AN
OPERATOR CONTROLLED MOBILE ROBOT
WITH
MINIMAL COMPUTING POWER

by

Andrew M. J. Hames

B.Sc. Sussex University 1971

A PROJECT SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING

in the School

of

Engineering Science

SIMON FRASER UNIVERSITY

October 2003

APPROVAL

Name: Andrew M. J. Hames

Degree: Master of Engineering

Title of project: **An Operator Controlled Mobile Robot with Minimal Computing Power**

Supervisory Committee:

Dr. Kamal K. Gupta
Senior Supervisor
Professor
School of Engineering Science,
Simon Fraser University

Dr. Andrew Rawicz
Professor
School of Engineering Science,
Simon Fraser University

Date approved: 2003-10-30

ABSTRACT

When driving an electrically power wheelchair, the operator has two tasks, navigation and obstacle avoidance. Handling both may be particularly difficult if the operator is limited by dexterity or cognitive ability. Much work has been done to assist the operator with one or both tasks but the results have often needed a large computing power.

This project explores the possibility of providing useful obstacle avoidance capability with simple proximity sensors and least computing power. The test vehicle is a tele-operated robot with two drive wheels and differential steering. Its onboard computer is a Basic Stamp™ IIsx microprocessor running at 10,000 instructions per second and with a nonvolatile memory of 16K. The obstacle sensors are 19 infra red transmitter/detector pairs. The sensors are simple binary devices that sense the presence of an obstacle but not its range. However in this project the sensors were operated in a dual frequency mode to supply a sense of whether an obstacle is "near" or "very near."

With this equipment I have demonstrated a scale model of a robotic wheelchair that will assist the operator with the job of avoiding obstacles, and also make basic navigation decisions. However, this project has shown that scaling up to a full sized vehicle would require longer range sensors and a microprocessor with more memory than the one selected. My conclusions recommend ultrasonic sensors in place of the infra red ones, and either a Javelin Stamp™ or Motorola MC68HC11 microprocessor in place of the Basic Stamp™ IIsx.

ACKNOWLEDGEMENTS

I am grateful to Placer Dome Inc. for the loan of a digital camera, and for preparing the photographic image files.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Background

Wheelchairs for people with disabilities have been in widespread use for less than 300 years [1], and only within the last 100 years has the design been enhanced to benefit the user.

During the twentieth century innovators began using plastics and metal alloys to make the wheelchair lighter, stronger, more comfortable and more reliable. In the 1970's the powered wheelchair was introduced and gave the user greater mobility and independence. More recently robotics devices have assisted the user with collision avoidance, path planning and navigation. Such wheelchairs use considerable computing power plus a range of sensors including infra red and ultrasonic sensors[1], laser rangefinders, cameras and gyroscopes. Since these devices translate to additional cost for the user, the question that immediately arises is: "What are the smallest computer and most basic sensors that we can use and yet still provide useful assistance to the user?"

---

1 In this report, the word "sensor" will refer to the light-emitting diode together with its photo-transistor receiver. "Detector" refers only to the photo-transistor receiver.

Objectives

My goal with this project is to address that question by building and testing a small robot having the obstacle avoidance capabilities of a basic robotic wheelchair.

A human operator will handle path planning and navigation, while the robot will assist by avoiding obstacles and returning to course afterwards. The robot must do this with minimal computing power and a few simple infra red binary sensors.

Overview of the Work Performed

A local supplier made a range of chassis, microprocessors and sensors available for the project. I selected the Parallex Resources EasyBot™ chassis because of its large electronics bay, and a manoeuverability similar to a manual wheelchair. The servomotors driving the main wheels were retained rather than replaced with stepper motors. There were two reasons for this decision: firstly stepper motors and the associated drivers would have added to the cost of the project; secondly, stepper motors would not improve the robot's ability to avoid obstacles, but only to get back on course. Since giving the robot the ability to turn back on course was added late in the project, I chose to work with the servo motors rather than risk delaying the project by switching.

For the computer, I selected the Basic Stamp™ IIsx which was easy to program and debug, had adequate program memory, and was one of the fastest of the Basic Stamp™ family. I felt that it satisfied the requirement of "minimal computing power"

The infra red sensors provided were compact, lightweight and inexpensive and were selected instead of the more costly ultrasonic sensors. The infra red sensors were simple binary devices that sense the presence of an obstacle but not the distance to it. However they did provide the opportunity to experiment with a dual frequency technique (explained later) which promised to provide rudimentary range information without using the more
costly ultrasonic sensors.

It was recognized at the outset that, while the infra red sensors may be satisfactory for this small scale robot, the greater range of the ultrasonic sensors would be necessary for a full-sized project.

Before assembling the robot I defined the level of assistance that the robot was to provide to the operator, and established the division of responsibility between robot and operator. This was to be the charter from which the robot was designed and programmed

A series of experiments was performed to establish how many sensors to fit onto the robot, and where they should be placed so that it would detect obstacles of all colours, textures, sizes, orientations and positions. The experiments showed that the robot would need 19 sensors for reliable obstacle detection. Since the Basic Stamp™ had only 16 I/O ports, two custom-made expansion boards were built to connect the sensors to the Basic Stamp™ .

Following development and refinement of the control program the robot was tested in three test courses. These courses were designed to test the robot's ability to avoid downward steps, and to negotiate a field of randomly placed obstacles, and a corridor containing obstacles along the walls.

I began the project with the goal of finding the smallest computer that could control a robot and manage obstacle avoidance for the user. I complete it with the opinion that the smallest computer needs to have greater capability - specifically additional memory - than the one that I had chosen. The report ends with specific design improvements that may be of interest to future investigators.

CONTROL PHILOSOPHY

The user will control the robot with a small hand-held controller (See figure 23) linked to the robot chassis by a flexible cable.

The primary aim in designing the robot's control system is to leave the user with as much control as possible. When the robot's sensors detect an obstacle, the control system will initially deny the user the ability to steer towards it, and amplify the ability to turn away from it. A full take-over of speed and steering by the control system will occur only in an emergency.

For example, if the wheelchair is being driven on a course that is converging with a long wall, the control system will initially limit the steering range to prevent the user from steering into the wall. If the wheelchair continues to approach the wall, the control system will guide the wheelchair away from it again. The user can initiate or increase a turn away at any time.

An example in which the user's steering instructions are amplified is when an obstacle can be passed on either side. If the user is steering left when the obstacle is detected,

the control system will increase the turn to pass left of the obstacle. If the user is steering right when the obstacle is detected then the control system will increase the turn to pass right of the obstacle.

Once an obstacle is passed, the robot will attempt to steer back on course. This attempt will be overridden if the user selects a new course by turning the steering wheel.

PRELIMINARIES - CALIBRATING THE SENSORS

In operation, the robot will encounter obstacles of various sizes, textures, colours and orientations. So that the robot would detect any obstacle successfully, I carried out several experiments to establish how many sensors would be needed, and where they should be placed on the robot's body. Each experiment used a single LED with a single detector alongside it. The pair was mounted with their axes horizontal, parallel and 124mm above the test bed. Light emitted from the LED travelled along the test bed and was reflected by the obstacle, back to the detector.

Experiment #1: Response to Different Coloured Obstacles.

The test obstacles were squares of coloured paper 100mm x 100mm in size with their centres on the LED axis. A signal modulated at 38.5kHz was applied to the LED and the distance at which each target was detected is shown in figure 1.

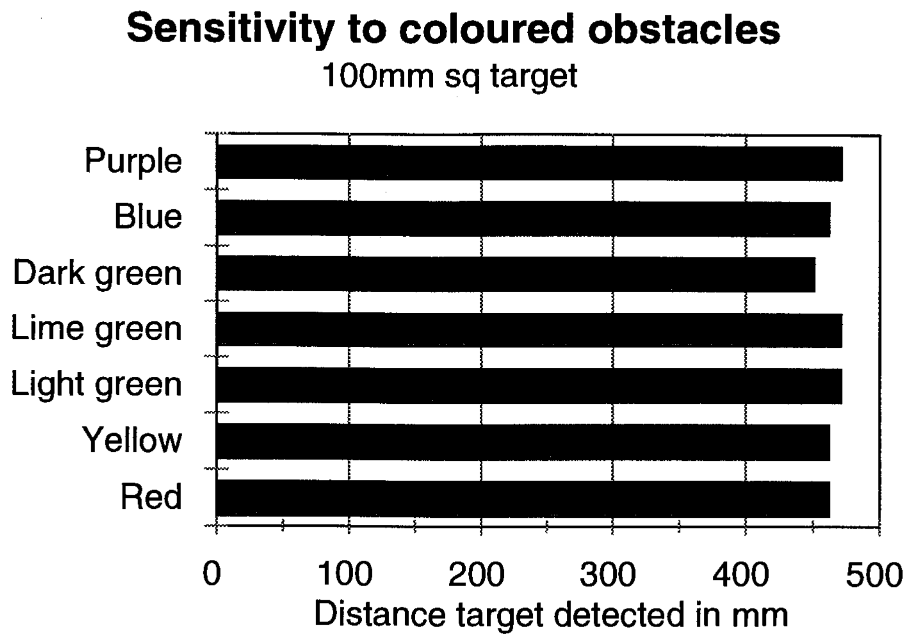


Figure 1. The effect of an obstacle's colour on sensing distance.

The results show a flat response across the spectrum of colours; the sensing distance for each colour is within ± 3% of the mean. The results suggest that the colour of an obstacle will not affect the robot's ability to detect and avoid it.

Experiment #2: A Polar Plot of Detector Sensitivity

To establish the number of sensors needed on the front of the robot, I produced polar plots for a typical LED/detector pair. 100mm x 100mm square targets of white paper, beige cardboard, matt black felt and clear plastic film were used to test the sensitivity to obstacles of various materials and textures. The results in figures 2 to 5 show that, when the targets were within 5° of the sensor axis, the white paper target and the black felt were detected at a minimum range of more than 400mm. All four targets were detected at a range of at least 300mm. That meant that a single sensor would protect a 100mm wide area on the front of the robot. Assuming the polar diagram to be the same in all planes, it would protect an area 100mm high as well.

This robot is 190mm wide and165mm high, so to ensure sufficient coverage, I chose to mount three sensors high up across the front of the robot, and three lower down.

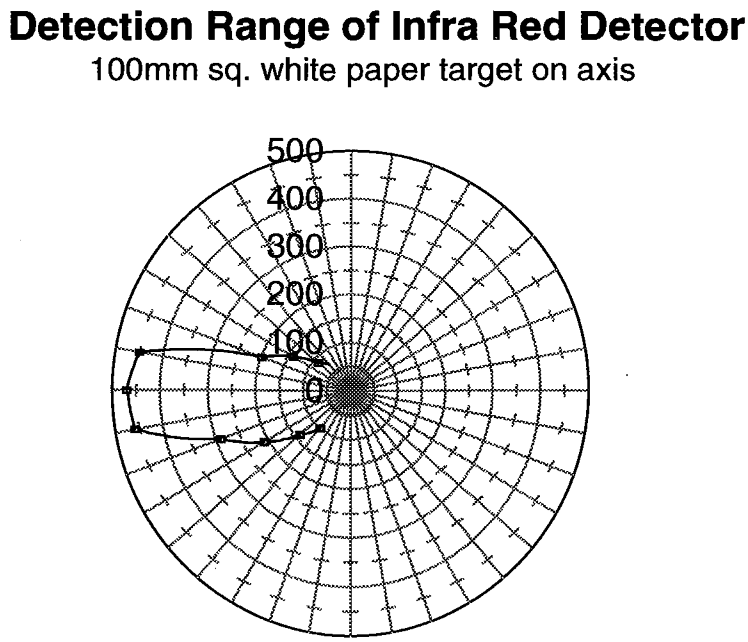


Figure 2. Polar plot of a typical sensor with a white paper target.

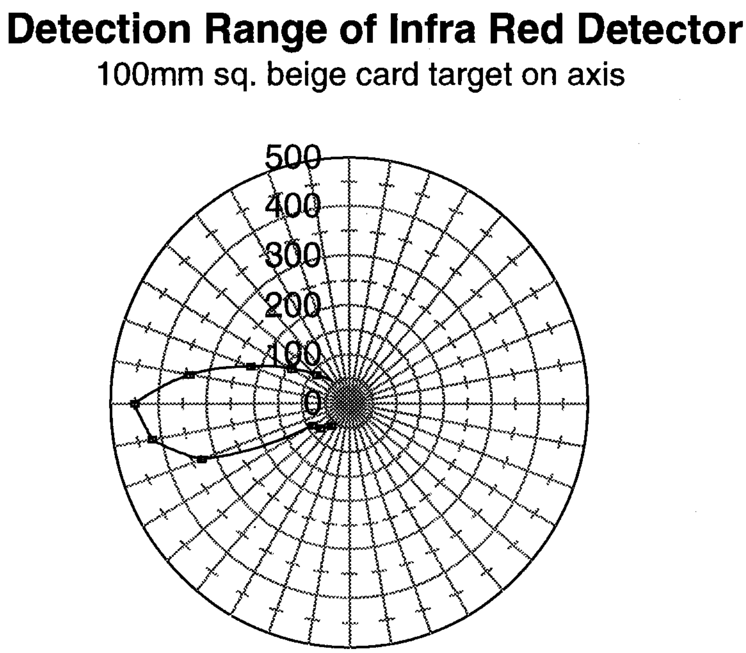


Figure 3. Polar plot of a typical sensor with a beige card target.

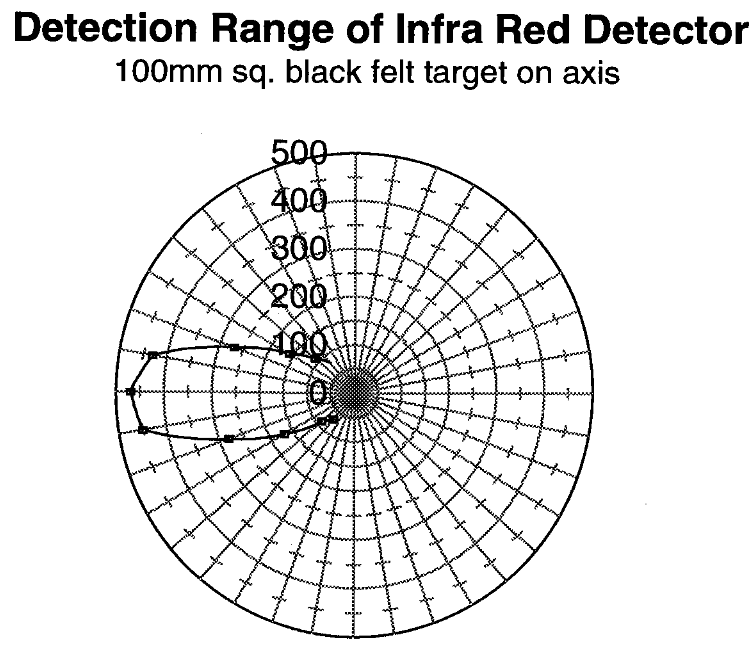


Figure 4. Polar plot of a typical sensor with a black felt target.

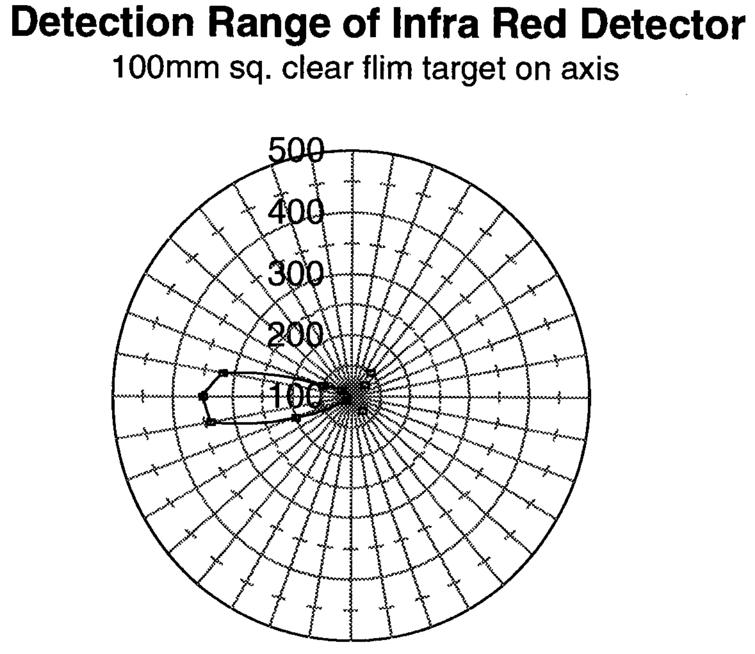


Figure 5. Polar plot of a typical sensor with a clear film target.

Experiment #3: The Effect of Target Orientation

This experiment had two functions. Firstly, I wanted to know if the robot would sense an obstacle if it met it at an angle rather than face on. Secondly, I planned to fit the robot with additional sensors for detecting downward steps. Would these sensors have to point vertically downwards or could they be mounted on the front of the robot and pointed down at some angle? If so, at what angle? If the angle were too shallow, little light would be reflected back to the detector and the robot would react to a step that was not there. If the angle were too steep, the sensor would be inspecting the ground very close to the robot. Then the robot may be too close to a real drop to react in time.

For this experiment the 100mm x 100mm white card target was rotated about a vertical axis and the detection distance measured for each angle. The results in figure 6 show that the detection distance starts to shrink rapidly when the target has been rotated more than 30 degrees from the sensor axis. This encouraged me to believe that an obstacle would be detected in time for the robot to take avoiding action, even if it were seen at an angle.

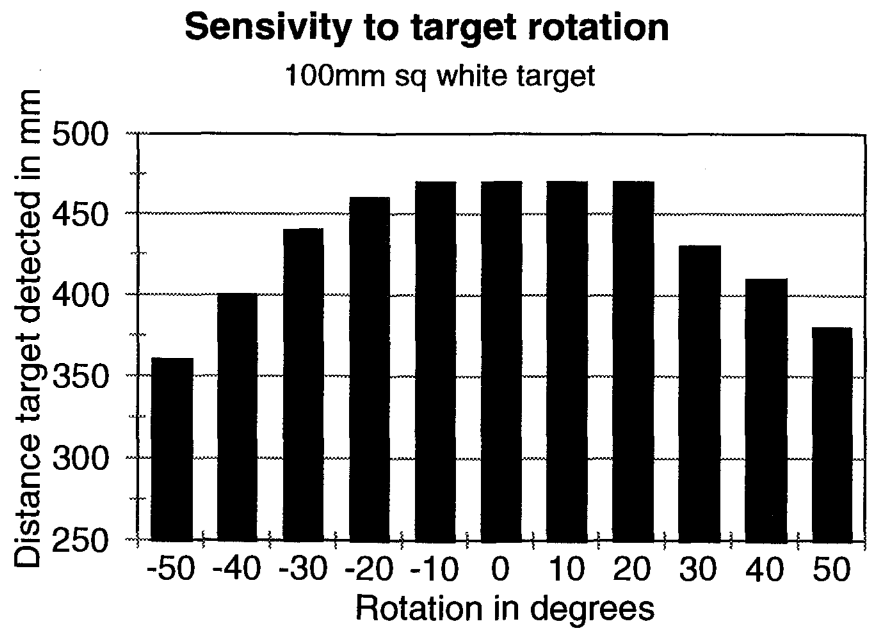


Figure 6. The sensor's response to target rotation.

For the case of the downward looking sensors, I noted from the graph that, if a target were at an angle of 45°, the detector would respond to it 375mm away. If I mounted the sensors on the front of the robot, looking downward at 45°, the ground would be 150mm away, so this mounting arrangement seemed satisfactory. In practice it was not. See Troubleshooting page 44 for details.

Experiment #4: The Effect of LED Modulation Frequency

Each infra red detector contains a bandpass filter with a centre frequency of 38.5kHz. At higher and lower frequencies the filter reduces the sensitivity of the detector so that obstacles must be closer before being detected. In this experiment I planned to explore this variation of sensitivity with frequency to see if I could provide the robot with a crude idea of the range of an obstacle.

With a 100mm x 100mm white target on the centred on the sensor's axis, I modulated the LED source at frequencies from 34.5kHz to 42.5kHz and recorded the distance at which the target was detected. Figure 7 shows the bandpass response of the detector centred on a modulation frequency of 38.5kHz.

Because the colour, surface texture and orientation of an obstacle also affect the sensing distance, Figure 7 cannot be used to determine accurately its range. However, the robot could get an idea of whether an obstacle is "near" or "very near." I would modulate the LEDs at two separate frequencies, and the robot would use the detector outputs at the two frequencies to decide whether to initiate a slow or a sharp turn. At this stage in the project only the term "near" was defined. An obstacle would be considered "near" if it were detected by a sensor operating at a modulation frequency of 38.5kHz. "Very near," "slow turn" and "sharp turn" would be selected based on the robot's performance in an obstacle field.

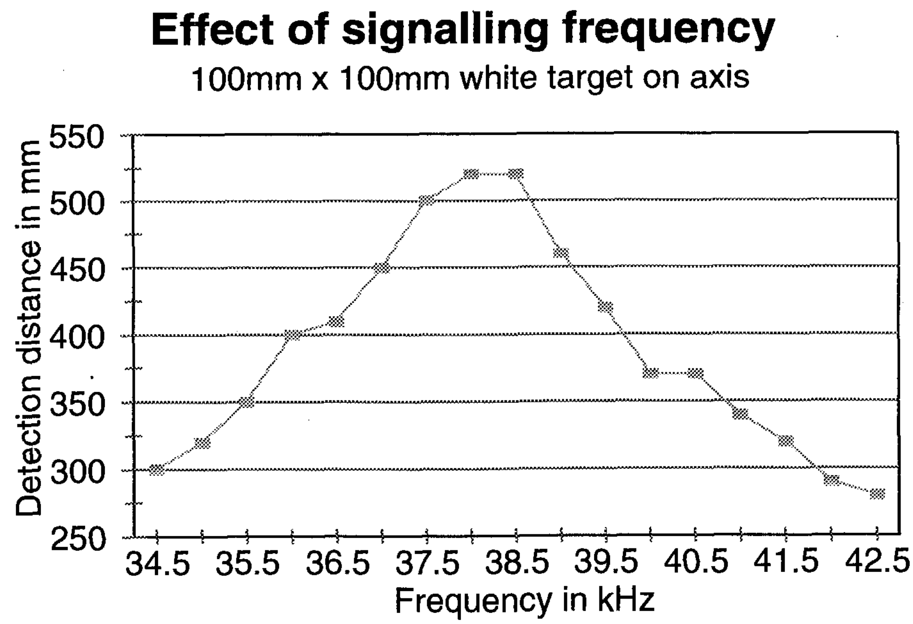


Figure 7. The sensor's response to various LED modulation frequencies.

Experiment #5: Detecting a Barrier at Ground Level.

Previous experiments determined that I would use six forward-looking sensors on the front of the robot, three high up and three low down. I needed to know though just how low to mount the lower three so that they would detect ground level obstacles.

For this experiment the light-emitting diode (LED) and the detector were arranged facing one of three white paper targets. The targets were 100mm wide and 100mm, 80mm and 60mm high respectively set up at ground level. This meant that the tops of the targets were 24mm, 44mm and 64mm respectively below the infra red beam from the LED.

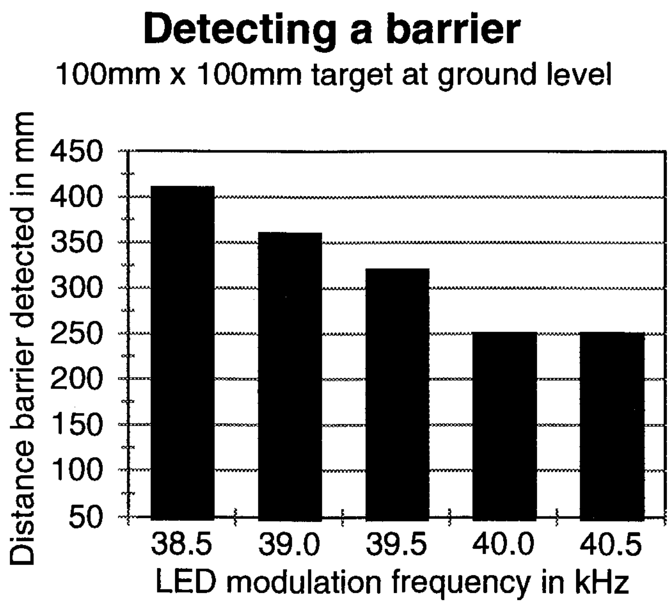


Figure 8. The sensor's response to a ground-level barrier.

Figure 8 shows the results for the 100mm high target. As with Experiment 4, the bandpass filter in the detector reduces the range at which obstacles are detected when the modulation frequency is higher than 38.5kHz.

Compare the first bar of this chart with the polar plot figure 2. The sensing range for a 100mm square target is reduced by only 13% when it is moved from the LED axis to ground level, a distance of 74mm off-axis.

More importantly though, were the results with the other two targets. The 100mm wide x 80mm high target was sensed at a range of 320mm, and only at 38.5kHz, the frequency at which the detector is most sensitive. At a range of less than 240mm the LED beam passed over the target instead of striking it so no reflected light was detected. The 60mm high target was not detected at any frequency.

Having the LED beam pass over the target suggested that a dangerous situation might arise if the robot turned toward an obstacle that was closer than 240mm. The obstacle would not be seen and a collision would occur. To guard against this, the lower three sensors would have to be mounted as low as possible on the body of the robot.

Experiment #6: Detecting an Overhead Obstruction

The natural choice for the next experiment was to detect an overhead obstacle. I wanted to know how high to mount the upper three sensors on the robot. Too high and they would cause it to stop when it could pass safely below an overhead barrier. Too low and the robot would collide with the barrier.

The experiment was performed with different sized rectangular obstacles positioned at various distances above the sensor's axis, and with the LED modulated at various frequencies. I recorded not only the point at which the target was first seen, but also where it was lost again as the target moved closer to the detector.

The results are shown in figures 9 to 20. Some graphs show unusual features that may be related to a lack of precision in the experimental setup. However, since the purpose of the experiments was to help in positioning the sensors, precise measurements were not required, and I did not investigate these features.

Figs. 9 and 13 show that the vertical size of the target has little influence on the distance at which it is first detected. The LED will illuminate most brightly the lower part of the target that is closest to the LED axis. Parts farther away from the axis will be less bright and contribute little to the light reflected back to the detector. The result showed that a thin overhead obstruction is as likely to be seen as a thick one, an encouraging result.

Figs. 10 and14 reveal that the target acquisition distance decreases with increasing modulation frequency, and that the results are almost unaffected by the vertical dimension of the target. The rise of the curve between 40.0kHz and 40.5kHz was unexpected; the cause was not investigated.

Figs. 11 and 12 show that the target is not detected if it gets too close to the sensor. The sensors then, must be closer than 36mm to the top of the robot if they are to protect it from collision with an overhead obstacle. When the base of the target was 16mm above the LED axis, the target was not lost from view when close to the sensor. The robot would be safe then, if the sensors then were mounted 16mm from the top.

Figs. 15 to 20 show the results for targets 100mm high but with various widths. The graphs show that the width of the target has a greater effect on the target acquisition distance than does the vertical size. These results show that, if the sensors are mounted within 16mm of the top of the robot, then even the narrowest target, 40mm wide, will be detected.

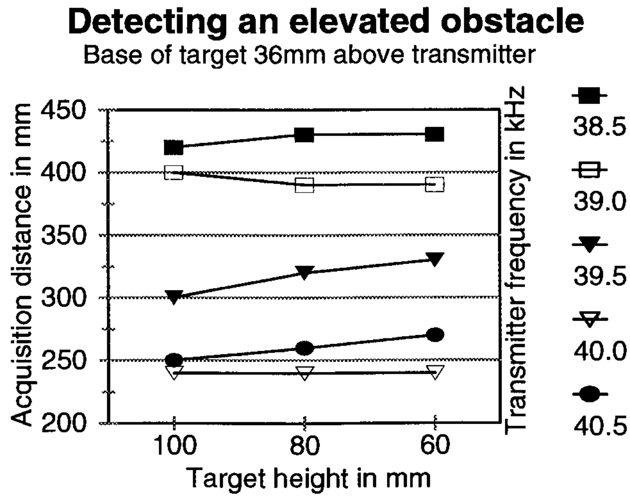


Figure 9. Detecting a target 36mm above the LED axis - effect of target height.

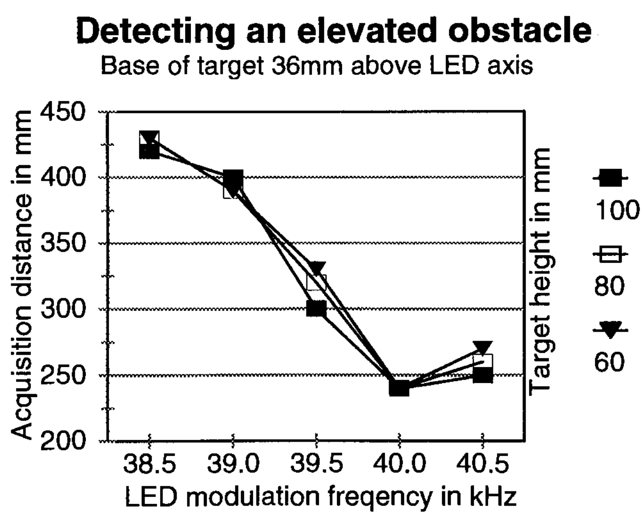


Figure 10. Detecting a target 36mm above the LED axis - effect of LED modulation frequency.

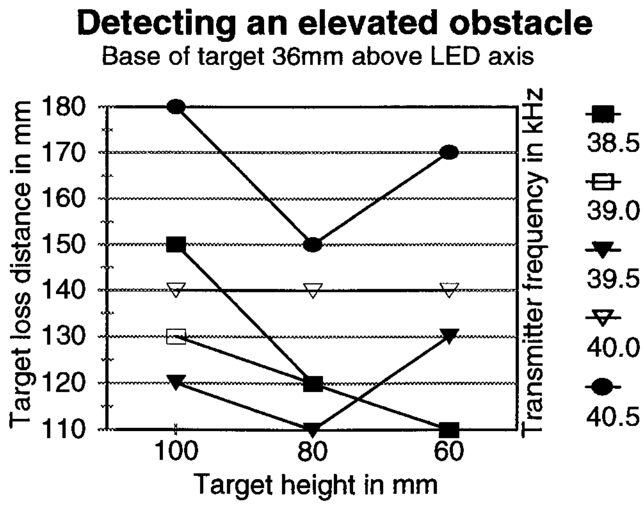


Figure 11. Losing sight of a target 36mm above the LED axis - effect of target height

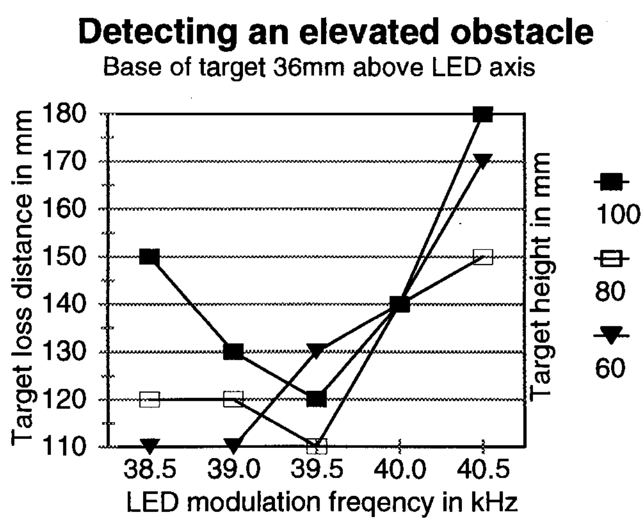


Figure 12. Losing sight of a target 36mm above the LED axis - effect of modulation frequency

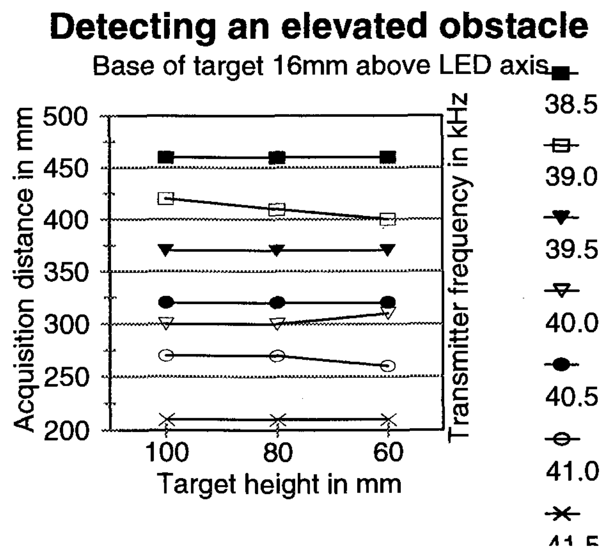


Figure 13. Detecting a target 16mm above the LED axis - effect of target height.

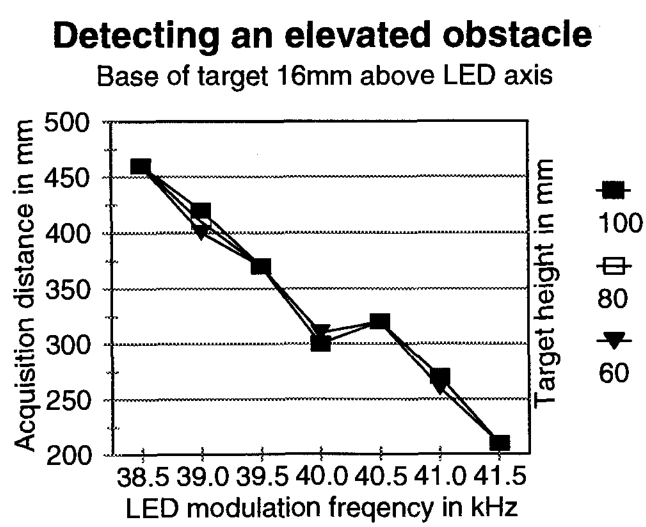


Figure 14. Detecting a target 16mm above the LED axis - effect of LED modulation frequency.

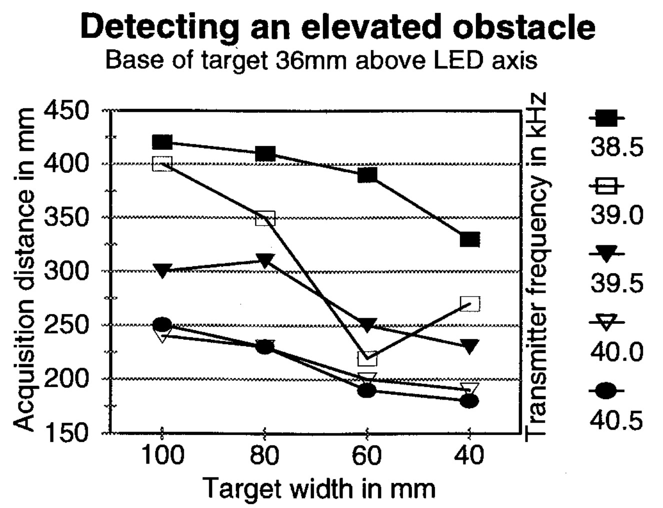


Figure 15. Detecting a target 36mm above the LED axis - effect of target width.

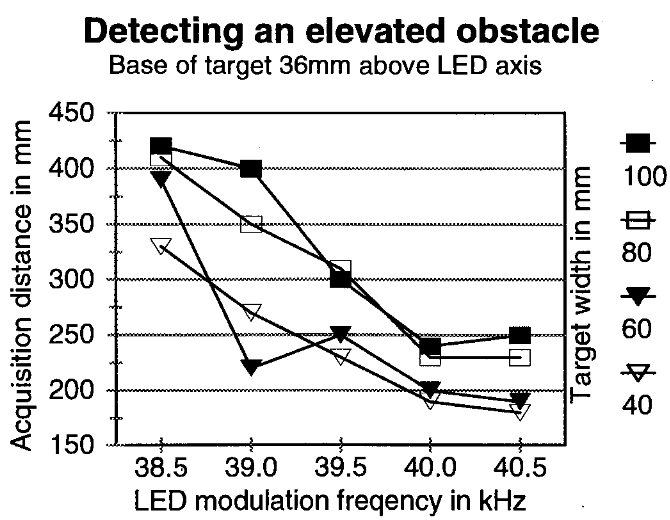


Figure 16. Detecting a target 36mm above the LED axis - effect LED modulation frequency.

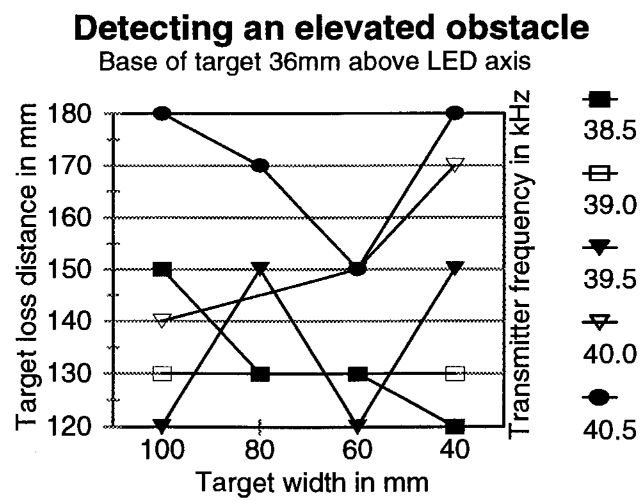


Figure 17. Losing sight of a target 36mm above the LED axis - effect of target width.

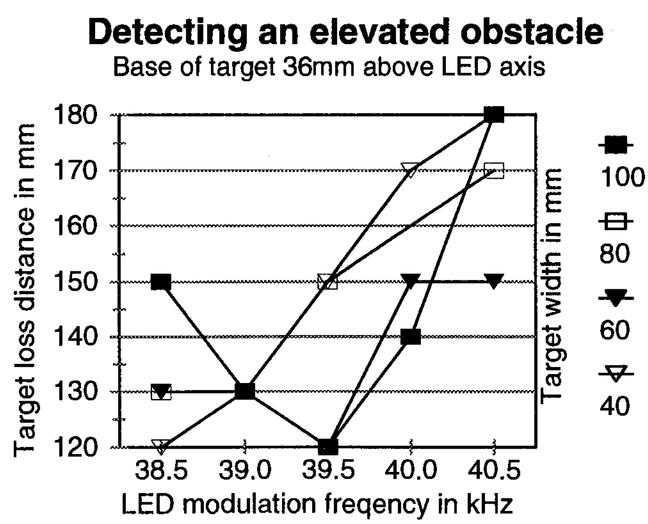


Figure 18. Losing sight of a target 36mm above the LED axis - LED modulation frequency.

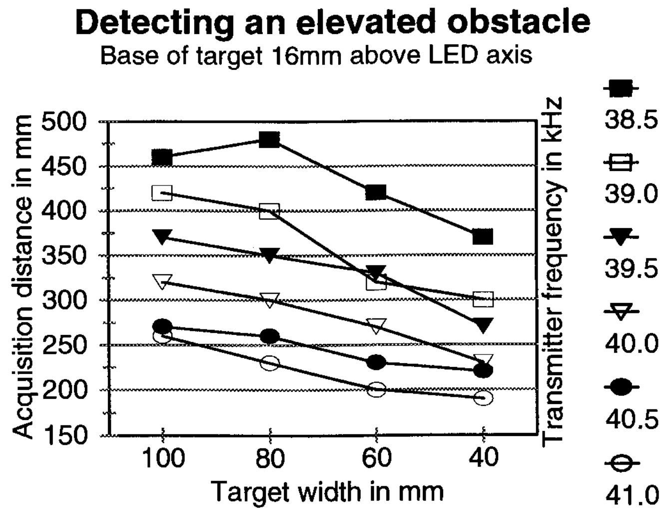


Figure 19. Detecting a target 16mm above the LED axis - effect of target width.

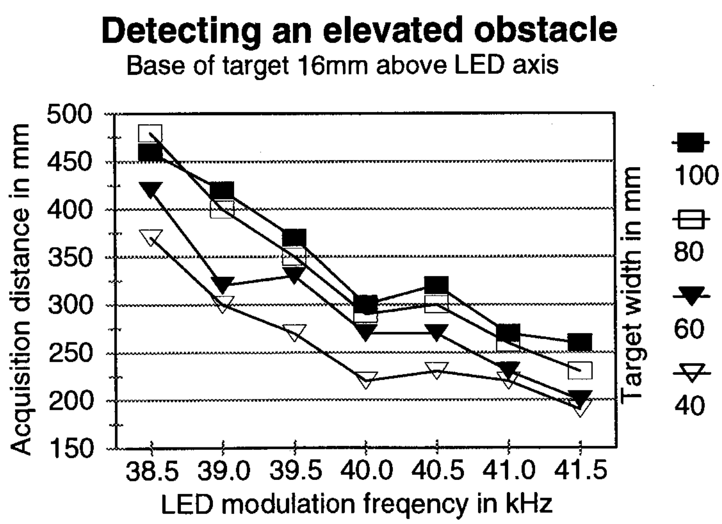


Figure 20. Detecting a target 16mm above the LED axis - effect of LED modulation frequency.

Experiment #7: Detecting a Narrow Post

For the final calibration experiment I checked how a sensor responded to a narrow vertical post. This experiment was not intended to help in positioning the sensors on the robot. Nevertheless, I felt that this type of obstacle would be common and I wanted to know how the robot would react.

The post was made of white paper 250mm high and mounted on the ground directly in front of the sensor. The range as which the post was first detected was found for various modulation frequencies and for differing widths of the post. The results are presented in Figs. 21 and 22.

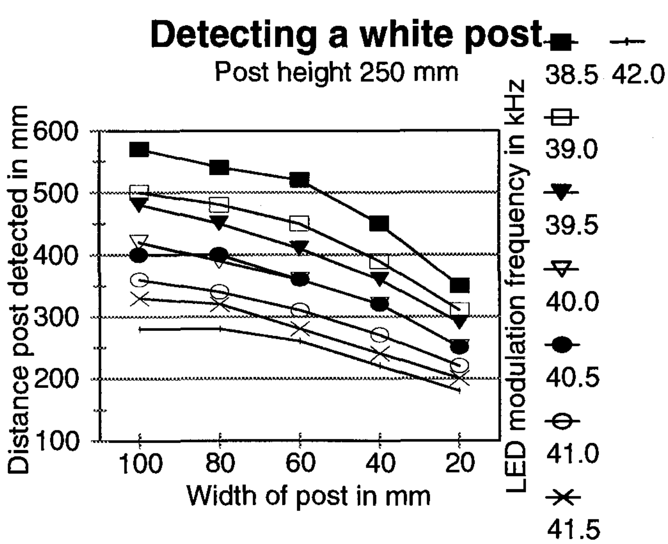


Figure 21. Detecting a white post - effect of post width.

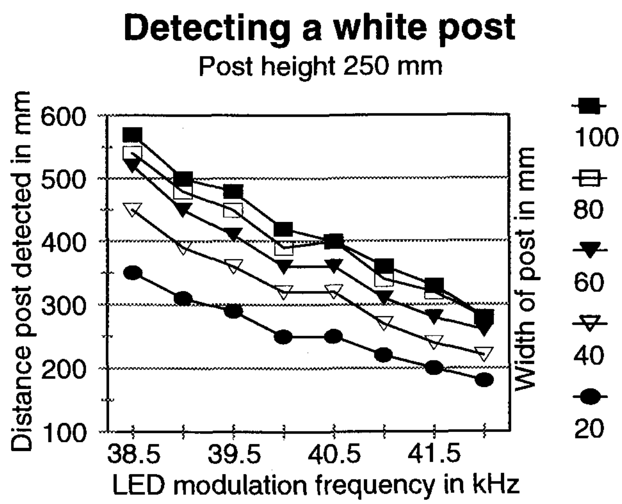


Figure 22. Detecting a white post - effect of LED modulation frequency.

As expected, the distance at which the post was detected diminished with post width and with increasing modulation frequency. I was pleased to find that, at the most sensitive frequency of 38.5kHz, even the narrowest post is detected at 350mm, an acceptable distance. At 40.5kHz, the highest frequency modulating the forward-looking sensors, the narrowest post is detected at 250mm range. A ‘hard’ turn will avert a collision at this range, but, if the post has low-reflective surface, it will not be detected until the range is less and a collision may occur.

THE ROBOT - CONSTRUCTION

Chassis

To simulate the wheelchair I used an EasyBot™ chassis from Parallax Inc. The EasyBot™ is a pre-built robot chassis 165mm high and 190mm wide and includes two servo motors driving 75mm rubber wheels for travel and steering. The drive wheels are mounted at the mid-section, and the unit is balanced with skids at the front and back. This chassis was chosen for two reasons: firstly it has the largest electronics bay of any similar sized chassis; secondly its shape and the location of the drive wheels give it a footprint and a manoeuvrability similar to a manually controlled wheelchair.

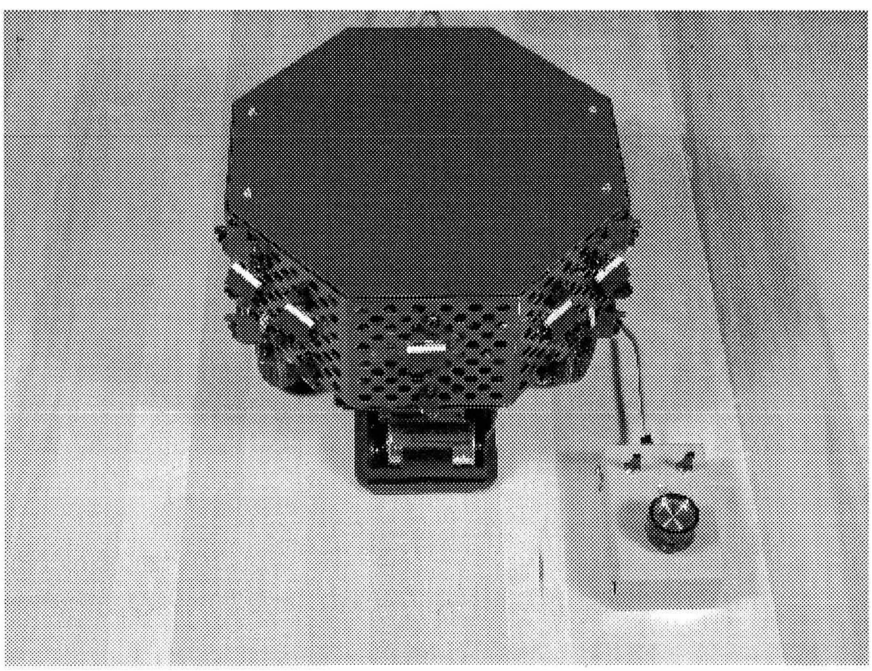


Figure 23. The robot and controller.

Computer

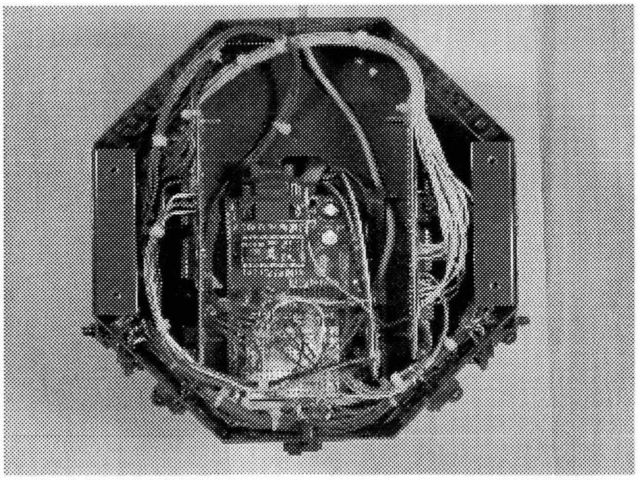
The EasyBot™ is controlled by a Basic Stamp™ IIsx microprocessor that runs at 10,000 instructions per second. It has a 16K nonvolatile memory to hold up to eight programs of 2K each. The Basic Stamp™ IIsx also has 26 bytes available for data storage and 63 bytes as a temporary "scratch pad." Sixteen input/output (I/O) ports may be configured by software as either an input or an output. The command set is 55 instructions.

Programs for the Basic Stamp™ microprocessor are written on a standard desktop computer with Basic Stamp™ Editor software. They are then downloaded by way of a serial cable and run immediately.

The Basic Stamp™ IIsx adequately fits the description of "small computer" as mentioned in the introduction.

Expansion Boards

I felt that 16 I/O ports on the microprocessor would not be enough so I designed and built two expansion boards to increase that number. The output expansion board is built around the 74HC595 serial in / parallel out shift register and provides an additional 32 output ports. Fifteen of these are used in the current project.



The input expansion board is based upon the 74HC165 parallel in / serial out shift register and provides an additional 64 input ports. Twenty-one are used in the current project.

Figure 24. The circuit boards seen from above.

Figures 24 and 25 show the circuit boards mounted inside the robot chassis. In Figure 24 the expansion boards can be seen mounted vertically on either side of the Basic Stamp™ microprocessor. The input expansion board can be seen on the right connected to the light-coloured cable harness; the output expansion board is on the left.

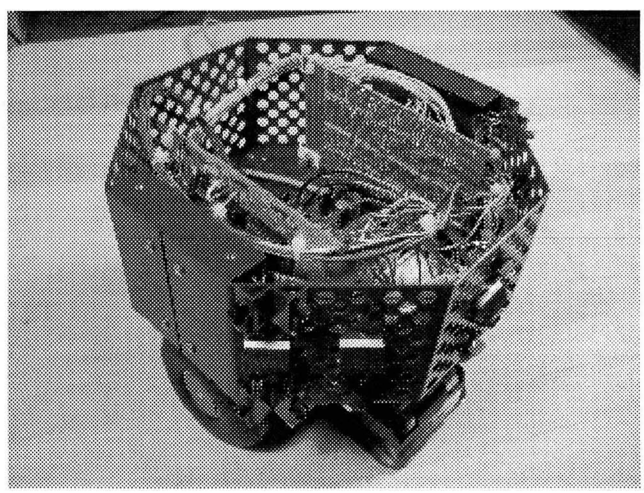


Figure 25. Another view of the expansion boards.

Sensors

Each sensor consists of an infra red light-emitting diode paired with an infra red detector. They are arrayed such that an object approaching any point on the front half of the robot from any direction will be detected. See the diagrams on page 77.

Six sensors look forward Three are mounted high up across the front of the robot's body, and three low down, to provide complete coverage in the direction of travel. Eight more sensors, four mounted high up and four low down, are aimed at 45° and 90° right and left of the direction of travel. These sensors will respond if the robot is approaching a long wall at an acute angle. Finally a group of five sensors is arrayed at mid-height around the front half of the robot. The sensors were mounted on brackets that angled them downwards to sense downward steps in the ground. The brackets can be seen in figure 27.

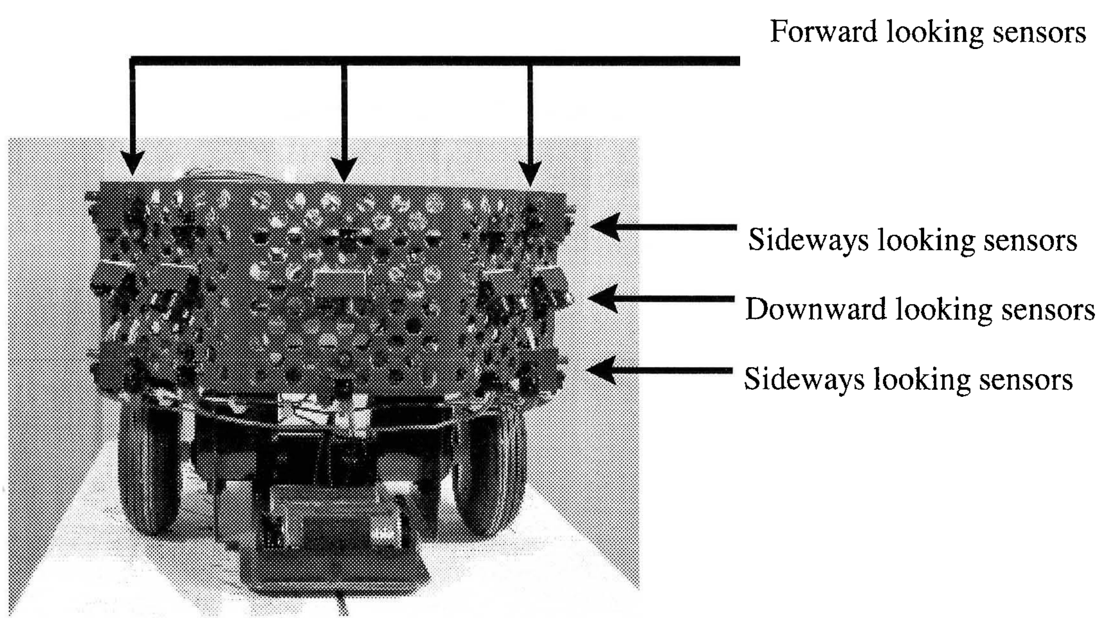


Figure 26. A front view of the robot showing all the sensors.

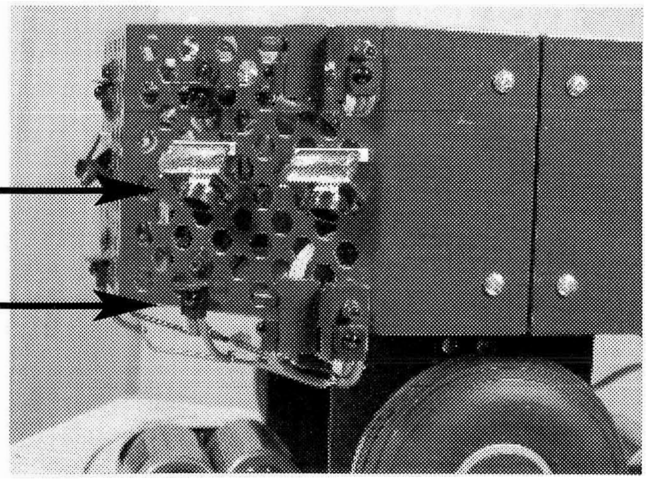


Figure 27. A close up of the robot from the side.

Figures 26 and 27 show the arrangement of the sensors on the robot body. Figure 26 is a general view of the front of the robot showing all the sensors.

Figure 27 is a view from the side and shows all of the techniques used to mount the sensors. Also visible are the methods that I used to prevent stray light from getting into the detectors. Short cylinders surround two of the downward looking sensors, and small square plates are fitted between the light-emitting diode (LED) and the detector on some horizontal facing ones.

THE ROBOT - OPERATION

Overview

The main features of the electronic circuit are shown in Appendix 1. At the left of the diagram is the Basic Stamp microprocessor. At top centre are the input shift registers that convert the sensor output signals into a serial data stream and feed it back to the Basic Stamp™. At bottom centre are the output shift registers. They receive a serial signal from the Basic Stamp™ and convert it into a parallel signal that controls the LED selection transistors. These transistors control which light-emitting diodes (LEDs) will be illuminated.

The user drives the robot using three controls; a start/stop switch, a manual/automatic switch and a potentiometer for steering. These three are mounted on a hand-held control box connected to the robot chassis by a flexible cable.

Details

Drive Wheels

Pins 12 and 13 of the Basic Stamp™ each send a pulse train to the servomotors that turn the main drive wheels. These servomotors are not shown in the diagram. If the length of each pulse in the train is 1.5ms, the wheels remain stationary. If the pulse length is longer, the wheels rotate in one direction, if shorter, they rotate the other way. The servomotors are mounted between the drive wheels so, to drive the robot forwards, one motor must rotate clockwise while the other rotates anticlockwise. That is, the pulses to one servomotor must be lengthened while pulses to the other one are shortened. If the pulses to both servomotors are lengthened, or shortened, by the same amount, the robot will spin on its axis.

The length of each pulse sent to each servomotor is given by

$$\mathbf{L}_{total} = \mathbf{L}_{manualsteering} + \mathbf{L}_{auto} + \mathbf{L}_{bias} + \mathbf{L}_{speed} \qquad \text{for the left wheel}$$

$$\mathbf{L}_{total} = \mathbf{L}_{manualsteering} + \mathbf{L}_{auto} + \mathbf{L}_{bias} - \mathbf{L}_{speed} \qquad \text{for the right wheel}$$

- $\mathbf{L}_{manualsteering}$ is a variable length, equal for each motor, which is related to the position of the steering wheel.

- $\mathbf{L}_{auto}$ is a variable length, equal for both wheels, generated by the program code. The program adjusts $\mathbf{L}_{auto}$ to steer the robot away from an obstacle and back on course again.

- $\mathbf{L}_{bias}$ is a constant length that causes a bias to the steering to compensate for differences between the two servomotors.

- $\mathbf{L}_{speed}$ is a length that increases $\mathbf{L}_{total}$ for one wheel and reduces $\mathbf{L}_{total}$ for the other, and is related to the commanded speed.

Manual Steering

Manual steering is accomplished by controlling $\mathbf{L}_{manualsteering}$ in response to the position of the steering wheel. Before each pulse is sent to the servomotors, capacitor C1 is recharged through R3 and the steering potentiometer VR1. The time taken for C1 to recharge depends on the resistance presented by VR1, and is measured by the Basic Stamp™. Scaling and limiting transforms the measured time into the part $\mathbf{L}_{manualsteering}$ of the pulse sent to the servomotors.

Starting and Stopping

When the start/stop switch is opened, a logic 1 is applied to pin1 of the Basic Stamp™. When the software code sees the logic 1 it accelerates the robot to its target speed. During acceleration, the steering that can be applied by VR1 is limited to prevent the robot from spinning on its axis.

When the start/stop switch is closed, the software reads a logic 0 at pin1 and responds by slowing and stopping the robot. Manual steering is increasingly restricted as the speed diminishes. Once the speed is zero, the robot enters a low power "nap" mode for 3ms before reading the logic state at pin1 again.

Automatic and Manual Modes

Opening the auto/manual switch places the robot in automatic mode by applying a logic 1 to pin 2 of the Basic Stamp™. The obstacle avoidance circuit is activated and contributes to steering the robot. Closing the switch sends a logic 0 to pin 2 and the software bypasses the obstacle avoidance circuit. The robot is then in full manual mode and is steered by VR1 alone.

The Output Expansion Subsystem and Light-Emitting Diodes

The output subsystem consists of two 74HC595 serial in/parallel out shift registers, eight transistor switches Q1 to Q8, and the associated LEDs. To allow future expansion, two shift registers are installed, although only one is in use for this project. Only transistors Q1 and Q2 are shown in Appendix 1.

The process of activating the LEDs (light-emitting diodes) begins by sending the binary sequence 00000001 to the upper 74HC595 shift register via the "Select LEDs" line. A latch pulse to the 74HC595 sends the binary sequence to the parallel outputs, turning on Q1 and turning off Q2 to Q8.

Next, two consecutive signals are sent from the Basic Stamp™ by way of the "Activate LEDs" bus. Since only Q1 is turned on, only LED1 and LED2 will light. The first signal is a 1mS burst at 38.5kHz, the second a 1ms burst at a higher frequency. After each burst the detectors PD1 to PD19 are read. Following the first signal burst, obstacles considered "Near" are recognized. After the second the sensors detect obstacles considered "Very Near" (See next section for a detailed explanation).

The diode D1 protects Q1 from the negative voltage component of this signal, and R5 limits the current in the LED chain to 15mA.

The binary sequence shifts to 00000010 to turn on Q2 and the sequence repeats. There are seven groups of LEDs to signal, so after 01000000 has been sent the sequence reverts to 00000001 and the entire process begins again.

The Input Expansion Subsystem

The input expansion subsystem consists of three 74HC165 parallel in / serial out shift registers and 19 infra red detectors. For simplicity only two shift registers are shown in the circuit diagram, each with four detectors. Each detector includes a bandpass filter centred on 38.5kHz and it outputs a logic 0 when illuminated with infra red light modulated at this frequency. At frequencies other than 38.5kHz sensitivity is reduced and so an object must be closer to the sensor before it is detected and the sensor output goes to logic 0. Now if the sensor receives two consecutive light pulses reflected from an obstacle, one modulated at 38.5kHz and the other modulated at a higher frequency, the robot can determine if the obstacle is "near" or "very near." This method is not precise because the material and the surface texture of an object also affect the distance at which it is detected.

Reading the Sensors and Storing the Results

A group of the LEDs is activated at 38.5kHz and the outputs from the sensors is read and loaded into the 74HC165's. The string of 1's and 0's representing these states is transferred to the Basic Stamp™ and stored as two binary words. Only two or three bits from these words are important each time the sensors are scanned. For example, if LED1 and LED2 have just been fired, we are only interested in the outputs from the sensors mounted alongside LED1 and LED2. Call these PD1 and PD2. These will tell us reliably if an obstacle is "near" and along their line of sight. The relevant bits are transferred to longer term storage and the two binary words are overwritten the next time that the state of the sensors is read.

Before moving on to the next group of sensors, the same LEDs are activated again at a higher LED modulation frequency so that "very near" obstacles may be detected. The frequency of the second signal burst was finally set at 41.75kHz for the 45° and 90° sideways looking sensors, and 40.5kHz for those facing forward.

When all seven groups of LEDs have been fired, the relevant bits in storage give a crude map of obstacles around the front half of the robot and an indication of whether they are "near" or "very near." These bits are then used to select the next incremental motion for the robot.

Locating, Wiring and Sequencing the Sensors

The circuit diagram in Appendix 1 shows that some LEDs are wired in groups of two, and some in groups of three. Where two are wired in series, they are mounted one above the other and aimed horizontally to detect above ground obstacles. It does not matter if the light emitted by the upper LED is detected by the lower sensor.

Where three LEDs are wired in series, two are aimed horizontally to detect above ground obstacles; the third is angled downward to examine the ground for downward steps or potholes. I made sure that the downward looking sensor was not located close to the horizontally aimed ones. My concern was that light emitted by the horizontally aimed LEDs and reflected by an above ground obstacle would be detected by a downward looking sensor. The latter will interpret the light as a reflection from the ground that may not be there, a dangerous situation. To solve this problem the downward looking LED and its associated detector were located 90 degrees away around the circumference of the robot from the other LEDs connected in series with it.

The sequence of firing the LEDs was carefully chosen so that, before the state of any sensor is read, it is allowed a "dark" period of approximately 1mS when it is not illuminated by any sensor on the robot. For example, first a group of LEDs on the left of the robot fires and reflected light illuminates several detectors on the left. During this time the detectors on the right are not illuminated and any residual charge in those detector circuits has time to decay.

Detailed Description of the Code

The program that controls the robot is shown on the next page in pseudo-code form. The main program loop is executed in 10ms in automatic mode, three times faster when manual mode is selected. It contains the instructions to:

- Read the switches and potentiometer on the operator's control box
- Read the infra red sensors
- Make turn decisions

All other functions are handled by subroutines called directly either by the main loop, or by other subroutines. For the full program code see Appendix 2

Table 1. The program pseudo-code

```
:Main

IF Start/Stop switch is set to Stop
        Make a normal stop
        GOTO Main
ENDIF


Read Steering Wheel


IF Auto/Manual switch is set to Manual
        Select manual mode and move a step
        GOTO Main
ENDIF


Read all the sensors


CASE 1 - An obstacle is detected
        Implement a turn to avoid collision and move a step
        GOTO Main

CASE 2 - No obstacle is sensed AND robot is off course
        Implement a turn to get back on course and move a step
        GOTO Main

OTHERWISE
        Select manual mode and move a step
        GOTO Main

END
```

Making Turn Decisions

The sensors look at the robot's surroundings in twelve different directions and seven of these directions are explored at two ranges. To examine every combination before making a turn decision would be an impossible task. Instead I have adopted a hierarchical method.

Firstly I considered the configurations of obstacles that the robot would encounter, and placed them in three groups by priority. The high priority group includes all the situations where the robot may encounter a downward step. Since such hazards are only detected when the robot is close to them, these are the most urgent ones to deal with. All the steering instructions in this group are "emergency right" or "emergency left." For these situations the robot takes control away from the operator and is fully responsible for manoeuvring.

The medium priority group contains the cases where the robot finds an obstacle "very near." The steering instructions are mainly "hard right" and "hard left" and the operator is permitted only to increase the rate of turn away from the obstacle. Two exceptions to the "hard right" and "hard left" instructions are intended to cover cases where the robot is moving along a corridor or close to a wall.

The first exception is where the robot senses an obstacle, possibly the wall of the corridor, "very near" on one side only. "Slow right" and "slow left" instructions were considered enough to steer the robot away from it again. The operator can use the steering wheel to increase the turn initiated by the robot, but cannot counteract it and steer into the obstacle.

The second exception covers the case where the robot is in a narrowing corridor. When obstacles are detected "very near" on both sides the robot considers the gap between them to be too narrow to pass. It stops, backs up about 100mm and rotates through 180° so that it can return the way it has come. The short backup moves the sensors out of range of the obstacles and ensures that the robot does not spin around a second time. The robot is responsible for this manoeuvre and returns control to the operator when the obstacles are no longer "very near."

The low priority group contains the instances in which the robot sees an obstacle that is "near" and the robot takes avoiding action mainly with "slow left" or "slow right" commands.

Again the exceptions cover cases where the robot is moving along a corridor or close to a wall. If an obstacle is detected "near" but not "very near" and on one side only, the robot will take no corrective action, but will limit the operation of the steering wheel to prevent the operator from steering into the obstacle.

If obstacles are detected "near" but not "very near" on both left and right sides, the robot will centre the steering wheel and steer straight ahead until either one obstacle becomes "very near" or some new configuration of obstacles has a higher priority for avoidance.

Within each of the three groups the situations are ranked by the complexity of the encounter. An encounter with three obstacles is given greater precedence than an encounter with one. This is important because the robot will respond to the first logical statement that generates a TRUE response, and will ignore all subsequent statements.

The robot searches through the high priority group, looking for a match between the sensor readings and the statements in the group. If a match is found, the program immediately branches to the action required. If no match is found in the high priority group, the program explores the medium and then the low priority groups.

If all the statements in all three groups are tested against the sensor readings and no match is found, then no obstacles are near. The robot checks to see if it is still off-course from a previous encounter, and if so, will steer back on course. Once back on course the robot returns full control to the operator.

Turning Back on Course

During a return to course the robot will note the direction that it is turning. If another obstacle is detected during the manoeuver, then as long as the operator does not move the steering wheel, the robot will continue turning in the same direction to avoid the new obstacle.

If the operator does turn the steering wheel, the course correction manoeuver is cancelled, the preferred course updated to the new course that the operator is steering, and the robot's steering preference at the next obstacle is nullified. At the next obstacle the robot will follow the operator's steering preference.

For example, suppose that the robot is steering left to get back on course when it encounters an obstacle that it can pass on either side. If the operator has not touched the steering wheel, the robot will turn left; if the wheel is moved, the robot will turn left or right, whichever way the operator is steering.

Sub-Routines

The subroutines are called by the main loop and handle the robot's steering, speed control and various housekeeping routines.

Also handled by subroutines are the decisions to turn left or right based on the direction that the operator is steering. These executive level decisions are handled by the subroutines simply to make the decision making section of the main loop easier to read. Recognizing that I was considering this code as having an application in an electric wheelchair, I smoothed most turns and speed changes by incrementing the steering and speed variables towards their target values. The only exceptions are the emergency stops and turns which are abrupt.

TROUBLESHOOTING

Downward Looking Sensors

The five downward looking sensors were initially mounted on brackets that angled them down at 45° to sense downward steps in the ground. This arrangement did not work very well, particularly if there were dark or non-reflective patches on the ground. The robot was apt to interpret these areas as potholes and to take avoiding action. To solve the problem small blocks were fitted behind the upper part of each angle bracket. These blocks forced the sensors down to an angle of approximately 57° below the horizontal and that improved their ability to detect the ground. At this angle they see the ground some 70mm ahead which is far enough for the robot to make an emergency stop.

Sensors Activated by Stray Light

I encountered two separate problems that were traced to stray light entering a detector. The first was a spurious response caused by the relative positioning of a LED (light-emitting diode) and its associated detector. Unless the front of the LED projected ahead of the detector, light scattered from the rim of the LED housing activated the adjacent detector directly. This problem was solved by mounting a tiny light barrier between the LED and the detector.

The second problem was caused by light being reflected from the robot's body into the downward looking detectors. This resulted in a signal of safe ground to travel on when there was none - a serious flaw. I cured it by mounting a short cylindrical light hood over each downward facing detector.

Robot Moves Faster in "Manual" Mode than in "Automatic" Mode

The main loop of the software code is fully traversed only when the start/stop switch is set to 'start' and the auto/manual switch is set to 'auto'. See page 34 for a description of the action of these two switches. Now because much of the main loop is bypassed in manual mode, the program completes more loops per second in manual mode than it does in automatic. Since the robot makes one incremental move per loop of the code, it travels faster when the switch is set to 'manual.'

To reduce the speed difference between manual and automatic modes, I located the part of the code that is read only in automatic mode and added eight additional calls to the 'move' subroutine. The additional calls are distributed evenly throughout the code so that the robot's movements do not become jerky.

There is a penalty for making these additional calls because the control switches, steering wheel, and obstacle sensors are only read once for each loop of the code. With the extra calls, the robot makes several incremental movements between each reading of these devices. At worst, the robot will travel an additional 5mm before responding to commands. The operator's "feel" of the controls is not noticeably changed by this delay. As far as the effect on collision avoidance is concerned, the delay means that when the sensors detect an obstacle, the robot will cover 2% of the distance to it before responding. This will not materially affect the risk of collision.

Turning Back on Course after an Obstacle

Getting the robot to turn back on course after passing an obstacle proved to be one of the more challenging parts of the project, and the results are not as good as I would like. In principle, the method is to minimize the function

$$\sum_{i=1}^{N} \left| \theta_{i,\,left} - \theta_{i,\,right} \right|$$

where the i th pulse (actually the fraction of the i th pulse which relates to automatic steering) rotates the drive wheel through an angle θ_i. This function is allowed to increase as the robot steers to avoid an obstacle. When no obstacles are in view, the robot is steered until the function is minimized. If the operator adjusts the steering wheel during the manoeuver, the function is set to zero and the robot continues on the new course set by the operator.

If the drive wheels are driven by stepper motors, or the wheels are fitted with encoders, it becomes an easy matter to measure the rotations θ_i. In this project, however, the main wheels are driven by servomotors so the function to be minimized is

$$\sum_{i=1}^{N} \left| L_{i,\,auto,\,left} - L_{i,\,auto,\,right} \right|$$

where L_i is the length of that part of the drive pulse that relates to automatic steering. Using this function I noted that the function varied as expected, increasing as the robot avoided an obstacle and decreasing again as the robot came back on course. However, when the function returned to its initial value, the robot was still not back on course. Worse, the amount by which it was off-course was not consistent.

After lengthy testing, I found some obvious and some possible reasons for the problem.

1) Numerical Overflow. The numerical value of the function became too large for the 16-bit memory location after the robot had turned through about 60 degrees. Dividing the function by ten solved the problem at the expense of some precision in the course correction ability.

2)Noise in the steering potentiometer. I noticed that the remote steering control was susceptible to vibration. A momentary change in resistance would be interpreted as the operator turning the steering wheel, so the program would then cancel any further course correction. I changed the software to increase the threshold where a turn of the wheel is signalled and this improved the noise problem somewhat. A better quality potentiometer would be a superior solution.

3) Differences in the characteristics of the servomotors driving the main wheels. I addressed this problem by adding a bias to the function above. The right wheel must receive move pulses that the left before the function reaches a minimum. The solution is not perfect as the angle between the original course steered and the course to which the

robot returns after avoiding an obstacle varies from one trial to the next. For future projects of this type I would use stepper motors or encoders.

EXPERIMENTAL RESULTS

Prior to any formal, structured testing I needed to adjust and optimize three characteristics on the robot.

- The range at which the robot would consider an obstacle to be "very near."
- The rate of turn for hard and slow turns.
- The rate at which the robot would accelerate into hard and slow turns.

The first of these would be established by the correct choice of the LED modulation frequency. The remaining two would require modifications to the body of the program code.

To set these characteristics I performed several small experiments in which the robot encountered single and multiple obstacles in various configurations. Following each test, one or more of the characteristics were adjusted and the test repeated until I found a combination of settings where, for all obstacles, the robot would:

- React to an obstacle in sufficient time to avoid it.
- Turn sharply enough to avoid collision but not so sharply as to be heading back the way it had come.
- Stop turning away from an obstacle as soon as possible after the threat of collision was gone.

The final setting for "the range at which the robot would consider an obstacle to be "very near"" was set by fixing the LED modulation frequencies for the 45°, and 90° left and right sensors, at 40.5kHz and 41.75kHz respectively. For the other two settings please refer to the program code in Appendix 2.

Once I was satisfied with these settings I tested the robot's abilities in a more formal way in three different environments

- A tabletop with a drop on all sides.
- A field of randomly placed obstacles.
- A corridor with obstacles arranged along its sides

The Tabletop

When the robot approached the edge of the tabletop, it would make an emergency stop, turn on its axis through 45° or 90° and then continue in a new direction. The emergency stop was always quick enough to prevent the robot from going over the edge but, on occasion, turning on its axis would cause one wheel to go over the edge. The problem was solved making the robot back up in a straight line for a short distance before turning.

The distance backed-up was approximately 100mm or roughly half the length of the robot and almost exactly reversed the path followed in approaching the edge. Consequently there was no danger of collision although the back up was made 'blind', that is without the benefit of sensor input.

I discovered a more serious problem that concerned the colour of the surface at the bottom of a step or a pothole. If the surface at the bottom is light or highly reflective, then it reflects sufficient light to activate the detector, giving a false indication of a surface safe to travel on, a dangerous situation. Reducing the sensitivity of the sensor caused the opposite problem; the robot would avoid dark patches of ground, taking them for downward steps that were not there. Not dangerous but a nuisance.

This problem might be solved in future projects by separating the LED and associated detector and using a triangulation approach to ground detection.

The Random Obstacle Field

The obstacle field consisted of several white boxes arranged randomly on a smooth floor. I drove the robot into the field and aimed it toward some particular exit. As the robot progressed I noted the frequency of obstacle avoidance manoeuvres, and the extent of operator control needed for it to reach the desired exit. The experiment was repeated with obstacle fields with other densities and the results are shown on the next page.

For all except the second trial at 0.3m obstacle separation, the LEDs were modulated at 38.5kHz to detect 'near' obstacles, and at 40.5kHz to detect 'very near' obstacles. For the second trial at 0.3 m obstacle separation, 41.75kHz instead of 40.5kHz was applied to the 45° and 90° left and right facing sensors. The effect was to reduce the range of these sensors by about 60% and so allow the robot to travel in narrower corridors.

Table 2. Results with the robot in the random obstacle field.

Min separation between obstacles	Results
1.0m (5 x robot diameter)	An easy course. Few obstacle avoidance manoeuvres. Minimal user input
0.75m (3.75 x robot diameter)	As above. Some user input
0.5m (2.5 x robot diameter)	Many obstacle avoidance manoeuvres. Unable to find the goal without much user assistance. Some collisions.
0.3m (1.5 x robot diameter) 1st trial Side sensors modulated at 40.5kHz	Failed. Gaps too narrow.
0.3m (1.5 x robot diameter) 2nd trial Side sensors modulated at 41.5kHz	Success with much user assistance but robot motion was very jerky

The Corridor with Obstacles

Two planks of wood formed the sides of the corridor and the obstacles were three rectangular boxes. Two were placed against one wall of the corridor and one against the other. The spacing between adjacent pairs of boxes was roughly equal.

As before, the robot was driven though the corridor and I noted the number of obstacle avoidance manoeuvres, and the amount of operator control needed for it to reach the exit. The experiment was repeated with increasingly narrow corridors and the results are shown below.

Table 3. Results with the robot in the corridor.

Width of Corridor	Results
1.25 m (6.25 x robot diameter)	Success. Occasional obstacle avoidance manoeuvres, no user action taken
1.0 m (5 x robot diameter)	Success. Few obstacle avoidance manoeuvres, little user action taken
0.75 m (3.75 x robot diameter) Side sensors modulated at 41.75kHz	Several obstacle avoidance manoeuvres, some successes with frequent user input. Many failures without. Narrow passages (2.0 x robot diameter) contributed to failures. Occasional collisions.
0.6 m (3.0 x robot diameter)	Failure

These results show, as expected, that the robot had more difficulty negotiating a narrow corridor. What is not quite so obvious is the feeling of a transfer of responsibility from the robot to the operator in the narrower corridors. When the spaces between obstacles are large, the robot may only need to respond to one obstacle at a time. Steering instructions are consistent and infrequent, and the operator has little to do beyond periodically updating the course steering. When the obstacles are close together, the robot is overwhelmed by sensor signals. Any change in direction brings a new pattern of signals, resulting in new steering instructions. The result may be a jerky or oscillating steering which may lead to a collision. The operator has to be vigilant to drive the robot in the direction required.

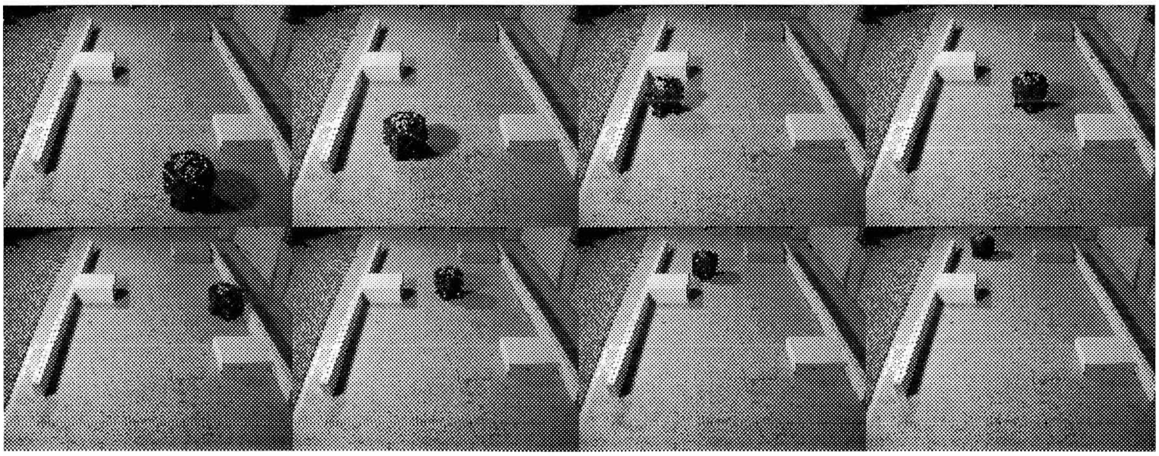


Figure 28. The robot negotiating a wide corridor without operator assistance.

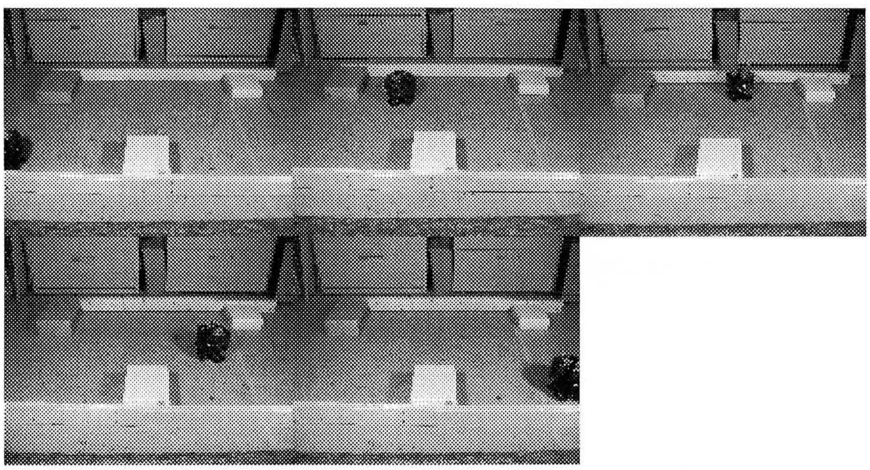


Figure 29. Another trial of the unassisted robot in a wide corridor.

CONCLUSIONS

General

The robot negotiated several obstacle courses successfully although it coped better in when the obstacles were far apart or where the corridors were wide. When the spacing between obstacles was at least four times the robot's diameter, it avoided collisions reliably, returned to its previous course, and worked in cooperation with the human operator. If the operator had no preference it made its own turn decisions to stay on course, but always deferred to the operator if the steering wheel was turned.

In confined spaces the robot was frequently overwhelmed by signals from the sensors and this caused it to have difficulty with the course. I made many adjustments to sensor range and steering characteristics to make the robot behave well in all courses but the exercise was not wholly successful.

In the introduction I noted that the infra red sensors should be replaced with ultrasonic ones for a full sized project. To store range information from an ultrasonic sensor would require one byte per sensor, whereas the infra-red sensors on this project need only two bits per sensor. The additional memory space is not available on the Basic Stamp™ IIsx with my current program code because it uses 97% of the space allocated for variables.

It is worthwhile to note that while my program code uses 97% of the space for memory variables, only 9% of program space is occupied, hardly an efficient use of the chip's resources.

This project has shown that a very small computer can take on minor navigational tasks and collision avoidance in a model robot, but that the Basic Stamp™ is not a suitable choice for a full-scale robotic wheelchair.

Specific Suggestions for Future Projects

Valuable improvements may be realized by replacing the infra-red sensors with ultrasonic ones which, although more costly, would supply an accurate range to each obstacle detected.

For future projects I suggest using a microprocessor such as the Javelin Stamp™ or the Motorola MC68HC11 although both are more expensive that the Basic Stamp™ IIsx. Both have 32k memory - twice a much as the Basic Stamp™ IIsx - and it is shared by the program and memory variables, a more efficient use of memory space.

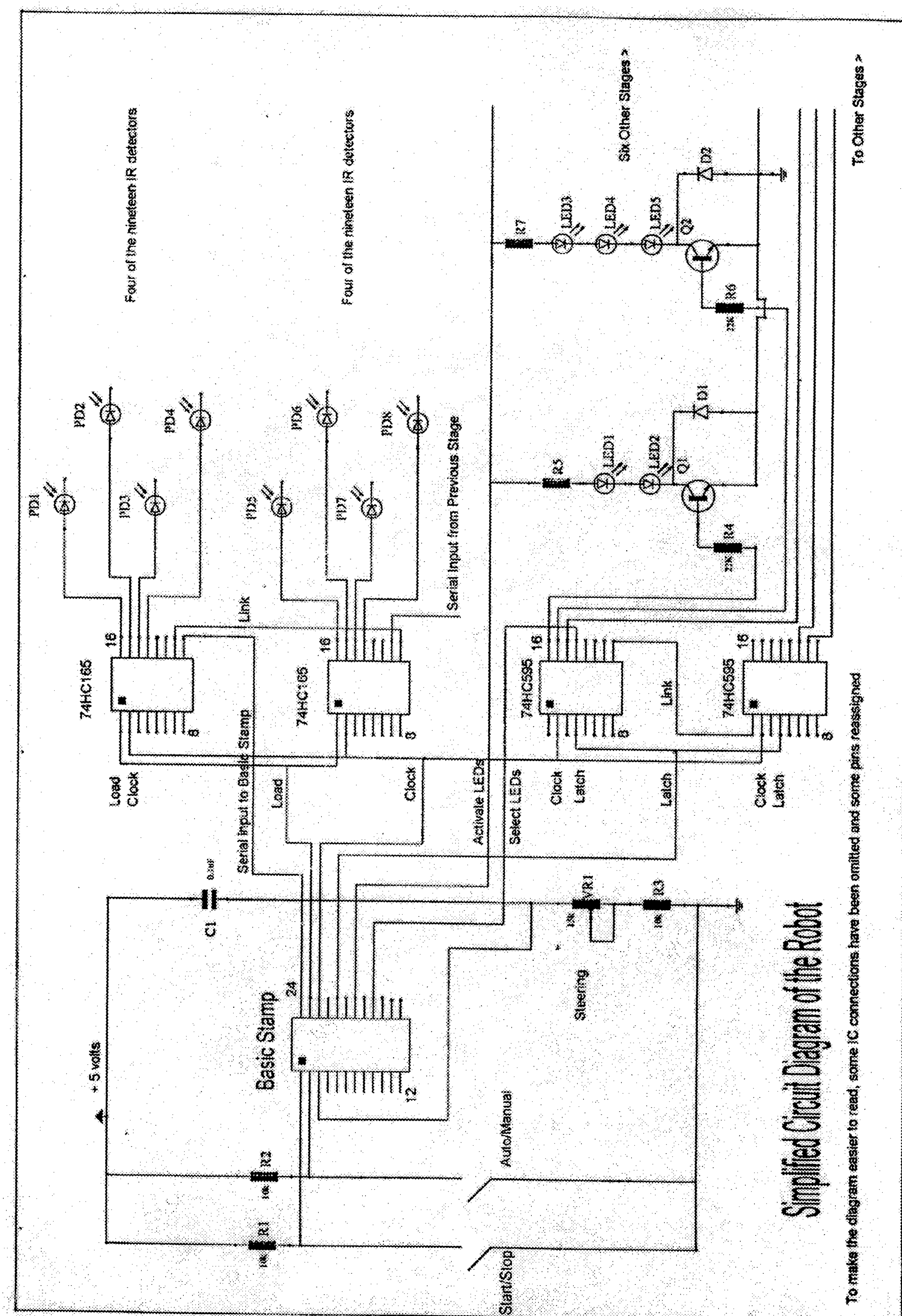
I recommend using stepper motors rather than servomotors to improve the robot's ability to return to a previous course. Servomotors without position feedback, as used in this project, have not allowed the robot to return to a previous course accurately. Servomotors with encoders would be acceptable but require additional I/O ports.

I suggest wiring the downward looking sensors so that they are independent of the forward-looking ones. This approach will use a little more power and require extra lines of software code. Nevertheless, these disadvantages are more than outweighed by the benefits. Firstly, the sensitivity of the downward sensors can then be adjusted independently of the forward-looking ones. Secondly, the sensitivity of each of the forward and sideways looking sensors can be adjusted separately without affecting the downward looking sensors. In this way, the robot's overall zone of sensitivity can be shaped to suit the environment.

APPENDICES

APPENDIX 1

Electronic Circuit Diagram



Simplified Circuit Diagram of the Robot

To make the diagram easier to read, some IC connections have been omitted and some pins reassigned

APPENDIX 2

Program Code

```
'{$stamp bs2sx}
'------- Declare motion variables ----------------------
LeftWheel     CON   12
RightWheel    CON   13
HighSpeed     CON   250
LowSpeed      CON   150
SetSpeed      VAR   BYTE
Speed         VAR   BYTE        'speed that the robot is actually making
ManlSteer     VAR   WORD        'fully manual steering direction
ManlSteerPrev       VAR   WORD        'previous value of ManlSteer
AutoSteer     VAR   WORD        'steering controlled by sensor input
BiasSteer     CON   1323        'steering bias to compensate for servo imbalance
ZeroSteer     CON   32766       'shifted zero for AutoSteer and AccmSteer to avoid
                                overflow
LimtSteer     VAR   WORD        'manual steering but range limited by sensor
                                 input
CntrSteer     CON   2195        'value of RCTIME with steering wheel centred
AccmSteer     VAR   WORD        'steering change accumulated by automatically
                                avoiding obstacles
PrefSteer     VAR   NIB         'the robot's preferred steering direction (0=none,
                                1=left, 2=right)
ReverseGearVAR      BIT         '1=reverse, 0=forward
'------- Declare I/O variables -------
D_out         CON   3           'Data out to 74HC595 pin 14
Clock         CON   4           'Clock to 74HC595 pin 11 & 74HC165 pin 2
Latch         CON   5           'Latch to 74HC595 pin 12
SigtoLED      CON   6           'Signal to IR LEDs
D_in          CON   9           'Data in from 74HC165 pin 9
Load          CON   11          'Load sensor states, 74HC165 pin 1
Pattern       VAR   byte        'LED selector
SensrHigh     VAR   word        'First 16 sensor signals
SensrLow      VAR   byte        'Remaining 8 sensor signals
LowFreq       CON   154         'IR frequency for long range
HiFreq        CON   162         'IR frequency for short range
Freq          VAR   WORD        'Actual IR transmitting freq
indx          VAR   WORD        'loop counter
AAA           VAR   BIT         'Near obstacle
BBB           VAR   BIT         'Near obstacle
CCC           VAR   BIT         'Near obstacle
DDD           VAR   BIT         'Near obstacle
EEE           VAR   BIT         'Near obstacle
FFF           VAR   BIT         'Near obstacle
GGG           VAR   BIT         'Near obstacle
AA            VAR   BIT         'Very near obstacle
BB            VAR   BIT         'Very near obstacle
```

```
CC          VAR   BIT          'Very near obstacle
DD          VAR   BIT          'Very near obstacle
EE          VAR   BIT          'Very near obstacle
FF          VAR   BIT          'Very near obstacle
GG          VAR   BIT          'Very near obstacle
a           VAR   BIT          'Downward step
b           VAR   BIT          'Downward step
d           VAR   BIT          'Downward step
f           VAR   BIT          'Downward step
g           VAR   BIT          'Downward step
i           VAR   WORD
'------- Initialize motion variables ---------
INPUT 0                        'robot stays still when powered up
INPUT 1                        'robot starts in manual mode
SetSpeed = 0
LimtSteer = CntrSteer          'Start with steering wheel centred
AutoSteer = ZeroSteer
AccmSteer = ZeroSteer
PrefSteer = 0                  'robot has no steering preference
ReverseGear = 0
'------- Initialize I/O variables ----
SensrHigh = %0000000000000000
SensrLow = %00000000

'----- Main routine ---------------
main:
ReverseGear = 0                'reverse mode is off
IF in0 = 0 THEN NormalStop     'start switch on the remote is set to stop
GOSUB ReadSteeringWheel        'start switch on the remote is set to start
IF in1 = 0 THEN manual         'manual/auto switch on the remote is set to
                                manual

'------ Read all the sensors --------
GOSUB Chkspeed

Pattern = %00000001
GOSUB ScanSensrLo
AAA=SensrLow.Bit5 & SensrLow.Bit6          'A or left/left
a=SensrLow.Bit7                            'a or front/down
GOSUB ScanSensrHigher
AA=SensrLow.Bit5 & SensrLow.Bit6           'A or left/left

GOSUB Chkspeed

Pattern = %00000010
```

```
GOSUB ScanSensrLo
BBB=SensrHigh.Bit0 & SensrHigh.Bit1          'B or left
b=SensrHigh.Bit2                             'b or right/right/down
GOSUB ScanSensrHigher
BB=SensrHigh.Bit0 & SensrHigh.Bit1           'B or left

GOSUB Chkspeed

Pattern = %00000100
GOSUB ScanSensrLo
CCC=SensrHigh.Bit3 & SensrHigh.Bit4          'C or front
GOSUB ScanSensrHi
CC=SensrHigh.Bit3 & SensrHigh.Bit4           'C or front

GOSUB Chkspeed

Pattern = %00001000
GOSUB ScanSensrLo
DDD=SensrHigh.Bit5 & SensrHigh.Bit6          'D or right
d=SensrHigh.Bit7                             'd or left/left/down
GOSUB ScanSensrHigher
DD=SensrHigh.Bit5 & SensrHigh.Bit6           'D or right

GOSUB Chkspeed

Pattern = %00010000
GOSUB ScanSensrLo
EEE=SensrHigh.Bit8 & SensrHigh.Bit9          'E or right/right
GOSUB ScanSensrHigher
EE=SensrHigh.Bit8 & SensrHigh.Bit9           'E or right/right

GOSUB Chkspeed

Pattern = %00100000
GOSUB ScanSensrLo
FFF=SensrHigh.Bit10 & SensrHigh.Bit11'F or front/left
f=SensrHigh.Bit12                            'f or right/down
GOSUB ScanSensrHi
FF=SensrHigh.Bit10 & SensrHigh.Bit11         'F or front/left

GOSUB Chkspeed

Pattern = %01000000
GOSUB ScanSensrLo
GGG=SensrHigh.Bit13 & SensrHigh.Bit14        'G or front/right
```

```
g=SensrHigh.Bit15                              'g or left/down
GOSUB ScanSensrHi
GG=SensrHigh.Bit13 & SensrHigh.Bit14           'G or front/right

GOSUB Chkspeed

'====== Decisions about obstacles ahead ======
'------ High priority
IF a=1 AND (d=1 OR g=1 OR AAA=0 OR FFF=0 OR BBB=0) THEN
EmergencyRight90
IF a=1 AND (f=1 OR b=1 OR EEE=0 OR GGG=0 OR DDD=0) THEN
EmergencyLeft90
IF a=1 THEN EmergencyLeftEmergencyRight
IF f=1 OR b=1 THEN EmergencyLeft45
IF g=1 OR d=1 THEN EmergencyRight45

'------ Medium priority
IF ((AA=0 OR BB=0) AND (DD=0 OR EE=0)) THEN EmergencyRight180
                        'Corridor
IF (FF=0 OR CC=0 OR GG=0) AND (BBB=0 OR AAA=0) THEN HardRight
                        'ahead and (left  or left/left)
IF (FF=0 OR CC=0 OR GG=0) AND (DDD=0 OR EEE=0) THEN HardLeft
                        'ahead and (right or right/right)
IF (FF=0 OR CC=0 OR GG=0) AND PrefSteer = 1 THEN HardLeft
                        'partial ahead
IF (FF=0 OR CC=0 OR GG=0) AND PrefSteer = 2 THEN HardRight
                        'partial ahead
IF (FF=0 OR CC=0 OR GG=0) AND PrefSteer = 0 THEN HardLeftHardRight
                        'partial ahead
 IF (AA=0 OR BB=0) AND (EE=1 AND DD=1) THEN Slowright
                        'left/left or left only
IF (EE=0 OR DD=0) AND (AA=1 AND BB=1) THEN Slowleft
                        'right/right or right only

'------ Low Priority
IF (FFF=0 OR CCC=0 OR GGG=0) AND (BBB=0 OR AAA=0) THEN SlowRight
                        'ahead and (left or left/left)
IF (FFF=0 OR CCC=0 OR GGG=0) AND (DDD=0 OR EEE=0) THEN
                        SlowLeft  'ahead and (right or right/right)

IF (FFF=0 OR CCC=0 OR GGG=0) AND PrefSteer = 1 THEN SlowLeft
                        'partial ahead
IF (FFF=0 OR CCC=0 OR GGG=0) AND PrefSteer = 2 THEN SlowRight
                        'partial ahead
IF (FFF=0 OR CCC=0 OR GGG=0) AND PrefSteer = 0 THEN SlowLeftSlowRight
```

```
                    'partial ahead

IF ((AAA=0 OR BBB=0) AND (DDD=0 OR EEE=0)) THEN NoLeftNoRight
                    'Corridor

IF (AAA=0 OR BBB=0) AND  PrefSteer = 2 THEN SlowRight 'left/left or left
IF (AAA=0 OR BBB=0) THEN Noleft                        'left/left or left
IF (EEE=0 OR DDD=0) AND  PrefSteer = 1 THEN SlowLeft  'right/right or right
IF (EEE=0 OR DDD=0) THEN Noright                       'right/right or right

'------ In the clear
IF (AccmSteer < (ZeroSteer-85)) THEN CorrectRight  'Accumulated steering is to
                        the left and robot is in auto mode
IF (AccmSteer > (ZeroSteer+65)) THEN CorrectLeft   'Accumulated steering is to
                        the right and robot is in auto mode
GOTO Manual

ReadSteeringWheel:
    ManlSteerPrev = ManlSteer'copy current value to previous value
    HIGH 2                          'discharge the capacitor
    PAUSE 1
    RCTIME 2,1,Manlsteer            'store the wheel position
    IF ((ManlSteerPrev - ManlSteer)<40 OR (ManlSteerPrev - ManlSteer)>65505)
        THEN NotTurned   'Steering wheel has not been turned
        GOSUB WheelTurned
    NotTurned:
    RETURN

ScanSensrLo:                'Long range scan
    Freq = LowFreq * 100
    GOSUB ScanSensrGroup
    RETURN

ScanSensrHi:                'Short range scan
    Freq = HiFreq * 100
    GOSUB ScanSensrGroup
    RETURN

ScanSensrHigher:                'Short range scan for side-looking sensors
    Freq = 170 * 100
    GOSUB ScanSensrGroup
    RETURN

ScanSensrGroup:
    HIGH Load                   '74HC165 loads on a hi to lo transition
```

```
        SHIFTOUT D_out, Clock,MSBFirst, [pattern\16]'select the LEDs
        PULSOUT Latch,3
        FREQOUT SigtoLED,2,Freq                         'transmit to LEDs
        PULSOUT Load,3                                  'read the sensors
        SHIFTIN D_in,Clock,MSBPre,[SensrHigh\16,SensrLow\8]     'return sensor
                                signals to STAMP
        RETURN

Manual:
        PrefSteer = 0               'robot has no preferred direction to steer
        GOSUB SmoothLimtSteer
        GOSUB SmoothAutoSteer
        AccmSteer = ZeroSteer
        SetSpeed = HighSpeed
        GOSUB ChkSpeed
        GOTO Main

ChkSpeed:
        IF SetSpeed>Speed THEN Faster
        IF SetSpeed<Speed THEN Slower
        GOSUB Move
        RETURN

Faster:
        Speed = (Speed + 4) MAX SetSpeed
        GOSUB Move
        RETURN

Slower:
        Speed = (Speed - (4 MAX Speed)) MIN SetSpeed      'To prevent
                            underflow
        GOSUB Move
        RETURN

Move:
        PULSOUT
        LeftWheel,(4330-LimtSteer)/4+BiasSteer+AutoSteer-ZeroSteer+Speed-((Spee
        d+100)
                *ReverseGear)
        PULSOUT
        RightWheel,(4330-LimtSteer)/4+BiasSteer+AutoSteer-ZeroSteer-Speed+((Spe
        ed+100)
                *ReverseGear)
        AccmSteer = AccmSteer + (Autosteer/10) - (ZeroSteer/10)
        RETURN
```

```
NormalStop:
        SetSpeed = 0
        GOSUB SteerStraight
        GOSUB SmoothAutoSteer
        IF (Speed = 0 AND in0 = 0) THEN Rest
        GOSUB ChkSpeed
        GOTO Main

Rest:
        NAP 3
        GOTO main

EmergencyLeft90:
        PAUSE 1000
        Speed = 0
        LimtSteer = CntrSteer                  'take control away from the operator and
                                                centre the steering wheel
        AutoSteer = ZeroSteer                  'Zero the autosteer
        ReverseGear = 1                        'reverse
        FOR i=1 TO 200
                IF (in0 = 0 OR in1 = 0) THEN Main'abort if stop or manual is selected
                GOSUB MOVE
        NEXT
        ReverseGear = 0                        'Forward
        PAUSE 500
        AutoSteer = ZeroSteer-100'autosteer left
        FOR i=1 TO 385
                IF (in0 = 0 OR in1 = 0) THEN Main
                GOSUB Move
        NEXT
        PAUSE 1000
        GOTO main

EmergencyLeft45:
        PAUSE 1000
        Speed = 0
        LimtSteer = CntrSteer                  'take control away from the operator and
                                                centre the steering wheel
        AutoSteer = ZeroSteer                  'Zero the autosteer
        ReverseGear =                          'reverse
        FOR i=1 TO 200
                IF (in0 = 0 OR in1 = 0) THEN Main
                GOSUB MOVE
        NEXT
        ReverseGear = 0                        'Forward
```

```
        PAUSE 500
        AutoSteer = ZeroSteer-100'autosteer left
        FOR i=1 TO 195
                IF (in0 = 0 OR in1 = 0) THEN Main
                GOSUB Move
        NEXT
        PAUSE 1000
        GOTO main

EmergencyRight90:
        PAUSE 1000
        Speed = 0
        LimtSteer = CntrSteer              'take control away from the operator and
                                            centre the steering wheel
        AutoSteer = ZeroSteer              'Zero the autosteer
        ReverseGear = 1                    'reverse
        FOR i=1 TO 200
                IF (in0 = 0 OR in1 = 0) THEN Main
                GOSUB MOVE
        NEXT
        ReverseGear = 0                    'Forward
        PAUSE 500
        AutoSteer = ZeroSteer+100          'autosteer right
        FOR i=1 TO 385
                IF (in0 = 0 OR in1 = 0) THEN Main
                GOSUB Move
        NEXT
        PAUSE 1000
        GOTO main

EmergencyRight45:
        PAUSE 1000
        Speed = 0
        LimtSteer = CntrSteer              'take control away from the operator and
                                            centre the steering wheel
        AutoSteer = ZeroSteer              'Zero the autosteer
        ReverseGear = 1                    'reverse
        FOR i=1 TO 200
                IF (in0 = 0 OR in1 = 0) THEN Main
                GOSUB MOVE
        NEXT
        ReverseGear = 0                    'Forward
        PAUSE 500
        AutoSteer = ZeroSteer+100          'autosteer right
        FOR i=1 TO 195
```

```
            IF (in0 = 0 OR in1 = 0) THEN Main
            GOSUB Move
        NEXT
        PAUSE 1000
        GOTO main

EmergencyRight180:
        PAUSE 1000
        Speed = 0
        LimtSteer = CntrSteer          'take control away from the operator and
                                        centre the steering wheel
        AutoSteer = ZeroSteer          'Zero the autosteer
        ReverseGear = 1                'reverse
        FOR i=1 TO 200
            IF (in0 = 0 OR in1 = 0) THEN Main
            GOSUB MOVE
        NEXT
        ReverseGear = 0                'Forward
        PAUSE 500
        AutoSteer = ZeroSteer+100      'autosteer right
        FOR i=1 TO 760
            IF (in0 = 0 OR in1 = 0) THEN Main
            GOSUB Move
        NEXT
        PAUSE 1000
        GOTO main

HardLeft:
        SetSpeed = LowSpeed
        LimtSteer = ManlSteer MIN CntrSteer MAX (CntrSteer+(4*Speed))
        AutoSteer = (Autosteer-20) MIN (ZeroSteer-Speed)
        GOSUB ChkSpeed
        GOTO main

HardRight:
        SetSpeed = LowSpeed
        LimtSteer = ManlSteer MAX CntrSteer MIN (CntrSteer-(4*Speed))
        AutoSteer = (Autosteer+20) MAX (ZeroSteer+Speed)
        GOSUB ChkSpeed
        GOTO main

SlowLeft:
        LimtSteer = ManlSteer MIN CntrSteer MAX (CntrSteer+(4*Speed))
        SetSpeed = HighSpeed
        AutoSteer = (Autosteer-15) MIN (ZeroSteer-(Speed/2))
```

```
        GOSUB ChkSpeed
        GOTO main

SlowRight:
        LimtSteer = ManlSteer MAX CntrSteer MIN (CntrSteer-(4*Speed))
        SetSpeed = HighSpeed
        AutoSteer = (Autosteer+15) MAX (ZeroSteer+(Speed/2))
        GOSUB ChkSpeed
        GOTO main

CorrectLeft:
        PrefSteer = 1          'Robot prefers to steer left
        SetSpeed = HighSpeed
        AutoSteer = (Autosteer-15) MIN (ZeroSteer-(Speed/2))
        GOSUB ChkSpeed
        GOTO main

CorrectRight:
        PrefSteer = 2          'Robot prefers to steer right
        SetSpeed = HighSpeed
        AutoSteer = (Autosteer+15) MAX (ZeroSteer+(Speed/2))
        GOSUB ChkSpeed
        GOTO main

NoLeft:
        LimtSteer = ManlSteer MAX CntrSteer MIN (CntrSteer-(4*Speed))
        SetSpeed = HighSpeed
        GOSUB ChkSpeed
        GOTO main

NoRight:
        LimtSteer = ManlSteer MIN CntrSteer MAX (CntrSteer+(4*Speed))
        SetSpeed = HighSpeed
        GOSUB ChkSpeed
        GOTO main

NoLeftNoRight:
        LimtSteer = CntrSteer                        'Centre the steering wheel
        SetSpeed = HighSpeed
        GOSUB ChkSpeed
        GOTO main

SlowLeftHardRight:
        IF ManlSteer > CntrSteer THEN SlowLeft       'Left if operator is
                                                     steering right
```

```
        GOTO HardRight                                  'Else right

SlowRightHardLeft:
        IF ManlSteer < CntrSteer THEN SlowRight         'Right if operator is steering
                                                        right
        GOTO HardLeft                                   'Else left

EmergencyLeftEmergencyRight:
        IF ManlSteer < CntrSteer THEN EmergencyRight90  'Right if operator is
                                                        steering right
        GOTO EmergencyLeft90                            'Else left

HardLeftHardRight:
        IF ManlSteer < CntrSteer THEN HardRight         'Right if operator is steering
                                                        right
        GOTO HardLeft                                   'Else left

SlowLeftSlowRight:
        IF ManlSteer < CntrSteer THEN SlowRight         'Right if operator is steering
                                                        right
        GOTO SlowLeft                                   'Else left

WheelTurned:
        AccmSteer = ZeroSteer    'Zero the accumulated steering if the operator
                                  sets a new course
        PrefSteer = 0         'robot relinquishes the preferred steering direction
        RETURN

SmoothLimtSteer:
        IF LimtSteer < ManlSteer THEN LimtSteerUp
                LimtSteer = LimtSteer-100 MIN ManlSteer MAX (CntrSteer+(4*Speed))
                        'limit steering range during acceleration so robot does not spin
                RETURN
        LimtSteerUP:
                LimtSteer = LimtSteer+100 MAX ManlSteer MIN (CntrSteer-(4*Speed))
                        'limit steering range during acceleration so robot does not spin
                RETURN

SmoothAutoSteer:
        IF AutoSteer < ZeroSteer THEN AutoSteerUp
                AutoSteer = AutoSteer-15 MIN ZeroSteer          'AutoSteer must be
                                                                higher then ZeroSteer
                RETURN
        AutoSteerUp:
                AutoSteer = AutoSteer+15 MAX ZeroSteer          g'AutoSteer must be
```

```
                                              lower then ZeroSteer
        RETURN

SteerStraight:
    IF LimtSteer < CntrSteer THEN LimtSteerUp2
        LimtSteer = LimtSteer-20 MIN CntrSteer   'limit steering range during
            acceleration so robot does not spin
        RETURN
    LimtSteerUP2:
        LimtSteer = LimtSteer+20 MAX CntrSteer   'limit steering range during
            acceleration so robot does not spin
        RETURN
```

APPENDIX 3

Sensor Schedule

Pattern sent from the Basic Stamp™	on cable tagged	allows these LEDs to fire	The signal from the detector goes to 74HC165 Chip #	Pin #	and memory variable
00000001	1	A High	3	4	SensrLow.Bit5
		A Low	3	5	SensrLow.Bit6
		a Downward	3	6	SensrLow.Bit7
00000010	2	B High	2	11	SensrHigh.Bit0
		B Low	2	12	SensrHigh.Bit1
		b Downward	2	13	SensrHigh.Bit2
00000100	3	C High	2	14	SensrHigh.Bit3
		C Low	2	3	SensrHigh.Bit4
00001000	4	D High	2	4	SensrHigh.Bit5
		D Low	2	5	SensrHigh.Bit6
		d Downward	2	6	SensrHigh.Bit7

Pattern sent from the Basic Stamp™	on cable tagged	allows these LEDs to fire	The signal from the detector goes to 74HC165		and memory variable
00010000	5	E High	1	11	SensrHigh.Bit8
		E Low	1	12	SensrHigh.Bit9
00100000	6	F High	1	13	SensrHigh.Bit10
		F Low	1	14	SensrHigh.Bit11
		f Downward	1	3	SensrHigh.Bit12
01000000	7	G High	1	4	SensrHigh.Bit13
		G Low	1	5	SensrHigh.Bit14
		g Downward	1	6	SensrHigh.Bit15

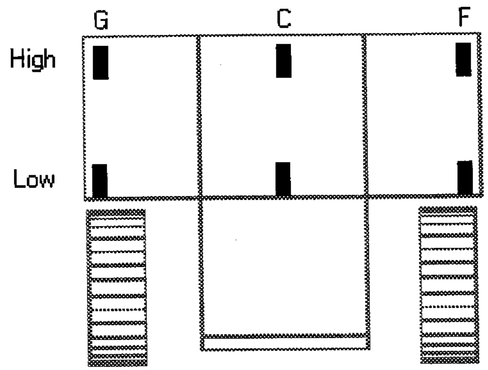


Figure 30. Forward-looking sensors - locations and identifying codes.

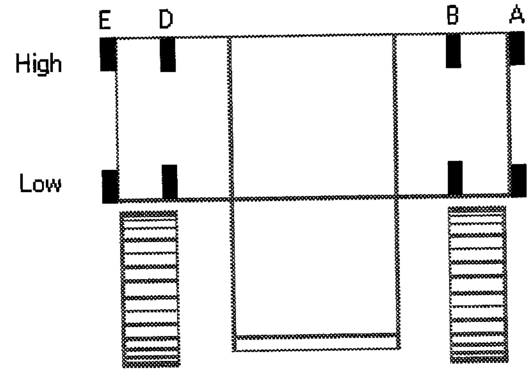


Figure 31. Radial sensors - locations and identifying codes.

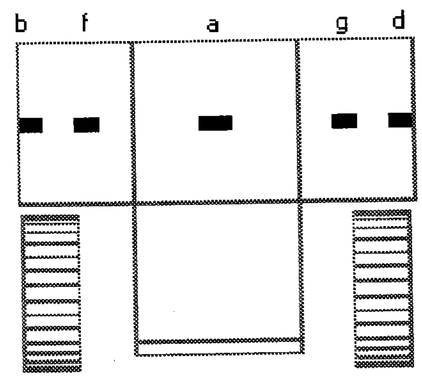


Figure 32. Downward-looking sensors - locations and identifying codes.

APPENDIX 4

Turn Instructions

N.B In this table "any" means either 45° or 90° from straight ahead

Obstacle Configuration	Operator's Preference	Robot's Preference	Action
High Priority Situations			
Front and any left	Ignored	Ignored	Emergency right 90°
Front and any right	Ignored	Ignored	Emergency left 90°
Front only	Right	Ignored	Emergency right 90°
Front only	Left	Ignored	Emergency left 90°
45° or 90° left	Ignored	Ignored	Emergency right 45°
45° or 90° right	Ignored	Ignored	Emergency left 45°
Medium Priority			
Front and any left	Ignored	Ignored	Hard right
Front and any right	Ignored	Ignored	Hard left
Any left and any right	Ignored	Ignored	Emergency right 180°
Front only	None	Left	Hard left
Front only	None	Right	Hard right
Front only	Left	Ignored	Hard left
Front only	Right	Ignored	Hard right
Any left	Ignored	Ignored	Slow right
Any right	Ignored	Ignored	Slow left

Obstacle Configuration	**Operator's Preference**	**Robot's Preference**	**Action**
Low priority			
Front and any left	Ignored	Ignored	Slow right
Front and any right	Ignored	Ignored	Slow left
Front only	None	Left	Slow left
Front only	None	Right	Slow right
Front only	Left	Ignored	Slow left
Front only	Right	Ignored	Slow right
Any left and any right	Ignored	Ignored	No left or right
Any left	Ignored	Left	No left
Any left	Ignored	Right	Slow right
Any right	Ignored	Left	Slow left
Any right	Ignored	Right	No right

BIBLIOGRAPHY

[1] Vijay Kumar, Tariq Rahman, and Venkat Krovi. "Assistive Devices for People with Motor Disabilities." *Wiley Encyclopaedia of Electrical and Electronic Engineering, Assistive Devices for People with Motor Disabilities. Kumar, Rahman, and Krovi. 1997*

[2] Axel Lankenau and Thomas Röfer. "A Versatile and Safe Mobility Assistant." *IEEE Robotics and Automation Magazine. March 2001. pp 29* - 35

[3] G. Bourhis, O. Horn, O. Habert and A. Pruski. "An Autonomous Vehicle for People with Motor Disabilities." *IEEE Robotics and Automation Magazine. March 2001. pp 20* - 28

[4] Edward Cheung and Vladimir J. Lumelsky. "Proximity Sensing in Robot Manipulator Motion Planning: System and Implementation Issues". *IEEE Transactions on Robotics and Automation. Vol 5. No 6. Dec 1989. pp 740 - 751*

[5] Edward Cheung and Vladimir J. Lumelsky. "Real-Time Collision Avoidance in Teleoperated Whole-Sensitive Robot Arm Manipulators". *IEEE Transactions on Systems, Man and Cybernetics. Vol 23. No 1. 1993. pp 194 - 203*

[6] Edward Cheung and Vladimir J. Lumelsky. "A Sensitive Skin System for Motion Control of Robot Arm Manipulators". *Robotics and Autonomous Systems. Vol 9. 1992. pp 9-32*

[7] Homayoun Seraji, Ayanna Howard and Edward Tunstel. "Safe Navigation on Hazard Terrain". *Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Seoul, Korea. May 21-26 2001. pp 3084 - 3091*

[8] Homayoun Seraji, Ayanna Howard and Edward Tunstel. "Fuzzy Rule-Based Reasoning for Rover Safety and Survivability". *Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Seoul, Korea. May 21-26 2001. pp 1413 - 1420*

[9] Homayoun Seraji. "Traversability Index: A New Concept for Planetary Rovers". *Proceedings of the 1999 IEEE International Conference on Robotics and Automation, Detroit, Michigan, USA. May 1999. pp2006 - 2013*