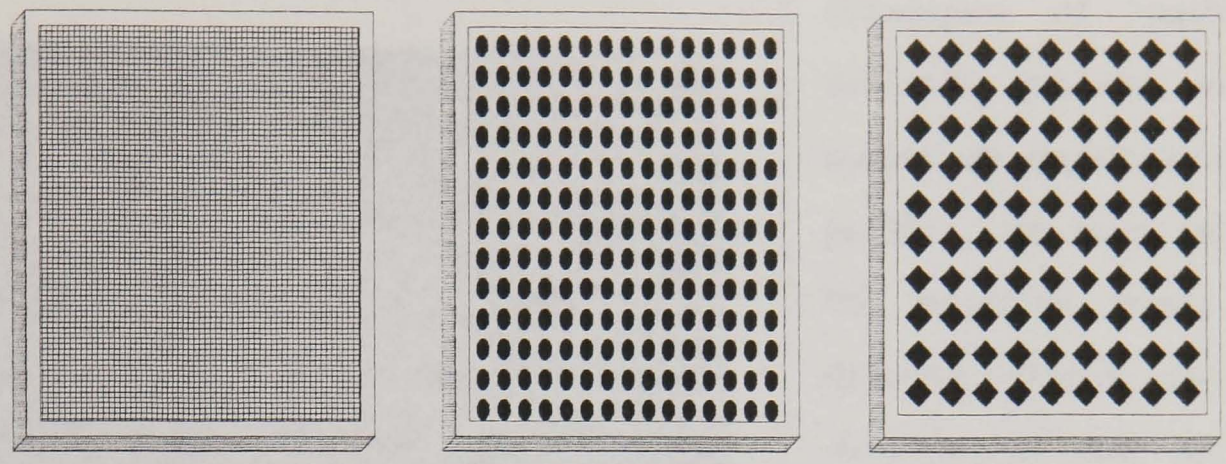


Figure 3.4.1
Projector Masks

were digitised into a matrix of grey levels. The resulting array was preprocessed using an edge detection and thresholding algorithm. Edge detection is essentially a high-pass filtering procedure which accentuates abrupt changes in contrast in an image. Thresholding is a selection procedure used to determine which edges are accepted for image analysis and which are ignored[60,61]. These processes help to eliminate the effects of disparate contrast caused by variable ambient lighting conditions and obstacle surfaces. The result of this computational stage produced the toroidal shapes visualised in figure 3.4.3. The presence and position of these shapes indicates the size and position of an obstacle.

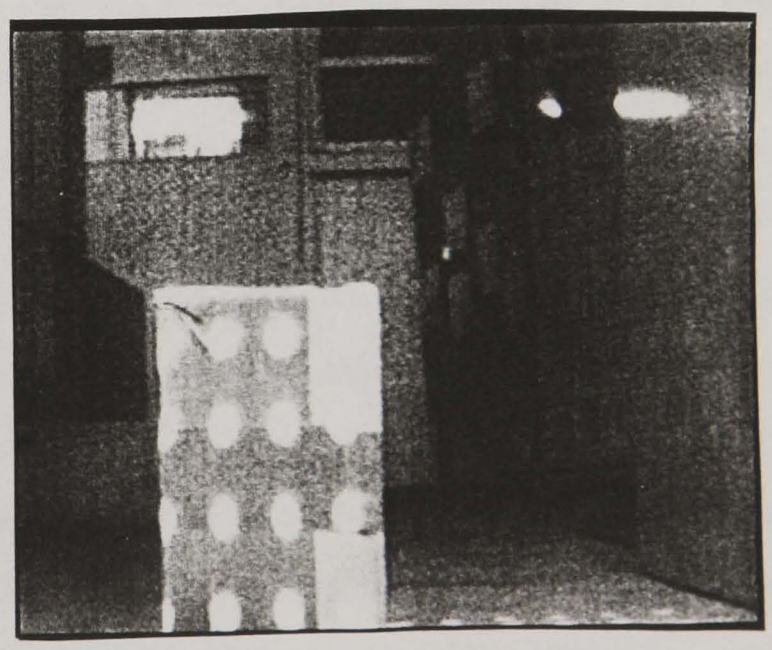


Figure 3.4.2
Obstacle Distorting Projected Dot Grid

The most effective masks are those which project patterns that are unlikely to occur naturally

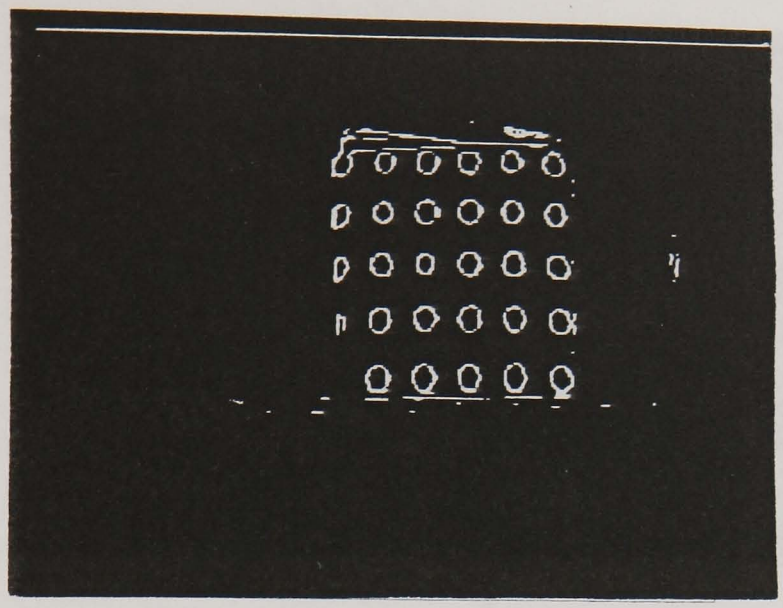
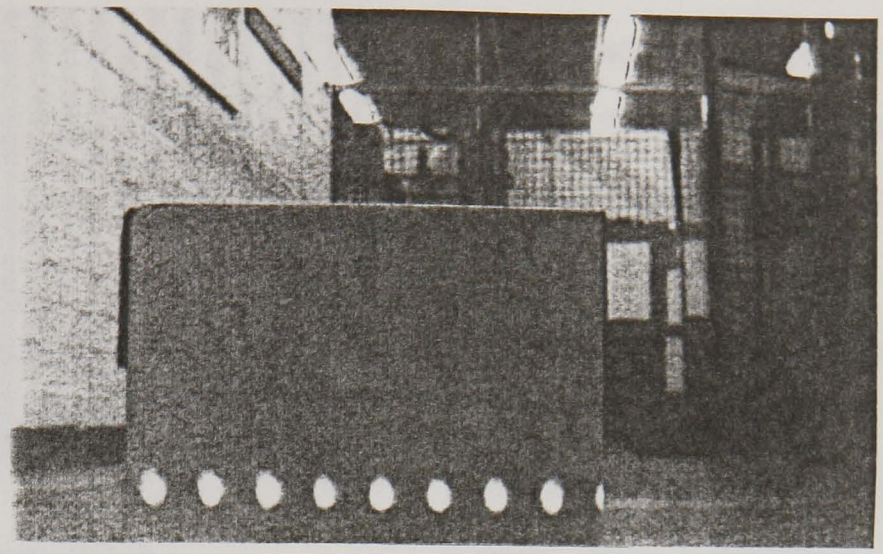


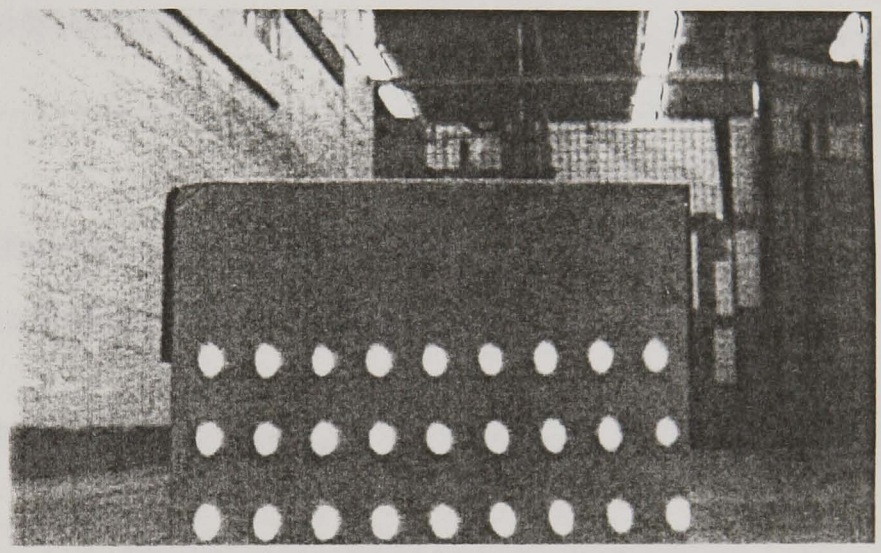
Figure 3.4.3

Distorted Dot Grid After Edge Detection and Thresholding

(examples of 'naturally' occurring patterns in this context are graphics or text on packing case sides etc). All the projection masks with discrete shapes such as diamonds or dots set in a predetermined grid performed well and obstacles could be reliably detected. However the tasks of two-dimensional edge detection and thresholding are too time consuming to be carried out by a low-cost embedded computer. The design was therefore simplified to reduce the processing demands whilst maintaining its reliability.

When objects enter the projection area, they disturb the light pattern. Consequently, this distortion appears to 'grow' from the floor and progress vertically in the image (figure 3.4.4). This effect is enhanced by virtue of the projector/camera geometric relationship as shown in the results presented in later chapters. Vertical bar patterns can be used to enable the image processing task to be reduced from two dimensions to one, providing that objects are stood on the floor (figure 3.4.5). In order to detect obstacles, the system needs to process only a narrow horizontal strip of the image which corresponds to the position where objects begin to distort the light pattern (figure 3.4.6). However, uniform bar patterns may become confused with patterns ocurring on objects which are not obstacles to be avoided. For example uniform markings on distant walls or packing cases, or iron railings with uniformly spaced vertical supports etc. A coded projection mask has been designed to assist in overcoming this possibility of false obstacle detection[53].





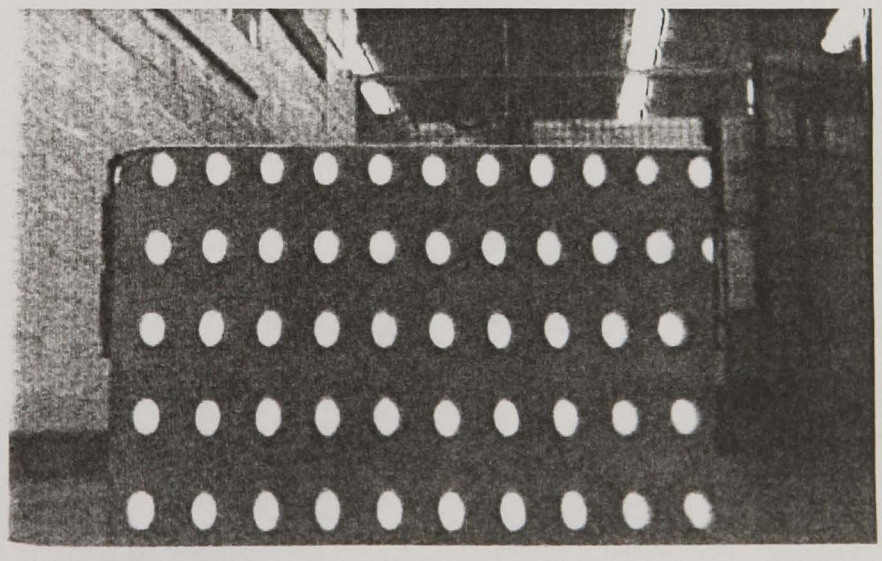


Figure 3.4.4
Projected Light Pattern Tends to 'Grow'
from the Ground

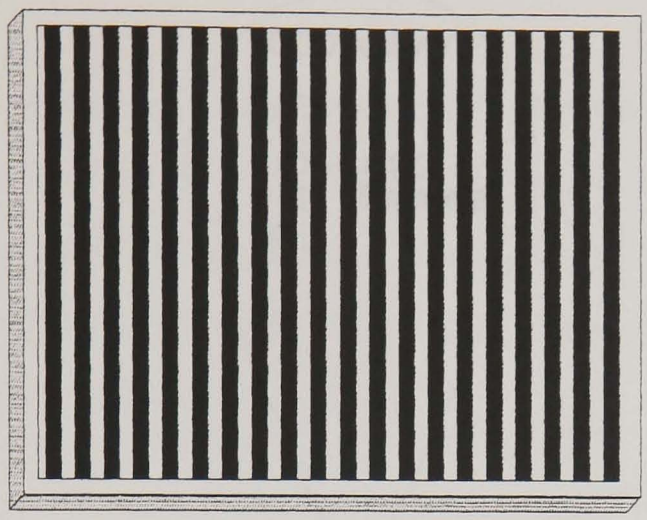


Figure 3.4.5
Uniform Vertical Bar Pattern

In general, codes with a large 'information' content will result in the most reliable obstacle detection. Such a code could be realised in the form of a projection mask consisting of several differently spaced vertical bars. However, the major disadvantage of this approach, is that only large obstacles, disturbing the whole projected pattern could be reliably detected. Since the system must also be able to detect 'thin' obstacles, several discrete codes across the image must be used and a compromise between code size and video system resolution must be found.

Several factors affect the design of a compromise light code including the

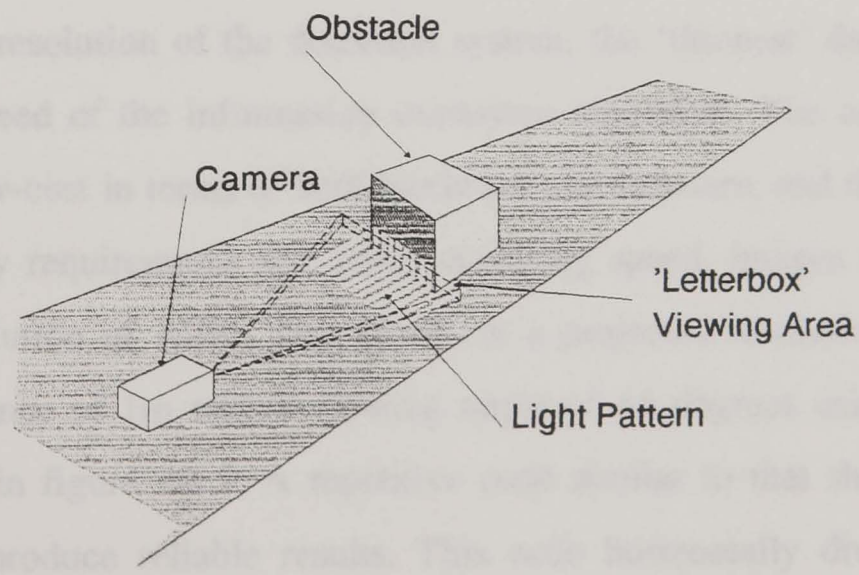


Figure 3.4.6
'Letterbox' Viewing Area

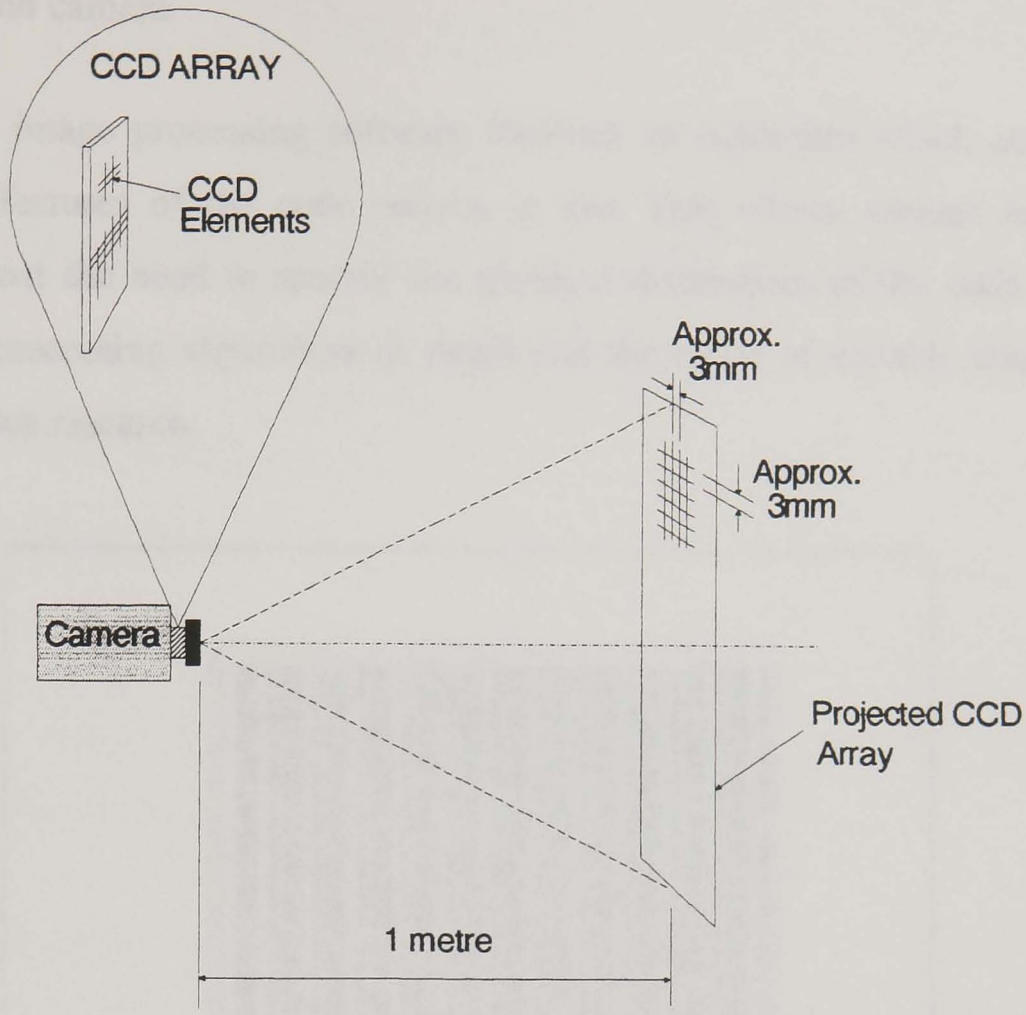


Figure 3.4.7

Projected CCD Elements Are Approximately
3mm X 3mm At a Distance of 1 metre

aformentioned resolution of the detection system, the 'thinnest' detectable object, and the required speed of the information extraction algorithm. The central theme of this design is its low-cost in terms of both hardware and software, and therefore to maintain modest memory requirements and high processing speed, images are digitised with a horizontal resolution of 1:256. This results in a projected resolution of approximately 3mm at a distance of 1m with a viewing angle of 45 degrees using a 12mm camera lens as shown in figure 3.4.7. A repetitive code similar to that shown in figure 3.4.8 was found to produce reliable results. This code horizontally divides the projection area into discrete regions. The detection of any complete code indicates the presence of an obstacle and its position in the image reveals the position of the obstacle in front of

the projector and camera.

The prototype image processing software includes an algorithm which automatically identifies the features of the code pattern in use. This allows various masks to be deployed without the need to specify the physical dimensions of the code. Chapter 5 describes the processing algorithms in detail and the range of suitable masks used in the course of this research.

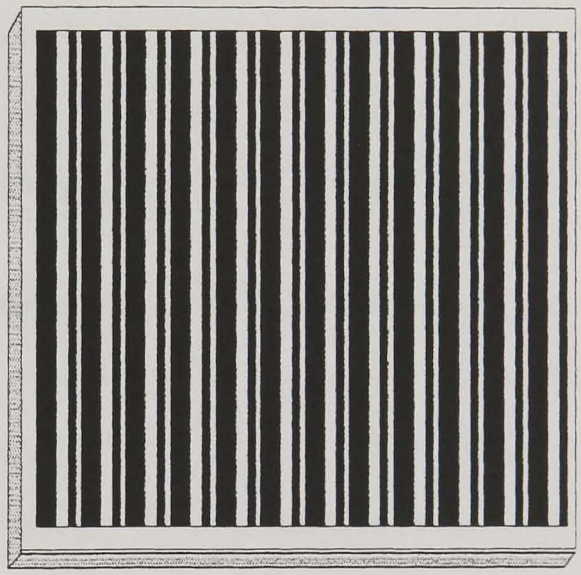


Figure 3.4.8
Vertical Bar Code Projection Mask

4 HARDWARE FOR THE OBSTACLE DETECTION AND EMBEDDED COMPUTER SYSTEM

4.1 Summary

The hardware for the obstacle avoidance system and the video camera used to detect obstacles are described in this chapter. The advantages of using robust Charged Coupled Device (CCD) image sensors are also highlighted.

The video digitising system is described in section 4.3. This unit takes a standard monochrome composite video signal as input and transforms it via a high-speed analogue to digital converter into a 256 X 256 array of grey levels. This array is in a form which can be processed by the embedded computer system.

In section 4.4, the main features of the Intel MCS-51 series microcontrollers are discussed and the design for the embedded computer system is presented. A shared memory access scheme is described which enables the microcontroller to gain fast access to the digitised video image array.

Finally in section 4.5, the complete embedded software development system is discussed with particular reference to the methods used to design and debug Intel 8051 assembler codes.

4.2 The CCD Video Camera as a Sensing Element

The falling cost of CCD arrays together with their improving quality makes them eminently suitable for use in machine vision systems. Also, the robustness of modern CCDs allows their use in systems that may be subject to noise, vibration and other harsh environmental conditions. A low-cost monochrome CCD video camera is used as the detection element of the obstacle avoidance system.

A particular feature of the 1/2'' CCD array used in this design is its sensitivity, which enables the camera to operate in light levels down to 0.5 Lux. Conversely, a built in auto-iris adjusts the camera aperture according to the average light intensity falling on the CCD array to prevent saturation in bright ambient light conditions.

The array consists of 370 X 350 light sensitive elements. Since the overall vision system resolution is limited by the Video Frame Store rather than the camera, this CCD camera fulfils the system requirements.

For objects to be avoided without collision, the projected light pattern must produce an image at least as wide as the automated vehicle when it is focused on the floor. A viewing angle of 45 degrees is required for the camera to detect the full width of such an image at a distance of approximately one metre. This is achieved by using a 12mm wide-angle camera lens.

4.3 Description of the Video Digitising System

A central feature of digital image processing systems is that they convert standard composite video signals generated by cameras of the type described in section 4.2 into a form which can be processed by a computer. This is normally achieved by digitising

the video signal and storing it in a memory array. Each picture element (or pixel) has a numeric value representing the light intensity, or grey level, of the corresponding point in the video image. Figure 4.3.1 shows the form of a composite video signal. As its name suggests, it is an amalgamation of video information and timing waveforms. As can be seen from the figure, each video line (corresponding to lines on a video monitor or television) takes 64 micro-seconds to update. Standard video frames consist of 625 lines (in the UK), and therefore the complete video image is updated in 50 milliseconds. However, since this update rate can be detected by the human eye and becomes irritating after a short viewing time, (the television or video screen 'flickers'), a system of 'interlacing' is employed to alleviate the effect. Rather than transmitting video signals as line 1 through to 625 consecutively, interlacing operates by transmitting odd lines in one screen update and even lines in the next. This effectively reduces the flicker effect and makes television and video screens less irritating to watch.

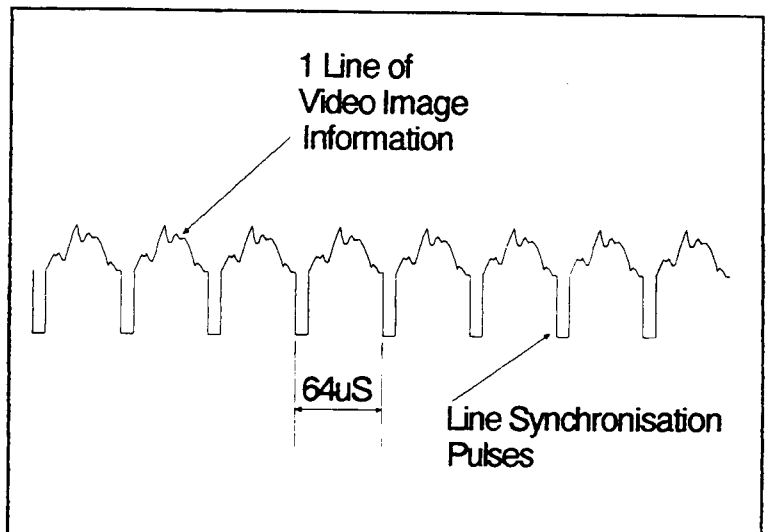


Figure 4.3.1
Composite Video Signal

If the full resolution of 625 video lines are not required, the video interlacing need not be used. For example if a video processing system has a vertical resolution of 256 lines, it is unimportant whether odd or even lines are used and therefore complete screen updates can take place in 25 milliseconds.

With reference to figure 4.3.1, each video line signal lasts 64 microseconds. For a horizontal resolution of say 256 pixels in a digitised image array, the video line signals must be sampled at 250 nanosecond intervals (for real-time operation). As already

mentioned this would result in a video memory array being completely updated in either 25 or 50 milliseconds depending on whether interlacing was used or not. High speed analogue to digital conversion, (with a sampling rate in the order of 8-10 megahertz) is required to achieve such real time video digitisation. Some video digitising systems overcome this need for high speed conversion by taking one sample from each video line per screen update. However, for a digitised image resolution of 256 X 256 pixels, this requires 256 screen updates. A system of this type will therefore take 6.4 seconds to completely digitise a video frame. This is an unacceptable performance for most robotic systems.

The monochrome Video Frame Store described here[62] operates in real-time with a resolution of 256 X 256 picture elements. Each pixel has a grey level value in the range 0-255. Interlacing is not used and therefore complete video images are digitised in 25 milliseconds.

Figure 4.3.2 shows the block diagram of the video frame store. An 8-bit analogue to digital converter was used in the system for the following reasons:

- The obstacle avoidance system is based on a low-cost 8-bit embedded microcontroller. Extra hardware and software overheads would be necessary for a word length greater than 8 bits and the operating speed of the system would be reduced.
- The cost of greater resolution analogue to digital converters that operate at speeds fast enough to digitise video signals in real-time is high.
- The limitations of 8-bit grey level and spatial resolution can be effectively overcome using digital filters implemented in software (see chapter 5).

With reference to the video frame store block diagram (figure 4.3.2), the synchronisation signals are separated from the video information and the resulting video signal is amplified and converted into 8 bit digital words. The control section of

the frame store writes these digitised samples to the correct addresses in the 64K video memory array via the memory access buffers shown in the diagram. (These are discussed in section 4.4).

In the prototype video frame store, a digital to analogue converter and associated control circuits are included to enable digitised images to be displayed on a standard composite video monitor. The digitised video data written to the memory array is constantly reread and converted back to an analogue signal. This signal is scaled and clamped to the correct voltage levels and synchronisation waveforms are mixed with it to reconstitute a composite video signal. The integrity of the video memory can be checked using this facility since a read or write failure will be reflected in the reconstituted video display. Also, the results of intermediate stages of video processing may be viewed as an aid to development. This section of the video frame store would be omitted from the finished product.

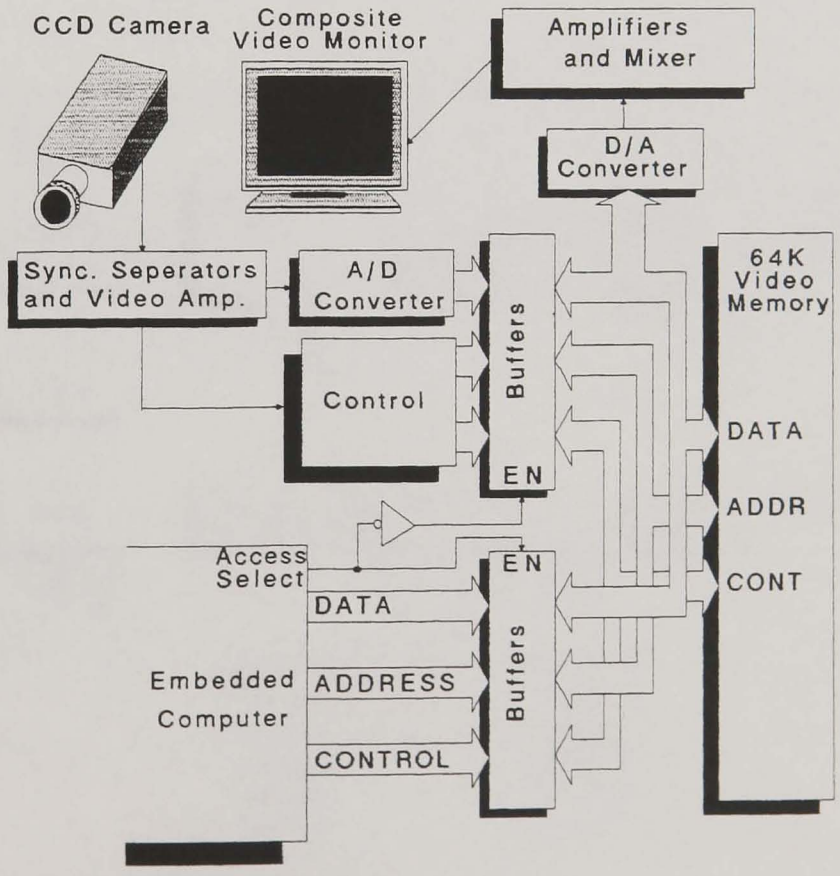


Figure 4.3.2
Video Frame Store and Memory Access Buffers

4.4 Introduction to the Intel MCS-51 Series Microcontroller

The MCS-51 series Intel microcontrollers consist of a family of 8 bit single-chip computers which are ideal for embedded applications. Figure 4.4.1 shows the block diagram of the original member of the family, the 8051. The main features of this integrated circuit are:

- ❑ 8 bit Central Processing Unit optimised for control applications.
- ❑ Boolean processing (ie. single bit) capabilities.
- ❑ 32 bidirectional and individually addressable I/O lines.
- ❑ 128 bytes of on-chip data RAM.
- ❑ Two fully programmable 16 bit timer / counters.
- ❑ Full duplex Universal Asynchronous Receiver Transmitter.
- ❑ 5 source interrupt structure with 2 priority levels.

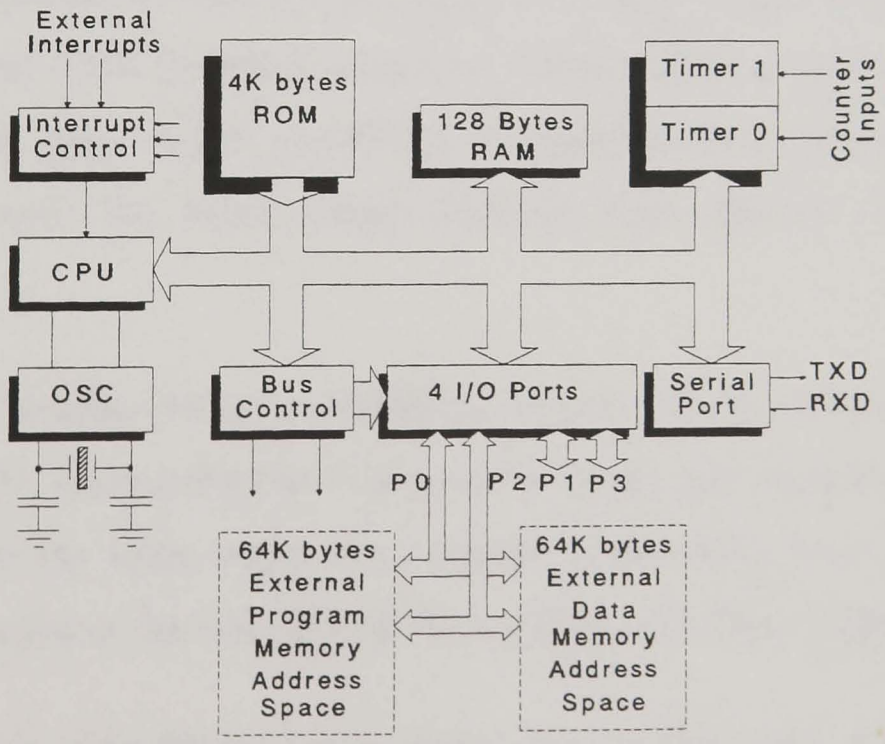


Figure 4.4.1
Intel 8051 Block Diagram

- On chip clock oscillator.
- 4K bytes on-chip program memory (one time only programmable ROM).
- 64K bytes program memory address space.
- 64K bytes data memory address space.

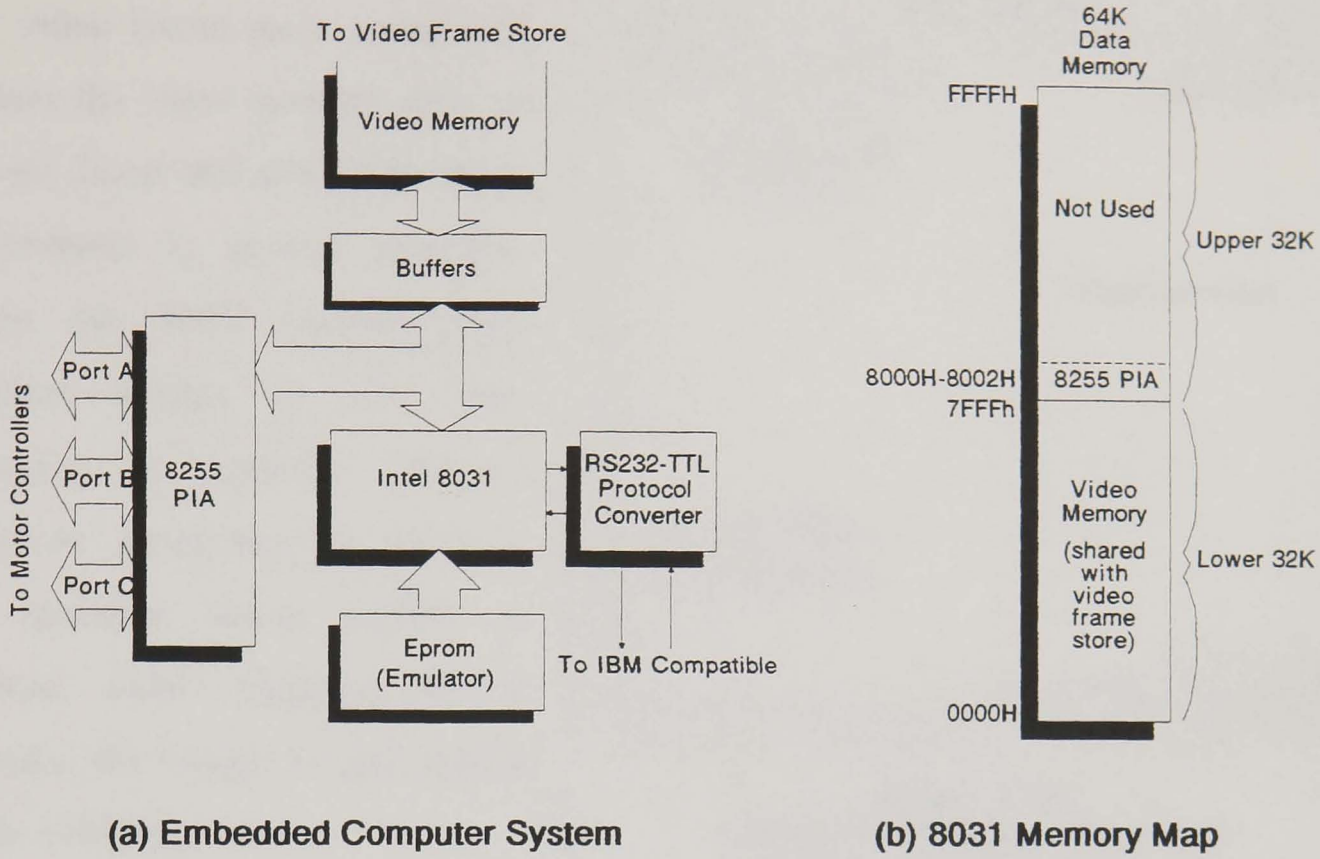
The second member of the family is the Intel 8031. This chip has all the features of the 8051 except for the 4K bytes on-chip ROM. Instead, the 8031 fetches all instructions from external program memory.

Other members of the family include 80CXX CMOS versions of the chip featuring low power consumption and 83CXX versions which incorporate facilities including a watch-dog timer and power down mode of operation.

In this work a single 8031 microcontroller is used for both obstacle detection and obstacle avoidance control. Figure 4.4.2.a shows the block diagram of the embedded computer system incorporating the Intel 8031 and associated interface circuitry and figure 4.4.2.b shows the data-memory map of the 8031 along with its allocation in this system. An Intel 8255 Peripheral Interface Adapter (PIA) is used for interfacing with the AGV main drive motor controllers discussed in later chapters. The Intel 8031 directly addresses the video image memory array directly in a shared access arrangement.

Shared access is achieved by incorporating memory access buffers in the circuits that enable the 8031 microcontroller to access the video data memory at high speed. No modification to the video frame store circuits is necessary since the memory access buffer circuit replaces the video RAM chips and uses the same connections.

If a single block of 64 Kbytes memory was used for the video frame store, the entire addressing range of the Intel 8031 would be occupied. This would not leave address



(a) Embedded Computer System **(b) 8031 Memory Map**

Figure 4.4.2

space for other devices such as the 8255 PIA and other possible system expansions. Furthermore, the speed constraints on the system are critical since digitising a video line into 256 pixels in real-time requires a write operation to the video memory approximately every 256 nanoseconds. These problems are overcome by using two interleaved 32 Kbyte 'pages' of memory rather than a single 64 Kbyte block. The organisation of these memory pages leaves 32 Kbytes Intel 8031 address space available and also relieves the system timing requirements. The two 32 Kbyte memory blocks are configured as shown in figure 4.4.3.

The two memory pages appear in the lower half of the 8031 data memory address map. Access to each 32K block is controlled via an additional control signal derived from one of the microcontroller input/output port pins. Each 32K memory is accessed once per two video-analogue to digital conversions effectively doubling the required write access time to around 500 nanoseconds.

The video frame store continually updates the video memory array so that the latest possible CCD image information is always available. When the 8031 microcontroller requires access to the video memory to perform obstacle detection processing, it switches the memory access buffers to suspend video updating (ie. it 'freezes' the image) to gain access to the memory.

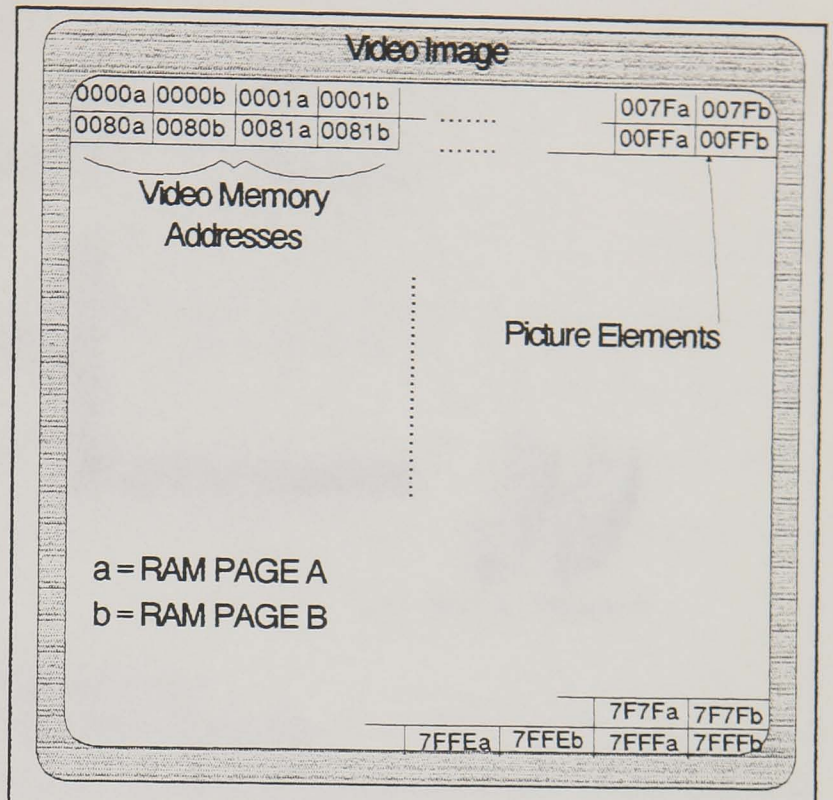


Figure 4.4.3
Method of Digitising Video Image
In Two Interleaved Blocks

Full details of the memory access buffers and embedded computer circuits are included in appendix 1.

4.5 The Embedded Computer Development System

Figure 4.5.1 shows the schematic diagram of the embedded computer development system. The program development cycle consists of the following stages:

i) Task evaluation and specification, (ie. what is required of the program and how will it be achieved).

ii) Write program source code in 8051 assembler language mnemonics. The source code consists of text files that can be generated using any text editor.

iii) Assemble source code files to produce 8051 machine code object files. This stage converts assembler mnemonics to Intel 8051 machine operation codes.

iv) Link object files to produce a single contiguous machine code file. Most larger assembler programs are developed in a modular fashion. This reduces

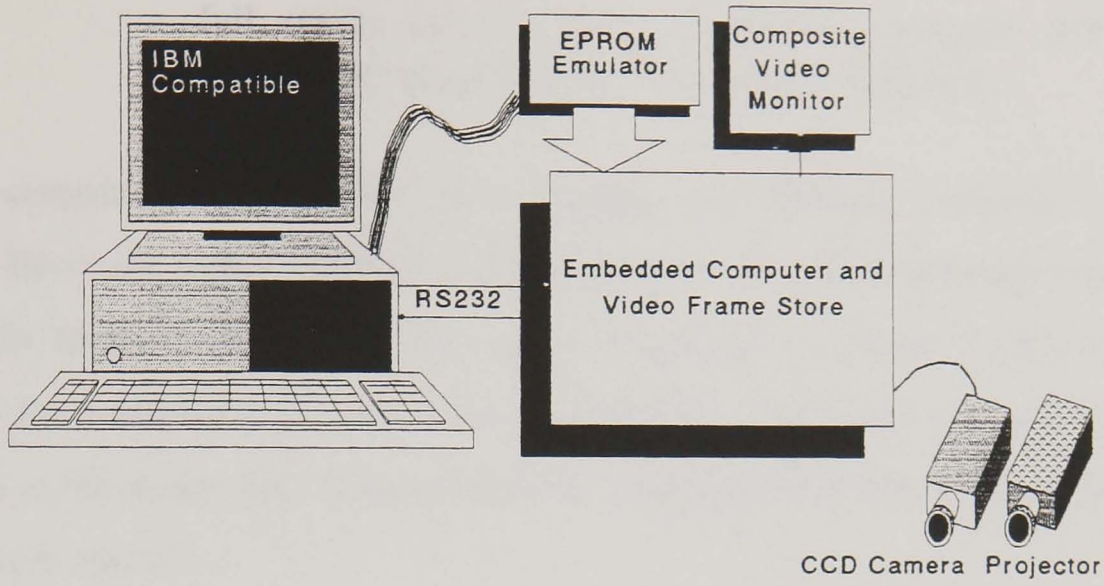


Figure 4.5.1
Development System

development time since only the source code file being edited needs assembling, (rather than the whole program). Also, small sections of large tasks are more manageable and errors more easily isolated.

v) Transfer machine code file to EPROM Emulator in the embedded system. (The code then appears as program memory to the embedded computer). In general, the programs used in embedded computer systems can be referred to as 'firmware'. This is because they are stored in non-volatile Read Only Memory (ROM) chips on the embedded system circuit board. This is in contrast to 'software' which generally describes computer programs which are loaded into volatile memory to be executed. The task of reprogramming and erasing an Erasable ROM (EPROM) during development is time consuming and therefore an EPROM emulator is used. This is essentially a volatile memory which can be quickly loaded with a machine code file, but which appears to the embedded computer as ROM firmware.

vi) Reset embedded system and test software. The software testing stage can be particularly difficult in embedded systems, since often none of the debugging tools and displays available in higher level programming environments are available. The Intel 8031 on-chip Universal Asynchronous Receiver Transmitter has therefore been used for serially communicating with an IBM PC compatible computer. This results in the additional development stage:

> Transfer data and variables via serial communications link to IBM compatible computer for debugging purposes. Hence the

full power of the IBM compatible computer keyboard and display can be used to aid software development.

In some complicated algorithms (for example the obstacle avoidance procedures described later) the task of coding software directly into 8051 assembler is extremely difficult. In these cases models were first developed in Turbo PASCAL and then simplified for conversion to Intel 8051 source code. Again full use was made of the final stage in the above development cycle to interface the PASCAL and machine code software as it evolved.

The following two chapters describe in detail the algorithms implemented on the embedded computer to process digitised video information and to detect obstacles.

5 DIGITAL SIGNAL PROCESSING

5.1 Summary

Coded patterns occur in the CCD video image due to the reflection of projected light patterns from objects on the floor ahead of the AGV. This chapter describes the techniques used to recover this code information from digitised video data.

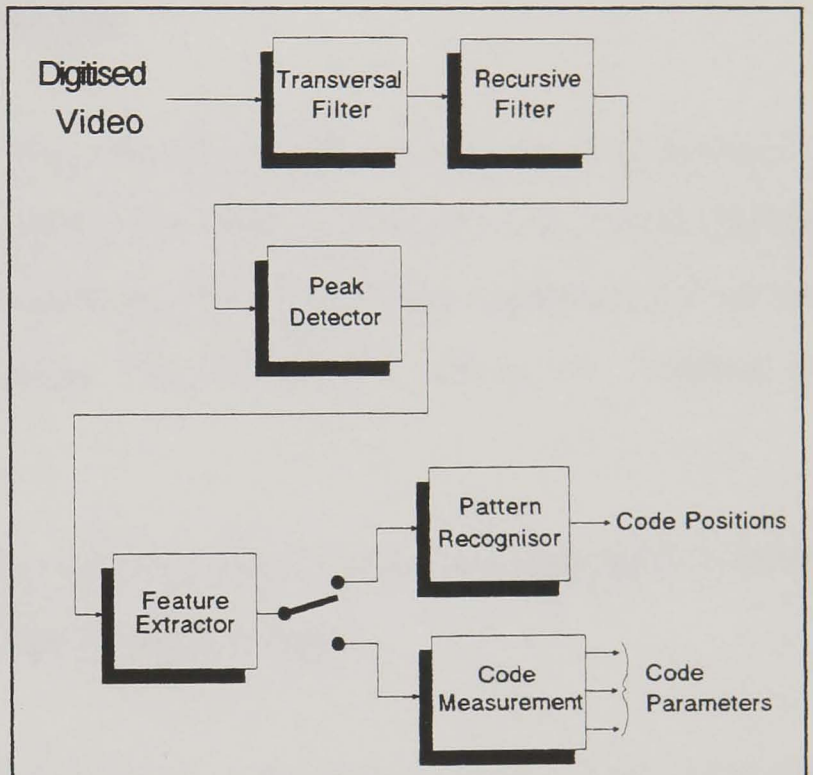


Figure 5.1.1
Obstacle Detection System

Figure 5.1.1 shows the block diagram for the obstacle detection system. With reference to the figure, the obstacle detection system operates on digitised video data and is divided into three sections:

- A preprocessing stage which employs two digital filters that are described in section 5.2.
- Peak detection and feature extraction stages which isolate potential code parameters (detailed in section 5.3).
- Code measurement and recognition stages which are discussed fully in chapter 6

The digital filters described in section 5.2 perform a vital role in maximising the operating speed of the obstacle detection system. They are used as a preprocessing

stage to 'clean up' the video signal and act as a coarse 'sieve' for the data. So called 'direct methods' (see section 5.3) are then applied to extract potential code features from the preprocessed video data. Finally, valid codes are detected using a 'decision theoretic', pattern recognition technique described in chapter 6.

5.2 Digital Filter Design

Since the low cost video frame store and associated hardware are relatively limited in terms of resolution and grey level accuracy, the system is subject to a certain amount of quantisation error. This manifests itself as wide band noise superimposed on the digitised video signal. The digital filters described in this section are designed to reduce this effect.

Figure 5.2.1 shows a video-still together with a typically noisy graphical representation of a single horizontal line scanned across the lower image(1).

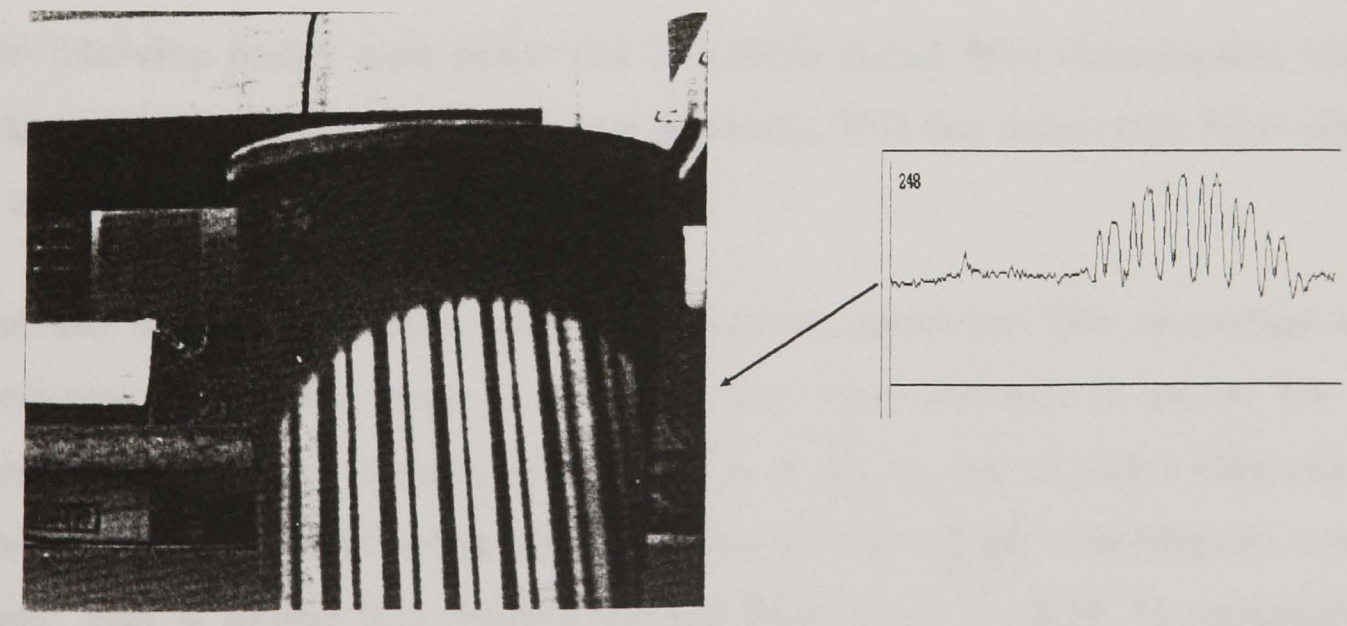


Figure 5.2.1
Grey Level Graph of a Horizontal Strip of the Image

(1) This graph along with others similar throughout this chapter were derived by transferring data from the embedded system to the IBM PC compatible computer via a serial communications link and using a specially written PASCAL program to present the data.

Digital filters were chosen in preference to analogue filters in this work, since after digitising, video data are in a highly suitable form for processing using discrete digital methods.

Dedicated digital signal processors were considered for the task of digital filtering. Although the operating speed of such devices is high, a comparison between dedicated signal processors and microcomputers showed the former to be relatively expensive with a low degree of flexibility. On the other hand, Signal processing using micro-computers is cost effective, flexible and requires a simple hardware design[63]. Hence, with the special design considerations described next, the required digital filters were implemented in software on the Intel 8031 microcontroller.

The design of tightly specified finite impulse response filters for microprocessors is a complex task which is exacerbated by the restriction of using assembler language. This is due to the need for numerical accuracy. Insufficient accuracy in both the storage of numbers and calculations can compound the quantisation errors in the system and in some cases render the filters unstable[64].

The following design work shows that acceptable digital filter characteristics can be achieved using exactly specified integer arithmetic. This fact makes such filters simple to implement with high execution speeds.

The first stage of processing uses a non-recursive transversal filter to average eight horizontal lines of the image in the 'letter box' view described in section 3.4. The operation of this filter is shown in figure 5.2.2. A special case of such a filter and one which does not require floating point arithmetic is when all the weightings a0...am are equal. This is similar to a moving average filter[64] and is simple to implement in assembler language. The one used in this work has the magnitude frequency response shown in figure 5.2.3. The result of this filtering operation is a single line array in which each element is the average of the corresponding elements from the original

horizontal lines. The effect of this operation is to enhance the influence of persistent vertical patterns in the image array whilst reducing the effect of spurious noise or 'snow'. The choice of the number of lines to average depends on both the minimum height of the object being detected and the required algorithm execution speed. Whilst a large number of lines results in greater enhancement of vertical patterns, if the object in the image is not tall enough to produce persistent vertical patterns, the process will fail. Conversely, if too few lines are averaged, the process has a reduced immunity to noise.

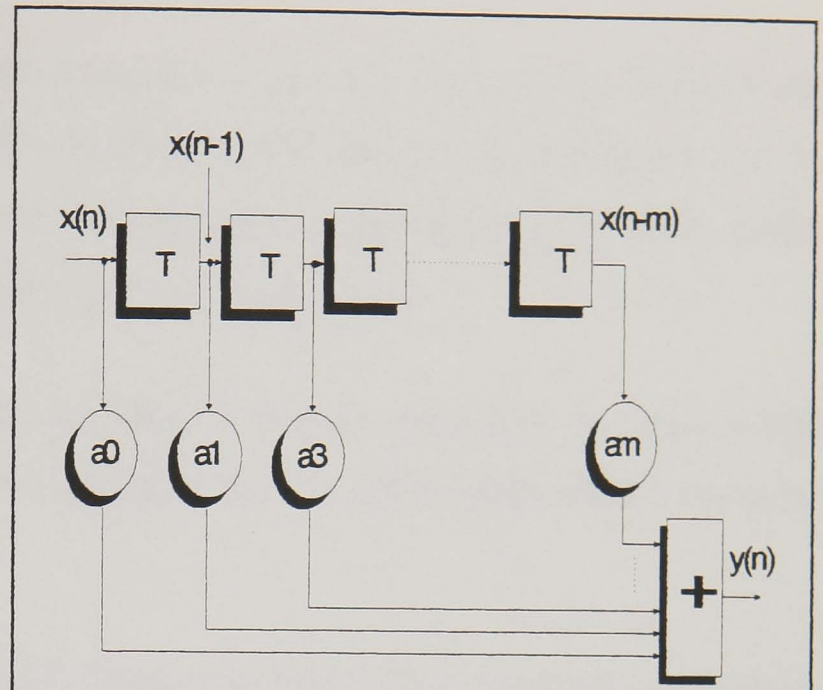


Figure 5.2.2
Digital Transversal Filter

In order to maximise the execution speed of the digital transversal filter, only 'powers of two' horizontal lines were considered (ie. 2, 4, 8, 16 etc.). These values enable division calculations to be carried out efficiently using machine code arithmetic shift

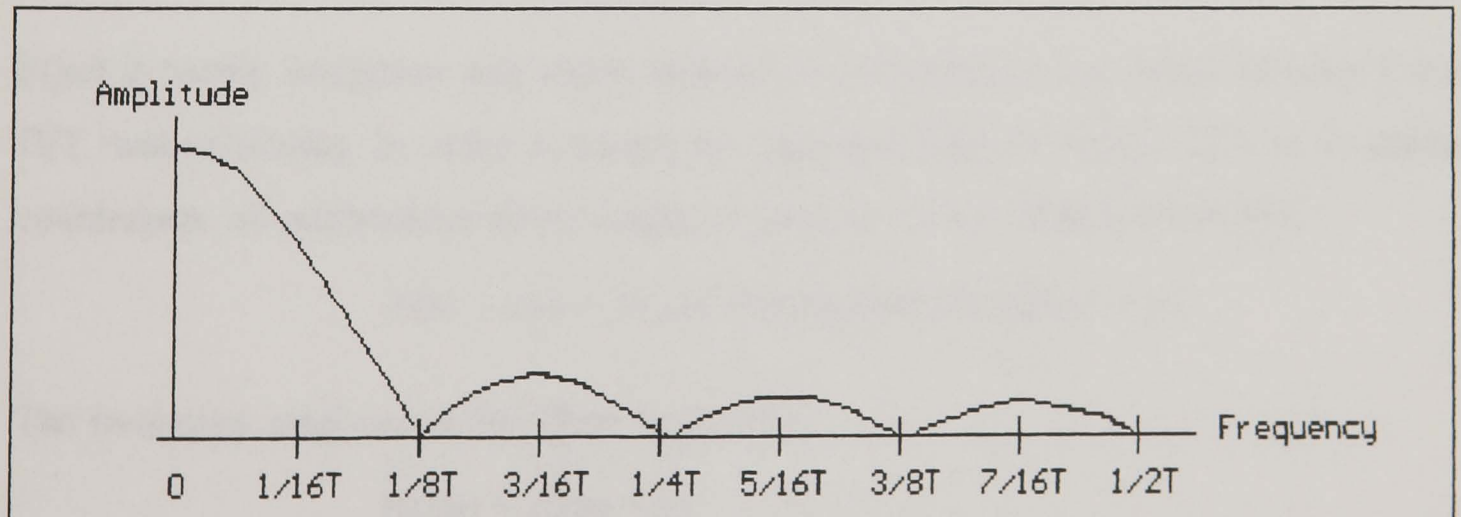


Figure 5.2.3
Transversal Filter Frequency Response

right operations. Experiments showed that eight horizontal lines of the video image provides reliable obstacle detection combined with high execution speed. In the prototype design, this corresponds to a physical horizontal strip approximately 20mm when projected onto an object one metre in front of the camera.

The resultant array of 256 averaged elements is further processed to remove high frequency components in the horizontal direction. This is achieved by using a recursive digital filter.

An effective method of designing digital filters is to model their analogue counterparts using the bilinear transform method. This is the so called 'frequency transformation' method and is described in the following work:

Consider the fraction $F(z) = (z-1)/(z+1)$ where $z = e(sT)$

This is 'bilinear' in that both numerator and denominator are linear in the variable z. In order to show the value of the bilinear transform method for converting analogue filters to their digital counterparts, the spectrum of F(z) must be determined as follows:

$$F(j\omega) = \frac{e(j\omega T)-1}{e(j\omega T)+1} = \frac{e(j\omega T/2)\{ e(j\omega T/2 - e(j\omega T/2)\}}{e(j\omega T/2)\{ e(j\omega T/2 + e(j\omega T/2)\}}$$

$$= j \tan \omega T/2$$

$F(j\omega)$ is purely imaginary and varies between 0 and infinity as ω varies between 0 and Π/T radians/second. In order to covert the analogue filter of figure 5.2.4 to its digital counterpart, all occurrences of the Laplace operator 'S' are replaced with F(z):

$H(s) = 1/(s + \alpha)$ and therefore $H'(z)\ 1/[F(z) + \alpha]$

The frequency response of the filters is given by:

$$H(j\omega) = 1/(j\omega + \alpha)$$

and $H'(j\omega) = 1/(F(j\omega) + \alpha) = 1/(j \tan \omega T/2 + \alpha)$

The magnitude response of the digital filter is shown in figure 5.2.5.

When the required time constant of this low pass digital filter is specified, it can be converted into a recurrence formula as follows:

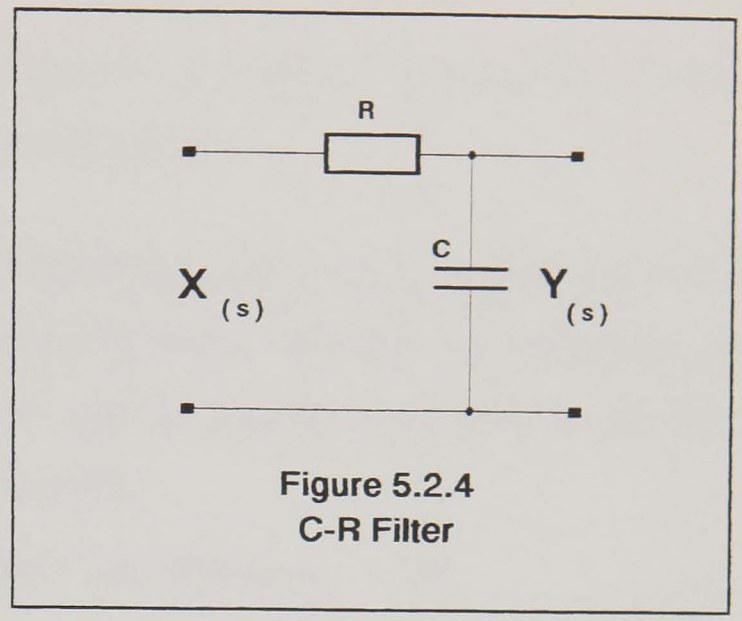


Figure 5.2.4
C-R Filter

$H'(z) = 1/[F(z) + \alpha] = 1/[(z-1)/(z+1) + \alpha]$ (where $\alpha = 1/\tau$)

therefore $Y(z)/X(z) = K\ (z + 1)/(z + A)$

where $K = 1/(1 + \alpha)$ and $A = (\alpha-1)/(\alpha+1)$

so $Y(z)\ (z + A) = X(z)\ (z+1)$

and the recurrence formula is:

$K\ (Yn + AYn-1) = Xn + Xn-1$

therefore $Yn = Xn + Xn-1/K + AYn-1$

An inspection of this result reveals that whilst the frequency response is of a suitable

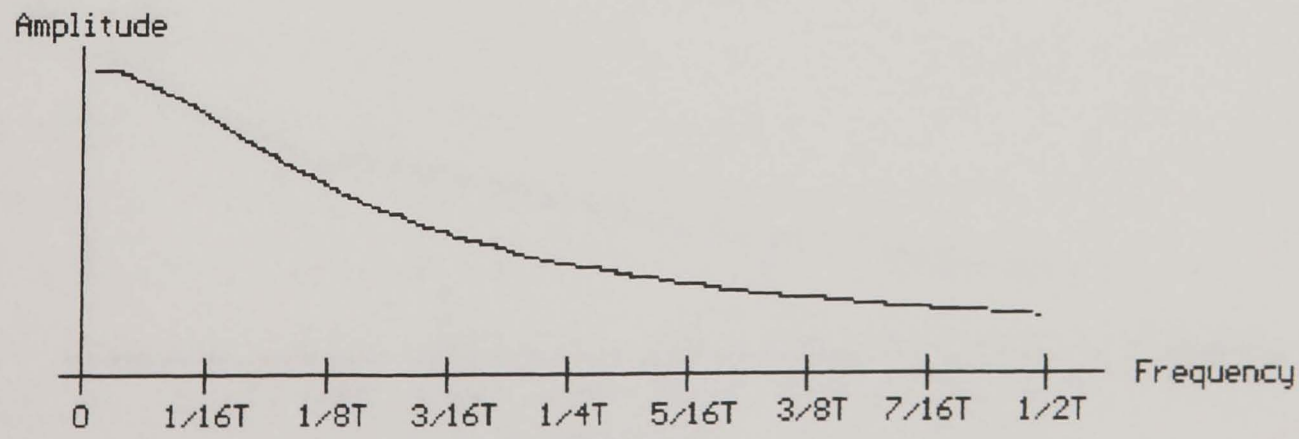


Figure 5.2.5
Frequency Response of a Filter Derived
Using Bilinear Transform Method

shape, in general the filter coefficients 'A' and 'K' will require an accuracy of around four decimal places to give a satisfactory performance[60].

A more direct approach can be taken when digitising analogue filters which results in a slightly degraded, but nevertheless acceptable, frequency response and which has the major advantage that the filter coefficients can be expressed precisely as integers. Considering again the analogue filter of figure 5.2.4

$$Y(s)/X(s) = 1/(1+SCR) = 1/(1+S\tau) \text{ since } \tau = CR$$

Writing this in the form of a difference equation (assuming a first order approximation for S):

$$Y(n) + \tau/T(Y(n)-Y(n-1) = X(n)$$

$$Y(n) =[X(n) + Y(n-1)\ \tau/T] /[1+\tau/T]$$

where T = sampling period

τ = Time constant

For integer implementation with a time constant of 3T, the difference equation is:

$$Y(n) =[X(n) + 3Y(n-1)]/4 \text{ (assuming T to be unity)}$$

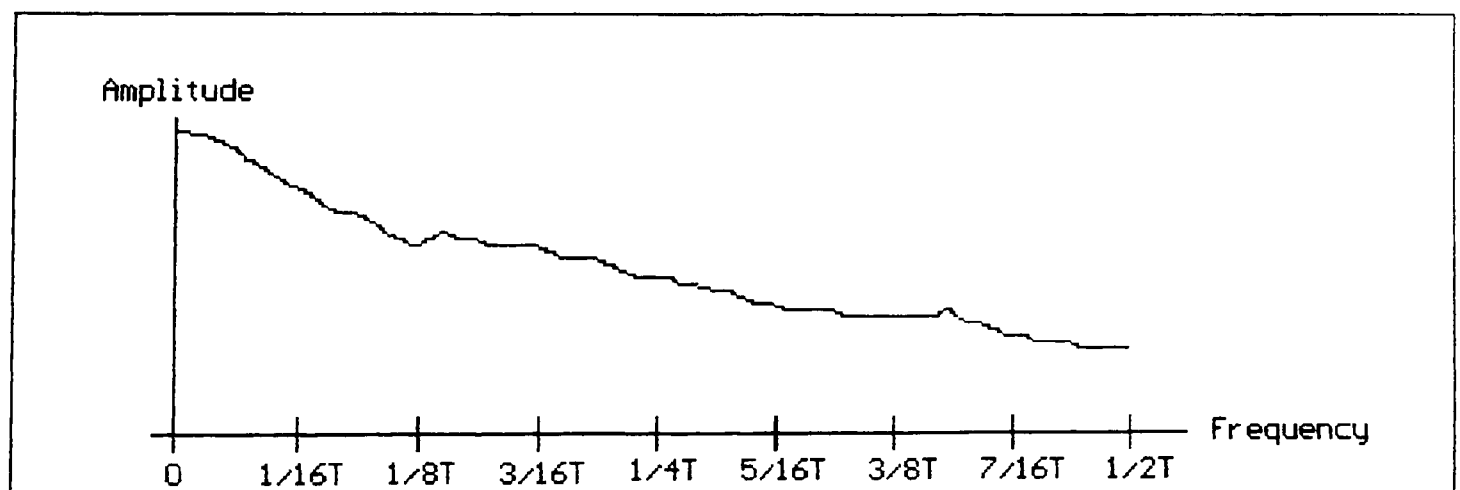


Figure 5.2.6
Simply Derived Digital Version of a C-R Analogue Filter
the 'Noise' on the Response is caused by Rounding
Errors in the Integer Calculations

For this time constant, the filter coefficients are shown to be exactly 3 in the numerator and 4 in the denominator. The magnitude frequency response of this filter is shown in figure 5.2.6. A comparison between figure 5.2.5 the filter characteristic derived through applying well known bilinear transform methods, and figure 5.2.6 using the simpler approach reveals that the latter's frequency response is only slightly degraded, but that the filter is much simpler and faster to implement. Furthermore, if a time constant is chosen which results in the denominator of the recurrence equation being a power of two, the algorithm can be efficiently encoded using arithmetic 'shift-rights' to perform the division calculation quickly.

Whilst the low pass filtering stages are essential for removing high frequency noise from the digitised image array to simplify later processing, care has to be taken not to filter out important information. The nature of the code pattern being sought from the array,(ideally high contrast light and dark bars) means that it has high frequency components in the abrupt changes between light and dark. If the cut off frequency of the low pass filter is excessively low, vital information may be lost. A compromise is therefore found by selecting a filter time constant to give the frequency response shown in figure 5.2.6. The final result of the filtering stages on the array from figure 5.2.1 is shown as the grey level graph in figure 5.2.7.

The recursive filtering stage affects the range of bar codes which can be used as projection masks due to the cut off frequency characteristics described in the previous paragraph. If the code period is short (ie. its frequency is high), true video data will be attenuated along with the quantisation noise which the low-pass filter is designed to remove. Conversely, when the code period is long, the 'thinnest' detectable object is limited. A compromise is reached in the prototype design with a code period of 16 'horizontal pixels', resulting in a thinnest detectable object of approximately 50mm.

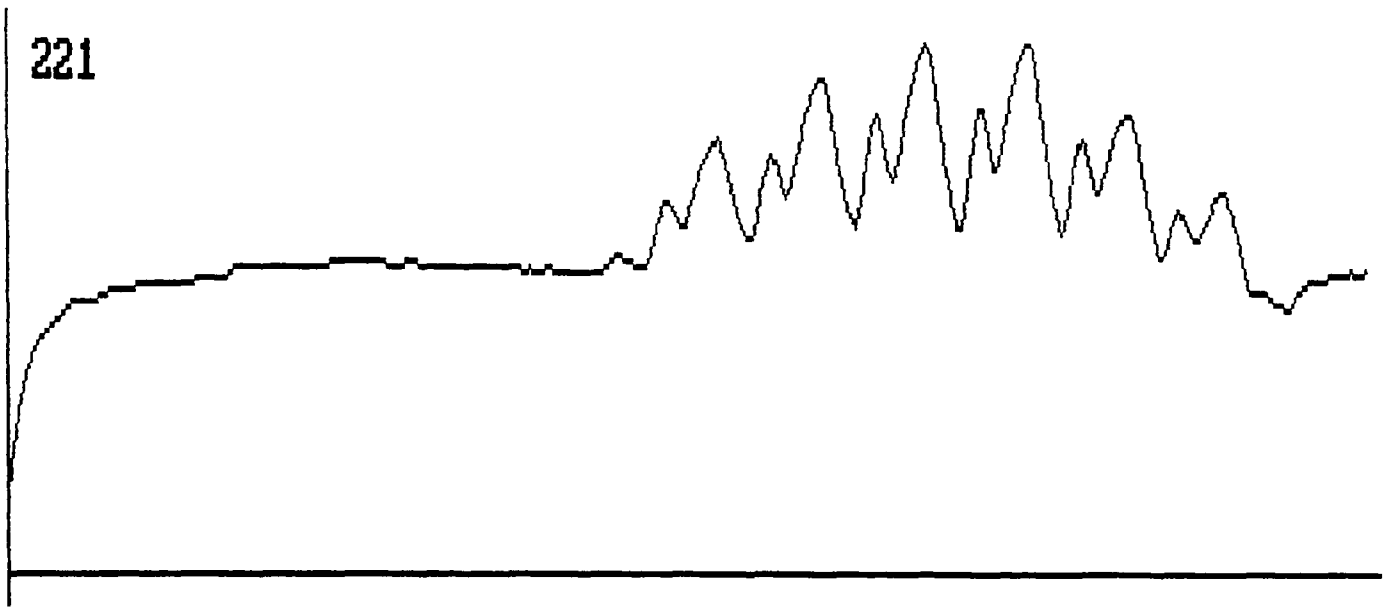


Figure 5.2.7
Video Data After Recursive Filtering

5.3 Direct Methods for Code Feature Extraction

Direct methods (as opposed to frequency domain methods) are widely used in pattern recognition, particularly in the fields of speech recognition, image analysis and medical science[65].

Frequency domain methods include Fourier analysis and matched filtering techniques that isolate frequency components of signals in order to match certain characteristic patterns. There are two main reasons why these methods are not adopted in this work:

- Frequency domain algorithms such as the Fast Fourier Transform generally require complex floating point arithmetic which is complicated to implement using assembler language.
- As already discussed in section 5.2, the video image of the binary code may be contaminated with noise of a similar frequency. It is not always possible to isolate useful information from the digitised signal using frequency domain methods alone.

Direct methods operate on the time series of signals directly rather than transforming them into the frequency domain. Two principle techniques of pattern recognition using direct methods are:

- ❑ The decision theoretic approach based on using numerically valued features for distinguishing a pattern class from all others.
- ❑ Syntactic and structured methods which use the models and techniques of formal language theory to analyse explicit or implicit characteristics of sub-patterns which form larger patterns[66].

Syntactic and structured methods are suited to recognising 'families' of patterns in applications such as electro-cardiogram analysis and speech recognition. However, in this application, the decision theoretic approach is most suitable since only one pattern must be recognised. Features are extracted from the filtered data (figure 5.2.7) and transformed into 'feature space' where they are tested against predetermined code parameters. Positive test results indicate the presence of an obstacle to be avoided.

Care must be taken in selecting the parameters which identify codes. In the digitised video signal the absolute magnitude of the signal cannot be used because this varies widely depending on the nature of the object reflecting the light pattern and the ambient lighting conditions. However, code patterns which appear in the image always have the same shape which is illustrated in figure 5.3.1. This general shape is obtained from the reflected light codes regardless of the absolute video signal magnitude. With reference to figure 5.3.1, the features which do not depend on the magnitude of the signal are the presence of maxima and minima of grey levels associated with the code and the spatial relationship between them (in this case the horizontal distance between them 'nT'). Figure 5.3.2 highlights these relationships.

The value of the digital filtering stages that act as a coarse 'sieve' for the video data can now clearly be appreciated. The signal has been effectively 'cleaned up' by the filters, reducing the number of maxima and minima that need to be detected and processed (compare figure 5.2.7 with figure 5.2.1).

The algorithm used to detect maxima and minima operates on groups of samples in the

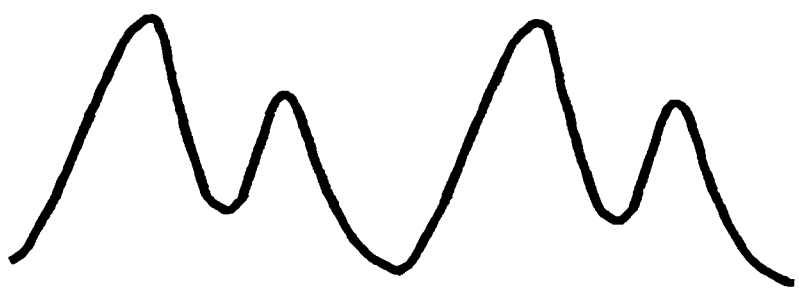


Figure 5.3.1
Typical Filtered Code Pattern Shape

digitised array to isolate local peaks and troughs. This operation may be viewed as a direct method of high pass filtering since it has the powerful effect of removing steady state signal levels (analogous to DC offset) leaving only the differentials. With reference to figure 5.3.3., the 'peak' detector algorithm works as follows:

For i = 2 to 253

if x[i] < x[i-2] and x[i] < x[i+2] then maxmin[i] = minima

else if x[i] > x[i-2] and x[i] > x[1+2] then maxmin[i] = maxima

else maxmin[i] = 0

next i

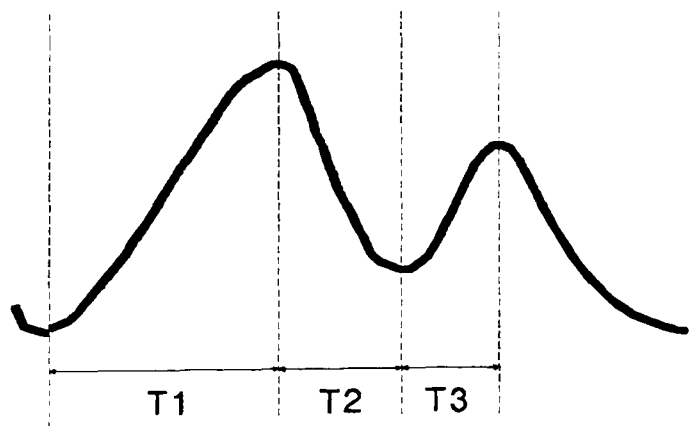


Figure 5.3.2
Relationship Between Code Features

The resultant array, maxmin[i] consists of markers indicating maxima, minima or neither with the index i corresponding to the spatial position of the turning points in the filtered data array x. Figure 5.3.4 shows this in graphical form, the filtered array is shown on the upper graph, with the corresponding position of maxima and minima shown beneath it.

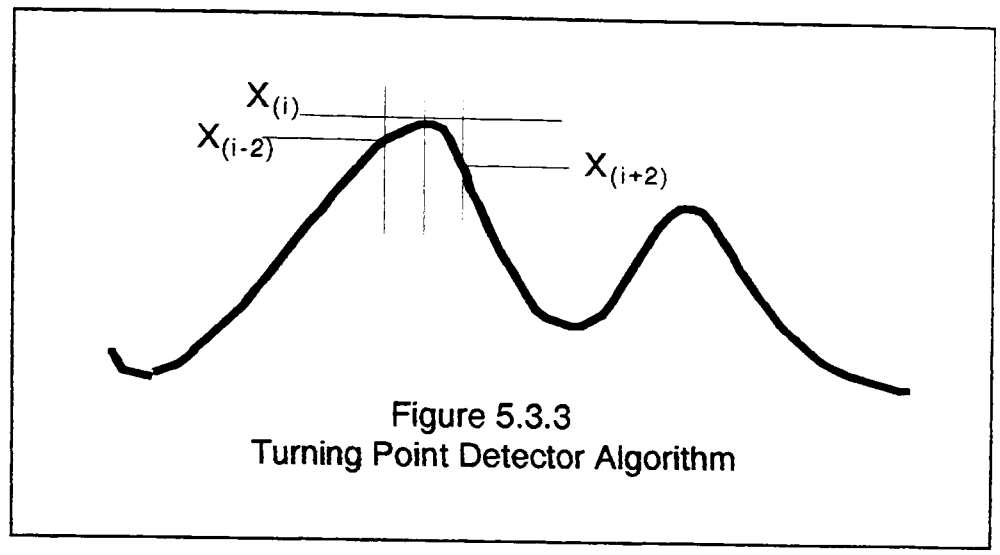


Figure 5.3.3
Turning Point Detector Algorithm

The remaining task of isolating and recognising valid codes is described in chapter 6. This task involves transforming the maxima-minima array into feature space, and testing it for membership of a predetermined pattern class.

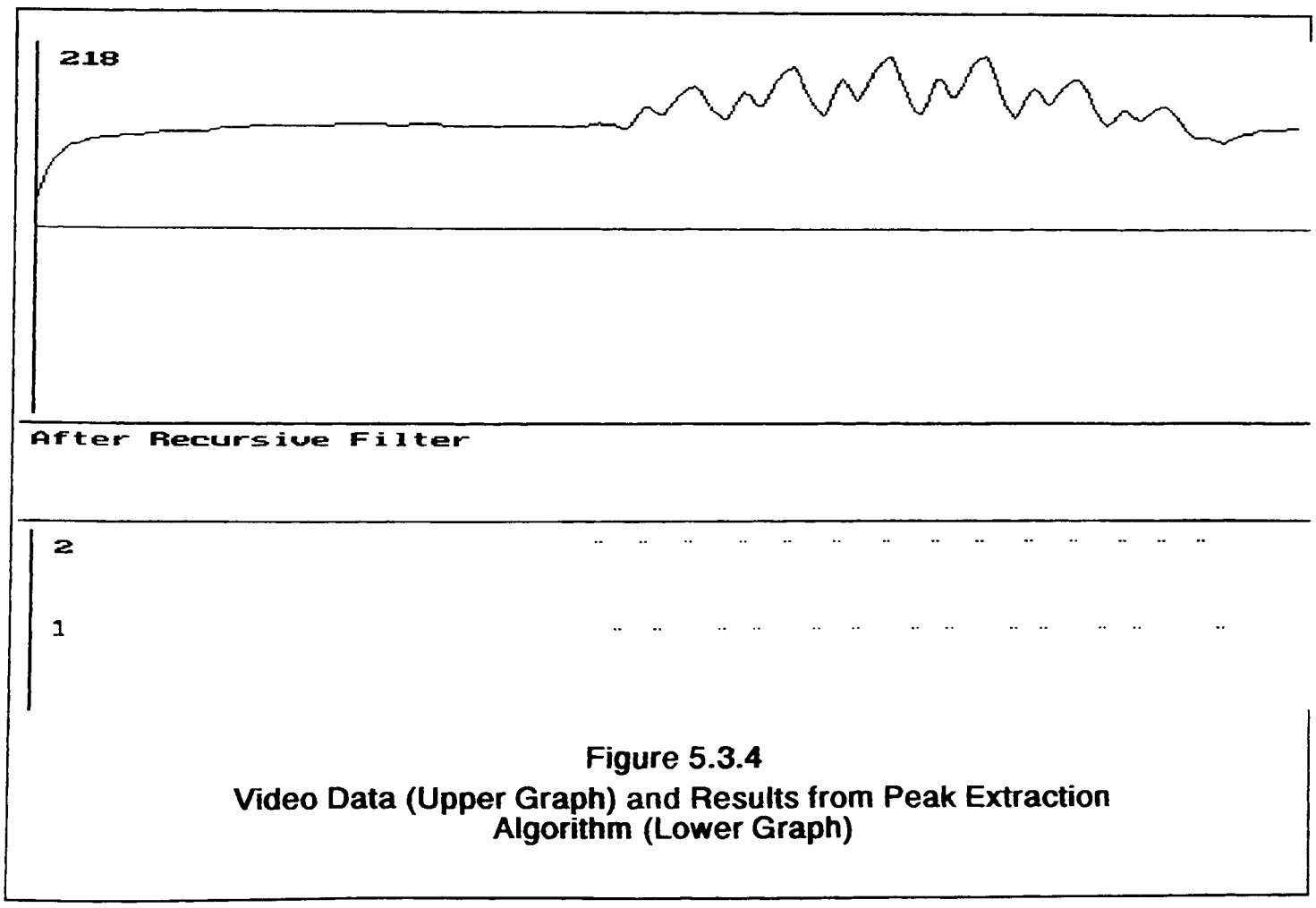


Figure 5.3.4
Video Data (Upper Graph) and Results from Peak Extraction Algorithm (Lower Graph)

6 REFLECTED LIGHT CODE RECOGNITION

6.1 Summary

This chapter describes the final processes that form the obstacle detection system. With reference to the block diagram in figure 5.1.1 these are the code measurement and pattern recognition stages.

A key factor contributing to the design presented in this research, is the ease with which the obstacle detection system can be configured. This has been achieved by the design of an automatic code measurement procedure (described in section 6.2) which alleviates the need for precise camera and projector positioning.

Section 6.3 describes the code recognition algorithm. This is based on the so called 'decision theoretic' approach of numerically quantifying pattern features and checking them against a predetermined template. Extra security is built into the system by incorporating a majority polling scheme to increase the surety of valid light code detection and reduce the sensitivity of the system to quickly moving obstacles.

6.2 Code Calibration Method

When a coded pattern is projected onto the floor ahead of the automated guided vehicle, the dimensions of the code detected by the CCD camera depend on the following criteria:

- The physical dimensions of the projection mask.
- The height and angle of the projector (and the lens system used).
- The relationship between the camera and projector.

- ❑ The camera lens system.

These criteria directly affect the code features T1, T2 and T3 in figure 5.3.2. The task of calculating these features based on the geometry of the system is difficult since it requires precise knowledge of the position of the projector and camera. One of the chief design features of this work is that the system is simple to configure. This is made possible by implementing a self calibration procedure in the obstacle detection software which enables the system to determine the code parameters automatically. This feature has also simplified experimentation with different code masks.

To operate the code learning procedure, the embedded computer system must be connected to a serial communications terminal or personal computer running a communications terminal emulator. A Turbo PASCAL program has been specifically developed for this task (see appendix 2). By default, the embedded computer uses a communications protocol of 4800 baud, 8 data bits, no parity, 1 stop bit. When the embedded system is reset, a message is transmitted to the terminal offering the code learning procedure as an option should it be required. If 'Q' is transmitted back to the embedded computer (by typing 'Q' at the terminal), the obstacle avoidance system is initiated using default code parameters programmed in EPROM. However, if any other key is pressed at the terminal, the embedded computer prompts the system installer to position a test board in front of the camera and press 'C' to continue.

For the system to measure the code parameters, the plain board placed in front of the camera must fill the horizontal field of view as shown in figure 6.2.1. The algorithm operates by measuring all the visible code parameters and recording their maximum and minimum values denoted by: T1max, T1min; T2max, T2min and T3max, T3min. The use of maximum and minimum values of the code features builds tolerance to slight spatial distortions at the periphery of the camera field of view and also allows for variations in obstacle surface inclination.

The immunity of the measurement procedure to spurious readings is increased by averaging eight complete measurement cycles resulting in aggregate values of the maximum and minimum values of T1, T2 and T3. Eight measurement cycles are averaged for the practical reasons outlined in section 5.2. The procedure of averaging eight numerical values can be easily and efficiently implemented within the constraints of integer arithmetic using simple addition and three binary shift-right operations. Figure 6.2.2 shows the flow chart of the automatic code calibration algorithm. A video screen is captured, filtered and processed as described in chapter 5. The resultant array of maxima and minima (maxmin introduced in section 5.3) is searched for groups of four points in the sequence: $maxima_1$, $minima_2$, $maxima_3$, $minima_4$, (see figure 6.2.3). T1, T2 and T3 are calculated from:

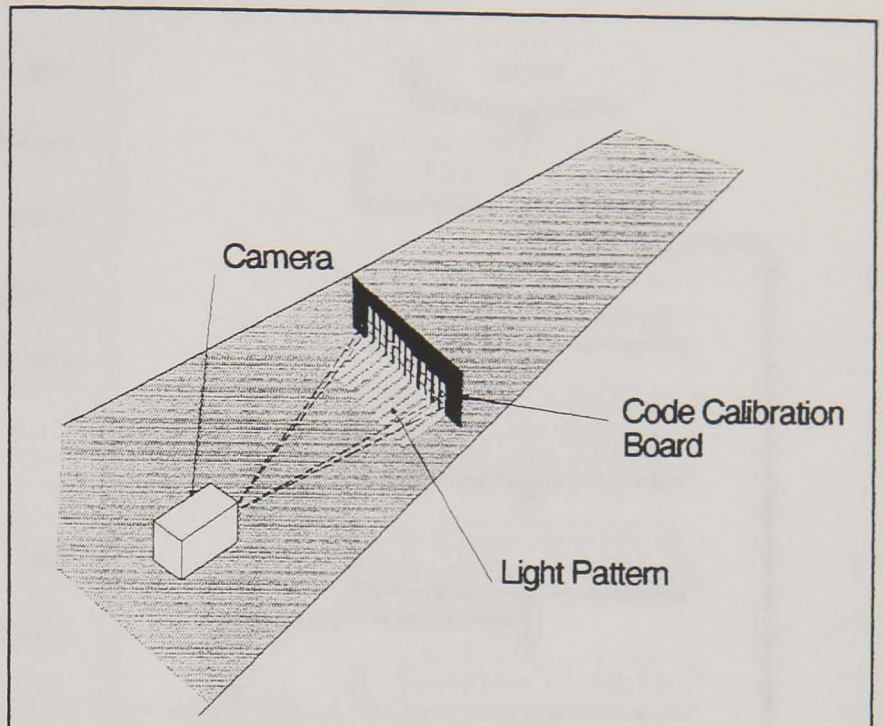


Figure 6.2.1
Code Calibration Board
Must Fill Camera Field of View

$$T1 = minima_2 - maxima_1$$

$$T2 = maxima_3 - minima_2$$

$$T3 = minima_4 - maxima_3$$

From these values, the features (T1min, T1max), (T2min, T2max) and (T3min, T3max) are updated. When eight passes have been made, the results are averaged to determine final minimum and maximum values. These values are used to form the feature template described in the next section. The calibration procedure is flexible in that the test board can be any surface which is wide enough to fill the camera field of view.

The identified code parameters are then transmitted to the serial communications terminal and displayed. These can be programmed into the embedded system EPROM to complete the installation.

The next section describes the procedure designed to detect code patterns and provide control signals to the obstacle avoidance control software.

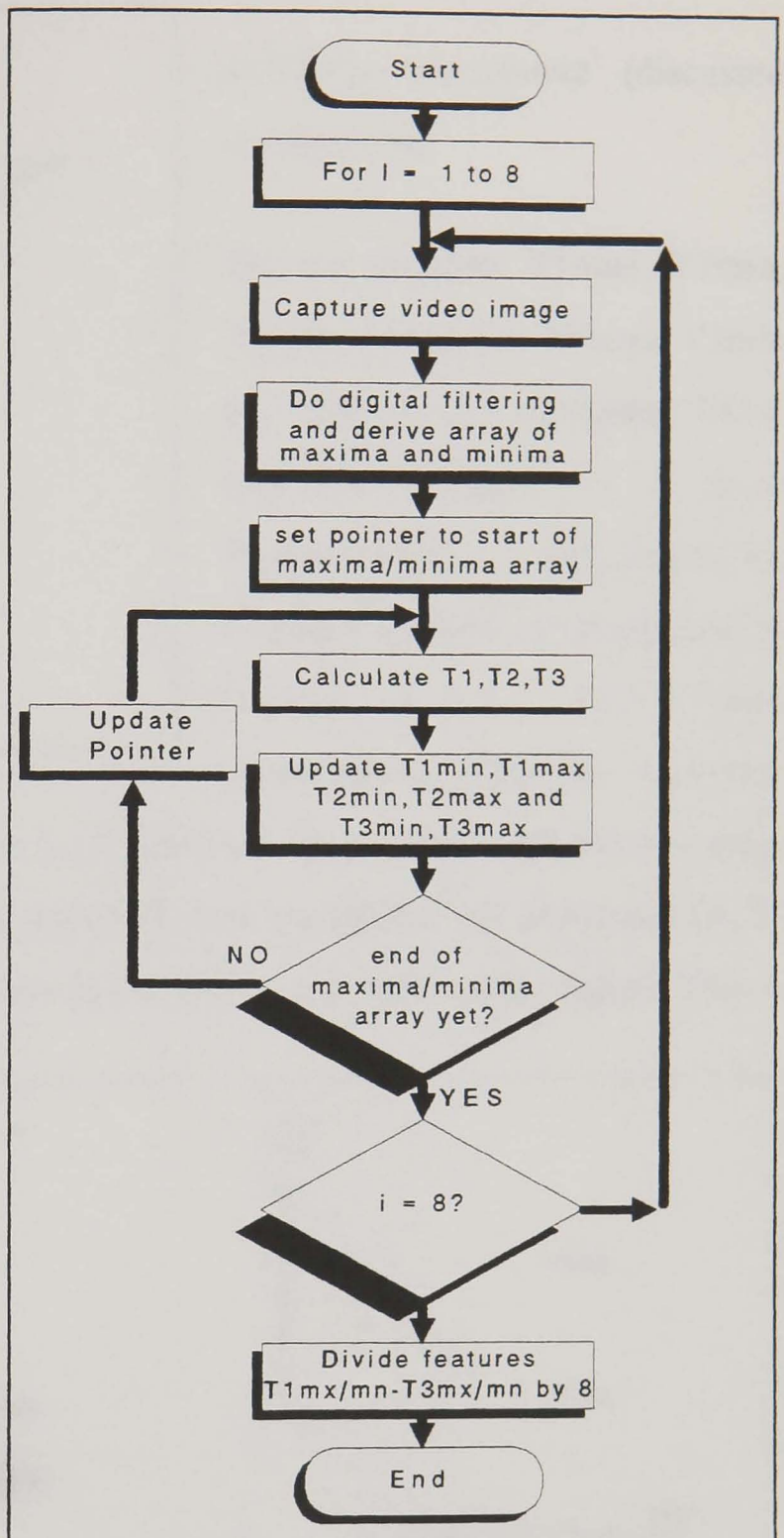


Figure 6.2.2
Flow Chart for Automatic Code Calibration Algorithm

6.3 Light Code Recognition

A decision theoretic approach has been adopted for recognising codes in the projected light pattern. This technique involves identifying key features of the pattern as described in chapter 5, transforming them into 'feature space' and testing them against the template derived in the last section. If a valid code is detected, it is due to an obstacle entering the light code projection area in front of the AGV. The distortion of the light pattern is detected by the camera and obstacle detection software, and the system consequently responds by initiating the obstacle

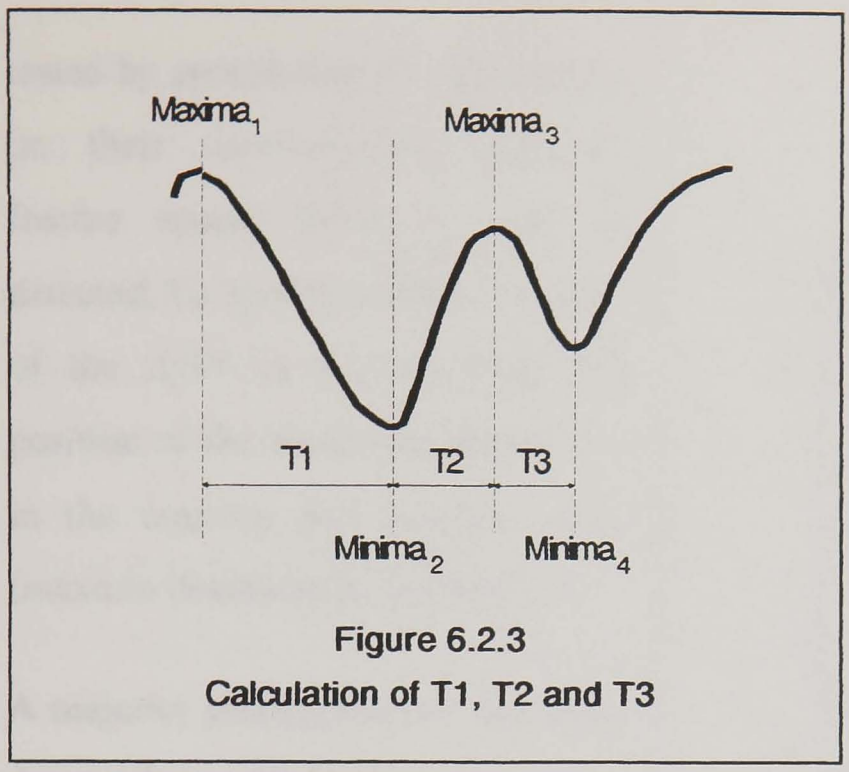


Figure 6.2.3
Calculation of T1, T2 and T3

avoidance manoeuvre (discussed in chapter 8).

The six features: T1max, T1min, T2max, T2min and T3max, T3min are used in the template. These may be mapped in a three dimensional representation ('feature space') as illustrated in figure 6.3.1. The code identification algorithm measures the parameters T1, T2 and T3 in figure 5.3.2 and tests them against the feature space template of figure 6.3.1. A code is only accepted as being valid if the measured T1, T2 and T3 result in a point in feature space lying inside the valid code region. This is

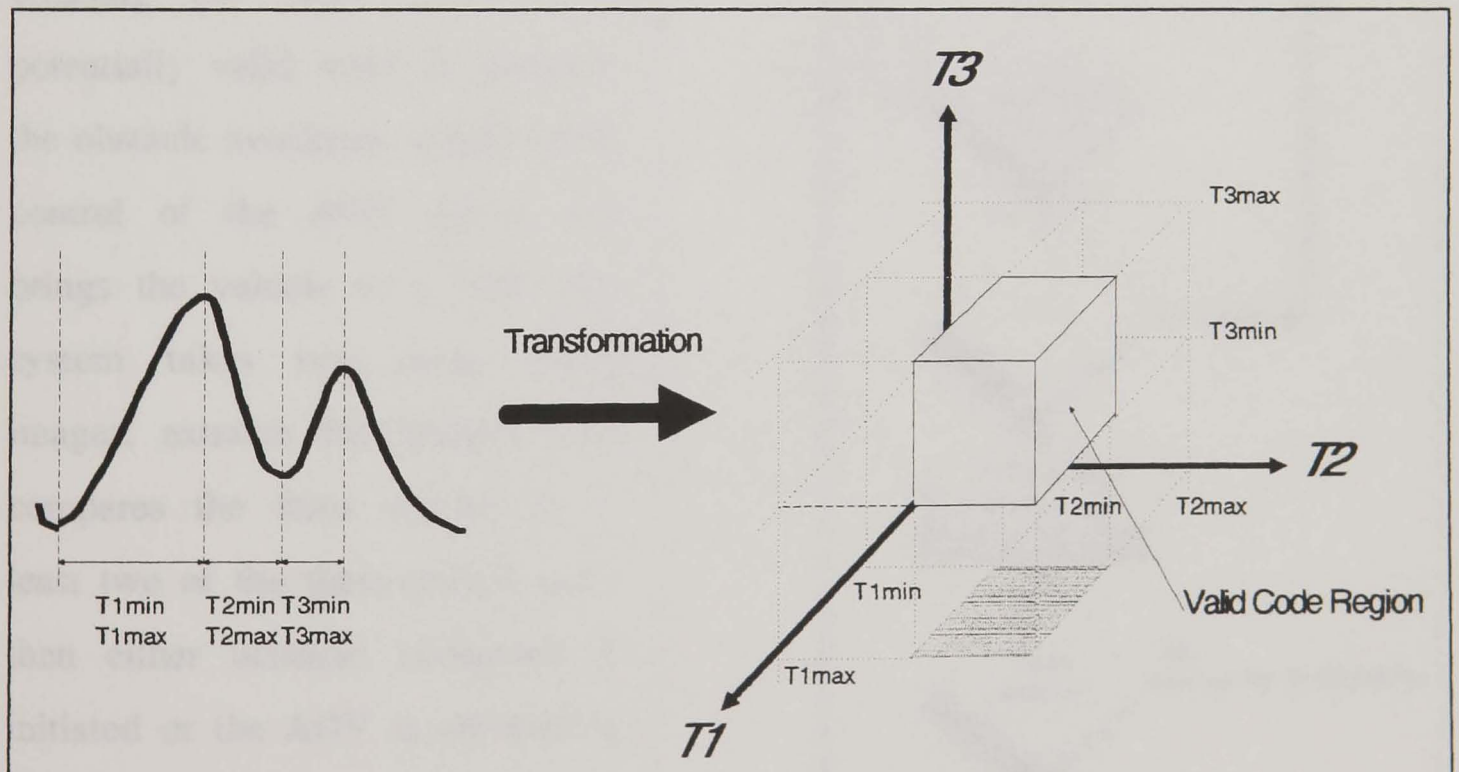


Figure 6.3.1
Code Features Transformed

tested by considering T1, T2 and T3 on their corresponding axes in feature space. When a code is detected, its spatial position in front of the AGV is derived from the position of the measured parameters in the maxima and minima array (maxmin described in section 5.3).

A majority polling scheme has been designed to add further security to the code detection system and to reduce its sensitivity to non-persistent obstacles such as people crossing the AGV path. If a potentially valid code is detected, the obstacle avoidance system takes control of the AGV drives and brings the vehicle to a halt. The system takes two more video images, extracts the features, and compares the three results. If at least two of the three results agree then either obstacle avoidance is initiated or the AGV is allowed to proceed, (depending on the outcome of the poll).

With reference to the flow chart of

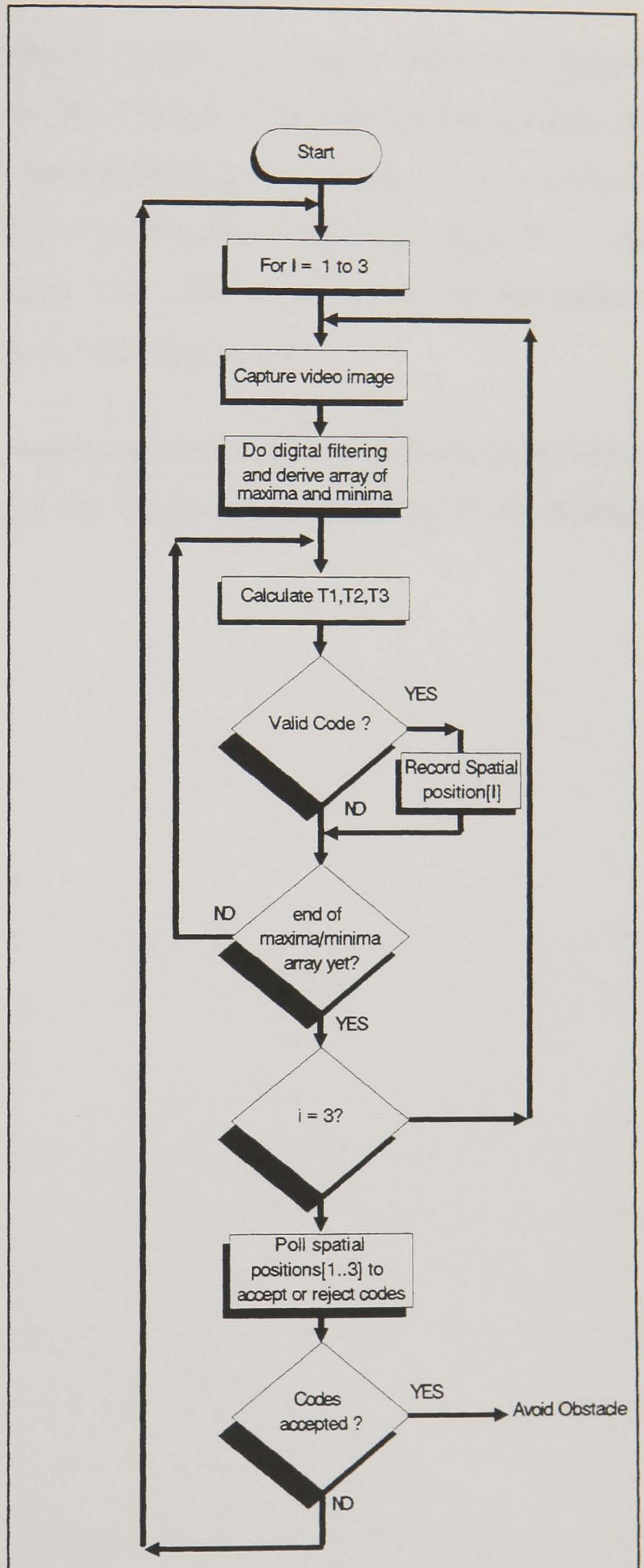


Figure 6.3.2
Flow Chart of Code Detection Algorithm

the complete obstacle detection algorithm in figure 6.3.2, video frames are captured and processed as described in chapter 5. The resultant array representing maxima and minima is processed and groups of values representing T1, T2 and T3 are calculated. These values are then tested against the feature template as shown in figure 6.3.1, and appropriate motion control action is taken. Full software listings for all the obstacle detection and associated programs can be seen in appendix 3.

In the next two chapters the practical implementation of the obstacle detection system on an experimental mobile platform and the design and development of the obstacle avoidance algorithms are described.

7 EXPERIMENTAL VEHICLE DESIGN

7.1 Summary

The next two chapters are devoted to the development of the obstacle avoidance strategy and the practical aspects of evaluating the obstacle avoidance system. In preparation for these topics, this chapter describes the experimental vehicle on which the system is mounted.

Two alternative AGV designs are discussed in section 7.2 with particular reference to their manoeuvrability and control. The design chosen for the experimental vehicle is based on a differential drive arrangement where steering is achieved by controlling the relative velocity of two drive wheels.

An overview of the physical design of the experimental vehicle is given in section 7.3. The completed vehicle is illustrated with the aid of photographs which clearly show how the camera and projector are mounted.

Dedicated single-chip microcontrollers are used to control the experimental vehicle drives. These are described in the final section of this chapter with reference to the interface with the Intel 8031 embedded controller (introduced in chapter 4). The various outputs and control modes available on the motor controllers are discussed together with the method used to select suitable digital control parameters.

7.2 Review of AGV Drive Configurations

Automated Guided Vehicles fall into two basic design categories, classified by the

method used to drive and steer them. The first type is a three wheeled arrangement as shown in figure 7.2.1. In this design, the single front wheel serves as a combined steering and drive wheel whilst the other two rear wheels idle. The three wheeled arrangement is well suited to 'flat back' pallet transporters because the combined drive and steering head can take the form of a compact unit at the front of the vehicle. This leaves a large rear area available for a low-profile pallet fork lift. However, a disadvantage of the design is that it can be difficult to control. This arises in bidirectional systems because the steering geometry is different depending on whether the AGV is travelling forwards or backwards.

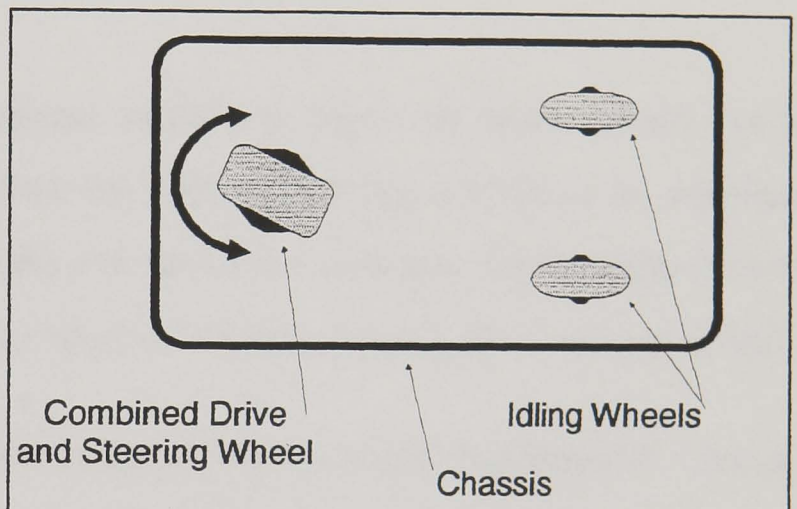


Figure 7.2.1
Three Wheel AGV Design

The other major AGV design uses two drive wheels situated centrally under the vehicle as shown in figure 7.2.2. Stability is achieved by using idling casters to support the front and rear of the vehicle and steering is achieved by varying the relative velocity of the two drive wheels. This technique is called differential steering and has advantages over the three wheeled design in certain operating conditions. The main advantage is that control of the vehicle is simplified by the fact that the steering geometry is the same for both directions of travel. Also, by driving each wheel at the same speed but in opposite directions, this

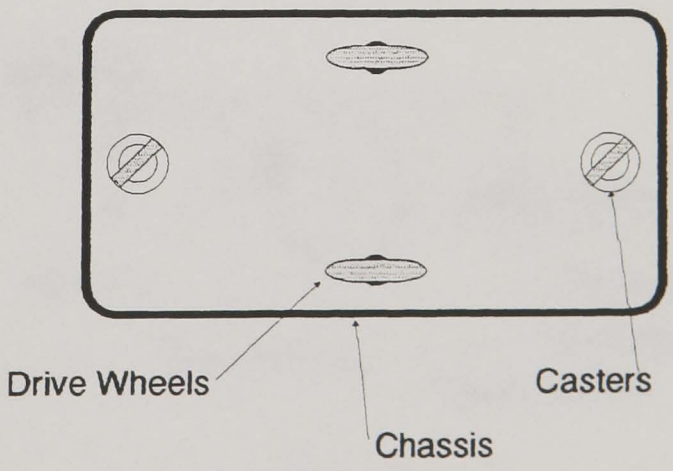


Figure 7.2.2
Differential AGV Drive Arrangement

type of vehicle can literally 'turn on the spot' in its own length (this is impossible with the three wheeled design).

The experimental vehicle uses a differential steering arrangement because the ability to turn in a small area is essential for a laboratory environment where space is at a premium.

7.3 Overview of the Experimental Vehicle Design

The physical design of the experimental vehicle is based on work carried out by Korean researchers[67]. They investigated the optimal steering control of an automated vehicle with two motorised drive wheels and developed optimum relationships between various dimensions. These include the radius of the drive wheels and their separation.

Due to the space limitations of the laboratory environment, the experimental vehicle is smaller than commercial AGVs with overall measurements of approximately 1 metre long by 0.5 metre wide. The chassis is constructed from lightened angle-iron ('Dexion'). The vehicle is equipped with a cantilever suspension system as illustrated in figure 7.3.1 to assist in damping vibrations transmitted through the drives. The 90 amp-hour battery is mounted in a tray towards the front of the vehicle whilst the motor drive electronics are mounted on an

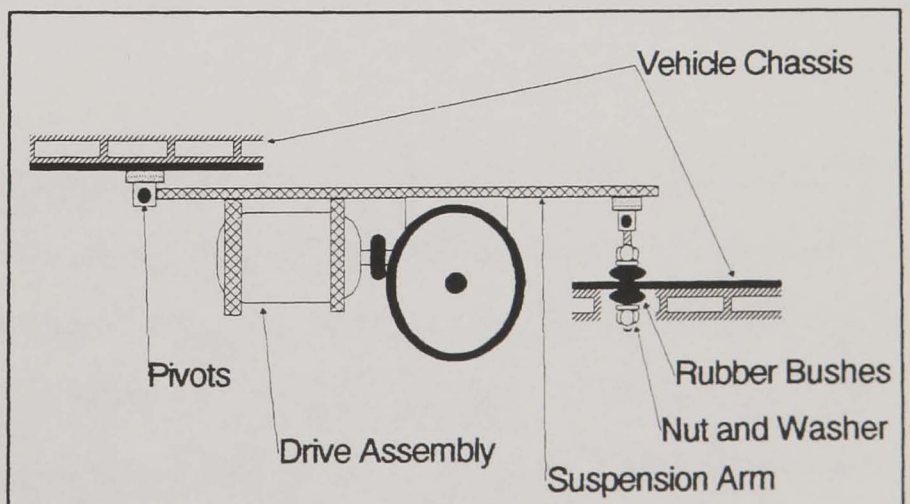


Figure 7.3.1
Experimental Vehicle Suspension System

aluminium heat sink which fits over the rear of the chassis. An emergency stop switch and ammeter are also mounted on the rear of the vehicle. A thermal circuit-breaker is connected directly in the motor power supply circuit to prevent damage to the electrical gear in the event of an overload.

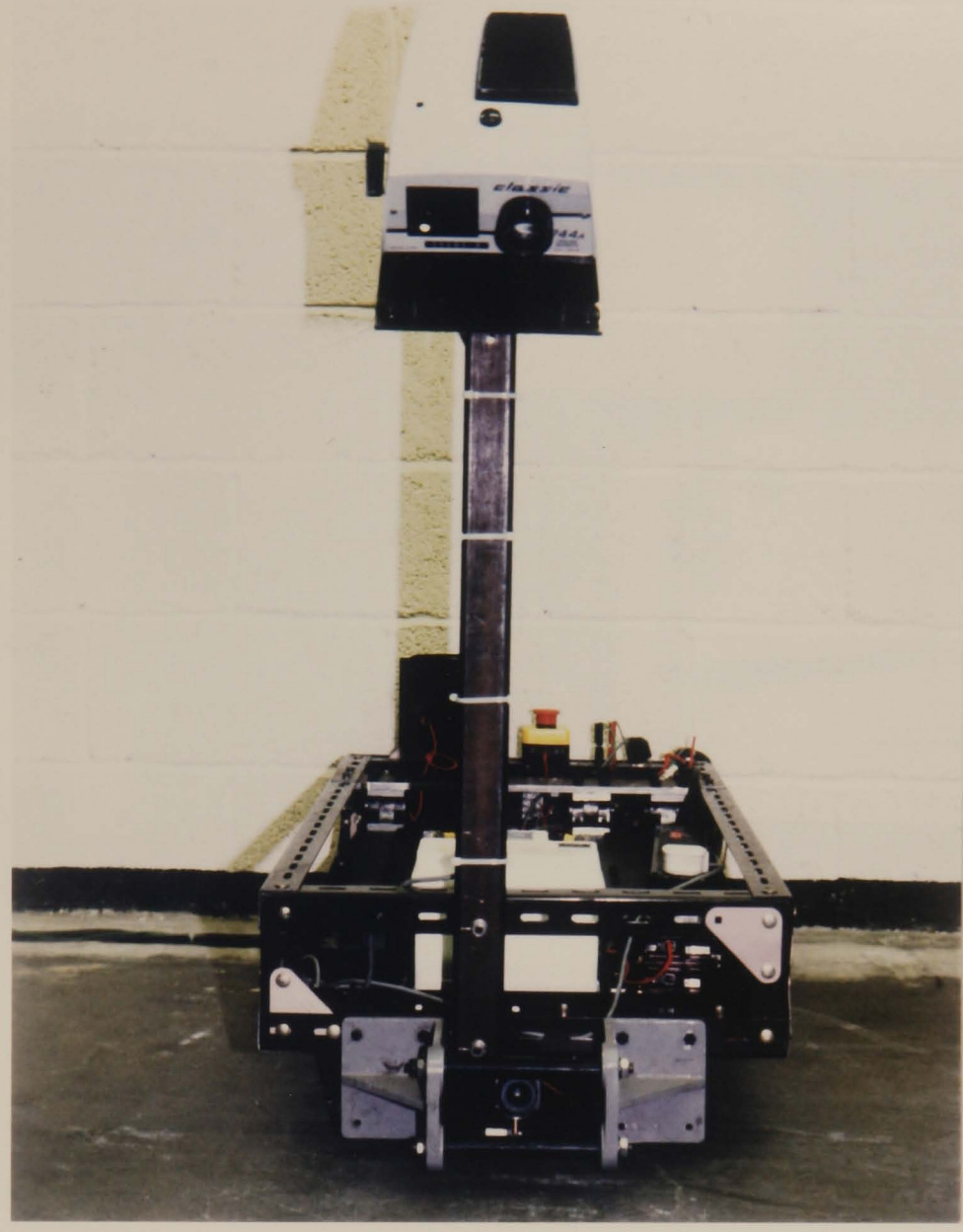
A special mounting post has been fitted to the prototype vehicle to carry the coded light pattern projector and CCD camera. Additional 'crash bars' are included to prevent accidental damage to the latter. The photographs of figure 7.3.2 illustrate the completed vehicle.

The relationship between the light pattern projector and the CCD camera can be seen from the photographs. The projector is mounted approximately 1 metre above the floor at an angle of approximately 45 degrees and the camera is in the protected position approximately 70mm from the floor. Figure 7.3.3 shows a more detailed view of the camera mounting. The physical relationship between these two items is not critical since the automatic calibration procedure (described in chapter 5), measures the code features of the light pattern after the camera and projector are fixed.

7.4 HCTL - 1100 Microprocessor-Based Motor Controllers

Figure 7.4.1 shows the schematic diagram of the experimental vehicle drives. Two 350 watt DC motors are mounted parallel to the longitudinal axis of the vehicle coupled to 'backlashless' 5:1 worm reduction gear boxes. The drive wheels are mounted directly on the gear box output shafts and are fitted with hard rubber tyres for good grip on the floor. The worm drive shafts are coupled to 500 pulse/rev optical encoders. These have quadrature outputs to allow the direction of rotation to be sensed. Figure 7.4.2 shows the block diagram of one motor drive.

The controller section of the system is realised using Hewlett-Packard type



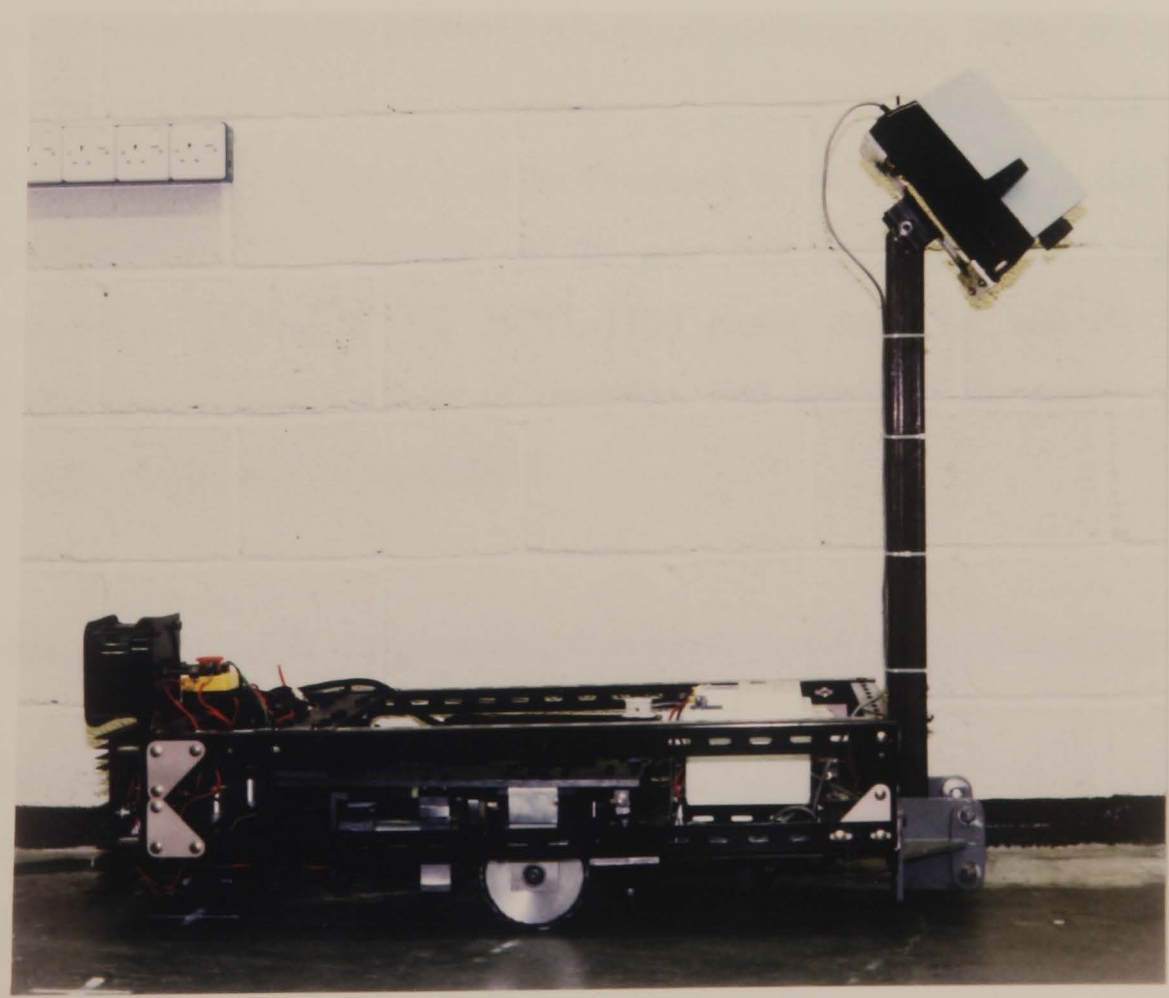


Figure 7.3.2
Experimental Vehicle

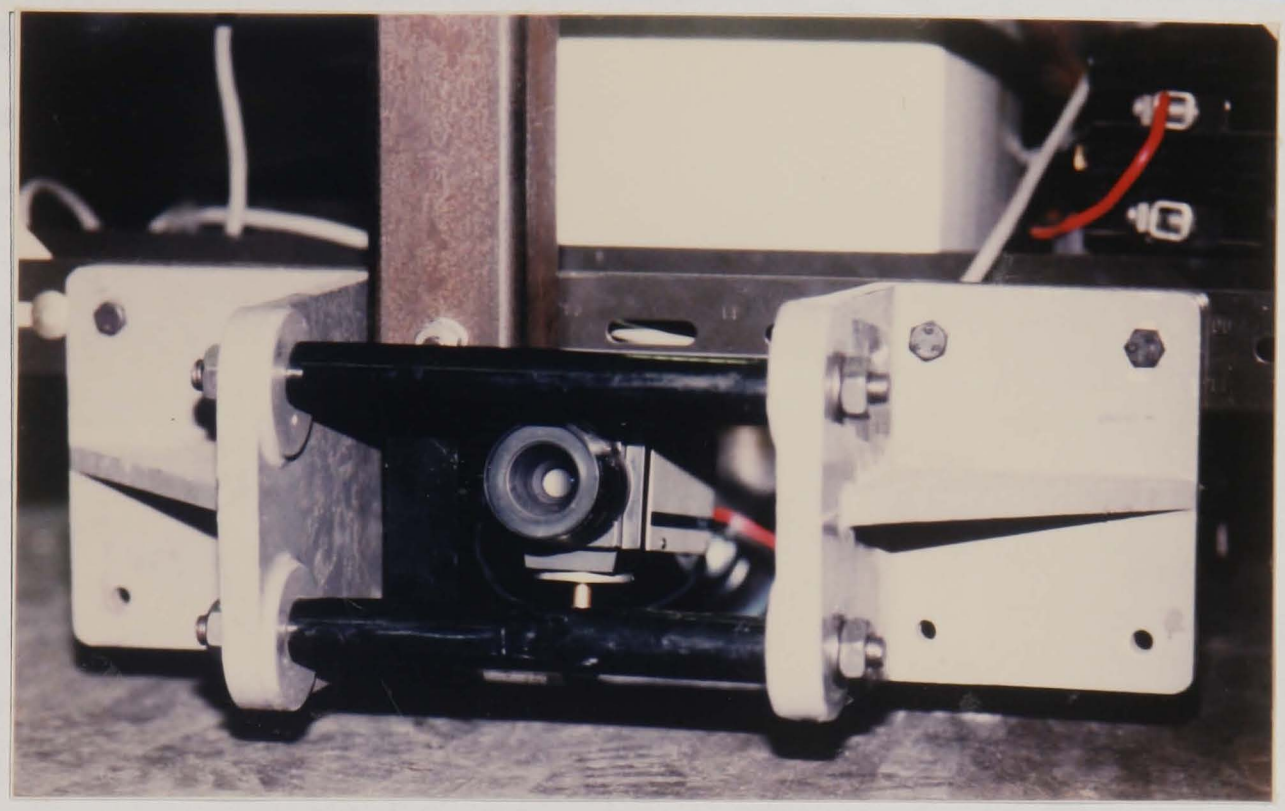


Figure 7.3.3

Camera Mounting Detail

HCTL-1100 general-purpose motion control integrated circuits. These are single-chip microcontrollers that incorporate firmware to perform common control functions required by many drive control systems. Feedback to the controllers is taken directly from the optical encoders mounted on the gear box shafts. Figure 7.4.3 shows the functional block diagram of the HCTL-1100 motor controller and the following sections discuss its various modes of operation. Full details of the specifications and circuit design can be seen in appendix 4.

7.4.1 Interface with Intel 8031 Embedded Microcontroller

The 8255 Peripheral Interface Adapter (PIA) included in the Intel 8031 embedded controller circuit was discussed in chapter 4. This provides 24 bidirectional input-output lines that are used to interface with the HCTL-1100 motor controllers.

The basic operation of the interface is that variables and constants are written to the motor controllers in an initialisation phase which establishes the mode of operation. This includes the mode of control to be used, digital filter constants and the sampling interval. After initialisation, the Intel 8031 controls the motors by writing demand values to the HCTL-1100 motor controllers. Full closed-loop control is carried out by the HCTL-1100s independently of the Intel 8031 which can be devoted to the task of detecting obstacles. Data is transferred between the Intel 8031 and the HCTL-1100s in parallel via the 8255 PIA. Communication is asynchronous to alleviate the need for accurate timing requirements which simplifies the interface. The Hewlett-Packard motor controllers appear as banks of registers to the Intel 8031. Each register has a particular function and may be write-only, read-only or bidirectional. A suite of assembler

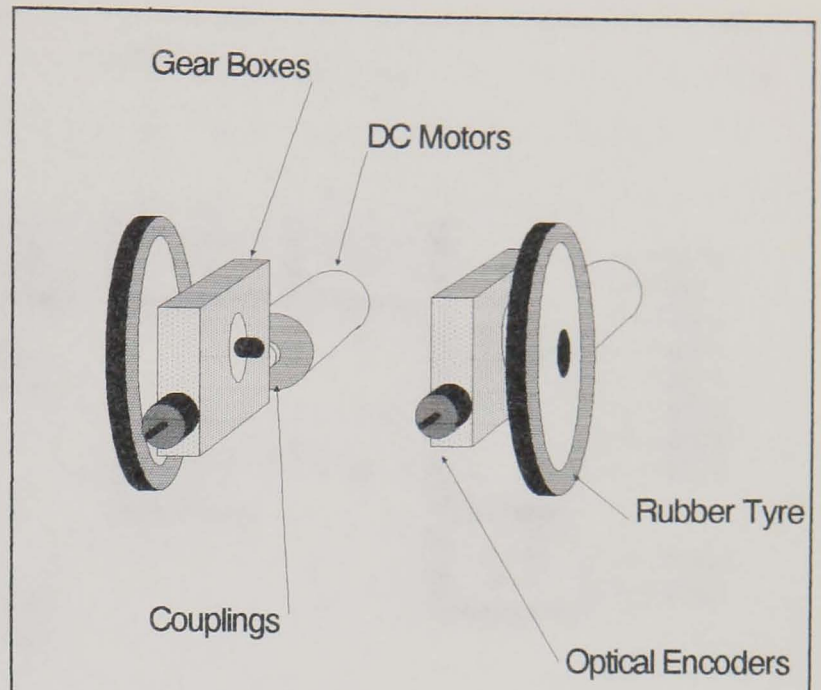


Figure 7.4.1
Experimental Vehicle Drives

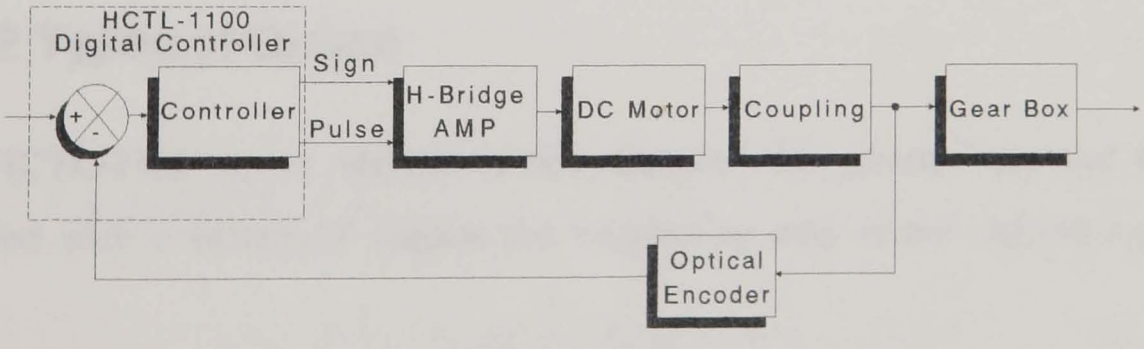


Figure 7.4.2
Experimental Vehicle Drive Block Diagram

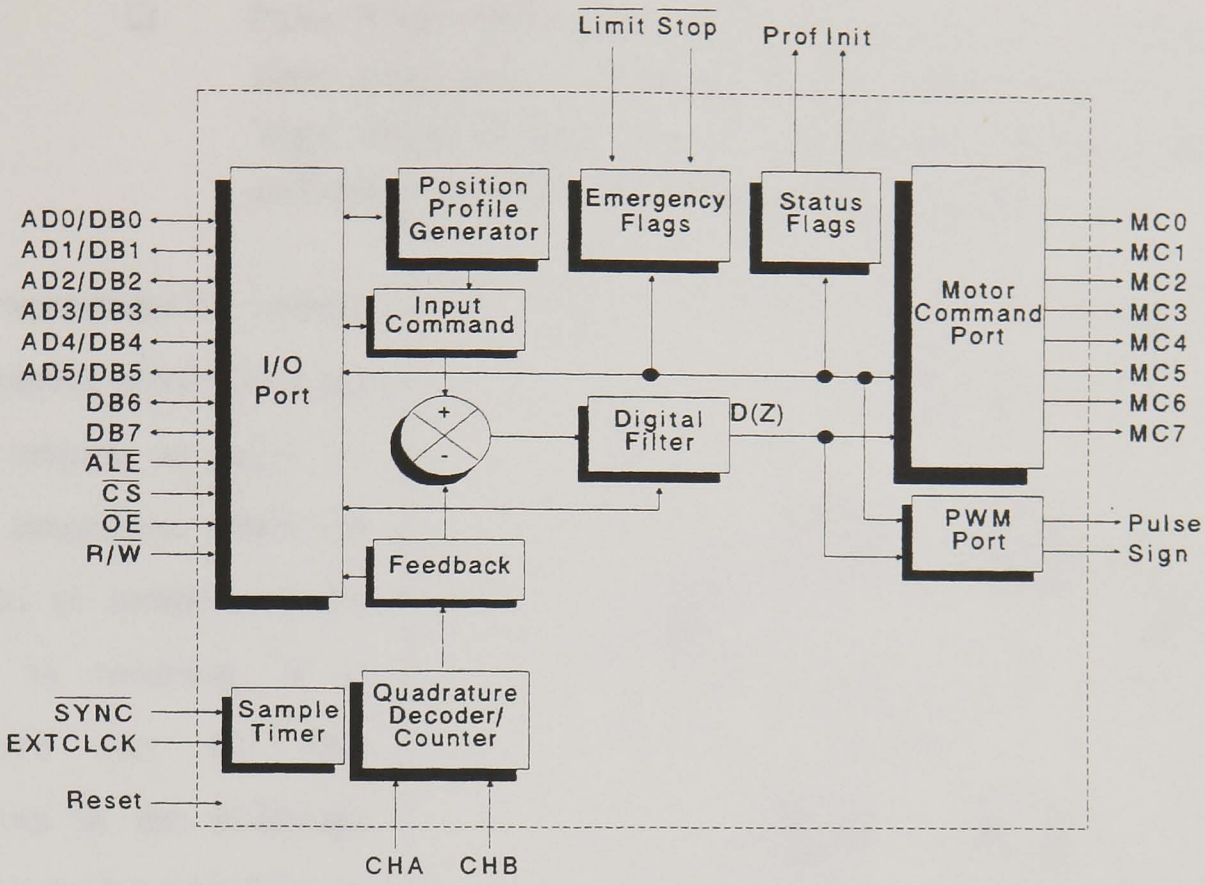


Figure 7.4.3

HCTL-1100

Simplified Functional Block Diagram

routines have been developed to operate the interface between the Intel 8031 and HCTL-1100s, full descriptions and listings of which can be seen in appendix 5. Overviews of the types of output and control modes available using the HCTL-1100 follow.

7.4.2 Types of Output

The HCTL-1100 motor controllers are designed for general use and therefore are supplied with a variety of outputs for interfacing with motor control systems. These are:

- 8-bit digital output for driving digital to analogue converters. This output can be configured in an unsigned (unipolar) format or a two's complement (bipolar) format.

- Pulse Width Modulated (PWM) sign and pulse outputs. These allow connection to H-Bridge amplifiers (see figure 7.4.2.1). The 'sign' output is used to reverse the polarity of the motor current and thus reverse the direction of motor rotation.

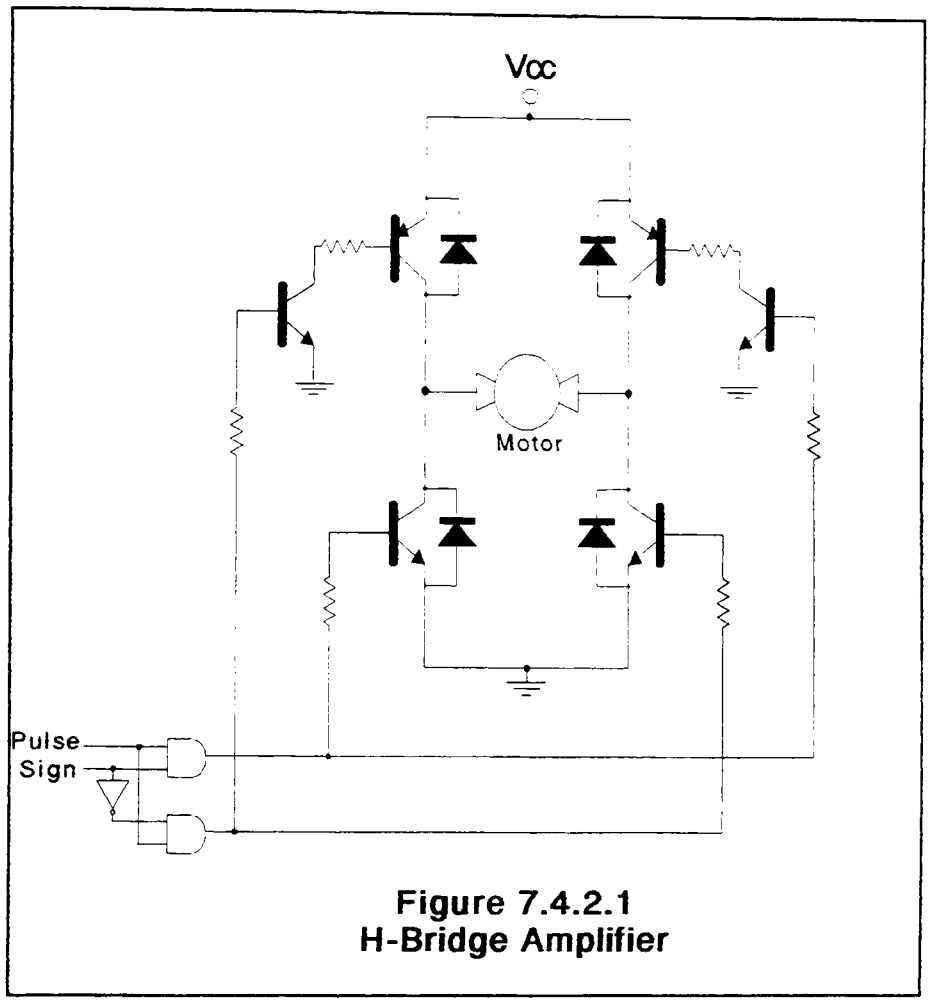


Figure 7.4.2.1
H-Bridge Amplifier

The experimental vehicle drive control systems use the PWM outputs to drive H-Bridge amplifiers. When the direction of rotation of the motors is reversed, it is possible for all the transistors in the H-Bridge amplifier to be switched on at the same time (refer to figure 7.4.2.1). This may result in a short circuit across the power supply which could damage the circuit. The HCTL-1100s can be configured to prevent this by missing a pulse at the time when the sign output changes state. This ensures that all the transistors are turned off at the instant of transition. Figure 7.4.2.2 shows the timing diagram illustrating this feature.

7.4.3 Types of Control

The HCTL-1100 motor controllers are pre-programmed to perform a range of control functions. In all cases some part of the digital filter equation shown below is used in the control loop.

$$D(Z) = K(Z-A/256)/(Z+B/256)$$

Where:

K = Digital filter gain

A = Digital filter zero

B = Digital filter pole

This filter is preprogrammed in firmware on the HCTL-1100 motor control chips and is used to provide the following modes of operation:

Proportional Velocity Control. The HCTL-1100 uses the following digital control algorithm to achieve proportional control of the motor speeds:

$$MCn = (K/4)Yn$$

where:

MCn = Motor command output at time n

Yn = (Command velocity - Actual velocity) at time n

When this control option is initialised, the demand speed is supplied to the relevant HCTL-1100 registers (see appendix 4) in the form of a 16-bit two's complement number. The HCTL-1100 interprets this as 12-bits integer and 4-bits fraction, with positive two's complement numbers resulting in rotation in one direction and negative two's complement numbers rotation in the opposite direction. The value written to the motor

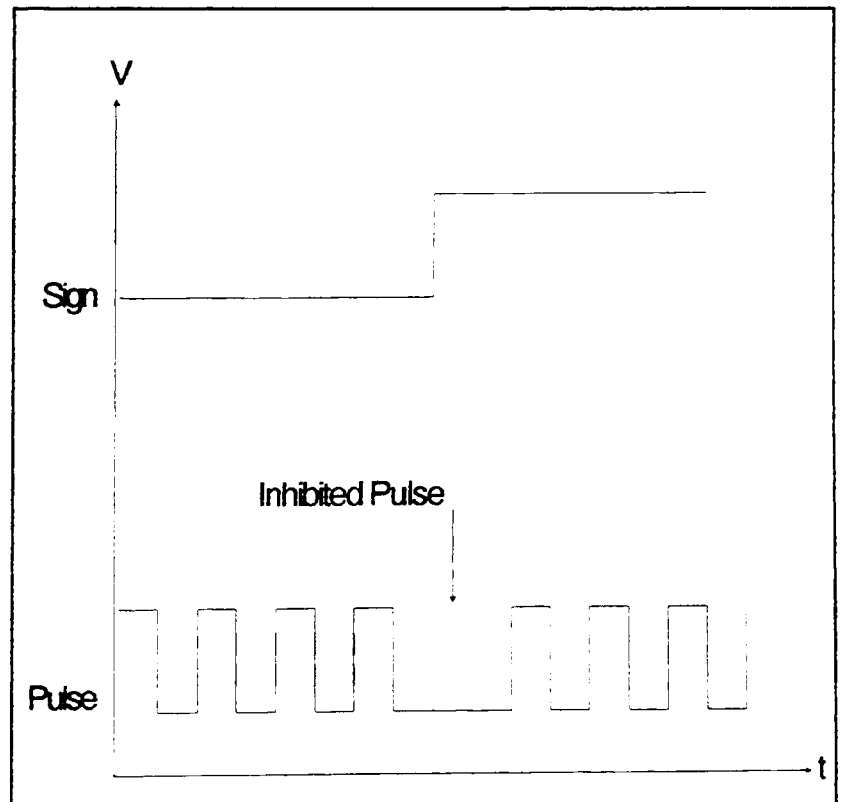


Figure 7.4.2.2
PWM Motor Controller Output Signals

controller physically corresponds to feedback encoder quadrature counts/sample time. Feedback encoder quadrature counts are derived from incremental pulse encoders mounted on the drive gearbox shafts. Each encoder produces two square wave signals with frequency in proportion to the rotational speed of the shaft. The two signals are 90 degrees out of phase with each other (one quadrant) which allows the direction of rotation to be established by detecting the 'leading' signal.

The sampling time of the HCTL-1100 motor controller is derived from the master clock frequency and a divisor stored in the sample timer register (see reference 72). In the prototype design the master clock frequency is 2 MHz and the value written to the sample timer register is 0F (hex). This results in a sampling interval of 128 microseconds which is sufficiently high to have little effect on the stability of the motor drive systems.

Integral Velocity Control. In this mode the HCTL-1100 performs continuous velocity profiling. Velocity is specified as an 8-bit two's complement number and acceleration by a 16-bit value. The acceleration value is interpreted as a scalar with 8-bits integer and 8-bits fraction (for accuracy). The units of velocity and acceleration are expressed in encoder quadrature counts/sample time and quadrature counts/(sample time)^2 respectively. Whilst this mode of control allows acceleration to be specified and therefore reduces the effects of manufacturing tolerances, it is not ideal for this work because it does not allow moves to be defined in terms of position.

Proportional + Integral Trapezoidal Profiling. For this control mode the HCTL-1100 uses the same algorithm as position control, but deviates from it in that the acceleration, maximum velocity and final positions are supplied to the motor controllers. This mode executes 'trapezoidal' motion profiling as illustrated in figure 7.4.3.1. Acceleration is supplied as a two byte scalar representing encoder quadrature counts/(sample time)^2 and velocity is expressed as a single byte scalar corresponding to quadrature counts/sample time. As before, position is given as a three byte two's complement number in units of

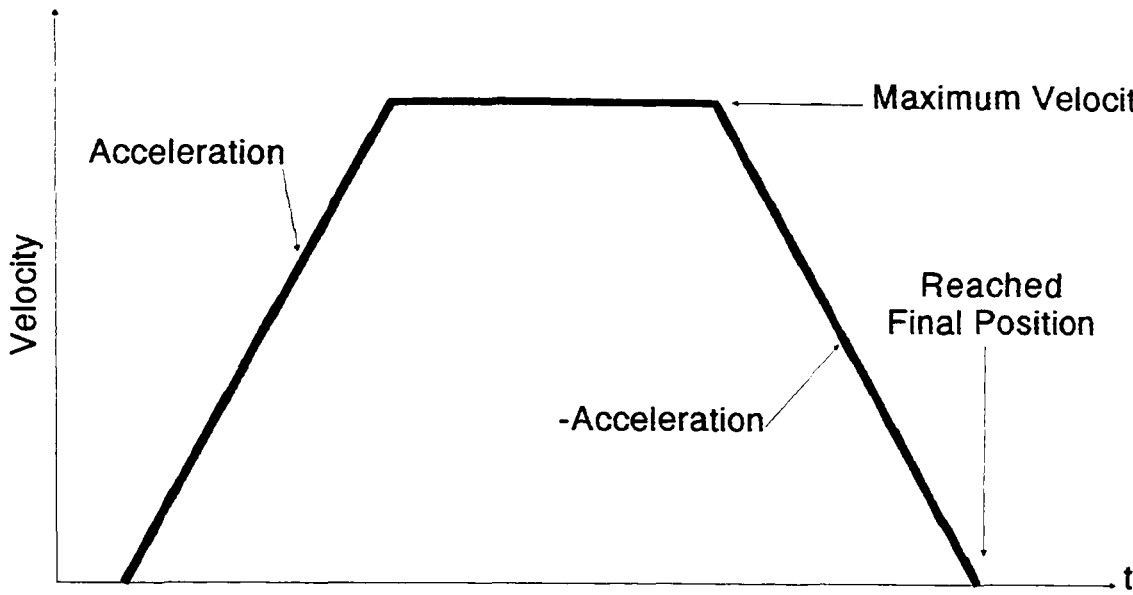


Figure 7.4.3.1
HCTL-1100
Trapezoidal Profile Mode

encoder quadrature counts. The HCTL-1100 determines which direction of rotation is required by the value of the two's complement demand position relative to the actual position.

This latter form of control is employed in the obstacle avoidance system. It enables full control of the acceleration, deceleration and velocity of the mobile vehicle during point-to-point position moves. Hence the effects of physical differences between the motor drives are overcome allowing precise motion control.

7.4.4 HCTL-1100 Digital Motor Controller Tuning

In the experimental vehicle system, excessive overshoot in position moves is generally undesirable since it is potentially dangerous. The digital filter constants: K, A and B (identified at the beginning of section 7.4.3) have been set to critically damp the drive control system.

To a large extent, these constants have been determined by 'trial and error'. Whilst the DC motor dynamic responses are relatively simple to establish, the complete system

including quadrature encoders and digital controller is somewhat difficult to model. Furthermore, flexibility of the motor-gear box couplings and inevitable backlash in the gear boxes introduces a certain degree of system non-linearity. The developed software enables point-to-point move performance tests to be carried out on the experimental vehicle. The software allows the digital filter constants to be changed quickly via the IBM compatible computer-keyboard and serial communications link with the Intel 8031 embedded microcontroller. During tests, the parameters were systematically varied and the dynamic performance of the system was carefully monitored. The final control parameters resulted in an almost critically damped drive system performance. The complete time-domain digital control algorithm applied by the HCTL-1100 motor controllers is therefore:

$$MCn = Xn - (0.25X(n-1) + 0.894\ MC(n-1))$$

These empirically derived constants have been used throughout the obstacle avoidance development work described in the next chapter and have proved to give reliable system performance.

8 DEVELOPMENT OF THE OBSTACLE AVOIDANCE STRATEGY

8.1 Summary

The procedures that enable the experimental automated vehicle to avoid obstacles are described in this chapter. Section 8.2 describes the design of algorithms for steering and advancing the automated vehicle and explains the method used to measure the distance the vehicle deviates from its original path. This work contrasts with that carried out by other researchers in that the automatic vehicle guide path is treated as a vector rather an orthogonally specified 'map'. Subsections of 8.2 describe how the motion of the vehicle has been reduced to basic 'turn' and 'advance' manoeuvres upon which all higher levels of obstacle avoidance are based.

A computer simulation of the obstacle avoidance algorithms has been developed in Pascal and is discussed in section 8.3. This program models the response of the automated vehicle to unexpected obstructions in the guide path. The program uses high resolution colour graphics to simulate the vehicle, a roadway and any number of obstacles. The latter are positioned interactively using the computer keyboard and a graphic cursor.

The model has been converted to Intel 8051 assembler language for testing on the experimental vehicle. Section 8.4 highlights the differences between the computer simulation and its real implementation.

8.2 Obstacle Avoidance Algorithms

In general, automated vehicles that follow fixed paths are not equipped with sensors to determine their absolute position within a 'world' coordinate system[68]. They simply follow the path and, if they deviate from it, at best can recognise the fact and halt.

Many obstacle avoidance researchers deal with this problem by specifying the automated vehicle environment in the form of a map, complete with boundaries indicating 'go' and 'no go' regions[69,70,71]. On-board electronics keep track of the vehicle position in the form of orthogonal coordinates determined from sensory data. Thus, if an obstacle is detected and the automated vehicle deviates from the guide path to avoid it, information is available to enable it to return to the path after the manoeuvre. However, the tasks of specifying and maintaining a 'map' of the AGV operating environment are critical. They require accurate measurement and constant re-evaluation in the face of environmental change. Furthermore, systems which use dead-reckoning sensors to obtain positional information over a large area are normally subject to unacceptable cumulative errors.

In this research, a method has been devised for AGV control which avoids the labour-intensive and expensive task of detailing a factory in terms of two-dimensional coordinates. A local map is developed when an unexpected obstacle is detected. This is based on the following rules:

- The section of the guide-path upon which the AGV is travelling when the obstacle is detected remains straight for the duration of the obstacle avoidance manoeuvre.
- If an obstacle completely fills the CCD camera field of view, the decision of which direction to take is based on a priori information about the AGV guide path layout. For example, if the guide path is towards the left hand side of aisles then most space for avoiding the obstacle is likely to be available to the right. This would therefore be the most likely direction to result in successful avoidance.

- The maximum distance that the automated vehicle is allowed to deviate from the guide path during an obstacle avoidance manoeuvre is governed by the overall width of the aisles (see section 8.2.3).

When the obstacle detection system senses an obstruction in the vehicle path, two variables representing the current deviation from the path and the current heading are set to zero. As the vehicle leaves the path to avoid the obstacle, its relative deviation and heading are referred to this initial 'zero vector'. The deviation and heading are denoted by 'y' and 'theta' respectively in figure 8.2.1. The vehicle rejoins the original guide path when the deviation and heading return to their initial zero values. The equation for the deviation y is: y = r sin(theta) where r is the distance travelled on a particular heading theta (see figure 8.2.1).

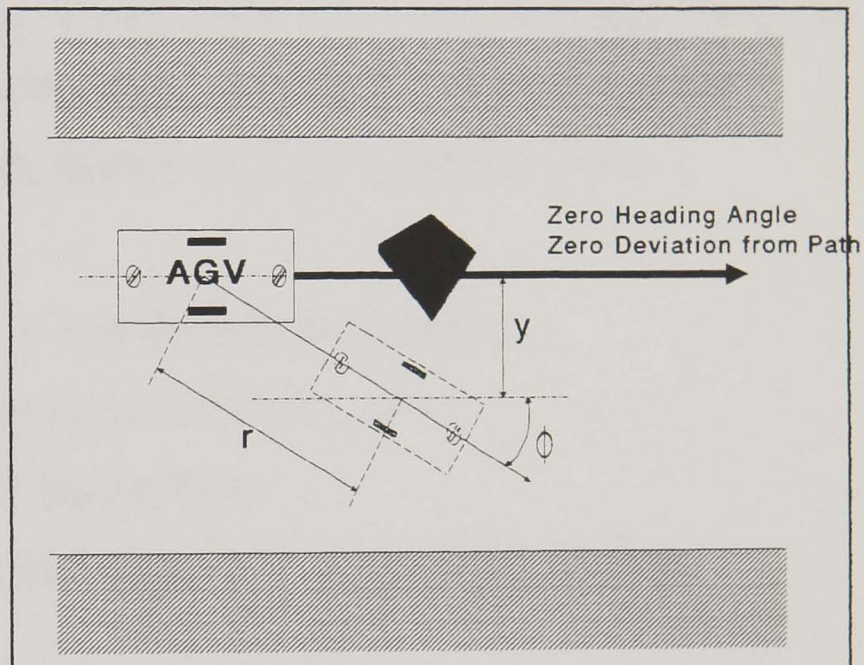


Figure 8.2.1
Automated Vehicle Heading and Deviation

The motion of the automated vehicle is simplified to a pair of 'primary' movements as follows:

- TURN(DIR): Turn the automated vehicle 5 degrees in the direction 'DIR'. Five degrees is chosen as the smallest increment of angular motion because it allows fast trigonometrical evaluation using a short look-up table. 'DIR' is a Boolean variable with TRUE representing an angular increment of -5 degrees (a clockwise turn), and FALSE +5 degrees (an anti-clockwise turn).

- ADVANCE: This primary manoeuvre advances the automated vehicle in a forward direction. The actual distance that one 'advance' moves the vehicle is set to 50 mm. This represents the increment that the automated vehicle travels in the forward direction before rechecking for obstacles (see section 8.2.2).

In general, a complete obstacle avoidance sequence is constructed from several primary TURNs and ADVANCEs, and may involve the avoidance of more than one obstacle. A running total of aggregate headings and deviations is maintained to enable the system to recover the vehicle to the guide path after circumnavigating the obstruction. The following pseudo-code operations explain how this is achieved:

NEW_HEADING = OLD_HEADING + (5 degrees * DIRECTION)

where:

DIRECTION = -1 for an anti-clockwise turn

DIRECTION = +1 for a clockwise turn }

NEW_DEVIATION = OLD_DEVIATION +

(ADVANCE_INCREMENT * sin(NEW_HEADING))

The complete obstacle avoidance procedure is broken down into subroutines. These are described in the following sub-sections under titles assigned in the actual software source code (see appendices 3 and 6):

8.2.1 'LeftOrRight' Subroutine

This subroutine determines which direction to turn to avoid an obstacle. The detection system provides information for obstacle avoidance in the form of the three Boolean variables: OBSTFROMLEFT, OBSTFROMCENTRE, and OBSTFROMRIGHT. These variables indicate whether an obstacle is emerging from the left, centre or right of the CCD camera field of view and are determined by dividing the video image into three equal regions. The obstacle detection software ascertains where obstacles lie in relation

to the AGV by establishing which region the detected light codes appear. The LeftOrRight subroutine uses the three Boolean variables to determine whether to turn to the left or right to avoid the obstacle. If the obstacle only occurs in the centre of the field of view, or if it fills the field of view, the decision is based on a priori knowledge of the automated vehicle environment. In the prototype, this is arbitrarily to the right. If the obstacle emerges in the left of the field of view then avoidance is attempted to the right. Conversely if an obstacle emerges from the right then the system will attempt to avoid the obstacle to the left.

8.2.2 'TURNAGV' Subroutine

Figure 8.2.2.1 shows the flow chart for the TURNAGV algorithm. All the sub-routines to be discussed return a Boolean variable 'RESULT' which is only FALSE if the automated vehicle can not pass the obstacle. All the subroutines which physically move the AGV execute a check for obstacles. This returns a Boolean variable 'OBSTDETECTED' which is TRUE when an obstacle is detected (regardless of its position in the field of view) and FALSE otherwise.

When the system detects an obstacle, the software determines which way to turn to avoid it using the LeftOrRight routine, and then turns in 5 degree increments until the obstacle no longer interferes with the projected light pattern. This results in the automated vehicle being orientated with a heading that will allow it to clear the obstacle. A further subroutine 'TESTTURN' is included, whose flow chart is shown in figure 8.2.2.2. This checks for:

- Another obstacle emerging which will prevent the vehicle turning to avoid the first one. This occurs, for example, if the AGV is in a narrow gap and while turning to the right to avoid an obstacle on the left, encounters another obstacle on the right before the one from the left is cleared. In this case the vehicle cannot avoid the obstruction and a FALSE result is returned.

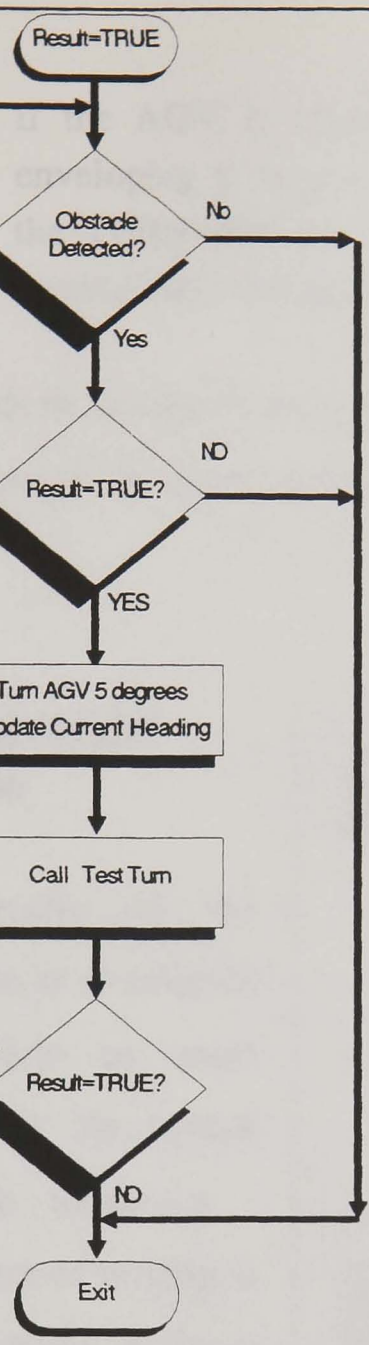


Figure 8.2.2.1
Flow Chart for 'TURN' Algorithm

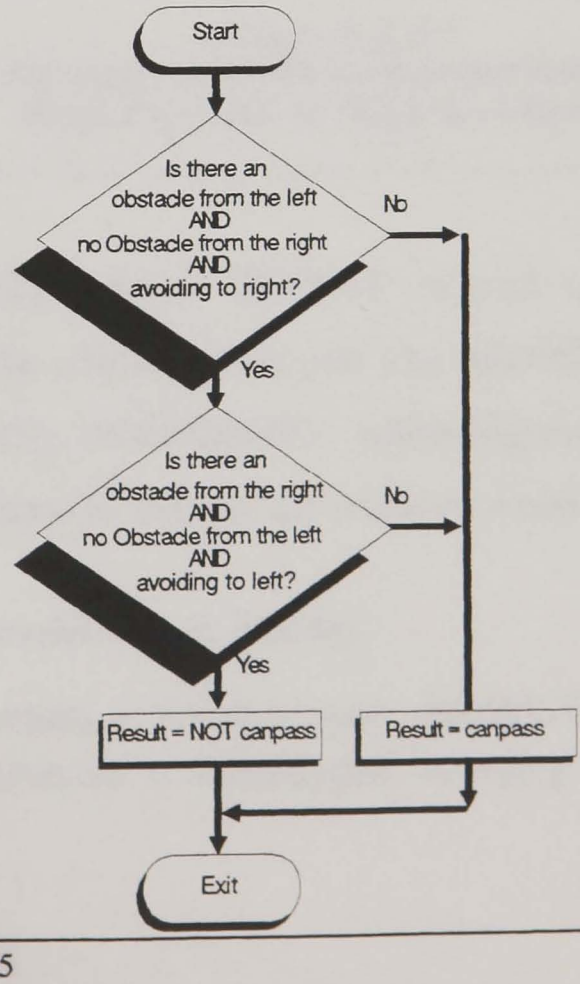


Figure 8.2.2.2
Flow Chart for 'TEST_TURN' Procedure

- If the AGV is faced with a semi-circular type of obstruction enveloping it then a maximum limit of 90 degrees is placed on the TURNAGV procedure. This prevents the automated vehicle turning back on itself.

If the vehicle succeeds in turning to avoid the obstacle, it must next advance in a series of small steps, far enough to allow recovery to the guide path without colliding with the obstacle.

8.2.3 'ADVANCEAGV' Subroutine

The primary objective of the ADVANCE procedure is to advance the automated vehicle in small increments - checking for further obstacles after each increment - until the vehicle centre of turning is past the obstacle as shown in figure 8.2.3.1.

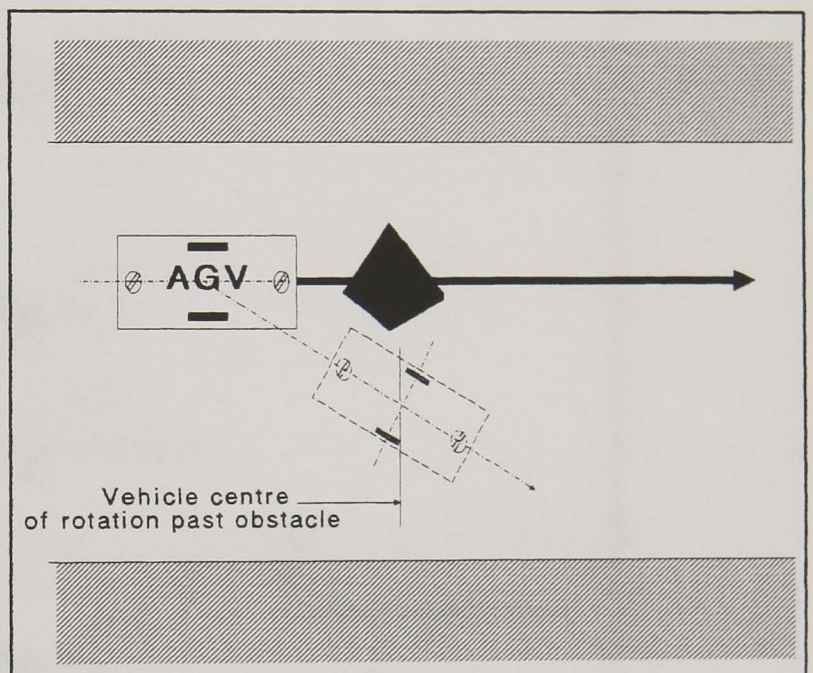


Figure 8.2.3.1
Distance that the Automated Vehicle Must Advance to Clear an Obstacle

Figure 8.2.3.2 shows the flow chart for the ADVANCEAGV procedure.

As in the previous subroutine, the Boolean variable 'RESULT' is used to indicate whether the automated vehicle can pass the obstruction or not. The ADVANCEAGV routine uses an additional Boolean variable 'PASSEDYET' which signals that the vehicle has advanced far enough past the obstacle to begin the recovery procedure.

The failure modes of the 'ADVANCE' procedure are as follows:

- The procedure will return a 'failed to pass' (RESULT=FALSE) result if a further obstacle is encountered emerging from the

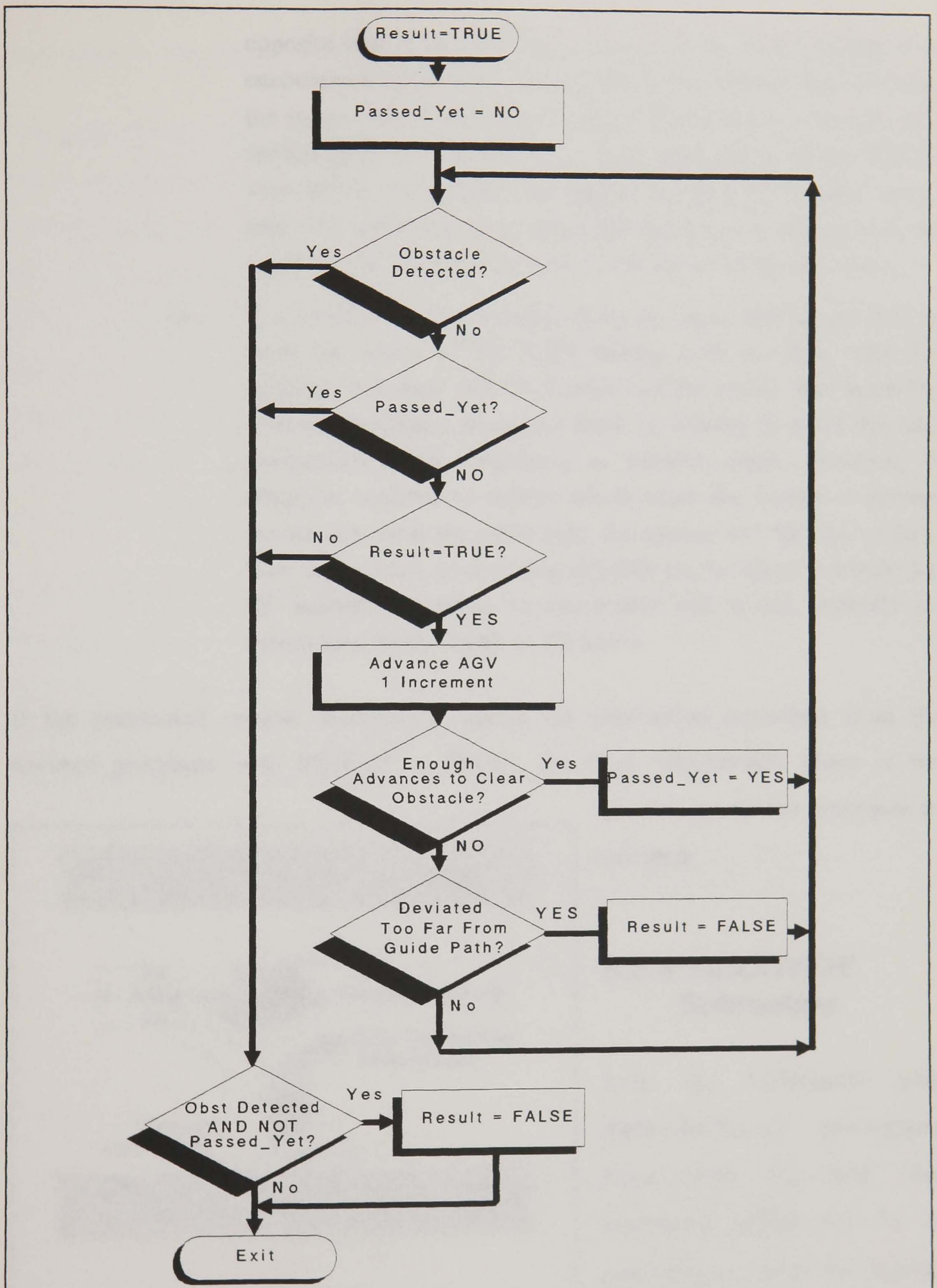


Figure 8.2.3.2
Flow Chart for 'ADVANCE' Procedure

opposite side to the first. For example, if an initial obstacle was encountered in the left region of the CCD camera field of view, the system would attempt to avoid it by turning to the right. If a second obstacle emerges in the right hand region of the field of view before the vehicle had passed the first, the system would halt. This condition arises when the gap between obstacles is too small for the vehicle to manoeuvre as shown in figure 8.2.3.3.

- If a further obstacle emerges from the same side as the first or from the centre of the CCD camera field of view, then the advance procedure will be aborted and the system will begin the obstacle avoidance procedure anew by turning to avoid the new obstruction before beginning to advance again. However, if obstacles continue to emerge which cause the vehicle to deviate excessively from the guide path, the system will halt the vehicle. This upper limit of deviation depends on the space available for the automated vehicle to manoeuvre and would normally be determined by the width of the aisles.

If the automated vehicle successfully passes the obstruction (returning from the advance procedure with RESULT = TRUE), the final 'RECOVER' phase of the obstacle avoidance procedure is initiated.

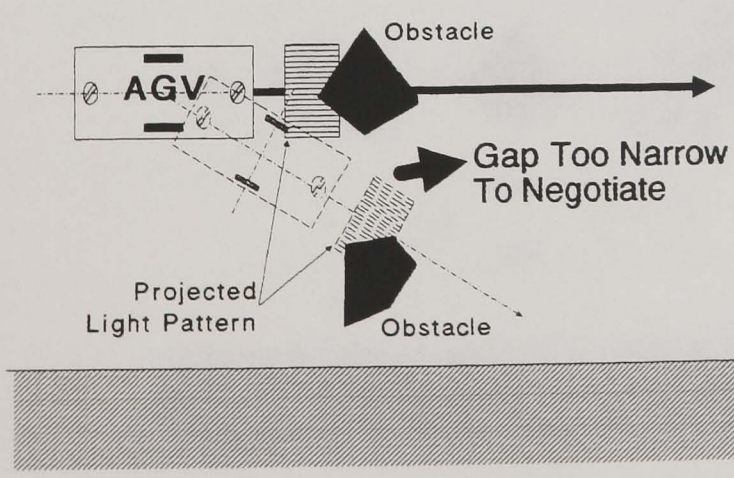


Figure 8.2.3.3
'ADVANCE' Procedure Fails if Gap is Too Small For Automated Vehicle to Negotiate

8.2.4 'RECOVER' Subroutine

After the TURNAGV and ADVANCEAGV procedures have been executed, the automated vehicle will be at some distance from the original guide path with an orientation which enabled it to pass the

final obstacle. The 'RECOVER' procedure returns the vehicle to its original trajectory.

The most direct method of achieving this would be to turn the automated vehicle so that it approached the original guide path in a perpendicular direction, and then turn it 90 degrees onto its original heading when the deviation from the path reaches zero. However, a severe disadvantage with this approach is that if the guide path runs closely parallel with a wall, the vehicle would need to turn and advance many times before returning to its original route as shown in figure 8.2.4.1. This method would eventually succeed if the wall was continuous. However, if it was discontinuous, the vehicle would at best meander back on course on an 's' shaped trajectory after overshooting the guide path.

The alternative approach developed in this work is to return the automated vehicle to the guide path on a hyperbolic trajectory as shown in figure 8.2.4.2. Using this approach, the greater the deviation from the guide path, the more severe the return heading angle. As the vehicle approaches its original guide path, the recovery angle converges with the original vehicle heading.

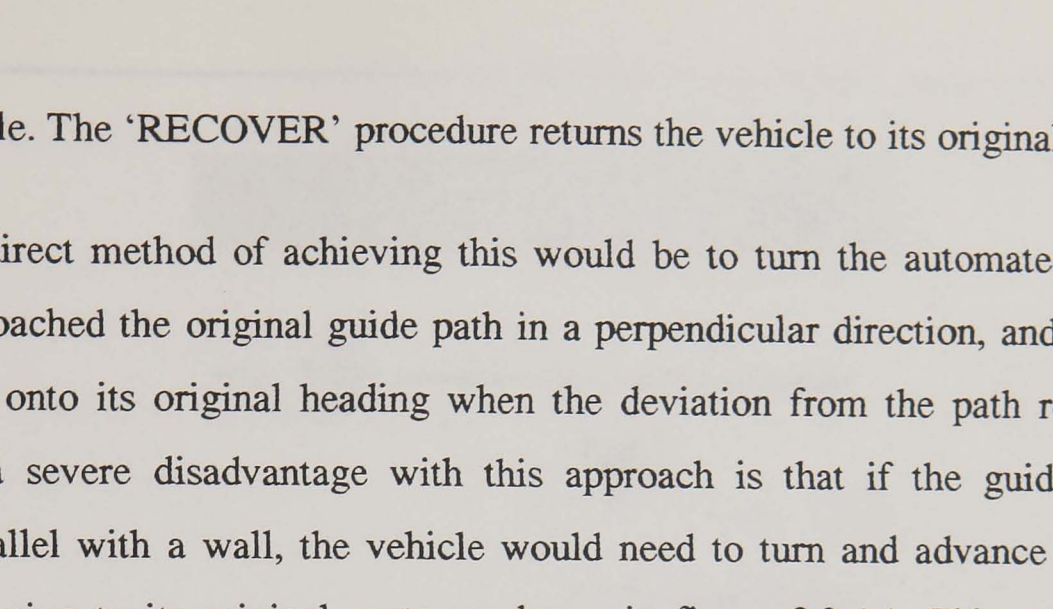


Figure 8.2.4.1
Simplistic 'RECOVER' Procedure
AGV Must Repeatedly 'Avoid' Corridor Wall

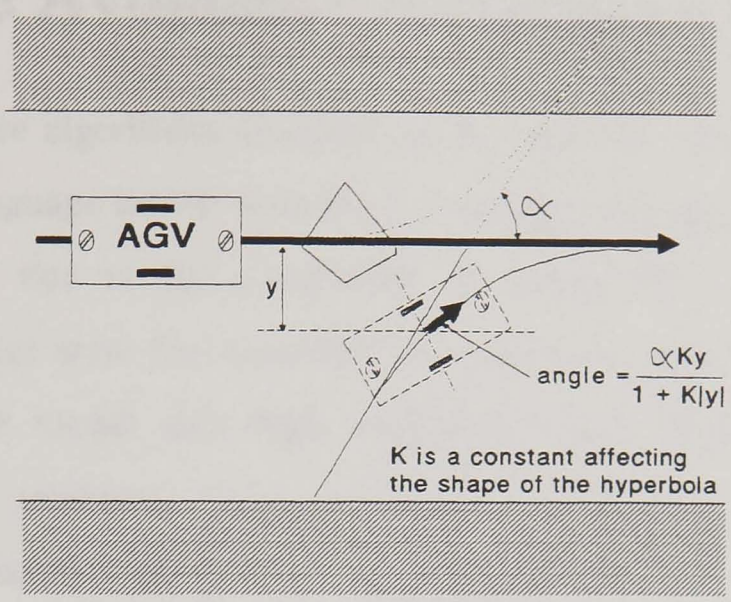


Figure 8.2.4.2

AGV 'RECOVER' PROCEDURE

AGV Returns to Guide Path on a Hyperbolic Trajectory

Figure 8.2.4.3 shows the flow chart for the recovery procedure. In order to avoid the need for floating point arithmetic, the hyperbolic equation in figure 8.2.4.2 has been reduced to a look-up table of headings (see appendix 3). Hence, for a given deviation from the guide path, a corresponding heading is determined (desired heading). The vehicle is then turned one 5 degree increment in the direction which will reduce the error: (desired heading - current heading). This process is repeated until the deviation and heading reduces and eventually converges to zero as the vehicle returns to its original guide path.

Should a further obstruction be encountered during the recovery phase of the obstacle avoidance process, the subroutine is abandoned and the whole obstacle avoidance sequence begins again until either the process fails or the vehicle regains its original path.

8.3 Obstacle Avoidance Simulation Package

The obstacle avoidance algorithms discussed in the previous sections are implemented in 8051 assembler language on the Intel 8031 embedded computer system described in chapter 4. However, due to the complexity of coding the algorithms directly in assembly language, they were first tested by developing a simulation package in Turbo Pascal. The computer model uses high resolution colour graphics to represent the automated vehicle, a roadway, and obstacles. Any number of obstacles can be placed in the simulated roadway by interactively positioning a graphic cursor on the screen.

Figure 8.3.1 illustrates the graphical models. The automated vehicle chassis is represented by a rectangle with a further rectangle for the projected light pattern. The roadway is shown between two cross-hatched regions in the upper and lower portions of the display. Obstacles are represented as random four sided polygons in the same colour as the road edges.

Procedures are included to turn

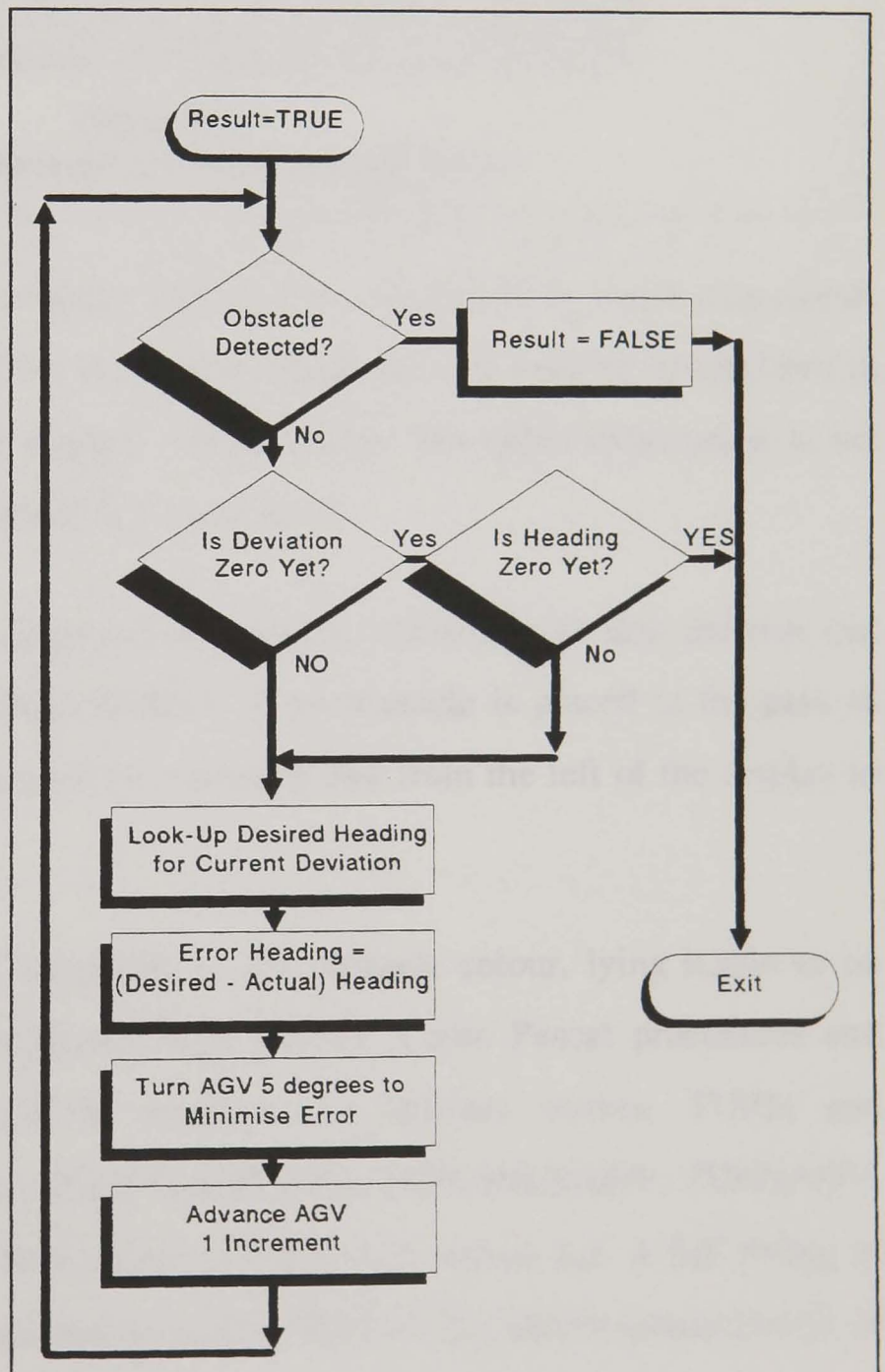


Figure 8.2.4.3
Flow Chart for 'RECOVER' Procedure

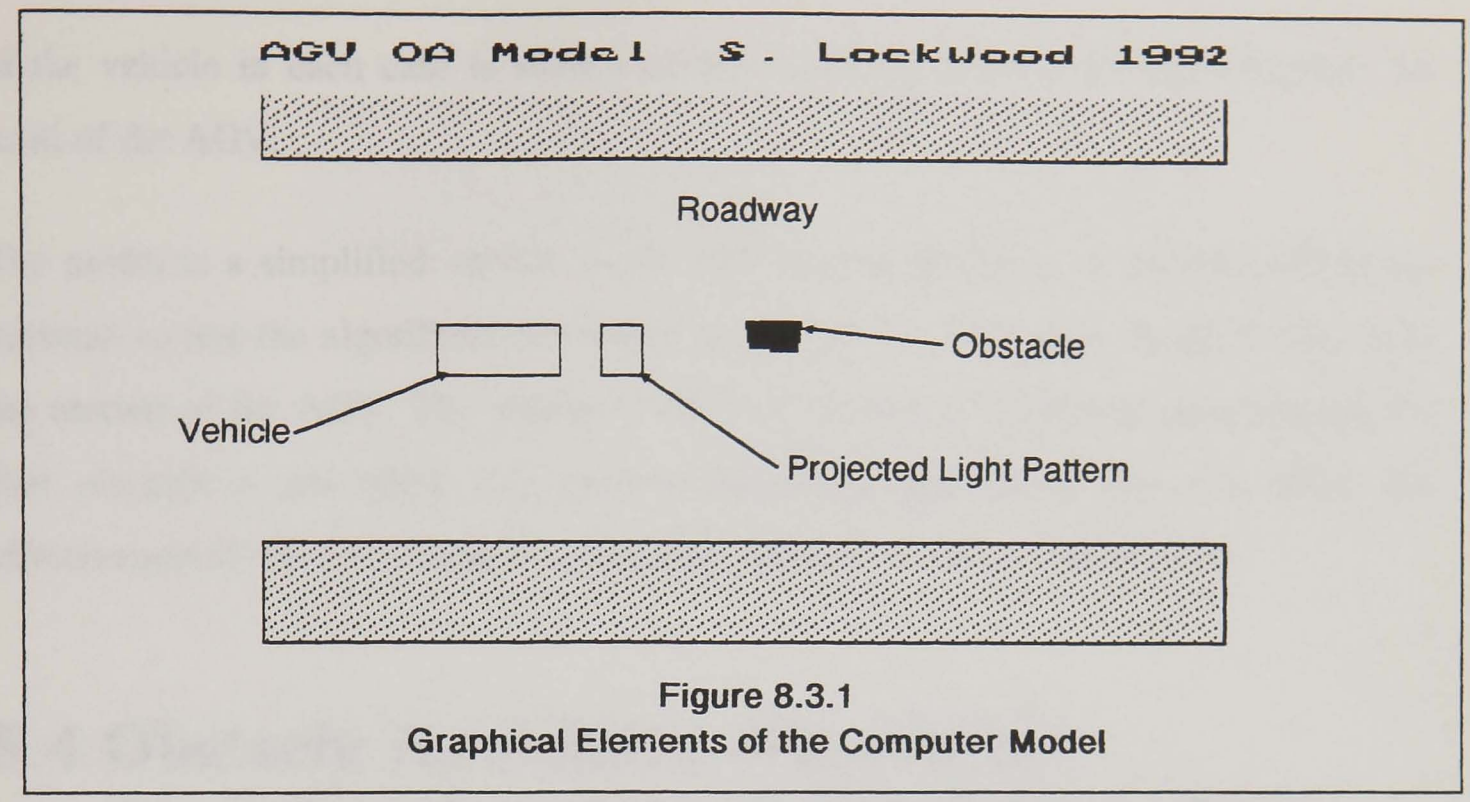


Figure 8.3.1
Graphical Elements of the Computer Model

the AGV model in 5 degree increments and also to advance it in small increments. Information about the position of the simulated vehicle and its heading is tabulated in the upper left hand corner of the display. To aid clarity, the latter information is not shown in the 'screen dumps' of figures 8.3.1 and 8.3.2.

Each time the AGV is turned or advanced, it is erased, redrawn in its new position and orientation, and the information table updated. If no obstacle is placed in the path of the AGV model it will simply proceed on a straight line from the left of the display to the right.

Obstacles are detected by testing for pixels of the obstacle colour, lying inside or on the boundary of the simulated projected light pattern. Turbo Pascal procedures and functions have been implemented to simulate the primary moves: TURN and ADVANCE and the secondary operations: LeftOrRight, ADVANCEAGV, TURNAGV, and RECOVER according to the flow charts developed in section 8.2. A full listing of the AGV system model can be seen in appendix 6. Figure 8.3.2 shows screen dumps of the model when faced with various obstacle situations. Figure 8.3.2a shows the response to single obstacles, 8.3.2b two obstacles and 8.3.2c three obstacles. The path

of the vehicle in each case is shown using a modified procedure which displays the 'trail of the AGV.

The model is a simplified version of the real system. However, it proved sufficiently accurate to test the algorithms developed in section 8.2. The main simplification is in the motion of the AGV. The inertia of the real vehicle as it halts on encountering the first obstacle is not taken into account. However, this factor does not affect the effectiveness of the simulation in testing the obstacle avoidance subroutines.

8.4 Obstacle Avoidance Algorithms Transferred to the Intel 8031 Embedded Microcontroller

The Pascal simulation program was converted to 8051 assembler language and tested on the real experimental vehicle. The only significant structural differences between the real system software and the model, are the language used for the source code and the method used to evaluate the sine function. In the Turbo Pascal model, this is carried out using the floating point sin(x) function included in the package. Floating point arithmetic has been avoided in 8051 assembler language by implementing a look-up table. Since the experimental vehicle is only moved in 5 degree increments as described earlier, the table needs only to hold sine values for multiples of five degrees. In order to both achieve accuracy and restrict the look-up table values to single bytes, the sine values have been stored as Ksin(x) where K is 256. The actual value of deviation is maintained as a two-byte two's compliment number with positive and negative values depending on which side of the guide path the vehicle is situated. Full listings of the assembler language obstacle avoidance program and the look-up tables are presented in appendix 3.

The subroutines developed in the preceding sections were evaluated on the real experimental vehicle system and the results are described in the next chapter.

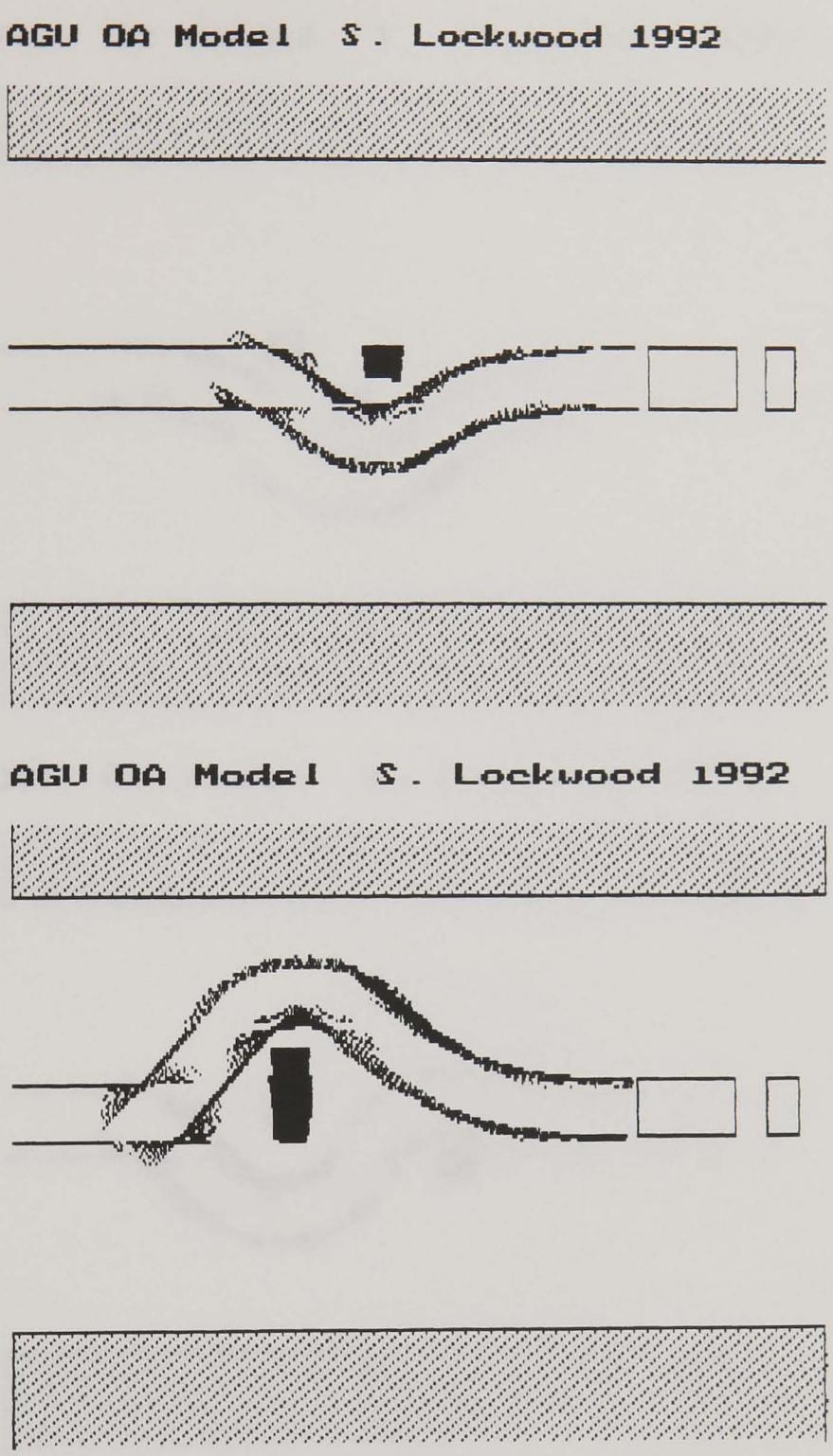


Figure 8.3.2.a
Single Obstacle Avoidance

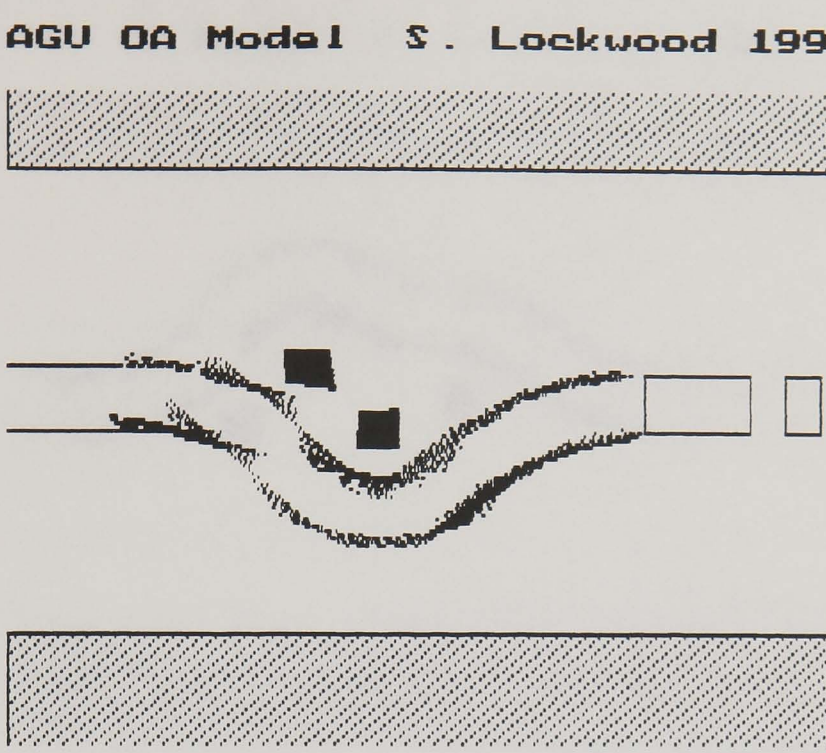


Figure 8.3.2.b
Double Obstacle Avoidance

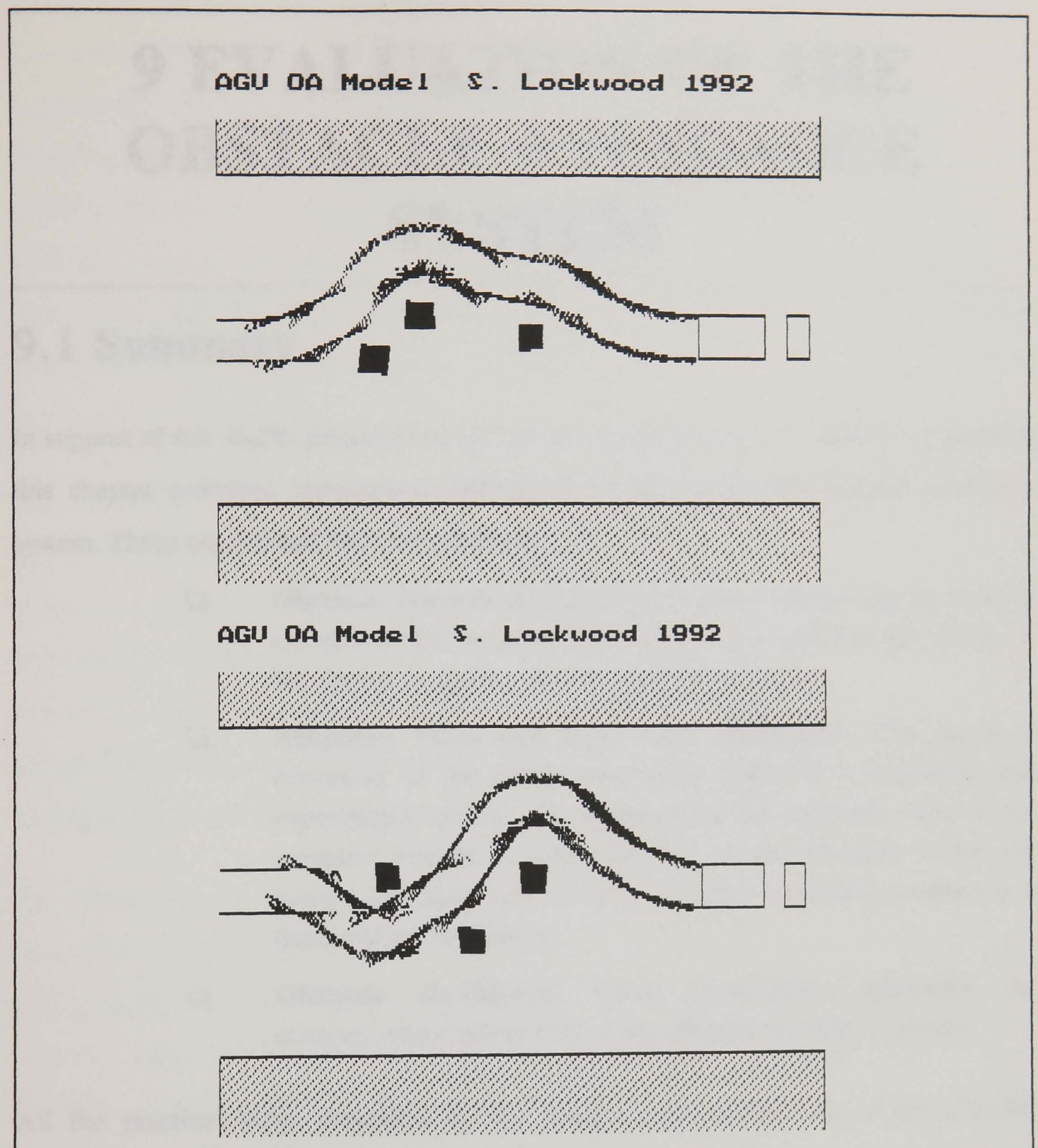


Figure 8.3.2.c
Avoidance of Three Obstacles

9 EVALUATION OF THE OBSTACLE AVOIDANCE SYSTEM

9.1 Summary

In support of the results presented as part of the design process in chapters 3,5,6 and 8, this chapter describes experiments carried out on the completed obstacle avoidance system. These are divided into three sections:

- **Obstacle Detection**. Experiments were carried out to confirm the size of the smallest detectable object and test the ability of the system to sense a diverse range of obstacles.
- **Response Time and Real-Time Operation**. The speed of execution of the image processing software is examined and experiments carried out to determine the response time of the complete system. A key objective of the research is that the system should operate in real-time. Tests have been conducted to demonstrate this feature.
- **Obstacle Avoidance**. These experiments determine the accuracy and repeatability of the obstacle avoidance system.

All the practical work described in this chapter was carried out in the Flexible Manufacturing laboratory at the University of Huddersfield.

9.2 Obstacle Detection System

Chapter 5 described how the system is designed to detect objects of minimum dimensions 50mm wide X 20 mm tall. This limit is imposed partly due to the method of processing video information and partly due to the physical size of the projected

light code. The moving average filter which processes images in the vertical direction requires that obstacles are tall enough to reflect at least 8 video lines. In the horizontal direction, they must be wide enough to reflect a complete projected light code. The ability of the system to detect an object of these minimum dimensions was tested by placing the 50mm X 20mm test card on the floor ahead of the vehicle as illustrated in figure 9.2.1. The resulting video camera view of figure 9.2.2 shows that a complete light code is reflected. The system successfully detected the test card and therefore the height criteria was also verified.

The hypothesis which forms the basis of the next experiment is that the use of a white light projector ensures sufficient light is reflected from objects with minimal reflective properties. An automobile radiator component with a black lustreless surface (specifically designed to minimise reflection and maximise heat absorption) was selected to test this premise. Figure 9.2.3 shows the video camera view of the component. The system successfully detected the radiator part even though the matt black surface was oblique to the projector and camera angle of incidence.

The ability of the system to detect objects with efficient reflective properties without saturating is also important. The video camera adjusts the average amount of light falling on the CCD array by means of an auto-iris. The performance of this device was tested by replacing the matt black radiator component with a large gloss white board. The video camera view of figure 9.2.4 shows that the auto-iris successfully limited the light falling on the CCD array and prevented light saturation. This can be seen by the fact that the reflected light codes remain clearly defined in the video image.

The speed of operation of the auto-iris device was tested by placing the video camera in complete darkness and then introducing it to a bright light. This was achieved by positioning the gloss white board in front of the obstacle detection system and placing a dark cover over the camera lens. The cover was quickly removed and the response of the auto-iris timed. The system took approximately 0.25 seconds to adjust. Since this is

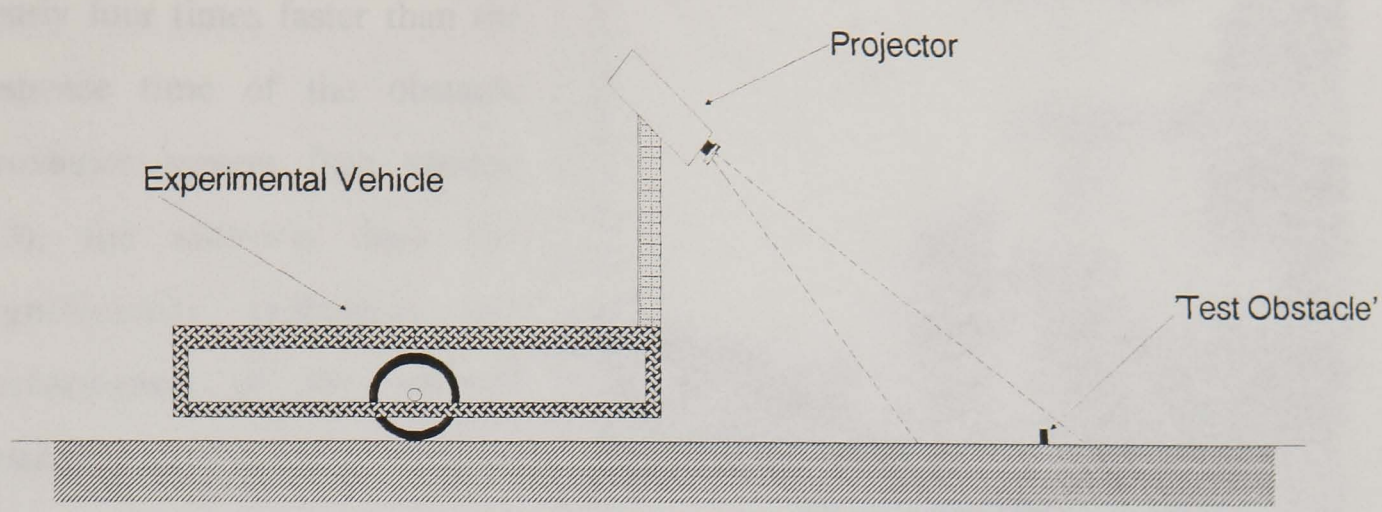


Figure 9.2.1
'Test Obstacle' Positioned in Light Pattern

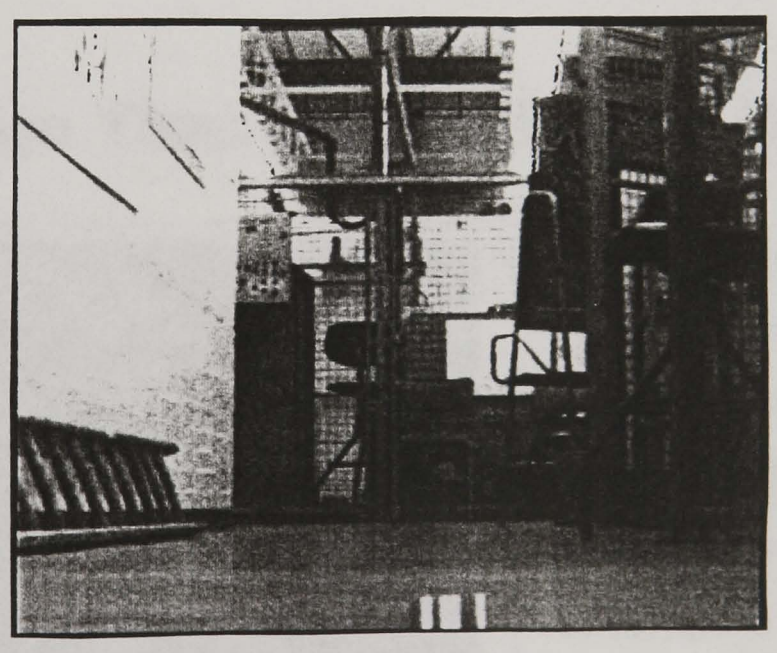


Figure 9.2.2
Video Camera View of 'Test Obstacle'

nearly four times faster than the response time of the obstacle avoidance system (see section 9.3), the auto-iris does not significantly influence the performance of the overall system.

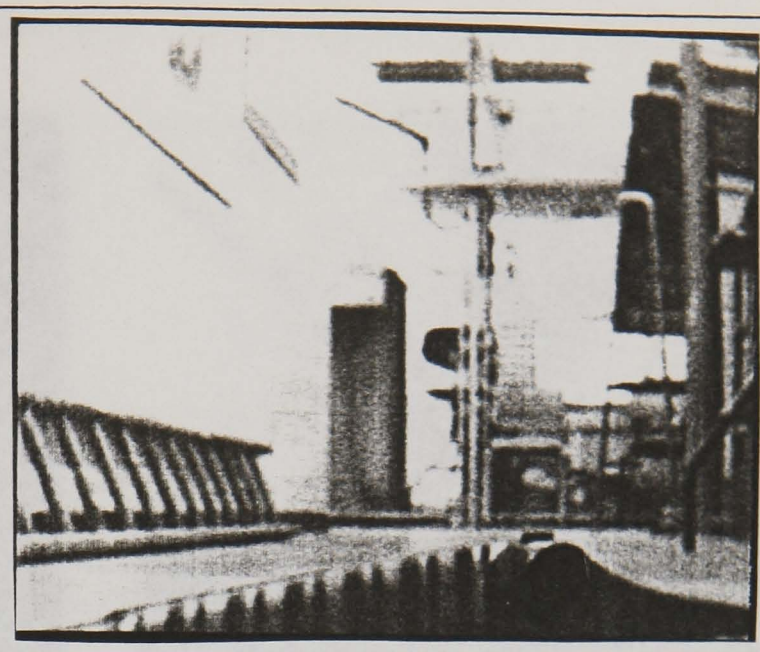


Figure 9.2.3
Video Camera View of a Matt Black Obstacle

It is not feasible here to provide evidence of the system detecting all possible obstructions, neither is there any particular definitive obstacle. However, further to the previous results, the video images of figure 9.2.5 show the camera view of a selection of obstacles as they were detected. These are typical of those likely to be encountered in factories and vary widely in colour, surface texture and shape.

9.3 Response Time and Real-Time Operation

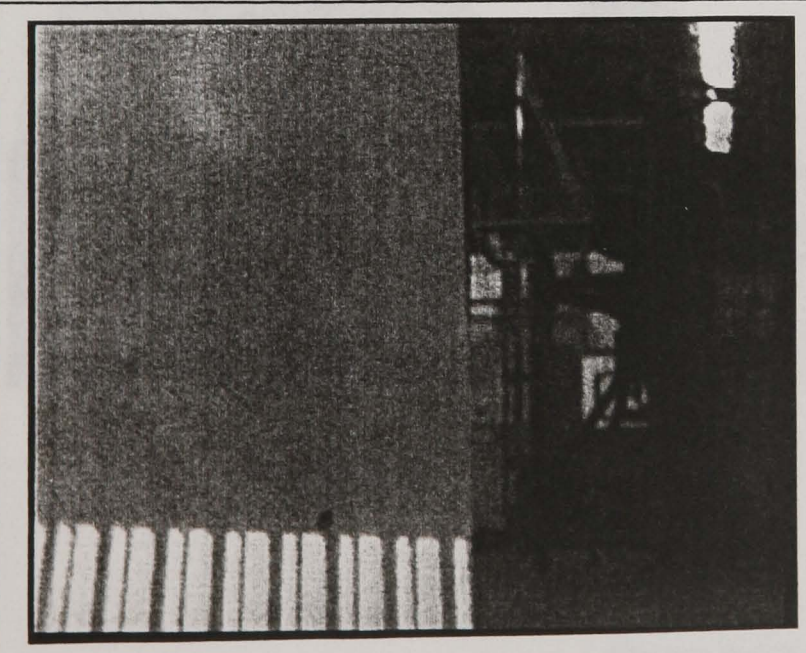
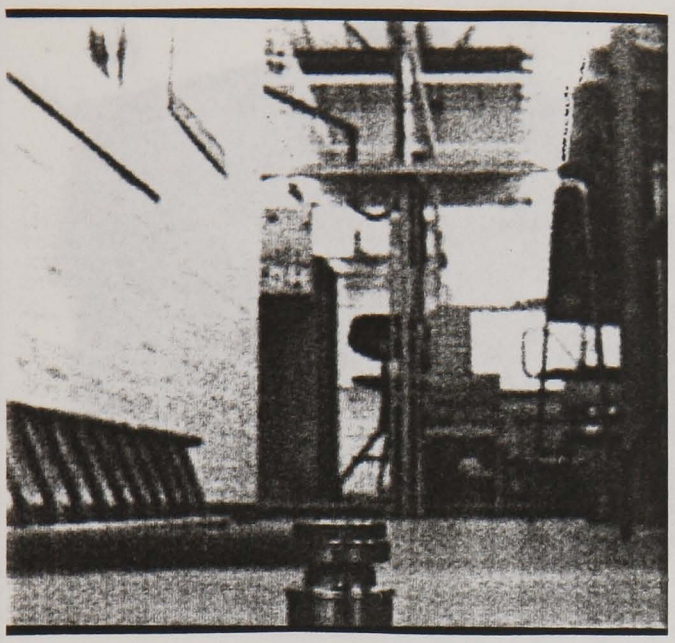
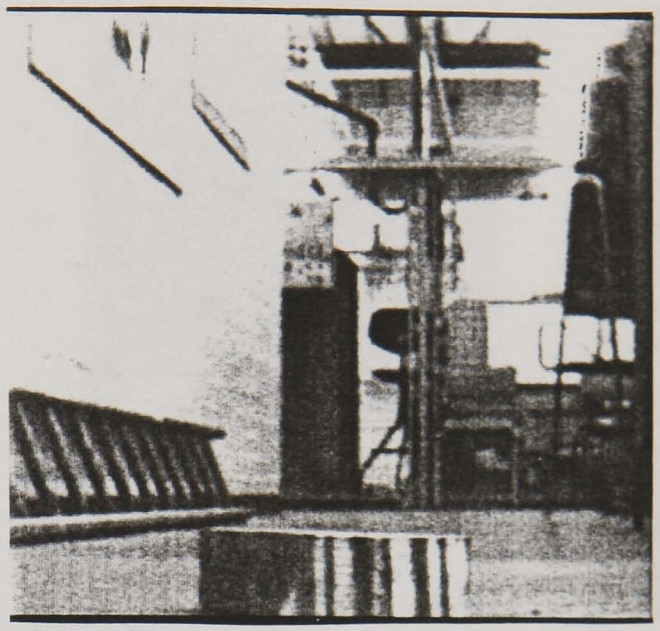


Figure 9.2.4
Video Camera View of a Gloss White Board

When the system initially detects an obstacle, the embedded computer system takes control of the vehicle drives and brings it to a halt. This is achieved by electronically switching buffer integrated circuits and supplying stop commands to the motor controllers. The obstacle detection system then performs



Steel Turned Component



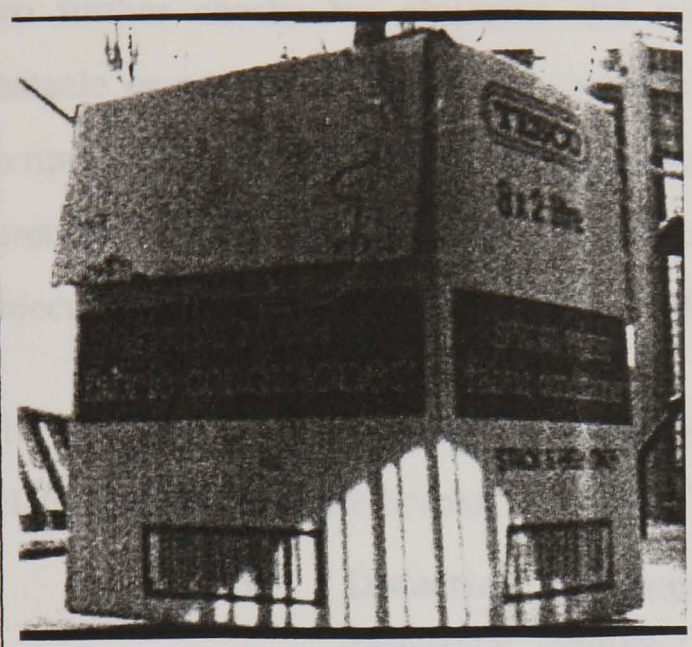
Wood Block

Plastic Sack

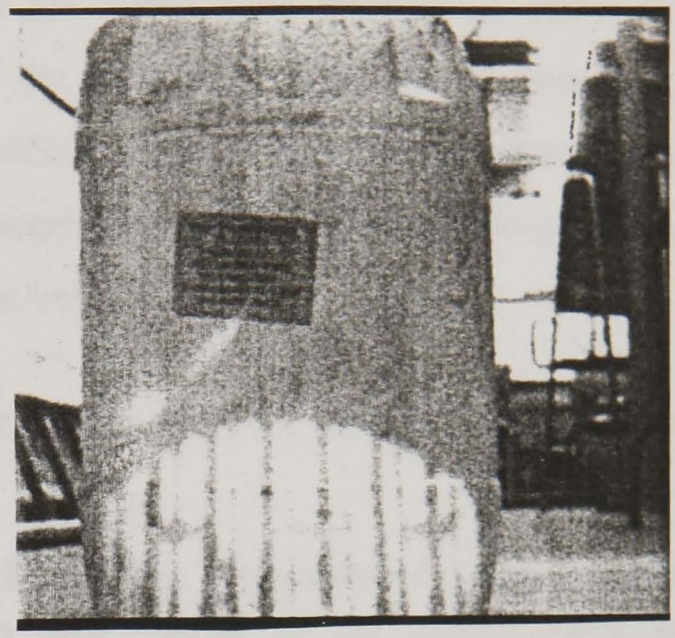
Copper Pipe

Figure 9.2.5

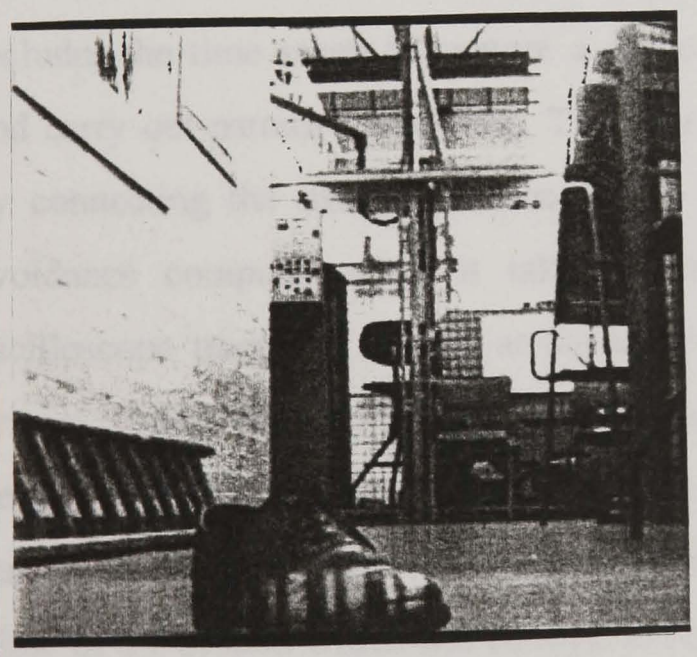
Typical Obstacles Encountered in Factories



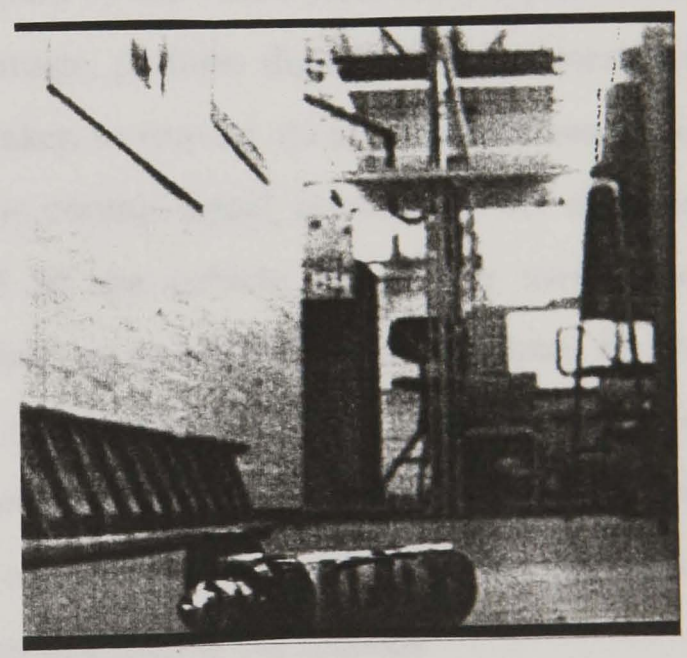
Cardboard Box



Oil Drum



Shoe



Power Tool

Figure 9.2.5

Typical Obstacles Encountered in Factories

two further checks for obstacles and bases the decision of whether to avoid the obstacle or not on the majority consensus of the three results. Section 6.3 provides a comprehensive description of this majority polling scheme. The process increases the surety of obstacle detection and reduces the sensitivity of the system to non-persistent objects (such as people crossing the path of the vehicle).

The response time of the obstacle avoidance system may therefore be considered as relating to two separate events:

- **Detection Response Time**. The time taken for the obstacle avoidance computer to take control of the vehicle after an obstacle is introduced to the system.
- **Avoidance Response Time**. The time interval from an obstacle being placed in front of the system to the instant when the vehicle begins to avoid it.

The time taken to detect an obstacle is governed by the video processing system. This includes the time taken to capture a video image, perform digital filtering operations and carry out pattern recognition. The time taken to execute these tasks was measured by connecting the storage oscilloscope to the control signal asserted by the obstacle avoidance computer when it takes control of the vehicle. In storage mode, the oscilloscope trace was started at the same time as an obstacle was introduced to the system. Figure 9.3.1 is a hard copy of the oscilloscope trace which shows the time interval from the point when the obstacle was introduced (the left hand edge of the trace), to the point where the signal level changes. This is the Detection Response Time of the system which can be seen to be approximately 0.8 seconds.

The Avoidance Response Time as defined above should be approximately three times the Detection Response Time (due to the majority polling scheme) and is therefore calculated as 3 X 0.8 = 2.4 seconds. This figure excludes the relatively negligible time taken to plan the motion of the vehicle.

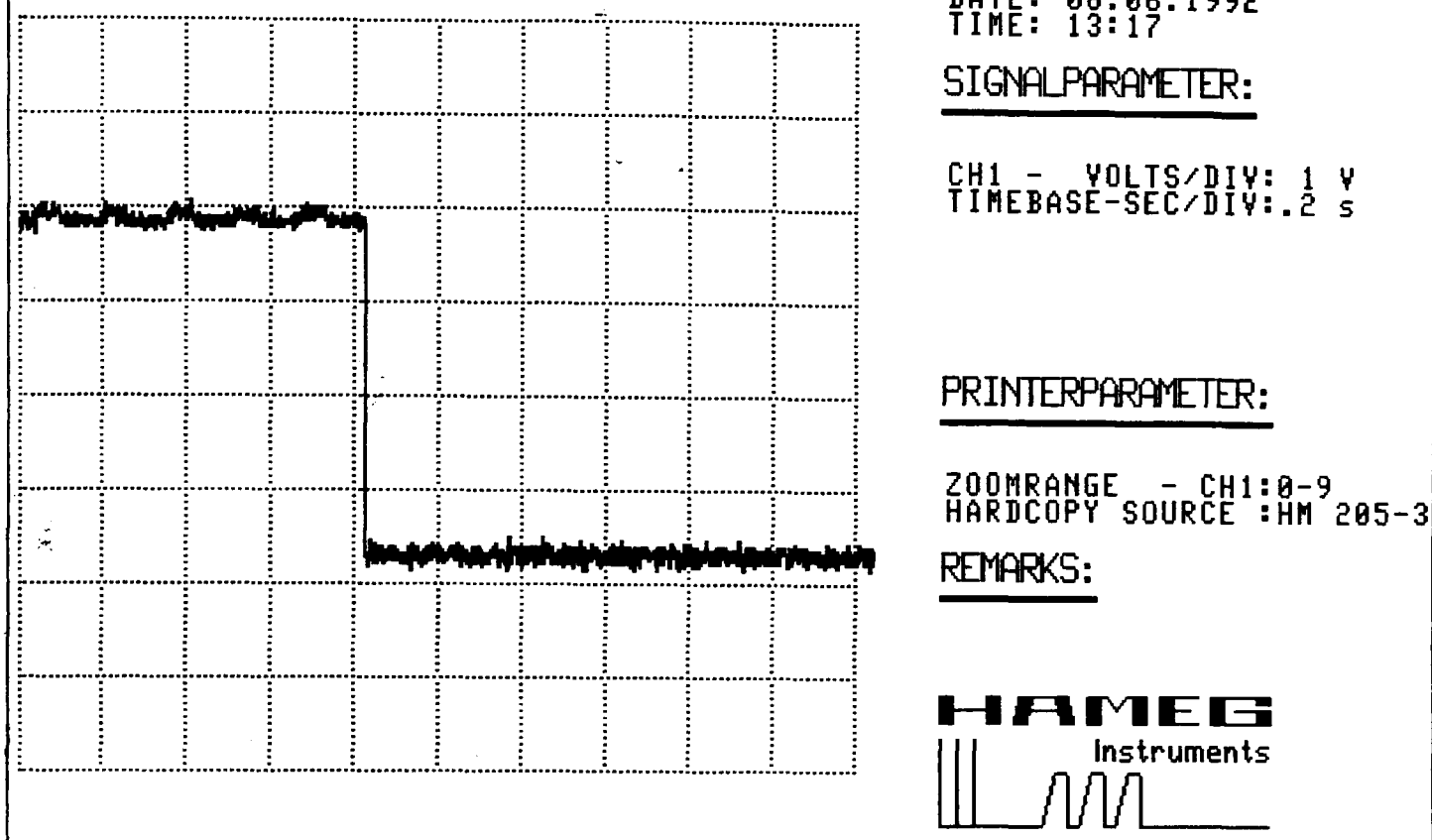


Figure 9.3.1

Oscilloscope Trace Showing Access Control Signal Asserted by Obstacle Avoidance Computer to Take Control of Vehicle

The Avoidance Response Time was measured by connecting the motor drive signals to two channels of a storage oscilloscope. In single sweep storage mode, the oscilloscope trace was triggered approximately simultaneously with an obstacle being introduced into the path of the vehicle. Figure 9.3.2 is a hard copy of the oscilloscope trace showing the motor control signals as the vehicle began to turn. These signals act with opposite polarity since to turn the vehicle, one motor is driven clockwise and the other anti-clockwise. The motor control signals are pulse width modulated, provided by the Hewlett-Packard digital motor controllers via H-Bridge amplifiers as described in chapter 7. The signals shown in figure 9.3.2 are subject to a smoothing effect because

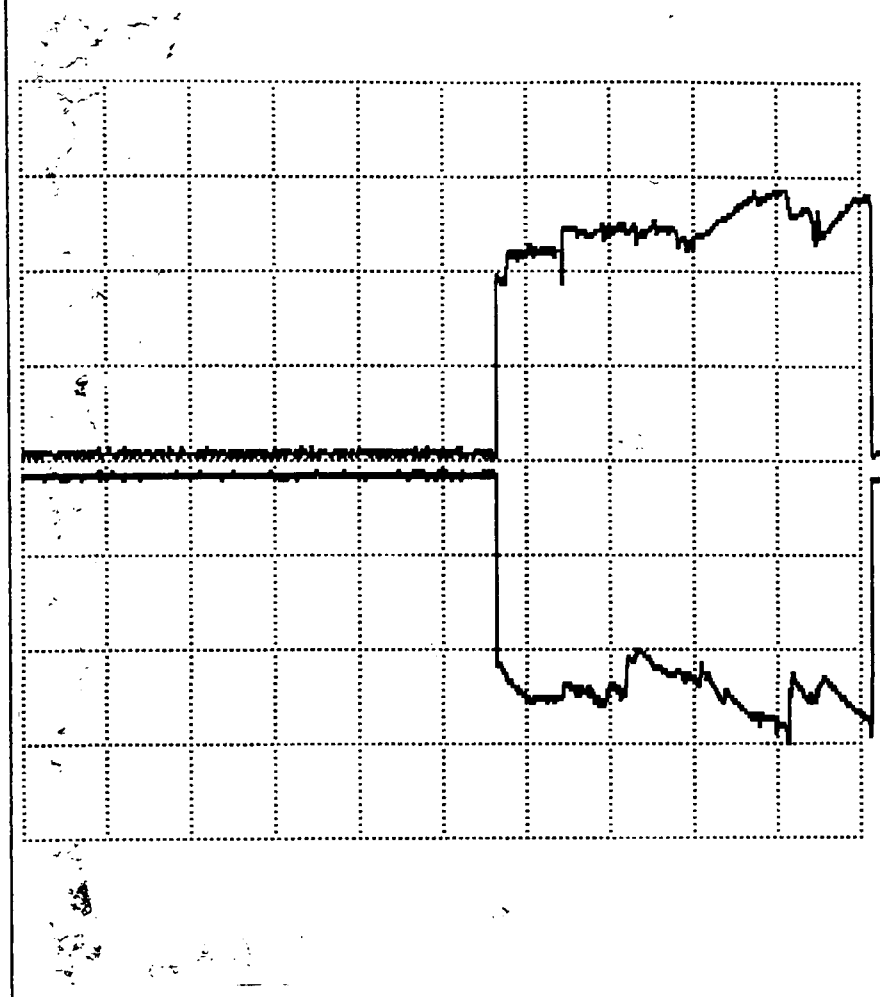


Figure 9.3.2

Motor Control Signals Illustrating Avoidance Response Time

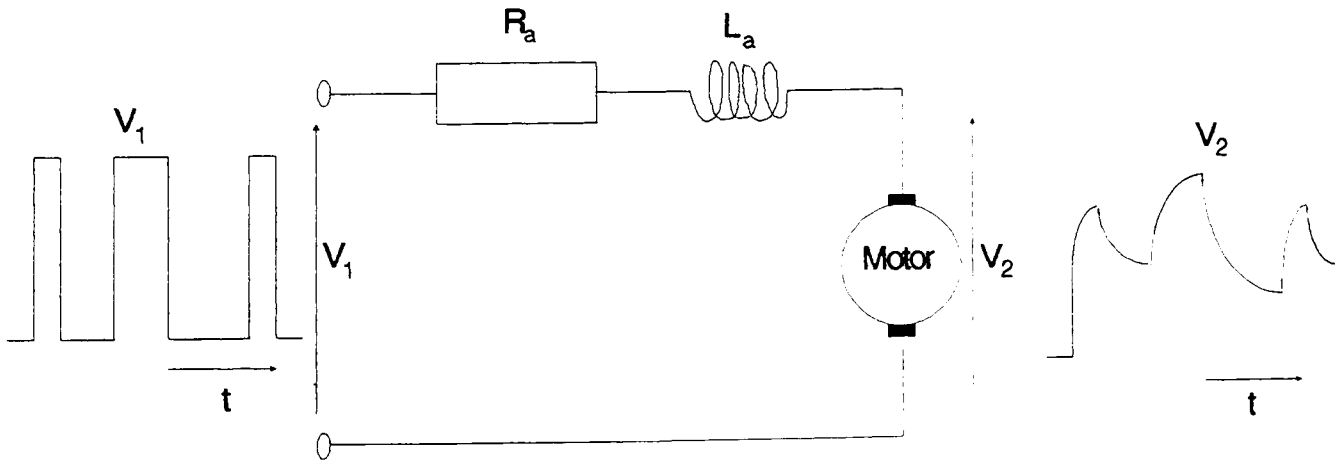


Figure 9.3.3

PWM Signal is Smoothed Due to Motor Inductance

of the large inductive motor loads (see figure 9.3.3). Nevertheless, the Avoidance Response Time of the system can be clearly seen as the interval from the left hand edge of the oscilloscope trace, to the point when the motor control signals are applied. With the time-base set to 0.5 seconds/division, this is approximately 2.48 seconds. The error between this and the predicted time of 2.4 seconds is mostly due to the difficulty of starting the oscilloscope trace at the same instant as the obstacle was introduced.

Factors affecting the motion of the experimental vehicle after the control signals have been applied include the effects of the inertia of the vehicle, the digital control system and the horse power of the motor drives.

Chapter 8 discussed how the motion of the experimental vehicle is reduced to two primary manoeuvres: 5 degree 'TURNs' and 50mm 'ADVANCEs'. In order to determine how long the experimental vehicle takes to turn in 5 degree increments, the motor control signals shown in figure 9.3.2 were subtracted from each other. The result was displayed on a single oscilloscope channel (figure 9.3.4). Since the separate signals are of opposite polarity, this combination highlights the instants when maximum control effort is applied to move the experimental vehicle to a new position. Referring to the overall motor drive block diagram of figure 7.4.2, these are the error signals from the motor controllers which decay as the vehicle moves from its current position to the desired position. With reference to figure 9.3.4, the time between each peak on the oscilloscope trace represents the time taken for the experimental vehicle to complete one 5 degree turn. This time varies slightly due to load disturbances on the system such as dirt under the road wheels and irregularities in the floor surface. Figure 9.3.4 shows that the approximate angular velocity of the vehicle during the 'TURN' phase of obstacle avoidance is approximately 5 degrees every 2 seconds or 2.5 degrees/second.

A similar test was carried out during the 'ADVANCE', phase of avoidance where identical control signals are applied to each motor. The oscilloscope trace of figure

9.3.5 shows the right hand motor signal. As before, the pertinent features of the figure are the peaks where maximum control effort is applied to initiate a move to a new position. The oscilloscope trace shows the period of the peaks to be approximately 2 seconds which corresponds to an average velocity during the 'ADVANCE' phase of avoidance of approximately 0.025m/second.

Real-time is defined as, 'denoting or relating to a data-processing system in which a computer is on-line to a source of data and processes the data as it is generated'. Hence the obstacle avoidance system operates in real-time because it responds to unexpected obstacles as they are encountered. The following experiments demonstrate this feature by testing the response of the vehicle to changing situations and multiple obstacles.

In the first experiment, an obstacle was placed in front of the experimental vehicle so that the system began to avoid it. As the vehicle turned to clear the obstacle, it was repositioned in the vehicle path as illustrated in figure 9.3.6. The obstacle avoidance system responded in real-time to the new situation and was repeatedly 'repelled' by the obstacle as it persisted in the path. The vehicle therefore continued turning in 5 degree increments to avoid the obstacle until the limit set up in software of 90 degrees was reached (see chapter 8). Figure 9.3.7 shows a series of photographs taken as the experiment progressed. The position of the obstacle as it was moved was marked periodically by orange card discs to aid the illustration.

The second experiment to demonstrate the real-time operation of the system was similar to the previous test, except in this case, the obstacle was moved in the opposite direction (away from the path of the vehicle) after it had initially been detected. Figure 9.3.8 shows a photograph taken after this experiment. The orange coloured card discs show the path that the vehicle took when the obstacle was placed fully in front of the vehicle. This original obstacle position is identified in the photograph by orange markers. On the second run, the obstacle was placed fully in front of the vehicle until it was detected and then moved to the new position where it is shown in the

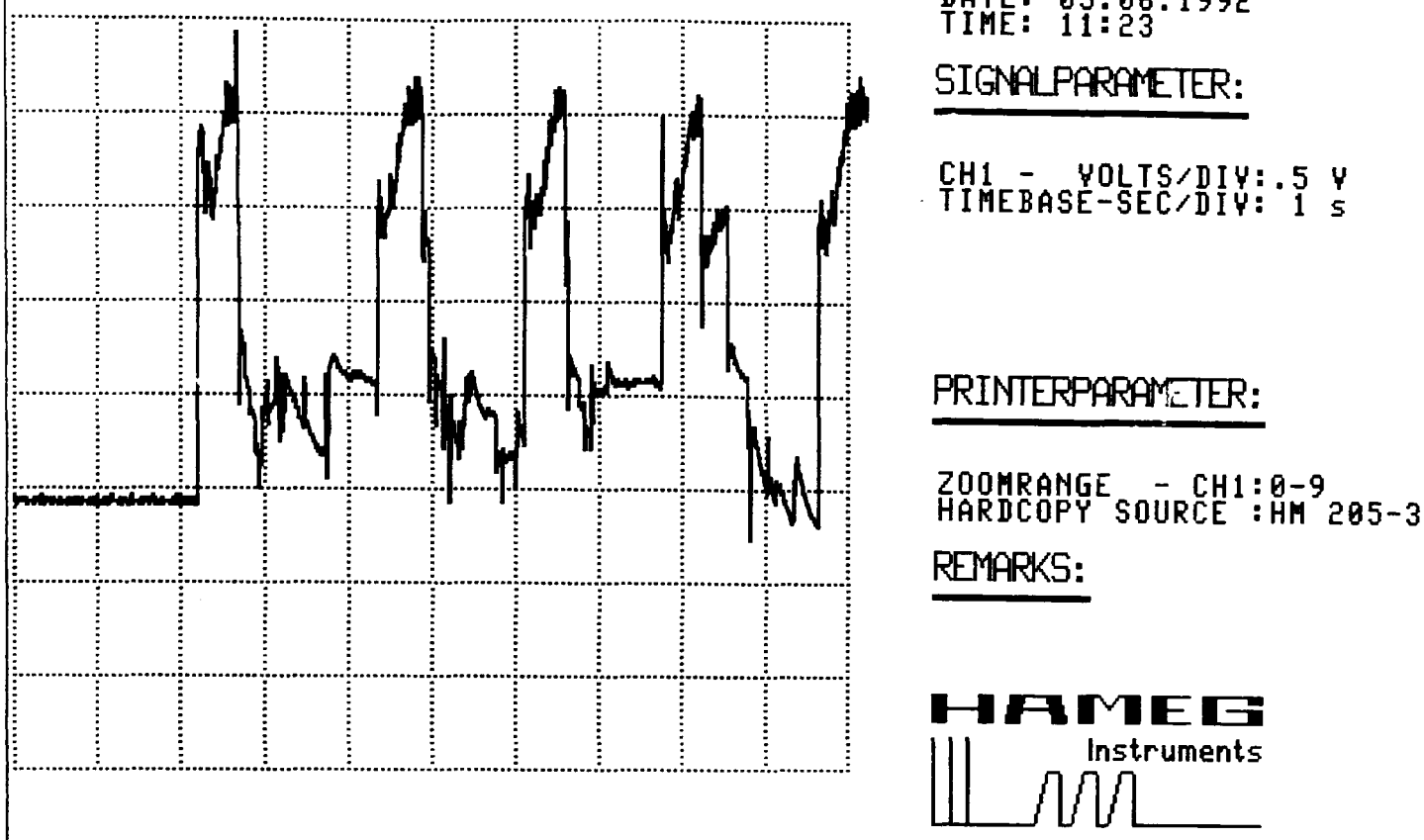


Figure 9.3.4
Subtracted Motor Control Signals During 'TURN' Phase of Obstacle Avoidance
Each Peak Indicates a 5 degree turn

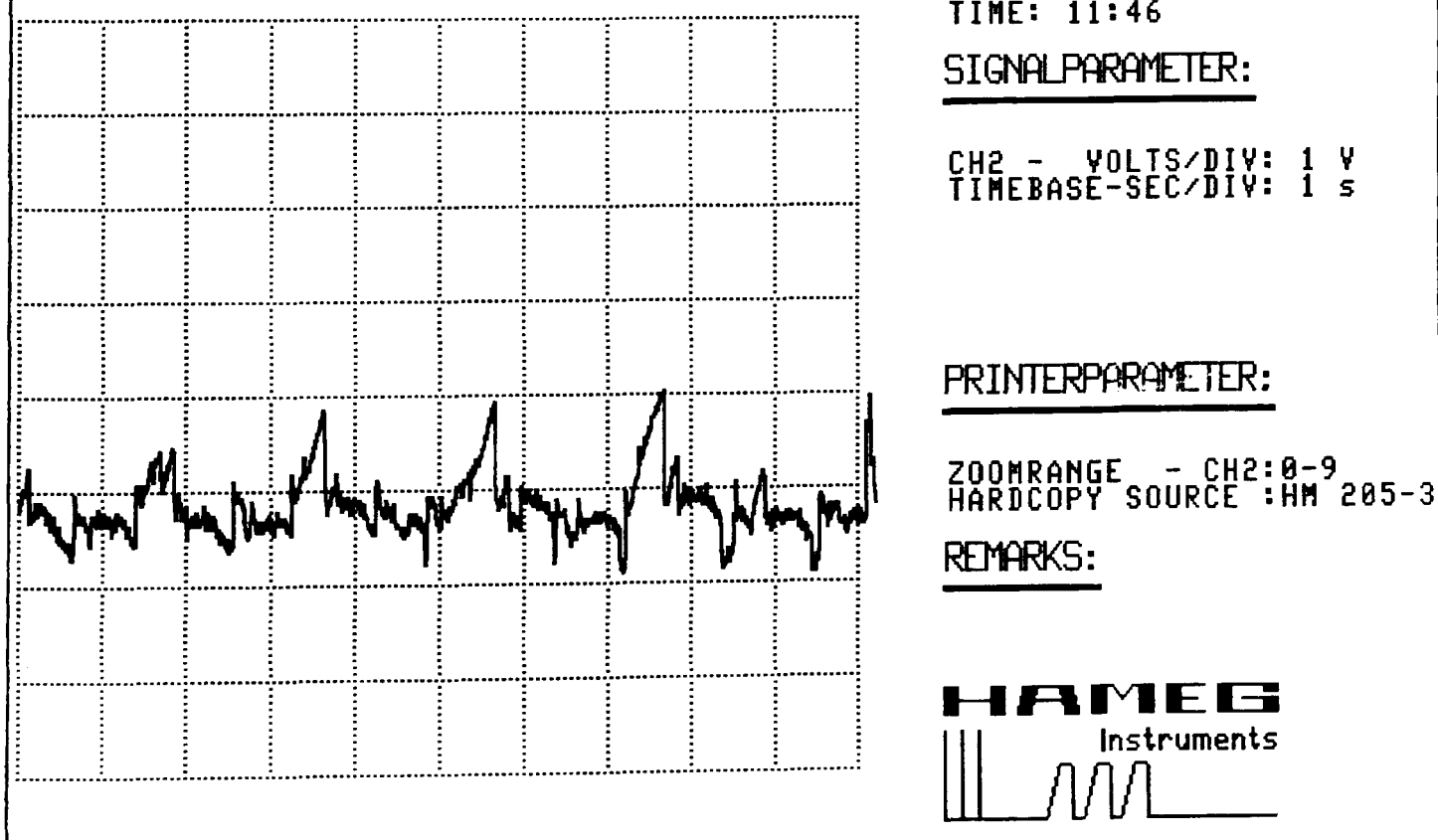


Figure 9.3.5
Right Hand Motor Control Signal During 'ADVANCE' Phase of Obstacle Avoidance
Each Peak Indicates a 50mm Advance

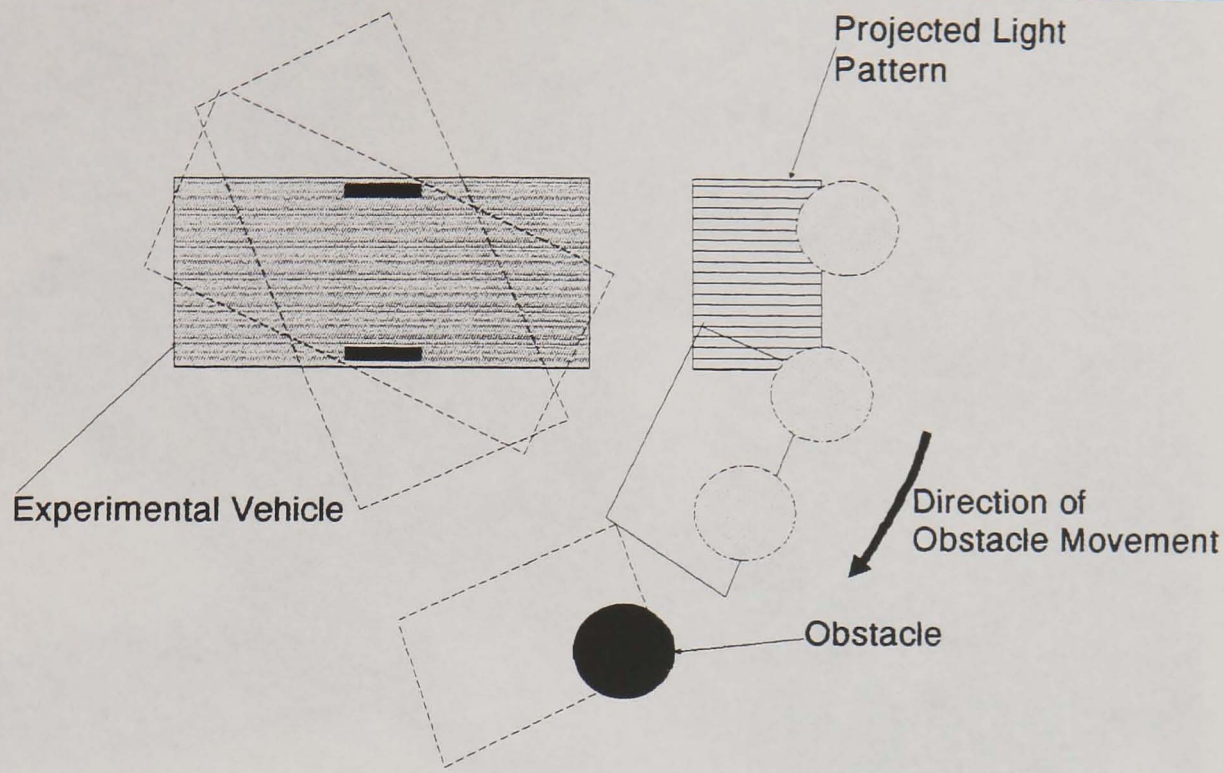


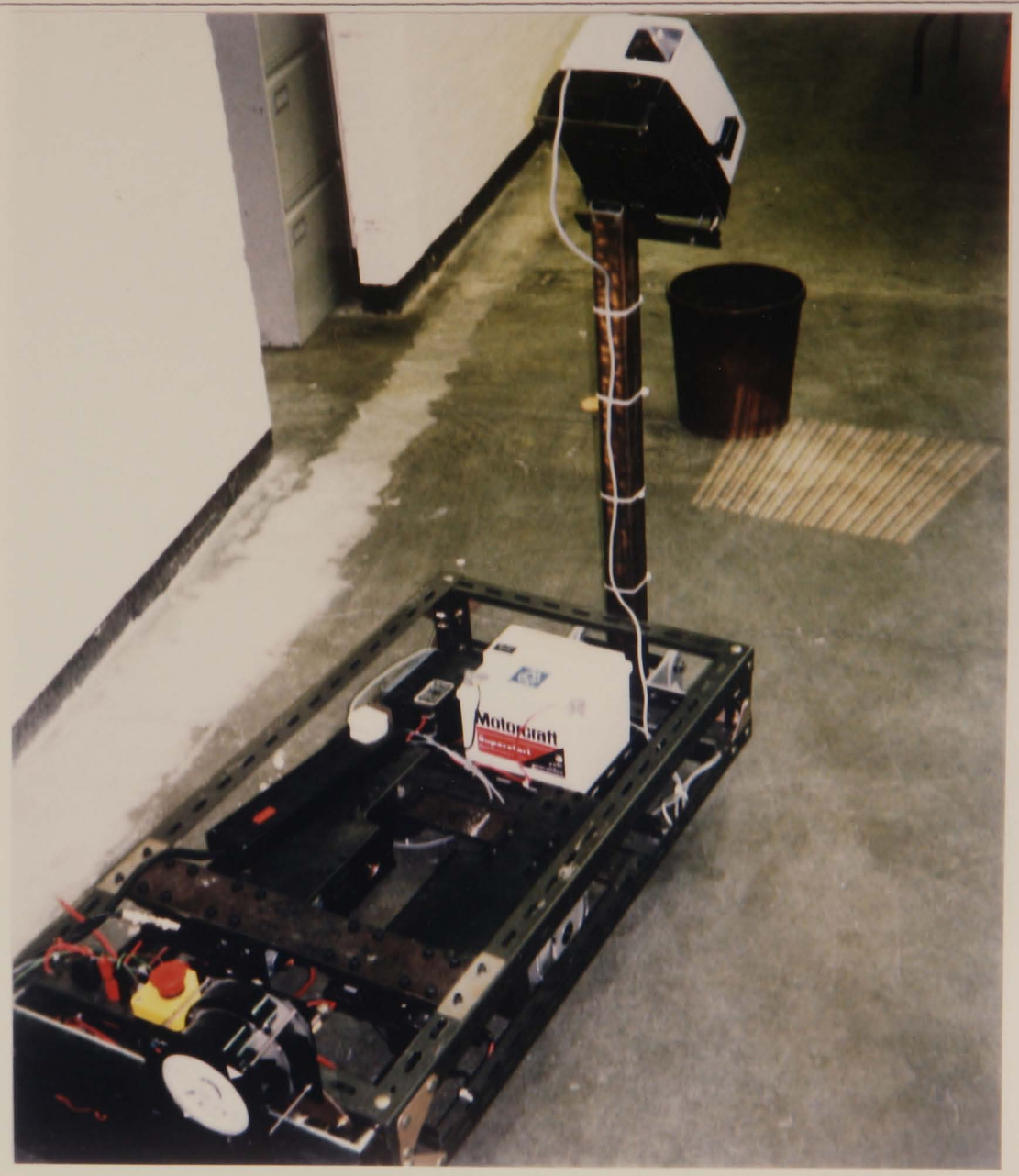
Figure 9.3.6
Moving Obstacle Test

photograph of figure 9.3.8. The white card discs show the corrected path of the vehicle. This result demonstrates that the system responded to the changing obstacle situation by deviating less on the second run.

The final real-time test was the introduction of a second obstruction whilst the system was already engaged in an avoidance manoeuvre. This experiment was conducted by placing a second obstacle ahead and to one side of the first one to be negotiated. The path that the experimental vehicle followed to avoid both obstacles was marked with coloured card discs and the photograph of figure 9.3.9 shows the result of the test. The system was able to negotiate the second obstacle in real-time and therefore successfully avoided both obstructions.

The next tests are concerned with the overall performance of the obstacle avoidance system.

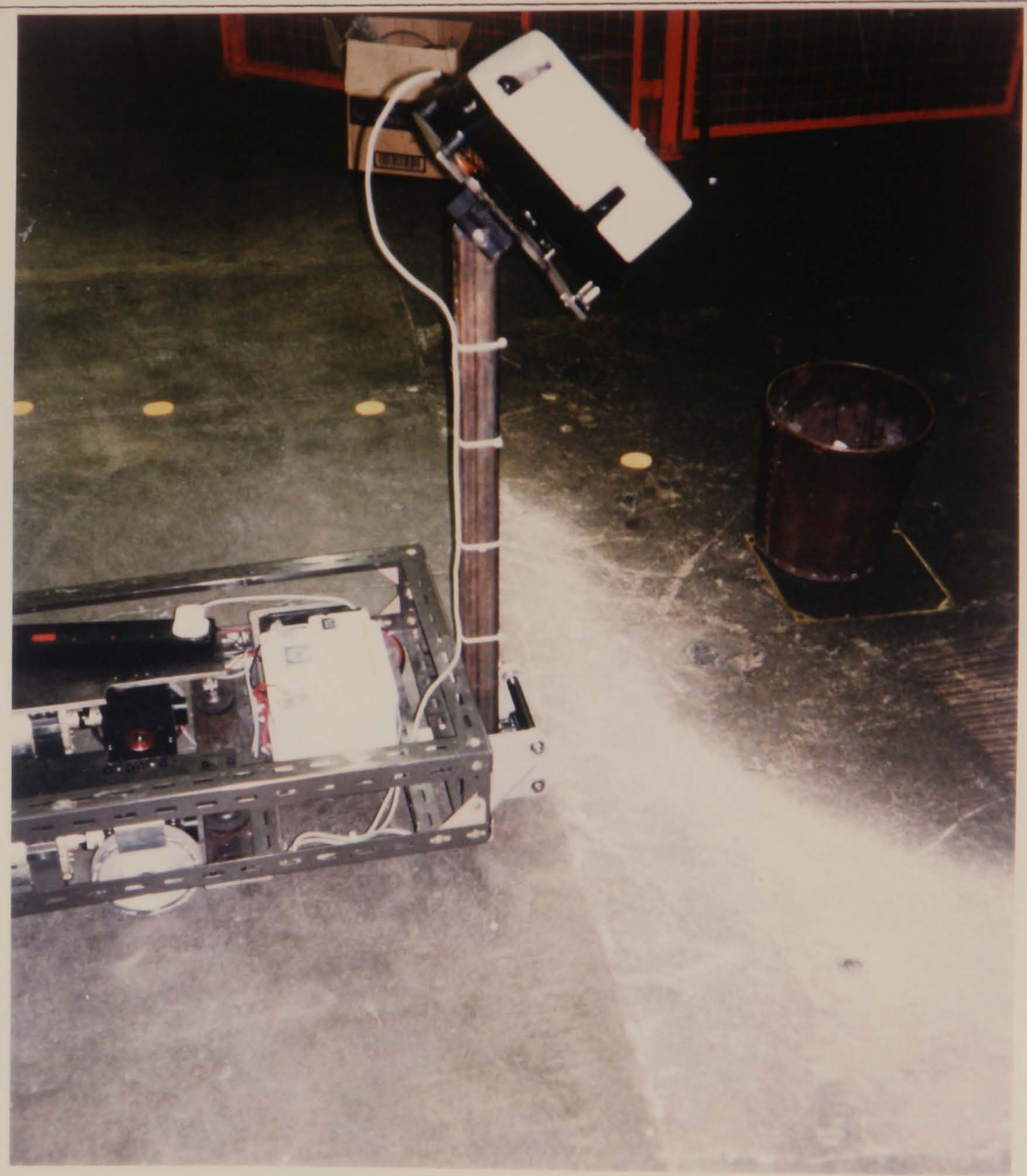
1



2

Figure 9.3.7
Sequence of Photographs Showing Moving Obstacle Real-Time Test

3



4

Figure 9.3.7
Sequence of Photographs Showing Moving Obstacle Real-Time Test

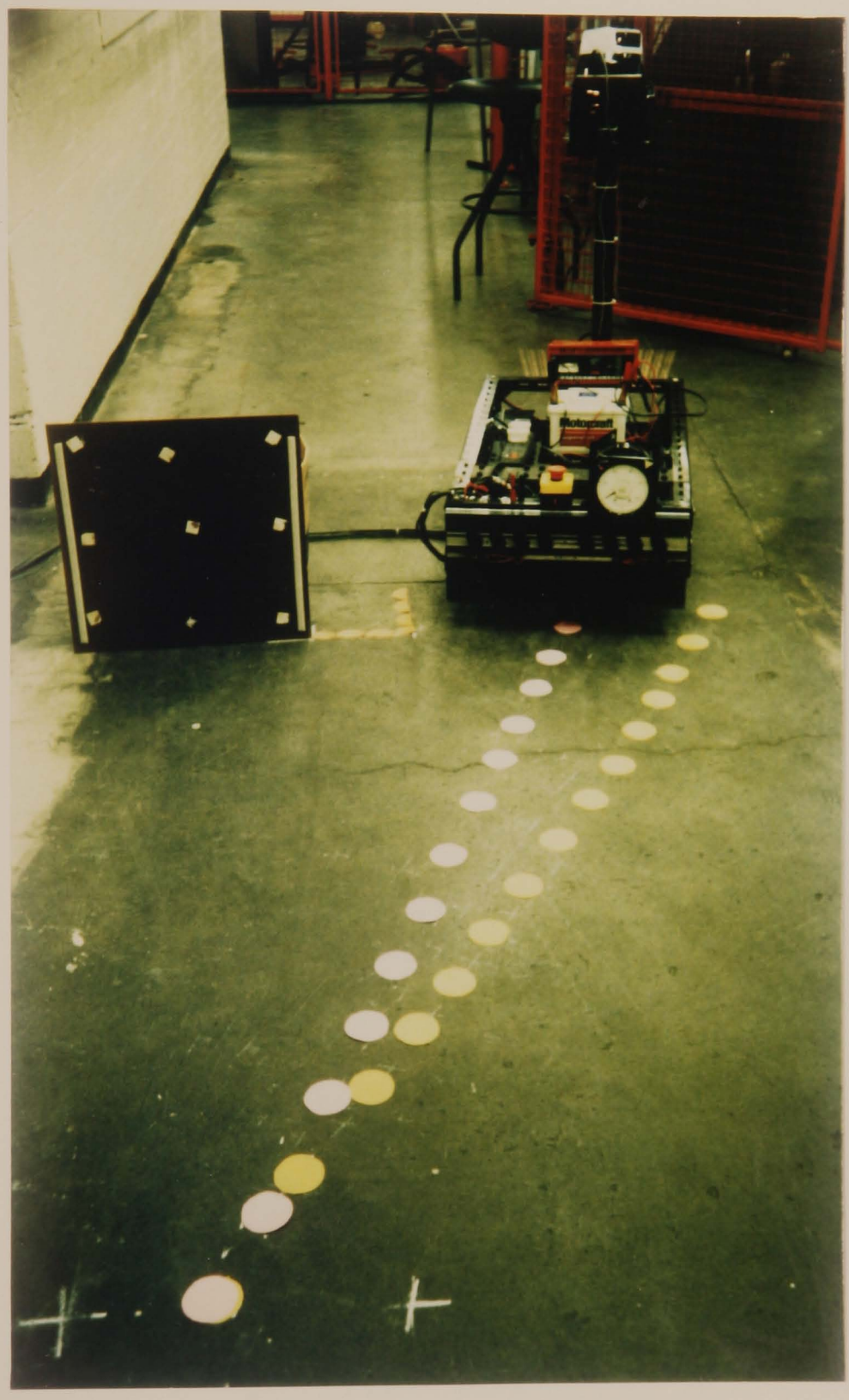


Figure 9.3.8
Photograph Showing Real-Time Path Correction

Figure 9.3.9
Photograph Showing System
Responding to Two Obstacles

9.4 Accuracy and Repeatability

The obstacle avoidance algorithms were tested in terms of their accuracy and repeatability. In this work, accuracy is defined as the error within which the experimental vehicle will return to the original guide path after avoiding an obstacle as shown in figure 9.4.1. Repeatability is defined as the ability of the system to avoid an obstacle using the same route given equivalent starting conditions.

For both experiments, the guide path of the experimental vehicle was defined parallel to the laboratory wall as shown in figure 9.4.2. The deviation of the vehicle was then referred to this datum.

In order to test the accuracy of the obstacle avoidance system, the vehicle was positioned 0.4 metres away from and parallel to the reference wall. An obstacle was placed in front of the vehicle and the system was allowed to avoid it. After the avoidance manoeuvre was complete, the deviation from the wall was measured. The experiment was repeated several times and the final position measurements were within the range 0.38 - 0.42 metres showing a deviation error of approximately +-0.02 metres. Figure 9.4.3 shows photographs taken during the experiment. The path of the vehicle was marked using coloured card discs to aid the illustration. Similar tests were performed with various obstacles causing the vehicle to take different routes. Figure 9.4.4 shows the results of a test where an obstacle was placed in front of the vehicle

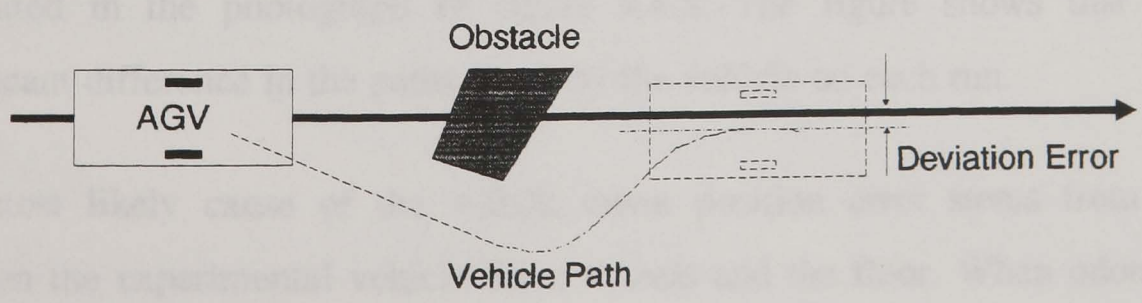


Figure 9.4.1
Accuracy of Obstacle Avoidance System

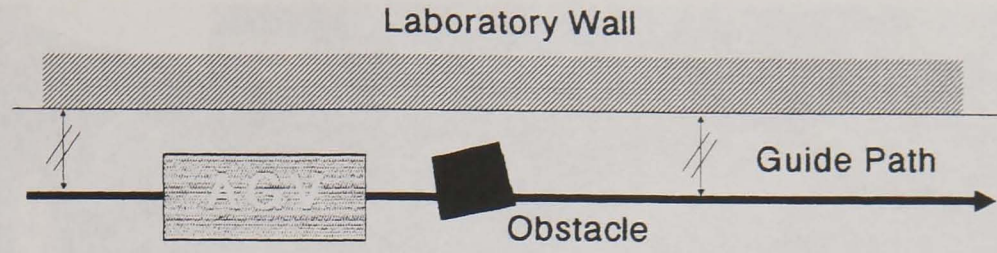


Figure 9.4.2
Vehicle Path Defined Parallel to Laboratory Wall

and the route taken around the obstacle was marked with yellow card discs. The obstacle was moved further into the path of the vehicle and the experiment repeated. In this second case, the route that the vehicle used was marked with orange discs. The photograph of figure 9.4.4 shows that whilst a different route was taken for each situation, the vehicle returned to the original guide path.

The repeatability of the obstacle avoidance system was tested by arranging for the experimental vehicle to avoid the same obstacle repeatedly from the same starting position.

The initial position of the experimental vehicle drive wheels were accurately marked on the laboratory floor and an obstacle introduced in front of the system. The route taken to avoid the obstacle was marked with yellow card discs. The vehicle was then returned to the previous initial position and the experiment repeated. On the second run, the vehicle route was marked with orange discs. The result of this experiment is illustrated in the photograph of figure 9.4.5. The figure shows that there is no significant difference in the paths taken by the vehicle on each run.

The most likely cause of the +-0.02 metre position error stems from the contact between the experimental vehicle drive wheels and the floor. When odometry is used as a means of position sensing in differentially steered vehicles, both drive wheels must have the same diameter. Furthermore, both wheels must maintain positive contact

Figure 9.4.3
Photographs Showing Accuracy Test

with the ground since any slippage will also result in errors.

The experimental vehicle wheels are accurately machined to the same dimensions and fitted with hard rubber tyres. Whilst the latter ensure good adhesion to the ground, they also necessarily compress under the weight of the vehicle. If this compression does not occur equally in both wheels, the relative diameters will be altered. A similar effect will occur if one wheel picks up debris from the floor which again, effectively changes the relative wheel diameters. In general, position errors occurring due to these factors are proportional to the distance travelled by the vehicle.

A further type of error which can occur in dead-reckoning systems is caused by uneven floor surfaces. If one drive wheel travels on a flat surface and the other over undulations, the result will be that the former wheel appears to travel a shorter distance than the latter and steering error will occur.

Section 2.2.2 discussed how many researchers attempt to combine odometry with another absolute position sensing method in order to periodically correct the accumulating position errors. A key advantage of this system is that the position of the automated vehicle is referred to the point where it leaves the guide path. Since the distance that the vehicle travels from this point is relatively short, excessive errors do not occur.

The distance from which commercial AGVs can sense their guide path depends on the design of the sensing head. In a commercial active wire-guided system installed by the collaborating establishment in Doncaster, England, the automated vehicles can detect the embedded guide wire from a distance in excess of 10cm. The position error in the obstacle avoidance system is well within this range and therefore odometry can be successfully used as the sole means of navigation.

The conclusions drawn from the results presented in the previous sections and

throughout the thesis are discussed in the next chapter. The limitations of the system are explored and suggestions made for further work on the obstacle avoidance system.

Figure 9.4.4
Photograph Showing Accuracy Test

Figure 9.4.5
Photograph Showing Repeatability Test

10 CONCLUSIONS LIMITATIONS AND RECOMMENDATIONS FOR FURTHER WORK

10.1 Conclusions

The guidance methods available for automated vehicles and the current state of obstacle avoidance research were reviewed in chapter 2. The key findings of the review were that:

- Most industrially based AGV systems use the inductive embedded wire method of guidance because of its ruggedness and proven reliability.
- At present, there are no commercially available AGV systems that can avoid unexpected obstacles.
- Although much research has been carried out in the field of obstacle avoidance, none has yet produced a system suitable for use in manufacturing factory environments.

The research presented in this thesis has therefore aimed to design a stand-alone obstacle avoidance system that could eventually be retrofitted to existing automated vehicles. The performance of commonly used wire guided vehicles would then be enhanced at a small cost in relation to that of the entire installation. A retrofitting system would also incur low installation costs since the overall factory need not be disrupted whilst obstacle avoidance systems were fitted to vehicles. Furthermore, as the systems would be independent, any isolated failures would have little effect on the manufacturing process as a whole.

Various sensor systems were considered for use in the design. These included ultrasonics, laser and other techniques. Ultrasonic methods are unsuitable because of

their susceptibility to extraneous high-frequency noise produced by some manufacturing processes. Also, sensors using moving parts such as delicate stepper motors or rotating mirrors etc., were considered unsuitable because of the risk of damage and the need for routine maintenance. Modern optical technology however, has produced extremely low-cost, robust and compact sensors in the form of CCD arrays. These use no moving parts and are maintenance free. A monochrome CCD video camera has therefore been employed as the sensing element of the new system.

Some obstacle avoidance systems use CCD cameras to passively detect obstacles. A disadvantage of these, is that complex scene analysis must be carried out to discriminate between sections of the CCD image that represent obstacles to be avoided, and those that represent incidental features or illusions. An active system based on a novel light pattern projection system has been successfully developed to overcome this disadvantage. The task of scene analysis has been simplified by introducing coded light information into the system which enables pertinent sections of the image to be clearly identified.

A study of the applications of structured light in computer aided engineering and design revealed that the systems used in this field are static and often take several seconds or in some cases minutes to process large amounts of video data. This research has successfully extended the use of structured light to a mobile system and has simplified the task of image processing by using a novel projected light, 'bar' coding scheme. When obstructions emerge into the path of the automated vehicle, they reflect the projected light bar pattern which is then detected by the CCD camera.

A major design objective set out in chapter 1 was that the system should be low-cost, compact and robust. This has been successfully achieved using a single Intel 8031 microcontroller to carry out image processing and obstacle avoidance tasks. The Intel 8031 computer was chosen for its low-cost and range of on-chip features. These include on-board timers, internal and external interrupts, bidirectional parallel and

serial communications ports.

Ancillary memory buffer and digital motor control circuits have been designed around this computer and the software developed entirely in assembler language. This has enabled optimum execution speed and efficiency to be achieved.

The methods used to process video information and hence detect obstacles comprise a combination of digital filters and direct methods. High processing speed has been achieved by using minimum word-length integer arithmetic and avoiding floating-point methods. The latter has placed constraints on the design of a suitable recursive digital filter since the coefficients usually require an accuracy in the order of three decimal places. However, a comparison of filters designed using the bilinear transform method and simple integer-only digital equivalents of a C-R filter has proved that a digitised C-R filter gives adequate and reliable performance. Direct methods have been successfully implemented to detect features in the filtered video information. A decision-theoretic method of pattern recognition analyses the relationships between these features which in turn, belies the presence of obstacles.

A crucial design aim was that the obstacle avoidance system should be simple to configure and maintain. An algorithm which automatically calibrates the projected light detection system successfully achieves this by alleviating the need for critical projector-camera positioning.

An experimental automatic vehicle has been designed and constructed to test the obstacle avoidance system. This is based on a differential steering arrangement which uses two drive wheels controlled by dedicated single-chip microcontrollers. The range of possible control actions has been investigated and suitable digital control parameters have been determined empirically.

The complex obstacle avoidance control actions are broken down into basic

manoeuvres upon which higher levels of control are based. The control algorithms have been designed with the aid of a specially developed computer simulation. This uses high resolution colour graphics and allows various obstacle situations to be simulated. The operation of the model has been described and the results obtained during the design of the control algorithms presented in chapter 8. The resulting obstacle avoidance computer program was converted from Pascal, to Intel MCS-51 assembler language for implementing on the embedded system.

Tests have been successfully carried out on the obstacle avoidance system, the results of which are presented in chapter 9. These show that the new system operates in real-time and has the ability to negotiate multiple obstacles. The system was found to have a position error of approximately +/- 0.02 metres. This was considered acceptable since the vehicles in an AMECAS commercial system can locate the embedded wire guide-path from a distance in excess of 0.1 metres. Tests to establish the repeatability of the system showed that there was no significant variation in the paths taken by the experimental vehicle on repeat runs under similar starting and obstacle conditions.

The overall cost benefits of the obstacle avoidance system to potential users are difficult to estimate as they depend on factory-specific factors such as the volume of traffic, the number of AGVs, and the size and complexity of the factory layout. However, it is estimated that each system would cost approximately 3 - 5% of the total value of the vehicle on which it would be installed (based on the approximate new cost of the AMECAS AGVs in the Doncaster installation).

10.2 Limitations of the System and Recommendations for Further Work

The pattern used for the projected light codes has provided reliable operation throughout the research. Nevertheless, it may be possible to design more efficient light

patterns for detecting smaller obstacles. This research has been primarily concerned with the design of a practical system embodying aspects of the mobile application of structured light and obstacle avoidance. The subject of developing optimum codes could be addressed in further research work.

At present, the new obstacle avoidance system avoids obstacles on straight guide-paths. The performance of the system would be enhanced by the ability to avoid obstacles on curves. However, extra sensing equipment may be required on both the vehicle and the factory floor so that the obstacle avoidance system could determine where curves started and ended, and in which direction they turned. Also, since AGV systems may have curves of varying dimensions and shape, some standardisation of existing layouts may be required. The problem of achieving obstacle avoidance on curves whilst maintaining the generality of the system could be addressed as the subject of a further research project.

Although the system operates in real-time, the obstacle detection response time of approximately 0.8 seconds is relatively modest. However, since the start of this research project, developments in electronics have resulted in the introduction of Intel MCS-51 series microcontrollers capable of operating with master clock frequencies over 30 megahertz (almost three times the speed of the present microcontroller). The detection response time of the obstacle avoidance system could be drastically reduced by using such a microcontroller. Time constraints have not allowed this modification to be carried out however, as the ancillary circuits would have to be redesigned in order to accommodate the increased processor operating speed.

Several printed circuit boards are used for the electronics in the prototype obstacle avoidance system. The circuits could be customised by combining elements in integrated units. This development would further reduce the cost of the system and improve its reliability.

The obstacle detection system is not limited to application in this research project. The novel light coding scheme has the inherent feature that the approximate width of objects can be determined by virtue of the number of light codes reflected. Furthermore, the system could be modified to provide object height information. With these developments, the obstacle detection system could find use in object classification or recognition systems.

The obstacle avoidance computer model proved extremely helpful in the design of the obstacle avoidance algorithms and although it was developed specifically for this project, it would also be suitable as a development tool in other AGV research projects and also as a teaching aid. Similarly, whilst the experimental vehicle was built specifically as a test bed for the obstacle avoidance system, it could also be useful for continuing research and demonstration.

REFERENCES

1 Hammond, G., AGVs at Work - Automated Guided Vehicle Systems, IFS Publications Ltd., 1986, Chapter 1.

2 Mechanical Handling Economic Development Committee, Advanced Handling Systems - Exploiting the Opportunities, 1986, PP 3-7.

3 Mortimer, J. (Editor), Ingersoll Engineers, The FMS Report, IFS Publications, 1982, Chapters 1 and 2.

4 Clauzier, I. F., Gibbons, D. T., Improvements on the Guidance Systems of Automatically Guided Vehicles, IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 1987, PP145-148.

5 Hollingham, J., Wagner AGVs Show Strength and Endurance, Industrial Robot, Volume 18 Number 12, 1991, PP 12-14.

6 Premi, S. K., Besant, C. B., A Review of Guidance Techniques That Can Be Used by Mobile Robots or AGVs, Proceedings of the 2nd International Conference on AGV Systems, 1983, PP 195-209.

7 Boegli, P., Comparative Evaluation of AGV Navigation Techniques, Proceedings of the 3rd International Conference on AGV Systems, 1985, PP 169-180.

8 Tsukagoshi, T., Miura, T., Yamauchi, F., Magnetic Lattice Lane Guide for AGVs, Proceedings of the 8th International Conference on AGV Systems, 1987, PP 137-144.

9 Kato, A., Suzuki, T., Hosoi, M., Spotmark Guided Unmanned Vehicle System, Proceedings of the Japan - USA Symposium on Flexible Automation, 1986, PP 361-364.

10 Takeda, Kato, Suzuki, Hosoi, Automated Guided Vehicle Guidance Using Spotmark, IEEE International Conference on Robots and Automation, 1986, Volume 2, PP 1349-1353.

11 Takeda T., Kato, A., Cleanroom Inspection Robot, Proceedings of the 5th International Conference on AGV Systems, 1987, PP 277-288.

12 Robins, M. P., Free Ranging Automatic Guided-Vehicle System, GEC Review, Volume 2 Number 2, 1986, PP 129-132.

13 Culley, G., Baldur, R., Free Wheel Approach to AGV Navigation, Proceedings of the ASME Conference - Computers in Engineering, 1988, PP 267- 274.

14 Julliere, M., Marce, L., Perrichot, H., A Guidance System for a Vehicle Which Has to Follow a Memorized Path, Nouvel Autom, Volume 28 Number 36, 1983, PP 52-60.

15 Kochan, A., British AGV is on the Right Course, The FMS MAgazine, July 1984, PP 136-137.

16 Walker, S. P., Premi, S.K, Besant, C. B., Imperial College Free-Ranging AGV (ICAGV) and Scheduling System, Proceedings of the 3rd International Conference on AGV Systems, 1985, PP 189-198.

17 Stephens, P., Robins, M., Roberts, M., Truck Location Using Retroreflective Strips and Triangulation with Laser Equipment, Proceedings of the 2nd European Conference on Automated Manufacturing, 1983, PP 271-282.

18 Rathbone, R., Valley, R. A., Kindlemann, P. J., Beacon Referenced Dead Reckoning: A Versatile Guidance System, Robotics Engineering, December 1986, PP 11-16.

19 Pears, N. E., Bumby, J. R., Guidance of an Autonomous Guided Vehicle Using Low-Level Ultrasonic and Odometry Sensor Systems, Transactions of the Institute of Measurement and Control, Volume 11 Number 5, 1989, PP 231-249.

20 Eaton-Kenway Inc. USA, Inertial Guidance: Is it a Viable Guidance Syatem for AGVs, Proceedings of the 4th International Conference on AGV Systems: AGVS-4, 1986, PP 301-320.

21 Tsumura, T., Recent Development of Automated Guided Vehicles in Japan, Robotersysteme, 1986, PP 91-97.

22 Hamel, W. R., Babcock, S. M., Hall, M. G., et al, Autonomous Robots for Hazardous and Unstructured Environments, Oakridge National Laboratories, 1985, PP 5.9-5.27.

23 Moravec, H. P., Elfes, A., High Resolution Maps from Wide Angle Sonar, IEEE International Conference on Robotics and Automation, 1985, PP 116-120.

24 Weisbin, C. R., et al, Hermies-II: A Mobile Robot for Navigation and Manipulation Experiments. Robots-9 Conference Proceedings, 1985, PP 1-24.

25 Normoyle, P. D., Huissoon, J. P., Ultrasonic Vision System for Use on Mobile Robots and Automated Guided Vehicles, UK Research in Advanced Manufacture, 1986, PP 55-60.

26 Hall, E. L., Oh, S. J., Katten, E. U., Experience With a Robot Lawn Mower, Robots-10 Conference Proceedings, 1986, PP 4.1-4.26.

27 Farsaie, A., McKnight, T. R., Ferren, B., Harrison, C. F., Intelligent Controllers for an Autonomous System, Proceedings of the IEEE International Symposium on Intelligent Control, 1987, PP154-158.

28 Corfield, S. J., Frazer, R. J. C., Harris, C. J., Architectures for Real-Time Intelligent Control of Autonomous Vehicles, Computing and Control Engineering Journal, November 1991.

29 McTamaney, L. S., Mobile Robots: Real-Time Intelligent Control, IEEE Expert, 1988, Volume 2 Part 4, PP 55-68.

30 Parodi, A. M., Nitao, J. N., McTamaney, L. S., Intelligent System for an Autonomous Vehicle, Proceedings of the IEEE International Conference on Robotics and Automation, 1986, PP 1657-1663.

31 Hollingam, J., Caterpillar Make the Earth Move: Automatically, Industrial Robot, 1991, Volume 18 Number 2.

32 Probert, P. J., Stamper, R., Designing a Controller That Works: Using Formal Techniques in Robotic Systems, Computing and Control Engineering Journal, November 1991.

33 Brooks, R., A Robust Layered Control System for a Mobile Robot, IEEE Journal of Robotics and Automation, March 1985, PP 31.

34 Ichikawa, Y., Kamimura, H., Autonomous Vehicle, Proceedings of the 3rd International Conference on AGV Systems, 1985, PP 199-208.

35 Murata, M., Yamashita, T., Udagawa, S., Tabata, H., Ultrasonic Guided Vehicle, Proceedings of the 5th International Conference on AGV Systems, 1987, PP 145-156.

36 Hock, C., Landmark Navigation with ATHENE, Proceedings of the 5th International Conference on Advanced Robotics - Robots in Unstructured Environments, 1991, PP 1099-1104.

37 Segovia, A., Rombaut, M., Preciado, A., Meizel, D., Comparative Study of the Different Methods of Path Generation for a Mobile Robot in a Free Environment, Proceedings of the 5th International Conference on Advanced Robotics - Robots in Unstructured Environments, PP 1667-1670.

38 Perrier, M., Zapata, R., Mobile Robot Navigation in Ill-Structured Worlds by Means of Prediction Functions, Proceedings of the 5th International Conference on Advanced Robotics - Robots in Unstructured Environments, PP 1675-1678.

39 Garibotto, G., Masciangelo, S., Path Planning Using the Potential Field Approach for Navigation, Proceedings of the 5th International Conference on Advanced Robotics - Robots in Unstructured Environments, PP 1679-1682.

40 Evans, J., Krishnamurthy, B., Pong, W., King, S., et al, Help mate: A Service Robot for Health Care, Industrial Robot, June 1989.

41 Borenstein, J., Koren, Y., Obstacle Avoidance with Ultrasonic Sensors, IEEE Journal of Robotics and Automation, 1988, Volume 4 Part 2, PP 213-218.

42 Xuan, G., Wang, L., Li, C., Liu, Y., Wang, J., Intelligent Searching Algorithm for Robot Obstacle Avoidance, Proceedings of the 8th International Conference on Pattern Recognition, 1986, PP 958-960.

43 Brady, M., Durrent-Whyte, H., Leonard, J., Probert, P., Rao, B. S. Y., Sensor-Based Control of AGVs, Computing and Control Engineering Journal, March 1990.

44 Borenstein, J., Koren, Y., High Speed Obstacle Avoidance for Mobile Robots, Proceedings of the 3rd International Symposium on Intelligent Control, 1988, PP 382-384.

45 Hollingham, J., Robots Free Range Between Town and Gown, Industrial Robot, 1991, Volume 18 Number 2, PP 19-20.

46 Bair, M. E., Sampson, R., Zuk, D., 3-D Imaging and Applications, SPIE Intelligent Robotics and Computer Vision Conference Proceedings, October 1986.

47 Dunlay, R. T., Obstacle Avoidance Perception Processing for the Autonomous Land Vehicle, Proceedings of the 1988 IEEE International Conference on Robotics and Automation, 1988, PP 912-917.

48 Annaratone, M., et al, Warp Architecture and Implementation, Proceedings of the 13th Annual International Symposium on Computer Architecture, June 1986.

49 Ferrari, F., Fossa, M., Grosso, E., Marassi, M., Sandini, G., A Practical Implementation of a Multilevel Architecture for Vision-Based Navigation, Proceedings of the 5th International Conference on Advanced Robotics - Robots in Unstructured Environments, 1991, PP 1092-1098.

50 Takeno, J., Hachiyama, S., New Technology on Stereo Vision for Mobile Robots, Proceedings of the 5th International Conference on Advanced Robotics - Robots in Unstructured Environments, PP 1383-1391.

51 Takeuchi, Enomoto, Nagai, Fuzzy Control of a Mobile Robot for Obstacle Avoidance, Information Sciences, 1988, Volume 45 Part 2, PP 231-248.

52 Takeuchi, T., Kajitani, M., Guidance of a Mobile Robot using Fuzzy Control for Obstacle Avoidance, Proceedings of the Japan-China Mechatronics Symposium, October 1988, PP 1-6.

53 Lockwood, S., Mehrdadi, B., Chandler, J. R., Design of an Obstacle Avoidance System for Automated Guided Vehicles, Proceedings of the 8th International Conference on Systems Engineering, 1991, PP 428-433.

54 Hatzitheodorov, M., Kender, J. R., Optimal Illumination Method for Surface Reconstruction, Sensor Fusion 2: Human and Machine Strategies, 1989, PP 367-376.

55 Vuylsteke, P., OOsterlinck, A., 3-D Perception with a Single Binary Coded Illumination Pattern, Optics, Illumination and Image Sensing for Machine Vision, 1987, Volume 7 PP 195-202.

56 Vuylsteke, P., OOsterlinck, A., Range Image Aquisition with a Single Binary Coded Light Pattern, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 12 Number 2, February 1990.

57 Feng, K., Sugihara, K., Sugie, N., Measurement of Three-Dimensional Objects by Pattern Projection and Camera Advance, Advanced Robotics, Volume 4 Number 4, 1990, PP 319-335.

58 Bhatnager, D., Pujari, A. K., Seetharomulu, P., Static Scene Analysis Using Structured Light, Image and Vision Computing, April 1991, Volume 9 Number 2.

59 Padgham, C. A., Saunders, J. E., The Perception of Light and Colour, G. Bell and Sons Ltd., 1975, Section 4.2.

60 Gonzales, R. C., Wintz, P., Digital Image Processing, Addison-Wesley Publishing Co., 1977, Sections 2.1-2.2.

61 Uhr, L., Pattern Recognition. Learning and Thought, Prentice Hall, 1973, Chapter 3.

62 Clarke, D. E. A., Video Frame Store, Electronics and Wireless World, Reed Business Publishing, November 1987, December 1987, January 1988.

63 Nolan, J., Working With DSP, Electronics and Wireless World, January 1989, PP 52-55.

64 Lynn, P. A., An Introduction to the Analysis and Processing of Signals, The Macmillan Press Ltd., 1982, Chapter 9.

65 Niemann, H., Pattern Analysis and Understanding, Springer-Verlag, 1990, Section 1.5.

66 Bunke, H., Sanfeliu A. (Editors), Syntactic and Structural Pattern Recognition: Theory and Applications, World Scientific, 1990, Chapter 1.

67 Ock Hyun Kim, Optimal Steering Control of an Auto-Guided-Vehicle with Two Motorized Wheels, Transactions of the Institute of Measurement and Control, April-June 1987, Volume 9 Number 2.

68 Meystel, A., Hoffman, G., Koch, E., Looney, T., Intelligent Mobile System for Industrial Application, Proceedings of the 2nd Annual International Motorcon'82 Conference, 1982, PP 220-240.

69 Olgac, N., Wood, M., Optimal Trajectory Control of a Robotic Vehicle, IEEE International Symposium on Intelligent Control, 1987, PP 234-238.

70 Hongo, T., Arakama, H., Sugimoto, G., Tange, K., Yamamoto, Y., An Automatic Guidance System of a Self-Controlled Vehicle, IEEE Transactions on Industrial Electronics, 1987, Volume IE-34, PP 5-10.

71 Shen, H. C., Pang, G. K. H., An Intelligent Control of Roving Robots, Transactions of the IEEE, 1988, PP 481-485.

72 Hewlett-Packard, General Purpose Motion Control IC Technical Data HCTL-1100 Series, Hewlett-Packard.

APPENDIX 1

Embedded Computer and Memory Access Buffer Circuit Diagrams

A1.1 INTEL 8031 COMPUTER CIRCUIT

The 8Intel 031 single chip microcontroller is a version of the MCS-51 series 8-bit computers for embedded applications which does not incorporate on-board Read Only Memory (ROM). Program memory is therefore included in the circuit separately. The microcontroller provides control signals to address 64K external ram in addition to 128 bytes internal RAM.

In the circuit of figure A1.1, the lower 32K of external RAM address space is occupied by the video frame store memories and the upper 32K is used for a spare 2K RAM and an INTEL 8255 Peripheral Interface Adapter (PIA). The PIA is used to interface with the motor control embedded computers.

Withe reference to figure A1.1, the INTEL 8031 has four 8 bit input/output ports P0-P3 that can provide various functions. Ports P0 and P2 are dedicated to providing address and data lines to the external program and data memories and three port 1 lines are used to operate switching control signals. Port 3 has so-called alternate functions which include timer inputs, interrupt inputs, serial input/outputs and external RAM read and write signals. In this circuit the serial port pins RXD (P3.0) and TXD (P3.1) are configured as an RS232 link for communication with a dumb terminal. Protocol conversion circuitry has therefore been included to convert the 0-5V single levels used by the 8031 RXD/TXD to RS232.

IC1 is the INTEL 8031 microcontroller. The chip has an on-board oscillator which is

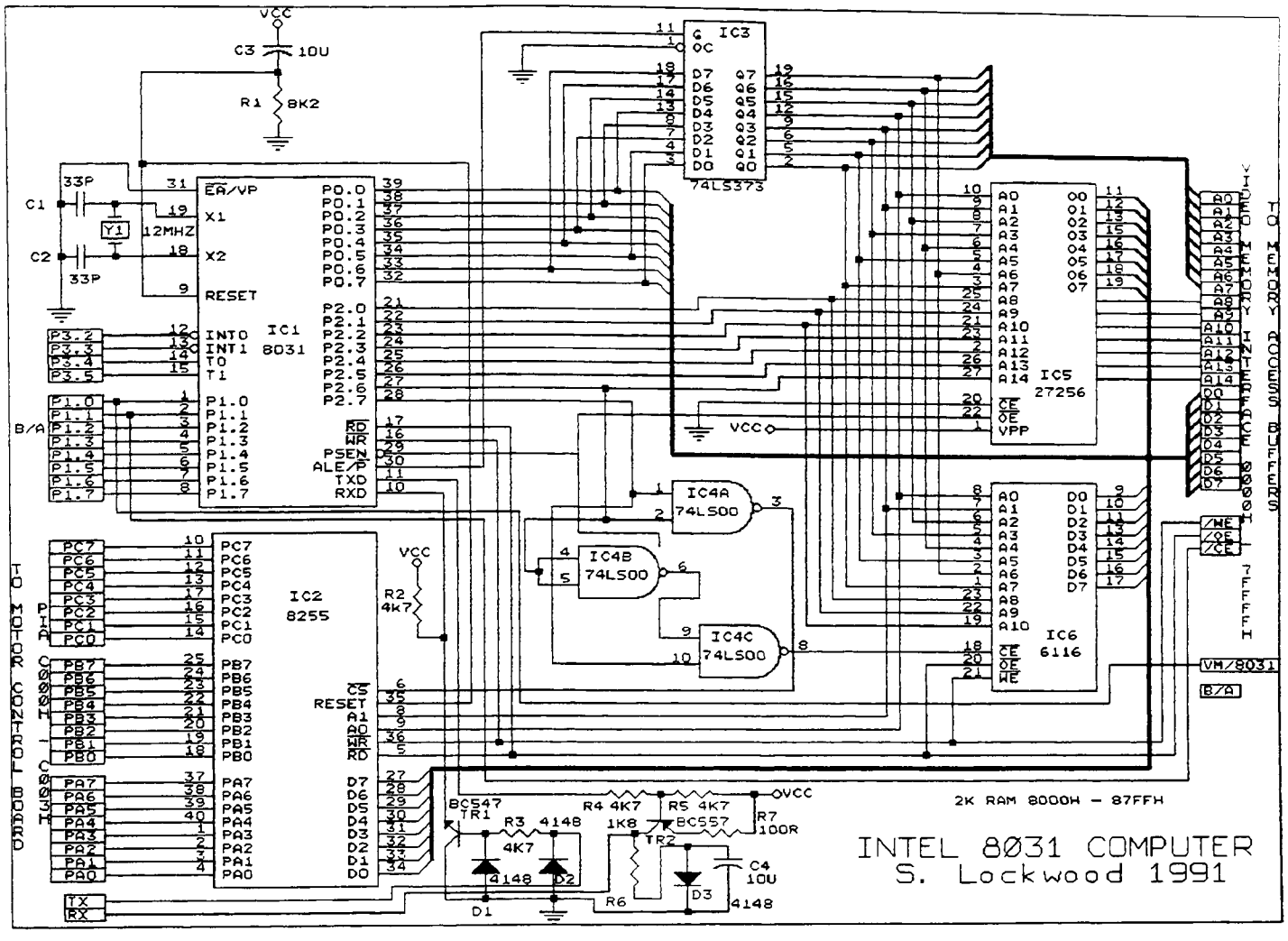


Figure A1.1
Intel 8031 Embedded Computer Circuit

configured by the external capacitors C1, C2 and the crystal Y1 to operate at 12MHZ. C3 and R2 provide a timing network which ensures that the 8031 is reset slightly after the power supply to the circuit has been switched on.

Address decoding for the video ram, 2K RAM (IC6) and the 8253 PIA (IC2) is provided by the quad NAND gate IC4. The video RAM decodes to addresses 0000h - 7FFFh, the 2K RAM to 8000h - 87FFh and the PIA to C000h-C003h.

The INTEL 8031 ports 0 and 2 are dedicated to bus functions when external program and date memory are used. The lower 8 bits (Port 0) form the multiplexed low order address byte and date byte and the 8 bit latch IC3 is included as a demultiplexer. The 8031 provides an Address Latch Enable (ALE) signal to control the latch.

In this application provision has been made for 32K program memory to be stored in

the 27256 EPROM IC5. The 8031 provides a read strobe signal (PSEN) directly to this chip.

Port pins P1.3-P1.7 and P3.2-5 are uncommitted and can be used for future expansion whilst P1.0 and P1.1 are used to control the video memory access buffers. Port pin P1.2 provides the memory select signal (video memory A or B, see Chapter 4).

The TTL-RS232 protocol converter is formed by TR1, TR2, R2-R7, C4 and D1-D3. This allows simplex communication between the INTEL 8031 and RS232 terminal.

A1.2 MEMORY ACCESS BUFFERS

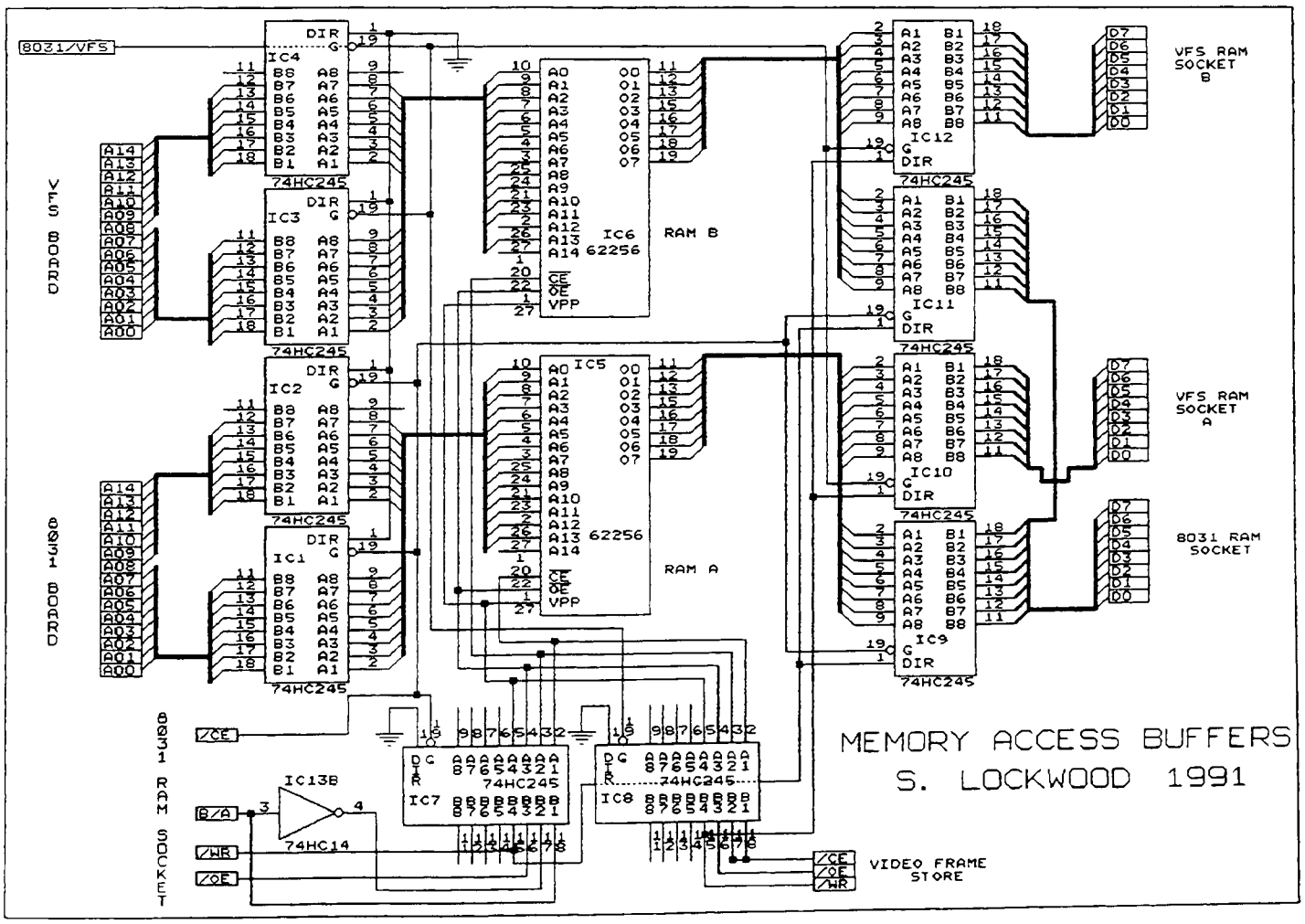


Figure A1.2
Memory Access Circuit

The circuit of figure A1.2 allows the INTEL 8031 microcontroller to 'share' the video image memory with the video frame store (62). Dual in-line header connections are used so that the circuit plugs directly into the video frame store circuit in place of the RAM chips.

When the 8031/VFS signal is low and /CE high, the 8 bit buffer chips IC3, 4, 8, 10 and 12 are enabled. The static RAMS IC5 and IC6 are then available to the video frame store and the memory access circuit is effectively transparent to the latter. However, when the 8031/VFS signal is high, ICs 3, 4, 10 and 12 are disabled with the pins in a high impedance state and the RAMs are unavailable to the video frame store. The 8031 can then gain access to the video RAMs using the /CE signal to control the buffers IC1, 2, 7, 9 and 11. The 8031 board selects which 32K RAM to access using the B/A control signal. The /CE RAM chip enable signals are made mutually exclusive by using the inverter chip IC13. This prevents both RAMS being enabled at the same time.

APPENDIX 2

Pascal Terminal Emulation Software

```
program dumbterminal;
{Program to communicate via serial port avoiding most DOS interrupts }
{ie. program accesses UARTs at low level and ignores DTS,RTS,DTR etc.}
{S. Lockwood 10/91}

  uses
    crt,dos;

  const
       com1 =1;
       com2 =2;
  var
    ch:char;
    buffer:array[0..$4000] of char;
    buffermark,i:word;

function IntToStr(i: Longint): string;
  { Convert any Integer type to a string }
  var
    s: string[11];
  begin
    Str(i, s);
    IntToStr := s;
  end;                {inttostring}

procedure auxinit(port,params:word);
{initialise serial communication port 'port' with parameter byte 'params'}

     inline(
       $58/                       {pop ax  ;pop parameters}
       $5A/                       {pop dx  ;pop port number}
       $B4/$00/                   {mov ah,0;code for initialize}
       $CD/$14);                  {int 14h ;call bios}

function charready(base:word):boolean;
{check line status to see if a character is ready for reading yet}
var temp:byte;
begin
     temp:=(port[base+5] and 1);         {isolate bit 0 of line status}
     if (temp=1) then charready:=true
          else charready:=false;
end;  {charready}

procedure readchar(comport:byte);
{read a character from serial port}
var temp,timeout,base:word;
begin
buffermark:=0;
```

```
if comport=2 then base:=$02F8 else base:=$03F8;
     timeout:=0;
     while timeout<50 do
      begin
       timeout:=timeout+1;
       if  charready(base) then
         begin
              buffer[buffermark]:=chr(port[base]);
              buffermark:=buffermark+1;
              timeout:=0            {wait until char ready}
         end;
      end;                                          {for reading}
end;  {readchar}

function charsent(base:word):boolean;
var  temp:byte;
begin
     temp:=(port[base+5] and $20); {isolate tx holding reg empty}
     if temp=$20 then charsent:=true else charsent:=false;
end;    {charsent}

procedure writechar(ch:char;comport:byte);
var base:word;
begin
     if comport=2 then base:=$02F8 else base:=$03F8;
     port[base]:=ord(ch);
     while not charsent(base) do
end;      {writechar}

procedure configure;
{configure serial port}
var
   exit:boolean;
   codebyte:byte;
   e:integer;
   baud:char;
   baudrate,parity:string;
   baudnum,chann:word;
   ch,pty,databits,stopbits:char;
begin
     ch:=' ';
     pty:=' ';
     databits:=' ';
     stopbits:=' ';
     ch:=' ';
     baud:=' ';
     exit:=false;
     codebyte:=0;
     clrscr;
```

```
  writeln;
  writeln('Serial Port Configuration');
  writeln;
  writeln;
  writeln('Which Channel? Com[1] or [2]  ');
repeat
  if keypressed then
    begin
         ch:=readkey;
         if ch=chr(27) then exit:=true;
    end;
 until (ch='1') or (ch='2') or (exit=true);
if not exit then
  begin
      writeln;
      writeln('Baud Rate? 1 - 110');
      writeln('           2 - 150');
      writeln('           3 - 300');
      writeln('           4 - 600');
      writeln('           5 - 1200');
      writeln('           6 - 2400');
      writeln('           7 - 4800');
      writeln('           8 - 9600');
     repeat
      if keypressed then
         begin
             baud:=readkey;
             if baud=chr(27) then exit:=true;
          end;
             until  (baud='1') or (baud='2') or (baud='3') or (baud='4') or  (baud='5')  or
(baud='6')
             or (baud='7') or (baud='8') or (exit=true);
       end;
       case baud of
            '1':baudrate:='110';
            '2':baudrate:='150';
            '3':baudrate:='300';
            '4':baudrate:='600';
            '5':baudrate:='1200';
            '6':baudrate:='2400';
            '7':baudrate:='4800';
            '8':baudrate:='9600';
       end;

    if not exit then
      begin
          writeln;
           writeln('Data Bits? 7,8  ');
         repeat
```

```
          if keypressed then
            begin
                 databits:=readkey;
                 if databits=chr(27) then exit:=true;
            end;
            until (databits='7') or (databits='8') or (exit=true);
end;
  if not exit then
    begin
         writeln;
         writeln('Parity? N,E,O  ');
       repeat
         if keypressed then
           begin
                pty:=readkey;
                if pty=chr(27) then exit:=true;
                pty:=upcase(pty);
           end;
           until (pty='N') or (pty='E') or (pty='O') or (exit=true);
          case pty of
               'N':parity:='No';
               'E':parity:='Even';
               'O':parity:='Odd';
          end;
end;
  if not exit then
    begin
         writeln;
         writeln('Stop Bits? 1,2  ');
      repeat
           if keypressed then
             begin
                  stopbits:=readkey;
                  if stopbits=chr(27) then exit:=true;
             end;
             until (stopbits='1') or (stopbits='2') or (exit=true);
end;
if not exit then
  begin
  case baud of
     '1':codebyte:=codebyte OR 0;
     '2':codebyte:=codebyte OR 32;
     '3':codebyte:=codebyte OR 64;
     '4':codebyte:=codebyte OR 96;
     '5':codebyte:=codebyte OR 128;
     '6':codebyte:=codebyte OR 160;
     '7':codebyte:=codebyte OR 192;
     '8':codebyte:=codebyte OR 224;
 end;
```

```
 case pty of
      'N':codebyte:=codebyte OR 0;
      'E':codebyte:=codebyte OR 24;
      'O':codebyte:=codebyte OR 8;
 end;
 case stopbits of
      '1':codebyte:=codebyte OR 0;
      '2':codebyte:=codebyte OR 4;
 end;
 case databits of
      '7':codebyte:=codebyte OR 2;
      '8':codebyte:=codebyte OR 3;
 end;
 val(ch,chann,e);
 auxinit(chann-1,codebyte);
end; {if}
writeln;
writeln;
if exit then writeln('**********  Serial Re-Configuration Aborted  **********')
else
    begin
         writeln('Serial Configuration is Now:');
         writeln;
         writeln('         Channel ',ch,', ',baudrate,' Baud, ',
              databits,' Data bits, ',stopbits,' Stop bits, ',parity,' Parity');
    end; end;
```

```
{begin dcomm}
begin
{     auxinit(1,$C3);           {initialise com2:4800,n,8,1}
{     auxinit(0,$E3);             {initialise com1:9600,n,8,1}
     configure;
     repeat
     readchar(com1);
       if buffermark<>0 then
         begin
              for i:=0 to buffermark-1 do
                  begin
                       if ord(buffer[i])<>0 then write(buffer[i]);
                  end;
         end;
     if keypressed then
        begin
            ch:=readkey;
            writechar(ch,com1);
        end;

      until false;
end.
```

APPENDIX 3

Assembler Obstacle Avoidance Software Source Code

```
c:\cbe\8051/
;Do complete preprocessing and pattern recognition
;and transmit the results on the serial port at 4800 baud
; S. Lockwood 1991

psw   equ         0D0h
b     equ         0f0h
p3    equ         0b0h
p1    equ         090h
scon  equ         98h
tl1   equ         08bh
th1   equ         08dh
ie    equ         0a8h
tmod  equ         89h
sbuf  equ         99h
tcon  equ         88h
pcon  equ         087h
dph   equ         083h
dpl   equ         082h
countequ          05fh
flagrequ          20h
rlinel            equ         31h
rlineh            equ         32h
tdpl  equ         33h
tdph  equ         34h
count1            equ         35h
count2            equ         36h
divcount          equ         37h
av1   equ         38h
avh   equ         39h
xi    equ         3ah
yi    equ         3bh
yminus1h          equ         3ch
yminus1l          equ         3dh
xarray            equ         3eh
yarray            equ         3fh
temp  equ         40h
xm2   equ         41h
xp2   equ         42h
threshold         equ         43h
f1minequ          44h
f1maxequ          45h
f2minequ          46h
f2maxequ          47h
f3minequ          48h
f3maxequ          49h
t1    equ         4ah
t2    equ         4bh
```

```
t3    equ         4ch
flminav           equ         4dh
flmaxav           equ         4eh
f2minav           equ         4fh
f2maxav           equ         50h
f3minav           equ         51h
f3maxav           equ         52h
learncount        equ         53h

caminusll         equ         54h
caminuslh         equ         55h
ip    equ         0b8h

contword          equ         21h                 ;keep track of control word
regadd                        equ         22h                 ;register address for m.c.
pendflags         equ         23h                 ;status flags for agv manual pendant
                                                        ;and motors
obstdetected      equ         24h

apl   equ         56h
apm   equ         57h
aph   equ         58h
cvl   equ         59h
cvh   equ         5ah
pl    equ         5bh
pm    equ         5ch
ph    equ         5dh
leftmost          equ         5eh
rightmost         equ         5fh
aph   equ         60h
apm   equ         61h
apl   equ         62h
genpurp           equ         63h
gain  equ         64h
pole  equ         65h
zero  equ         66h
turns equ         67h
mpointerh         equ         68h
mpointerl         equ         69h

;variables to do with obstacle avoidance

deverrorl         equ         6ah
deverrorh         equ         6bh
turnslog          equ         6ch
howmanysofar      equ         6dh
oavtempreg        equ         6eh
errtheta          equ         6fh
destheta          equ         70h
```

```
tempoa                  equ         71h
obavflags       equ         25h         ;bit addressable

            #define         maxdeverrorh #60h
            #define         maxdeverrorl #0ffh
            #define         enough #24h
            #define         allowable #30h
            #define         turnsinc #5
            #define         lor obavflags.0
            #define         canpass obavflags.1
            #define         passedyet obavflags.2

            #define         txint scon.1
            #define         rxint scon.0
            #define         sent flagr.0
            #define         recvd flagr.1
            #define res1 flagr.2
            #define         res2 flagr.3
            #define         ramflag flagr.4
            #define         eol flagr.5
            #define         eol2 flagr.6
            #define nzflag flagr.7

            #define         oeb contword.0
            #define         host8031 contword.1
            #define         ddrwr contword.2
            #define         rw contword.3
            #define         ale contword.4
            #define         csb contword.5
            #define         csa contword.6
            #define         oea contword.7

            #define         piacont #0c003h                 ;pia control register
            #define         piab #0c001h            ;pia port b
            #define         piac #0c002h            ;pia port c
            #define         piaa #0c000h            ;pia port a

            #define         keypressed pendflags.0  ;indicates if a pendant button is
                                                    ;pressed or not
            #define motoraorb pendflags.1 ;indicates which motor is addressed
            #define         actionreq pendflags.2   ;used in obstacle avoidance control
                                                    ;action routine
            #define         firstcodeflag pendflags.3           ;used in contaction

            #define forwardvel1 60h
            #define         forwardvelh 0h
            #define         backwardvel1 0e0h
            #define backwardvelh 0ffh
            #define         positionerror #80h      ;used in compareposition
```

```
            #define advinch #00h          ;used in straightmove
            #define         advincm #08h
            #define         advincl #00h

            #define         obstfromleft obstdetected.0
            #define         obstfromcentre obstdetected.1
            #define         obstfromright obstdetected.2

;define constants which represent a 5 degree turn

            #define         posinch #00 ;used in incangle
            #define         posincm #02h
            #define         posincl #090h           ;90h

            #define         neginch #0ffh           ;used in decangle
            #define         negincm #0fdh
            #define         negincl #070h           ;70h



            segment         byte at 0000-3fff 'eprom'

            public          waitabit
            public          propvelcont
            public          allstop
            public          hostop
            public          triplescan
            public          obstavop
            public          resetflags
            public          sendmessage
            public          convert
            public          turnright
            public          turnleft
            public          straightmove
            public          return
            public          readposition
            public          seeifmovefinished
            public          tempmess

            extern          report.w
            extern          absolute.w
            extern          obstav.w

start ljmp      initial
            skip  19h,0
            ljmp  timer1
            skip  05h,0
            ljmp  serial
```

```
          skip 019h,0

initial         lcall      setup                          ;set up baud rates and interrupts etc.

                      lcall learnseq    ;learn code features

aroundag        mov        dptr,#m7
                      lcall sendmessage ;send wait for key press message

                      clr  recvd                    ;ready for reception
waithere        jnb        recvd,waithere  ;wait for a char

                      mov  a,sbuf                   ;get char
                      cjne a,#'C',start;if not C then reinitialise


               lcall obstavop              ;take control of motor controllers
                   lcall resetflags       ;reset motor controller flag register

                   lcall resetmc

;*************************************************************************
;*
;*                         OBSTACLE DETECTION START
;*
;*************************************************************************

          ;scan for obstacles

                      lcall propvelcont ;host control mode
                      lcall allstop                 ;stop agv motors
                      lcall hostop                  ;select host operation

oa#1            lcall      triplescan  ;scan for obstacles
                      mov  a,obstdetected           ;if no obstacle was detected ie.

                      jz   oa#1                     ;a=0 then go round again

                      lcall obstav

          ljmp aroundag


;include motor control routines

          #include        "mcincl.asm"

;resetmc         resets and initialises motor controllers
;initidle        puts motor controllers into init/idle mode
```

```
;positioncont     puts motors into position control mode
;writeposition    writes the 24 bit 2's complement number in ph,pm,pl to
;                 motor controller position command registers
;                 if flag motoraorb=0 then write is to right motor
;                 if flag motoraorb=1 then write is to left motor
;readposition     read actual position of motor returning result in aph,apm,apl
;                 if flag motoraorb=0 then read is from right motor
;                 if flag motoraorb=1 then read is from left motor
;resetposition    Resets actual motor position registers to 0
;                 if flag motoraorb=0 then right motor position is reset
;                 if flag motoraorb=1 then left motor position is reset
;hostop           sets motor control board for host operation
;obstavop         sets motor control board for obstacle avoidance operation
;propvelcont      puts motors in proportional velocity control


;subroutine
;scan for obstacles but double check if it looks like there is one
;return result in obstdetected
triplescan
                        lcall scan                  ;scan for obstacles

                        mov  a,obstdetected         ;see if any was present
                        jz   endtriplescan          ;if not go to end

;double check                                        ;else double check
                        lcall scan
                        mov  a,obstdetected
                        jz   endtriplescan

;triplecheck
                        lcall scan

endtriplescan

                        lcall report

                        ret


;subroutine
;initiate point to point straight line trapezoidal move using
;acceleration and maximum velocities set up in trapezcont subroutine

straightmove

;reset current position command registers

                        clr  motoraorb              ;select motor a
```

```
                    lcall resetposition
                    mov   ph,#0
                    mov   pm,#0
                    mov   pl,#0

                    lcall writeposition


                    setb  motoraorb             ;select motor b
                    lcall resetposition

                    mov   ph,#0
                    mov   pm,#0
                    mov   pl,#0

                    lcall writeposition



;now write desired positions

                    clr   motoraorb   ;select motor a

                    mov   ph,advinch            ;advanceincrement
                    mov   pm,advincm
                    mov   pl,advincl

                    lcall tzwriteposition       ;write to motor controller
                    setb  motoraorb   ;select motor b
                    lcall tzwriteposition       ;write to motor controller
                    lcall trapezcont  ;start move

                    lcall triplescan  ;check for obstacles

movenotdone         lcall       seeifmovefinished       ;returns a:=1 if it is
                    jz    movenotdone
                    inc   howmanysofar
                    ret

;subroutine
;check to see if a point to point move has finished yet
;return a=0 if mov isn't finished or a=1 if it is
seeifmovefinished

                    clr   motoraorb
                    lcall compareposition
                    jnz   movnotfinished

                    setb  motoraorb
```

```
                    lcall compareposition
                    jnz   movnotfinished

                    mov   a,#1                  ;indicate that move is finished
                    ret

movnotfinished
                    clr   a
                    ret

;subroutine
;compare actual position registers with values in ph,pm,pl
;return a=0 if the error is less than positionerror else a=1
compareposition

          lcall readposition;return with actual position valuse in
                                              ;aph,apm,apl

          mov   dptr,#actual
          lcall sendmessage
          mov   a,aph
          lcall convert
          mov   a,apm
          lcall convert

          mov   dptr,#desired
          lcall sendmessage
          mov   a,ph
          lcall convert
          mov   a,pm
          lcall convert


          mov   a,ph                    ;put command position high byte in a
          subb  a,aph                   ;a=ph-aph
          jnz   cpfinish     ;jump if values aren't equal

           mov  a,pm                    ;put command middle byte in a
          subb  a,apm                   ;a=pm-apm
          jnz   cpfinish

           mov  a,pl                    ;put command low byte in a
          subb  a,apl                   ;a=pl-apl
          lcall absolute     ;get absolute error
          clr   c
          subb  a,#40h                  ;allowable position error
          jnc   cpfinish     ;if a>allowable error then jump

          clr   a
```

```
          ret                         ;positions are equal

cpfinish
          mov  a,#1                   ;positions are not equal
          ret


;subroutine
;turn AGV on the spot by writing a positive position increment to left motor
;and a negative position increment to right wheel
;wait until move is complete
turnright

          lcall triplescan            ;see if any obstacles are in the way

          lcall positioncont


          clr  motoraorb              ;prepare to address right motor
          lcall resetposition                   ;zero actual position registers
          mov  ph,neginch             ;load position registers with
                                                ;position move
          mov  pm,negincm
          mov  pl,negincl-28h
          lcall writeposition                   ;write new values to right motor cont

          setb motoraorb              ;now write to left motor
          lcall resetposition                   ;zero actual position registers
          mov  ph,posinch             ;load position registers with
                                                ;position move
          mov  pm,posincm
          mov  pl,posincl+28h
          lcall writeposition                   ;write new values to left motor cont


trwaitagain      ;mov       count,#0    ;reset counter

incwaitforpos                 ;wait until move is finished


trcarryon
          clr  motoraorb              ;access right motor
          mov  ph,neginch             ;load position registers with
                                                ;position move
          mov  pm,negincm
          mov  pl,negincl
          lcall compareposition                 ;see if right motor is in its final
                                                ;position yet
```

```
            jnz   incwaitforpos                    ;if it isn't then jump
            setb  motoraorb              ;now test left motor
            mov   ph,posinch             ;load position registers with
                                                   ;position move
            mov   pm,posincm
            mov   pl,posincl
            lcall compareposition                  ;see if left motor is in its final
            jnz   incwaitforpos                    ;position yet
            ret

;subroutine
;turn AGV on the spot by writing a negative position increment to left motor
;and a positive position increment to right wheel
;wait until move is complete
turnleft
            lcall triplescan             ;see if any obstacles are in the way

            lcall positioncont


            clr   motoraorb              ;prepare to address right motor
            lcall resetposition                    ;zero actual position registers
            mov   ph,posinch             ;load position registers with
                                                   ;position move
            mov   pm,posincm
            mov   pl,posincl
            lcall writeposition                    ;write new values to right motor cont

            setb  motoraorb              ;now write to left motor
            lcall resetposition                    ;zero actual position registers
            mov   ph,neginch             ;load position registers with
                                                   ;position move
            mov   pm,negincm
            mov   pl,negincl
            lcall writeposition                    ;write new values to left motor cont


tlwaitagain       ;mov        count,#0    ;reset counter


decwaitforpos                ;wait until move is finished


tlcarryon

            setb  motoraorb              ;access right motor
            mov   ph,neginch             ;load position registers with
                                                   ;position move
            mov   pm,negincm
```

```
            mov   pl,negincl
            lcall compareposition                  ;see if right motor is in its final
                                                         ;position yet
            jnz   decwaitforpos

            clr   motoraorb              ;now test left motor
            mov   ph,posinch             ;load position registers with
                                                         ;position move
            mov   pm,posincm
            mov   pl,posincl
            lcall compareposition                  ;see if left motor is in its final
            jnz   decwaitforpos                    ;position yet
            ret

;subroutine
;idling loop to give screen time to update

wait25ms          mov         count1,#50


w251p1                        mov         count2,#255
w251p2                        djnz        count2,w251p2
                        djnz  count1,w251p1

                        ret

;subroutine cause a delay
pause mov           r0,255
lp    mov           r1,255
lp1   djnz          r1,lp1
            djnz    r0,lp
            ret


;subroutine
;scan for obstacles and return:
;                   obstdetected = 0 for no obstacle
;                                  = 1 for obstacle approaching from left
;                                = 2 or 3 if obstacle is in mid view
;                                = 4 if obstacle is emerging from right
;                                = 5 or 7 if obstacle is filling field of view
;                   caminus11     = suggested control action

scan

                        clr   p1.0                 ;select 8031 control of video RAM
                        lcall preprocess  ;do digital filtering and cleaning up
                        lcall parse                ;find positions of valid codes
```

```
                    lcall contaction  ;send results of recognisor to serial
                                                ;port and also return with value in
                                                ;caminus11 register

                    setb p1.0                   ;select vfs control of video RAM
                    lcall wait25ms    ;wait for screen to update

                    ret


;subroutine
;Assuming avoidance is always to alter course to the right
;work out control action and send it down serial line
contaction



;find left and right most codes

                    mov  rightmost,#0;address of right most code
                    mov  leftmost,#255          ;register holding leftmost code

;start at right hand side of image

                    mov  dptr,#76ffh ;76ffh     ;array base address
                    clr  p1.1                   ;select ram b
                    clr  firstcodeflag
contactionlp1       movx       a,@dptr
                    cjne a,#16,canv  ;if not a valid code then jump

;if firstcodeflag has been set then don't look for right code anymore

                    jb   firstcodeflag,lookforleft
                    mov  rightmost,dpl          ;else get address of code
                    setb firstcodeflag          ;set flag to say that rightmostcode
                                                      ;has been found
lookforleft
                    mov  leftmost,dpl

canv                dec         dpl
                    mov  a,dpl
                    jnz  contactionlp1          ;go round until line completed

cafnd               mov         a,rightmost

cahere2
                    mov  obstdetected,#0        ;assume no obstacle is present
                    mov  a,rightmost ;get control action
                    jz   noobstacle
```

```
                    mov  a,leftmost
                    clr  c
                    subb a,#55h                 ;a=leftmost-55h
                    jnc  notfromleft ;if leftmost>55h then obstacle isn't
                                                ;emerging from left
                    setb obstfromleft;else it is
                    sjmp noobstacle
notfromleft
                    mov  a,rightmost
                    clr  c
                    subb a,#0aah                ;a=rightmost-aah
                    jc   notfromright;if a<aah then obstacle
                                                ;isn't emerging from right
                    setb obstfromright          ;else it is
                    sjmp noobstacle

notfromright    setb     obstfromcentre         ;else it is mid-view
                setb obstfromleft;if obstacle is emerging from centre
                                                ;then default to obstfromleft
noobstacle

                    ret


;subroutine
;stop agv motors
allstop

                        ;motor a

                                clr  motoraorb   ;first motor a
                                mov  cvl,#0                 ;zero velocity
                                mov  cvh,#0h

                                lcall commandvel  ;send it to mc

                                setb  motoraorb   ;now motor b
                                mov  cvl,#0                 ;zero velocity
                                mov  cvh,#0
                                lcall commandvel

                                ret


;subroutine
;send a message starting at address in dptr delimited by 0 chr to serial port
sendmessage     clr         a
```

```
                    movc a,@a+dptr   ;read string table entry
                    jnz  senmessok   ;if the chr isn't a delimiter cont.
                    mov  sbuf,#0                ;send delimiter
                    lcall wait                  ;wait until its gone
                    ret                         ;else return
senmessok       mov      sbuf,a                 ;put chr in serial register
                    lcall wait                  ;wait until its gone
                    inc  dptr
                    ljmp sendmessage ;get next character


;subroutine
;learn feature parameters
learnseq        mov      dptr,#m1               ;point at first message
                    lcall sendmessage ;send it down serial line
                    mov  sbuf,#0
                    clr  recvd                  ;clear for serial reception
learnseq1p1     jnb      recvd,learnseq1p1      ;wait here for serial reception

                    mov  a,sbuf                 ;get character
                    mov  sbuf,a                 ;echo character
                    lcall wait

                    cjne a,#'Q',learnseqcont1   ;if it isn't a Q jump
learnseqendpt   mov      dptr,#m2               ;point at aborted message
                    lcall sendmessage ;send it to serial port
                    ret                         ;finished

learnseqcont1   mov      dptr,#m3               ;point at continue message
                    lcall sendmessage ;send it

                    clr  recvd                  ;get ready for serial reception
learnseq1p2     jnb      recvd,learnseq1p2      ;wait for chr

                    mov  a,sbuf                 ;get character
                    mov  sbuf,a                 ;echo character
                    lcall wait

                    cjne a,#'C',learnseqendpt   ;if its not a C jump

                    lcall learn                 ;else learn parameters
                    mov  dptr,#m4               ;point at results message
                    lcall sendmessage ;send the message

                    mov  dptr,#m4a   ;send parameter list
                    lcall sendmessage ;
                    mov  a,flmin
                    lcall convert
```

```
mov   dptr,#m4b
lcall sendmessage
mov   a,f1max
lcall convert

mov   dptr,#m4c   ;send parameter list
lcall sendmessage ;
mov   a,f2min
lcall convert

mov   dptr,#m4d
lcall sendmessage
mov   a,f2max
lcall convert

mov   dptr,#m4e   ;send parameter list
lcall sendmessage ;
mov   a,f3min
lcall convert

mov   dptr,#m4f
lcall sendmessage
mov   a,f3max
lcall convert

mov   dptr,#m5
lcall sendmessage

ret

;subroutine
;Learn feature parameters for use by pattern recognisor
;uses f1minav,f1maxav,f2minav,f2maxav,f3minav,f3maxav
;     f1min,f1max,f2min,f2max,f3min,f3max
;leaves with parameters in f1min,f1max...
learn
                    mov   f1minav,#0
                    mov   f1maxav,#0
                    mov   f2minav,#0
                    mov   f2maxav,#0
                    mov   f3minav,#0
                    mov   f3maxav,#0

                    mov   learncount,#8          ;number of complete cycles
learnlp1      lcall       preprocess  ;process video signal
                    mov   dptr,#7640h ;base address of processed array;7610h
                    clr   p1.1                   ;select ram b
                    clr   eol                    ;reset end of line flag
```

```
                mov   f1min,#255
                mov   f1max,#0
                mov   f2min,#255
                mov   f2max,#0
                mov   f3min,#255
                mov   f3max,#0

learnlp2        lcall extfeat               ;extract t1,t2,t3 from array
                mov   a,t1
                clr   c
                subb  a,t3                  ;a=t1-t3
                jnc   learnt6               ;if t3 isn't bigger than t1 then
                                                  ;ignore this peak

                mov   a,t2
                clr   c
                subb  a,t3                  ;a=t2-t3
                jnc   learnt6               ;if t3 isn't bigger than t2 then jump

                clr   c
                mov   a,f1max
                subb  a,t1                  ;a=f1max-t1
                jnc   learnt1               ;if f1max >= t1 then jump
                mov   f1max,t1    ;else f1max=t1
learnt1         clr   c
                mov   a,t1
                subb  a,f1min               ;a=t1-f1min
                jnc   learnt2               ;if t1 >= f1min then jump
                mov   f1min,t1    ;else f1min=t1

learnt2         clr   c
                mov   a,f2max
                subb  a,t2                  ;a=f2max-t2
                jnc   learnt3               ;if f2max >= t2 then jump
                mov   f2max,t2    ;else f2max=t2
learnt3         clr   c
                mov   a,t2
                subb  a,f2min               ;a=t2-f2min
                jnc   learnt4               ;if t2 >= f2min then jump
                mov   f2min,t2    ;else f2min=t2

learnt4         clr   c
                mov   a,f3max
                subb  a,t3                  ;a=f3max-t3
                jnc   learnt5               ;if f3max >= t3 then jump
                mov   f3max,t3    ;else f3max=t3
learnt5         clr   c
                mov   a,t3
                subb  a,f3min               ;a=t3-f3min
```

```
                    jnc   learnt6                ;if t3 >= f3min then jump
                    mov   f3min,t3     ;else f3min=t3

learnt6       jnb         eol,learnlp2;if not at end of line jump

                    clr   c
                    mov   a,f1min
                    addc  a,f1minav
                    mov   f1minav,a

                    clr   c
                    mov   a,f1max
                    addc  a,f1maxav
                    mov   f1maxav,a

                    clr   c
                    mov   a,f2min
                    addc  a,f2minav
                    mov   f2minav,a

                    clr   c
                    mov   a,f2max
                    addc  a,f2maxav
                    mov   f2maxav,a

                    clr   c
                    mov   a,f3min
                    addc  a,f3minav
                    mov   f3minav,a

                    clr   c
                    mov   a,f3max
                    addc  a,f3maxav
                    mov   f3maxav,a

                    dec   learncount
                    mov   a,learncount
                    jz    learnfin
                    ljmp  learnlp1
learnfin

                    mov   a,f1minav
                    lcall divby8
                    mov   f1min,a

                    mov   a,f2minav
                    lcall divby8
                    mov   f2min,a
```

```
                        mov   a,f3minav
                        lcall divby8
                        mov   f3min,a

                        mov   a,f1maxav
                        lcall divby8
                        mov   f1max,a

                        mov   a,f2maxav
                        lcall divby8
                        mov   f2max,a

                        mov   a,f3maxav
                        lcall divby8
                        mov   f3max,a

                        ret                 ;results are now in place

;subroutine to divide single byte number in a by 8 and leave result in a
divby8          mov         b,#8
                        div   ab
                        ret


;subroutine
;preprocess video signal etc
preprocess      lcall       average                 ;transversal filter 8 lines vertically
                        lcall lpf                   ;recursive low pass
                                                          ;filter horizontally
                        lcall locatepeaks ;locate features in result
                        ret


;subroutine
;Average 8 lines starting at address 7b00h ,destination 74h


average
            mov   rline1,#00            ;this is where the result will go
            mov   rlineh,#74h
            mov   dptr,#7b00h           ;starting address
            mov   count2,#128           ;load count2 with column counter

count2loop      mov         tdpl,dpl    ;temporarily store data pointer
                        mov   tdph,dph    ;or column address

;ram a
                        setb  p1.1                  ;select RAM A
                        mov   av1,#0                ;set current average to 0
```

```
                    mov   avh,#0
                    mov   count1,#8   ;load count1 with line number
count1loopa         movx        a,@dptr     ;get first value from ram a

                          clr   c           ;add this value to average so far
                          addc  a,avl
                          mov   avl,a
                          clr   a
                          addc  a,avh
                          mov   avh,a
                    mov   a,#128                    ;increment data pointer to next line
                          clr   c
                          addc  a,dpl
                          mov   dpl,a
                          clr   a
                          addc  a,dph
                          mov   dph,a

                          djnz  count1,count1loopa      ;go round until
                                                                  ;8 lines have been
                                                                  ;done



                    mov   divcount,#3 ;prepare to divide average by 8
divloopa            mov         a,avh                 ;divide the average by 8
                          rrc   a
                          mov   avh,a
                          mov   a,avl
                          rrc   a
                          mov   avl,a
                          djnz  divcount,divloopa

                          mov   dpl,rline1  ;prepare to store the result
                          mov   dph,rlineh
                          movx  @dptr,a               ;store it
                          mov   dpl,tdpl    ;retrieve column address
                          mov   dph,tdph

;ram b
                clr   p1.1                            ;select RAM B
                          mov   avl,#00               ;set average to 0
                          mov   avh,#00
                          mov   count1,#8   ;load count1 with line count
count1loopb               movx        a,@dptr     ;get a value

                                clr   c           ;add it to average so far
                                addc  a,avl
                                mov   avl,a
```

```
                        clr  a
                        addc a,avh
                        mov  avh,a

                        mov  a,#128      ;set dptr to next line
                        clr  c
                        addc a,dpl
                        mov  dpl,a
                        clr  a
                        addc a,dph
                        mov  dph,a

                        djnz count1,count1loopb      ;go round again until
                                                                ;8 lines have been
                                                                ;done

                mov  divcount,#3 ;prepare to divide result by 8
divloopb        mov      a,avh                   ;do division
                rrc  a
                mov  avh,a
                mov  a,avl
                rrc  a
                mov  avl,a
                djnz divcount,divloopb
tst2

                mov  dpl,rlinel  ;get ready to store result
                mov  dph,rlineh
                movx @dptr,a                    ;store it
                inc  dptr                       ;increment result address
                mov  rlinel,dpl
                mov  rlineh,dph
                mov  dpl,tdpl    ;retrieve column address
                mov  dph,tdph
                inc  dptr                       ;increment to next column

                dec  count2                     ;have all the columns been done yet?
                mov  a,count2
                jz   finished
                ljmp count2loop
finished        ret                             ;finished averageing eight lines


;subroutine
                ;now do low pass filter
                ;using algorithm y[i]:=(x[i]+y[i-1]*t) div (t+1) for t=3
                ;use 1500-75ff to store result
```

```
                    ;also put result in 7600h - 76ff RAM A

;set initial values

lpf             mov     tdpl,#0
                mov     tdph,#76h   ;destination

                mov     yminus1l,#0
                mov     yminus1h,#0
                mov     xarray,#74h ;source
                mov     yarray,#75h ;destination
                mov     rline1,#0   ;source array low byte
                mov     dpl,#00



filterloop              setb    p1.1                ;select ram a
                        mov     dph,xarray
                        mov     dpl,rline1  ;source low byte
                        movx    a,@dptr             ;get xi
                        lcall   docalcs             ;calculate filter algorithm
                        mov     a,yi
                        mov     dph,yarray
                        movx    @dptr,a             ;store result

;also put result in array starting at address 7600h RAM A

                        setb    p1.1                ;select ram a
                        mov     rline1,dpl  ;save source low byte
                        mov     dph,tdph    ;destination
                        mov     dpl,tdpl    ;
                        movx    @dptr,a             ;store result
                        inc     dptr                ;point at next location
                        mov     tdpl,dpl    ;save address

                        clr     p1.1                ;select ram b
                        mov     dpl,rline1  ;retrieve source address
                        mov     dph,xarray
                        movx    a,@dptr             ;get next xi value
                        lcall   docalcs             ;calculate filter algorithm
                        mov     a,yi
                        mov     dph,yarray
                        movx    @dptr,a             ;store result

;also put result in array starting at address 1600h RAM A

                        setb    p1.1                ;select ram a
```

```
                        mov   rlinel,dpl   ;save source low byte
                        mov   dph,tdph     ;get destination address
                        mov   dpl,tdpl     ;
                        movx  @dptr,a                ;store value
                        inc   dptr
                        mov   tdpl,dpl               ;save destination address

                        inc   rlinel                 ;increment address to next
                                                           ;value
                        mov   a,rlinel
                        cjne  a,#128,filterloop          ;finished yet?
                        ret

;subroutine
;now extract maxima and minima from the remaining array which starts at
;address [xarray]-75h and put resulting peaks in [yarray]-76h

locatepeaks  ;destination array 7600h-76ffh RAM B

            mov   xarray,#75h            ;input array
            mov   yarray,#76h            ;output array
            mov   rlinel,#01             ;use rlinel as xarray low byte
            mov   tdpl,#01               ;use tdpl as yarray low byte
            mov   dpl,#01                          ;input and output array low byte
            mov   count2,#128            ;use count2 to store column ref

            clr   p1.1                             ;select ram b
            mov   dptr,#7600h            ;set undefined zero location of
            clr   a                                ;output array to 0
            movx  @dptr,a
lploop2           mov         count1,#02  ;use count1 to reference RAM A & B

                        clr   ramflag                ;use ram flag to keep track of p1.1
                        mov   c,ramflag   ;p1.1=0
                        mov   p1.1,c

lploop1           cpl         ramflag                ;select other ram
                        mov   c,ramflag
                        mov   p1.1,c

;get x[i],x[i-2] and x[i+2]

                        mov   dpl,rlinel  ;get xarray low byte
                        mov   dph,xarray  ;set up for getting xi
                        movx  a,@dptr                ;get x[i]
                        mov   xi,a

                        dec   dpl
```

```
                        movx  a,@dptr                ;get x[i-2]
                        mov   xm2,a

                        inc   dpl
                        inc   dpl

                        movx  a,@dptr                ;get x[i+2]
                        mov   xp2,a

                        dec   dpl                    ;set dptr to point back to x[i]

                        mov   rline1,dpl  ;save xarray column count

                        mov   dph,yarray  ;set dptr to point at output array
                        mov   dpl,tdpl    ;get yarray column count

;next get values of:          x[i]-x[i-2] and x[i]-x[i+2]
;the result is res1 set if:   x[i] < x[i-2] and res2 set if x[i] < x[i+2]
;             res1 clear if x[i] >= x[i-2] and res2 clear if x[i] >=x[i+2]

                        clr   c
                        mov   a,xi
                        subb  a,xm2                  ;a:=x[i]-x[i-2]

                        mov   res1,c                 ;carry will be set if result < 0

                        clr   c
                        mov   a,xi
                        subb  a,xp2                  ;a:=x[i]-x[i+2]

                        mov   res2,c

;now do tests to detect maxima and minima
;if res1 AND res2 (x[i] < x[i-2] AND x[i] < x[i+2]) then yarray[i]:=1 (-ve)
;if not res1 AND res2 (x[i] >= x[i-2] AND x[i] >= x[i+2] then
;                                                                    yarray[i]:=2 (+ve)
;else yarray[i]:=0


lpnotzero       mov         c,res1
                        anl   c,res2                 ;if x[i] < x[i-2] AND x[i] < x[i+2]
                        jc    lpneg                  ;then yarray[i] is a minimum
                        mov   c,res1
                        cpl   c                      ;res1:=-res1
                        anl   c,/res2                ;if x[i] >= x[i-2] AND x[i] >= x[i+2]
                        jc    lpposi                 ;then yarray[i] is a maximum
                        clr   a                      ;else yarray[i]:=0
                        ljmp  lpnext
```

```
lpneg           mov       a,#01                 ;yarray[i]:=1 (-ve)
                      ljmp lpnext

lpposi                    mov       a,#02                 ;yarray[i]:=2 (+ve)

lpnext                    clr       p1.1                  ;select ram b
                      movx @dptr,a               ;write value to yarray
                      inc  dptr                  ;next yarray column
                      mov  tdpl,dpl    ;save yarray low byte

                      djnz count1,lploop1        ;go round again for next RAM

          inc  rline1                 ;next xarray address
          djnz count2,lploop2         ;go round again for next column
          ret

;subroutine
docalcs                   ;do (y[i-1]*3+xi)/(4)
                     ;enter this subroutine with a=xi
                     ;return with result in yi

                     mov  xi,a
                     mov  a,yminus11
                     mov  b,#7
                     mul  ab                        ;do y[i-1]*t
                     mov  yminus1h,b
                     mov  yminus11,a  ;store result

                     ;do y[i-1]*3+x[i]

                     clr  c
                     addc a,xi
                     mov  yminus11,a
                     clr  a
                     addc a,yminus1h
                     mov  yminus1h,a

                     ;do (y[i-1]*3+x[i]) div 4

                     mov  divcount,#3 ;prepare to divide by 4
divlpfilt       mov       a,yminus1h  ;do division
                     rrc  a
                     mov  yminus1h,a
                     mov  a,yminus11
                     rrc  a
                     mov  yminus11,a
                     djnz divcount,divlpfilt
                     mov  yi,yminus11
                     ret
```

```
;subroutine
;parse remaining peaks array for valid codes
parse           mov       dptr,#7600h ;base address of array
                clr  p1.1                    ;select ram b - peaks array]
                clr  eol2                    ;clear end of line flag


parselp         lcall     extfeat            ;extract fetures from peaks array
                ;t1,t2,t3 now = features
                ;dptr = address of first +ve peak

                lcall testfeat    ;test features for pattern class
                movx @dptr,a                 ;put result in peaks array
                                                   ;ie. peaks[] = 10h if a code exists

                jnb  eol2,parselp
                ret

;subroutine
;extract t1,t2,t3 from peaks array
;enter with current address in dptr
extfeat         mov       a,#02              ;+ve peak
                lcall fnp                    ;find next +ve peak
                ;returns with dptr=address, a=diff between exit and entry adds.

                mov  a,#01                   ;-ve peak
                lcall fnp                    ;find next -ve peak
                mov  t1,a                    ;first feature

                mov  a,#02                   ;+ve peak
                lcall fnp                    ;find next +ve peak
                mov  t2,a                    ;second feature

                mov  tdpl,dpl     ;starting point of next search

                mov  a,#01                   ;-ve peak
                lcall fnp                    ;find next -ve peak
                mov  t3,a                    ;third feature

                mov  dpl,tdpl     ;start for next search
                ret

;subroutine
;find next peak
;enter with sense in a (1=-ve, 2=+ve)
;enter with current address in dptr
;exit with a = difference between current address and address of next
```

```
;peak with sense 'sense'
fnp             mov     temp,a                  ;save sense
                mov     rline1,dpl  ;save current address
fnplp           inc     dptr                    ;next location
                mov     a,dpl
                cjne    a,#239,fnokcont1 ;used for learning only;239
                setb    eol
fnokcont1       cjne    a,#255,fnokcont2 ;are we at end of line yet?
                setb    eol2                     ;yes set flag


fnokcont2       movx    a,@dptr                 ;get value from peaks array
                cjne    a,temp,fnpnxt           ;no this isn't correct peak goto
                                                        ;end of line test
                ljmp    fnpfound    ;else leave loop because peak found
fnpnxt                  mov         a,dpl
                jnz     fnplp                   ;if not end of line go round again
                mov     a,#0                    ;if end of line set this feature to 0
                ret                             ;leave subroutine

fnpfound        ;dptr=address of peak
                mov     a,dpl                   ;current address low byte
                clr     c
                subb    a,rline1    ;find difference between addresses
                ret

;subroutine
;test features for membership of code class pattern
; rules are : f1min <= t1 <= f1max
;  AND                 f2min <= t2 <= f2max
;  AND       f3min <= t3 <= f3max    in order to be a code class pattern

;first find individual results t1<=f1max, t1 >=f1min etc

testfeat        clr         c
                mov     a,f1max
                subb    a,t1                    ;a=f1max-t1
                mov     res1,c
                cpl     res1                    ;if res1 = 1 then f1max>=t1

                clr     c
                mov     a,t1
                subb    a,f1min                 ;a=t1-f1min
                cpl     c                       ;if c=1 then t1>= f1min
                anl     c,res1                  ;c=1 if f1min <= t1 <= f1max
                mov     res1,c                  ;res1 = result

                clr     c
                mov     a,f2max
```

```
subb  a,t2                    ;a=f2max - t2
cpl   c                       ;c=1 if f1max > t2
anl   c,res1                  ;
mov    res1,c                 ;res1 = 1 if a code so far

clr    c
mov   a,t2
subb  a,f2min                 ;a=t2-f2min
cpl   c
anl   c,res1
mov   res1,c

clr   c
mov   a,f3max
subb  a,t3                    ;a=f3max - t3
cpl   c
anl   c,res1
mov   res1,c

clr   c
mov   a,t3
subb  a,f3min                 ;a=t3-f3min
cpl   c
anl   c,res1
;c=1 if this is a code else c=0
                      clr   a
                      jnc   testfend
                      mov   a,#10h                ;indicate code present
testfend          ret


;subroutine
; to idle while waiting for a serial interrupt
wait  jnb         sent,wait
            clr   sent
            ret


;subroutine
convert                                   ;convert single byte to 2 hex digits
            mov   r5,a
            swap  a
            anl   a,#0fh
            lcall decode
            mov   r6,a
            mov   a,r5
            anl   a,#0fh
            lcall decode
```

```
            mov   r7,a
            mov   sbuf,#24h    ;$
            lcall wait
            mov   sbuf,r6
            lcall wait
            mov   sbuf,r7
            lcall wait
            mov   sbuf,#' '
            lcall wait
            ret

;subroutine
decode
            mov   r4,a
            clr   c
            subb  a,#10
            jnc   big
            mov   a,r4
            add   a,#30h
            ret
big   mov         a,r4
            add   a,#37h
            ret


;subroutine
setup
            clr   p1.0                  ;select 8031
            clr   sent                  ;initialise sent flag
            clr   nzflag                ;flag indicating a nonzero control action
            clr   actionreq   ;flag indicating that oa is required

            setb  ie.7                  ;enable all interrupts
            orl   pcon,#80h   ;set double baud rate bit in pcon
            mov   th1,#243    ;(0F3)timer 1 reload value to give 4800 baud
            orl   tmod,#20h   ;set timer 1 for mode 2 auto-reload
                                        ;8 bit timer/counter
            setb  scon.6                ;set serial port in mode 1
            clr   scon.7                ;(start bit, 8 data bits, 1 stop bit)
            clr   tcon.2                ;intl low levekl triggered
            setb  ip.4                  ;intl high priority

            setb  tcon.6                ;set timer 1 run control bit (ie. start tl)
            setb  ie.4                  ;enable serial interrupts
;     setb        ie.2                  ;enable intl
            clr   rxint                 ;clear serial interrupt rx
            setb  scon.4                ;enable serial reception

            lcall pause                 ;allow pia time to reset
```

```
            mov   a,#90h                    ;control word for pia
            mov   dptr,piacont;control register
            movx  @dptr,a                   ;set ports b and c outputs, a input
                                                  ;8 bit timer/counter

            mov   f1min,#2    ;2            ;set default values for features
            mov   f1max,#6    ;5
            mov   f2min,#2    ;2
            mov   f2max,#6    ;5
            mov   f3min,#4    ;4
            mov   f3max,#18h  ;9

            lcall hostop                    ;make sure that agv is under host
                                                  ;computer operation
            mov   caminus1l,#0;initialise control action registers
            mov   caminus1h,#0

;initialise digital filter parameters for motor controllers
            mov   gain,#05h   ;01h
            mov   pole,#040h  ;40h
            mov   zero,#0e5h  ;e5h

            ret

;interrupt subroutine
serial          jnb       txint,nottx           ;jump if not a transmit interrupt
            setb  sent                  ;set busy flag
            clr   txint                 ;reset serial interrupt
            reti

nottx setb      recvd                   ;must be a receive interrupt
            clr   rxint                 ;reset serial interrupt
            reti

;interrupt subroutine
timer1          reti


m1    string    13,10,13,10,"OBSTACLE AVOIDANCE LEARNING SEQUENCE",13,10,13,10
          string  "If Not Required Press 'Q' Now, Any Other Key Will Continue The Sequence",13,10,0
m2    string    13,10,13,10,"  Learning Sequence Aborted, System Now Using Defaults",13,10,0
m3    string    13,10,13,10,"  Position Calibration Object and Press 'C' To Continue",13,10,0
m4    string    13,10,13,10,"  Learned Feature Parameters Are:",13,10,13,10,0
m4a   string    13,10,"     T1min = ",0
m4b   string    13,10,"     T1max = ",0
m4c   string    13,10,13,10,"     T2min = ",0
m4d   string    13,10,"     T2max = ",0
```

```
m4e  string      13,10,13,10,"     T3min = ",0
m4f  string      13,10,"     T3max = ",0
m5   string      13,10,13,10,13,10,"  System is Now Running",13,10,0
m6   string      13,10,13,10,"  Cont Act. is:",13,10,0
m7     string      13,10,13,10,"  Press 'C' to continue, any other  key  will re-initialise the
system",13,10,13,10,0
actual           string      13,10,"Actual Position : ",0
desired          string      "  Desired Position : ",0
propvela         string      13,10,"Right Motor velocity is : ",0
propvelb         string      "      Left Motor velocity is : ",0
posa string      13,10,"Right motor actual position : ",0
posb string      13,10,"Left motor actual position : ",0
obstpos          string      13,10,"  Obstacle position code : ",0
camessage        string      "   Control Action : ",0
obstdet          string      13,10,"Obstacle Detected Code : ",0
turned           string      13,10,"AGV is now turning to avoid obstacle",0
proceed          string      13,10,"AGV is now proceeding past obstacle",0
movesmessage     string      13,10,"Details of move segments made : ",13,10,0
return                       string 13,10,0
tempmess         string      13,10,"Writing moves table",13,10,0

heremess         string      13,10,"here",13,10,0


            end
```

```
c:\cbe\8051/
;obstacle avoidance routines


            #include        "definitl.asm"

            segment         'eprom'

            extern          waitabit.w
            extern          propvelcont.w
            extern          allstop.w
            extern          hostop.w
            extern          triplescan.w
            extern          obstavop.w
            extern          resetflags.w
            extern          sendmessage.w
            extern          convert.w
            extern          turnright.w
            extern          turnleft.w
            extern          straightmove.w
            extern          return.w
            extern          readposition.w
            extern          seeifmovefinished.w
            extern          tempmess

            public          absolute
            public          report
            public          obstav


obstav                      ;subroutine name called from obstav4 when an obstacle has
                        ;been detected

                        mov  dptr,#hellomess
                        lcall sendmessage

;take control and stop agv

;entry code

            lcall obstavop    ;take control of motor controllers
            lcall resetflags  ;reset motor controller flag register

;obstacle avoidance routine
```

```
            mov  howmanysofar,#0          ;zero advances counter
            mov  deverrorl,#0;zero deviation from path low byte
            mov  deverrorh,#0;zero deviation high byte
            mov  turnslog,#00h            ;zero turns made log
            clr  passedyet   ;flag to say if AGV has been passed or not
            setb canpass                  ;flag to say that AGV can avoid obstacle


;REPEAT
oarepeat
            lcall leftorright ;find out which way to avoid obstacle
                                          ;returns lor=0 if left
                                          ;     lor=1 if right

            jnb  canpass,ao#1;if can't pass obstacle then jump
            lcall turnagv                 ;else turn to avoid it
            clr  passedyet   ;clear flag to say not passed obstacle



ao#1  jb          passedyet,ao#2          ;if passed obstacle then jump
            jnb  canpass,ao#2;if can't pass then jump
            lcall advanceagv  ;if not passedyet AND canpass then advance AGV

ao#2  mov         a,obstdetected                ;if obstacle detected then jump
            jnz   ao#3
            jnb   passedyet,ao#3                ;if not passedyet then jump
            lcall recover                       ;if not obstdetected AND passedyet then


                                                      ;recover to original path

ao#3  mov         a,obstdetected                ;if not obstdetected then
            jz    oapossfin             ;possible finish

oaorcond
            jnb   canpass,oafinished    ;if not canpass then finished
            ljmp  oarepeat              ;if not obstdetected AND not passedyet then
                                                ;go round again
ao#4
oapossfin
            jb    passedyet,oafinished  ;if passedyet then finished
            ljmp  oaorcond

oafinished
```

```
;UNTIL (not obstdetected) and (passedyet) or (not canpass)

;Exit Code

            lcall propvelcont ;host control mode
            lcall allstop               ;stop agv motors
            lcall hostop                ;select host operation
            ret


leftorright
;subroutine to determine whether to attempt to pass the obstacle from the
;left or right
;returns lor=0 for left
;     lor=1 for right

            jnb  obstfromleft,lr#1      ;if obstacle is mostly to the right
                                                ;then jump
            setb lor                            ;else set lor and avoid to right
            ljmp lr#end

lr#1 clr        lor                             ;clr lor and avoid to left

lr#end          ret                                     ;return

turnagv
;subroutine to turn agv in the direction given by lor to avoid obstacle
;ie. if lor=0 then turn left
;    lor=1 then turn right
;returns canpass=0 if agv can't turn to avoid obstacle

            cpl  lor

tagvwhile
            mov  a,obstdetected
            jz   tagv#1                         ;WHILE obstacle detected
            jnb  canpass,tagv#1                 ;AND canpass
                        lcall turndir           ;DO turn (uses lor and returns obstdetected)

                        lcall testturn  ;returns canpass=0 if can't avoid obstacle



            ljmp tagvwhile              ;end WHILE

tagv#1                      jnb         canpass,tagv#3          ;if can't pass then jump
                        lcall turndir               ;else do a couple of turns to make
                        lcall turndir
```

```
tagv#2
                    cpl   lor
                    ret                         ;return

tagv#3          mov         dptr,#turnfail
          lcall sendmessage
          ljmp  tagv#2

turndir
;subroutine to turn according to lor
;increments or decrements turnslogas appropriate

          jb    lor,td#left             ;if lor = 1 then turn right
                lcall turnright         ;else turn right
                dec   turnslog
                ljmp  td#end

td#left                 lcall       turnleft    ;turnleft
                inc   turnslog
td#end          ret

testturn
;subroutine which tests to see if the agv can turn effectively to avoid obstacle
;returns canpass=1 if it can, or canpass=0 if it can't

          jnb   obstfromleft,tt#1       ;IF obstafromleft
          jb    obstfromright,tt#1      ;AND not obstfromright
           jnb  lor,tt#1                ;AND lor=left
                clr   canpass                   ;THEN can't pass
                ljmp  tt#end

tt#1 jnb        obstfromright,tt#2      ;OR IF obstfromright
          jb    obstfromleft,tt#2       ;AND not obstfromleft
          jb    lor,tt#2                ;AND lor=right
                clr   canpass                   ;THEN can't pass
                ljmp  tt#end

tt#2 setb       canpass                         ;ELSE canpass = true

tt#end          ret


advanceagv
;subroutine to advance agv passed an obstacle
;uses maxdeverror
;returns canpass=0 if can't pass obstacle, otherwise canpass=1
;returns passedyet=true if obstacle has been passed

          setb  canpass                 ;assume obstacle can be passed
```

```
            mov   howmanysofar,#0          ;advance counter
            clr   passedyet    ;flag to say that obstacle hasn't been passed yet

advwhile
            mov   a,obstdetected           ;WHILE not obstdetected
            jnz   adv#1
            jb    passedyet,adv#1          ;AND not passedyet
            jnb   canpass,adv#1            ;AND canpass
                  lcall straightmove;DO advance agv
                  lcall enoughadvances          ;inc counter and check range
                                                      ;returns passedyet=true or false
;look up deviation error
            lcall getdeviation;uses current deverror and turnslog

            clr   c
            mov   a,deverrorl ;a=deverror low byte
            mov   b,deverrorh ;b=deverror high byte
            lcall absolute16  ;get abs(b:a)

            clr   c
            subb  a,maxdeverrorl           ;a=a-maxdeverror low byte
            jc    adv#2                    ;if a<maxdeverror low byte then jump
            mov   a,b                      ;otherwise check deviation high byte
            clr   c
            subb  a,maxdeverrorh
            jc    adv#2                    ;jump if a<maxdeverror high byte
            clr   canpass                  ;else can't pass obstacle
            mov   dptr,#advfail
            lcall sendmessage

adv#2 ljmp        advwhile     ;go round again

adv#1 mov         a,obstdetected           ;IF obstdetected
            jz    adv#3
            jb    passedyet,adv#3          ;AND not passedyet
            mov   a,howmanysofar           ;AND......
            clr   c
            subb  a,#4                     ;a=a-4
            jc    adv#3                    ;...AND howmanysofar>=4

            jb    lor,adv#100 ;then if lor=left
            jnb   obstfromleft,adv#100     ;and obstfromleft
            clr   canpass                  ;then can't pass obstacle
            mov   dptr,#advfail
            lcall sendmessage
            ljmp  adv#3

adv#100
            jnb   lor,adv#3    ;OR if lor=right
```

```
            jnb   obstfromright,adv#3      ;AND obstfromright
            clr   canpass                  ;then can't pass obstacle
            mov   dptr,#advfail
            lcall sendmessage


adv#3 ret


enoughadvances
;subroutine to increment advances counter and check to see if it is
;in range
;returns passedyet=true if agv has passed obstacle, else passedyet=false

            mov   a,howmanysofar
            clr   c
            subb  a,enough
            jc    ea#1                     ;if a>=enough then passedyet=true
            setb  passedyet
            ljmp  ea#end

ea#1 clr          passedyet

ea#end

            ret
```

```
absolute16
;converts the 16 bit number in B:A to its 2's complement +ve form
;result returned in B:A

            mov   tempoa,a                 ;put lsbyte in temporary location

            mov   a,b                      ;transfer b to a for logical operations
            anl   a,#80h                   ;see if msb of b is set
            jz    abs16#end   ;if its not then jump

            mov   a,tempoa                 ;retrieve a value
            clr   c
            cpl   a                        ;do 2's complement
            addc  a,#01                    ;increment a
            mov   tempoa,a                 ;save a temporarily
            mov   a,b                      ;get b
            cpl   a
            addc  a,#0                     ;add carry flag
            mov   b,a                      ;restore value to b
            mov   a,tempoa                 ;restore value to a
            ljmp  abs16#return
```

```
abs16#end
            mov   a,tempoa                ;restore value in a

abs16#return
            ret


absolute
;converts the number in a to a positive value
;result returned in a

            mov   tempoa,a
            anl   a,#80h        ;see if msb is set
            jz    abs#end       ;jump if its not

            mov   a,tempoa      ;retrieve value
            cpl   a             ;do 2's complement
            inc   a             ;
            mov   tempoa,a
abs#end
            mov   a,tempoa
            ret
            clr   passedyet
            ret

getdeviation
;subroutine to look up deviation error and update deverror

            mov   a,turnslog
            jz    getdev#end

            lcall absolute              ;get abs(turnslog)
            mov   dptr,#ltdeverror      ;set dptr to beginning of look up table
            movc  a,@a+dptr             ;get table entry
            mov   tempoa,a                        ;temporarily store table entry
            mov   a,turnslog
            anl   a,#80h                          ;see if turnslog is -ve
            jz    getdev#pos            ;jump if its positive
            mov   a,deverrorl           ;a=deverror
            clr   c
            subb  a,tempoa                        ;deverror low byte=a-table entry
            mov   deverrorl,a           ;
            mov   a,deverrorh           ;subtract borrow off high byte
            subb  a,#0
            mov   deverrorh,a
            ljmp  getdev#end

getdev#pos
            mov   a,deverrorl
```

```
            clr  c
            addc a,tempoa                          ;a=a+table entry
            mov  deverrorl,a
            mov  a,deverrorh
            addc a,#0
            mov  deverrorh,a

getdev#end
            ret


recover
;subroutine to return agv to original guide path
;uses allowable (error) = 2
;uses turnsincrement


recwhile
            mov  a,obstdetected
            jz   rec#1carryon              ;WHILE not obstdetected
            ljmp rec#1

rec#1carryon

            mov  a,deverrorl ;a=deverror low byte
            mov  b,deverrorh ;b=deverror high byte
            lcall absolute16  ;b:a=abs(b:a)

            mov  tempoa,a                  ;temporarily store a
            mov  a,b                       ;test to see if high byte is zero
            jnz  rec#2                     ;if high byte isn't 0 then no point testing low byte

            mov  a,tempoa                  ;else test deverror low byte
            clr  c
            subb a,allowable ;a=a-allowable
            jnc  rec#2                     ;if a>=allowable then jump
            mov  a,turnslog  ;a=turnslog
            jnz  rec#2                     ;if turnslog<>0 then jump
            ljmp rec#1


;look up new desired theta value
rec#2

            mov  dptr,#ltdestheta          ;base of look up table
            mov  a,deverrorl               ;a=deverrorl
            mov  b,deverrorh               ;b=deverrorh
            lcall absolute16               ;b:a=abs(b:a)
            mov  tempoa,a                          ;temporarily save deverror low byte
```

```
            mov   a,b                                   ;a=deverror high byte
            jnz   rec#1000                  ;if its zero then look up desired theta
            mov   destheta,#5
            ljmp  rec#200

rec#1000
            clr   c
            subb  a,#12                                 ;a:=a-3
            jnc   rec#5000                  ;if a >3 then jump
            mov   destheta,#20              ;else destheta=20 degrees
            ljmp  rec#200
rec#5000
            mov   a,b
            mov   destheta,#40              ;else set destheta to 40 degrees
            ljmp  rec#200

rec#100           mov         a,tempoa                          ;restore a with deverror low byte
            movc  a,@a+dptr                 ;get table entry pointed at by
                                                        ;deverror low byte
            mov   destheta,a                ;=desired theta

rec#200           mov         a,deverrorh             ;a=deverrorh
            anl   a,#80h                                ;see if deverror is -ve
            jnz   rec#3                                 ;if it is jump
            mov   a,destheta                ;else       if its +ve
            cpl   a                                     ;destheta=-destheta
            inc   a
            mov   destheta,a                ;destheta=a

rec#3 mov         a,turnslog                ;a=turnslog
            anl   a,#80h                                ;see if a is -ve
            jz    recmul#pos                ;if its +ve then jump
            mov   a,turnslog
            lcall absolute                  ;make turnslog +ve
            setb  nzflag                                ;set a flag to say that it has been done
            ljmp  rec#domul

recmul#pos
            clr   nzflag                                ;clear flag to say turnslog was +ve
            mov   a,turnslog

rec#domul
            mov   b,turnsinc                ;b=turnsinc
            mul   ab                                    ;b:a=a*b
            jnb   nzflag,rec#mulend         ;if original turnslog was +ve then jump
            cpl   a                                     ;else do 2's comp
            inc   a

rec#mulend
```

```
            mov   tempoa,a                  ;temporarily store a
            mov   a,destheta          ;a=destheta
            clr   c
            subb  a,tempoa                  ;a=destheta-tempoa
            jb    lor,rec#4           ;if avoidanceis to right then jump
            cpl   a                         ;else a=-a
            inc   a

rec#4 mov         errtheta,a            ;errtheta=a

            anl   a,#80h                    ;see if errtheta<0
            jnz   rec#5                     ;jump if it is

            mov   a,errtheta
            clr   c
            subb  a,turnsinc          ;a=errtheta-turnsinc
            jc    rec#5                     ;if a<turnsinc then jump
            lcall turndir                   ;turn agv
            ljmp  rec#6

rec#5 cpl         lor                       ;
            lcall turndir                   ;turn(-lor)
            cpl   lor

rec#6 lcall       straightmove          ;advance agv a bit
            lcall getdeviation        ;get current deviation from look up table
             ljmp recwhile                ;go round again

            mov   dptr,#heremess
            lcall sendmessage

            lcall report

rec#1 ret                                                 ;end


report
;subroutine to report status of various variables and flags
            mov   dptr,#reporttitle
            lcall sendmessage
            mov   a,turnslog
            lcall convert

            mov   dptr,#advancesmade
            lcall sendmessage
            mov   a,howmanysofar
            lcall convert

            mov   dptr,#deviat
```

```
        lcall sendmessage
        mov   a,deverrorh
        lcall convert
        mov   a,deverrorl
        lcall convert

        mov   dptr,#dest
        lcall sendmessage
        mov   a,destheta
        lcall convert

        mov   dptr,#errt
        lcall sendmessage
        mov   a,errtheta
        lcall convert

        mov   dptr,#passornot
        lcall sendmessage
        mov   c,canpass
        lcall sendboolean

        mov   dptr,#passedornot
        lcall sendmessage
        mov   c,passedyet
        lcall sendboolean

        mov   dptr,#leftright
        lcall sendmessage
        mov   c,lor
        lcall sendboolean

        mov   dptr,#leftdetected
        lcall sendmessage
        mov   c,obstfromleft
        lcall sendboolean

        mov   dptr,#centredetected
        lcall sendmessage
        mov   c,obstfromcentre
        lcall sendboolean

        mov   dptr,#rightdetected
        lcall sendmessage
        mov   c,obstfromright
        lcall sendboolean

        ret

sendboolean
```

```
;subroutine to send a boolean variable in c to serial port
            jc    sb#true
            mov   dptr,#falsehood
            ljmp  sb#end

sb#true           mov         dptr,#truthhood

sb#end            lcall       sendmessage
            ret

falsehood         string      "FALSE",0
truthhood         string      "TRUE",0

reporttitle       string      12,13,10,"AGV STATUS REPORT",13,10,13,10
turnsmade         string      "Turns made so far:    ",0
advancesmade      string      13,10,"Advances made so far: ",0
passornot         string      13,10,13,10,"CANPASS:        ",0
deviat                        string      13,10,"Deviation from path:  ",0
dest              string      13,10,"Desired Theta:        ",0
errt              string      13,10,"Error Angle:          ",0

passedornot       string      13,10,"PASSEDYET:      ",0
leftright         string      13,10,"LOR:            ",0
leftdetected      string      13,10,"OBSTFROMLEFT:   ",0
centredetected    string      13,10,"OBSTFROMCENTRE: ",0
rightdetected     string      13,10,"OBSTFROMRIGHT:  ",0
advfail                       string      13,10,"FAILED DURING ADVANCE",0
turnfail          string      13,10,"FAILED DURING TURN",0
hellomess         string      13,10,"Now Doing Obstacle Avoidance",13,10,0
bell              string      0
heremess          string      13,10,"here",13,10,0

            #include          "agvltdev.tbl"
            #include          "agvltdes.tbl"


                        end
;File agvltdev.tbl
;Look-up table for deviation from path
;These values are approximately 256*sine(index in table*5)

ltdeverror
ltdev#1           byte        0,22,44,66,87,108,128,146,164,180
ltdev#2           byte        195,209,221,231,240,246,251,254,255

;File agvltdes
;Look-up table for desired angle values
```

```
ltdestheta
            byte  0,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  5,5,5,5,5,5,5,5
            byte  10,10,10,10,10,10,10,10
            byte  10,10,10,10,10,10,10,10
            byte  15,15,15,15,15,15,15,15
            byte  15,15,15,15,15,15,15,15
            byte  15,15,15,15,15,15,15,15
            byte  15,15,15,15,15,15,15,15
            byte  15,15,15,15,15,15,15,15
            byte  15,15,15,15,15,15,15,15
            byte  15,15,15,15,15,15,15,15
            byte  15,15,15,15,15,15,15,15
```

APPENDIX 4

Motor Control Circuit Diagrams

A4.1 DUAL MOTOR CONTROLLERS

The circuit of figure A4.1 enables the 8031 embedded obstacle avoidance computer to gain control of the experimental vehicle drive motors from the Host navigation computer. The interface between this and the INTEL 8031 computer is that the INTEL 8255 PIA (IC2) on the computer board (see figure A4.1).

When the HOST/8031 signal is low, 8 bit buffers IC1 and 3 are enabled and ICs 2 and 4 disabled. The 8031 obstacle avoidance computer has access to the HCTL-1100 motor

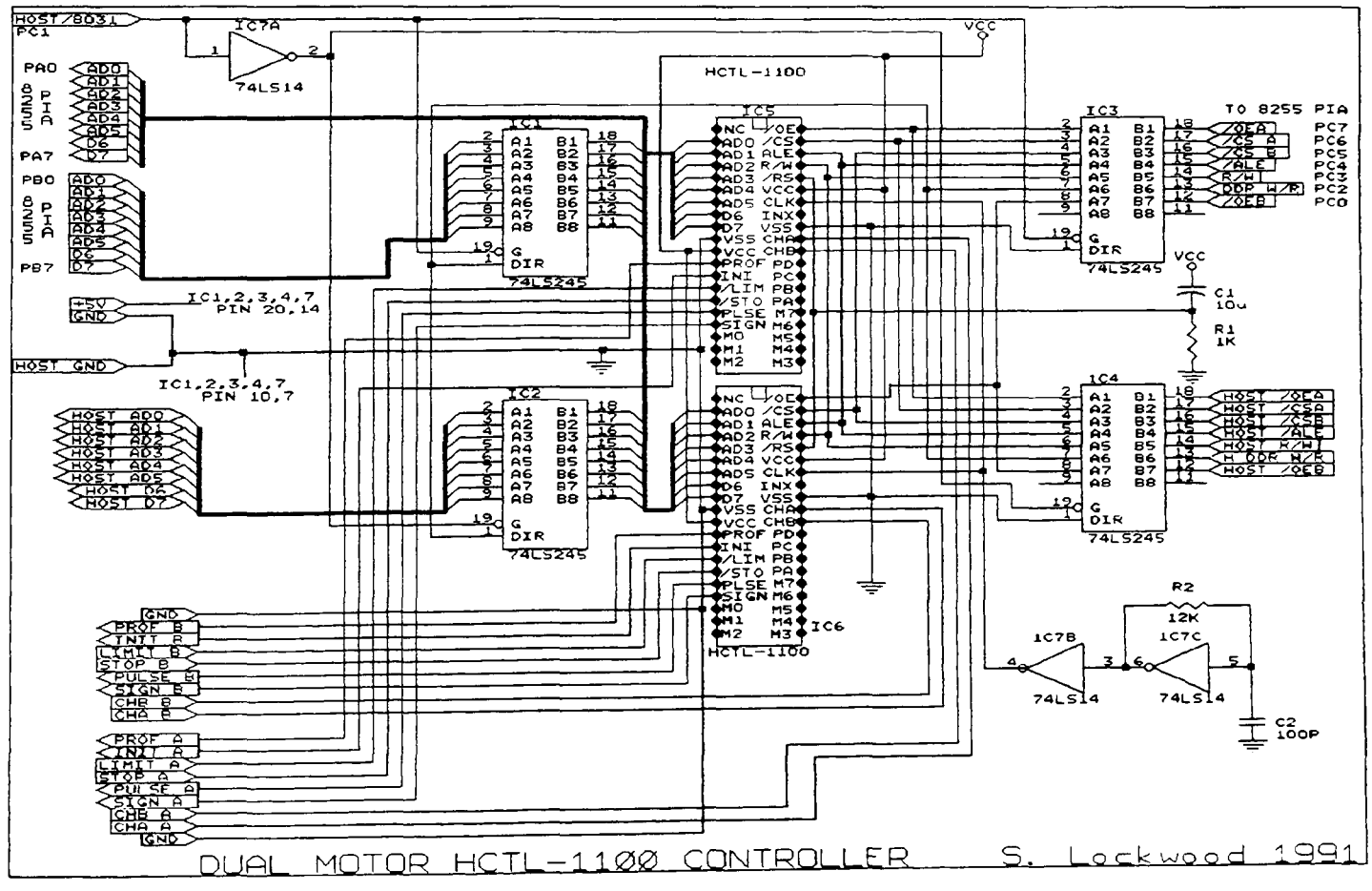


Figure A4.1
Dual Motor Controllers

controllers (IC5 and 6) via the 8255 PIA on the computer board. When the HOST/8031 signal is high, access is denied to the 8031 and the host computer has control of the motors.

The detailed operation of the HCTL-1100 motor controllers is discussed in Chapter 7 and in the specification of section A4.3. Various outputs and status flags are produced by the controllers. In this application however, only the PULSE and SIGN (PULSE A, PULSE B, SIGN A, SIGN B), signals are used to drive the H-Bridge circuit described in section A4.2.

Feedback to the HCTL-1100 motor controllers is taken directly from the experimental vehicle wheel encoders. These devices have quadrature outputs connected to the CHA A, CHA B, and CHB A, CHB B inputs of the motor controllers.

The inverters IC7B and IC7C, together with R2 and C2 form the master clock oscillator for the HCTL-1100 chips. The timing network formed by R1 and C1 ensures that the motor controllers are reset when the power supply is initially switched on.

A4.2 H-BRIDGE MOTOR DRIVE AMPLIFIER

The amplifier of figure A4.2 enables the vehicle drive motors to be driven in both directions from a unipolar power supply. This is achieved using the SIGN signal from the HCTL-1100 motor controllers.

Referring to figure A4.2, when the SIGN and PULSE signals are high, Pin 3 of IC5 is low and the transistors TR6 and TR7 are switched off. IC5 pin 6 is high and therefore current flows through the motor via transistors TR5 and TR8. When the SIGN signal is low and PULSE high however, the transistors are switched over and current flows via TR6 and TR7, reversing the direction of the motor. The HCTL-1100 motor controllers have a 'sign reversal inhibit' feature which prevents all the transistors being turned on

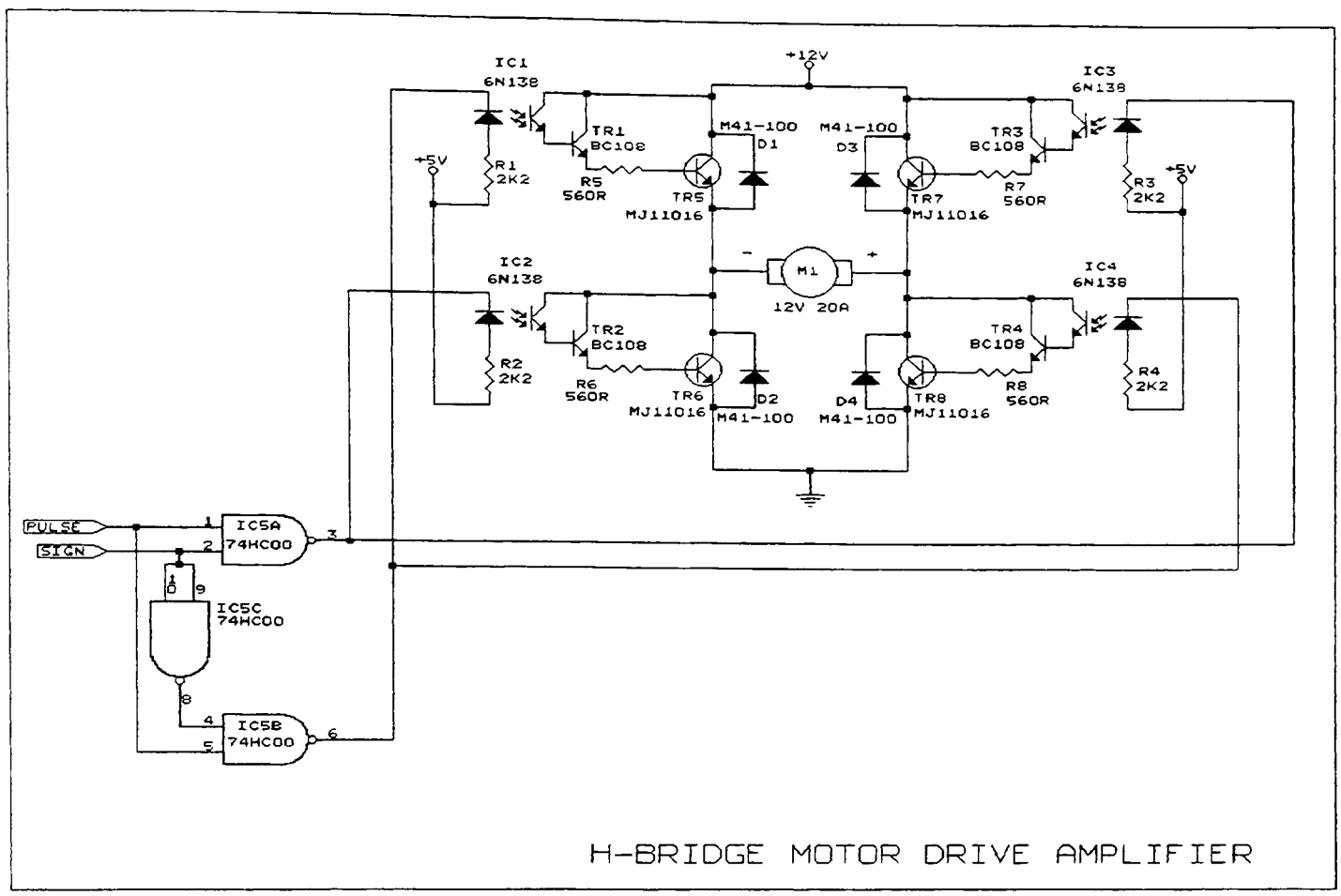


Figure A4.2
H-Bridge Drive Amplifier

at the same time and hence prevents a short circuit.

The H-Bridge amplifier is operated from a separate power supply to the computer circuits to prevent damage in the event of a failure. Opto-isolators IC1-4 are used to eliminate any physical connection between the TTL signals and the motor drive circuit.

Power transistors TR5-8 and free wheeling diodes D1-D4 are mounted on a large heat sink which covers the rear of the experimental vehicle.

APPENDIX 5

Assembler Motor Control Software Source Code

```
;Include file for Motor control routines

;*************************************************************************
;*
;*              MOTOR CONTROLLER READ AND WRITE ROUTINES
;*
;*************************************************************************


setregmc
;subroutine to configure pia registers ready for motor control
;enter with required motor controller register address in regadd

            mov  contword,#0ffh          ;set motor controller control word all 1s
            clr  host8031    ;select motor control board

            setb ddrwr       ;set data direction bit

            mov  a,contword
            mov  dptr,piac
            movx @dptr,a                 ;send to motor board

            mov  a,regadd    ;get motor controller register address
            mov  dptr,piab   ;ready to write to pia port b
            movx @dptr,a                 ;set up on pia port

            mov  dptr,piac   ;ready to write to pia port c
            clr  ale                     ;send ale low
            mov  a,contword
            movx @dptr,a                 ;send to pia port c
            setb ale                     ;set ale
            mov  a,contword
            movx @dptr,a                 ;send control word to pia port c

            ret


writemc
;subroutine to write a value to motor controller register already set up
;with setregmc
;enter with value to be written in a reg and 0 in flag 'motoraorb' for motor a
;or 1 in flag 'motoraorb' for motor B

            mov  temp,a                  ;save value to be written
            setb rw                      ;send r/w high
            mov  a,contword
            mov  dptr,piac   ;ready to write to pia port c
            movx @dptr,a                 ;write control word to pia
```

```
            mov  dptr,piab   ;ready to write to port b of pia
            mov  a,temp                 ;retreive data to be written
            movx @dptr,a                ;write to pia

            mov  dptr,piac   ;ready to write to pia port c
            jb   motoraorb,motorb1      ;if motoraorb is set then send csb low
            clr  csa                    ;else send csa low
            ljmp motora1

motorb1         clr         csb                     ;send csb low
motora1         mov         a,contword  ;get control word
                      movx  @dptr,a                 ;send to pia port c

            clr  rw                     ;send r/w low
            mov  a,contword
            movx @dptr,a                ;send control word to pia port c

            jb   motoraorb,motorb2      ;if motoraorb is set then its motor b
            setb csa                    ;else it's motor a
            ljmp motora2

motorb2         setb        csb         ;set csb
motora2         mov         a,contword  ;get control word
                      movx  @dptr,a                 ;send it to pia port c

            setb rw                     ;send r/w high
            mov  a,contword
            movx @dptr,a                ;send control word to pia port c

            ret

readmc
;subroutine to read a value from the motor controller register already
;set up using setregmc.
;if 'motoraorb' is low value is read from motor A, if it's high then
;value is read from motor B

            setb p1.0

            clr  ddrwr                  ;ready to read
            mov  a,contword
            mov  dptr,piac   ;ready to write to pia port c
            movx @dptr,a                ;write to pia port c

            jb   motoraorb,motb1        ;if motoraorb is set then access motor b
            clr  csa                    ;else must be motor a
            ljmp motal
motb1 clr       csb                     ;clear csb
```

```
mota1 mov       a,contword  ;get contword
          movx  @dptr,a                 ;write to pia port c

          setb  rw                      ;send rw high
          mov   a,contword
          movx  @dptr,a                 ;write to pia port c

          jb    motoraorb,motb2         ;if motoraorb is set then access motor b
          setb  csa                     ;else its motor a
          ljmp  mota2
motb2 setb      csb                     ;set csb
mota2 mov       a,contword
          movx  @dptr,a                 ;write control word to pia port c

          clr   rw                      ;send r/w low
          mov   a,contword
          movx  @dptr,a                 ;write control word to pia portc

          jb    motoraorb,motb3         ;if motoraorb is set then access motor b
          clr   oea                     ;else it's motor a
          ljmp  mota3
motb3 clr        oeb                    ;clera oeb
mota3 mov       a,contword
          movx  @dptr,a                 ;write to pia port c

          mov   dptr,piaa   ;ready to read from pia port a
          movx  a,@dptr                 ;get value from pia port a

          mov   temp,a                  ;save data

          mov   dptr,piac   ;ready to write to pia port c
          jb    motoraorb,motb4         ;if motoraorb is set then it's motor b
          setb  oea                     ;else it's motor a
          ljmp  mota4
motb4 setb      oeb                     ;set oeb
mota4 mov       a,contword
          movx  @dptr,a                 ;write to pia port c
          mov   a,temp                  ;leave with a = result

          clr   p1.0

          ret


;************************************************************************
;*
;*            GENERAL MOTOR CONTROLLER SET UP ROUTINES
;*
;************************************************************************
```

```
;subroutine
;clear al flags in flag register
resetflags
            mov   regadd,#00h ;address of flag register
            lcall setregmc    ;get ready

            clr   motoraorb   ;select motor a
            clr   a                       ;ready for flag0 reset
            lcall writemc                 ;reset flag0
            setb  motoraorb   ;now do it for motor b
            clr    a
            lcall writemc

            mov   regadd,#00h ;address of flag register
            lcall setregmc    ;get ready

            clr   motoraorb   ;select motor a
            mov   a,#02h                  ;ready for flag2 reset
            lcall writemc                 ;reset flag2
            setb  motoraorb   ;now do it for motor b
            mov   a,#02h
            lcall writemc

            mov   regadd,#00h ;address of flag register
            lcall setregmc    ;get ready

            clr   motoraorb   ;select motor a
            mov   a,#03h                  ;ready for flag3 reset
            lcall writemc                 ;reset flag3
            setb  motoraorb   ;now do it for motor b
            mov   a,#03h
            lcall writemc

            mov   regadd,#00h ;address of flag register
            lcall setregmc    ;get ready

            clr   motoraorb   ;select motor a
            mov   a,#04h                  ;ready for flag4 reset
            lcall writemc                 ;reset flag4
            setb  motoraorb   ;now do it for motor b
            mov   a,#04h
            lcall writemc

            mov   regadd,#00h ;address of flag register
            lcall setregmc    ;get ready

            clr   motoraorb   ;select motor a
```

```
            mov  a,#05h                 ;ready for flag5 reset
            lcall writemc               ;reset flag5
            setb motoraorb   ;now do it for motor b
            mov  a,#05h
            lcall writemc

            ret

;subroutine
;to reset and initialise motor control chips
resetmc

            mov  regadd,#05h ;address of mc to be accessed
            lcall setregmc   ;get ready

            clr  motoraorb              ;select motor a
            clr  a                      ;ready for reset
            lcall writemc               ;reset mca
            setb motoraorb              ;now do it for motor b
            clr   a
            lcall writemc

;set pwm duty cycle (write to R09h)

            mov  regadd,#09h ;address of mc to be accessed
            lcall setregmc   ;get ready

            clr  motoraorb   ;select motor a
            mov  a,#064h                ;write pwm duty cycle
            lcall writemc               ;
            setb motoraorb   ;now do it for motor b
            mov  a,#064h                ;write pwm duty cycle
            lcall writemc

;set digital filter parameters (R22h - gain)

            mov  regadd,#22h ;address of mc to be accessed
            lcall setregmc   ;get ready

            clr  motoraorb              ;select motor a
            mov  a,gain                 ;gain
            lcall writemc               ;reset mca
            setb motoraorb              ;now do it for motor b
            mov  a,gain
            lcall writemc

;R21h pole

            mov  regadd,#21h ;address of mc to be accessed
```

```
            lcall setregmc    ;get ready

            clr  motoraorb              ;select motor a
            mov  a,pole                 ;gain
            lcall writemc               ;reset mca
            setb motoraorb              ;now do it for motor b
            mov  a,pole
            lcall writemc

;R20h zero

            mov  regadd,#20h ;address of mc to be accessed
            lcall setregmc    ;get ready

            clr  motoraorb              ;select motor a
            mov  a,zero                 ;gain
            lcall writemc               ;reset mca
            setb motoraorb              ;now do it for motor b
            mov  a,zero
            lcall writemc

;set sample timer (write 20d to R0fh)

            mov  regadd,#0fh ;address of mc to be accessed
            lcall setregmc    ;get ready

            clr  motoraorb              ;select motor a
            mov  a,#020h                ;ready for writing
            lcall writemc               ;write to mca
            setb motoraorb              ;now do it for motor b
            mov  a,#020h
            lcall writemc

;set sign inhibit bit

            mov  regadd,#07h
            lcall setregmc

            clr  motoraorb
            mov  a,#1
            lcall writemc

            setb motoraorb
            mov  a,#1
            lcall writemc

            ret
```

```
;subroutine to put motor controllers in init/idle mode
initidle

            mov  regadd,#05h ;address of mc to be accessed
            lcall setregmc   ;get ready

            clr  motoraorb              ;select motor a
            mov  a,#01                  ;ready for writing
            lcall writemc               ;write to mca
            setb motoraorb              ;now do it for motor b
            mov  a,#01
            lcall writemc

            ret

hostop
;subroutine to reset motor control board for host operation

            setb host8031    ;send host/8031 high
            mov  a,contword
            mov  dptr,piac   ;ready to writr to pia port c
            movx @dptr,a                ;write to pia port c
            ret


obstavop
;subroutine to set motor control board for obstacle avoidance operation

            clr  host8031    ;send host/8031 low
            mov  a,contword
            mov  dptr,piac   ;ready to writr to pia port c
            movx @dptr,a                ;write to pia port c
            ret

;*************************************************************************
;*
;*                          PROPORTIONAL VELOCITY SUBROUTINES
;*
;*************************************************************************

;subroutine to put motorcontrollers in proportional velocity control mode
propvelcont



            acall resetflags
            acall resetmc

            mov  regadd,#05h ;address of mc to be accessed
```

```
            lcall setregmc    ;get ready

            clr  motoraorb              ;select motor a
            mov  a,#03                  ;ready for writing
            lcall writemc               ;write to mca
            setb motoraorb              ;now do it for motor b
            mov  a,#03
            lcall writemc

;set flag f3 to begin  (R00 = 0bh)

            mov  regadd,#00h ;address of mc to be accessed
            lcall setregmc    ;get ready

            clr  motoraorb              ;select motor a
            mov  a,#0bh                 ;ready for writing
            lcall writemc               ;write to mca
            setb motoraorb              ;now do it for motor b
            mov  a,#0bh
            lcall writemc

            ret

;subroutine to write the 16 bit 2's cpmplement number in cvh, cvl
;to command velocity registers r24h (high byte) R23h (low nibble+fraction)
;enter with motoraorb=0 for motor a or motoraorb=1 for motor b

commandvel

                                    mov   regadd,#24h;mc register to be accessed
                                    lcall setregmc    ;set up mc

                                    mov  a,cvh                  ;get high byte of value
                                    lcall writemc

                                    mov   regadd,#23h;mc register to be accessed
                                    lcall setregmc    ;set up mc

                                    mov  a,cvl                  ;get low byte of value
                                    lcall writemc

                                    ret

;subroutine to read the contents of the Actual velocity registers (R35h,R34h)
;and send the contents to the serial port
;enter with motoraorb=0 for motor a and motoraorb=1 for motor b
;result in cvh, cvl

readpropvel
```

```
                mov   regadd,#35h;mc register to be accessed
                lcall setregmc    ;set up mc

                lcall readmc                ;get high byte of value
                mov   cvh,a

                mov   regadd,#34h;mc register to be accessed
                lcall setregmc    ;set up mc

                lcall readmc                ;get low byte of value
                mov   cvl,a

                ret
```

```
;*************************************************************************
;*
;*                          POSITION CONTROL SUBROUTINES
;*
;*************************************************************************

;subroutine to put controllers in position control mode (R05h=03,flags clear)

positioncont

                acall resetflags
                acall resetmc

                mov   regadd,#05h ;address of mc to be accessed
                lcall setregmc    ;get ready

                clr   motoraorb             ;select motor a
                mov   a,#03                 ;ready for writing
                lcall writemc               ;write to mca
                setb  motoraorb             ;now do it for motor b
                mov   a,#03
                lcall writemc

;clear flags to begin  (R00 = 00h)

                mov   regadd,#00h ;address of mc to be accessed
                lcall setregmc    ;get ready

                clr   motoraorb             ;select motor a
                mov   a,#0h                 ;ready for writing
```

```
            lcall writemc               ;write to mca
            setb  motoraorb             ;now do it for motor b
            mov   a,#0h
            lcall writemc

            ret

;subroutine to write 24 bit 2's complement number in ph,pm,pl to
;motor controller position command registers
;if motorab=0 then write to motor a, resl=1 then write to motor b

writeposition
                                    mov    regadd,#0ch;mc register to be accessed
                                    lcall setregmc    ;set up mc

                                    mov   a,ph                  ;get high byte of value
                                    lcall writemc

                                    mov    regadd,#0dh;mc register to be accessed
                                    lcall setregmc    ;set up mc

                                    mov   a,pm                  ;get middle byte of value
                                    lcall writemc

                                    mov    regadd,#0eh;mc register to be accessed
                                    lcall setregmc    ;set up mc

                                    mov   a,pl                  ;get low byte of value
                                    lcall writemc

                                    ret

;subroutine to read the actual position of the motors and put the result in
;apl,apm,aph registers (R12 R13, R14)
;enter with motoraorb=0 for motor a and motoraorb=1 for motor b

readposition
                        jnb   motoraorb,rdmota      ;if motorab is clear then message a
                        mov   dptr,#posb  ;else write message b to serial port
                        ljmp  writeposmess
rdmota
writeposmess
                        mov   regadd,#12h ;prepare to read registers
                        lcall setregmc

                        lcall readmc

                        mov   aph,a                 ;transfer value to holding register
```

```
                        mov   regadd,#13h ;prepare to read registers
                        lcall setregmc

                        lcall readmc
                        mov   apm,a

                        mov   regadd,#14h ;prepare to read registers
                        lcall setregmc

                        lcall readmc
                        mov   apl,a

                        ret

;subroutine to reset motor actual position registers
;if motorab=0 then write to motor a, resl=1 then write to motor b

resetposition
                                mov    regadd,#13h;mc register to be accessed
                                lcall setregmc    ;set up mc

                                mov   a,#0                    ;write zero to register
                                lcall writemc

                                ret

;*********************************************************************
;*
;*                          INTEGRAL VELOCITY CONTROL SUBROUTINES
;*
;*********************************************************************


;subroutine to put motor controllers in integral velocity control mode

intvelcont

            acall resetflags
            acall resetmc

            mov   regadd,#05h ;address of mc to be accessed
            lcall setregmc    ;get ready

            clr   motoraorb   ;select motor a
            mov   a,#03                   ;ready for writing
            lcall writemc                 ;write to mca
            setb  motoraorb   ;now do it for motor b
            mov   a,#03
```

```
            lcall writemc

;set flag f5 to begin  (R00 = 0dh)

            mov  regadd,#00h ;address of mc to be accessed
            lcall setregmc   ;get ready

            clr  motoraorb              ;select motor a
            mov  a,#0dh                 ;ready for writing
            lcall writemc               ;write to mca
            setb motoraorb              ;now do it for motor b
            mov  a,#0dh
            lcall writemc

            ret



;subroutine to write an acceleration (R26h lsb, R27h msb)
;enter with values to be written in cvh, cvl
;motoraorb=0 - motor a, motoraorb=1 motor b

writeaccel
                                    mov   regadd,#27h;mc register to be accessed
                                    lcall setregmc   ;set up mc
                                    mov  a,cvh                  ;get high byte of value
                                    lcall writemc

                                    mov   regadd,#26h;mc register to be accessed
                                    lcall setregmc   ;set up mc

                                    mov  a,cvl                  ;get low byte of value
                                    lcall writemc

                                    ret


;subroutine to write the 8 bit 2's complement number in a to the integral
;command velocity register (R3Ch)
;enter with motoraorb=0 for motora and motoraorb=1 for motor b

intvel
                        mov  cvl,a
                        mov  regadd,#3ch ;get addres to be accessed
                        lcall setregmc   ;set up mc

                        mov  a,cvl
                        lcall writemc               ;write value in a register to mc
```

```
                    ret

;subroutine to read velocity command register
;if motoraorb=0 then read motor a motoraorb=1 read motor b
;exit with result in a

readintvel
                    mov  regadd,#3ch ;get addres to be accessed
                    lcall setregmc   ;set up mc

                    lcall readmc                ;read value from mc

;                lcall      convert             ;send it down serial line

                    ret

;*************************************************************************
;*
;*                         TRAPEZOIDAL CONTROL ROUTINES
;*
;*************************************************************************

;subroutine
;put motors into trapezoidal control mode

trapezcont
          acall resetflags  ;clear flag register
          acall resetmc

;put motors in position control mode

          mov  regadd,#05h ;address of mc to be accessed
          lcall setregmc   ;get ready

          clr  motoraorb             ;select motor a
          mov  a,#03                 ;ready for writing
          lcall writemc              ;write to mca
          setb motoraorb             ;now do it for motor b
          mov  a,#03
          lcall writemc

;write acceleration to motor controllers

          clr  motoraorb             ;first right motor
          mov  cvh,#00h                        ;acceleration high byte
          mov  cvl,#01h              ;acceleration low byte
          acall writeaccel           ;write values to motor controller
```

```
            setb  motoraorb             ;now do left motor
            mov   cvh,#00h              ;acceleration high byte
            mov   cvl,#01h              ;acceleration low byte
            acall writeaccel            ;write values to motor controller

;write maximum velocity

            clr   motoraorb             ;first do right motor
            mov   a,#01h                        ;command velocity
            acall tzintvel              ;write velocity to motor controller

            setb  motoraorb             ;now do left motor
            mov   a,#01h                        ;command velocity
            acall tzintvel              ;write velocity to motor controller

;motors should now repond to position commands written to command
;position registers using writeposition

;set flag f0 to begin  (R00 = 08h)

            mov   regadd,#00h ;address of mc to be accessed
            lcall setregmc    ;get ready

            clr   motoraorb             ;select motor a
            mov   a,#08h                        ;ready for writing
            lcall writemc                       ;write to mca
            setb  motoraorb             ;now do it for motor b
            mov   a,#08h
            lcall writemc

            ret

;subroutine to write 24 bit 2's complement number in ph,pm,pl to
;motor trapezoidal controller position command registers R2Bh,R2Ah,R29h
;if motorab=0 then write to motor a, res1=1 then write to motor b

tzwriteposition
                        mov   regadd,#2bh;mc register to be accessed
                        lcall setregmc    ;set up mc

                        mov   a,ph                  ;get high byte of value
                        lcall writemc

                        mov   regadd,#2ah;mc register to be accessed
                        lcall setregmc    ;set up mc

                        mov   a,pm                  ;get middle byte of value
                        lcall writemc
```

```
                                mov    regadd,#29h;mc register to be accessed
                                lcall setregmc    ;set up mc

                                mov  a,pl                    ;get low byte of value
                                lcall writemc

                                ret

;subroutine to write the 8 bit 2's complement number in a to the integral
;trapezoidal command velocity register (R28h)
;enter with motoraorb=0 for motora and motoraorb=1 for motor b

tzintvel
                        mov  cvl,a
                        mov  regadd,#28h ;get addres to be accessed
                        lcall setregmc    ;set up mc

                        mov  a,cvl
                        lcall writemc               ;write value in a register to mc

                        ret
```

APPENDIX 6

Pascal Obstacle Avoidance Simulation Source Code

```
program agvmodel;
uses crt,graph;
const
     roadleft=100;                                    {Constant Definitions}
     roadright = 400;                                 {left hand edge of roadway}
     obstaclecolor = lightgreen;                      {right hand edge of road way}
     colorin=true;                                    {colour of obstacles}
in}                                                   {Boolean constant to indicate colour fill-


     check = true;                                    {Indicates check for obstacles}
     nocheck = false;                                 {indicates    don't    check    for
obstacles}
     turnsincrement = 5;                              {5 degree turn increment}
     advanceincrement = 5;                            {5 unit advance increment}
     enough = 15;                                     {number  of advances  required  to
clear an obstacle}
     failed=false;                                    {indicates failure of a process}
type
    arr = array[0..50] of integer;  {define array type}

var                                                   {Variable definitions}
   n,i:integer;
   Gd, Gm : Integer;
   obstdetected,obstfromleft,obstfromright,obstfromcentre:boolean;
   oldx,oldy:integer;
   oldagvx,oldagvy,oldheading:integer;
   head:integer;
   x,y:integer;
   exitt:boolean;
   cnt:word;
   ystart:word;
   xx:array[1..4,0..20] of integer;
   yy:array[1..4,0..20] of integer;
   noofobstacles:integer;
   failedtopass:boolean;
   passedyet:boolean;
   howmanysofar:word;
   turnswithoutadvance:word;
   startdev,startheading:integer;
   ltdestheta:array[0..100] of integer;
   ltdeverror:array[0..20] of integer;
   t,deverror:integer;
   destheta,errtheta:integer;

function IntToStr(i: Longint): string;
  { Convert any Integer type to a string }
  var
    s: string[11];
```

```
 begin
   Str(i, s);
   IntToStr := s;
 end;                {inttostring}

function direction(angle:integer):real;
{converts an angle in degrees to a direction in radians}
begin
    angle:=angle-90;
    direction:=(angle*pi)/180 ;
end;

procedure detectcollision(x6,y6,x7,y7:integer);
{detects when AGV light pattern has touched an object on front edge}
const
    n=30;
var
    i,x,y:integer;
    xi,yi:real;
begin
obstdetected:=false;
obstfromcentre:=false;
obstfromleft:=false;
obstfromright:=false;
if not obstdetected then
  begin
    xi:=(x7-x6)/n;
    yi:=(y7-y6)/n;
    for i:=0 to n do
        begin
            x:=round(xi*i+x6);
            y:=round(yi*i+y6);

            if getpixel(x,y)=obstaclecolor then obstdetected:=true;
            if (i<10) and (getpixel(x,y)=obstaclecolor) then
                  obstfromright:=true;
            if (i>=10) and (i<=20) and (getpixel(x,y)=obstaclecolor) then
                  obstfromcentre:=true;
            if (i>20) and (getpixel(x,y)=obstaclecolor) then
                  obstfromleft:=true;
        end;
   end;
end;

procedure detectcollisionleft(x6,y6,x7,y7:integer);
{detects when AGV light pattern has touched an object on left hand edge}
const
    n=30;
```

```
var
     i,x,y:integer;
     xi,yi:real;
begin

     xi:=(x7-x6)/n;
     yi:=(y7-y6)/n;
     for i:=0 to n do
         begin
              x:=round(xi*i+x6);
              y:=round(yi*i+y6);

              if getpixel(x,y)=obstaclecolor then
                begin
                     obstfromleft:=true;
                     obstdetected:=true;
                end;
         end;
end;

procedure detectcollisionright(x6,y6,x7,y7:integer);
{detects when AGV light pattern has touched an object on right hand edge}

const
     n=30;
var
     i,x,y:integer;
     xi,yi:real;
begin
     xi:=(x7-x6)/n;
     yi:=(y7-y6)/n;
     for i:=0 to n do
         begin
              x:=round(xi*i+x6);
              y:=round(yi*i+y6);

              if getpixel(x,y)=obstaclecolor then
                begin
                     obstfromright:=true;
                     obstdetected:=true;
                end;
         end;
end;


procedure drawroad(colorin:boolean);
{draws road on screen}
var
   oldcolor:word;
```

```
   linest:linesettingstype;
begin
oldcolor:=getcolor;
setcolor(green);
setfillstyle(ltslashfill,getcolor);
setcolor(obstaclecolor);
moveto(0,roadright);
lineto(639,roadright);
lineto(639,479);
lineto(0,479);
lineto(0,roadright);
moveto(0,roadleft);
lineto(639,roadleft);
lineto(639,50);
lineto(0,50);
lineto(0,roadleft);
if colorin then floodfill(400,450,obstaclecolor);
if colorin then floodfill(400,80,obstaclecolor);
setcolor(oldcolor);
setcolor(oldcolor);


end;

procedure updateobst(noofobstacles:integer);
{redraw obstacles on screen}
var
   i:integer;
   oldcolor:word;
   oldfillpattern:fillpatterntype;
begin
     oldcolor:=getcolor;
     getfillpattern(oldfillpattern);
      setcolor(obstaclecolor);
     setfillstyle(solidfill,obstaclecolor);

     for i:=0 to noofobstacles do
       begin
            moveto(xx[1,i],yy[1,i]);
            lineto(xx[2,i],yy[2,i]);
            lineto(xx[3,i],yy[3,i]);
            lineto(xx[4,i],yy[4,i]);
            lineto(xx[1,i],yy[1,i]);
            floodfill(round((xx[3,i]-xx[1,i])/2+xx[1,i]),round((yy[3,i]-
yy[1,i])/2+yy[1,i]),getcolor);
        end;
            setfillpattern(oldfillpattern,oldcolor);
            setcolor(oldcolor);
end;
```

```
procedure drawagv(x,y:integer;angle:integer;e:boolean);
{Draws AGV on screen}
const
     pi=3.142;
     l = 36.06;
     r: array[0..7] of real = (1,1,1,1,53.85,53.85,72.8,72.8);
     theta: array[0..7] of real = (4.124,5.3,0.983,2.159,1.95,1.19,1.292,1.85);
var
   xp:array[0..7] of integer;
   yp:array[0..7] of integer;
   i,colonentry:integer;
   heading:real;

begin

     outtextxy(10,70,'Deviation: '+inttostr(deverror));
     outtextxy(10,80,'Head err : '+inttostr(errtheta));
     outtextxy(10,60,'DesTheta : '+inttostr(round(destheta)));
     outtextxy(10,90,'turns log : '+inttostr(t));
     if obstdetected then outtextxy(50,30,'Obstacle Detected');

     y:=(getmaxy - y);
     colonentry:=getcolor;

     if e then setcolor(getbkcolor) else setcolor(yellow);
heading:=direction(angle);
xp[0]:=round(x+(r[0]*cos(theta[0]+heading)));
yp[0]:=round(y-(r[0]*sin(theta[0]+heading)));
for i:=0 to 3 do
  begin
    xp[i]:=round(x+(r[i]*cos(theta[i]+heading)));
    yp[i]:=round(y-(r[i]*sin(theta[i]+heading)));
  end;
  xp[4]:=round(x+(r[4]*cos(theta[4]+heading)));
  yp[4]:=round(y-(r[4]*sin(theta[4]+heading)));


for i:=5 to 7 do
  begin
    xp[i]:=round(x+(r[i]*cos(theta[i]+heading)));
    yp[i]:=round(y-(r[i]*sin(theta[i]+heading)));
  end;

detectcollision(xp[6],yp[6],xp[7],yp[7]);
detectcollisionleft(xp[4],yp[4],xp[7],yp[7]);
detectcollisionright(xp[5],yp[5],xp[6],yp[6]);
```

```
     moveto(xp[0],yp[0]);
     for i:=1 to 3 do
     lineto(xp[i],yp[i]);
     lineto(xp[0],yp[0]);
     moveto(xp[4],yp[4]);

     if not e then setcolor(lightred);
     for i:=5 to 7 do
     lineto(xp[i],yp[i]);
     lineto(xp[4],yp[4]);
     setcolor(getbkcolor);
     outtextxy(50,30,'Obstacle Detected');
     outtextxy(10,70,'Deviation: '+inttostr(deverror));
     outtextxy(10,80,'Head err : '+inttostr(errtheta));
     outtextxy(10,60,'DesTheta : '+inttostr(round(destheta)));
     outtextxy(10,90,'turns log : '+inttostr(t));
     setcolor(colonentry);
if obstdetected then begin
   drawroad(not colorin);
   updateobst(noofobstacles);
end;

end;

procedure placeobstacle(x,y:integer);
{Defines a random obstacle}
const
     size = 5;

var  x1,y1,x2,y2,x3,y3,x4,y4:integer;
     oldfillpattern:fillpatterntype;
     oldcolor:word;
     i:integer;

     polygon:array[0..3] of pointtype;

begin
     noofobstacles:=noofobstacles+1;
     randomize;
     x1:=x-random(size)-10; if (x1<1) then x1:=1; if (x1>639) then x1:=639;
     y1:=y-random(size)-10;   if  (y1<roadleft)  then  y1:=roadleft;  if   (y1>roadright)   then
y1:=roadright;
     x2:=x+random(size)+10; if (x2<1) then x2:=1; if (x2>639) then x2:=639;
     y2:=y-random(size)-10;   if  (y2<roadleft)  then  y2:=roadleft;  if   (y2>roadright)   then
y2:=roadright;
     x3:=x+random(size)+10; if (x3<1) then x3:=1; if (x3>639) then x3:=639;
     y3:=y+random(size)+10;   if  (y3<roadleft)  then  y3:=roadleft;  if   (y3>roadright)   then
y3:=roadright;
     x4:=x-random(size)-10; if (x4<1) then x4:=1; if (x4>639) then x4:=639;
```

```
    y4:=y+random(size)+10;   if  (y4<roadleft)  then  y4:=roadleft;  if   (y4>roadright)   then
y4:=roadright;
      xx[1,noofobstacles]:=x1;
      xx[2,noofobstacles]:=x2;
      xx[3,noofobstacles]:=x3;
      xx[4,noofobstacles]:=x4;
      yy[1,noofobstacles]:=y1;
      yy[2,noofobstacles]:=y2;
      yy[3,noofobstacles]:=y3;
      yy[4,noofobstacles]:=y4;
      updateobst(noofobstacles);
end;


procedure drawcursor(x,y:integer;visible:boolean);
{Draws cursor used for placing obstacles}
var
   oldcolor:word;
begin
     if x>633 then x:=633;
     if x<6 then x:=6;
     if y<(roadleft+6) then y:=roadleft+6;
     if y>(roadright-6) then y:=roadright-6;
     oldcolor:=getcolor;
     setcolor(getpixel(oldx,oldy));
     moveto(oldx-5,oldy-5);
     lineto(oldx+5,oldy-5);
     lineto(oldx+5,oldy+5);
     lineto(oldx-5,oldy+5);
     lineto(oldx-5,oldy-5);
     if visible then setcolor(lightblue);
     moveto(x-5,y-5);
     lineto(x+5,y-5);
     lineto(x+5,y+5);
     lineto(x-5,y+5);
     lineto(x-5,y-5);
     oldx:=x;
     oldy:=y;
     setcolor(oldcolor);
end;

procedure positionobstacle;
{Allow cursor to be moved and obstacles to be placed}
var
   x,y,i,xp,yp:integer;
   exitt:boolean;
   ch:char;
   tempcolor:word;
   finished:boolean;
```

```
begin
     x:=(getmaxx div 2);
     y:=(getmaxy div 2);
     outtextxy(10,10,'Position Cursor Using <ARROW> Keys and Press <RETURN>');
     finished:=false;
 repeat
     exitt:=false;
   repeat
     drawcursor(x,y,true);

     if keypressed then
       begin
            ch:=readkey;
            case ch of
              chr($48):y:=y-10;
              chr($50):y:=y+10;
              chr($4B):x:=x-10;
              chr($4D):x:=x+10;
              chr(13):exitt:=true;
              chr(27):finished:=true;
            end; {case}
       end; {if}
      until (exitt) or (finished) ;
      if not finished then
       begin
            placeobstacle(x,y);
            x:=x+50;y:=y+50;
        end;
 until finished;
       tempcolor:=getcolor;
       setcolor(getbkcolor);
       outtextxy(10,10,'Position Cursor Using <ARROW> Keys and Press <RETURN>');
       drawcursor(x,y,false);
       setcolor(tempcolor);


 end;

 procedure putagv(x,y,heading:integer);
{Puts AGV on screen}
 const
      erase=true;
 begin

      drawagv(oldagvx,oldagvy,oldheading,erase);
      drawagv(x,y,heading,not erase);

      oldagvx:=x;
      oldagvy:=y;
```

```
     oldheading:=heading;

end;

procedure advance(advancestep:word;check:boolean);
{advance agv 1 step at current heading}
var
   heading:integer;
   x,y:integer;
begin

     heading:=round(direction(oldheading+90));
     x:=oldagvx+round(advancestep*cos(heading));
       if t>=0 then deverror:=deverror+ltdeverror[abs(t)]
              else deverror:=deverror-ltdeverror[abs(t)];
      y:=getmaxy-roadleft-150+deverror;
     putagv(x,y,oldheading);
     if  not check then obstdetected:=false;
end;

procedure turn(angleincrement:integer);
{if dir=true turn to right else turn to left}
var
   heading:integer;
begin
heading:=oldheading+angleincrement;
t:=t+(angleincrement div turnsincrement);
putagv(oldagvx,oldagvy,heading);
end;


procedure readlookuptables;
{read look up tables}
type
    fil =text;
var
   i:integer;
   f:fil;
begin
     assign(f,'agvltdes.dta');
     reset(f);
     for i:=0 to 99 do
       begin
            read(f,ltdestheta[i]);
       end;
     close(f);
     assign(f,'agvltdev.dta');
     reset(f);
     for i:=0 to 19 do
```

```
      begin
           read(f,ltdeverror[i]);
           ltdeverror[i]:=ltdeverror[i] div 2;
      end;
    close(f);
end;

procedure initialize;
{Initialise graphics and variables etc}
begin
   oldx:=0;oldy:=0;
   exitt:=false;
   cnt:=0;
   detectgraph(gd,gm);
   gd:=0;gm:=1;
   InitGraph(Gd, Gm, '');
   if GraphResult <> grOk then
     begin
          writeln('get lost');
          Halt(1);
      end;


   setcolor(yellow);
   head:=0;
   noofobstacles:=0;
   failedtopass:=false;
   turnswithoutadvance:=0;
   startdev:=0;
   startheading:=0;
   x:=30;
   y:=getmaxy-roadleft-150;
   ystart:=y;
   destheta:=0;
   for i:=1 to 4 do
     begin
          xx[i,0]:=0;
          yy[i,0]:=0;
     end;
     setcolor(yellow);
   oldagvx:=x;oldagvy:=y;oldheading:=head;
   putagv(x,y,head);
   drawroad(colorin);
   positionobstacle;
   readlookuptables;
   t:=0;
   deverror:=0;
end;
```

```
procedure plottext(x,y:word;txt:string);
{Draws messages on screen}
var
   oldcolor:word;
begin
     oldcolor:=getcolor;
     setcolor(lightred);
     outtextxy(x,y,txt);
     repeat until keypressed;
     setcolor(getbkcolor);
     outtextxy(x,y,txt);
     setcolor(oldcolor);
end;



function waytoavoid(oleft,ocentre,oright:boolean):integer;
{Work out which direction to turn to avoid an obstacle}
const
     left=-1;
     right = 1;
var
   leftorright:integer;

begin
     if oright then leftorright:=left else leftorright:=right;
     waytoavoid:=leftorright;
end;

function testturn(detected:boolean;lor:integer):boolean;
const
     left=-1;
     right = 1;
var
   result:boolean;
begin


     if ((obstfromleft) and (not obstfromright) and (lor=left)) or
        ((obstfromright) and (not obstfromleft) and (lor=right)) then result:=failed
     else result:=(not failed);
     testturn:=result;
end;



procedure turnagv(lor:integer;var result:boolean);
{returns true if agv failed to pass obstacle}
var
```

```
   i:word;
begin
     turnswithoutadvance:=turnswithoutadvance+1;
     result:=not failed;
     while (obstdetected) and (result = not failed) do
       begin
           turn(-lor*turnsincrement);
           result:=testturn(obstdetected,lor);
       end; {while}
       if result= not failed then for i:=0 to 1 do turn(-lor*turnsincrement);

end; {turnagv}

function enoughadvances(var howmanysofar:word):boolean;
{see if AGV has advanced far enough to recover yet}
const
     passed = true;
var
   result:boolean;

begin
     if howmanysofar>=enough then result:=passed
       else
         begin
              result:=not passed;
              howmanysofar:=howmanysofar+1;
         end;
     enoughadvances:=result;
end;

procedure advanceagv(var passedyet,result:boolean);
{Advance AGV past obstacle}
const
     passed=true;
     maxdeverror=60;

begin

               result:=not failed;
               howmanysofar:=0;
               passedyet:=not passed;
               while (not obstdetected) and (not passedyet) and (result=not failed) do
                 begin
                      advance(advanceincrement,check);
                      if enoughadvances(howmanysofar) then passedyet:=passed
                        else passedyet:=not passed;
                      if abs(deverror)>maxdeverror then result:=failed;
                  end; {while}
               if (obstdetected) and (not passedyet) and (howmanysofar>3) then result:=failed;
```

```
end;

procedure recover(startdev,startheading,leftorright:integer);
{recover AGV back onto original path}
const
     k = 0.1;
     allowable=2;
var
   deviation:integer;
begin
     while (not obstdetected) and ((abs(deverror)>allowable) or (t*turnsincrement<>0)) do
     begin
          destheta:=ltdestheta[abs(deverror)];
          if deverror>=0 then destheta:=-destheta;
          errtheta:=leftorright*((destheta)-t*turnsincrement);
          if errtheta>= turnsincrement then turn(leftorright*turnsincrement)
             else turn(-leftorright*turnsincrement);
          advance(advanceincrement,check);
          if (abs(deverror)<35) and (abs(deverror)>15) then obstdetected:=false;
     end;

end;

procedure avoidobstacle(var failedornot:boolean);
{obstacle avoidance procedure}
var
   anotherobstacle:boolean;
   i:integer;

   leftorright:integer;
   canpass:boolean;

begin
       deverror:=0;
       t:=0;

       passedyet:=false;
       canpass:=true;
       repeat
            leftorright:=waytoavoid(obstfromleft,obstfromcentre,obstfromright);
            if (canpass) then
              begin
                   turnagv(leftorright,canpass);
                   passedyet:=false;
              end;
            if not canpass then
              plottext(10,30,'Failed during TURN');
```

```
            if (not passedyet) and (canpass) then advanceagv(passedyet,canpass);
            if     not     canpass     then     plottext(10,30,'Failed     During      ADVANCE:
'+inttostr(howmanysofar));

            if (not obstdetected) and (passedyet)
            then recover(startdev,startheading,leftorright);

            if not canpass then failedornot:=true;


       until ((not obstdetected) and (passedyet)) or (failedornot=true);
end;



begin {program}
repeat

 initialize;

repeat

     advance(5,true);
     if obstdetected then
       begin
            startdev:=oldagvy;
            oldheading:=oldheading;
            avoidobstacle(failedtopass);
        end;
     if oldagvx>500 then exitt:=true;
until (exitt) or (failedtopass) ;
repeat until keypressed;
until false;

{    CloseGraph;}
  end.
```

```
{File AGVLTDES.DTA}
{used to look-up required heading angle for current deviation}

0 5 5 10 10 15 15 15 20 20 20 25 25 25 25 25 25 30 30 30 30 30
30 30 30 30 30 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 40 40 40 40 40 40 40 40 40 40 40 40
40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40


{File AGVLTDEV.DTA}
{used to look up deviation for a given heading}

0 1 2 3 3 4 5 6 6 7 8
8 9 9 9 10 10 10 10
```

DESIGN OF AN OBSTACLE AVOIDANCE SYSTEM FOR AUTOMATED GUIDED VEHICLES

S. Lockwood, B. Merhdadi, J. R. Chandler

Control and Power Group, School of Engineering

The Polytechnic of Huddersfield, Queensgate, Huddersfield, HD1 3DH, U.K.

This paper describes a modular AGV obstacle avoidance system which operates auxiliary to the primary guidance systems with a minimum of modification. The system uses a CCD image sensor to detect the distortion caused by obstacles to a coded light pattern projected onto the floor ahead of the vehicle. The light encoding scheme and image processing methods employed allow an embedded controller to be incorporated in a compact and low cost design.

INTRODUCTION

Automated Guided Vehicles (AGVs) have emerged as a key element of materials transport in many modern flexible manufacturing systems. They offer many benefits over continuous conveyor or manual transport systems[1]. The most common method of AGV guidance used in such systems is an inductive buried wire scheme[2] where AGV routes are marked by embedding large signal carrying loops a few centimetres beneath the floor surface. Inductive sensing heads on board AGVs detect the presence and position of the wires. Buried wire guidance has a proven record of reliability and is relatively straight forward to operate and maintain.

However, the uninterrupted flow of materials is of paramount importance to the performance of manufacturing systems and a major drawback of wire guidance and many other systems is that disruptions to materials transport can occur. This is often caused by a lack of suitable mechanisms on AGVs to avoid unexpected obstacles which have been placed in the guide path.

Research work aimed at solving the problem of obstacle avoidance has included the use of scanning LASER range finders[3,4] and ultrasonic scanners[5,6,7]. However, these systems employ sensitive moving parts which are not generally suitable for use in harsh industrial environments. A vision system has been described[8] which uses a powerful lamp to illuminate the floor ahead of the AGV and a CCD video camera to detect the edges of obstacles as they are approached. A disadvantage of this system however is that it is susceptible to false detection due to the reflection of overhead lights and flat objects or markings on the floor.

This paper describes a modular obstacle avoidance vision system which

overcomes the problems of false detection described above. The system is intended to operate auxiliary to the primary guidance system with a minimum of modification. During normal operation, the obstacle avoidance system is transparent to the primary AGV guidance system. When an obstacle is encountered however, the system takes temporary control of the vehicle drives, circumnavigates the obstacle (if possible) and returns the vehicle to the primary guide path.

OPERATION OF THE OBSTACLE AVOIDANCE SYSTEM

The obstacle avoidance system projects a sharp coded illumination pattern onto the floor ahead of the AGV and uses a video camera to detect distortion to this pattern caused by obstacles in the path of the vehicle. The way in which the pattern is distorted depends on the position of the projector with respect to the point of view. In this system the distortion is confined to the vertical plane by positioning the projector vertically above the camera as shown in figure 1. When no objects are near enough to interfere with the projected light pattern, the code is not detected by the camera. However, when the AGV nears an obstacle, the projected light pattern is disturbed and the code is visible to the camera. Figures 2a-2c show a sequence of images as an obstacle is approached. It can be clearly seen from figure 2 that the coded light pattern 'grows' from the bottom of the image confirming that the distortion occurs in the vertical direction only.

The illumination code is a series of stripes similar to a bar code. This type of code is insensitive in the vertical direction but provides information about the horizontal position of obstacles.

The use of spatially coded light patterns is a well established technique in the analysis of images[9], but is usually restricted to static image processing applications due to the large amount of data processing involved. The novel geometry of the obstacle avoidance system together with a bar type illumination code allows the image processing task to be greatly simplified. This is because

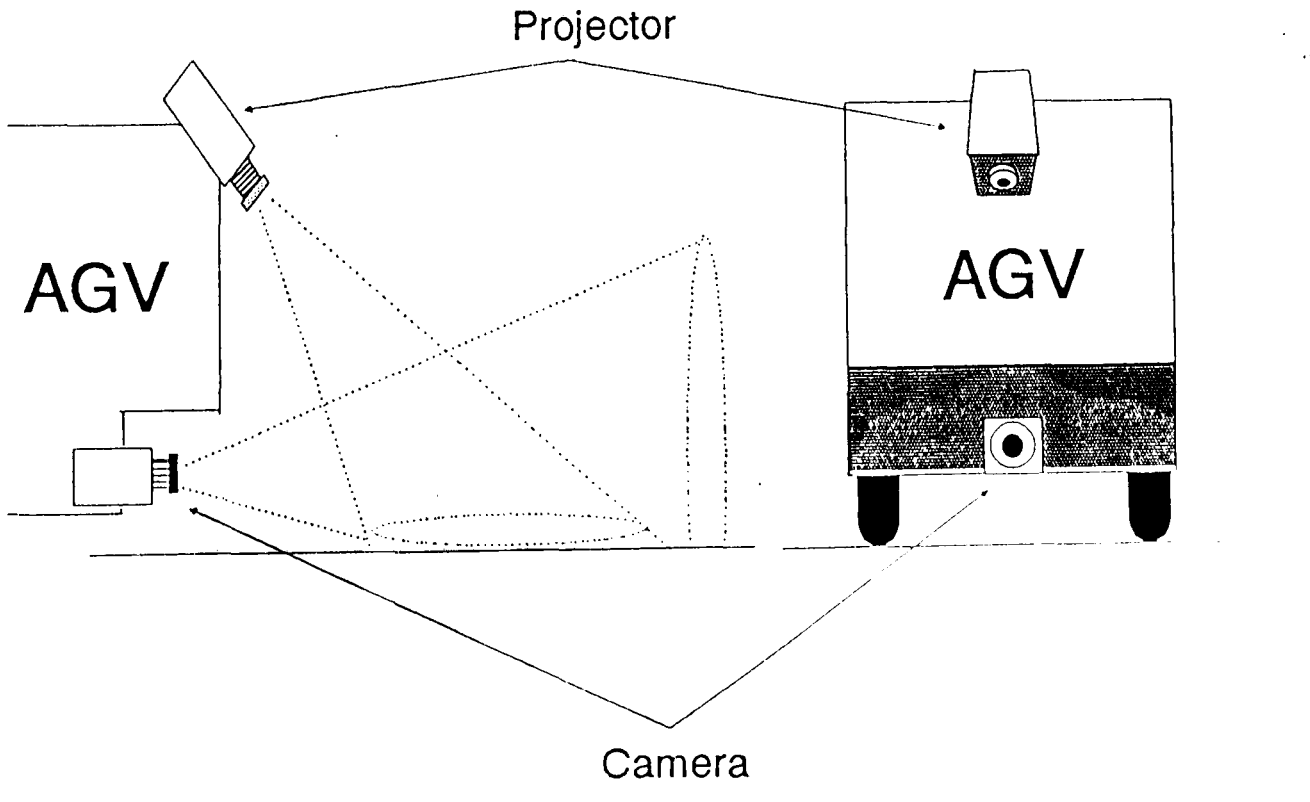
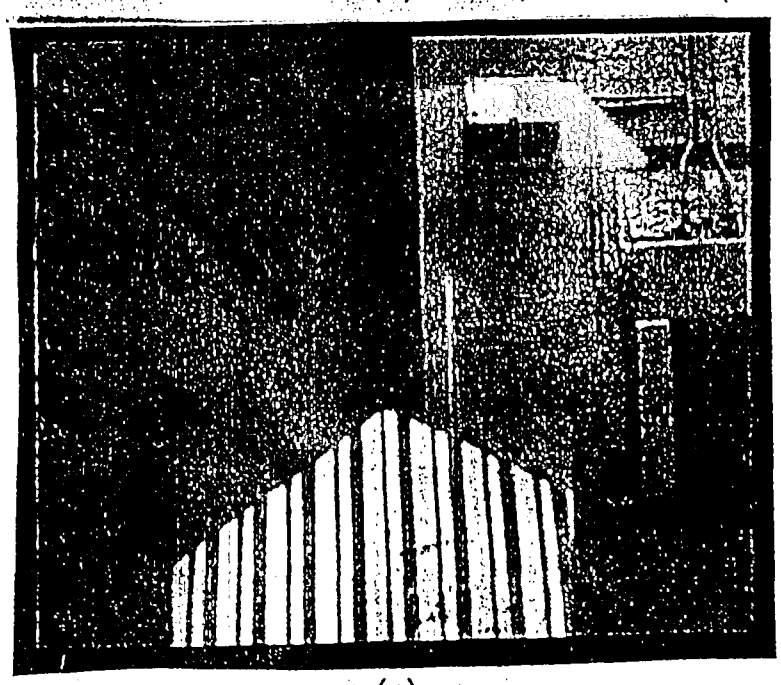


Figure 1 : Relative Position of Projector and Camera

(a)

(b)



(c)

Figure 2 - Sequence of Images as an Obstacle is Approached

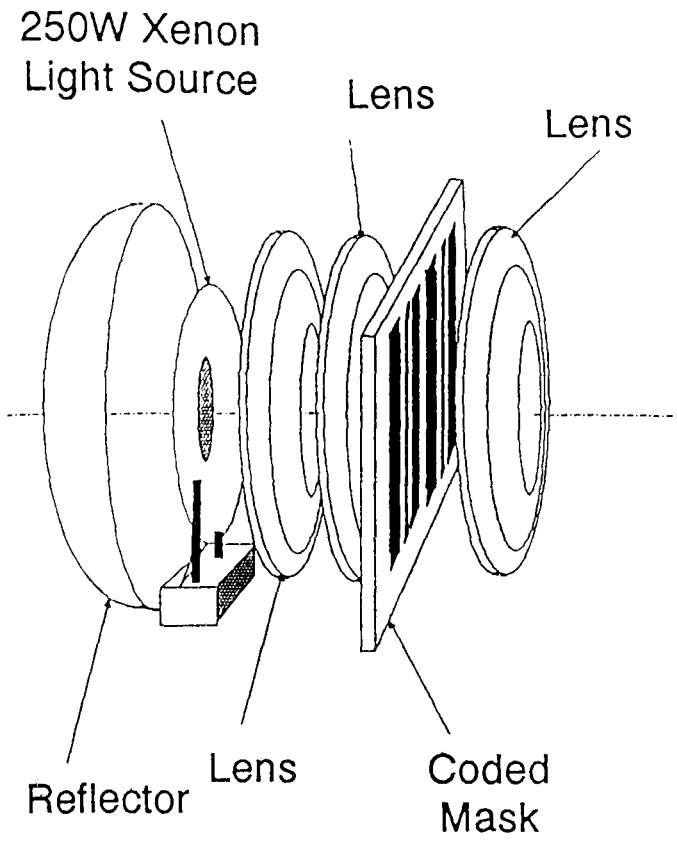


Figure 3 - Coded Light Projector

sufficient information for obstacle avoidance can be obtained from a horizontal band across the lower portion of the image.

DESCRIPTION OF THE OBSTACLE AVOIDANCE SYSTEM

The coded illumination pattern is recorded on a 24mm X 36mm transparency and projected onto the floor with the slide projector arrangement shown in figure 3.

The sensing element in the obstacle avoidance system is a monochrome CCD video camera. A high sensitivity CCD array allows the camera to operate in near darkness, whilst an auto-iris device prevents saturation in bright light.

A video frame store has been developed to capture digitised images in real-time (figure 4). Captured images are stored as a 256 X 256 array of picture elements with grey levels from 0 to 255.

Since obstacle information is obtained from a horizontal band in the lower portion of the image, an embedded controller can be used for image processing. This results in a very compact, low cost design.

The video frame store memory is directly accessible to the embedded controller by making it appear as the controller main data memory.

IMAGE PROCESSING SOFTWARE

Since the dimensions of the projected light code are precisely known and remain constant (fixed by the projector-camera relationship), filters can be designed with optimal responses for extracting code information from images. Considering the coding scheme shown in figure 5, T_l and T_h represent the longest and shortest periods of the code. Since the sampling rate is known (the time between video line syncs/ horizontal resolution), a bandpass filter can be designed with cut off frequencies $1/T_l$ and $1/T_h$ as shown in figure 6. This results in

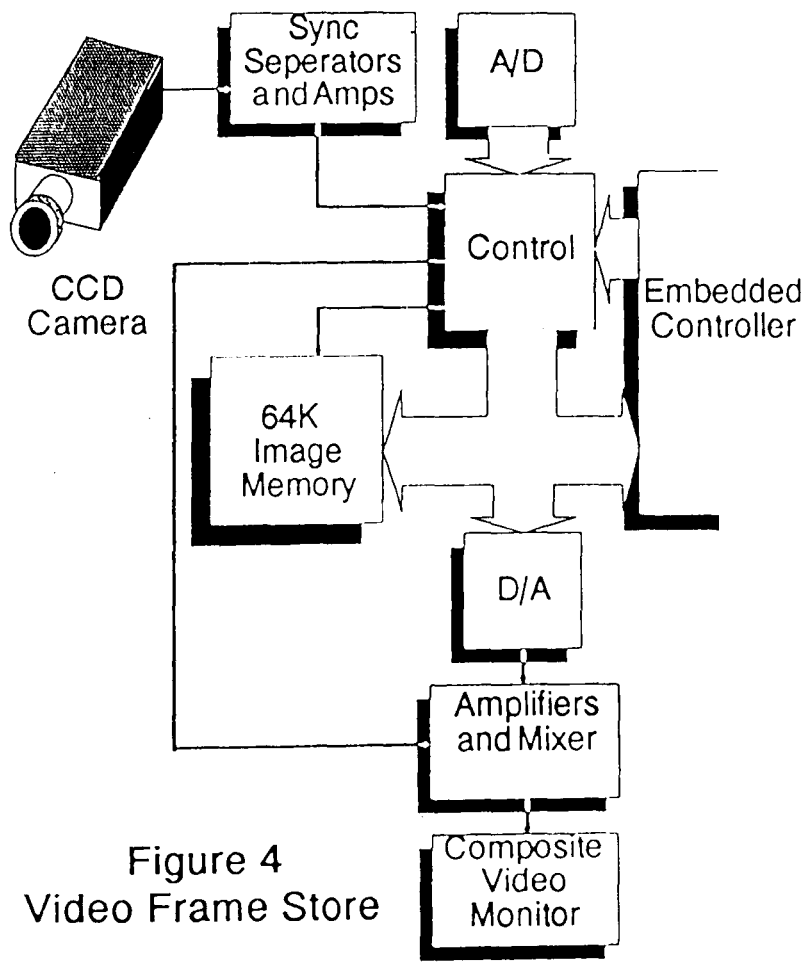


Figure 4
Video Frame Store

very high rejection of signals due to high frequency noise, and low frequency grey level variations across the image.

In the experimental system the following stages of processing are used to extract the coded pattern:

- Average a band of 8 lines close to the bottom of the image.
- Pass the resultant array through a low pass filter to remove signals above $1/T_h$ Hz and then through a high pass filter to remove frequency components below $1/T_l$ Hz.
- derive a binary array by comparing the filtered array with a threshold level.
- Compare the binary array with a copy of the pattern originally projected to determine the position of obstacles if any are present.

AVOIDANCE ALGORITHMS

On detecting an obstacle, the AGV manoeuvres in an attempt to remove the distortion from the projected light pattern. The decision on which way to turn first depends on the position of the obstacle in the image and the layout of the primary guide path. For example if AGVs keep to the left of the factory aisles, the most likely direction to result in successful avoidance would normally be to the right. All the avoidance manoeuvres must be recorded in order to use the information for calculating a resultant return vector to rejoin the primary guide path.

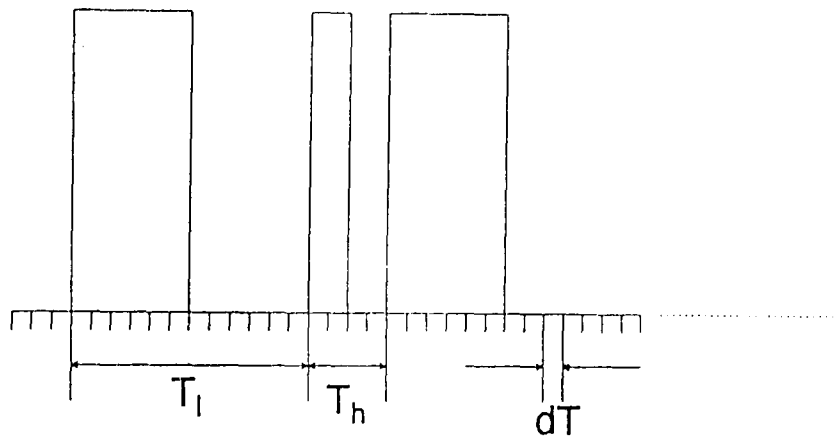


Figure 5 - Bandwidth of a Coding Scheme

After turning, the AGV advances to clear the corner of the obstacle. If a wall or other obstacle interferes with the projected light pattern as the AGV advances, the control software would determine if enough space is available to complete the manoeuvre (determined by the geometry of the system). If the advance is successful, the control system calculates the return course, and the AGV proceeds to rejoin the main guide path.

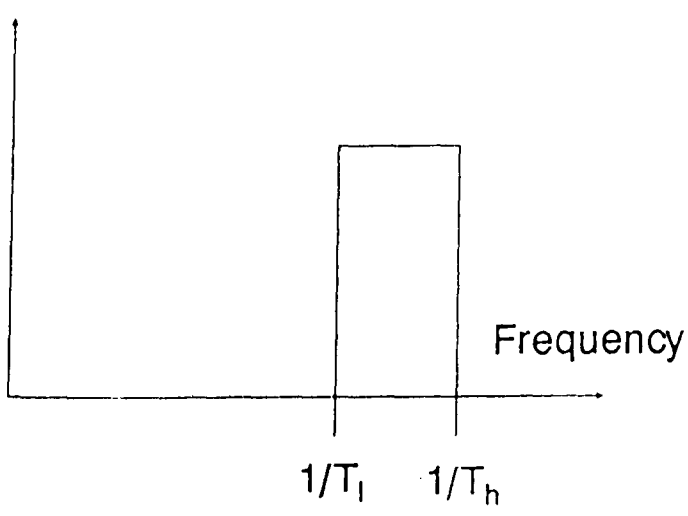


Figure 6 - Filter Cut-Off Frequencies

A more general case would be a larger obstacle or multiple objects, where the AGV could not immediately rejoin the guide path. In this situation, when the AGV attempts to rejoin the guide path, the obstacle will continue to interfere with the projected light pattern. The control system must then adjust the course, advance a further distance and calculate a new return vector. The vehicle again attempts to rejoin the

guide path. This procedure repeats until the obstacles have been cleared.

RESULTS AND CONCLUSIONS

A machine vision for improving the performance of wire guided or other AGV systems which lack facilities to avoid unexpected obstacles has been described.

The use of a coded light pattern has minimised the image processing task since sufficient obstacle information can be extracted from a relatively small section of the image. This allows a low cost embedded controller to be used for image processing and control tasks, resulting in a compact modular unit.

Initial tests have shown that obstacle detection is reliable and flat objects or changes in floor colour and texture have no effect on the performance of the system. The work is part of an on-going research project and further development is at an advanced stage to implement the system on an experimental AGV. Particular emphasis is placed on optimising the coded light pattern in terms of the system bandwidth (the smallest detectable object) and the amount of data processing necessary for reliable obstacle detection. Whilst discrete digital filters have been implemented in software to reject signals outside the code bandwidth, their analogue counterparts are currently being implemented in hardware to further reduce the signal processing burden placed on the embedded controller.

REFERENCES

1 Mortimer, J. (Ed.), Ingersoll Engineers, The FMS Report, IFS Publications, Chapter 1,2, 1982

2 Dayal, R.; Rao, G. N.; Sen, A.P., Design of an Automated Guided Vehicle, Proceedings of the 12th All India Machine Tool Design & Research Conference, PP 203-207, 1986

3 Dunlay, R.T., Obstacle Avoidance Perception Processing for the Autonomous Land Vehicle, Proceedings - 1988 IEEE International Conference on Robotics and Automation, PP 912-917, 1988

4 McTamaney, L. S., Mobile Robots: Real Time Intelligent Control, IEEE Expert, PP 55-68, Volume 2, Part 4, 1988

5 Brady, M., Durrant-Whyte, H., Huoshong, H., Leonard, J., Probert, P., Rao, B.S.Y., Sensor Based Control of AGVs, Computing and Control Engineering Journal, PP 64-70, March 1990

6 Ichikawa, Y., Kamimura, H., Autonomous Vehicle, Proceedings of the 3rd International Conference on Automated Guided Vehicle Systems, PP 199-208, 1985

7 Xuan, G., Wang, L., Li, C., Liu, Y., Wang, J., Intelligent Searching Algorithm for Robot Obstacle Avoidance, Eighth International Conference on Pattern Recognition, PP 958-960, 1986

8 Takeuchi, T.; Nagai, Y.; Enomoto, N., Fuzzy Control of a Mobile Robot for Obstacle Avoidance, Information Sources, PP 231-248, Vol. 45, Part 2, 1988

9 Vuylsteke, P., Oosterlinck, A., Range Image Acquisition With A Single Binary-Encoded Light Pattern, IEEE Transactions on Pattern Analysis and Machine Intelligence, PP 148-164, Volume 12, Number 2, 1990