The GuideCane
-
A Computerized Travel Aid for the Active Guidance of Blind Pedestrians

by

Iwan Ulrich

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science

Thesis Committee

Dr. Johann Borenstein (Advisor)
Prof. Daniel Koditschek

The University of Michigan
College of Engineering
Department of Mechanical Engineering and Applied Mechanics
Ann Arbor, August 1997

Research was sponsored by the Whitaker Foundation

To my parents, Hedy and Emil

ACKNOWLEDGMENTS

I would like to take this opportunity and express my gratitude to Dr. Johann Borenstein who gave me the possibility to work on the GuideCane project for the last two years.

Special thanks to Joey Jean and Ivy Zhou for helping build the GuideCane. Also thanks to Patrick Kenny, Chris Minekime, Brian Costanza, Brad Hold and Jim Berry for their help and discussions.

And most important, thanks to Catherine Cartier for all her support and for encouraging me to continue my studies on a different continent.

LIST OF FIGURES

Figure 1: The GuideCane **2**
Figure 2: An obstacle avoidance maneuver **3**
Figure 3: GuideCane interior **8**
Figure 4: Current wheelbase **9**
Figure 5: New proposed wheel-base structure **11**
Figure 6: Brake: a) front view, b) side view **11**
Figure 7: Sonar placement **12**
Figure 8: The GuideCane prototype block diagram **13**
Figure 9: The final GuideCane block diagram **15**
Figure 10: Interface schematic **16**
Figure 11: The MC68HC11A8 block diagram **18**
Figure 12: The main HC11 **19**
Figure 13: The HC11 slave **20**
Figure 14: Write command **22**
Figure 15: Read command **23**
Figure 16: SPI architecture **24**
Figure 17: FIFO bus control for: a) slave 1, b) slave 2 **26**
Figure 18: Timing of quadrature decoder reading **27**
Figure 19: LCD switches **29**
Figure 20: Voltage regulator system **34**
Figure 21: Single battery voltage curve **37**
Figure 22: EERUF example **39**
Figure 23: Conventional system without crosstalk **40**
Figure 24: Conventional system with crosstalk **40**
Figure 25: EERUF system with crosstalk **41**
Figure 26: Servo interior **43**
Figure 27: Servo input: a) without velocity limit, b) with velocity limit **45**
Figure 28: Variable main servo speed limit **45**
Figure 29: Servo input signal (PWM) **48**
Figure 30: Generation of PWM signals: a) regular method, b) better method **49**
Figure 31: Reference system **58**
Figure 32: Virtual scrolling borders **60**
Figure 33: VFH Example: a) obstacle course, b) histogram grid **64**
Figure 34: Polar Histogram **64**
Figure 35: Enlargement angle **68**
Figure 36: Approximation of trajectories: a) without dynamics, b) with dynamics **69**
Figure 37: Example of blocked directions **70**
Figure 38: a) Polar histogram, b) binary polar histogram, c) masked polar histogram **72**
Figure 39: Need for short-term memory **75**
Figure 40: Problematic obstacle shape for a local obstacle avoidance algorithm **79**
Figure 41: Projected trajectories **82**
Figure 42: Effective direction of motion k_p **87**
Figure 43: Necessity of l **89**
Figure 44: VFH* examples **92**

LIST OF TABLES

Table 1: Sonar positions and orientations **12**
Table 2: Command bytes **22**
Table 3: LCD connections **28**
Table 4: Power-on switches **29**
Table 5: Backlight connections **30**
Table 6: Basic power requirements **31**
Table 7: Additional power requirements during development **31**
Table 8: Battery types **32**
Table 9: Battery specifications **33**
Table 10: Battery test results **35**
Table 11: Power system test results **35**
Table 12: Single battery test data **36**
Table 13: Input device types **46**
Table 14: Main HC11 tasks **47**
Table 15: HC11 slave tasks **50**
Table 16: EERUF firing schedule **52**
Table 17: Fire signals **53**
Table 18: Ordered fire signals **54**
Table 19: Look-up table for the first HC11 slave **55**
Table 20: Look-up table for the second HC11 slave **55**
Table 21: VFH* goal depth comparison **93**

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1 The GuideCane 1
1.1.1 System Description 1
1.1.2 Functional Description 2

1.2 Review of Existing Devices 3
1.2.1 The White Cane 4
1.2.2 Conventional Electronic Travel Aids 4
1.2.3 Mobile Robots as Guides for the Blind 5
1.2.4 The NavBelt 5

1.3 Discussion of the GuideCane Concept 6
1.3.1 Guidance Signals versus Obstacle Information 6
1.3.2 Information Transfer 7

2. THE MECHANICAL HARDWARE 8

2.1 The GuideCane 8

2.2 The Wheel-base Structure 9
2.2.1 The Current Structure 9
2.2.2 Proposition of Improved Structure 10

2.3 The Brakes 11

2.4 The Sonar Alignment 12

3. THE ELECTRONIC HARDWARE 13

3.1 The System 13

3.2 The Computer 14
3.2.1 The PC/104 Standard 14
3.2.2 The Cyrix 486 15

3.3 The Electronic Interface 16
3.3.1 The Microcontroller HC11 17
3.3.1.1 The Motorola MC68HC11 Family 17
3.3.1.2 The MC68HC11E2 17
3.3.1.3 The Main HC11 19
3.3.1.4 The HC11 Slaves 20
3.3.2 The PC - HC11 Communication 21
3.3.2.1 The Write Command 21
3.3.2.2 The Read Command 23
3.3.3 The SPI Communication 24
3.3.4 The FIFO 25
3.3.4.1 The HC11 - FIFO Communication 25
3.3.4.2 The PC - FIFO Communication 26

3.3.4.3 FIFO Data Encoding 27
3.3.5 The HCTL Quadrature Decoders 27

3.4 The LCD Screen 28
3.4.1 The Power-Up Sequence 29
3.4.2 The Backlighting 30

3.5 The Power System 31
3.5.1 The Power Requirements 31
3.5.2 The Battery Types 32
3.5.3 The NiMH Batteries 33
3.5.4 The Power Circuitry 33
3.5.5 Power System Evaluation 35

4. THE SENSORS AND ACTUATORS 38

4.1 The Ultrasonic Sensors 38
4.1.1 Properties of Sonars 38
4.1.2 EERUF - Sonar Control 39

4.2 The Steering Angle Measurement 42

4.3 The Encoders 43

4.4 The Servos 44

4.5 The Input Device 46
4.5.1 The Pointer 46
4.5.2 Other Input Devices 46

5. THE HC11 SOFTWARE 47

5.1 The Main HC11 Software 47
5.1.1 Task #1 - Communication and Execution 47
5.1.2 Task #2 - Generation of the PWM Signals 48
5.1.3 Task #3 - Continuous A/D Conversions 49

5.2 The HC11 Slave EERUF Implementation 49
5.2.1 The Multitasking Architecture 50
5.2.2 Task #1 - Communications and Treatment of Buffer 50
5.2.3 Task #2 - The Generation of the Fire Signals 51
5.2.4 Task #3 - Checking for Echoes 51
5.2.5 Task #4 - The Generation of the BINH Signals 51
5.2.6 The Fire Signal Table 52

6. THE PC SOFTWARE 56

6.1 The Main Loop 56

6.2 Odometry 56

6.3 Local Map Building 58

6.3.1 The Map Representation 58
6.3.2 The Map Building - HIMM 59
6.3.3 The Scrolling 60

7. OBSTACLE AVOIDANCE 61

7.1 Original VFH 61
7.1.1 The VFH Algorithm 61
7.1.2 First Stage - The Building of the Polar Histogram 61
7.1.3 Second Stage - Selection of the Steering Direction 65

7.2 VFH+ 66
7.2.1 Threshold with Hysteresis - The Binary Polar Histogram 66
7.2.2 Consideration of the Robot Size 67
7.2.3 Consideration of the Robot Trajectory 69
7.2.4 Cost-Based Direction Selection 72
7.2.5 Performance of the VFH+ Method 77

7.3 VFH* 78
7.3.1 Extremely Local Nature of VFH Method 78
7.3.2 Local Planning 80
7.3.3 A* Search 80
7.3.4 Search Parameters 81
7.3.5 The Expansion Step 81
7.3.5.1 Projection of Position and Orientation 82
7.3.5.1.1 The Projection Equations 82
7.3.5.1.2 The Projection Look-Up Lists 84
7.3.5.2 Cost Function 86
7.3.5.3 Heuristic Function 89
7.3.6 Reducing the Branching Factor 90
7.3.7 Performance of the VFH* Method 92

7.4 Wall Following 94

8. THE DEVELOPMENT ENVIRONMENT 96

8.1 The Tether Environment 96

8.2 The Palmtop Environment 96

8.3 The LCD Environment 97

8.4 The GuideCane Simulator 97

9. FUTURE IMPROVEMENTS 99

9.1 The Compass 99

9.2 Computer Vision 99

9.3 GPS - Global Navigation 100

9.4 Speech Input/Output **100**

9.5 Additional Sonars **101**

10. CONCLUSION 102

1. Introduction

The topic of this Master thesis is the *GuideCane*, a novel device designed to help blind or visually impaired people navigate safely and quickly among obstacles and other hazards. The GuideCane is based on techniques developed at the *University of Michigan Mobile Robotics Laboratory* during the past ten years.

1.1 The GuideCane

This section describes the components of the GuideCane system, and how they are used to provide the desired functional capabilities. The hardware components will be described in more detail in the chapters 2 to 4, while the details of the software will be explained in the chapters 5 to 7.

1.1.1 System Description

Figure 1 shows a user walking with the GuideCane. Much like the widely used *white cane*, the user holds the GuideCane in front of himself[1] while walking. The GuideCane is much heavier than the white cane, but it rolls on wheels that support the GuideCane's weight during regular operation. A pair of wheels are located at the distal end of the GuideCane. A steering servo motor, operating under the control of the GuideCane's built-in computer, can steer the wheels left and right, relative to the cane. An array of ultrasonic sensors is mounted in a semi-circular fashion above the wheel-base. Attached to each wheel is an incremental encoder, which is used by the “onboard” computer to compute (i.e., by means of odometry) the relative motion of the traveler, as well as the momentary travel speed. A miniature pointer, which can be operated by the thumb, allows the user to specify a desired direction of motion.

[1] For the remaining part of this thesis, we will assume that the user is male for the purpose of readability.

Figure 1: The GuideCane

1.1.2 Functional Description

During operation, the user holds the GuideCane in one hand, so that the wheels are on the ground right in front of him. The GuideCane is slightly offset to the side of the hand that holds the cane. The user prescribes a desired direction of motion with the input pointer. This direction is understood to be relative to the current absolute orientation of the GuideCane. For example, if the GuideCane is facing north and the user pushes the pointer forward, then the system would lock into “north” as the desired direction of travel and steer the wheels accordingly. If the user indicated “left” as the desired direction of travel, then the computer would add 90° to the current direction of travel and steer the wheels to the left as soon as this direction is free of obstacles.

While traveling, the ultrasonic sensors detect any obstacle in a 180° wide sector ahead of the user. Using the University of Michigan's previously developed, patented *Error Eliminating Rapid Ultrasonic Firing* (EERUF) method for firing the sonars, in combination with *VFH**, an improved version of UM's patented obstacle avoidance technique called *Vector Field Histogram* (VFH), allows for travel at fast walking speeds [6][9].

These techniques enable the system to instantaneously determine an optimal direction of travel even among densely cluttered obstacles. For example, if the system was "locked" into a desired travel direction of north, but an obstacle blocked the way (see Step 1 in Figure 2), then the obstacle avoidance algorithm would prescribe an alternative direction that would clear the obstacle but would be facing north as close as possible (see Step 2 in Figure 2).

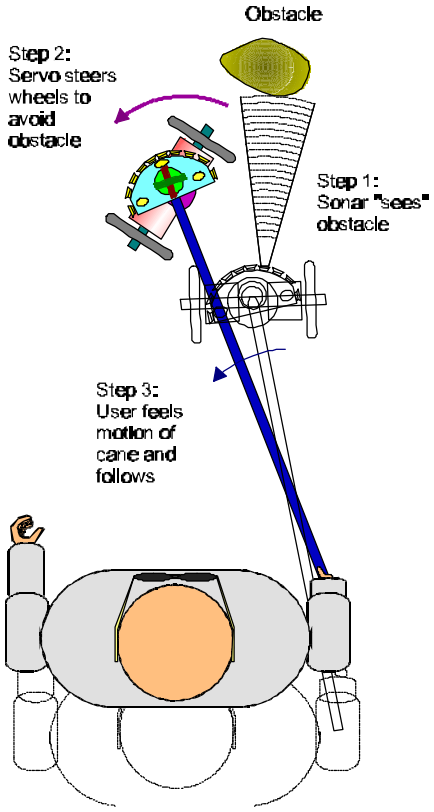


Figure 2: An obstacle avoidance maneuver

Once the wheels begin to move sideways to avoid the obstacle, the user feels the resulting horizontal rotation of the cane (see Step 3 in Figure 2). In a fully intuitive response, requiring virtually no training time, the traveler changes his orientation to align himself with the cane at the "nominal" angle. In practice, the user's walking trajectory follows the trajectory of the GuideCane similar to the way a trailer follows a truck. Once the obstacle is cleared, the wheels steer back to the original desired direction of travel. The new line of travel will be offset from the original line of travel. Depending on the circumstances, the traveler may wish to continue walking along this new line of travel, or the system can be programmed to return to the original line of travel. This latter option is made possible by the full odometry capability provided by the wheels and the attached encoders.

One particularly difficult problem for blind pedestrians is that of stairs. The GuideCane offers separate solutions for down-steps and up-steps. Down-steps are detected in a fail-safe manner: when a down-step is encountered, the wheels of the GuideCane drop off the edge – without a doubt a signal that the user can not miss. Up-steps can be detected by additional front-facing sonars that are not implemented yet. A simple method for detecting up-steps is described in [10].

1.2 Review of Existing Devices

1.2.1 The White Cane

The most successful and widely used travel aid for the blind is the *white cane*. This mechanical device is used to detect obstacles, uneven surfaces, borders, holes, steps and other hazards or clues. The white cane is inexpensive and is so lightweight and small that it can be folded and tucked away in a pocket. However, users must be trained in the use of the white cane over periods of hundred hours which is a substantial "hidden" cost. An inconvenience of the white cane is that the user is required to actively scan the small area ahead of him. In addition, the white cane is not well suited to detect potentially dangerous obstacles at head level.

1.2.2 Conventional Electronic Travel Aids

During the past three decades, several *electronic travel aids* (ETAs) were introduced that aimed at improving the blind users' mobility in terms of safety and speed. These more high-tech devices have been on the market for many years but appear to lack utility, and, consequently, are not widely used [4].

The *C-5 Laser Cane* – was introduced by Benjamin et al. [2]. It is based on optical triangulation with three laser diodes and three photo-diodes as receivers. The *Laser Cane* can detect obstacles at head-height, drop-offs in front of the user, and obstacles up to a range of 1.5 m or 3.5 m ahead of the user.

The *Mowat Sensor* – is a hand-held ultrasonic-based device that informs the user of the distance to detected objects by means of tactile vibrations [38]. The frequency of the vibration is inversely proportional to the distance between the sensor and the object.

The *Nottingham Obstacle Detector (NOD)* – is a hand-held sonar device that provides auditory feedback, in which eight discrete levels of distance are distinguished by different musical tones [3].

The *Binaural Sonic Aid (Sonicguide)* – comes in the form of a pair of spectacle frames, with one ultrasonic wide-beam transmitter mounted between the spectacle lenses and one receiver on each side of the transmitter [20]. Signals from the receivers are frequency shifted and presented separately to the left and right ear. The resulting interaural amplitude difference allows the user to determine the direction of an incident echo and thus of an obstacle. The distance to an object is encoded in the frequency of the demodulated low-frequency tone.

Three fundamental shortcomings can be identified in all ETAs discussed in the foregoing sections:

1. The user must actively scan the environment to detect obstacles (no scanning is needed with the *Sonicguide*, but that device does not detect obstacles at floor level). This procedure is time-consuming and requires the traveler's constant activity and a conscious effort.

2. The traveler must perform additional measurements when an obstacle is detected in order to determine the dimensions of the object. The user must plan a path around the obstacle – Again, a time-consuming, conscious effort that reduces the walking speed.

3. One problem with all ETAs based on acoustic feedback is their interference (called *masking*) with the blind person's ability to pick up environmental cues through hearing [12][20][25].

1.2.3 Mobile Robots as Guides for the Blind

In general, one could argue that any mobile robot with obstacle avoidance can be used as a guide for the blind. However, mobile robots are inherently unsuited to the task of guiding a pedestrian. The foremost limitation of mobile robots is that they are large, heavy, and incapable of climbing up or down stairs or boardwalks. This approach would actually burden the blind person with the additional, severe handicap of limited mobility.

Another problem of this approach is that the speed of the robot can make the user feel uncomfortable, pulling a cautious user or slowing a confident user unnecessarily down. To overcome this problem, an additional interface function would be needed with which the user could indicate the desired speed to the robot. However, with the GuideCane, the user is in direct control of the speed so that it is much more intuitive and much easier to use.

1.2.4 The NavBelt

During the past six years, the University of Michigan Mobile Robotics Laboratory has conducted active research in applying mobile robot obstacle avoidance technologies to assistive devices for the handicapped. In 1989, the concept of the *NavBelt* was developed. The NavBelt is a portable device equipped with ultrasonic sensors and a computer. A prototype of this system was built and tested [31].

The NavBelt provided two modes of operation:

1. In the *image mode*, the NavBelt produced a 120° wide view of the obstacles ahead of the user similar to a radar screen image. This image was then translated into a series of directional (stereophonic) audio cues through which the user could determine which directions were blocked by obstacles and which directions were free for travel. The problem with this method lay in the fact that a considerable conscious effort was required to comprehend the audio cues. Because of the resulting slow response time, our test subjects could not travel faster than roughly 0.3 m/sec. And even this marginal level of performance required hundreds of hours of training time.

2. Another mode of operation was called *guidance mode*. In this mode, it was assumed that the system knew the traveler's momentary position and the traveler's desired target location. Under these conditions, the NavBelt only needed to generate a single (thus, low-bandwidth) signal that indicated the recommended direction of travel. It was much easier to follow this signal, and walking speeds of 0.6 - 0.9 m/sec were achieved. The main problem was that in reality the system would not know the user's momentary position, as required by the guidance mode.

1.3 Discussion of the GuideCane Concept

The GuideCane is unique in its ability to physically direct the user around obstacles and toward a user-prescribed target. Indeed, the uniqueness is twofold:

1.3.1 Guidance Signals versus Obstacle Information

Existing ETA's are designed to notify the user of obstacles, usually requiring the user to perform some sort of scanning action. Then, the user must evaluate all of the obstacle information, which comprises of the size and proximity of each obstacle, and choose a suitable travel direction. In sighted people, such relatively high bandwidth information is processed almost reflexively, usually without the need for conscious decisions. Nature had millions of years to perfect this skill. However, the evaluation of obstacle information presented acoustically is a new skill that must be acquired over hundreds of hours of learning. Even then, using such a skill takes a great deal of conscious effort, and thus processing time. The required effort further increases with the number of detected obstacles.

The GuideCane is fundamentally different from other devices in that it “views” the environment and computes the momentary optimal direction of travel. The resulting guidance signal is a single piece of information – a direction – which means that the bandwidth of the information is much smaller. The consequence is that it is far easier, safer, and faster to follow the low-bandwidth guidance signal of the GuideCane than to follow the high-bandwidth information of other existing systems.

1.3.2 Information Transfer

In prior research with the NavBelt, different methods of using binaural (stereophonic) signals to guide the user around obstacles were tested. Test users found that it is generally very difficult to recognize and react to such signals at walking speed. Even after nearly 100 hours of training, the Ph.D. student who conducted this research could not walk safely at walking speed.

By contrast, tests have shown that it is much easier and more intuitive to follow the GuideCane. As an initial test, before obstacle avoidance was implemented, a radio-controlled joystick receiver was installed inside the sensor head, which allowed a sighted assistant to steer the GuideCane remotely. A sightless person could then walk with the GuideCane, "steered" by the assistant. As expected, this test showed that any subject could immediately follow the GuideCane at walking speed and among densely cluttered obstacles. This test verified the key-hypothesis, namely, that following the GuideCane's path was completely intuitive, even at fast walking speed.

This success can be credited to another unique feature of the GuideCane: Information transfer through direct physical force. This process is completely intuitive, which means that anybody can use the system immediately and without learning how to interpret artificially defined acoustic or tactile signals (as with existing ETAs). Yielding to external forces is a reflexive process that does not require a conscious effort. Moreover, most visually impaired people are used to being guided by other people in a very similar way.

Even though the GuideCane's wheels are unpowered, the GuideCane can apply a substantial amount of physical force F_d on the user if he fails to respond to a change of direction. This force is the result of the sideways motion of the wheels when avoiding an obstacle. The resulting rotation of the cane forces a clearly noticeable rotation of the hand that holds the near end of the cane.

A second force, immediately noticeable after the guide wheels change their orientation (but even before the user feels the rotation of the cane), is the increased reaction force that is opposed to pushing the cane forward. When walking while the cane and the wheels are aligned, the user must only overcome the reactive force F_r resulting from the friction in the bearings and the roll resistance of the wheels. If the wheels steer an angle θ in either direction, then the traveler has to push the cane with a higher force F_p in order to overcome the reactive force of the wheels:

$$F_p = \frac{F_r}{\cos\theta} \geq F_r \qquad \text{with} \quad \theta \in \left]-\pi/2, \pi/2\right[$$

This change in reactive force is immediately felt by the user and prepares him for an upcoming steering maneuver.

The user also feels the torque of the main servo turning the wheel-base. However, the force required to counteract the servo torque is relatively small due to the distance between the main axis and the user.

2. The Mechanical Hardware

2.1 The GuideCane

The GuideCane must be as light as possible, so that the user can easily lift it up, e.g. for going up- or downstairs. For this reason, aluminum was chosen to be the material for most of the parts that are subject to forces. Acrylic, a plastic, is used for the light housing. In a commercial version of the GuideCane, the plastic housing could consist of only two pieces produced by injection molding or vacuum forming. All of the electronics are placed inside the acrylic housing for protection. The part that holds eight sensors in a 120° arc was heat treated and then formed into an arc.

The GuideCane is currently equipped with 10 Polaroid ultrasonic sensors. The sensor devices are held in place by rubber O-rings. Each sonar is connected to a small interface which is in turn connected to the main interface. The ten interfaces are split up into two stacks of five, placed in the two front corners. Lightweight plastic spacers are used to stack and insulate them. To reduce the effects of electronic crosstalk, the connections between the interfaces and sonars are made so that the length of the wires is minimized. Furthermore, these wires are placed as far apart from each other as possible.

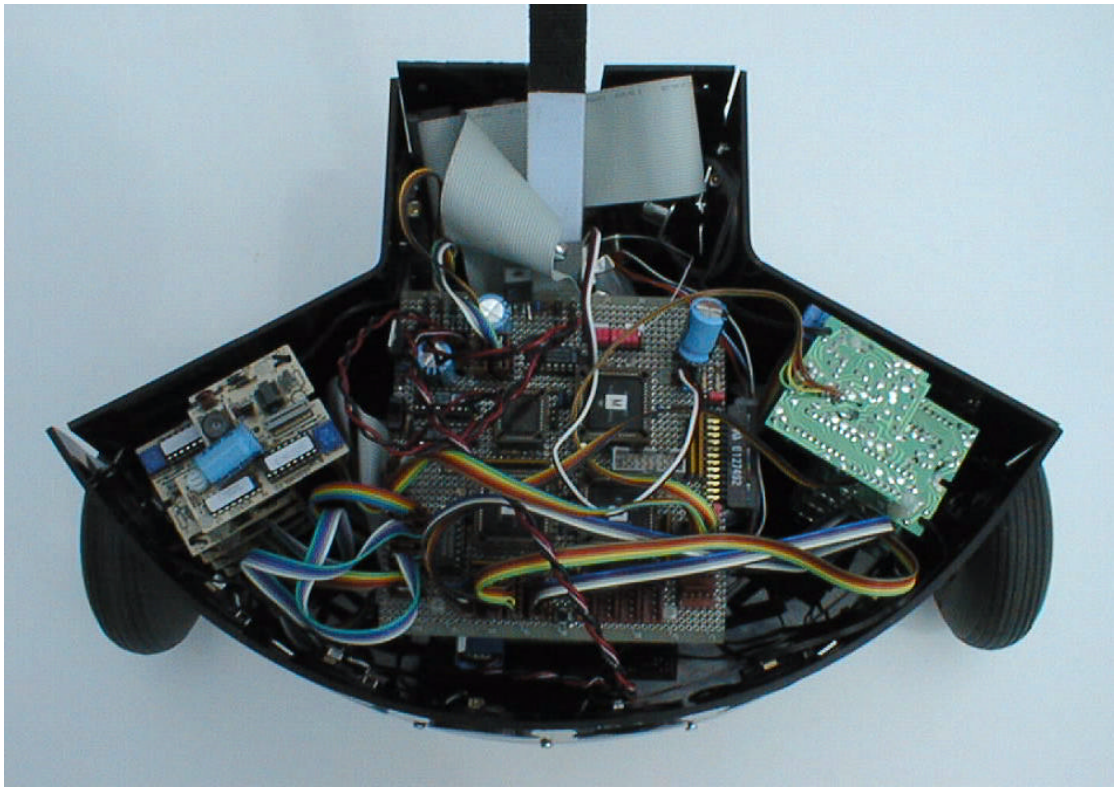


Figure 3: GuideCane interior

The aluminum cane is attached to a hinge which is fixed to a pivot screw. In this way, the paths of all forces go through the metal parts and not through the plastic of the housing. The angle of the cane can be adjusted by a wing nut and a thumb screw to accommodate users of different heights. The two aluminum brackets for the adjustment screw are attached to the housing base. To reduce the stress in the plastic when the GuideCane is lift up, the connection between the bracket fixations and the pivot screw is strengthened with an aluminum plate.

2.2 The Wheel-base Structure

2.2.1 The Current Structure

The wheel-base consists of an aluminum bracket which is reinforced by an L-shaped steel bracket. Each wheel is attached to the wheel-base by two ball bearings. Each wheel axis is also equipped with an encoder, allowing the GuideCane to perform odometry as explained in section 6.3. A photo of the wheel-base is shown in Figure 4:

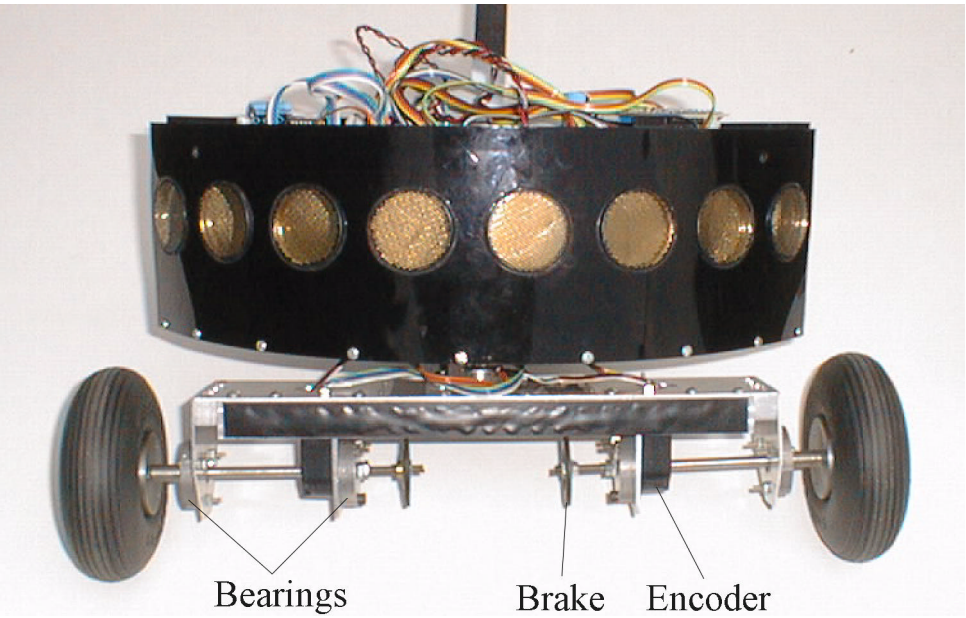


Figure 4: Current wheelbase

The wheel-base is attached to the GuideCane housing by a pivot screw, consisting of a ball bearing and an axis with one threaded end. The main servo is fixed to the housing bottom. To reduce the risk of damaging the main servo by an object, the servo is placed behind the wheel-base. The wheel-base is coupled to the main servo through a push-rod. One end of the push-rod is connected to the servo arm through a steel clevis while the other end is connected to the wheel-base through a ball link.

Unfortunately, the wheel-base design is not optimal. The main problem is its high moment of inertia, which requires a very strong servo to rotate it without oscillating. Strong servos are big and consume a lot of power, which is a disadvantage in a battery powered system. To reduce the wheel-base moment of inertia, several holes were drilled into the reinforcement bracket and the two attachments located at its extremities.

Another problem is the possibility of very strong mechanical shocks on the servo shaft. When a wheel bumps into an obstacle, e.g. a small irregularity in the ground that can not be detected by the sonars, this shock translates into a very high torque on the servo shaft because of the long wheel-base.

Furthermore, as the GuideCane is placed on only two wheels, the user has to hold the cane so that the mobile robot stays horizontal. When the wheels are turned to either side, the user has to apply a torque to keep the mobile robot from tipping over. The amount of this torque is reasonable, however, it could become stressful if the GuideCane is used for more than 15 minutes.

2.2.2 Proposition of Improved Structure

To overcome these problems, a new wheel-base structure is proposed as shown in Figure 5. With this configuration, the GuideCane is equipped with three wheels. Two of the wheels are attached to the base by ball bearings. Again, each of these wheels is equipped with an encoder. The third wheel is rotated by a servo. This new configuration solves all the above mentioned problems.

First of all, only the third wheel is rotated by the servo. Moreover, no large moment is required to turn this wheel as it is directly in line with the servo axis of rotation. As a result, a much weaker, smaller, and less power consuming servo can be used.

Secondly, the rotational shocks are reduced by at least a factor of 3.5. In the worst case, the distance between the point of contact and the servo axis is only half the wheel diameter.

Finally, as the GuideCane is placed on three wheels, it is inherently stable. This is not only much more comfortable for the user, but it also guarantees that the GuideCane stays always horizontal.

Unlike in the current configuration, the cane can not be fixed rigidly to the robot. To guarantee that all three wheels stay in contact with the ground, the cane must have a vertical degree of freedom. It turned out that this is a very nice feature, as the angle of the cane adapts automatically to the height of the user. So, the mechanism for the cane angle adjustment is not necessary anymore. The cane fixation must also have a horizontal degree of freedom. Otherwise, the user could override the direction of motion. The angles of these two degrees of freedom must be limited mechanically so that the GuideCane can easily be lift up and put down.

To test these hypotheses, a simple prototype was built similar to the one described in section 1.3.2. This prototype showed that it was still very intuitive to follow it. With the angle limitations on the cane fixation, it is no problem to lift it up and put it down.

The most striking feature was the fact that it is much more comfortable to hold this prototype. One reason is that the robot is firmly placed on three wheels, so that the user does not have to apply any torques at all to balance it. Another reason is the vertical degree of freedom which adapts to vertical movements of the user's hand.

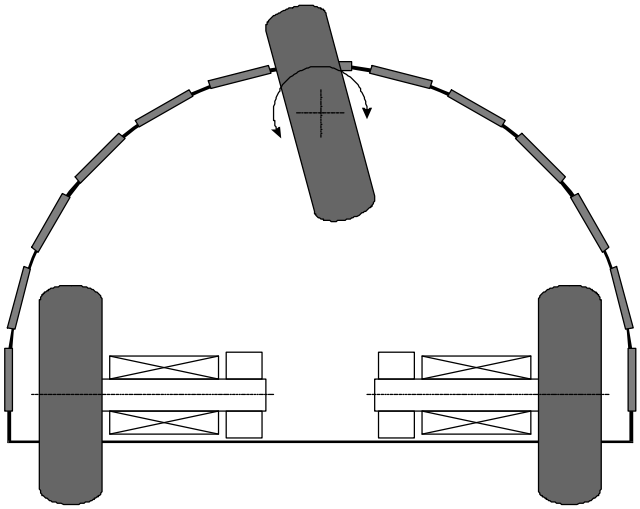


Figure 5: New proposed wheel-base structure

2.3 The Brakes

Both wheels are equipped with a brake, so that the GuideCane can slow the user down in densely cluttered environments or even stop him in a dead-end. Each brake consists of a small servo, a plug pin, and a hard rubber disk as shown in Figure 6:

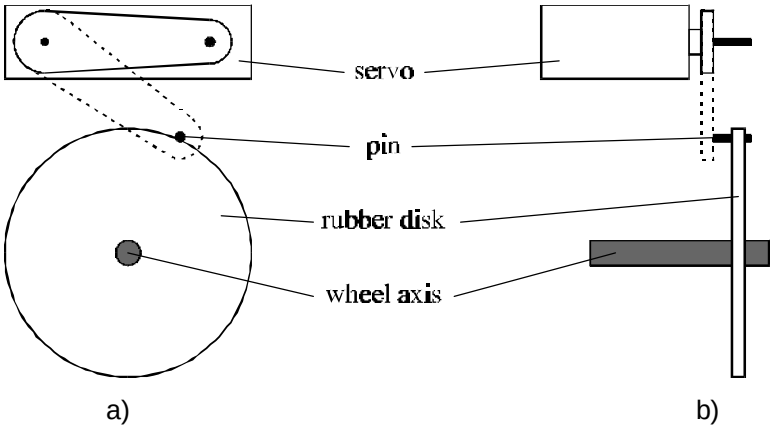


Figure 6: Brake: a) front view, b) side view

2.4 The Sonar Alignment

The current version of the GuideCane is equipped with ten sonars as shown in Figure 7. Eight sonars are placed at the front of the GuideCane on an arc with a 20 cm radius. The center of the arc lies 6.5 cm behind the center of the GuideCane which is defined as the center of the main axis. The angular spacing between these sonars is 15°, which assures complete coverage as agreed by most mobile robotics researchers. These sonars cover the area ahead of the GuideCane with an angular spacing of 120°. The other two sonars, facing to both sides, are very useful for wall following and for going through narrow openings.

The positions and the directions of the sonars, relative to the center of the GuideCane are summarized in Table 1. These values are stored in a table for faster software execution.

Sonar #	1	2	3	4	5	6	7	8	9	10
Position x [mm]	21	58	84	98	98	84	58	21	-100	-100
Position y [mm]	158	121	76	26	-26	-76	-121	-158	70	-70
Orientation	52.5°	37.5°	22.5°	7.5°	-7.5°	-22.5°	-37.5°	-52.5°	90°	-90°

Table 1: Sonar positions and orientations

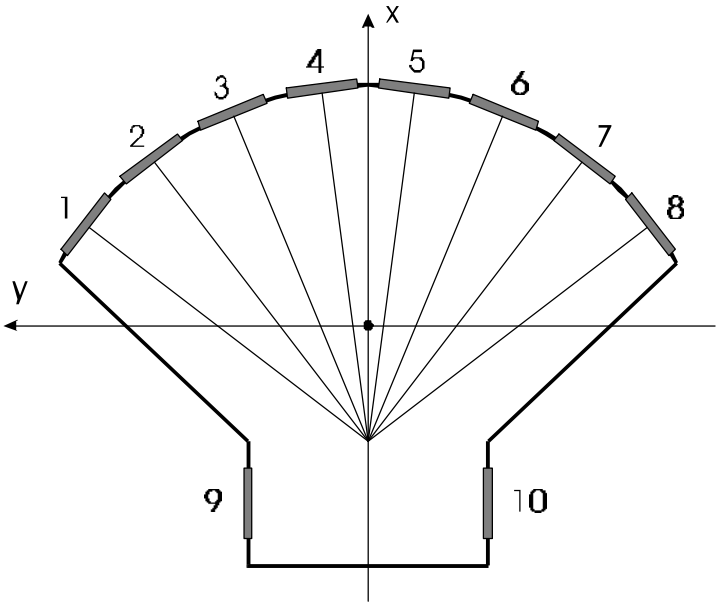


Figure 7: Sonar placement

3. The Electronic Hardware

3.1 The System

The electronic architecture of the GuideCane is shown in Figure 8. The main brain of the GuideCane is a PC/104 486 running at 33 MHz. The main advantage of using a PC instead of a customized microcontroller board is its development environment. The PC can easily be hooked up to a CRT monitor and a keyboard. It can even be connected to a LCD display as explained in section 8.3. The PC beeper can be used to give acoustic clues to either the user or the developer. The first serial port is used by the compass, while the second serial port is used by the input pointer. The currently used pointer can easily be replaced by a different input device as most of these devices connect to the serial port.

The PC is connected to the main interface through its bi-directional parallel port. This board serves as the interface between the PC and the sensors (encoders, sonars, potentiometer) and the actuators (main servo and brakes). This interface allows the PC to get information from the different sensors. It also generates the specific control signals for the sonars and servos. The interface consists of three MC68HC11E2 microcontrollers, two counters and some logic devices. In short, the GuideCane is an embedded system equipped with four microprocessors working fully in parallel.

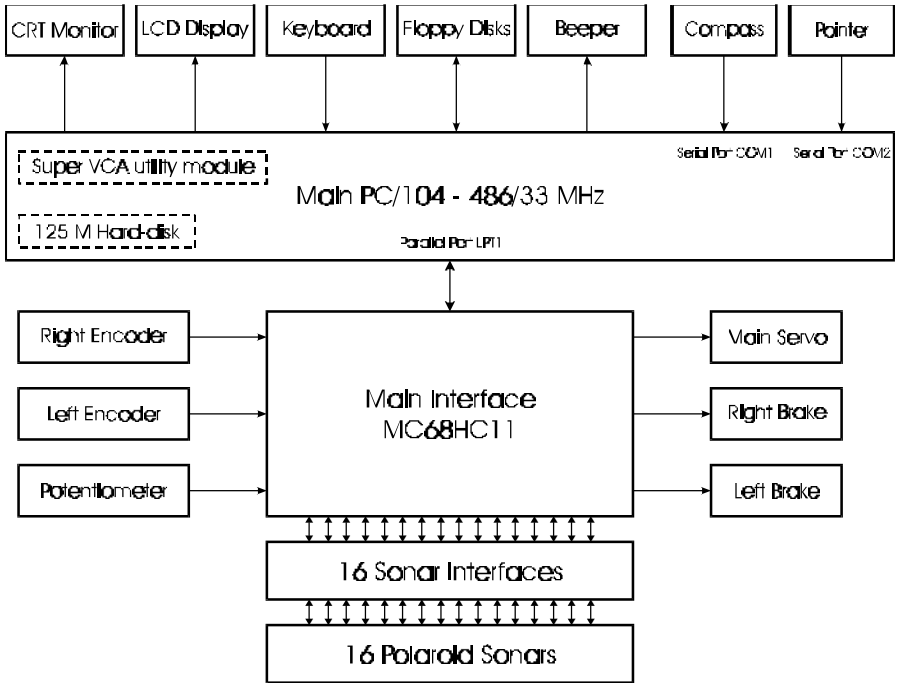


Figure 8: The GuideCane prototype block diagram

The multiprocessor interface executes many time-critical tasks which would take too many resources if executed by the main PC. The interface receives commands from the PC and then takes care of the sensors and actuators without any further involvement of the PC. Most of the sensor data is preprocessed by the board before being communicated to the PC. This concept minimizes not only the communications between the PC and the board, but it also minimizes the computational power required by the PC to control the sensors and actuators. So, most of the PC's computational power can be used for medium and high level software.

3.2 The Computer

3.2.1 The PC/104 Standard

The main brain of the GuideCane is an embedded PC/104 computer. PC/104 is a relatively new standard defining the electrical and mechanical specifications for a compact version of the IEEE P996 (PC and PC/AT) bus [29]. These specifications have been optimized for the unique requirements of embedded system applications. The main features of this standard are:

- Small size: 3.550 x 3.775 inches.
- Connectors: 64 and 40 pin male/female headers replace standard PC edge connectors.
- Stacks: Backplanes and card cages are eliminated by a self-stacking bus. Stacked modules are just 0.6 inches apart.
- Power: Most signals have reduced bus drive of 4-6 mA which reduces the power consumption (1-2 W per card) and heat dissipation.

These features make the PC/104 standard very attractive for embedded applications. PC/104 controllers combine high computing power with excellent resistance to contamination, shock, and vibration.

Thousands of products and utility packages are available from over hundred manufacturers. Some of these packages are especially interesting for the GuideCane, e.g. the GPS module. And as a PC/104 offers full architecture, hardware and software compatibility with the PC bus, all widely available products for regular PC's can also easily be integrated.

3.2.2 The Cyrix 486

The current stack consists of three PC/104 modules:

1. Main CPU: Cyrix 486 at 33 MHz with math co-processor and 4 MB DRAM.
2. Super VGA utility module.
3. 125 MB Hard-disk.

However, two of the three modules are only required during development. It is obvious that the VGA utility module is unnecessary after development. In addition, even the hard-disk module can be eliminated. Once the software is completed, it can be stored in an EPROM that can be placed in the empty socket U10 of the main CPU board. In addition to eliminating one module, this also eliminates potential problems due to the moving parts in the hard-disk. Being a solid state device, the EPROM is much less sensitive to shocks and vibrations than the hard-disk is.

Therefore, the proposed final version of the GuideCane will consist of only the main CPU module which is not only a very compact and inexpensive solution but also requires less power than the current stack. The proposed final architecture of the GuideCane is shown in Figure 9.

If it turns out that the chosen main CPU is too slow, it could be replaced by a more powerful module. There are currently commercially available PC/104 modules that are equipped with a 586 processor running at 133 MHz.

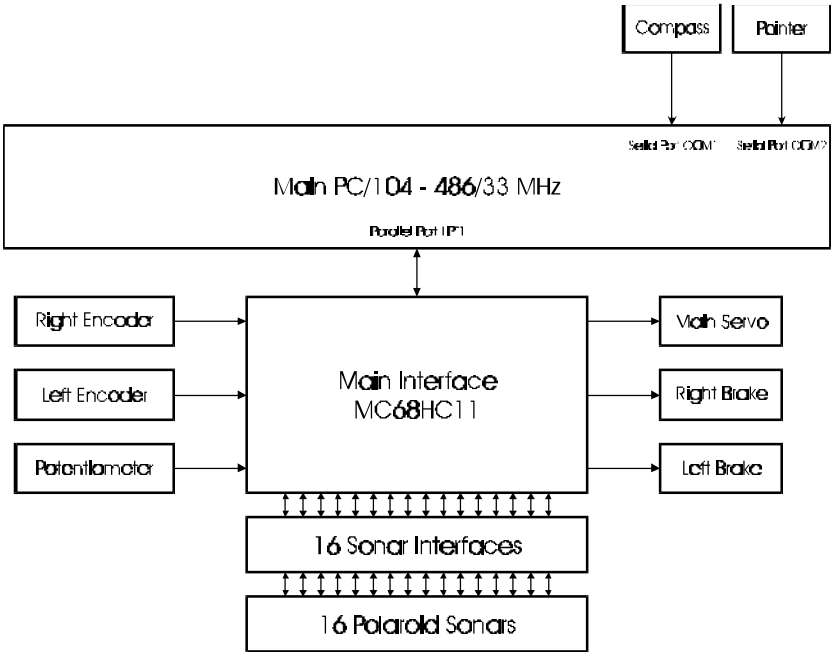


Figure 9: The final GuideCane block diagram

3.3 The Electronic Interface

The architecture of the electronic multiprocessor interface is shown in Figure 10. The communication between the PC and the interface is based on the parallel port of the PC. The parallel port consists of an 8-bit bi-directional data bus, 4 digital outputs and 5 digital inputs. In the current version of the interface, all but 2 digital inputs are in use. The 8-bit bi-directional data bus together with the 4 digital outputs and an additional 2-4 decoder allows the PC to communicate with the main HC11, the FIFO and the two HCTL quadrature decoders. Two of the digital inputs are used for the handshaking with the HC11. Another digital input is used to supervise a FIFO flag.

This bus architecture permits parallel communication at high speed. Furthermore, most communications are buffered to minimize delays.

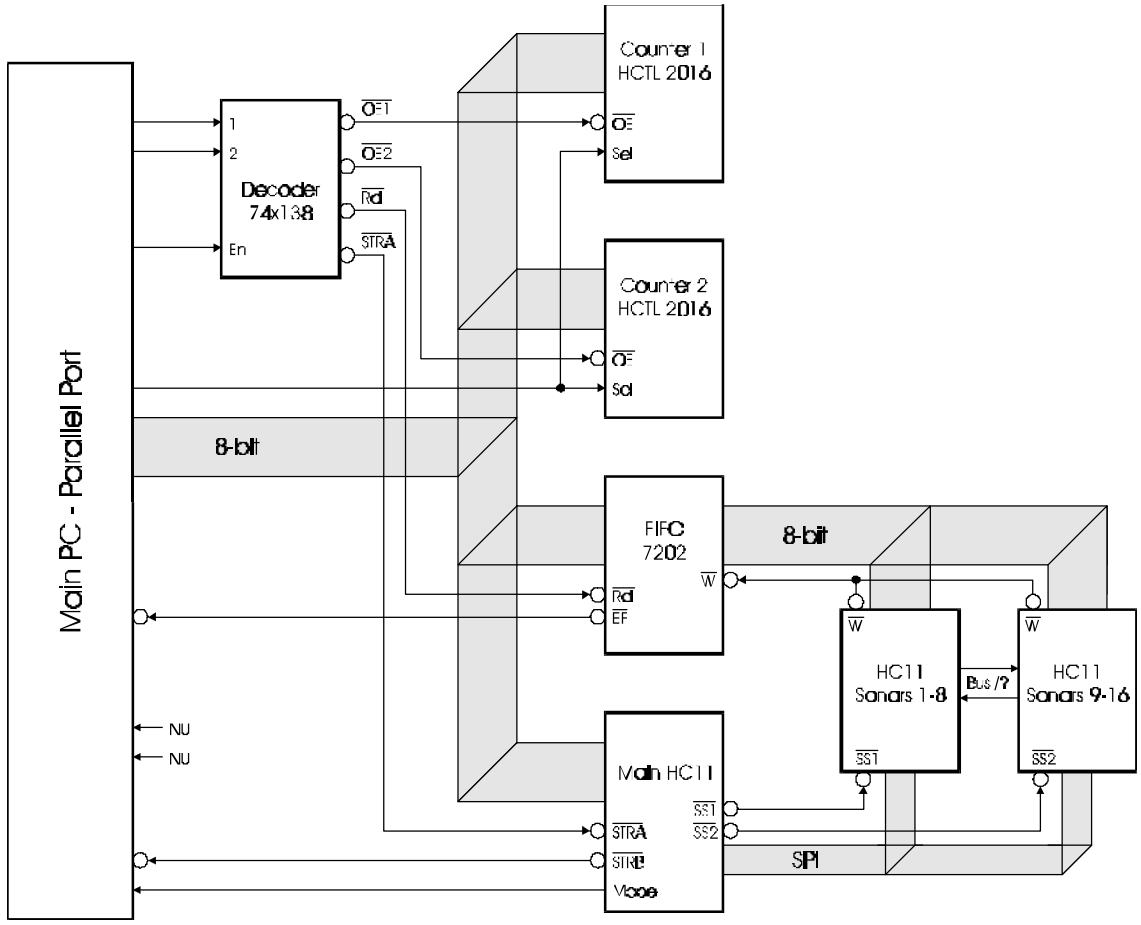


Figure 10: Interface schematic

3.3.1 The Microcontroller HC11

3.3.1.1 The Motorola MC68HC11 Family

The Motorola MC68HC11 family consists of a wide range of advanced 8-bit microcontrollers [28]. A microcontroller is defined as a microprocessor with memory and major peripheral functions on the same chip. Microcontrollers are especially suited for embedded applications. The block-diagram of a typical MC68HC11 member is shown in Figure 11 [28].

The peripheral functions of all MC68HC11 members include:

- Eight-channel A/D converter with an eight bit resolution.
- Asynchronous serial communications interface (SCI).
- Synchronous serial peripheral interface (SPI).
- 16-bit timer with three input-capture lines, five output-compare lines and a real-time interrupt function.
- An 8-bit pulse accumulator can count external events or measure external periods.
- Self-monitoring circuitry: watchdog, clock monitor system and illegal opcode detection.
- Two software-controlled power-saving modes, WAIT and STOP, are available to conserve power.

3.3.1.2 The MC68HC11E2

The difference between the MC68HC11 family members lies mainly in the amount of their different kinds of memories. A very popular member in the robotics community is the MC68HC11E2 which has 2 K of EEPROM, more than any other family member. This was the main reason for the selection of the MC68HC11E2.

The advantage of having that much EEPROM is that it allows one to store reasonably sized programs. Two kilobytes of program space may not seem much to the regular programmer, but it is mostly enough for microcontroller-type programs written in assembly. As the HC11[2] bus runs at only 2 MHz and EEPROM space is limited, it is preferable to program it in assembly instead of a higher level language. Fortunately, the HC11 has a very nice instruction set which is one of the most orthogonal ones.

As the software can be stored in the internal EEPROM, the microcontroller can be used in single-chip mode with no external memory.

[2] For the remaining part of this thesis, the term MC68HC11E2 will be abbreviated by HC11.

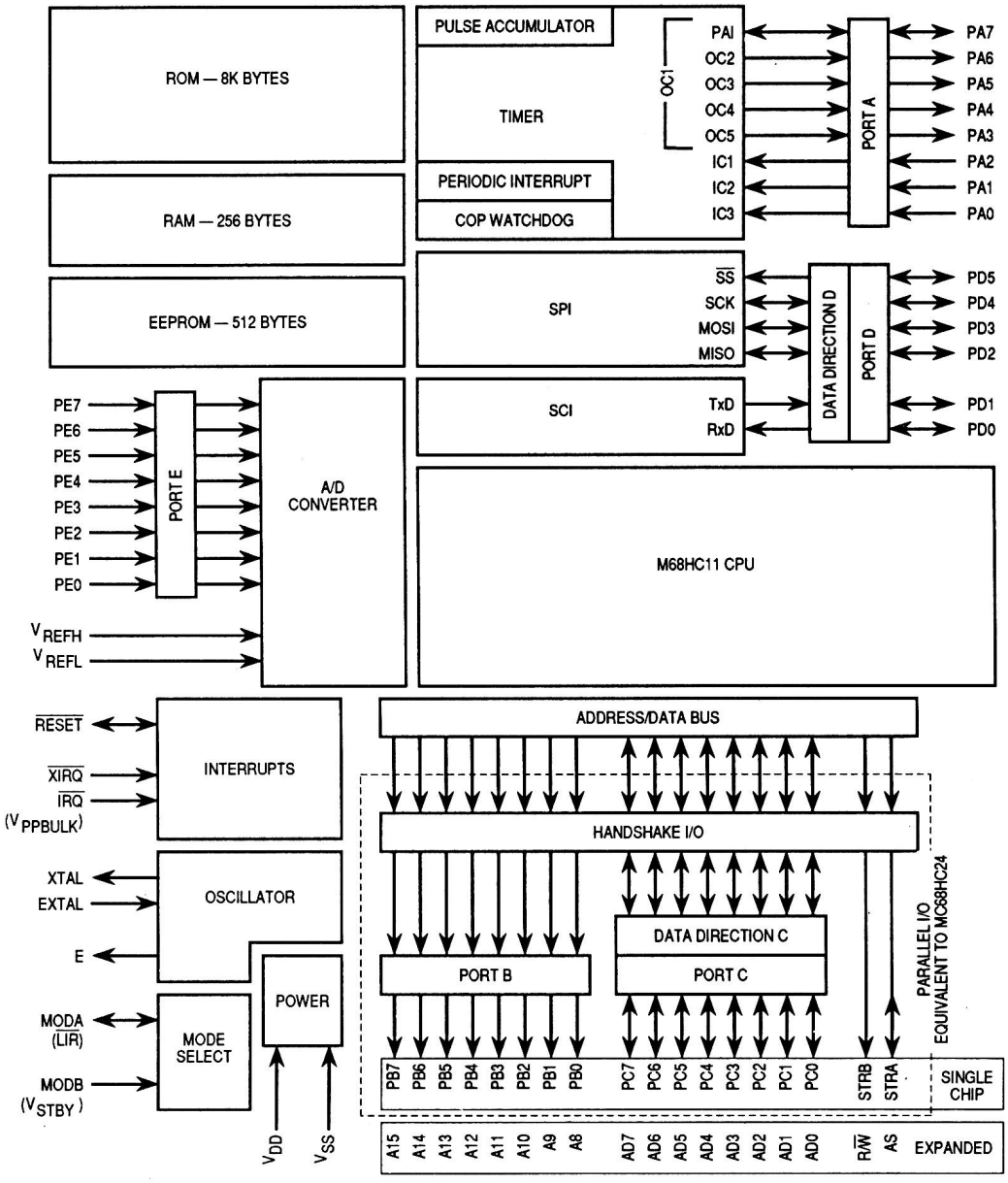


Figure 11: The MC68HC11A8 block diagram

3.3.1.3 The Main HC11

A simplified diagram of the main HC11 is shown in Figure 12. Port A is used to generate the PWM signals for up to four servos. Three of the four unused pins on this port are input capture lines available for future extensions. The fourth currently not used pin is the pulse accumulator input.

Port B is used to independently reset the other logic devices on the interface, to select a SPI slave, and for hand-shaking with the PC. Port C together with the *R/W* and *AS* lines is used to communicate with the PC over its bi-directional parallel port.

The asynchronous serial communications interface (SCI) of Port D is used to download the program from the PC into the HC11's internal EEPROM. For future extensions, a standard serial port device can be connected to this port.

The synchronous serial peripheral interface (SPI) is used to communicate with the two HC11 slaves as explained in section 3.3.3. Additional SPI devices can also be easily added to the current architecture.

Port E is the eight-channel A/D converter with eight bits of resolution. Currently only one of the eight analog inputs is used by the main servo potentiometer.

The *E* signal, a clock output running at 2 MHz is used to drive the quadrature decoder devices as explained in section 3.3.5.

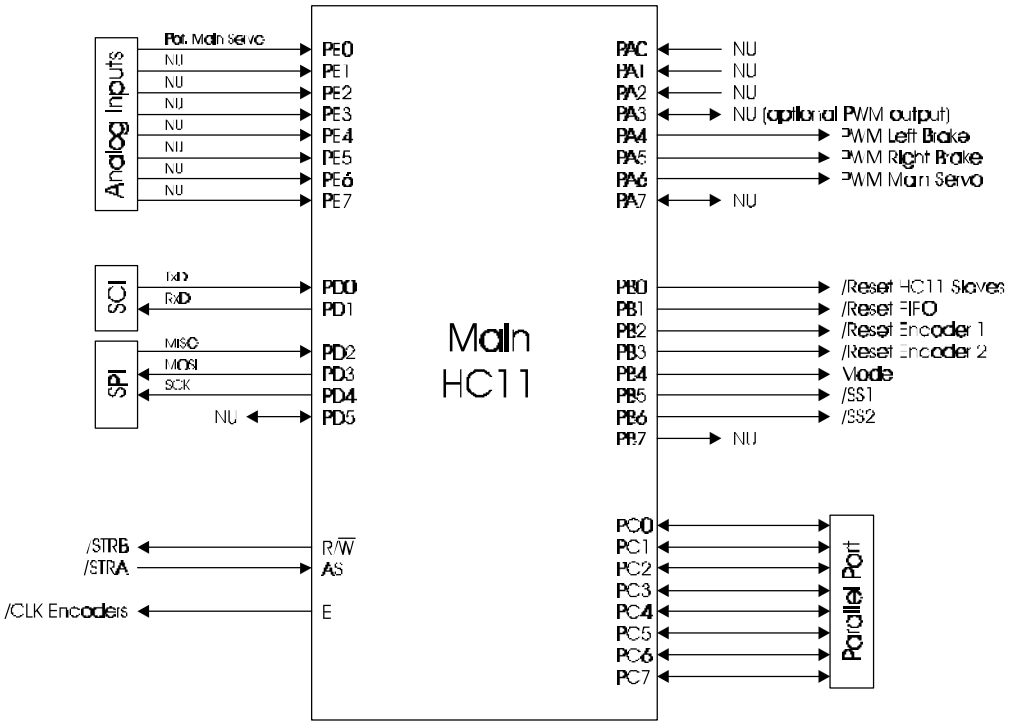


Figure 12: The main HC11

3.3.1.4 The HC11 Slaves

A simplified diagram of a HC11 slave is shown in Figure 13. Both slaves are identically integrated into the hardware system. However, there are a few minor differences in their software. For reasons of limited input/output lines and speed, each HC11 slave takes care of eight sonars.

Four outputs of Port A together with a double 2-4 decoder are used to generate the eight BINH signals for the sonars. Port B is used to generate the fire signals for the sonars. Port C together with *PA3* is used in an open-collector mode to write the sonar results into the FIFO. Port D is used the same way as for the main HC11. The *R/W* and *PA0* signals are used to synchronize the access to the FIFO bus.

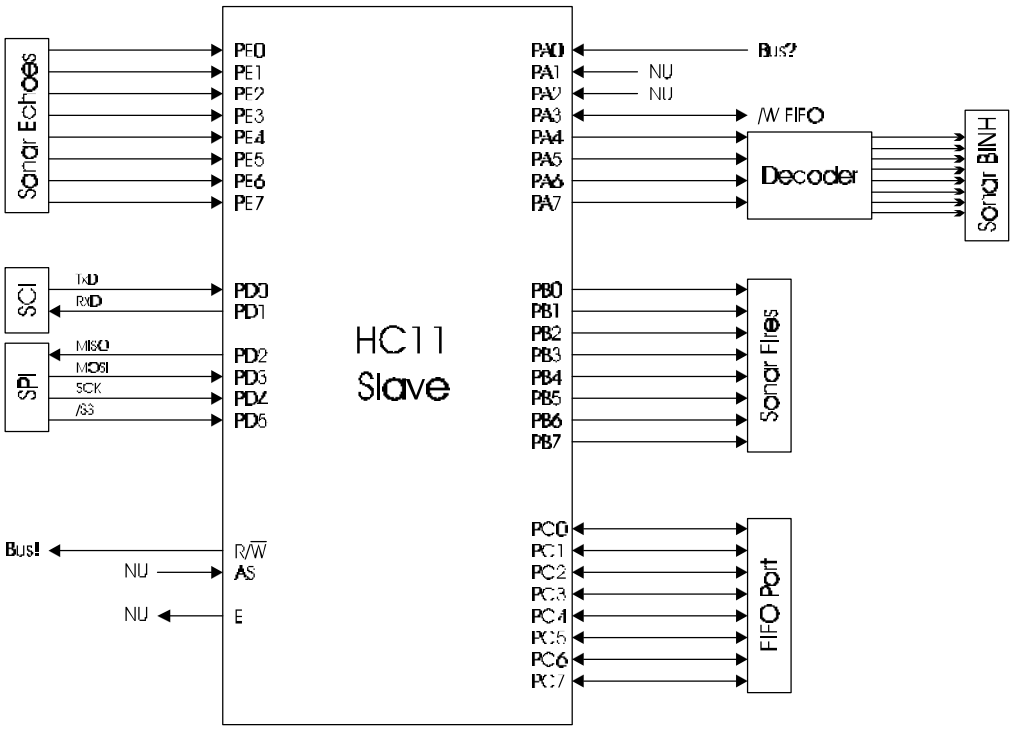


Figure 13: The HC11 slave

3.3.2 The PC - HC11 Communication

The communication between the PC and the HC11 microcontrollers is done through the main HC11 and the FIFO. The communication between the PC and the main HC11 is based on handshaking supported by the HC11 hardware. The procedure of sending a command from the PC to the main HC11 is explained in section 3.3.2.1. The procedure of reading from the main HC11 is explained in section 3.3.2.2.

The PC can not communicate directly with the two HC11 slaves. To send a command to the HC11 slaves, the PC sends the command to the main HC11 which will then send it to the HC11 slaves through the SPI as explained in section 3.3.3. To read the sonar data from the HC11 slaves, the PC reads the buffered data from the FIFO as explained in section 3.3.4.

3.3.2.1 The Write Command

The time diagram for a write command is shown in Figure 14. The *STRB* signal is automatically generated by the HC11 hardware. The *STRA* signal and the data are controlled by the PC through its software. The *STAF* is an internal flag of the HC11. The *ReadPORTCL* represents the corresponding HC11 software command .

Normally, the main HC11 is in input mode waiting for a new command from the PC. The *STRB* signal indicates to the PC that the HC11 buffer is ready for a new command byte. It is important to note that even though the HC11 may be busy executing some task, the PC can latch a command byte into the HC11 buffer whenever the buffer is empty. This asynchronous communication allows the system to take full advantage of the multiprocessor architecture.

If the *STRB* signal is high, the PC can output the command data on the bus and latch it into the HC11 buffer by asserting the *STRA* signal (falling edge). This operation will automatically deassert the *STRB* signal indicating to the PC that the HC11 buffer is full. It will also assert the *STAF* flag which tells the HC11 that there is new data in its buffer. When the HC11 detects that the *STAF* is asserted, it will read the data from the buffer. This action will deassert the *STAF* flag and automatically assert the *STRB* signal. The asserted *STRB* signal indicates to the PC that the HC11 is ready for another command. Meanwhile, the HC11 interprets the new command byte and executes the desired operation.

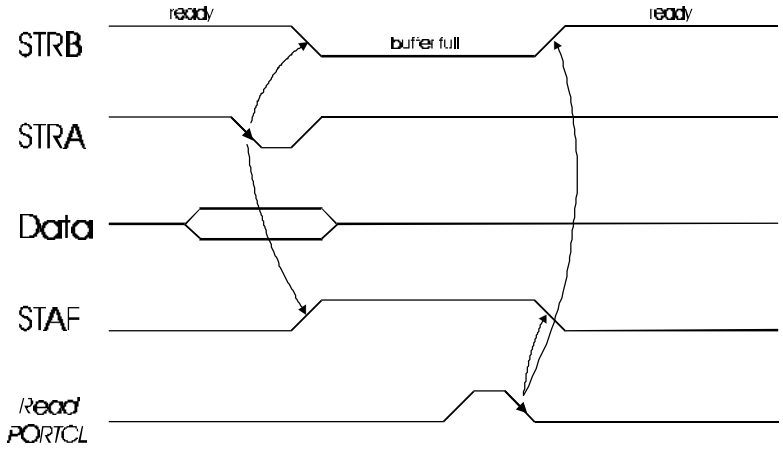


Figure 14: Write command

To minimize the communication delays, it is preferable to send only one byte for each command to the HC11. As long as only one byte is sent, the PC can simply latch it into the HC11 buffer without waiting for the HC11. If the command consisted of two bytes, two consecutive writings would be necessary, but these can not be handled by the HC11 hardware alone. In such a case, the PC could still latch the first byte into the HC11 buffer, but it would then have to wait for the HC11 to read this byte out of its buffer before sending the second byte.

In the current implementation, all commands are encoded in one byte as summarized in Table 2:

Command Byte	Command
1xxx'xxxx	Position of main servo (128 positions)
0000'xxxx	Activate resets on Port B (PB0-PB3)
0001'xxxx	EERUF modes (1-15) and stop sonars (0)
0010'xxxx	Fire single sonar (index 0-15)
0011'xxxx	Not used yet (16 values)
0100'xxxx	Right brake command (16 positions)
0101'xxxx	Left brake command (16 positions)
0110'xxxx	Main servo speed (16 values)
0111'0000	Read potentiometer of steering axis
0111'xxxx	Not used yet (15 values): xxxx ≠ 0

Table 2: Command bytes

3.3.2.2 The Read Command

The time diagram for a read command is shown in Figure 15. All signals have the same meaning as in the previous section. In addition, the *Mode* signal indicates to the PC if the main HC11 is in the input or output mode, and the *WritePORTCL* represents the corresponding HC11 software command.

To read a byte from the HC11, the PC first has to send a command to the HC11 indicating the desired information. Therefore, the entire communication consists of a write operation followed by a read operation. The write operation is identical to the process described in the previous section. However, once the command byte is interpreted, the HC11 retrieves the desired information and puts it into an internal register. This automatically changes the HC11 port to a readable output mode and deasserts *STRB*. To indicate to the PC that the data is ready, the *Mode* signal is set high. The PC then reads the data and asserts *STRA* indicating to the HC11 that it received the data. The *STRA* signal also automatically resets the HC11 output port into an input port to liberate the data bus for other communications. The HC11 then clears the *STAF* flag by reading its buffer, and deasserts the *Mode* signal to indicate to the PC that it is ready for a new command.

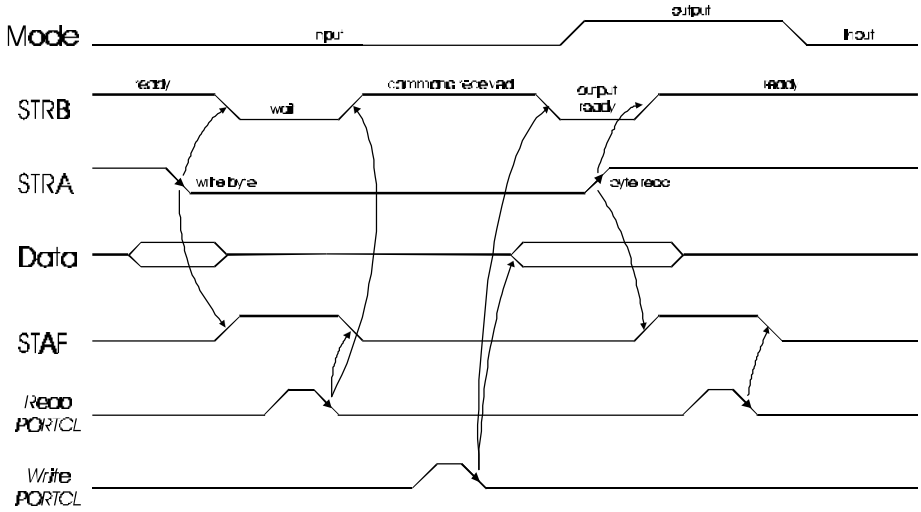


Figure 15: Read command

This mode is currently only used when the PC reads the value of the potentiometer. For future extensions, this mode will be useful to read additional sensors connected to the main HC11 analog port. It will also be useful to read data from devices that can be connected to the main HC11's SPI or SCI port.

3.3.3 The SPI Communication

Both HC11 slaves are connected to the main HC11 through the synchronous serial peripheral interface (SPI). The SPI interface is used primarily to allow the microcontroller to communicate with peripheral devices. Peripheral devices range from simple shift registers to complete subsystems, such as an A/D converter or another HC11. The SPI system is flexible enough to interface with numerous standard peripherals from several manufacturers. Data rates as high as 1 Mbit/sec are accommodated with one of the HC11 as the SPI master.

An example with four SPI devices connected to the main HC11 is shown in Figure 16. In the current interface architecture, there are only two SPI slaves. The main HC11 acts as the SPI master while the two HC11 slaves act as SPI slaves. This architecture allows one to easily add more SPI devices for future extensions.

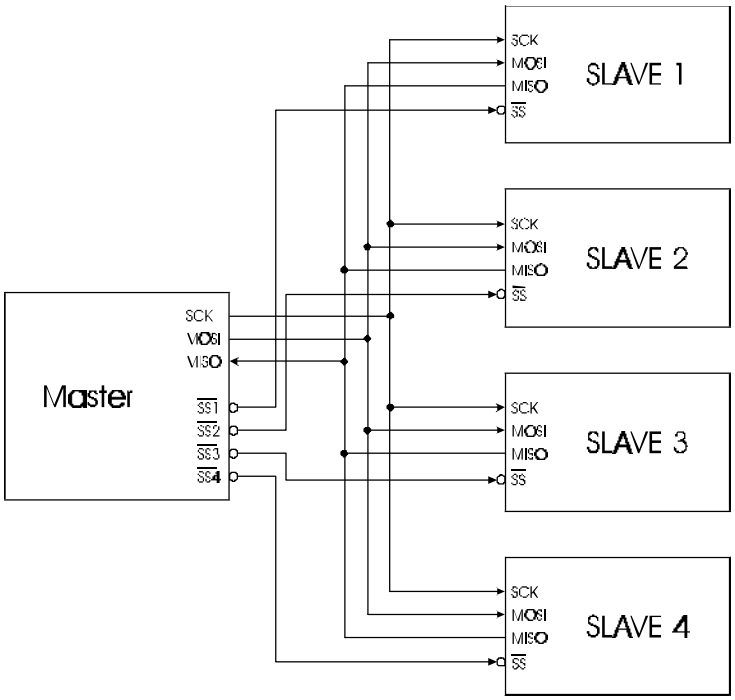


Figure 16: SPI architecture

During an SPI transfer, an 8-bit character is shifted out one data pin while an 8-bit character is simultaneously shifted in a second data pin. So, one byte is simultaneously transmitted and received. The serial clock line (*SCK*) synchronizes shifting and sampling of the information on the two serial data lines (*MOSI* and *MISO*). The slave select line (*SS*) allows individual selection of a SPI slave.

The use of the SPI is extremely simple as it is fully supported by the HC11 hardware. All SPI transfers are started and controlled by a master SPI device, in this case the main

HC11. To transfer a byte to a HC11 slave, the main HC11 simply asserts the corresponding slave select signal and writes the byte into its SPDR register. The hardware of the two involved HC11's then takes care of the communication. At the end of the communication, the bytes that were originally stored in the two SPDR registers are swapped. To indicate the end of the SPI communication, the *SPIF* (SPI Transfer Complete Flag) is asserted.

3.3.4 The FIFO

The interface is equipped with a FIFO whose purpose is to buffer the outputs of the two HC11 slaves. The currently used FIFO is a high-density first-in first-out buffer with a depth of 1024 bytes. This buffer allows the HC11 slaves to write their data into the FIFO whenever they have a sonar reading as explained in section 3.3.4.1. It also allows the PC to read the sonar data from the FIFO whenever the PC wants to as explained in section 3.3.4.2. Hence, due to the FIFO, the PC can asynchronously read the data from the HC11 slaves. This buffered communication allows the architecture to take full advantage of its distributed computing power.

3.3.4.1 The HC11 - FIFO Communication

The two HC11 slaves share an output bus and the */W FIFO* output to write the sonar data into the FIFO. To avoid access conflicts, the HC11 slaves use their *Bus?* and *Bus!* lines. The *Bus!* output of each slave is connected to the *Bus?* input of the other slave. Asserting the *Bus!* output means that the slave is in control of the bus or that it desires to be it. A problem only occurs when both slaves try to take control of the bus simultaneously. To resolve this conflict, the first slave is given priority. The algorithms for the two slaves are shown in Figure 17. The precedence of the first slave over the second one is realized by the different reaction to the case when the result of the second *Bus?* inquiry is positive.

If this method failed for whatever reason, one or both slaves could become damaged by short-circuiting each other. To eliminate this risk, however unlikely it is, the nine shared output lines are used in an open-collector mode with pull-up resistors. Therefore, if both slaves put their data on the shared bus no damage would occur. The only effect would be that wrong data would be written into the FIFO. However, in most cases, this would be detected by the PC reading the wrong data from the FIFO.

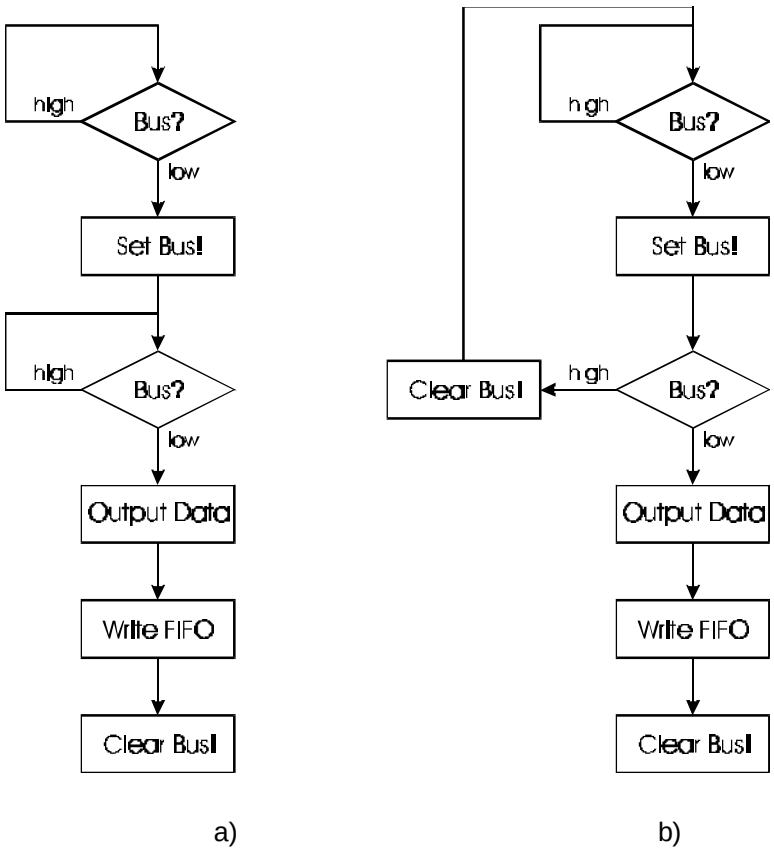


Figure 17: FIFO bus control for: a) slave 1, b) slave 2

3.3.4.2 The PC - FIFO Communication

The PC can easily determine if there is any data in the FIFO by checking the FIFO's "empty flag" (*EF*). If the *EF* flag is deasserted, the PC knows that there is new sonar data in the FIFO. The PC can then read out all sonar data by continuing reading until the EF flag asserts again. This method is very efficient, as the PC only has to check one input line to determine if new sonar data is available.

The PC could also be connected to the "half full" (*HF*) and "full full" (*FF*) flags of the FIFO. These flags are currently not used to save some of the parallel port input lines for future extensions. The *HF* could be used to tell the PC that it should read out the FIFO data. If the *FF* flag was asserted, it would indicate that probably a data overflow occurred because the PC waited too long before reading the FIFO. The PC would then have to send a reset command to the main HC11 to reset the two slaves and the FIFO. Next, the PC would have to send the command to start the sonars again. Therefore, if for whatever reason the FIFO filled up, the PC would instantly be aware of the problem and could go on without any interaction by the user.

A potential problem of the parallel port is "ringing". Misreadings due to ringing occur if the FIFO lines are directly connected to the PC parallel port. To eliminate ringing, each of the eight data lines is terminated by a resistor/capacitor network.

3.3.4.3 FIFO Data Encoding

As both operations, writing to and reading the FIFO, are extremely fast, there is no need to encode the sonar data in as few bytes as possible. For each sonar reading, five bytes are transmitted:

1. Start byte, currently $60[3].
2. Sonar index between 1 and 16.
3. High byte of 16 bit time of flight in increments of 8 µs.
4. Low byte of 16 bit time of flight in increments of 8 µs.
5. Stop byte, currently $80.

The start and stop byte are not necessary, but they increase the probability of detecting a communication problem. Even without these two bytes, the PC could detect a problem by verifying that the sonar index is between 1 and 16. As the extra time required by the start and stop bytes is practically negligible, these two bytes are kept for safety reasons.

3.3.5 The HCTL Quadrature Decoders

The output of each encoder consists of two quadrature signals. By quadrature decoding these two signals, the encoder resolution is multiplied by a factor of four as each edge, rising and falling, is taken into account. However, quadrature decoding is a time-intensive task with restrict requirements.

A very effective and simple solution is the use of the *HCTL quadrature decoder interface* integrated circuits from Hewlett Packard. The *HCTL-2016* features full quadrature encoding with a 16-bit up/down counter, latched outputs, and high noise immunity due to Schmitt trigger inputs and digital noise filters. Moreover, due to the 8-bit tristate output, the two quadrature encoders can simply be connected to the 8-bit PC-interface bus.

The quadrature decoders require a clock signal. Instead of adding some clock circuitry, the *E* output of the main HC11 is used. The *E* output is a clock signal with a frequency of 2 MHz. To read the 16-bit content of the counter, the low and high bytes can be accessed independently by the use of the *SEL* signal as shown in Figure 18:

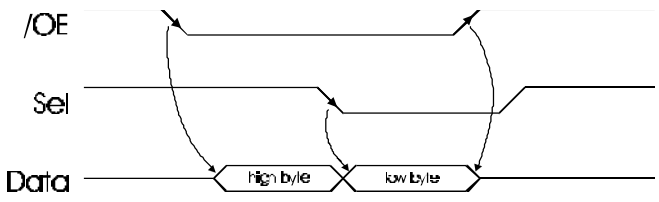


Figure 18: Timing of quadrature decoder reading

[3] The $ symbol means that the following number is in the hexadecimal format.

3.4 The LCD Screen

The GuideCane is equipped with a color LCD display to improve the development environment (see also section 8.3). The PC's CM106 super VGA utility module supports most available LCD displays. For reasons of price, weight, size, and power consumption, the Optrex DMF-50414 STN color display was selected. Its resolution is 640 x 480 pixels. The power consumption is about 1.9 W, the sum of the logic power consumption (0.8 W) and the backlight power consumption (1.1 W).

The CM106 module must be configured according to the LCD panel class. The Optrex color LCD display is classified as a C8DD-16 panel. Therefore, the switches 1 through 3 on the DIP switch S1 of the CM106 module must be set to On-Off-On or Low-High-Low.

The LCD display is connected to connector P4 of the VGA utility module through two band connectors. The connections are summarized in Table 3:

Pin name	CM106 P4	Optrex LCD
DL0 / Blue0	1	17
DL1 / Blue1	2	18
DL2 / Blue2	3	19
DL3 / Green0	4	20
DL4 / Green1	5	21
DL5 / Green2	6	22
DL6 / Red0	7	23
DL7 / Red1	8	24
DU7 / Red3	9	15
DU6 / Red2	10	14
DU5 / Green5	11	13
DU4 / Green4	12	12
DU3 / Green3	13	11
DU2 / Blue5	14	10
DU1 / Blue4	15	9
DU0 / Blue3	16	8
Data Shift Clock	18	3
Panel Frame Clock	22	1
VCC (+5VDC)	32	5
Data Latch Signal	35	2
VSS (GND)	19	6

Table 3: LCD connections

3.4.1 The Power-Up Sequence

LCD displays require special considerations upon power-up [19]. Power-on sequencing is mainly required to protect the liquid crystal from exposure to any DC voltage. The *VCC* (5 VDC) must be started first. This allows the on-board logic to become active and starts the internal *M* clock which sets up an AC wave form on the display electrodes. Even with very short intervals of exposure to *VEE*, without the *M* clock started first, the liquid crystal will begin to break down and change state. After *VCC* has stabilized, the external clock and data signals can be introduced. After the clock and data signals are stable, *VEE* can be turned on. Finally, the *display on/off* signal can be activated. To turn the LCD display off, the inverse sequence should be followed.

In the GuideCane implementation, the *VCC* is started first anyway by turning the main power switch on. To turn the LCD display on, "panel" must be typed on the keyboard. This will set the CM106 module in LCD mode and also turn off the signal for the standard CRT monitor. However, if the LCD display is turned on when the GuideCane is not connected to a CRT monitor, the user does not know if he typed the command correctly. Therefore, it is recommended to start the "lcdon" batch file instead. This batch file will run the "panel" command followed by a little sound sequence.

Then, switch 1 on the power board can be turned on. Afterwards, switch 2 can be turned on and the PC output should appear on the LCD display. The delay between all operations should be at least 50 ms. The positions of the two switches are shown in Figure 19 and their meaning is summarized in Table 4.

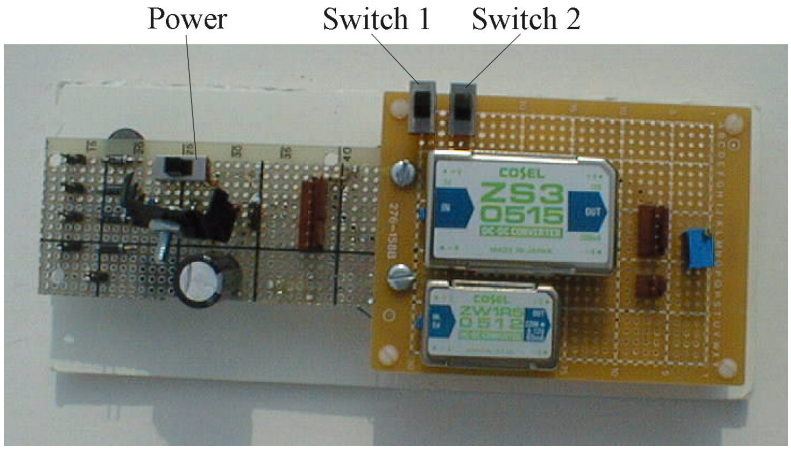


Figure 19: LCD switches

Pin name	Power Board	Optrex LCD
VEE (24 VDC)	Switch 1	7
Display On/Off	Switch 2	4

Table 4: Power-on switches

The turn-on sequence is summarized as:

1. Turn on power
2. Type “lcdon” and wait for sound sequence
3. Turn on switch 1 (push towards inside)
4. Turn on switch 2 (push towards inside)

The turn-off sequence is summarized as:

1. Turn off switch 2 (push towards outside)
2. Turn off switch 1 (push towards outside)
3. Type “lcdoff” and wait for sound sequence
4. Turn off power (optional)

3.4.2 The Backlighting

The type of the LCD backlighting is CFL. It requires an input signal of 900 VAC at 30 kHz. To generate this high voltage signal, an *inverter* is implemented. The inverter is a small PCB that requires an input of 12 VDC. The connections between the inverter output and the LCD backlight input are shown in Table 5:

LCD CN3	Cable Color	Voltage
1,2	Purple	Hot (900 VAC)
3,4	-	NC
5,6	White	GND

Table 5: Backlight connections

Unfortunately, the high voltage signal can induce a lot of noise into the system. Most of the GuideCane system is digital and not very sensitive to such noise. However, the sonars analog output is very sensitive to such noise, as it is amplified by the sonar interfaces. As a result, the sonars were not able to look further than 1.5 m with the LCD backlight turned on. To overcome this problem, the inverter is put in an aluminum box that shields the rest of the system from the high voltage signal. With this shielding, the noise is reduced to such a level that its influence on the sensor electronics is negligible.

3.5 The Power System

3.5.1 The Power Requirements

The GuideCane is powered by rechargeable batteries. The run-time requirement for the end user is at least 2 hours. As a comparison, the standard goal for laptop computers is a run-time of about 4 hours. However, actual laptops, when measured at nominal usage rates, only remain in the 2+ hour range.

The available power is mainly determined by the type and amount of batteries used. However, software and hardware also play a key role. Parts of the robot can be shut down by the software to save battery power, e.g. turning off the sonars when pausing. The hardware design also has an influence on the run-time. As a conclusion, all aspects of the power system must be considered and optimized.

The power requirements for the different parts of the robot are summarized in Table 6. During development, there are additional power requirements because of the development tools. The additional power requirements are shown in Table 7. The power requirements are about 36% higher during development than for the end product.

	Voltage	Nominal Current	Peak Current	Nominal Power
On-board PC	5 V	1.0 A	1.1 A	5.0 W
Sonars	5 V	0.5 A	2.0 A	2.5 W
Main Servo	5 V	0.5 A	0.9 A	2.5 W
Brakes	5 V	10 mA	1.0 A	0.05 W
Interface	5 V	100 mA	100 mA	0.5 W
Encoders	5 V	60 mA	90 mA	0.3 W
Pointer	5 V	10 mA	20 mA	0.05 W
Compass	5 V	30 mA	50 mA	0.15 W
Total	5 V	~ 2.2 A	~ 5.3 A	~ 11 W

Table 6: Basic power requirements

	Voltage	Nominal Current	Peak Current	Nominal Power
PC Hard-disk	5 V	0.2 A	0.3 A	1.0 W
PC VGA board	5 V	0.2 A	0.2 A	1.0 W
LCD logic	5 V	0.035 A	0.040 A	0.175 W
LCD power	24 V	0.025 A	0.030 A	0.6 W
LCD backlight	12 V	0.09 A	0.1 A	1.08 W
Total	5, 12, 24 V	~ 0.8 A at 5 V	~ 1.0 A at 5 V	~ 4 W

Table 7: Additional power requirements during development

3.5.2 The Battery Types

As the GuideCane is a portable device, the battery selection is very important as it will have an effect on the weight, size, and cost of the robot. A summary of the different battery type characteristics is shown in Table 8 [14][17].

Today, nickel-cadmium (NiCd) batteries are the most widely used type due to their availability and well-known technology. However, NiCd batteries have a mediocre energy density and are an environmentally hazardous substance because of the cadmium.

A major improvement in battery technology is the nickel-metal hydride (NiMH) chemistry. NiMH batteries are better than NiCd in both their energy density and their volumetric efficiency. Furthermore, NiMH cells do not contain any hazardous substances. As the open-circuit voltage of these batteries is the same as for NiCd batteries, they have replaced NiCd in many applications. Another advantage of NiMH is that they do not exhibit a memory effect as NiCd cells do. However, a NiMH battery will exhibit a *voltage depression* if it is repeatedly recharged after being only partially discharged. Several such cycles will result in a lower open-circuit voltage at full charge, so that it appears as if the cells remember the lower voltage level. Fortunately, a few complete discharge-recharge cycles will restore normal operation. The disadvantages of NiMH batteries are their increased cost, their higher self-discharge rate, and their need for special charging circuits.

Lithium based (Li-Ion, Li-Metal) batteries will probably be the next generation of batteries. Li-Ion batteries have only recently appeared on the market, while Li-Metal batteries will hopefully soon be available. These batteries are the most efficient rechargeable types currently known. However, they are still expensive and electronic charger components are still hard to find.

	NiCd	**NiMH**	**Li-Ion**	**Li-Metal**
Voltage/cell [V]	1.2	1.2	3.6	3.0
Energy density [Wh/kg]	45	70	100	140
Volumetric efficiency [Wh/ltr]	150	230	225	300
Cost [$/Wh]	0.75 - 1.5	1.5 - 3.0	2.5 - 3.5	1.4 - 3.0
Memory effect	Yes	Yes / No	No	No
Self-discharge (%/month)	25	20 - 25	8	1 - 2
Environmental concerns	Yes	No	No	No

Table 8: Battery types

NiMH batteries were selected for the GuideCane, because they are much better than NiCd while still at a reasonable price. Moreover, the price of NiMH batteries is likely to become cheaper.

Even though Li-Ion batteries have a higher energy density than NiMH batteries, their volumetric efficiency is about the same. Hence, the required volume for Li-Ion batteries is about the same as for NiMH batteries. As Li-Ion batteries are hard to find, delicate and more expensive for only a little gain, NiMH batteries are the better choice at this time. When reliable Li-Metal batteries become available at a reasonable cost, they would be a good replacement.

3.5.3 The NiMH Batteries

The GuideCane is currently powered by 6 HydriMax NiMH battery packs in parallel. The specifications of the batteries are summarized in Table 9:

Criteria	1 HydriMax NiMH Battery Pack	6 HydriMax NiMH Battery Packs
Nominal Voltage	6.0 V	6.0 V
Cut-Off Voltage	4.5 V	4.5 V
Capacity	1.8 Ah	10.8 Ah
Max. Const. Discharge Current	750 mA	4500 mA
Max. Temp. Discharge Current	2 - 3 A	12 - 18 A
Weight	200 g	1200 g
Temperature	10° - 40° C	10° - 40° C

Table 9: Battery specifications

These batteries should be recharged with a current of 100 mA. After full recharge, they should still be charged with 10 mA to counteract the self-discharge.

Although the minimum temperature of the batteries is 10° C, the GuideCane can still be used at colder temperatures as long as it is turned on inside a heated building and run for a couple of minutes. This will allow the PC and other electronic components inside the GuideCane to become hot enough to heat the interior, including the batteries.

At high temperatures during summer, it might be necessary to incorporate a temperature activated fan into the GuideCane to ensure that neither the batteries nor the PC overheat.

3.5.4 The Power Circuitry

As the current required by the GuideCane electronics exceeds by far the maximum constant discharge current of one battery pack, several battery packs have to be used in a parallel configuration. This also increases the capacity of the power system so that the run-time of the GuideCane is extended.

Diodes are put in series with each battery pack to prevent mutual charging and discharging. Without these diodes, a bad battery pack could actually destroy the other ones. Schottky diodes are used to minimize the associated power loss. These diodes have a maximum instantaneous forward voltage of only 0.45 V.

The battery output voltage is not constant. Even though the nominal voltage of the battery pack is 6 V, its actual output voltage varies from almost 7 V to 5.5 V. However, the PC/104 system requires an input voltage of 5.0 V ± 5 %. Therefore, a voltage regulator is needed between the battery packs and the GuideCane electronics.

As the battery output voltage is only slightly higher than the required 5 V, the best solution is a *low dropout voltage regulator*. Most of these regulators are not capable of outputting more than 1 A. Fortunately, the *LT1529-5* low dropout voltage regulator from *Linear Technology* became commercially available just recently [26]. This regulator has a dropout voltage of only 0.5 V at its maximum output current of 3 A. The efficiency of this regulator is 83% in the worst case. A power system consisting of such a regulator with *n* battery packs is shown in Figure 20. Capacitor C_{in} actually consists of two capacitors with the values of 0.1 µF and 3.3 µF. Capacitor C_{out} consists of three capacitors with the values of 0.1 µF, 3.3 µF, and 2.2 mF.

Unfortunately, the GuideCane system requires more than 3 A, so that two of these voltage regulators are needed. In the current implementation, the first regulator serves the sonars while the second one serves the PC, the interface, the encoders, and the servos. The reason for this configuration is to minimize noise on the sonars which are the most sensitive devices in the system. The first regulator is equipped with two battery packs, while the second one is equipped with four battery packs. The resulting run-time exceeds 3 hours.

As the final version of the GuideCane will consume about 2 A, only one voltage regulator will be needed. In addition, with the new proposed wheel-base structure, also a much less power consuming servo will be used.

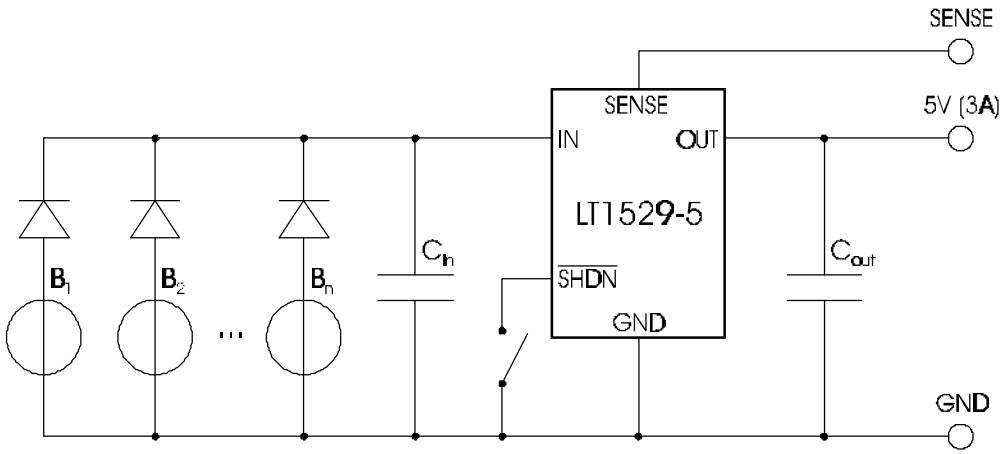


Figure 20: Voltage regulator system

3.5.5 Power System Evaluation

Several tests were conducted to verify the battery specifications and to evaluate the power circuitry. The battery tests consisted of connecting one or several fully charged batteries to a power resistor and monitor the voltage over time. From these measurements, the current, power, and capacity can be calculated. The data and the voltage curve of the first test are shown in Table 12 and Figure 21 respectively. The data and voltage curves of the second and third test are in the appendix. A summary of the test results is shown in Table 10 and Table 11:

Test #	# Batteries	Load	Current	Run-Time	Capacity /Battery
1	1	17.0 Ω	0.3 - 0.4 A	5.1 hours	2.02 Ah
2	1	10.3 Ω	0.5 - 0.7 A	3.4 hours	1.98 Ah
3	2	10.3 Ω	0.5 - 0.7 A	7.1 hours	2.03 Ah

Table 10: Battery test results

Test #	# Batteries	Load	Current	Run-Time
4	2	3.3 Ω	1.5 A	2.25 hours
5	4	2.2 Ω	2.25 A	3 hours

Table 11: Power system test results

The first two tests consisted of applying a fully charged battery directly to two different loads. These tests confirm the specified capacity of the batteries. A possible reason for measuring a slightly higher capacity is that the load was not constant during the test but was increased because of the increase of the resistor temperature.

For the third test, two batteries were put in parallel. Each battery had a Schottky diode in series to avoid mutual discharging as explained in section 3.5.4. Again, our test yields the expected results.

The last two tests included the full board with the voltage regulator yielding a steady 5V output. The measurement of the run-time was stopped when the output voltage dropped below 4.85 V.

Time	Δ Time	Voltage [V]	Current [A]	Power [W]	Δ [Ah]
2:35	0:00	6.87	0.404	2.776	
2:36	0:01	6.81	0.401	2.728	0.007
2:37	0:02	6.78	0.399	2.704	0.007
2:38	0:03	6.75	0.397	2.680	0.007
2:39	0:04	6.73	0.396	2.664	0.007
2:40	0:05	6.71	0.395	2.648	0.007
2:47	0:12	6.58	0.387	2.547	0.046
2:53	0:18	6.49	0.382	2.478	0.038
3:00	0:25	6.42	0.378	2.424	0.044
3:05	0:30	6.38	0.375	2.394	0.031
3:10	0:35	6.37	0.375	2.387	0.031
3:15	0:40	6.36	0.374	2.379	0.031
3:20	0:45	6.35	0.374	2.372	0.031
3:30	0:55	6.34	0.373	2.364	0.062
3:45	1:10	6.33	0.372	2.357	0.093
4:00	1:25	6.32	0.372	2.350	0.093
4:15	1:40	6.31	0.371	2.342	0.093
4:34	1:59	6.29	0.370	2.327	0.117
5:05	2:30	6.25	0.368	2.298	0.191
5:16	2:41	6.23	0.366	2.283	0.067
5:32	2:57	6.20	0.365	2.261	0.097
5:45	3:10	6.16	0.362	2.232	0.079
6:00	3:25	6.12	0.360	2.203	0.090
6:12	3:37	6.07	0.357	2.167	0.072
6:16	3:41	6.05	0.356	2.153	0.024
6:30	3:55	6.00	0.353	2.118	0.083
7:00	4:25	5.81	0.342	1.986	0.174
7:18	4:43	5.63	0.331	1.865	0.274
7:28	4:53	5.45	0.321	1.747	0.054
7:39	5:04	5.10	0.300	1.530	0.057
7:40	5:05	5.04	0.296	1.494	0.005
7:41	5:06	4.59	0.270	1.239	0.005
7:42	5:07	4.00	0.235	0.941	0.004
7:43	5:08	3.96	0.233	0.922	0.004
7:44	no charge	5.38 V			2.024

Table 12: Single battery test data

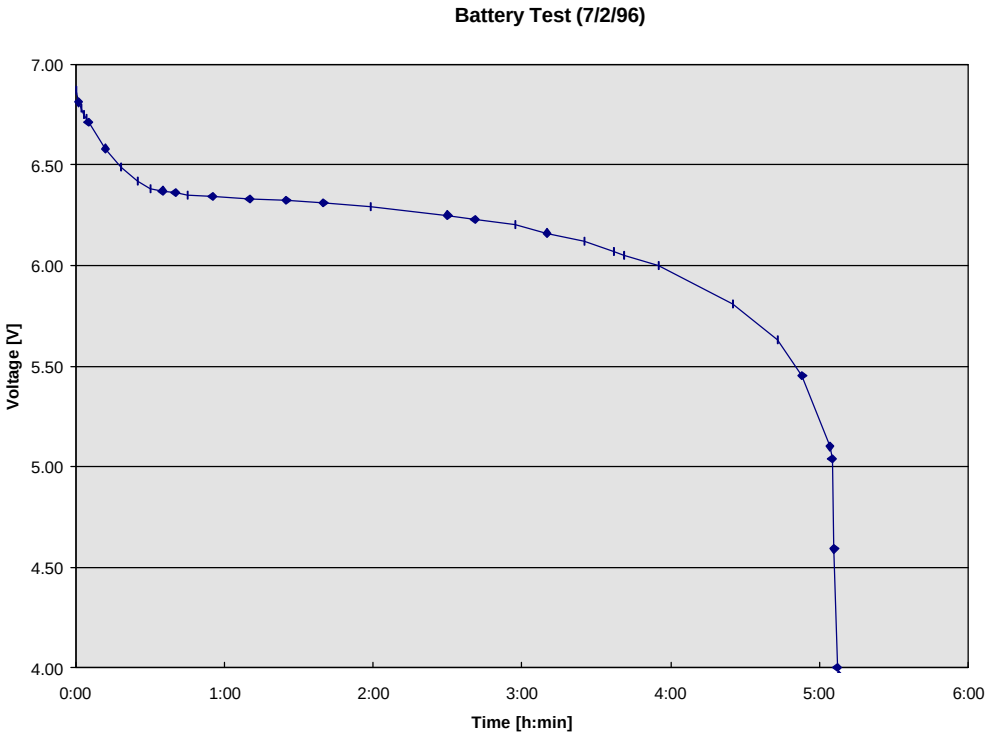


Figure 21: Single battery voltage curve

4. THE SENSORS AND ACTUATORS

4.1 The Ultrasonic Sensors

4.1.1 Properties of Sonars

To detect obstacles and measure their position relative to the GuideCane, the robot is currently equipped with ten Polaroid Ultrasonic sensors, or in short *sonars* [30]. The principle of ultrasonic sensing is based on emitting a short high frequency signal (~40 kHz) and measuring the time of flight (*TOF*) it takes for the signal to propagate from the sensor to an object and back. As the speed of travel is known, the distance to the object can be calculated. The change of speed dependent on the temperature is 7% from 0° to 40°C.

The speed of the ultrasonic wave at 20°C is:

$$c = 343.2 \text{ m/s} = 1125 \text{ ft/s}$$

Hence, the distance to an object based on the time of flight is:

$$d = \frac{TOF * c}{2}$$

Therefore, a time of flight of 1 ms corresponds to 34 cm of flight distance, or 17 cm of distance to the corresponding object. The sonar accuracy is about 1%. The standard sensing range is 40 cm to 11 m. By asserting the *BINH* signal , the minimum sensing range can be decreased to 10 cm for the best sonar devices, and to 15 cm for most other devices. The minimum sensing range is set to 10 cm for the eight forward facing sonars, and to 15 cm for the two side facing sonars.

Although sonars provide good range data, they offer only poor directionality. An opening angle of 20-30° is typical for the Polaroid sensor. Another problem of sonars is specular reflections from smooth surfaces. This is particularly a problem if the robot has to perform wall-following. A method to reduce the effects of specular reflections on the robot's wall-following behavior is explained in section 7.4.

The Polaroid sonars come with an electronic interface that generates the high voltage necessary to emit the powerful but short sound wave. The current consumption of each electronic interface is about 50 mA. For the short duration of 0.5 ms during the pulse emission, the current peaks up to 2 A. To reduce the constraints on the power system, a high value capacitor is added to the power lines of each interface.

4.1.2 EERUF - Sonar Control

The sonars are controlled by the *EERUF* (Error Eliminating Rapid Ultrasonic Firing) method to take maximum advantage of their capabilities [6]. EERUF allows ultrasonic sensors to fire at rates that are five to ten times faster than those in conventional applications. The problem of crosstalk, a result of the fast firing rate, is practically[4] eliminated by the crosstalk detection algorithm in EERUF. The faster firing rate improves the reliability and robustness of the mobile robot obstacle avoidance and is necessary for safe travel at higher speed.

The EERUF method is explained in detail in [6]. In short, EERUF is based on the principle of comparison of consecutive readings, but, in addition, employs alternating delays before firing each sensor.

To give the reader a small insight into EERUF, an example of the interaction of two sonars is explained in the following section. Figure 22 shows the environment for this example.

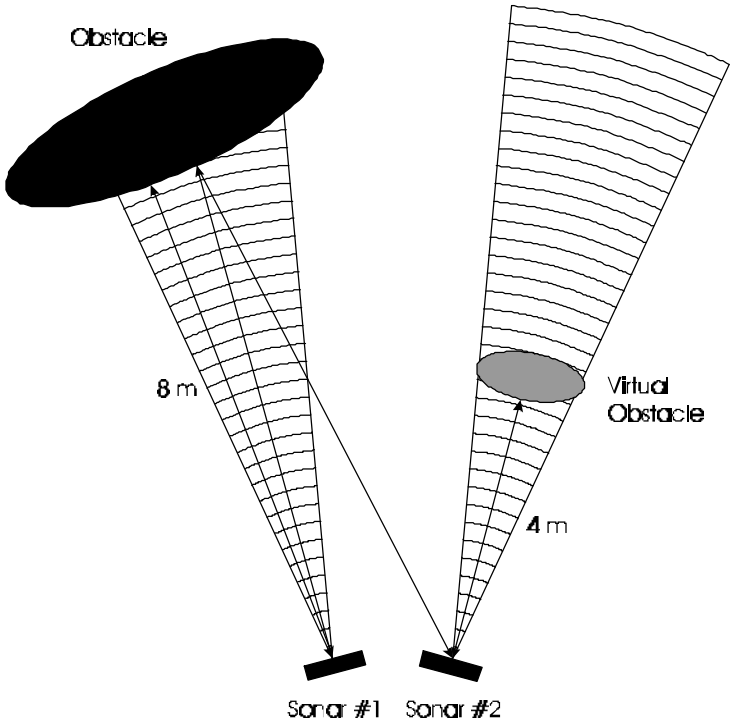


Figure 22: EERUF example

[4] Crosstalk is not detected by EERUF in cases of very special and unlikely object arrangements that result in a sonar receiving specific crosstalk from more than one sonar.

In the ideal case, only sonar #1 would detect the obstacle at a distance of 8 m corresponding to a time of flight of about 50 ms. Unfortunately, the ultrasonic wave emitted by sonar #1 could not only be reflected back to sonar #1 but also to sonar #2. This phenomenon is called *crosstalk*. If the sonars were fired once every 100 ms based on a conventional sonar system, this environment could lead to undetected crosstalk.

Let's assume that we had a system of four sonars, with sonar #2 firing 25 ms after sonar #1. In a situation without crosstalk, the timing diagram would look like in Figure 23. However, if crosstalk occurred, the reflected wave would hit both sonars #1 and #2 at about the same time. So, sonar #2 would indicate that there is an obstacle in its direction at about 4 m. As a result, the robot would try to avoid this virtual obstacle. It is also important to note that the comparison of consecutive readings does not help to eliminate these crosstalk readings.

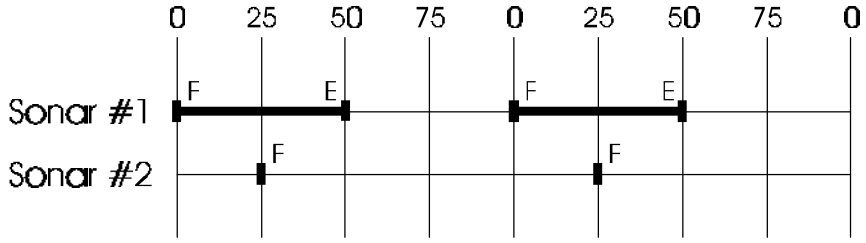


Figure 23: Conventional system without crosstalk

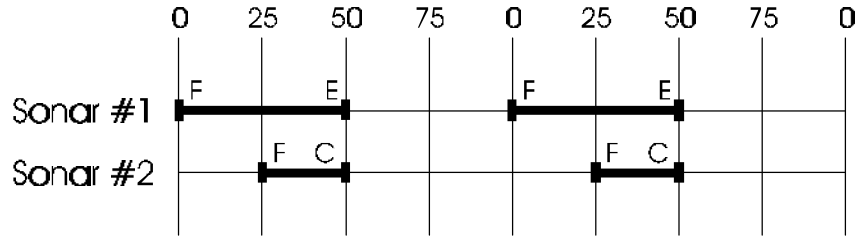


Figure 24: Conventional system with crosstalk

The problem of crosstalk is overcome by EERUF by applying alternating time delays before firing a sonar. For example, the time delays specified by [6] for sonar #1 are 24 and 18 ms. For sonar #2, the time delays are 49 and 37 ms. Without crosstalk, the situation is similar to the conventional system. However, with crosstalk, EERUF is able to detect the problem. The time diagram for our example is shown in Figure 25. Sonar #2 still returns the crosstalk readings, but due to the alternating time delays, they are not identical anymore. If the situation was static, sonar #2 would return 25 ms, 31 ms, 25 ms, 31 ms and so on. As the difference of the consecutive readings is more than 1 ms, the limit set by this specific EERUF mode, the readings of sonar #2 will be rejected as crosstalk.

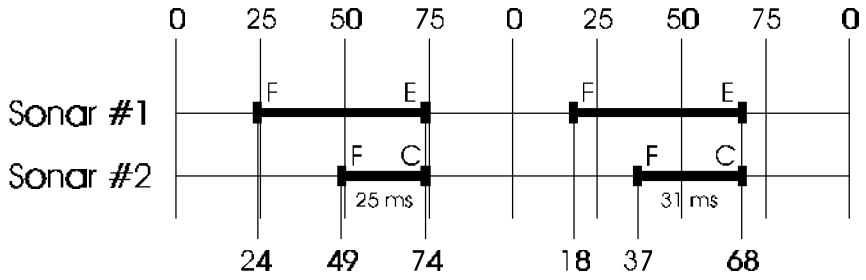


Figure 25: EERUF system with crosstalk

For a better performance of the robot, the crosstalk readings should not simply be discarded. In our example, when sonar #2 returns a value of 31 ms after a value of 25 ms, the reading indicates that there is nothing up until a distance corresponding to 31 ms. If sonar #2 returns a value of 25 ms after a value of 31 ms, it indicates that there is nothing up until a distance corresponding to 25 ms.

Therefore, even though these readings are rejected as crosstalk, the sensor data can still be used to decrement certainty values in the local map as explained in section 6.3.2. As a result, there will be less cells with incorrect high certainty values. This is especially useful in environments with moving objects, e.g. other people.

4.2 The Steering Angle Measurement

The GuideCane needs to know its current steering angle to correctly update the map based on the sonar readings. It also needs to know this value to determine the correct new servo output.

A simple solution would be to assume that the actual steering angle is equal to the desired steering angle. However, after every large change in the desired steering angle, this assumption would not hold until the servo reached the new desired angle. Consequently, readings from the sonars would be placed into wrong map locations.

There are several possibilities to measure the actual steering angle. The currently implemented method uses the potentiometer inside the servo. As the servo shaft is rigidly linked to the wheel-base, the servo angle is identical to the wheel-base angle. This method is very simple, as the servo potentiometer can be connected to the HC11 analog input port.

Another possibility would have been to add a potentiometer to the main axis. This method has several inconveniences. Firstly, there is not much free space on the steering axis where the potentiometer could be fixed. Secondly, the most accessible place to put the potentiometer is rather dirty, which could cause wrong outputs unless it is put inside a sealed housing. Finally, this method requires additional parts and an installation which is more delicate than the first method.

A third possibility would have been to add a digital encoder to the main axis. However, this solution is the most expensive of all three solutions. It would also require more space and a more delicate installation than the first method. Its main advantage would be the encoder's precision. However, the precision of the steering angle measurement is not essential.

The interior of the currently used servo is shown in Figure 26. The internal potentiometer of the servo outputs voltages between 0.81 V and 1.63 V for angles between 50° and -50° respectively. The output signal (white wire) is connected to the analog input port of the main HC11. With the 8-bit resolution of the HC11 A/D converter, the precision of the steering angle measurement is 2.3°.

To increase the precision of the analog-digital conversion to 0.39°, an amplifier could be added between the potentiometer and the HC11. This electronic circuit would map the range of 0.81 V to 1.63 V to the range of 0 to 5 V, the full input range of the HC11 A/D converter. This mapping is not considered necessary as the current servo position precision is good enough.

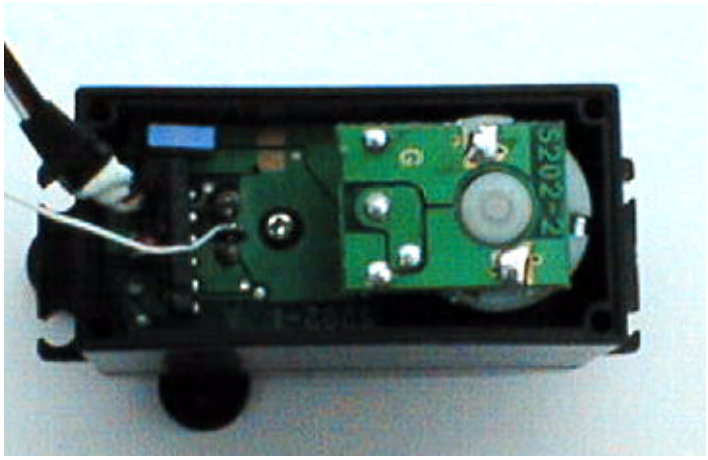


Figure 26: Servo interior

4.3 The Encoders

The GuideCane is equipped with encoders for the purpose of odometry. The first wheel-base was equipped with the RG encoders from Photocraft. These encoders had the advantage of internal ball bearings which simplified the wheel fixation design. However, it turned out that the encoders were too heavy, resulting in a too high moment of inertia of the wheel-base. Although the encoders weigh only 400 grams, their placement at the wheel-base extremities resulted in a high moment of inertia, which would have required a very strong and power consuming servo.

To reduce this moment of inertia, the new wheel-base is equipped with the light HEDS-5540-A06 encoders from Hewlett Packard. The resolution of these encoders is 500 pulses per revolution. Using full quadrature encoding, the resulting resolution becomes 2000 pulses per revolution. With wheels of a diameter of 4.5 inches, each pulse corresponds to a wheel advancement of just 0.18 mm.

4.4 The Servos

The wheel-base is rotated by a servo motor[5]. Servo motors have the advantage of being very easy to use. They usually consist of a DC-motor, a gear reduction, a potentiometer, and an electronic control board. The servo input is a pulse-width modulated (*PWM*) signal. The width of the PWM signal [1ms, 2ms] specifies the desired output angle [-60°, 60°] through a linear mapping. The control board takes care of the motor control.

The GuideCane is currently equipped with three *Futaba* servos, one for the main axis and two for the brakes. Futaba servos are high quality and better documented than other servos.

For the main servo, the first tests were done with the S125 model which has an output torque of 129.3 oz-in. This servo has a high gear reduction resulting in a relatively small speed of 97°/s. This speed was too slow for fast maneuvers. And because of the high gear reduction, the servo axis got easily destroyed by mechanical shocks applied to the wheels.

The currently used servo is the S3801 model which has a higher output torque of 200 oz-in. This servo has also a smaller gear reduction resulting in a higher speed of 272°/s. Because this servo is stronger and has a smaller gear reduction, it supports much higher shocks. However, this servo is bigger, heavier, and consumes more power.

With the improved wheel-base structure, as proposed in section 2.2.2, a much weaker, smaller, lighter, cheaper, and less power consuming servo could be used.

As already mentioned, a servo has its own control board, usually consisting of an analog PD controller. This is a very nice feature, as one can simply specify a desired output angle and it will take care of the closed-loop control. However, the control parameters, which can not be changed, are adjusted for small loads. With the current wheel-base, the moment of inertia is rather high. As a result, if a step input was applied to the servo, its output would overshoot and oscillate in a very unsatisfactory way. The source of the problem is that the servo would accelerate too much, trying to achieve a performance that is not feasible.

Fortunately, this problem can be overcome with the low-level software. The solution is to slow the servo down by giving it a maximum velocity limit. An example is shown in Figure 27. Without a velocity limit, a new desired input j is given to the servo every sampling time t_i of the PC. This is basically a succession of step inputs. With the heavy load of the wheel-base, the servo overshoots and oscillates.

A velocity limit can be implemented by providing a trapezoidal input signal to the servo, where the slope of the signal corresponds to the maximum velocity. Transforming the step input signal to a trapezoidal signal is performed by the HC11. With this transformation, the servo neither overshoots nor oscillates.

[5] A hobby servo motor, not a DC servo motor with a linear velocity-torque characteristic.

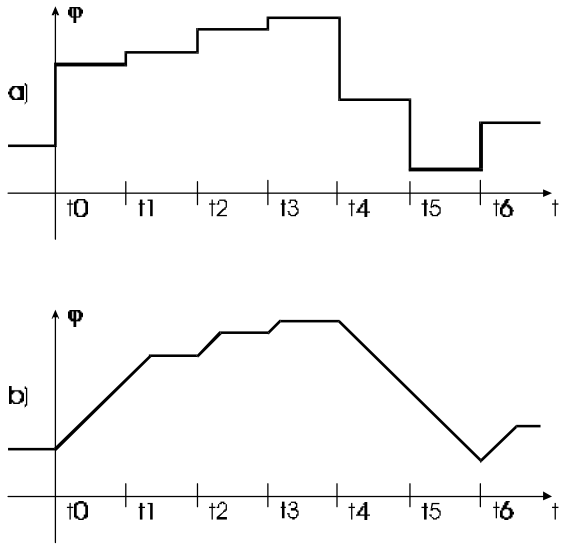


Figure 27: Servo input: a) without velocity limit, b) with velocity limit

However, another problem appeared because of the discrete nature of the histogram grid. In the current implementation, the histogram resolution a is 5°. When the robot is going down a hallway, the obstacle avoidance algorithm often oscillates between two values separated by the histogram resolution. As a result, the servo heavily oscillates between these two values resulting in a very jerky behavior.

This problem could be overcome by choosing a slower servo velocity limit. Another possibility would have been to filter the obstacle avoidance output. Both methods improve the hallway following behavior, but they slow down the robot's reaction to an obstacle in its way. Both methods would result in a trade-off between trajectory smoothness and obstacle avoidance performance.

A better method is to have a servo velocity limit that is a function of the difference between the actual orientation q and the desired new orientation j_n as shown in Figure 28. With this method, the robot reacts fast to an obstacle avoidance maneuver, but still provides a smooth trajectory.

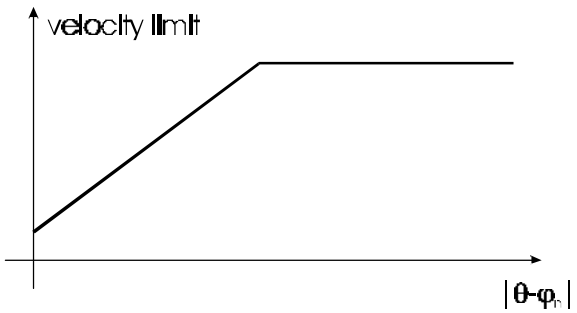


Figure 28: Variable main servo speed limit

4.5 The Input Device

The GuideCane needs an input device so that the user can indicate the desired direction of travel. This direction can be understood as either relative to the orientation of the robot or relative to the orientation of the cane. The input device should also allow the user to indicate the desire for special behaviors, like going through a narrow door.

The input device is connected to the serial port of the PC. As most input devices are compatible with the PC serial port, the user will be able to use his preferred input device by simply connecting it to the serial port.

4.5.1 The Pointer

The currently implemented input device consists of a *ProPoint* pointer made by *Interlink Electronics*. This pointer consists of a mouse button which can be pressed in any direction. The mouse button is sensitive to the applied force. The pointer is also equipped with two additional buttons.

The mouse button is used to indicate the desired direction of travel. In the current implementation, this direction is discretized in either four or eight directions. The more discrete directions are available, the harder it is to use the mouse button correctly.

The two buttons are not used yet. As mentioned above, they could be used to select specific behaviors. If more than two modes were necessary, these buttons could be combined with the mouse button to offer more input possibilities.

4.5.2 Other Input Devices

If the user did not like the pointer, he could easily replace it with another serial port compatible input device. An overview of existing device types is shown in Table 13.

Name	Directions	Buttons
ProPoint Pointer	Continuous	2
Trackball	Continuous	2-3
Touchpad	Continuous	2-4
Joystick	Continuous	2-4
Game Controller	8	6
Keypad	-	4-32

Table 13: Input device types

5. The HC11 Software

5.1 The Main HC11 Software

The main HC11 takes care of most of the communications, samples up to eight analog inputs, resets other devices, and generates the PWM signals for the servos. As these tasks have different time constraints, the software is organized in a multi-tasking architecture. The tasks running in parallel are summarized in Table 14:

#	Task	Type	Time constraints
1	Execute command as requested by PC	Regular Program	Low
2	Generation of PWM signals	Software interrupt: OC1 Hardware interrupts: OC2 - OC5	Every 20 ms
3	Continuous A/D conversions	Software interrupt: OC2	Every ~20 ms (when PA6 goes high)

Table 14: Main HC11 tasks

In the current implementation, the regular program is interrupted about every 20 ms by tasks #2 and #3. Because the interrupt service routines are small, most of the computing power is reserved for the regular program.

Currently, only a small amount of the main HC11's computing power is used, so that there is enough computing power and program space left for future extensions.

5.1.1 Task #1 - Communication and Execution

The regular program spends most of its time waiting for a new command byte from the PC. When it receives a new command, it will decode it according to Table 2 (page 22) and execute it. If the command is for the servos, the HC11 will put the corresponding value in an internal register, which will be read later by the servo interrupt routine (task #2).

If the PC needs to know the value of an analog input, the HC11 can simply read the value from the last A/D conversion and send it back to the PC trough handshaking as explained in section 3.3.2.

If the command is for the HC11 slaves, the main HC11 will send it to them trough the SPI line as explained in section 3.3.3.

5.1.2 Task #2 - Generation of the PWM Signals

Task #2 generates the PWM signals for up to 4 servos. Servos are typically controlled by a PWM signal that repeats every 20-30 ms with a variable pulse width ranging from 1 to 2 ms in duration:

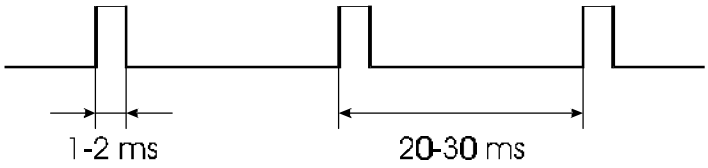


Figure 29: Servo input signal (PWM)

The generation of these signals can be implemented by two different methods. In the first method, the main HC11 generates the PWM signals with a 32.77 ms period. Even though this period is higher than the typical period, the servos still work properly. The reason for the choice of this particular period is that the internal timer of the HC11 overflows after 32.77 ms. Therefore, the PWM signals can be generated based on hardware interrupts only, without requiring any software interrupt servicing. This is very efficient, as it takes no computation power of the HC11. However, if a new command with a smaller pulse width is given to the HC11 in the time span between the falling edges of the new and old command, the PWM signal will stay high for an entire period. As the critical time span is rather small, this problem occurs rarely. If it happens, the effect is small as the momentarily wrong command is filtered by the inertia of the motor and wheel-base.

In the second method, the PWM signals are generated with the typical period of 20 ms. This method eliminates the potential problem of the first method. It is also advantageous over the first method if the sampling rate of the GuideCane software is smaller than 32.77 ms. As a disadvantage, this method requires software interrupt servicing, and therefore takes some computation power of the HC11. As the interrupt service routine is small and the main HC11 has more than enough computation power for its other tasks, the second method was implemented as it is safer and allows higher sampling rates.

This second method can be implemented with the HC11 *output compare functions*. These real-time interrupt functions allow one to efficiently implement up to four PWM signals. In the current GuideCane configuration, one PWM signal is used for the main servo, and two are used for the brake servos. So, another servo can easily be added for future extensions. A regular way of implementation and a better version are shown in Figure 30.

In the regular method, the OC1 (output compare 1) hardware interrupt is used to simultaneously generate the rising edges of the three PWM signals every 20 ms. The OC2-OC5 hardware interrupts are used for the falling edges of the four respective PWM signals.

In the improved method, the OC1 hardware interrupt is used to simultaneously generate the falling edges of the three PWM signals every 20 ms. The OC2-OC5 hardware interrupts are used for the rising edges of the four respective PWM signals.

With both methods, only the OC1 interrupt is serviced. This interrupt service routine calculates the output compare timer values for the next events and writes them to the corresponding registers. For the regular method, as shown in Figure 30, these values must be updated no later than 1 ms after the OC1 interrupt occurred. In the current implementation, 1 ms is far enough time. However, if the main HC11 has to take care of more parallel tasks in future extensions, this time limit could be a problem. With the improved method, the time limit is 18 ms, so that there should be no problem even if several more tasks were added. For this reason, the improved method was implemented.

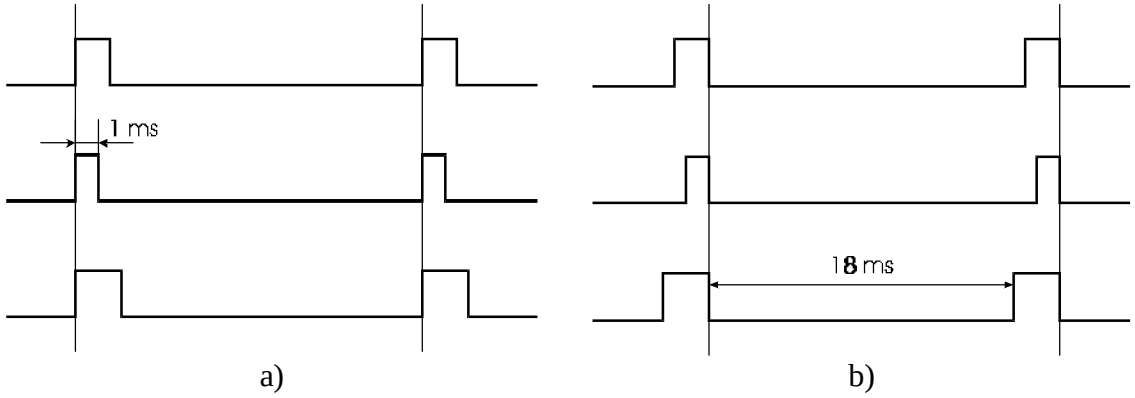


Figure 30: Generation of PWM signals: a) regular method, b) better method

5.1.3 Task #3 - Continuous A/D Conversions

To measure the steering angle orientation, the HC11 reads the analog value of the servo potentiometer. However, this signal is invalid from the falling edge of the main servo input signal for a duration that is a function of the applied torque. A safe time to get a valid reading is to sample this voltage at the rising edge of the main servo input signal. As this edge is generated by the OC2 hardware interrupt, the OC2 interrupt service routine is used to sample the signal at that specific time.

In the current implementation, the potentiometer signal is actually converted four consecutive times. Once the A/D conversion is started by the OC2 interrupt service routine, the HC11 hardware automatically makes these four conversions in a time interval of only 64 µs.

The results of these conversions are stored in registers which can be read by task #1. These four measurements are averaged before being sent to the PC.

5.2 The HC11 Slave EERUF Implementation

5.2.1 The Multitasking Architecture

For reasons of limited input/output lines and speed, each HC11 slave takes care of up to eight sonars. Each HC11 slave has to accomplish several tasks at different rates and with different time constraints. To implement this efficiently, the HC11 slave software is also organized in a multi-tasking architecture. The four different tasks with their properties are summarized in Table 15:

#	Task	Type	Time constraints
1	Compute TOF and write into FIFO	Regular program	Low
2	Generate fire signals and set time for next BINH interrupt	Software interrupt: OC4	13 times in 200 ms according to EERUF schedule
3	Check echo signals	Software interrupt: OC5	Every 50 µs
4	Generate BINH signals	Hardware interrupt: OC1	Exactly 0.6 ms after corresponding fire signal

Table 15: HC11 slave tasks

The regular program (task #1) is constantly interrupted by the two software interrupts (tasks #2 and #3) which generate the fire signals and check the echo signals. The fourth task (task #4) is supported by the HC11 hardware so that it does not require interrupt servicing. However, the times at which the BINH signals are changed are set during the interrupt servicing of task #2. Tasks #2 and #3 communicate with task #1 through an internal FIFO buffer and several global variables.

5.2.2 Task #1 - Communications and Treatment of Buffer

Task #1 reads the internal FIFO buffer containing the data from tasks #2 and #3. It analyzes this data and computes the time of flight (TOF). This task also takes care of all the other not time-critical tasks, e.g. the SPI communication with the main HC11.

The time constraints of task #1 are low. However, if it is not executed often enough, the internal buffer may overflow. As the timing of task #2 is fixed by the EERUF schedule and task #4 is only a hardware interrupt, only the timing of task #3 can be changed to influence the time allocation of task #1. Basically, the higher the execution rate of task #3, the better the sonar resolution, but also the less time is allocated to task #1, and hence the higher the risk of an internal buffer overflow. However, tests have shown that task #3 can even be executed every 25 µs without problems. The algorithm of task #1 is summarized as following:

1. New request from main HC11? If yes execute, e.g. start sonars, stop sonars, change EERUF mode, fire single sonar.

2. Check internal buffer. If empty go back to step 1.
3. Read next byte from internal buffer.
4. If the byte is equal to *No_Echo*, the corresponding sonar(s) has not received back an echo. Write index and TOF equal to zero into FIFO. Go back to step 1.
5. Otherwise, compute the TOF for corresponding sonar(s). Write index and TOF into FIFO.
6. Go back to step 1.

5.2.3 Task #2 - The Generation of the Fire Signals

Task #2 is responsible for generating the fire signals and setting the time for the next BINH interrupt. The fire signals must be generated as specified by the EERUF time schedule. This task is executed 13 times in an interval of 200 ms. Its algorithm is the following:

1. If a fire signal is disactivated, check if the corresponding echo was received. If not, write the *No_Echo* byte and the index of the corresponding sonar(s) into the internal buffer.
2. Output new fire signals and save time into array p_list_fire[index].
3. Set time and output for next BINH signal.
4. Set time for next fire signal event.
5. Increment EERUF table pointers.
6. Execute task #3.

5.2.4 Task #3 - Checking for Echoes

Task #3 is the most time intensive as it checks for new echoes every 50 μs. The more often this task is executed, the better the sonar resolution. Hence, it is important to make this routine as short and fast as possible. For this reason, task #3 only checks for new echoes, but neither identifies the sonar(s) nor computes the TOF. It just saves the current time and a representative byte into the internal buffer. The representative byte contains bits set to one for sonars with new echoes. This data is then treated by task #1. The algorithm of the interrupt service routine is as follows:

1. Read echoes.
2. Check for new echoes with: $\neg$(previous_echoes) $\wedge$ (current_echoes).
3. If the result is not zero, save it and the current time in internal buffer.
4. Save current echoes as previous_echoes.
5. Set time for next interrupt.

5.2.5 Task #4 - The Generation of the BINH Signals

Task #4 is the most time constrained as the BINH signal must be activated exactly 0.6 ms after the fire signal. This can be done by using the hardware interrupt OC1 without any interrupt servicing. When the HC11 timer becomes identical to the time set in step 3 of task #2, the OC1 interrupt will automatically output the four bits defined in the same step. This assures exact activation of the BINH signals independent of other interrupts.

5.2.6 The Fire Signal Table

The EERUF fire schedule in [6] is tailored for a sonar system consisting of 12 sonars. It is reproduced in Table 16. Each sonar is fired once every 100 ms interval. The difference between two consecutive sonar echoes must be smaller than 1 ms to be accepted as a valid reading. These fire signals are generated by task #2 according to the EERUF schedule. For an efficient software implementation of task #2, the EERUF time schedule is stored in a look-up table. The remaining part of this section develops the look-up table for the first HC11 slave, which uses the schedule for the first eight sonars.

Sonar	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$T_{fire,a}$ [ms]	24	49	74	99	24	49	74	99	24	49	74	99
$T_{fire,b}$ [ms]	18	37	56	75	20	39	58	77	22	41	60	79

Table 16: EERUF firing schedule

For the second HC11 slave, the schedule for the sonars 9-12 and 1-4 is currently implemented for the sonars 9-16. The selection of repeating the EERUF schedule for the sonars 1-4 is arbitrary and has no effect on the current GuideCane performance because the sonars 13-16 are not connected. If the GuideCane got equipped with more than 12 sonars in future extensions, it might be necessary to develop a new EERUF schedule for 16 sonars to make sure that cross-talk gets rejected. However, if it is very unlikely for a pair of sonars to receive their respective signals, the fire schedule of these sonars can be the same. The sonars 9 and 10 of the current GuideCane are an example of such a pair. By placing the sonars carefully, the existing EERUF schedule can likely be kept.

The sonar fire signal has to be held high as long as the sensor should be listening to an echo. For maximum flexibility, the fire signal is held high for longer than 58 ms which, theoretically, allows the sonars to detect an obstacle up to at least 10 m. In practice, the electronic environment is too noisy for the sonars to detect obstacles reliably at this distance. However, choosing a large look-ahead gives us maximum flexibility. The PC software can easily reduce the look-ahead, but it can not extend it. For more flexibility during the HC11 software design process, we will start out with an even larger look-ahead of 72.6 ms.

As each of the eight sonars fires once in 100 ms, we would expect to need 16 interrupts in the same time interval, eight for asserting the fire signals and eight for

deasserting them. However, during every other period, two sonars are activated and disactivated together as shown in the upper part of Table 17. This reduces the number of interrupts to 12 in a time interval of 100 ms.

Event [0.1 ms]	Fire	Stop
240	1, 5	
966		1, 5
490	2, 6	
1216		2, 6
740	3, 7	
1466		3, 7
990	4, 8	
1716		4, 8
1180	1	
1200	5	
1906		1
1926		5
1370	2	
1390	6	
2096		2
2116		6
1560	3	
1580	7	
2286		3
2306		7
1750	4	
1770	8	
2476		4
2496		8
2240	1, 5	

Table 17: Fire signals

To generate these time-critical fire signals, the HC11 uses its timer interrupt OC4. The HC11 timer is prescaled by a factor of 16 to give it a 8 µs resolution. The timer has to interrupt the HC11 whenever a fire signal has to change its state. After each interrupt, a new interrupt time has to be set for the next change. Therefore, what matters is the difference Δ in 8 µs steps between consecutive time events. After ordering the fire signal events, we get the information shown in Table 18:

Event [0.1 ms]	Event [8 µs]	Δ [8 µs]	Fire	Stop
240	3000	3125	1, 5	
490	6125	3125	2, 6	
740	9250	2825	3, 7	
966	12075	300		1, 5
990	12375	2375	4, 8	
1180	14750	250	1	
1200	15000	200	5	
1216	15200	1925		2, 6
1370	17125	250	2	
1390	17375	950	6	
1466	18325	1175		3, 7
1560	19500	250	3	
1580	19750	1700	7	
1716	21450	425		4, 8
1750	21875	250	4	
1770	22125	1700	8	
1906	23825	250		1
1926	24075	2125		5
2096	26200	250		2
2116	26450	1550		6
2240	28000	575	1, 5	
2286	28575	250		3
2306	28825	2125		7
2476	30950	250		4
2496	31200			8

Table 18: Ordered fire signals

Now, we can reduce the number of events even further by combining some of the stop events with others. When we built our first fire signal schedule, we were using a look-ahead of 72.6 ms which is much longer than the required look-ahead of 58 ms. Therefore, we can combine a stop event with another one that occurs not earlier than 14.6 ms. This allows us to reduce the number of events in a time interval of 200 ms to only 13 as shown in Table 19. This table also contains the data for the generation of the BINH signals. The data of the last 5 columns is stored in the form of look-up lists in the HC11 memory.

Event [0.1 ms]	Event [8 µs]	Δ [8 µs]	Fire	Stop	ΔBINH	Byte BINH
240	3000	3125	1, 5	3, 4, 7, 8	BINH	0000|0000
490	6125	3125	2, 6		BINH	0101|0000
740	9250	3125	3, 7		BINH	1010|0000
990	12375	2375	4, 8	1, 5	BINH	1111|0000
1180	14750	250	1		BINH	1100|0000
1200	15000	2125	5	2,6	BINH	0000|0000
1370	17125	250	2		BINH	0001|0000
1390	17375	2125	6	3,7	BINH	0101|0000
1560	19500	250	3		BINH	0110|0000
1580	19750	2125	7	4,8	BINH	1010|0000
1750	21875	250	4		BINH	1011|0000
1770	22125	2500	8		BINH	1111|0000
1970	24625	3375		1, 2, 5, 6	0	0000|0000
2240	28000		1, 5		BINH	0000|0000

Table 19: Look-up table for the first HC11 slave

The data for the second HC11 slave is shown in Table 20:

Event [0.1 ms]	Event [8 µs]	Δ [8 µs]	Fire	Stop	ΔBINH	Byte BINH
240	3000	3125	9,13	11,12,15,16	BINH	0000|0000
490	6125	3125	10,14		BINH	0101|0000
740	9250	3125	11,15		BINH	1010|0000
990	12375	2375	12,16	9,13	BINH	1111|0000
1180	14750	500	13		BINH	0011|0000
1220	15250	1875	9	10,14	BINH	0000|0000
1370	17125	500	14		BINH	0100|0000
1410	17625	1875	10	11,15	BINH	0101|0000
1560	19500	500	15		BINH	1001|0000
1600	20000	1875	11	12,16	BINH	1010|0000
1750	21875	500	16		BINH	1110|0000
1790	22375	2875	12		BINH	1111|0000
2020	25250	2750		9,10,13,14	0	0000|0000
2240	28000		9,13		BINH	0000|0000

Table 20: Look-up table for the second HC11 slave

6. The PC Software

6.1 The Main Loop

The PC software is written in C and runs under DOS. The high-level algorithm of the main loop is shown in the following list. For good robot performance, it is important that the sampling period of the main loop is as low as possible. The sampling period should not exceed 60 ms to avoid large oscillations in the robot's trajectory. Specific parts of the algorithm are explained in more detail in the following sections.

1. Read values from HCTL decoders and update odometry.
2. Read value from potentiometer.
3. Plot robot position on screen (optional).
4. Read input device. If new input, discretize it into 4 or 8 directions and update the desired direction.
5. Read sonar data from FIFO and update histogram grid accordingly.
6. Build polar histograms.
7. Determine new steering direction.
8. Set new servo direction.
9. Output histograms and other data (e.g. sampling time) on screen (optional).

6.2 Odometry

The odometry equations for the GuideCane are the same as for a differential drive mobile robot. However, as the GuideCane wheels are unpowered, there is less risk of wheel slippage. Hence, the amount of non-systematic odometry errors should be smaller. Based on the encoder outputs, the GuideCane can perform dead reckoning by using simple geometric equations to compute its momentary position relative to a known starting point. For completeness, the well-known equations for odometry are repeated below [13][21].

First we need to determine the conversion factors that translate the encoder pulses into linear wheel displacements:

Right wheel: $c_r = \pi * \frac{d_r}{r_r}$

Left wheel: $c_l = \pi * \frac{d_l}{r_l}$

where: d = nominal wheel diameter [mm]
r = encoder resolution [pulses per revolution]

At each sampling interval i, we can compute the incremental travel distance for both wheels based on the encoder pulse increments N_r and N_l:

$$\Delta U_{r,i} = -c_r * \left(N_{r,i} - N_{r,i-1} \right)$$
$$\Delta U_{l,i} = c_l * \left(N_{l,i} - N_{l,i-1} \right)$$

The incremental linear displacement ΔU_i of the robot's center-point C:

$$\Delta U_i = \frac{\Delta U_{r,i} + \Delta U_{l,i}}{2}$$

The incremental change of orientation $\Delta\theta_i$:

$$\Delta\theta_i = \frac{\Delta U_{r,i} - \Delta U_{l,i}}{b}$$

where b is the wheel-base of the vehicle, ideally measured as the distance between the two contact points between the wheels and the floor.

The robot's new relative orientation can be computed in a recursive way:

$$\theta_i = \theta_{i-1} + \Delta\theta_i$$

This orientation could also by computed in an absolute way:

$$\theta_i = \theta_0 - \frac{c_r * \left(N_{r,i} - N_{r,0} \right) + c_l * \left(N_{l,i} - N_{l,0} \right)}{b}$$

If the encoders are reset at $t = 0$:

$$\theta_i = \theta_0 - \frac{c_r * N_{r,i} + c_l * N_{l,i}}{b}$$

And the relative position of the center-point is:

$$x_i = x_{i-1} + \Delta U_i * \cos\theta_i$$
$$y_i = y_{i-1} - \Delta U_i * \sin\theta_i$$

6.3 Local Map Building

To achieve a good obstacle avoidance performance, the GuideCane needs to build a local map of the surrounding obstacles based on the input from its sonars. This allows the robot to take into account not only the current sonar readings but also previous readings. In other terms, this map represents a world model which allows the GuideCane to be more than just a simple reflex agent.

6.3.1 The Map Representation

The map is represented by a two-dimensional array, called a *histogram grid*, similar to the *certainty grid* [27] and *occupancy grid* [15]. Each cell contains a *certainty value* (CV) that indicates the measure of confidence that an obstacle exists within the cell area. This representation is especially suited to sensor fusion, as well as the accommodation of inaccurate sensor data such as range measurements from ultrasonic sensors.

In the current implementation, the dimensions of the map are 18 m by 18 m. The dimensions of a cell are 10 cm by 10 cm. Therefore, the map is represented by an array of 180 by 180 bytes. The origin of the x-y system is at the center of the map. To simplify the screen output for development purposes, the reference system is selected as shown in Figure 31.

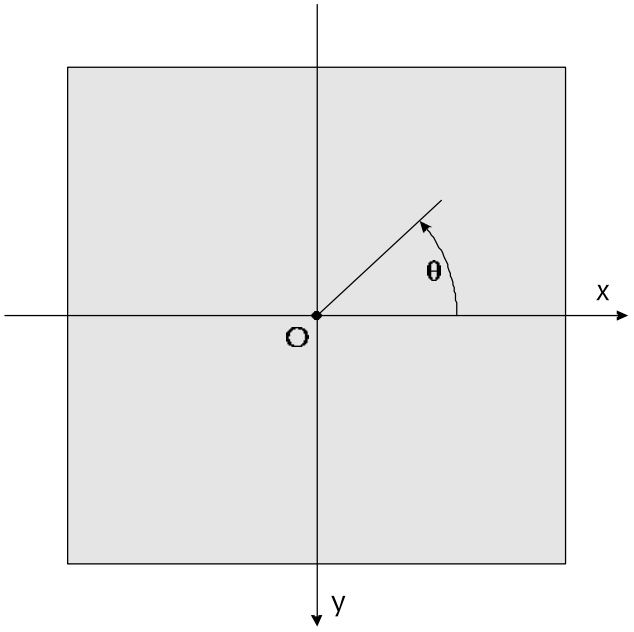


Figure 31: Reference system

6.3.2 The Map Building - HIMM

Histogramic In-Motion Mapping (HIMM) is a real-time map building method for a mobile robot in motion [7]. Contrary to other methods, the histogram method increments only one cell in the histogram grid for each range reading. For the sonars, the incremented cell is the one that corresponds to the measured distance and lies on the acoustic axis of the sensor. The histogram method also decreases the certainty value of the cells between the robot and the incremented cell. While this approach may seem to be an oversimplification, a probability distribution is actually obtained by continuously and rapidly sampling each sensor while the vehicle is moving. This results in a histogramic probability distribution.

In the current implementation, the certainty value range is between 0 and 63. The increment I^+ of a cell is +20 for the front-facing sonars, and +50 for the side-facing sonars. This parameter is higher for the side-facing sonars to achieve a better wall-following behavior. Because this parameter affects only cells on the robot's side, a false reading would not have a big influence on the obstacle avoidance behavior. The decrement I^- is set to -10. To increase the signal/noise ratio, the obstacle avoidance response of the VFH algorithm is proportional to the square of a certainty value.

To compensate for the adverse scattering effects caused by in-motion sampling, a *growth rate operator* (GRO) is added. A cell is incremented faster by the GRO if the immediate neighbors hold high CV's according to the following equation:

$$cv_{x,y} = cv_{x,y} + I^+ + \frac{cv_{x-1,y-1} + cv_{x-1,y} + cv_{x-1,y+1} + cv_{x,y-1} + cv_{x,y+1} + cv_{x+1,y-1} + cv_{x+1,y} + cv_{x+1,y+1}}{2}$$

In short, HIMM produces high CV's for cells that correspond to obstacles and keeps low CV's for cells that were incremented due to misreadings or moving objects. Moreover, any range reading is immediately represented in the map and has immediate influence on the concurrent obstacle avoidance algorithm.

6.3.3 The Scrolling

As the size of the local map is limited, it is necessary to incorporate scrolling. However, the scrolling algorithm has to be efficient in order not to slow down the sampling rate of the robot. *Continuous scrolling*, e.g. to scroll the whole map by one pixel whenever the robot's position changes from one pixel to another, would be a much too time consuming process. *Discrete scrolling* is much more time efficient which is the reason why this method was implemented on the GuideCane.

In the current implementation, scrolling occurs when the robot passes one of the virtual borders shown in Figure 32. These borders are located half-way between the center of the map and the map border at $x = \pm 4.5$ m and at $y = \pm 4.5$ m. Whenever the odometry values x or y are outside this range, the map is moved by 4.5 m. Actually, only ¾ of the map are moved and the other ¼ is set to zero.

As the look-ahead of the sonars is well below 4.5 m, this method performs well and is very efficient.

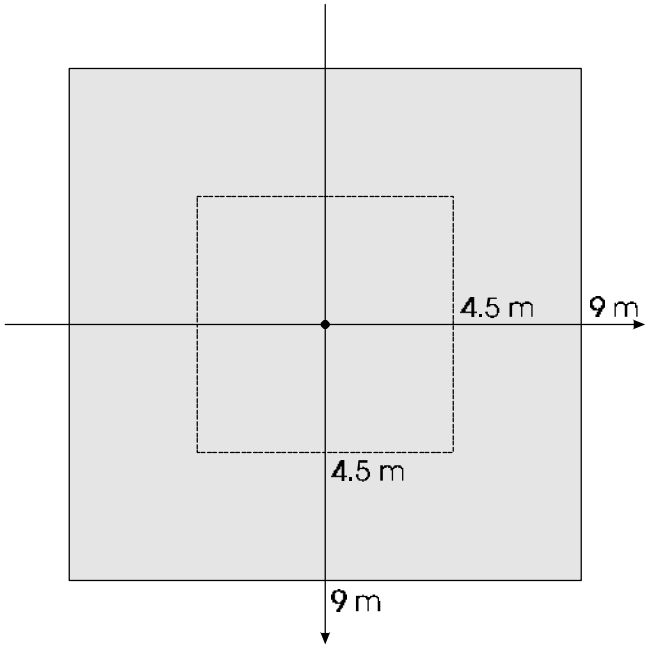


Figure 32: Virtual scrolling borders

7. Obstacle Avoidance

The main task of the GuideCane is to steer around obstacles and proceed toward the original target position or direction. The performance of the GuideCane is directly related to the performance of its obstacle avoidance algorithm.

First, the original *Vector Field Histogram* (VFH) method was implemented. This method was then successively improved resulting in the *VFH+* algorithm. The final algorithm with a less local behavior resulted in the *VFH** method.

7.1 Original VFH

7.1.1 The VFH Algorithm

The original VFH obstacle avoidance algorithm is described in detail in [9]. This powerful and efficient method uses the histogram grid as its world model. The VFH method employs a two-stage data reduction process in order to compute the desired control commands for the robot. In the first stage, the histogram grid is reduced to a one-dimensional *polar histogram* that is constructed around the robot's momentary location. Each sector in the polar histogram contains a value representing the *polar obstacle density* in that direction. In the second stage, the algorithm selects the most suitable sector from all polar histogram sectors with a low polar obstacle density, and the steering of the robot is aligned with that direction.

7.1.2 First Stage - The Building of the Polar Histogram

The first data reduction stage maps the active region C^* of the histogram grid C onto the polar histogram H. The active region C^* is a window that moves with the robot, overlying a square region of $w_s \times w_s$ cells in the histogram grid. The content of each active cell in the histogram grid is treated as an obstacle vector. The vector direction $\beta_{i,j}$ is determined by the direction from the cell to the vehicle center point (*VCP*):

$$\beta_{i,j} = \tan^{-1}\left(\frac{y_j - y_o}{x_i - x_o}\right)$$

where:

$\beta_{i,j}$: Direction from the *VCP* to the active cell (i,j).
x_0, y_0: Present coordinates of the *VCP*.
x_i, y_j: Coordinates of active cell (i,j).

The vector magnitude is given by:

$$m_{i,j} = \left(c_{i,j}\right)^2 \left(a - bd_{i,j}^2\right)$$

where:

$m_{i,j}$: Magnitude of the obstacle vector at active cell (i,j).
$c_{i,j}$: Certainty value of active cell (i,j).
$d_{i,j}$: Distance between active cell (i,j) and *VCP*.
a, b: Positive parameters.

Note that $c_{i,j}$ is squared. This expresses our confidence that recurring range readings (high $c_{i,j}$) represent actual obstacles, as opposed to single occurrences of range readings (low $c_{i,j}$) which may be caused by noise.

The vector magnitude is also a function of the distance $d_{i,j}$. Occupied cells produce large vector magnitudes when they are close to the robot, and smaller ones when they are further away.

The parameters a and b are chosen according to the parameter equation:

$$a - bd_{\max}^2 = 1$$

where

$$d_{\max} = \frac{w_s - 1}{2}$$

This way, $m_{i,j}$ is equal to 1 for the farthest active cell and increases linearly for closer cells. Note that the last two equations are slightly different from the original version of the VFH method [9]. With these equations, the VFH amplitude function is point symmetrical around the robot's center. As a result, the robot's behavior is independent of the direction in which an obstacle is encountered.

Based on the obstacle vectors, the polar histogram H is built. H has an arbitrary angular resolution α so that $n = 360°/\alpha$ is an integer. In the current implementation, α is set to 5° resulting in $n = 72$ angular sectors.

Each angular sector k corresponds to a discrete angle θ discretized as multiples of α, so that:

$$\theta = k\alpha \qquad \text{where } k = 0,1,2,\ldots,n\text{-}1.$$

Correspondence between an active cell $c_{i,j}$ and a sector k is established through:

$$k = INT\left(\frac{b_{i,j}}{a}\right)$$

For each sector k, the polar obstacle density h_k is calculated by:

$$h_k = \sum_{i,j} m_{i,j}$$

Because of the discrete nature of the histogram grid, the result of this mapping may appear ragged and causes errors in the selection of the steering direction. To overcome this problem, a smoothing function is applied to H:

$$h'_k = \frac{h_{k-l} + 2h_{k-l+1} + \ldots + lh_k + \ldots + 2h_{k+l-1} + h_{k+l}}{2l+1}$$

For l, an empirical value of 5 is proposed in [9]. This smoothing filter is eliminated by an improvement in the VFH+ method which takes in account the size of the robot as explained in section 7.2.2.

As the histogram is built around the position of the robot independent of its orientation, this first stage of the VFH algorithm can be implemented very efficiently by the use of tables.

For an example, Figure 33 shows an obstacle course with a typical corresponding histogram grid. Figure 34 shows the corresponding polar histogram grid.

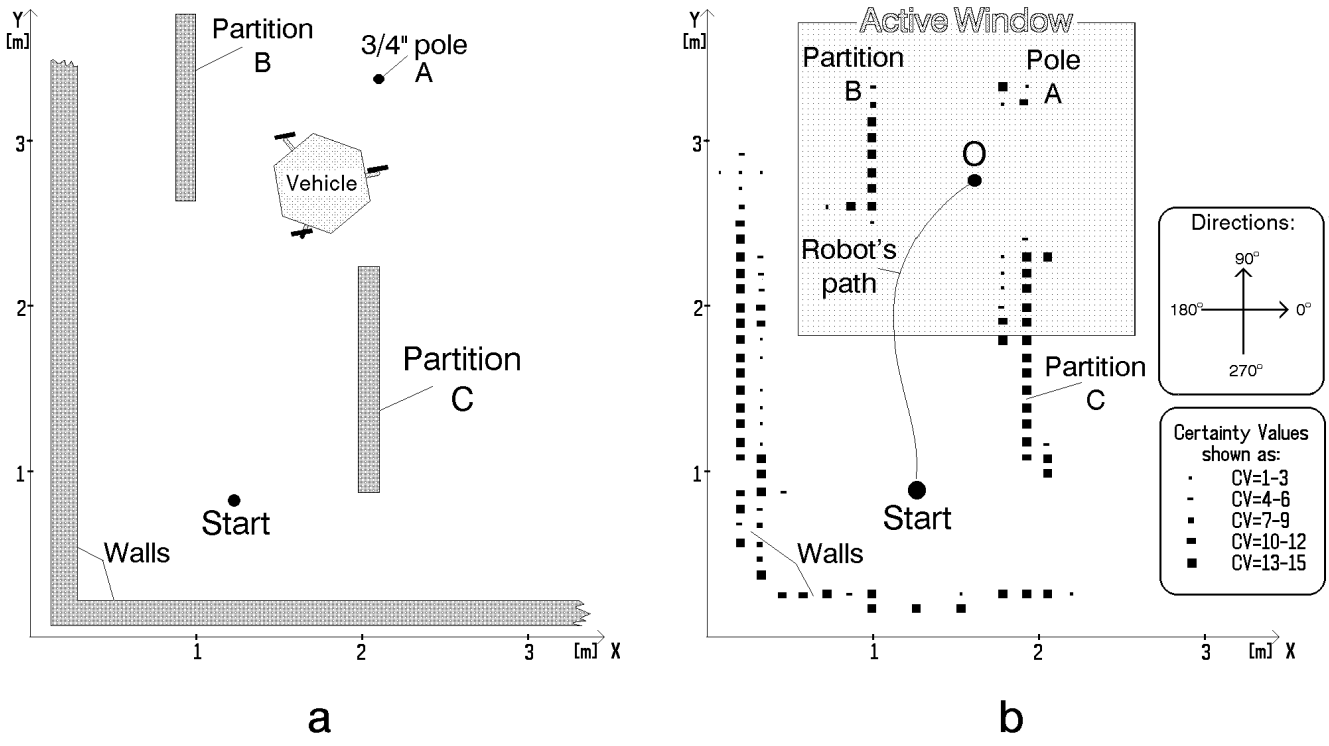


Figure 33: VFH Example: a) obstacle course, b) histogram grid

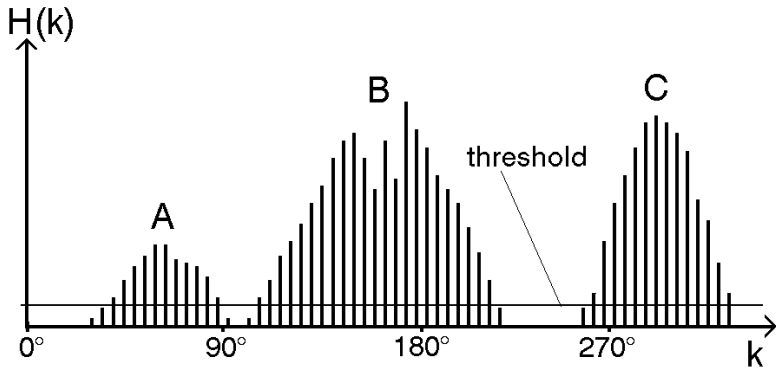


Figure 34: Polar Histogram

7.1.3 Second Stage - Selection of the Steering Direction

The second data reduction step computes the new steering direction φ_n from the polar histogram which typically has peaks and valleys. Any valley comprised of sectors with polar obstacle densities (*POD*) below a certain threshold τ is called a candidate valley. Usually, there are two or more candidate valleys. In the original implementation of VFH, the direction that is closest to the target direction is selected. Once a valley is selected, it is then necessary to choose a suitable direction within that valley.

First, the VFH algorithm measures the size of the selected valley. Here, two types of valleys are distinguished, namely, *wide* and *narrow* ones. Wide valleys result from wide gaps between obstacles or from situations where only one obstacle is near the robot. Narrow valleys are created when the mobile robot travels between closely spaced objects. The limit angle that distinguishes between a wide and a narrow opening is φ_{wide}. A valley is considered wide if more than s_{wide} consecutive sectors fall below the threshold with $s_{wide} = \varphi_{wide} / \alpha$. Otherwise it is narrow.

Then, for each valley, the two *border sectors* k_1 and k_2 are determined. A border sector is a sector whose *POD* value is below the threshold and which has a neighboring sector whose *POD* value is above the threshold.

For a narrow opening, the new steering direction is in the center of the opening:

$$\varphi_{n,narrow} = \frac{k_1 + k_2}{2}\alpha$$

For a wide opening, the border sector that is closer to the target direction is determined. This sector is denoted by k_n. The new steering direction is then computed by:

$$\varphi_{n,wide} = \left(k_n + \frac{s_{wide}}{2}\right)\alpha$$

Compared to the *Virtual Force Field* (VFF) method, an important advantage of the VFH method is the elimination of vivacious fluctuations in the steering control. With the averaging effect of the polar histogram, the two border sectors k_1 and k_2 vary only mildly between sampling intervals. Thus, the VFH method does not require a low-pass filter in the steering control loop and is therefore able to react much faster to unexpected obstacles. Similarly, the VFH controlled robot does not oscillate when traveling in narrow corridors.

7.2 VFH+

During the development of the GuideCane, a few shortcomings of the original VFH method were identified. The different problems and their solutions are explained in the following sections. The improved version of the obstacle avoidance algorithm is called the VFH+ method.

7.2.1 Threshold with Hysteresis - The Binary Polar Histogram

For most applications, a smooth trajectory is desired and oscillations in the steering command should be avoided. The original VFH method usually displays a very smooth trajectory. However, in densely cluttered environments with several narrow openings, oscillations in the steering commands are observable. The source of this problem is the fixed threshold t. It is possible that a narrow opening in the histogram switches several times between an open and blocked state during a few sampling times. In such a situation, the robot's heading can switch between the heading corresponding to this opening and another heading. The result is not only an oscillatory behavior, but the mobile robot can also get very close to obstacles.

This problem can easily be reduced by implementing a hysteresis based on two thresholds, namely t_{low} and t_{high}. Based on the polar histogram *H* and the two thresholds, a *binary polar histogram* H^b is built. The binary polar histogram is similar to the regular polar histogram, but instead of having polar density values, each sector is either *free* (0) or *blocked* (1). The binary polar histogram is updated by the following rules:

$$
\begin{aligned}
H^b_{k,i} &= 1 && \text{if } H_{k,i} > t_{high} \\
H^b_{k,i} &= 0 && \text{if } H_{k,i} < t_{low} \\
H^b_{k,i} &= H^b_{k,i-1} && \text{otherwise}
\end{aligned}
$$

7.2.2 Consideration of the Robot Size

The original VFH method does not take into account the size of the robot. One associated problem is that the robot has a tendency to cut corners. This problem was originally reduced by adding a small *Virtual Force Field* (VFF [8]) component to the direction given by the VFH method. However, the addition of the VFF component also added the problems associated with this method. The VFF method can result in considerable fluctuations in the steering control. These oscillations can be reduced by adding a low-pass filter to the VFF output. However, this filter introduces a delay that adversely affects the robot's steering response to unexpected obstacles. Another problem of the VFF method is that it results in oscillatory and unstable motion under certain conditions [22][23]. It would be preferable if the VFH method took in account the size of robot so that the addition of the VFF component would not be necessary anymore.

The standard method to compensate for the size of a mobile robot is to enlarge the obstacle cells in the map by the robot radius r_r. To make sure that the robot does not get too close to an obstacle, the obstacle cells are actually enlarged by a radius r_{r+s}:

$$r_{r+s} = r_r + d_s$$

with: r_{r+s} = total radius for enlarging obstacle cells
r_r = distance from robot center to its furthest perimeter point
d_s = minimum distance between robot and obstacle cell

With the obstacles enlarged by r_{r+s}, the robot can be treated as a point-like vehicle, which simplifies the obstacle avoidance algorithm. This method works well, as the shape of most mobile robots can be approximated by a disk of radius r_r. If the robot's shape was very elongated in one direction, the obstacles would have to be enlarged corresponding to the dimensions and orientation of the robot. In the case of the GuideCane, the shape of the sensor head is simply approximated by a disk.

A straightforward approach would be to enlarge the obstacles represented in the histogram grid before building the polar histogram. However, this procedure would require two maps, one with the original certainty values and one with the enlarged obstacles.

A simpler and faster method is to directly enlarge the obstacles while building the polar histogram. Instead of updating only one histogram sector for each cell, all histogram sectors that correspond to the enlarged cell are updated. An example for a cell is shown in Figure 35.

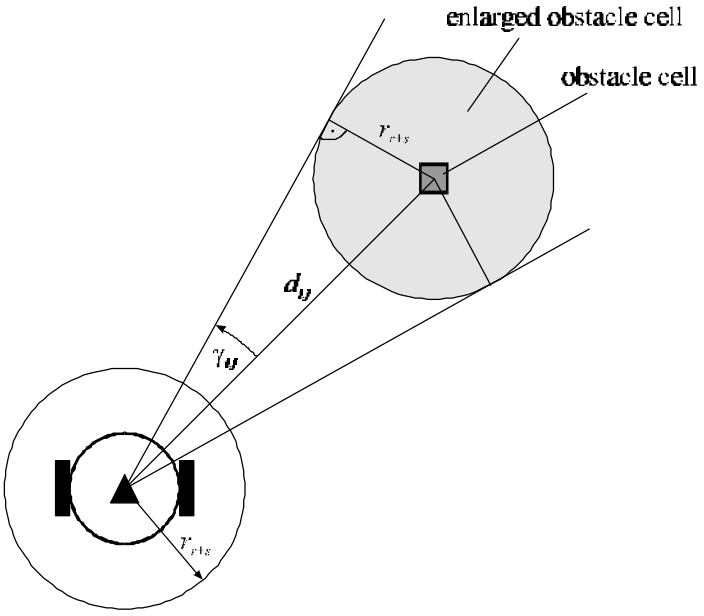


Figure 35: Enlargement angle

For each cell, the enlargement angle $\gamma_{i,j}$ is defined by:

$$\gamma_{i,j} = \arcsin\frac{r_{r+s}}{d_{i,j}}$$

The $\gamma_{i,j}$ values for each cell in the VFH window should be stored in a table for faster performance.

For each sector *k*, the polar obstacle density h_k is then calculated by:

$$h_k = \sum_{i,j} m_{i,j} * h'_{i,j}$$

with: $h'_{i,j} = 1$ if $k*\alpha \in \left[\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}\right]$

$h'_{i,j} = 0$ otherwise

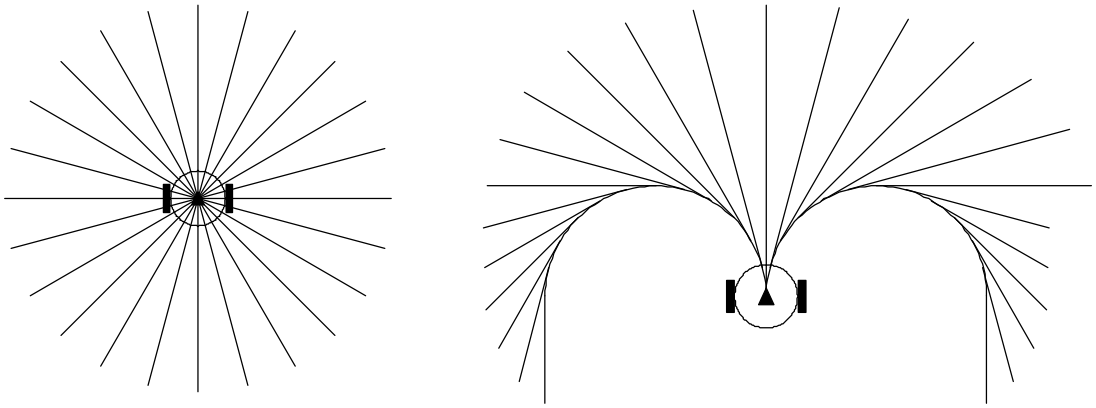
The result of this process is a polar histogram that takes in account the size of the robot. The resulting polar histogram is a good approximation of a polar histogram built by the standard VFH method applied to an enlarged map. However, this method is much more efficient as it does both steps in one step.

The *h'* function also serves as a low-pass filter. As a consequence, the filter used in the original VFH method is not necessary anymore.

7.2.3 Consideration of the Robot Trajectory

The original VFH method takes neither the dynamics nor the kinematics of the robot into account. It automatically assumes that the robot is able to change its direction of travel instantly as shown in Figure 36a. Unless the robot stops at every sampling time, this assumption is clearly violated.

The new proposition assumes that the robot's trajectory is based on arcs of a circle (constant curvature curves) and straight lines. This assumption is a simple, but close approximation of the real trajectory of a mobile robot. Examples of such trajectories are shown in Figure 36b.



a) b)

Figure 36: Approximation of trajectories: a) without dynamics, b) with dynamics

The curvature k of a curve is defined by:

$$k = 1/r$$

The trajectory curvature of a mobile robot is often a function of the robot velocity. The faster the robot travels, the bigger the radius of the trajectory curve and the smaller its curvature. For certain mobile robots, e.g. the GuideCane, these values are even different for right and left turns. For example, a right-handed user can do much sharper turns to the left than to the right.

For a differential drive mobile robot that is stopped or is turning on the spot, the minimum steering radius can be zero. For other mobile robots, like the current GuideCane, or robots based on the *Ackerman steering* or the *tricycle drive* (new proposition of GuideCane) mechanism, the minimum steering radius is never zero.

The values for the minimum steering radius as a function of the robot velocity can easily be measured. For the GuideCane, the minimum steering radius is independent of the velocity. We define these radius for both sides:

$$r_r = 1/k_r$$

$$r_l = 1/k_l$$

With these values and the histogram grid, we can determine which sectors are blocked by obstacles. An example with two obstacles is shown in Figure 37. Again, to take into account the size of the robot, the obstacles are enlarged by r_{r+s}. We can see that if the trajectory circle and the enlarged obstacle overlap, all directions from the obstacle to the backwards direction of motion are blocked by the obstacle. In our example, obstacle A blocks all directions to its left because of the robot dynamics. On the other hand, obstacle B does not block the directions to its right.

With the original VFH method, the directions to the left of obstacle A are considered to be suitable directions of motion. Therefore, if a new desired direction of travel to the left was entered, the GuideCane would start to turn to the left and hit obstacle A.

With the VFH+ method, the robot would proceed to go between the obstacles A and B and make a left turn after obstacle A was cleared.

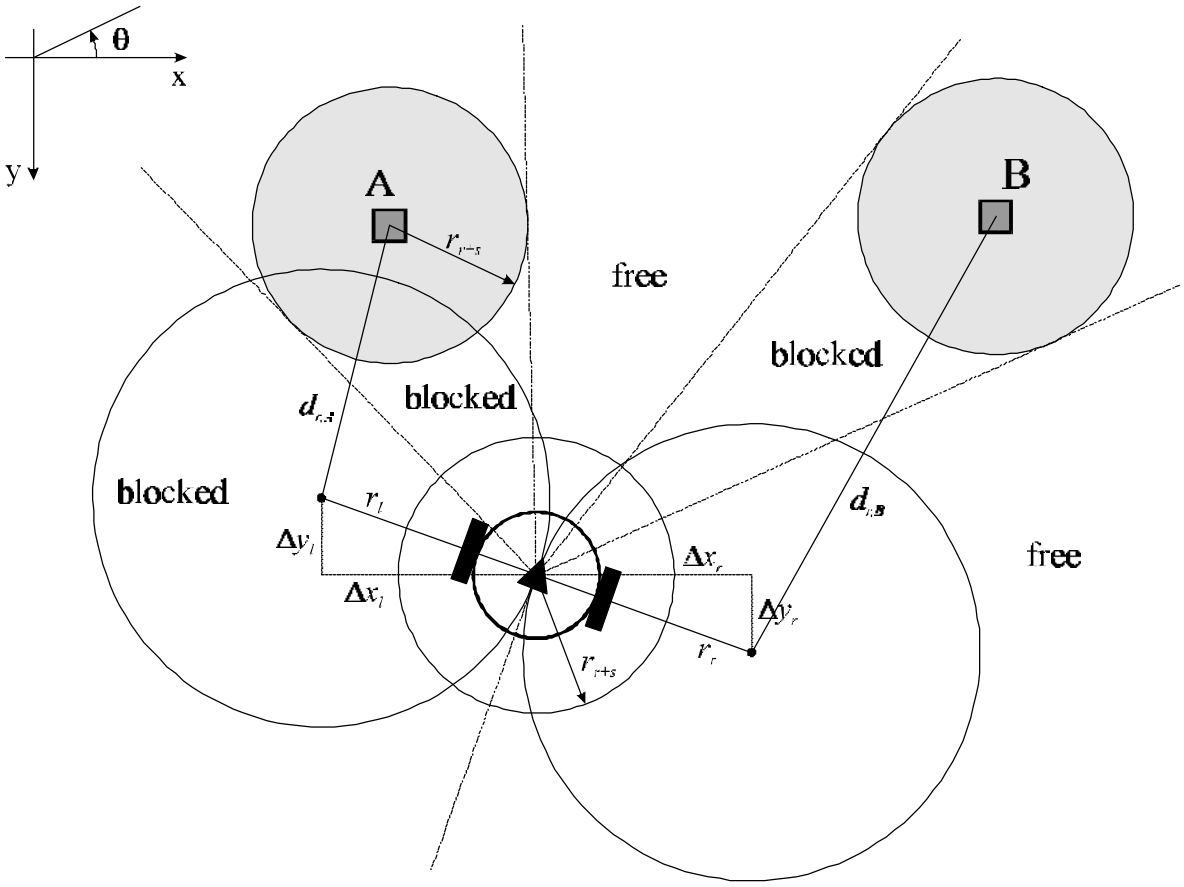


Figure 37: Example of blocked directions

The position of the right trajectory center relative to the robot position is defined by:

$$\Delta x_r = r_r * \sin q$$

$$\Delta y_r = r_r * \cos\theta$$

Similar for the left trajectory center:

$$\Delta x_l = -r_l * \sin\theta$$
$$\Delta y_l = -r_l * \cos\theta$$

Now, we can calculate the distance from a histogram cell $c_{i,j}$ to the two trajectory centers:

$$d_r^2 = \left(\Delta x_r - \Delta x(j)\right)^2 + \left(\Delta y_r - \Delta y(i)\right)^2$$
$$d_l^2 = \left(\Delta x_l - \Delta x(j)\right)^2 + \left(\Delta y_l - \Delta y(i)\right)^2$$

An obstacle blocks the directions to its right if:

$$d_r^2 < \left(r_r + r_{r+s}\right)$$

And an obstacle blocks the directions to its left if:

$$d_l^2 < \left(r_l + r_{r+s}\right)$$

By checking every cell in the active window with these two conditions, we will get two limit angles, θ_r for right angles and θ_l for left angles. We also define θ_b as the direction backwards to the current direction of motion:

$$\theta_b = \theta + \pi$$

This method can be implemented very efficiently by an algorithm that only considers cells that have an influence on either θ_r or θ_l. The algorithm is the following:

1) Determine θ_b. Set θ_r and θ_l equal to θ_b.
2) For every cell $c_{i,j}$ in the active window C^* with a CV > τ:
   a) if $\beta_{i,j}$ is to the left of θ and to the right of θ_l, update θ_l.
   b) if $\beta_{i,j}$ is to the right of θ and to the left of θ_r, update θ_r.

If the robot sensors are not very reliable, θ_r and θ_l could also be determined in a more stochastic way. Instead of comparing the cell certainty value to a threshold, one could build a histogram whose sector values indicate the certainty that it is blocked because of the robot dynamics. By applying a threshold to this histogram, one could get the values for θ_r and θ_l. However, the first method is more efficient so that this method should only be applied if the sensors are not very reliable.

With j_r, j_l, and the binary polar histogram, we can build the *masked polar histogram*:

$$H_{k,i}^m = 0 \qquad \text{if } H_{k,i}^b = 0 \text{ AND } (k * a) \in \{[j_r, q] \wedge [q, j_l]\}$$
$$H_{k,i}^m = 1 \qquad \text{otherwise}$$

The masked polar histogram tells the robot which directions of motion are possible at the current speed. If all values were set to one, the robot could not proceed at the current speed. The robot would have to determine a set of new values (j_r, j_l) based on a slower speed. If the masked polar histogram was still set to one for every direction, the robot would need to stop immediately.

In Figure 38, the polar histogram, the binary polar histogram, and the masked polar histogram are shown for our example. The binary polar histogram incorrectly indicates that the directions to the left of obstacle A are free. The masked polar histogram correctly blocks these directions.

Note that the vector magnitudes for obstacle A are bigger than for obstacle B. The reason is that obstacle A is closer to the robot. Also note that obstacle A occupies more sectors than obstacle B. As obstacle A is closer to the robot, its enlargement angle is bigger.

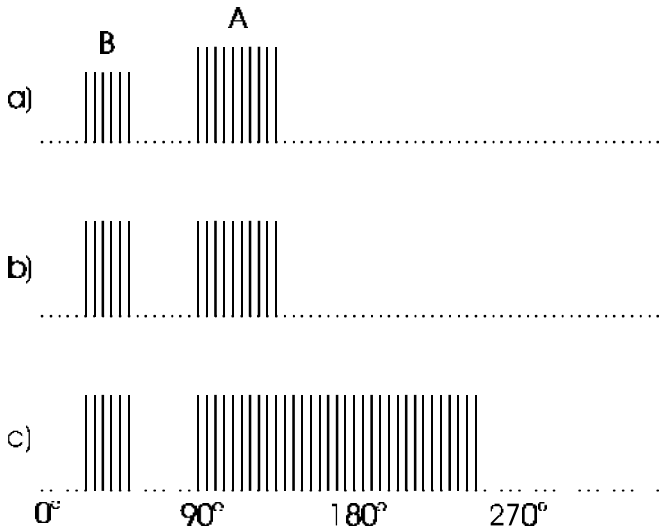


Figure 38: a) Polar histogram, b) binary polar histogram, c) masked polar histogram

7.2.4 Cost-Based Direction Selection

The masked polar histogram shows which directions are free of obstacles and which ones are blocked. However, some free directions are better candidates than others for the new direction of motion.

The original VFH method selects the *candidate valley* that most closely matches the direction to the target direction k_t. This method is very goal-oriented as it is based solely on the difference between the target direction and the candidate valley direction. The candidate direction is then determined dependent on the size of the valley.

The new improved method first finds all valleys in the masked polar histogram and then determines all possible candidate directions. A cost function that takes into account more than just the difference between the candidate and the target direction, is then applied to these candidate directions. The candidate direction k_n with the lowest cost is then chosen to be the new direction of motion j_n:

$$\mathrm{j}_n = \mathrm{a} * k_n$$

In the first step, the right and left borders k_r and k_l of all valleys in the masked polar histogram are determined. Similar to the original VFH method, two types of valleys are distinguished, namely, *wide* and *narrow* ones. A valley is considered wide if the difference between its two borders is bigger than s_{max} sectors (in our system s_{max} = 16). Otherwise, the valley is considered narrow.

For a narrow valley, there is only one candidate direction so that the robot drives through the center of the gap between the corresponding obstacles:

$$c_n = \frac{k_r + k_l}{2} \qquad \text{centered direction}^+$$

For a wide valley, there are two to three candidate directions dependent on the target direction:

$$c_r = k_r + \frac{s_{\max}}{2} \qquad \text{direction towards the right side of the valley}^+$$

$$c_l = k_l - \frac{s_{\max}}{2} \qquad \text{direction towards the left side of the valley}^+$$

$$c_t = k_t \quad \text{if} \quad k_t \in \left[k_r, k_l\right] \qquad \text{direction equal to the target direction}$$

The candidate directions c_r and c_l make the robot follow an obstacle contour at a safe distance, while c_t leads the robot towards the target direction.

[+] Because of the histogram boundaries, some care must be taken when applying these equations.

For robots that are not goal-orientated, other candidate directions could be added. For example, for a robot that should randomly explore its environment, we could add the following candidate directions:

$$c_q = k_q \quad \text{if } k_q \in [k_r, k_l] \qquad \text{direction equal to current direction of motion}$$

or:

$$c_j = k_j \quad \text{if } k_j \in [k_r, k_l] \qquad \text{direction equal to previous direction}$$

In the case of the goal-orientated robot, we get between one and three candidate directions for each valley in the masked polar histogram. Next, we need to define an appropriate cost function, so that the robot selects the most appropriate candidate direction as its new direction of motion j_n. We propose the following cost function g as a function of a candidate direction c:

$$g(c) = \mu_1 * \Delta(c, k_t) + \mu_2 * \Delta\left(c, \frac{\theta_i}{\alpha}\right) + \mu_3 * \Delta(c, k_{n,i-1})$$

where $\Delta(c_1,c_2)$ is a function that computes the absolute angle difference between two sectors c_1 and c_2 so that the result is $\leq \alpha/2$. One possible implementation is:

$$\Delta(c_1, c_2) = \min\{|c_1 - c_2|, |c_1 - c_2 - \alpha|, |c_1 - c_2 + \alpha|\}$$

The first term represents the cost associated with the difference of a candidate direction and the target direction. The bigger this difference is, the more the candidate direction will guide the robot away from its target direction, and hence the bigger the cost.

The second term represents the cost associated with the difference of a candidate direction and the robot's wheel orientation. The larger this difference, the larger the required change of the direction of motion is.

The third term represents the cost associated with the difference of a candidate direction and the previously selected direction of motion. The bigger this difference, the bigger the change of the new steering command is. This term has also a short-term memory effect.

In short, the first term is responsible for the goal orientated behavior, while the second and third term make the mobile robot commit to a direction. The need for a commitment effect can be explained with an example as shown in Figure 39. This is a typical situation where the mobile robot encounters a small obstacle. The two possible candidate directions are to the right or to the left of the obstacle. Both candidate directions deviate about the same amount of the target direction. If the cost function consisted of only the first term, similar to the original VFH method, both candidate directions would seem to be equally good. In tests with a mobile robot, the robot's heading often switched several times between the two directions. It seemed as if the robot did not make up its mind on which side to avoid the obstacle. As a result, the robot could get very close to the obstacle, and in some cases even crash into it. The problem is that with the first term alone, the mobile robot does not commit to either direction, so that small changes in its world representation can make it change its mind.

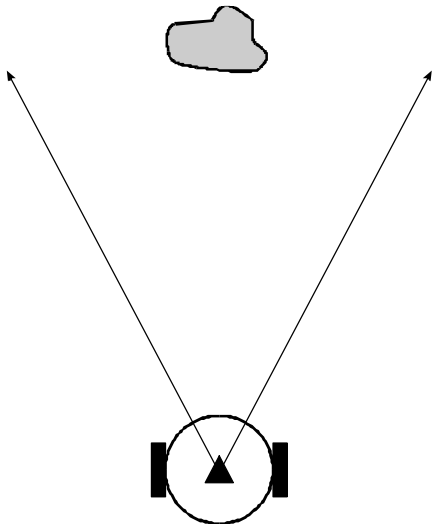


Figure 39: Need for short-term memory

This problem is overcome with the second and third term. These terms provide a form of short-term memory to the robot. The second term is similar to a mechanical memory. However, this term alone would not completely fix the problem as the time between two samples is often too small for an observable change in the robot's orientation. For this reason, the third term is added. With the help of this term, the robot commits to a direction even before its orientation has changed.

The higher μ_1 is, the more goal oriented the robot's behavior is. The higher μ_2 is, the more the robot tries to execute an efficient path with a minimum change of direction of motion. The higher μ_3 is, the more the robot tries to head towards the previously selected direction and the smoother is the trajectory.

The absolute values of the three parameters are not important. Only the relation between the three parameters is important. To guarantee a goal orientated behavior, the following condition must be satisfied:

$$\mu_1 > \mu_2 + \mu_3 \qquad \text{[condition 1]}$$

If this condition was not satisfied, the robot would try to continue towards its current direction of motion even if it could head towards the target direction.

If an efficient path is more important than variations in the steering commands, then μ_2 should be set higher than μ_3. Otherwise, if the smoothness of the steering commands is more important than the efficiency of the robot trajectory, then μ_3 should be set higher than μ_2.

Experiments have shown that a good set of parameters for a goal oriented mobile robot is:

$$\mu_1 = 10, \qquad \mu_2 = 4, \; \mu_3 = 4$$

It is also possible to add other terms to the cost function. For example, we can make the mobile robot avoid narrow openings like doors by adding a term that takes in account the valley width:

$$\mu_4 * \Delta(k_r, k_l)$$

On the other hand, the cost function could also be temporarily modified to make the mobile robot look for and go through narrow openings by adding the following term:

$$\mu_4 * \frac{1}{\Delta(k_r, k_l)}$$

The cost function allows the user to develop a more subtle behavior than the coarse approach of the original VFH method. With the current cost function, the second and third term are responsible for a more efficient and smoother trajectory. Another advantage is that the mobile robot behavior can easily be changed by modifying either the cost function parameters or the cost function itself.

7.2.5 Performance of the VFH+ Method

In short, the VFH+ method is very fast and very effective. On a PC 486 running at 67 MHz, the algorithm takes less than 6 ms in the worst case. The speed of an obstacle avoidance algorithm is essential for two reasons. Firstly, the higher the sampling rate of the obstacle avoidance method is, the faster the robot can travel without oscillations and without risk of bumping into an obstacle. Actually, due to the speed of this algorithm, the robot speed is now limited by the sampling rate of its sensors. Secondly, the faster the obstacle avoidance method is, the more computational power can be used for usually time-consuming high-level behaviors.

The VFH+ method is about as fast as the original VFH method. Only the consideration of the robot trajectory requires additional computation time. However, due to the consideration of the robot size, the addition of a small VFF component is not required anymore, so that the total time of these algorithms should be about the same.

The performance of the VFH+ method is superior over the original VFH method. The robot trajectory is smoother and has fewer oscillation due to the threshold hysteresis and the commitment effect of the cost function. With the consideration of the robot size, the robot goes nicely around corners without the need for VFF. It also allows one to easily apply this method to a robot of any size without much parameter adjusting. It especially eliminates the need to find a suitable smoothing filter, which was usually done through long procedures of trial and error.

The main improvement of the VFH+ method is the consideration of the robot trajectory. With the original VFH method, the robot could be directed into an obstacle if a new desired direction of motion was entered at the wrong time. The performance of the original VFH method also degraded when the robot's orientation was more than 90° different from the target direction. The VFH+ method eliminates these problems.

Another very nice feature of the VFH+ method is that it is very insensitive to its parameter values. As long as condition 1 is satisfied and the parameter values are selected reasonably, the VFH+ method performs well. Unlike most other obstacle avoidance methods, the VFH+ method requires almost no time adjusting the parameter values.

The VFH+ method allows the GuideCane user to travel at speeds up to 1 m/s. The speed is actually not limited by the VFH+ performance, but by the number of sonars and their sampling rate.

7.3 VFH*

Although the performance of the VFH+ obstacle avoidance method is very satisfactory, this method sometimes directs the mobile robot into dead-ends that could be avoided. To overcome this problem, the VFH* method was developed. This method is a combination of the VFH+ method and the A* search algorithm

7.3.1 Extremely Local Nature of VFH Method

Both, the original VFH and the VFH+ method, are *extremely local* obstacle avoidance methods. Their purpose is to provide fast local obstacle avoidance behavior to a mobile robot. The robot's high-level planning is either done by a global planner in the case of an autonomous robot, or by a human in the case of a tele-operated robot.

The task of an obstacle avoidance algorithm is to find a suitable path around local obstacles, not to plan a path to the goal. This explains the need for a fast *local* obstacle avoidance algorithm. However, in certain situations, the VFH+ method is too local to effectively guide the mobile robot around obstacles.

The VFH+ method analyzes only the robot's immediate surroundings before deciding to head towards a seemingly appropriate direction. This reasoning is extremely local as it does not look ahead to verify if the proposed new direction of motion j_n will guide the robot around the obstacle or if it will guide it into a dead-end.

For a better understanding of the local nature of the VFH+ method, let's analyze the robot's encounter with an obstacle as shown in Figure 40. Remember that the VFH amplitude function is point symmetrical with a radius r_{VFH}. A certainty cell is only considered by the VFH+ method if its distance to the robot is smaller than $r_{VFH.}$ Because of the function's symmetry, any obstacle getting inside this radius first appears in the polar histogram as if it was a small obstacle independent of its real shape. As a result, the polar histogram indicates two possible directions: to the right or to the left of this obstacle part. This works fine as long as the obstacle is really as small as indicated by the polar histogram. However, if the obstacle is bigger, as shown in Figure 40, one direction is often much better than the other one. While the direction to the right would lead the robot around the obstacle, the direction to the left would lead it into the corner where it would end up being trapped. By the time the full corner is represented in the polar histogram, the robot is already too close to get out of the corner without stopping or dramatically slowing down.

Whenever the robot encounters an obstacle that should be avoided on one specific side, the VFH+ method chance of choosing the right direction is only about 50%. If there are several successive local dead-ends, these methods will lead the mobile robot very likely into a dead-end.

It is important to note that increasing r_{VFH} would not reduce this problem. Independent of this parameter, an obstacle always looks narrow during the early approach. Another approach of temporarily increasing r_{VFH} when an obstacle is detected and resetting it when the obstacle is cleared also fails in many situations. One such example is if a new obstacle is encountered while the robot is still moving around another obstacle. In addition, increasing r_{VFH} is also a dangerous approach as it may unnecessarily block all directions as explained in section 7.1.2.

It is also important to note that an obstacle avoidance algorithm can only be blamed for guiding a robot into dead-ends that are represented as such in its map while it still has the possibility to avoid them. However, the sensor look-ahead of most mobile robots is usually bigger than r_{VFH}. In the case of the GuideCane, r_{VFH} is set to 1.5 m while the look-ahead of its sonars is 2.5 m. This means that the VFH+ method only uses the information stored in the certainty grid inside a range of 1.5 m even though it would have information for the range of 2.5 m. In the situation shown in Figure 40, the obstacle is likely to be represented as a corner in its map. Therefore, even though the map representation would provide enough information to avoid the dead-end, the VFH+ method leads the robot into the dead-end with a chance of 50%.

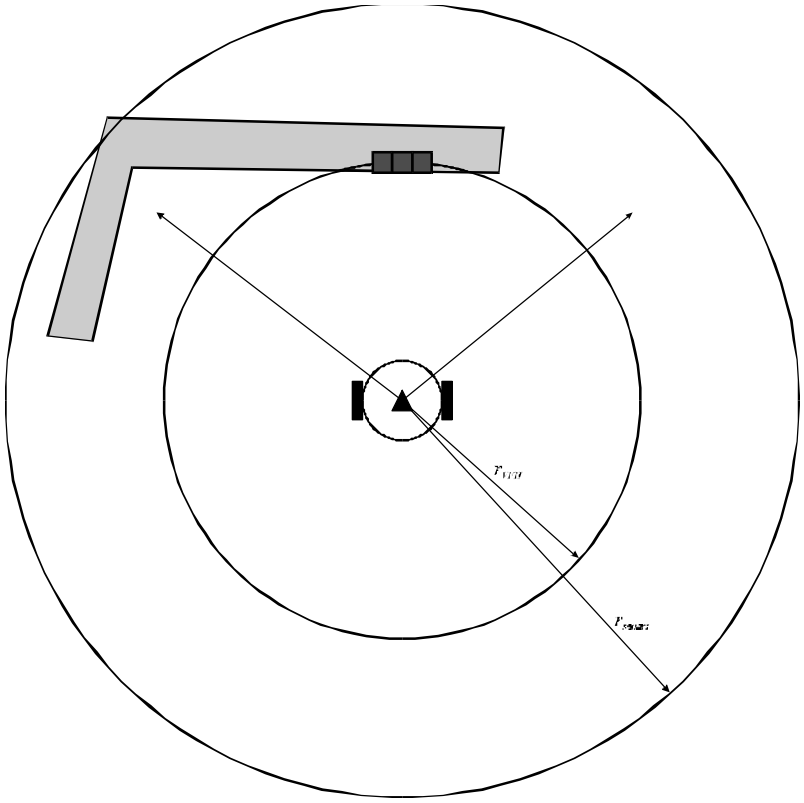


Figure 40: Problematic obstacle shape for a local obstacle avoidance algorithm

7.3.2 Local Planning

The only way to overcome this problem is to make the obstacle avoidance algorithm less local. This can be done by adding a little planning to the existing method, resulting in a dynamic re-planner with a smooth cut-off. The term "little" is used as it is impracticable to plan a long way ahead, unless the robot has an accurate and reliable representation of its environment. However, if the robot had such a map, it could simply rely on its global path planner without requiring a local obstacle avoidance algorithm. As most environments are dynamic, planning far ahead can be very inefficient.

In most cases, the obstacle avoidance algorithm should not plan much further ahead than it can reliably detect obstacles with its sensors. With its sensors, the robot has more certainty about its immediate surrounding than about its path ahead.

The general idea of the VFH* method is to project the outcome of the trajectories for each current candidate direction. These directions are called *primary candidate directions*. We compute for each primary candidate direction the new position and orientation the robot would have after having moved for a projected distance of d_p in this direction. At every projected position, a new polar histogram is build. These histograms are then again analyzed for candidate directions, called *projected candidate directions*. By repeating this process n_g times, we get a search tree of depth n_g, where the *end nodes* (goals) correspond to a total projected distance of $d_t = n_g * d_p$. Therefore, the goal of the search is to find a projected trajectory of distance d_t.

Nodes in the search tree represent the projected positions and orientations of the mobile robot. Branches represent the candidate directions leading from one position to another.

For every candidate direction, a cost $g(n)$ is calculated similar to the cost function defined in section 7.2.4. The cost associated with a node is the sum of the costs associated with the branches leading to this node. The primary candidate direction that leads to the end node with the minimum total cost is then selected as the new direction of heading j_n.

7.3.3 A* Search

For a better understanding of the concept of the local planner, the VHF* method was described in section 7.3.2 as if it was based on the *breadth-first search* algorithm. In practice, it is more efficient to implement a *uniform cost search* algorithm based on the cost function. This way, the node with the lowest cost is expanded next. As the projected total path cost never decreases, this search method is optimal and complete.

To make the search more efficient, we can introduce a heuristic function $h(c)$ similar to the cost function. This allows us to apply the *A* search* method which is more efficient, but still optimal and complete as long as the heuristic function never overestimates the cost to reach the goal [33]. The estimated cost of the cheapest solution $f(c)$ is then defined by:

$$f(c) = g(c) + h(c)$$

7.3.4 Search Parameters

An important parameter is the total projected distance d_t. The goal depth n_g is proportional to this parameter. The minimum value of zero makes this algorithm similar to the VFH+ method with its extremely local behavior.

The higher d_t is selected, the bigger is the planner look-ahead, and the better are the results of this method. However, if this parameter is selected too high, the obstacle avoidance algorithm will be substantially slowed down as it is $O(b^{n_g})$. In any case, it is not recommended to choose a total projected distance value that exceeds by far the look-ahead of the robot sensors, unless the robot has an accurate map of a mostly static environment.

Therefore, the selection of d_t is a trade-off between the algorithm speed and the quality of the algorithm result. A good compromise is to set this parameter equal to the look-ahead of the sensors.

Another important parameter is the length d_p of how far the new position is projected ahead at each step. This parameter together with d_t determines the goal depth $n_g = d_t/d_p$.

If d_p is selected too big, the new position might be projected across or right into an obstacle. As this parameter value increases, the higher the risk of such an incorrect projection becomes. As a result of such a projection, the search method could come up with a new direction of heading j_n that is not desirable.

If d_p is selected too small, the effect of a projection is too small and a high value of n_g would be required. This would result in an unnecessary deep search tree that would require too much computation resources and would substantially slow down the obstacle avoidance algorithm.

Therefore, the selection of d_p is a trade-off between the algorithm speed and the validity of the result. Experiments have shown that a good compromise is to set this parameter equal to the diameter of the robot.

7.3.5 The Expansion Step

The expansion of a node consists of building the masked polar histogram at the node's projected position, determining the corresponding candidate directions, calculating the projected position and orientation of the following nodes, determining the cost of reaching these nodes, and determining their heuristics.

The first two steps, building the masked polar histogram and determining the corresponding candidate directions, are done the same way as in the VFH+ algorithm. The other three steps are described in more detail in the following sections.

7.3.5.1 Projection of Position and Orientation

The computation of the projected position and orientation for a candidate direction φ_c can be done in many ways. In our approach, the projected robot trajectory is again approximated by arcs of a circle and straight lines. This approximation is a good trade-off between accuracy and speed of the algorithm. Examples of such trajectories are shown in Figure 41:

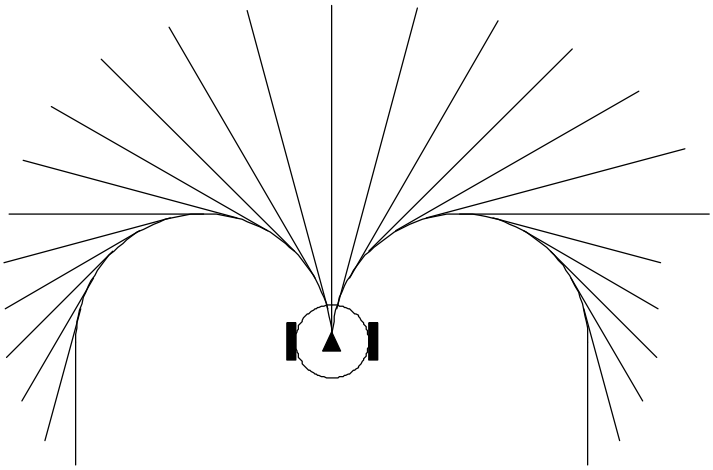


Figure 41: Projected trajectories

The current position and orientation of the robot are defined as x_i, y_i and φ_i. The projected variables are defined as x_{i+1}, y_{i+1} and φ_{i+1}. The parameters are the projected distance d_p and the minimum steering radius r_r and r_l as defined in section 7.2.3.

7.3.5.1.1 The Projection Equations

For a given candidate direction φ_c, we need to determine first if the robot can reach this orientation during the projected distance. If not, the trajectory is simply approximated by a constant curvature curve. The maximum direction to the right is defined by:

$$\varphi_r = \varphi_i - \frac{d_p}{r_r}$$

Similar for the maximum direction to the left:

$$\varphi_l = \varphi_i + \frac{d_p}{r_l}$$

We also need to distinguish between candidate directions to the right and to the left of the robot, as the corresponding equations are different. We end up with four sets of equations:

1. If the candidate direction φ_c is to the right of the robot and it exceeds θ_r:

$$x_{i+1} = x_i + \Delta x_{r,\max} * \cos\theta_i + \Delta y_{r,\max} * \sin\theta_i$$
$$y_{i+1} = y_i - \Delta x_{r,\max} * \sin\theta_i + \Delta y_{r,\max} * \cos\theta_i$$
$$\theta_{i+1} = \theta_r$$

with: $$\Delta x_{r,\max} = r_r * \sin\frac{d_p}{r_r}$$

$$\Delta y_{r,\max} = r_r * (1 - \cos\frac{d_p}{r_r})$$

2. If the candidate direction φ_c is to the left of the robot and it exceeds θ_l:

$$x_{i+1} = x_i + \Delta x_{l,\max} * \cos\theta_i + \Delta y_{l,\max} * \sin\theta_i$$
$$y_{i+1} = y_i - \Delta x_{l,\max} * \sin\theta_i + \Delta y_{l,\max} * \cos\theta_i$$
$$\theta_{i+1} = \theta_l$$

with: $$\Delta x_{l,\max} = r_l * \sin\frac{d_p}{r_l}$$

$$\Delta y_{l,\max} = r_l * (\cos\left(\frac{d_p}{r_l}\right) - 1)$$

It is important to note that if r_r and r_l are constant, the values for $\Delta x_{r,max}$, $\Delta y_{r,max}$, $\Delta x_{l,max}$ and $\Delta y_{l,max}$ need only be initialized once at the start of the program.

3. If the candidate direction φ_c is to the right of the robot but does not exceed θ_r:

$$x_{i+1} = x_i + \Delta x_r * \cos\theta_i + \Delta y_r * \sin\theta_i + \Delta d_s * \cos(\varphi_c)$$
$$y_{i+1} = y_i - \Delta x_r * \sin\theta_i + \Delta y_r * \cos\theta_i - \Delta d_s * \sin(\varphi_c)$$
$$\theta_{i+1} = \varphi_c$$

with: $$\Delta x_r = r_r * \sin(\theta_i - \varphi_c)$$

$$\Delta y_r = r_r * (1 - \cos(\theta_i - \varphi_c))$$

$$\Delta d_s = d_p - r_r * (\theta_i - \varphi_c)$$

4. If the candidate direction φ_c is to the left of the robot but does not exceed θ_l:

$$x_{i+1} = x_i + \Delta x_l * \cos\theta_i + \Delta y_l * \sin\theta_i + \Delta d_s * \cos(\varphi_c)$$
$$y_{i+1} = y_i - \Delta x_l * \sin\theta_i + \Delta y_l * \cos\theta_i - \Delta d_s * \sin(\varphi_c)$$
$$\theta_{i+1} = \varphi_c$$

with:
$$\Delta x_l = r_l * \sin(\varphi_c - \theta_i)$$
$$\Delta y_l = r_l * \left(\cos(\varphi_c - \theta_i) - 1\right)$$
$$\Delta d_s = d_p - r_l * (\varphi_c - \theta_i)$$

7.3.5.1.2 The Projection Look-Up Lists

A computationally much more efficient solution is to calculate the projected values for a certain orientation θ_0, to store the results of different candidate directions φ_c in look-up lists, and to use this data for any orientation through a simple rotary transformation.

Based on the previous equations, we can calculate the values Δx_0, and Δy_0 as a function of $\Delta\varphi = (\varphi_c - \theta)$ for an arbitrary orientation of $\theta_0 = 0°$. These values are then stored in look-up lists. In the current implementation, these values are stored in increments of 1° for $\Delta\varphi$. For any orientation θ_i, the corresponding values can then be calculated by:

$$x_{i+1} = x_i + \Delta x_0(\Delta\varphi) * \cos\theta_i + \Delta y_0(\Delta\varphi) * \sin\theta_i$$
$$y_{i+1} = y_i - \Delta x_0(\Delta\varphi) * \sin\theta_i + \Delta y_0(\Delta\varphi) * \cos\theta_i$$

with: $\Delta\varphi = (\varphi_c - \theta) \in [-\pi, \pi]$ in steps of 1°

The equations to initialize the two look-up lists are as follows:

1. If $\Delta\varphi < -\frac{d_p}{r_r}$:

$$\Delta x_0(\Delta\varphi) = r_r * \sin\frac{d_p}{r_r}$$
$$\Delta y_0(\Delta\varphi) = r_r * (1 - \cos\frac{d_p}{r_r})$$

2. If $\Delta j > \frac{d_p}{r_l}$:

$$\Delta x_0(\Delta j) = r_l * \sin\frac{d_p}{r_l}$$
$$\Delta y_0(\Delta j) = r_l * (\cos\left(\frac{d_p}{r_l}\right) - 1)$$

3. If $\Delta j \in \left[-\frac{d_p}{r_r}, 0\right]$:

$$\Delta x_0(\Delta j) = -r_r * \sin(\Delta j) + \left(d_p + r_r * \Delta j\right) * \cos(\Delta j)$$
$$\Delta y_0(\Delta j) = r_r * \left(1 - \cos(\Delta j)\right) - \left(d_p + r_r * \Delta j\right) * \sin(\Delta j)$$

4. If $\Delta j \in \left[0, \frac{d_p}{r_l}\right]$:

$$\Delta x_0(\Delta j) = r_l * \sin(\Delta j) + \left(d_p - r_l * \Delta j\right) * \cos(\Delta j)$$
$$\Delta y_0(\Delta j) = r_l * \left(\cos(\Delta j) - 1\right) - \left(d_p - r_l * \Delta j\right) * \sin(\Delta j)$$

7.3.5.2 Cost Function

In section 7.2.4, we defined the cost function for a primary candidate direction c_0 leading from the root node at depth 0 to its successor node as following:

$$g(c_0) = \mu_1 * \Delta(c_0, k_t) + \mu_2 * \Delta\left(c_0, \frac{\theta_i}{\alpha}\right) + \mu_3 * \Delta(c_0, k_{n,i-1})$$

The purpose of the three terms is the same as in the cost function of the VFH+ method. While the first term is responsible for the goal orientated behavior, the other two terms make the robot commit to a direction. For a goal orientated robot, we still have the following condition:

$$\mu_1 > \mu_2 + \mu_3 \qquad \text{[condition 1]}$$

For a projected candidate direction c_n of a node at depth n bigger than zero, we propose a slightly modified cost function as following:

$$g_n(c_n) = \lambda^n * \left[\mu'_1 * \max\left\{\Delta(c_n, k_t), \Delta(k_p, k_t)\right\} + \mu'_2 * \Delta\left(c_n, \frac{\theta_n}{\alpha}\right) + \mu'_3 * \Delta(c_n, c_{n-1}) \right]$$

with: $k_p = -\arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right)$

and: $0 < \lambda \leq 1$

In short, the first term is responsible for the goal-orientated behavior, while the second and third term give preference to projected trajectories that are smooth and efficient.

The first term represents again the cost associated with the deviation from the target direction, resulting in the goal orientated behavior of this obstacle avoidance method. However, this term is slightly different for a projected candidate direction than it is for a primary candidate direction.

In the case of a projected candidate direction, this term also considers the effective direction of motion, or in other terms, the forward progress of a trajectory. There is an important difference between a candidate direction c_n and the corresponding effective direction of motion k_p. Ideally, we want them both to be in the same direction as the target direction. However, dependent on the robot's current orientation, it is possible for either of them to be equal to the target direction while the other one deviates largely from it.

If we were not considering the effective direction of motion, a part of the projected trajectory could be very cheap even though it does not make any forward progress. Such an example, where the cost associated with the first term would even be zero, is shown in Figure 42. With the modified first term, the cost of this trajectory is correctly very high, as it provides no forward progress at all.

It is important to note that the first term of the cost function for a primary candidate direction does not consider the effective direction of motion. If the robot can select a primary candidate direction that is close to the target direction, the associated cost should be small even if the corresponding effective direction of motion deviates largely from the target direction. The effective direction of motion for a primary candidate direction depends a lot on the current orientation, while the effective direction of motion for a projected candidate direction depends on the projected trajectory. As the robot has no control over the current orientation, there is no reason to associate a cost with the effective direction of motion of a primary candidate direction. On the other hand, the robot has control over its projected trajectory, so that it makes sense to associate a cost with the effective direction of motion of a projected candidate direction.

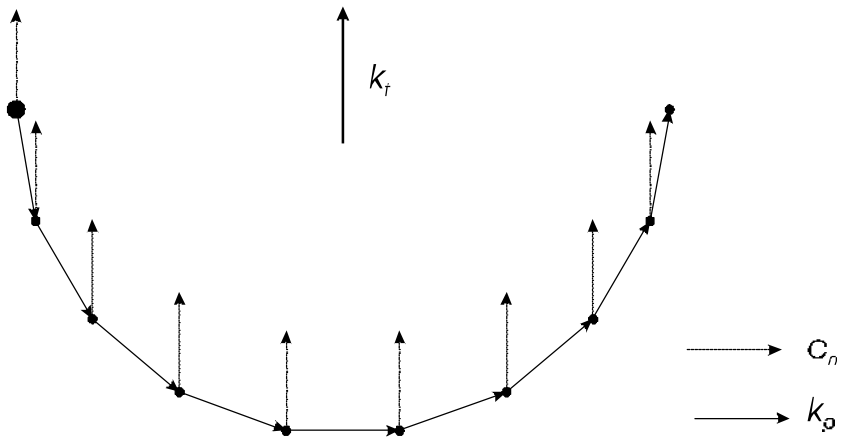


Figure 42: Effective direction of motion k_p

The second and third term have a different meaning for a projected candidate direction than for a primary candidate direction. In the case of a primary candidate direction, these terms represent a short-term memory effect that makes the robot commit to a direction. In the case of a projected candidate direction, the second term represents the cost associated with the efficiency of a projected trajectory, while the third term represents the cost associated with its smoothness. These terms quantify the quality of a projected trajectory. These terms are not necessary, but they can provide a better trajectory.

The higher μ_1' is, the more goal oriented the robot's behavior is. The higher μ_2' is, the more the robot tries to find an efficient path. The higher μ_3' is, the more the robot tries to find a smooth path.

The absolute values of the three parameters are again not important. Only the relation between them is important. For a goal oriented robot, the following condition must be satisfied:

$$\mu_1' > \mu_2' + \mu_3' \qquad \text{[condition 2]}$$

To emphasize the importance of a primary candidate direction over a projected candidate direction, the following condition must also be satisfied:

$$\mu_1 \geq \mu_1' \qquad \text{[condition 3]}$$

Experiments have shown that a good set of parameters for a goal oriented mobile robot is:

$$\mu_1 = 10, \quad \mu_2 = 4, \quad \mu_3 = 4$$
$$\mu_1' = 10, \quad \mu_2' = 2, \quad \mu_3' = 2$$

Another important parameter is the *discounting factor* λ. Instead of giving equal weight to all candidate directions, they are weighted by a factor λ^n. Therefore, for a candidate trajectory, the cost of its candidate directions (branches) decreases exponentially with rate λ. There are three reasons for the introduction of this factor.

First of all, it decreases the problem of having a fixed goal depth n_g which results in a sharp cut-off. Without λ, all branches would have the same weight and the obstacle avoidance algorithm would not always behave as we desired. An example of such a case with n_g set to 7 is shown in Figure 43. Without λ, the total cost of trajectory B could be cheaper than the one of trajectory A. As a result, the robot could keep going towards the right without ever trying to get closer to wall #1. The reason for this behavior is the fixed goal depth n_g and the associated sharp cut-off. Without λ, the obstacle avoidance algorithm has a tendency to find trajectories that stop shortly before being influenced by an obstacle. If n_g was increased by just one, trajectory B would become much more costly, maybe even more costly than trajectory A. However, even if trajectory B became more costly than A, the obstacle avoidance algorithm would pick trajectory C as its cheapest trajectory, again trying to stop shortly before being influenced by an obstacle. By introducing λ, the cut-off at the last branch is less sharp as the weight of the last branch becomes much smaller. Consequently, trajectory A of our example becomes clearly cheaper than trajectory B. If we set λ higher than one, the obstacle avoidance would tend even more to find trajectories that stop shortly before an obstacle gets inside the histogram range.

Secondly, the discounting factor l takes in account the uncertainty of the map information. Due to its sensors, the mobile robot has more certainty about its immediate surroundings. The further the projected position is away from the current position, the more uncertain is the content of the map at that position. Therefore, it makes sense to weight the branch cost according to its node depth.

Finally, by simply adding all branch costs together without l is the same as setting l equal to one. With the introduction of l , we have more control about the weights of the branches at different depths. By having the possibility to adjust l , we gain more control over the search algorithm's behavior.

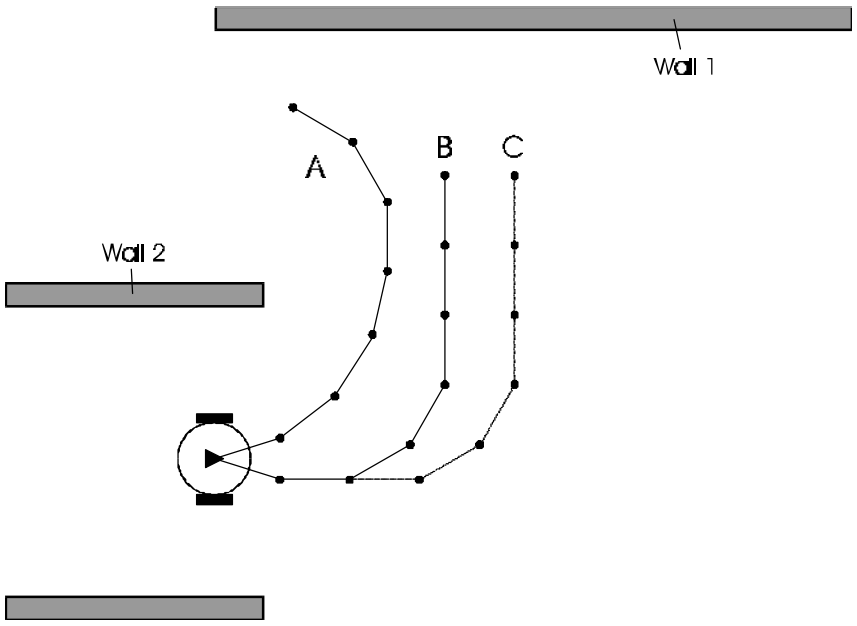


Figure 43: Necessity of l

7.3.5.3 Heuristic Function

The heuristic function $h(c)$ is the estimated cost of the cheapest path from the state at node n to a goal state. A function is an admissible heuristic if it never overestimates the cost to reach the goal.

With condition 2 satisfied, the cost is cheapest if the robot can head towards the goal direction k_t at every following node. We can get a simple admissible heuristic by replacing c_n in the cost function by k_t:

$$h_n(n_n) = \lambda^{n} * \left[\mu_2 * \Delta\left(k_t, \frac{\varphi_n}{\alpha} \right) + \mu_3 * \Delta\left(k_t, c_{n-1} \right) \right]$$

This heuristic only considers the cost associated with the next branch. It also does not consider the cost associated with the effective direction of motion. Therefore, this

heuristic is not optimal as it underestimates the minimum path to reach a goal node. However, this heuristic is admissible and computationally very efficient.

A better admissible heuristic that also considers the cost associated with the effective direction of motion is as following:

$$h_n(n_n) = \lambda^n * \left[\mu_1 * \Delta(k_p, k_t) + \mu_2 * \Delta\left(k_t, \frac{\theta_n}{a}\right) + \mu_3 * \Delta(k_t, c_{n-1}) \right]$$

with: $k_p = -\arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right)$ based on $c_n = k_t$

This heuristic is better but less computationally efficient that the previous one. It is also not optimal because it considers the minimum cost associated with only the next branch.

To get an optimal heuristic, one could simply *expand* (without building the masked polar histogram and determining the corresponding candidate directions) the current node until the goal depth by using the target direction as the candidate direction at each node. By adding up the corresponding costs, we could get the optimal heuristic value. However, this heuristic requires much more computational power.

The difference between these heuristics is a trade-off between quality and speed of the heuristic function. The first heuristic is currently implemented.

7.3.6 Reducing the Branching Factor

By reducing the branching factor *b* of the search tree, the search algorithm becomes faster and requires less memory. This reduction can be done by eliminating redundant nodes.

Each node has a number of successor nodes equal to the number of its candidate directions. Because of the limited projected distance d_p, the projected position and orientation of several successor nodes can be identical. All candidate directions to the right of the robot that exceed θ_r have the same projected position and orientation. Also all candidate directions to the left of the robot that exceed θ_l have the same projected position and orientation. To reduce the branching factor, all but the cheapest candidate direction for both sides can be eliminated.

The explanation for the validity of this approach is simple. Let's assume that a node has several candidate directions that exceed θ_l. As their projected positions and orientations are identical, their polar masked histograms would be identical as well. Therefore, the nodes of the remaining search trees for these candidate directions would be identical as well. However, the costs associated with these branches would be different. As long as condition 2 is satisfied, the costs of the search tree nodes starting at the candidate direction with the lowest cost are always cheaper than the corresponding nodes of the other candidate directions.

Due to this node elimination method, the branching factor *b* is rarely bigger than three.

Another important speed improvement is achieved by only expanding the search tree if there are more than one primary candidate direction. If there is only one primary candidate direction, there is no need to expand the search tree as the robot has no choice anyway. Expanding such a search tree would only make sense, if the robot not only considered the selected primary candidate direction but also the corresponding projected trajectory.

7.3.7 Performance of the VFH* Method

Simulated examples of the VFH* method based on the first heuristic and a goal depth of five are shown in Figure 44. This figure shows the search trees at different times during the traversal of an obstacle course.

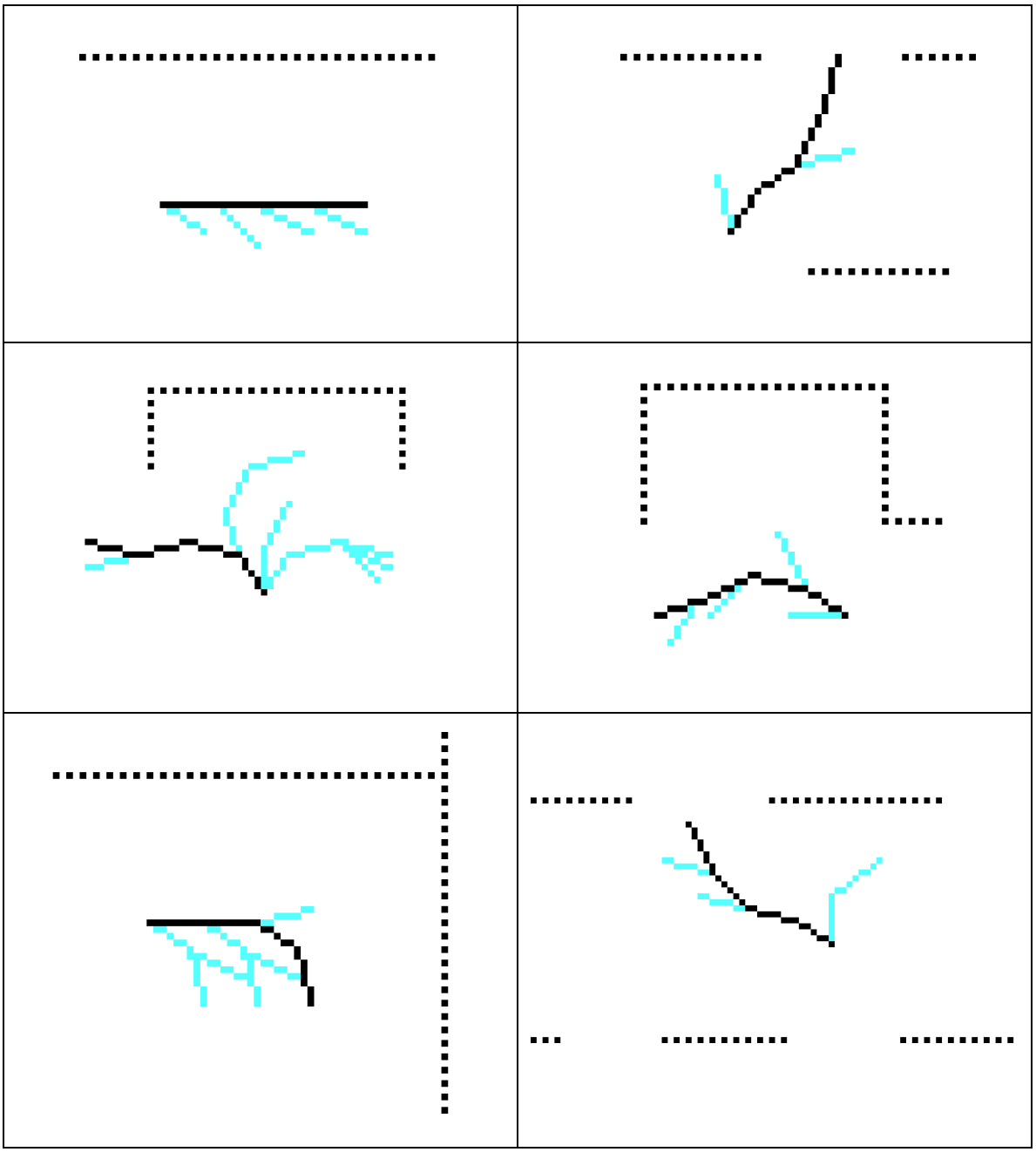


Figure 44: VFH* examples

The VFH* method is identical to the VFH+ method if the goal depth d_g is set to one. The higher d_g is selected, the better the VFH* performance becomes, but also the slower the algorithm becomes. By allocating more computational power to the obstacle avoidance algorithm, an even better performance than the VFH+ method can be obtained with the VFH* method

A comparison based on tests on a PC 486 running at 67 MHz is shown in Table 21. These tests were based on the first heuristic. The second column averages the computation time over an obstacle course without taking in account the times when there is only one primary candidate direction. The third column shows the maximum observed required computation time.

d_g	$T_{average}$	$T_{maximum}$
2	5 ms	11 ms
3	8 ms	22 ms
4	10 ms	39 ms
5	12 ms	82 ms
10	30 ms	242 ms

Table 21: VFH* goal depth comparison

The comparison table shows that the VFH* method is still fast as long as the goal depth is small. The maximum required time could be decreased by applying additional branching factor reduction techniques.

Tests with the simulated and the real robot have shown that even a goal depth of two is often enough to avoid most small dead-ends. As the GuideCane's on-board computer only runs at 33 MHz, this parameter value was selected in the current implementation.

With the look-ahead of the sonars, a goal depth between 3 and 5 would be optimal. However, the corresponding sampling rate could be too slow with the current computer.

The VFH* method is also very insensitive to its parameter values, and thus requires almost no time adjusting them. As long as conditions 1 to 3 are satisfied and the parameter values are selected reasonably, the VFH* method performs well.

The VFH* method could also be used as a planner, especially in static and known environments. This could easily be implemented by defining the target position as the goal instead of a total projected distance. However, if the distance to the target position is very large, the *iterative deepening A** search should be implemented to reduce the memory requirements.

7.4 Wall Following

As long as the wall is represented in the histogram grid, both the VFH+ and the VFH* methods perform the wall following task well. If some parts of the wall are not represented in the histogram grid, the GuideCane believes that there is an opening in the wall. As a result of this misrepresentation, the robot could decide to head towards this virtual opening.

Tests showed that the sonars have much more difficulty representing a wall in the histogram grid than a small obstacle. They especially have problems with walls that have a very smooth surface, as the ultrasonic wave emitted by the forward-facing sensors is reflected away by the specularly reflective walls. Only the signals from sensors that are nearly perpendicular to the smooth wall are reflected back. When the GuideCane follows a smooth wall, usually only the side-facing sonars receive an echo, and increase the corresponding certainty values in the histogram grid. The first problem of representing a smooth wall in the histogram grid is that only one sonar is able to detect it. This problem will be reduced with the new proposed GuideCane structure that has three additional sonars. With this version, it is expected that at least two sonars will always detect the wall.

The problem becomes worse, because the wall is only detected by the sonar that travels perpendicularly to it. There is a fundamental difference how an obstacle in front of the GuideCane and how an obstacle on its side is represented in the histogram grid. In the current implementation, the look-ahead of the sonars is 2.5 m, while the VFH radius r_{VFH} is 1.5 m. The desired walking speed is 1 m/s and the sampling frequency of the sonars is 10 Hz. Therefore, the GuideCane gets a new sonar reading for every 10 cm of travel. However, 10 cm is also the size of the histogram grid cells. Now, when the GuideCane approaches an obstacle in front of it, the corresponding cell is updated about 10 times before it enters the VFH radius. On the other hand, when the GuideCane follows a smooth wall, every cell corresponding to the wall is on average only updated once by a side facing sonar. Therefore, while the certainty value of a cell corresponding to an up-coming obstacle can be increased several times before influencing the VFH output, a cell corresponding to a smooth wall is increased only once.

To reduce this problem, a high value of +50 is given to the increment parameter I^+ for cells updated by the side-facing sonars. Setting I^+ for a side-facing sonar to such a high value is still safe, as a falsely updated cell on the robot's side would rarely have a big influence on the obstacle avoidance behavior. Unfortunately, this method reduces the problem only a little.

To really overcome this problem, a wall-following behavior was added to the GuideCane. The idea of this method is that the wall is often much better represented in the histogram grid at a distance d_{lag} behind the robot. So, one possibility would be to build the masked polar histogram at d_{lag} behind the robot instead of at its current position. This method would work well for the wall-following behavior, but it would fail with obstacles in the robot's path.

A better solution is to combine the current masked polar histogram with the masked polar histogram built at d_{lag} behind the robot. The combination is done by the *OR* function. With this *combined polar histogram* as the input for the obstacle avoidance algorithm, the GuideCane follows smooth walls very well and still reacts well to upcoming obstacles. The parameter d_{lag} is set to 50 cm. This position corresponds to a location between the user and the GuideCane center. This method is simple and safe, as the GuideCane can assume that the location of the user is different from the wall location.

In densely cluttered environments, it is possible that the combined polar histogram is blocked in every direction while the current masked polar histogram is not blocked. In that case, the robot could incorrectly believe that it is trapped. To overcome this potential problem, the GuideCane first analyzes the combined polar histogram. If all directions are blocked, it analyzes the current masked polar histogram instead. Only if this histogram is also blocked, will the robot stop.

In the current implementation, the user can switch between the regular obstacle avoidance behavior and the wall-following behavior by pressing the lower button on the input pointer. With the wall-following behavior, the GuideCane ignores small openings like doors in the wall. With the regular behavior, the GuideCane tries to go through doors.

8. The Development Environment

The quality of the development environment is essential for any project involving a real mobile robot. During the GuideCane project, several development environments have been built, each having its advantages and disadvantages. These development environments are explained in the following sections.

8.1 The Tether Environment

A simple but effective development environment utilizes a *Cybex Companion* tether [16]. This product consists of a signal buffer and a long cable which allows one to connect a keyboard and a VGA (or better) monitor up to 250 feet away from the GuideCane PC. The tether is fixed to the GuideCane. The tether cable runs from the on-board PC to a ceiling-mounted rotating beam and then to a stationary keyboard and monitor on the developer's desk. The robot can thus travel freely throughout a relatively large work area (about 6m × 6m). A particular advantage of the system is that it behaves exactly like a single desktop unit. Moreover, a power cable can be put in parallel to the tether cable, so that the GuideCane can be connected to a power outlet. Hence, unlimited testing independent of the battery capacity is possible. The power cable is connected to a small AC/DC converter delivering 5V at up to 5A. Disadvantages of this system are the limited area of operation and the fact that the tether has to be unwound periodically.

This environment is very useful during the early stages of development when a large area for testing is not required. It is also useful to allow development when there is no set of charged batteries available. For very long editing sessions, this method is the preferred one as the development time is not limited by the batteries.

8.2 The Palmtop Environment

The *PTV30 Palmtop* PC together with the *Anywhere* DOS program is another effective combination for the development of a mobile robot. The PTV30 Palmtop is a small lightweight laptop PC powered by two rechargeable AA batteries. The keyboard is small enough to allow touch-typing at about 75% of normal typing speed. The monitor is a greyscale LCD with a 640×200 pixel resolution.

The palmtop is connected to the main PC through the serial line. The *Anywhere* DOS program is run in the main PC background, while the *Aterm* DOS program is run on the palmtop. The *Anywhere* program sends all text output from the main PC to the palmtop through the serial line. The *Aterm* program then displays the text on the palmtop screen. In addition, whatever is entered on the palmtop keyboard is sent to the main PC where it is processed by the *Anywhere* program so that the data looks like a standard keyboard input to the main PC.

This combination allows full text development in the sense that it allows software editing and text output. Its main advantage over the tether method is that it does not require a cable from the GuideCane to a fixed development station. Therefore, the testing of the GuideCane is not restricted to a small development area anymore.

The main disadvantage of this system is that it does not allow graphical output like maps or polar histograms. Another disadvantage is that the *Anywhere* program slightly slows down the sampling rate of the main program.

8.3 The LCD Environment

For optimal development, graphic runtime output in color is necessary. However, CRT monitors are too heavy to be carried around. They also require too much power. A good alternative is the use of a color LCD which is much lighter and consumes much less power than a CRT.

For editing, either the palmtop connected to the serial port of the PC or a small standard keyboard can be used. The palmtop slightly slows down the sampling rate of the PC and does not allow maximum speed typing. The small keyboard does not affect the sampling rate and allows maximum speed typing, but it is a bit heavier and bigger than the palmtop.

8.4 The GuideCane Simulator

To speed up the development process, a simulation of the GuideCane was written and can be run under DOS. If used appropriately, the simulator can be a tremendous tool. As there is neither noise nor any other random input, the simulated robot behaves the same way on every run. This repeatability is extremely useful as it helps to understand specific situations much better than with tests on a real robot where every run is different. The simulator also allows the developer to stop a test run to verify variables. The simulation can then be continued at normal speed or step by step.

However, simulations can be misleading, especially if they are used inappropriately. The fact that a simulated robot performs well does not at all mean that the real robot will do as well. Hence, a simulation should never replace experiments with a real robot.

On the other hand, a simulated robot that performs badly will likely not do well in reality either. Hence, it is a good approach to get the robot's performance to a satisfying degree in the simulation before implementing and improving it on the real robot. Another advantage of the simulations is that new software routines can be debugged more easily.

The simulator can be loaded with different histogram maps. At each iteration, the simulation updates the position and orientation of the robot dependent on the new direction j_n specified by the obstacle avoidance algorithm.

The simulation is only an approximation of the real GuideCane. The first approximation is that the robot is already given a map with certainty values. Therefore, the poor directionality and other problems of the sonars are neglected. In fact, this

simulator only tests the behavior of the obstacle avoidance. It does not test the sensor routines at all, as this is better tested on the real robot.

The second approximation is the robot trajectory. There is no need for an exact simulation of the robot trajectory. The input parameters for the robot trajectory are the robot speed v_t and the speed of the change of orientation v_r. The current parameters are:

$$\begin{aligned} v_t &= 800 \text{ mm / s} \\ v_r &= 90 \text{ ° / s} \end{aligned}$$

The change of the robot orientation based on the new steering command φ_n is then approximated by:

$$\Delta\varphi_i = \varphi_n - \varphi_i$$

But this change of orientation is limited by the sampling interval T:

$$\Delta\varphi_{\max} = v_r * T$$

The orientation is updated by:

$$\begin{aligned} \varphi_{i+1} &= \varphi_i + \Delta\varphi_i && \text{if} \quad |\Delta\varphi_i| \le \Delta\varphi_{\max} \\ \varphi_{i+1} &= \varphi_i + \varphi_{\max} && \text{if} \quad \Delta\varphi_i > \Delta\varphi_{\max} \\ \varphi_{i+1} &= \varphi_i - \varphi_{\max} && \text{if} \quad \Delta\varphi_i < -\Delta\varphi_{\max} \end{aligned}$$

The position is then updated by the following approximation:

$$\begin{aligned} x_{i+1} &= x_i + v_t * T * \cos\left(\frac{\varphi_{i+1} + \varphi_i}{2}\right) \\ y_{i+1} &= y_i - v_t * T * \sin\left(\frac{\varphi_{i+1} + \varphi_i}{2}\right) \end{aligned}$$

9. Future Improvements

9.1 The Compass

The vehicle heading is the most significant of the three navigation parameters in terms of its influence on accumulated dead-reckoning errors [11]. A magnetic compass would be useful to keep the odometry orientation error bounded. The reliability of the compass data should be good enough for improvements in the outdoor navigation. Indoors however, the earth's magnetic field is often distorted near power lines or steel structures.

There are many small fluxgate compass modules commercially available. *KVH Industries* offers a complete line of fluxgate compasses and related accessories [24]. The *C100 Compass Engine* is a versatile low-cost developer's kit that includes a microprocessor-controlled stand-alone fluxgate sensor subsystem based on a two-axis toroidal ring-core sensor. Two different options are offered with this compass, limiting the tilt range to either ±16° or ±45°. This compass can simply be connected to the PC through the serial line.

The compass should be placed as far away from the GuideCane's electronics as possible. The best place would probably be on the cane near the user's hand.

9.2 Computer Vision

The main problem of the GuideCane is currently the outdoor navigation, especially sidewalk following. The problem is that the sonars are not able to detect the borders of a sidewalk and hence, the GuideCane is not able to follow sidewalks.

An appropriate method to solve this problem is computer vision. Vision systems have been developed that were capable of driving a car on a highway [35]. It should not be that hard to implement a simplified version of such a system on the GuideCane. The sidewalk following problem is easier than the road following problem because the speed of travel is much smaller. It also requires less safety features. If the sidewalk following behavior failed, the GuideCane may hit the side of the sidewalk which would instantly be perceived by the user. The user could then bring it back on the sidewalk and try again with hopefully more success. And due to the smaller speed, smaller masses and a *friendlier* environment, crashes are not as disastrous as in the case of a road following system.

Computer Vision could also be used for other purposes, like landmark recognition. Together with advanced speech output, it would even have the potential to read text on signs to the user.

9.3 GPS - Global Navigation

The GuideCane can also be equipped with a *Global Positioning System* (GPS). Several GPS systems have become commercially available. Some of these systems are even in the PC/104 standard so that they can simply be stacked on top of the current PC.

Outdoors, standard GPS can currently provide global positioning information to within 20 meters accuracy. This makes it possible for the blind individual to prescribe a desired trajectory to a target location (e.g. the supermarket or the post office) to the system and to have the GuideCane automatically guide the user to that location. Alternatively, the system could learn a desired path by recording path segments during an initial *lead-through* run with a sighted person.

Indoors, where GPS is not effective, the same path programming or lead-through techniques can be used to have the GuideCane automatically guide the user to a desired location, using dead-reckoning based on encoder and compass readings. This latter method is not suitable for long distances because of the unbounded accumulation of odometry errors, but it is suitable for shorter indoor paths.

An ongoing project that is based on GPS technology is the *MoBIC* project which is supported by the Commission of the European Union [18]. The project goal is a route planning system that allows a blind person to access electronic maps of the locality. It also allows the user to access information from many sources such as bus and train timetables.

The MoBIC system is complementary to the GuideCane. MoBIC can be used for the global navigation while the GuideCane is used for the local navigation.

9.4 Speech Input/Output

A large variety of functions can be implemented with the help of speech output and/or input. Speech output is much easier to implement than speech input. Speech output could be implemented by adding a speaker and a *speech synthesis processor* from *Texas Instruments* or a *voice storage controller* from *TriTech* to the current interface [34][36]. An even simpler, but more space consuming possibility would be to purchase a PC/104 speech and sound module.

Speech output would allow the GuideCane to give more information to the user. For example, the GuideCane could ask the user to slow down. Or even more subtle, as the GuideCane knows the length and the orientation of the cane, it knows where the user is relative to its map and can so warn the user if he gets too close to an obstacle because of the trailer effect. The GuideCane could even tell the user on which side he is getting too close. Or in the case of a dead-end, the GuideCane could ask the user to back up until it finds a new suitable direction. In short, speech output, if cautiously used, could be a very valuable feature.

Speech input on the other hand may sound exciting, but the ratio of usefulness over implementation difficulty is small compared to speech output. Currently, the main user input is the desired direction of travel, which is entered through the pointer. It does not make much sense to replace this input by speech input. It is much more convenient for the user to communicate the desired direction through a standard input device than through speech input. Even for other desired behaviors, like going through a narrow door, it is more convenient to communicate by pressing a button than by speech input. However, there are cases where speech input could be valuable.

9.5 Additional Sonars

With the current electronic system, six additional sonars can be connected to the main interface. Additional sonars could be useful to detect overhanging obstacles, a hazard that is feared most by white cane users. They could also be used to detect and find up-steps.

The more sonars the GuideCane is equipped with, the safer this robot becomes. However, additional sonars increase the cost and the power consumption of the GuideCane.

10. Conclusion

The GuideCane concept is revolutionary. This new device overcomes the fundamental shortcomings of conventional electronic travel aids for the blind. It is easy to use and requires little training time. The GuideCane does not simply tell the user the obstacle information but directly delivers a guidance signal that is intuitive to follow.

The main physical result of this Master thesis is the realization of a fully autonomous mobile robot. This embedded system is based on a total integration of the mechanical structure, the electronic hardware, and the software.

The current mechanical structure performs well. However, a better structure based on three wheels was proposed. This structure will be more stable, more comfortable to use, more resistive to mechanical shocks and it will require a less power consuming servo. It will also be equipped with three additional sonars which should improve the performance of the obstacle avoidance.

The compact electronic hardware consists of a PC/104 computer and a multiprocessor interface. This system performs well and is very reliable and efficient. Moreover, it was designed in such a way that future extensions can easily be integrated into the current system.

The software takes care of the interface, the odometry, the map building and scrolling, and most importantly the obstacle avoidance algorithm. The parts that still need the most improvements are the user-robot interface and special behaviors, like finding a door. Also, several development environments were developed which should prove to be very useful tools for this project in the future.

The main research result consists of the four improvements of the original VFH method resulting in the VFH+ method. First of all, by using a hysteresis instead of a fixed threshold, the robot trajectory becomes smoother. Secondly, the VFH+ takes explicitly in account the robot size, so that this method can easily be applied to robots of different sizes without time-consuming parameter adjusting. In addition, this improvement eliminates the need of VFF so that the obstacle avoidance algorithm can be based solely on VFH+. Thirdly, the VFH+ method takes in account the trajectory of the mobile robot by masking free sectors that are blocked by obstacles in other sectors. As a result, the robot can not be guided into an obstacle anymore, as it was possible with the original VFH method. Finally, by applying a cost based direction selection, the performance of the obstacle avoidance algorithm becomes better and more reliable. This also gives the possibility of switching between behaviors by simply changing the cost function or its parameters.

The second important research result is the identification of the extremely local nature of both the VFH and the VFH+ algorithm. This problem is overcome by introducing local planning into the obstacle avoidance algorithm. This planning is efficiently done with the A* search algorithm, an appropriate cost function, and a heuristic function. VFH*, the resulting obstacle avoidance algorithm, eliminates the shortcomings of the original VFH method, yet it retains all advantages of its predecessor.

BIBLIOGRAPHY

[1] Bell, D., Borenstein, J., Levine, S.P., Koren, Y. and Jaros, L., “An Assistive Navigation System for Wheelchairs Based upon Mobile Robot Obstacle Avoidance”, IEEE International Conference on Robotics and Automation, San Diego, CA, May 1994, pp. 2018-2022.

[2] Benjamin, J. M., Ali, N. A., and Schepis, A. F., “A Laser Cane for the Blind”, Proceedings of the San Diego Biomedical Symposium, 1973, Vol. 12, pp. 53-57.

[3] Bissitt, D. and Heyes, A. D., “An Application of Biofeedback in the Rehabilitation of the Blind”, Applied Ergonomics, 1980, Vol. 11, No. 1, pp. 31-33.

[4] Blasch, B. B., Long, R. G., and Griffin-Shirley, N., “National Evaluation of Electronic Travel Aids for Blind and Visually Impaired Individuals: Implications for Design”, RESNA 12th Annual Conference, New Orleans, Louisiana, 1989, pp. 133-134.

[5] Borenstein, J. and Feng, L., “Measurement and Correction of Systematic Odometry Errors in Mobile Robots”, IEEE Transactions on Robotics and Automation, December 1996, pp. 869-880.

[6] Borenstein, J. and Koren, Y., “Error Eliminating Rapid Ultrasonic Firing for Mobile Robot Obstacle Avoidance”, IEEE Transactions on Robotics and Automation, February 1995, Vol. 11, No. 1, pp. 132-138.

[7] Borenstein, J. and Koren, Y., “Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance”, IEEE Transactions on Robotics and Automation, August 1991, pp. 535-539.

[8] Borenstein, J. and Koren, Y., “Real-time Obstacle Avoidance for Fast Mobile Robots”, Sept./Oct/ 1989, pp. 1179-1187.

[9] Borenstein, J. and Koren, Y., “The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots”, IEEE Journal of Robotics and Automation, June 1991, Vol. 7, No. 3, pp. 278-288.

[10] Borenstein, J. and Ulrich, I., “The GuideCane - A Computerized Travel Aid for the Active Guidance of Blind Pedestrians”, IEEE Conference on Robotics and Automation, Albuquerque, April 1997, pp xx - xx.

[11] Borenstein, J., Everett, H.R. and Feng, L., “Navigating Mobile Robots”, A K Peters, Ltd., 1996.

[12] Brabyn, J. A., “New Developments in Mobility and Orientation Aids for the Blind”, IEEE Transactions on Biomedical Engineering, 1982, Vol. BME-29, No. 4, pp. 285-289.

[13] Crowley, J. and Reignier, P., “Asynchronous Control of Rotation and Translation for a Robot Vehicle”, Robotics and Autonomous Systems, 1992, Vol. 10, pp. 243-251.

[14] Dan, P., "Recent Advances in Rechargeable Batteries", Electronic Design, February 3, 1997, pp. 112-116.

[15] Elfes, A., “Using occupancy grids for mobile robot perception and navigation”, Computer Magazine, June 1989, pp. 46-57.

[16] Feng, L., Borenstein, J. and Wehe, D., “A Completely Wireless Development System for Mobile Robots”, ISRAM Conference, Montpellier, France, May 1996, pp. 277-282.

[17] Frye, B., "Rechargeable Power Options For Portable Computers", Electronic Design, December 16, 1996, pp. 105-112.

[18] Gill, J., “An Orientation and Navigation System for Blind Pedestrians”, http://www.cs.uni-magdeburg.de/~mobic, 1996.

[19] Guthrie, C., “Power-On Sequencing for Liquid Crystal Displays; Why, When, and How”, Sharp LCD Application Note, pp. 2.1-2.9.

[20] Kay, L., “A Sonar Aid to Enhance Spatial Perception of the Blind: Engineering Design and evaluation”, Radio and Electronic Engineer, 1974, Vol. 44, No. 11, pp. 605-627.

[21] Klarer, P.R., “Simple 2-D Navigation for Wheeled Vehicles”, Sandia Report SAND88-0540, Sandia National Laboratories, Albuquerque, April 1988.

[22] Koren, Y., and Borenstein, J., “Analysis of Control Methods for Mobile Robot Obstacle Avoidance”, IEEE International Workshop on Intelligent Motion Control, Istanbul, Turkey, August 1990, pp. 457-462.

[23] Koren, Y., and Borenstein, J., “Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation”, IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991, pp. 1398-1404.

[24] KVH Industries, C100 Compass Engine Product Literature, 110 Enterprise Center, Middletown, RI 02840, 401-847-3327.

[25] Lebedev, V. V. and Sheiman, V. L., “Assessment of the Possibilities of Building an Echo Locator for the Blind”, Telecommunications and Radio Engineering, 1980, Vol. 34-35, No. 3, pp. 97-100.

[26] Linear Technology, "LT1529 - 3 A Low Dropout Voltage Regulators with Micropower Quiescent Current and Shutdown", Product Literature, Linear Technology Corporation, 1630 McCarthy Blvd., Milpitas, CA 95035-7487, (408) 432 1900, 1996.

[27] Moravec, H.P., “Sensor fusion in certainty grids for mobile robots”, AI Magazine, summer 1988, pp. 61-74.

[28] Motorola, M68HC11 Reference Manual, 1991.

[29] PC/104 Consortium, “PC/104 Specification Version 2.3”, June 1996.

[30] Polaroid, "Ultrasonic Ranging System", Product Literature, Polaroid Corporation, 784 Memorial Drive, Cambridge, MA 02139, (617) 386 3964, 1991.

[31] Shoval, S., Borenstein, J., and Koren, Y., “Mobile Robot Obstacle Avoidance in a Computerized Travel Aid for the Blind”, Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA, May 1994, pp. 2023-2029.

[32] Simmons, R., “The Curvature-Velocity Method for Local Obstacle Avoidance”, International Conference on Robotics and Automation, Minneapolis, MN, April 1996.

[33] Stuart, R. and Norvig, P., “Artificial Intelligence - A Modern Approach”, Prentice-Hall, New Jersey, 1995.

[34] Texas Instruments, "Speech Synthesis Processors", Product Literature, Texas Instruments Incorporated, (810) 305 5700, 1996.

[35] Thorpe, C.E., “Vision and Navigation - The Carnegie Mellon Navlab”, Kluwer Academic Publishers, Norwell, MA, 1990.

[36] TriTech, "Voice Storage Controller TR83100CF", Product Literature, TriTech Microelectronics International Inc., 1400 McCandless Drive, Milpitas, CA 95035-8800, (408) 894 1900, 1996.

[37] Wolfensberger, M. and Wright, D., “Synthesis of reflexive algorithms with intelligence for effective robot path planning in unknown environments”, Proceeding of the 8th International Conference on Mobile Robots, 1993, pp. 770-780.

[38] Wormald International Sensory Aids, 6140 Horseshoe Bar Rd., Loomis, CA 95650.