A Navigation Aid for the Visually Impaired

For the award of

Masters of Engineering

by

Laurens P. Kallewaard B.Eng

Dublin City University

Supervised by

Professor Charles McCorkell

School of Engineering and Design

September 1997

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters of Engineering is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work

Signed L Kallenbach ID No 93701420

Date

Acknowledgements

Firstly thanks to Professor McCorkell for his patience and support Many people contributed to the project through their criticism and suggestions, especially the members of the EASI and BLIND-L discussion groups Thanks also to Prof Goodrich for his help and encouragement

Thanks also to my friends for support and sessions and to Dee without whom this thesis would never have been completed

But mostly, thanks to my parents, who gave me life, love and opportunity

Abstract

The explosion of new technologies in recent times has created many as yet unexplored possibilities in the area of assistive technology Assistive technology may be thought of as any technology that alleviates the affects of an impairment, disability or handicap One of the greatest challenges facing designers of assistive technologies is to accurately define the user requirements The background research chapter of this report attempts to define the user requirements for a navigation aid that can be used any person, regardless of visual ability, both through literature surveys and through dialogue with potential end users The conclusions drawn from this research are used as a basis for an investigation into possible technological solutions, culminating in a technical summary

Subsequently, the design and implementation of a software package capable of providing environmental information and navigational suggestions is discussed in the final chapters The package is designed to be flexible enough to act as a basis for further research and eventual implementation of a fully functioning navigation aid

1. Introduction

Over the past few decades an increasing amount of research has been carried out in an effort to find new technologies that compensate for physical disabilities Society is structured in a such a way that even minor physical disabilities can prevent a person from integrating fully into society Technologies that allow people to overcome such handicaps are therefore highly desirable

This project was prompted by the increasing number of visually disabled students attending the university While the overall numbers are still very small (four students have severe impairments and four are legally blind), it is the policy of the university to accommodate all students regardless of physical ability

The project is primarily concerned with the mobility of visually disabled people around the university campus or a similar complex of buildings Students with severe visual impairments can require several months of familiarisation time This problem is further complicated by the rapid expansion of the university and the many building programs being undertaken there A navigation aid capable of giving information about a person's position and environment within a complex of buildings would therefore be a useful teaching and consulting device Furthermore, it should be possible for the navigation aid to guide a person from one position to another

There are many problems associated with the development of such a device One of the most difficult and under-rated challenges is the determination of user requirements Design considerations include the type of information to be relayed to the user, the format of the information, as well as the methods by which the information is requested, selected and retrieved The second chapter of the dissertation is solely devoted to the task of ascertaining answers to these questions

The background research covered in the second chapter also includes a section on existing navigation aids These aids include simple devices such as canes and tactile maps, simple electronic travel aids such as the Sonic Torch and Polaran as well as more complicated travel aids such as the PathFinder and the Sonic Guide

The reasons for the successes and failures of these devices are examined and conclusions are drawn

2. Background Research

2.1 Basic Definitions

The aim of this section is to examine how both sighted and blind people perceive environmental information and how they use this information for basic mobility tasks This is essential if we are to make an attempt at identifying the nature and content of any information that may improve a blind person's mobility skills

Firstly, it is important to know exactly what is meant when referring to words such as impairment, disability and handicap An impairment occurs as a result of a disease or defect and interferes with the normal operation of an organ A disability is present when an impairment interferes with the carrying out of activities considered normal for the average human being The term handicap is used when a person's disability prevents them from carrying out a role considered normal in everyday life The difference between a disability and a handicap refers to how a person 'fits in', i e how a disability affects one's ability to act as part of a society A person with a visual disability is therefore a person whose (corrected) vision is not sufficient to allow that person to carry out all everyday activities Of particular concern here is a person's mobility and his or her ability to orientate and navigate without the full benefit of sight The people effected to the greatest extent are the people who are totally blind, those that cannot perceive anything more than changes in light levels

Mobility can be thought of as the ability to travel independently and safely Orientation on the other hand deals with the directions of objects in relation to a person and in relation to each other The ability to orientate thus allows a person to determine the direction of travel, the direction of potential obstacles and the direction of any perceived objects within the environments Navigation deals with a person's position within their immediate environment and their ability to travel or direct themselves to a desired destination Navigation is therefore not possible without basic mobility and orientation skills

2.2. Perception and understanding of the spatial information necessary for navigation [1-11]

Most people take basic mobility skills for granted and little thought is given as to how we move within our environment. To safely (and confidently) move within a particular environment, certain aspects about the environment must be perceived [11]. It is essential that obstacles blocking our pathways are detected in time so that they can be avoided or negotiated. This is true for both static objects such as buildings and lampposts as well as moving objects such as cars. Changes in the surface one is travelling on must also be detected in time to prevent accidents, such as kerbs and steps. In order to plan the next step in a reasonable amount of time, a traveller must confidently be able to detect any potential hazards. Another aspect of independent travel is the ability to find (and traverse) a desired path [2,3,4].

Light perception, or the sense of sight, is ideally suited to supply the information required for mobility, orientation and navigation. Its complete or partial loss can therefore greatly affect everyday living. The initial parts of this section are directed to a discussion of some of the basic principles of perception and spatial understanding. In the latter parts of this section, the application of a number of senses to navigation are discussed.

2.2.1. Redundancy

Under well-lighted conditions, there are innumerous light rays reflected to a person's eyes providing a vast volume of information [3,4]. The amount of information available can greatly exceed what is required for a person to have basic navigation skills. This is referred to as perceptual redundancy, that is, if more information is available than is necessary for a particular application, redundancy occurs. This, in particular, is where visual perception is on its own. Although sound can be used in many different ways to extract information about the environment, there is a lack of auditory perceptual redundancy, in other words, in comparison to the vast quantities of light available, most environments are largely silent. This effectively means that the amount of information available through the auditory system in comparison to the visual system is meagre and blind traveller must be constantly alert for sounds that may aid them to travel. Some blind people have reported that it is less difficult to

travel in the rain, since the sound the rain produces on different surfaces can reveal much about their nature Despite this, the auditory system cannot compete with the visual system for perception of the environment The visual system is able to discard information it does not consider vital for mobility, but when a person hears many different sounds at once, it can be very difficult to extract useful information about the nature of these sounds The auditory system can thus get overloaded with information more easily than the visual system

2.2.2. Invariance

In order for a perceptual system to provide useful information, it must be subjected to stimuli that change with time[3,4] It is this requirement that allows useful information to be extracted from the senses This is true for all the perceptual systems, including the visual system If an image remains constant on the retina of the eye, it disappears This is prevented by small movements of the eye, thus ensuring that the image is constantly changing Invariance is important to understanding how a travel aid such as a cane can provide useful information (see Sect 2 3)

2.2.3. Illusions

Illusions can cause a person to incorrectly perceive certain aspects about his or her environment [3,4] Most of the senses are prone to illusions Two common illusions are the Muller-Lyer illusion and the horizontal-vertical illusion The Muller-Lyer illusion occurs when comparing two line segments of equal length, one terminated with arrow heads the other with arrow tails The line with arrow tails appears shorter than the line with the arrow heads The horizontal-vertical illusion occurs when vertical and horizontal lines are compared for length, the vertical lines usually being underestimated Both visual and haptic (see section 2 2 7) perceptions are prone to these illusions The haptic illusion occurs with raised lines and therefore can have negative consequences when reading Braille maps

2.2.4. Spatial understanding

Another aspect of navigating is the storing in the brain of environmental information necessary for travel [3,4,9] Mentally stored spatial information of an area is known as a cognitive map If a person who is familiar with route A to B becomes familiar

with route A to C, it should be possible for them to go directly from B to C and take detours if the way is blocked To do this, their cognitive maps of the routes A,B and A,C must demonstrate a good understanding of the positions of objects and destinations in relation to each other (in terms of distances and angles) Their cognitive map of the area must change from two separate routes to an integrated map that includes knowledge of locations in relation to each other Studies have been made to investigate the comparison between the way sighted and blind people store this type of spatial information When cognitive maps of a well known building were derived from blind and sighted subjects in one such study [9], all showed Euclidean organisation with the sighted subjects being slightly more accurate In a follow-up study, the same subjects were asked to make specific Euclidean judgements The sighted subjects proved to be more precise and flexible The study concluded that many blind people have more difficulty in organising their knowledge of an area in Euclidean form

2.2.5. The visual system and navigation

How is information essential for mobility extracted from the sense of sight, and can this information be perceived by any of the other senses? Answering this question will identify if it is possible to aid the blind traveller by providing information to him or her normally only obtainable by visual means[1]

The sense of sight can be used to detect and recognise obstacles and pathways, estimate distances, judge motion, help us orientate and can enable a person to detect potentially dangerous situations Sight is therefore ideally suited for mobility and navigation A sighted person can at a glance judge safety factors involved in locomotion How sight is used to judge the direction one is travelling in, and how it can tell us whether or not an obstacle is on a collision course with us, has been subject to research in the past [3] It is based on what is known as an optical expansion pattern This refers to the way in which objects seem to expand when we get closer to them If the expansion pattern produced is symmetrical or similarly if the centre of the expansion pattern is in the middle of our visual field, we are heading directly for

[1]Section 5 deals with how this information can be presented to a blind traveller

that point When walking, the point one wishes to walk towards should remain at the centre of the optical expansion pattern Similarly if we wish to avoid an obstacle (such as an approaching car), we move in a direction that causes the expansion pattern produced by the car to be asymmetrical or off-centre

Different surfaces can be detected either by the way in which one surface hides another surface as one moves past (known as occlusion and disocclusion) or by stereoscopic vision This allows people to detect changes in the surfaces they are walking on and thus prevent tripping or falling Different surface textures scatter light in different ways and/or absorb different wavelengths, enabling sighted people to recognise different objects and thus allowing them to navigate

Sight is also well suited for orientation Frames of reference can be used to tell a sighted person what direction he or she is facing as well as information about his or her position Static vertical and horizontal surfaces are best used for frames of reference and are available in most environments

2.2.6. The auditory system and navigation

The auditory system is another major source of information that can be used to locomote within an environment[3,6] Sound can be used in many different ways It is possible to establish the position of a sound emitting object to within a degree or two by means of binaural localisation or movement of the head[2] Moving objects emitting sound can thus be tracked The Doppler effect can be used to tell if a sound source is moving away from or towards the listener This is can be very useful when dealing with fast moving traffic

Objects that do not emit sound can be detected using echo-location This requires a lot of experience but can be very useful The difference in time between when a sound is emitted and when its echo is heard reveals information about the distance of the object from which the sound reflected Reflected sound can be used to avoid large obstacles The presence or absence of echoes can be used to establish the position of objects around the listener and can therefore be used to identify position and orientation Another advantage of echo-location is that sound can easily be

---

[2] The accuracy obtainable by sound localisation is a function of frequency, the mid range of frequencies being more difficult to localise accurately

created for the purpose The tapping of a long cane, snapping of the fingers or simply footfalls can all be used to generate sound for echolocation

Sound-shadowing is a technique used to detect the presence or absence of relatively large objects It is based on the blocking of sound by these objects, such as the blocking of traffic noise by bus shelters or other large objects This can enable a person to avoid obstacles as well as help to establish position, for instance when sound previously blocked by a building suddenly becomes audible, one can safely assume to be at a corner

Sound waves are structured by the environment in a similar manner to light waves, thus allowing objects to be uniquely identified by the sounds reflected off them or by the sound they produce This allows a person to identify objects and hence establish their position For example, the sound created by an appliance can help to establish one's position in a particular room or corridor Sound may also aid orientation It is possible to know in which direction one is walking down a busy street simply by keeping the sound of the traffic on one side

2.2.7. Haptic perception and navigation

Haptic[3] and tactile information can also be used for the purpose of mobility[5] When moving, one is always m contact with the ground, allowing (to some extent) the identification of surface features such as roughness (or smoothness), regularity and slope It is possible to recognise, locate and detect discontinuities in objects using the sense of touch, however, since it only applies to objects in contact with us, the unaided sense of touch is of little use when navigating The aided sense of touch on the other hand, can become the single most valuable sense for blind people (see section 2 3)

2.2.8. Other senses used for navigation

Other senses can also provide relevant information for mobility [2,3,6] The inner-ear, or the vestibular system is essential for maintaining orientation with respect to

[3] By definition, haptic perception occurs when a person actively touches an object, while tactile perception occurs when an object touches the person These definitions are not normally strictly adhered to

gravity Besides enabling a person to keep their balance, it can tell a person the slope of the surface they are travelling on This is useful not only for safe travel but also for identifying one's whereabouts The vestibular system also provides information about linear and angular acceleration, based upon which a traveller may keep track of his position

Heat radiation detection can also be useful This can be used mainly to detect the position of the sun and can thus provide information relevant for orientation purposes The sense of smell can be quite keen and can allow us to identify objects and areas associated with particular odours, one of the best examples is a bakery shop

It is possible to keep an approximate track of one's position by using motor outflow (efference) information To put it simply, if we intentionally turn to the right, we expect that we have indeed turned to the right even without sensory feedback Thus if a person takes ten steps forward from a know position, even if they are not able to visually (or otherwise) confirm their whereabouts, they will still have a good idea of where they are

2.2.9. Summary

Having looked at the various senses and how they can be used to provide to the brain the information necessary for navigation, it would seem that all that should be necessary is to isolate elements lacking in the non-visual channels and provide them to a person using the other perceptual systems We have seen that a considerable amount of perceptual redundancy exists for the visual system, which explains why many people with low vision have no problem locomotmg The aim then is to provide environmental information normally obtained through vision at or above the point at which redundancy occurs Insufficient redundancy leads to a greater amount of errors however it is very easy to swamp the non-visual senses with an excessive amount of information Studies were carried out by Brabyn and Strelow[10] to determine the best method by which spatial information could be presented to test subjects with varying degrees of sight, in an effort to optimise sensory information Results showed that this optimisation was impossible, as different levels of environmental complexity greatly affected the problems involved in optimising the sensory information Studies such as this demonstrate the impracticality of trying to

establish specific parameters such as the level and format of information required for navigation since different environments create different demands of perceptual systems Further studies by Barth concluded that a person needs to perceive at least 3m ahead (known as preview) for relaxed travel but that sensory aids should not exceed this distance to prevent information overload All this points to the need to keep the informational flow relatively simple and easy to understand if the information is to be interpreted correctly and within an acceptable period of time

2.3. Evaluation of existing Mobility, Orientation and Navigation Aids

In the past few decades, several products have been developed whose purpose is to aid people in dealing with disabilities or handicaps Electronic Travel Aids (ETAs) are generally used to heighten peoples awareness of their environments, such as assisting them to perceive obstacles in sufficient time to avoid them The number and range of ETAs currently in use is very small The reasons for the general failure of ETAs to revolutionise blind travel, and the reasons for the fact that the ordinary long cane is such a successful travel aid, will be discussed in this section

2.3.1. Existing Aids

2.3.1.1 The Long Cane

The long cane is one of the most important of all mobility aids available to the blind traveller It is a primary mobility aid, which means that its use is sufficient to make a person mobile This simple device can, with appropriate training, make a blind person completely mobile The sense of touch can only give a person information about objects in contact with the body, however, with a long cane, this is extended so as to enable a user detect obstacles several feet away A person's perception of his immediate environment is thus increased sufficiently to enable him or her to negotiate obstacles successfully Since obstacle detection is one of the most important aspects of mobility, this skill can give a person sufficient confidence to travel freely The cane can be used by moving it from left to right, ideally with wrist action only The cane should thus describe a constant arc and the angles created by the cane and the ground remain constant if no obstacles are detected In this method the invariance

necessary for perception is produced, known as kinaesthetic invariance. However, this is not the only benefit that the long cane provides. The constant tapping sound created by the cane can be used for echo-location purposes (see Subsection 2.2.6 above).

The cane is less than adequate in the detection of obstacles that occur at waist height or higher. For instance, many stairways in public buildings are open and can be approached from the back and side, creating a head height obstacle that can result in serious injury if it is not detected in time. The cane may also fail to detect sudden drops in the surface such as those caused by street works. The objectives of many of the obstacle detectors developed recently is to overcome one or more of these problems whilst at the same time increasing a person's awareness of the immediate environment. In effect, this means that modern ETAs are designed to supplement primary mobility aids, not replace them, in contrast to earlier aids.

2.3.1.2 The Sonic Torch

The Sonic Torch is the first of the ETAs, developed by Professor Leslie Kay in the 1960's [3,7] The device is an ultrasonic, hand held obstacle detector with an audio output. When the device is pointed at an object, an audible tone is emitted. The closer an object is to the Sonic Torch, the lower the frequency of the output. This enables a user detect obstacles in time to avoid them. Very few people were able to make use of the device however, which was mainly attributed to the complexity and quantity of information presented to the traveller [3]. As discussed in Subsection 2.2.1, the auditory system is not able to process large quantities of information to the same extent as the visual system, therefore the information provided by the Sonic Torch often led to confusion. The Sonic Torch, as all ultrasonic devices, suffers from certain drawbacks which no matter how complicated the device and how clever the output, cannot be overcome. One is the wavelength of ultrasound. Many surfaces consist of grain sizes smaller than ultrasound wavelengths which means that ultrasonic devices cannot be trusted to identify all surface types and small discontinuities in surfaces. Another, perhaps more serious problem, is that some surfaces reflect ultrasound instead of scattering it. This may lead to what is termed as a 'false negative', in that an ultrasonic device may signal no objects ahead when in fact there may be a potentially dangerous obstacle.

2.3.1.3 The Mowat Sensor

One of the more commercially successful devices, the Mowat sensor is a hand held obstacle detector that uses pulsed ultrasound to detect obstacles [3,7] It can be set to detect obstacles at either of two ranges, 4 metres which is suitable for outdoor travel, and 1 metre, which is more suitable for indoors It has a vibrating display that responds only to the nearest target The frequency of the vibrations is inversely proportional to the distance of the object detected Audio output is available as an option No vibrations occur if the echo is below a predetermined threshold and no information about the nature of the detected object is presented, which although making it less accurate than the Sonic Torch, also makes the output clearer and easier to understand It can easily be used in conjunction with the cane or guide dog and is low in cost Training required for the Mowat sensor is small in comparison to other ETAs

2 3 1 4 The Laser Cane

This is a device specifically designed to eliminate some of the problems with the cane [3,7,16] It is a cane fitted with three laser sensors designed to detect obstacles at head and waist heights as well as providing advanced warning of step-downs The output consists of two (constant) auditory and one vibrotactile display, each output specific to an obstacle within a certain range and at a certain height Since it is incorporated into a cane, it leaves one free hand, unlike the obstacle detectors described earlier It can be subject to producing what is known as 'false positives', that is, on wet days it may signal the presence of a non-existent drop which is inconvenient but not dangerous It is however quite expensive and training is needed for correct operation

2 3.1.5 The PathFinder

This head-mounted device divides the space in front of a user into three zones [3] Information about obstacles in any of the zones is not displayed simultaneously, instead the three zones are each given a small amount of time Distances of objects in each of the zones are represented by musical scales, altogether there are eight notes, providing approximate distance information about objects Each note represents a 30cm bracket within the range of 0 to 210 cm If an obstacle is on a collision path

within a zone, the other zones are silenced and attention is drawn to the pending danger. It can be particularly useful for 'shore-lining', that is, by keeping the tone created by say, a building, constant, it is possible to walk in a straight line. PathFinders mountable to wheelchairs are also available. Training is required for use.

2.3.1.6 The Sonic Guide

Also developed by Professor Kay, this device provides a binaural description of obstacles in its path [3,12,16]. Two ultrasonic sensors mounted on spectacles allow the device to accurately detect the position and distance of objects. Frequency varies with distance while the different volume levels of the sound presented to each ear indicates the direction of the object, allowing the person to binaurally localise the object. As with the Sonic Torch, auditory output is produced almost constantly. The field of view is about 60^{o}. Similar to the PathFinder, the Sonic Guide can with appropriate training not only act as an obstacle detector but also allows the wearer to form a more complete image of his environment. It is quite expensive.

2.3.1.7 The Polaron

The Polaron is an ultrasonic mobility aid that can be hand held or chest mounted which detects object in one of three ranges; 4, 8 and 16 feet[3]. The display can be tactile or auditory, the frequency depending on the distance of the object. One advantage of the Polaron is that it can also be wheelchair mounted so as to allow shore-lining.

2.3.1.8 The Silva Compass

The Silva Compass is an ordinary magnetic compass that can be used as an orientation aid. It has a tactual interface. The major problem encountered with this device is its inaccuracy due to artificially generated magnetic fields, especially in urban areas.

2.3.1.9 The Auditory Compass

This compass is a magnetic compass with audio output. Can be set for specific destination. It suffers from the same drawbacks as the Silva compass.

2.3 1.10 Tactile maps

Various types of maps designed for visually disabled people exist Strip and bead maps show a reader suitable routes for travelling by means of symbols, while other types of map show significant details of areas

2.3.2. More recent developments

The information for this section was obtained through research on the Internet, not through a literature survey The EASI and BLIND-L discussion groups were the main sources of information

1 The Open University has developed navigation aids known as Talking Signs (designed by D B Jones) Talking signs are devices that contain pre-recorded audio information about surroundings and other useful information The messages are stored in solid state memory Two types of talking signs exist, Automatic Talking Signs are triggered by people walking past while the Triggered Talking Signs are triggered by people carrying a small pocket activator The size of the transmitter is about half the size of a bag of sugar and the activator about the size of a packet of cigarettes Signs can be mains or battery powered Cost is about £150 for signs and £27 for triggers The information given to passers by is the same irrespective of the direction of approach The Smith-Kettlewell Rehabilitation engineering centre is also developing a talking sign system

2 John DeWitt is currently developing a navigation system for Tampa airport, Florida No details are available as yet

3 Prof Reg Gooledge, Jack Loomis and Klatsky developed a Personal Guidance System (the PGS) at University College Santa Barbara It is a GPS based system, coupled to a Geographic Information System The interface is based on virtual auditory reality It is a backpack system that incorporates a flux-gate compass Objects and destinations call out to users, the sound appearing to come from the actual positions of the objects It is hoped that with further development, the PGS can be reduced to belt pack size

4 Dr Ronald Stephens of the Porthsmouth Institute of Rehabilitation Technology in co-operation with Tony Longley of Possum Controls Limited are working on the OPEN project, or Orientation by Personal Electronic Navigation The system is designed exclusively for way-finding in underground systems It is based on narrow beam infra-red beacons that transmit information and allow a degree of orientation

5 Chicago state university recently developed a system of localised transmitters which when placed at street intersections can produce speech signals identifying the intersection for any person carrying a receiver

6 Several years ago a bar-code reader with speech output was developed Bar codes can be placed at intersections, building fronts etc (presented at Closing the Gap conference, Minneapolis, 1989)

7 The RNIB has developed a system that allows travellers to identify objects like pedestrian crossings and phone boxes

8 The British Guide Dog Association has developed a similar system that allows pedestrians to identify objects by means of a triggered pre-recorded message

9 The MoBIC (Mobility of Blind and Elderly People Interacting with Computers) consortium is developing a GPS based navigation system The system involves two stages, the first is a pre-journey planning system and the second is the actual outdoor device that provides the traveller with the required information

2.4. Follow-up surveys of ETA use

The following are studies which were conducted in an effort to evaluate the usefulness of ETAs

In 1973, Airasian [12] conducted a mail-questionnaire based follow-up study of the SonicGuide 10% reported they no longer had the device, 11% had not decided about returning the devices while 79% reported they would keep the SonicGuides they had

In 1977, Darling, Goodrich and Wiley [13] conducted a follow-up survey of 23 army veterans trained to use ETAs Five did not use ETAs Twelve had sonic guides of which only 5 said they still made use of Of the 6 people trained to use a Laser canes, 3 said they still used it Of the people originally trained, 35% were still using an ETA

In 1981, Morrissette[14] conducted a telephone survey of 15 veterans who had been trained in the use of the Mowat sensor All reported that they used the sensor on occasions

In 1984, Simon[15] reported on five individuals trained to use ETAs Four had been trained to use the Laser cane and one the SonicGuide All subjects still used their devices

Blasch, Long and Griffin-Shirley[16] conducted a national survey of electronic travel aid use The survey was much more comprehensive than any of the above and included interviews with people who no longer used their devices A total of 298 people were interviewed Of all the participant, 40% were trained in the Mowat Sensor, 33% the SonicGuide, 25% the Laser Cane and 2% were trained to use the PathFinder

A total of 37% reported they found it very easy to use their ETA, 41% fairly easy and 23% said it was difficult The table below shows the type of ETA and the numbers that reported having used it within 3, 30 and 180 days The most commonly used devices were the Laser Cane and the Mowat sensor which are also the devices that have the simplest outputs

Type of aid trained on	Number trained	Use within last 3 days	Use within last 30 days	Use within last 180 days
Mowat Sensor	123	35%	59%	78%
SonicGuide	109	13%	25%	40%
Laser Cane	75	53%	65%	69%
Pathsounder	10	10%	20%	60%

Table 2.1

The survey goes on to define former users as those not having used their device within the last 30 days There were 140 users and 158 former users

Of the users, 52% reported that they travelled more often after having been trained to use their ETA Exactly half reported using the device equally in familiar and unfamiliar areas, 31% said use increased in unfamiliar areas Just over 44% of users always carried their devices with them while half of all the ETA users reported using their device constantly when they carried it with them Close to 65% said they used their device both inside and out while 60% reported 'more rapid, more efficient, more confident, safer, and less stressful travel' compared to travel before being trained in the use of their ETA Over 90% said they had fewer body contacts when using their aid To 58% of the ETA users, environmental noise was not a problem A total of 87% said their device was comfortable and 68% of Mowat Sensor users and 47% of SonicGuide users reported having at some time used their devices without a primary mobility aid

All the ETA users were asked the following question, "Does using your ETA make a big difference in your ability to negotiate these common travel situations?" The table 2 2 below shows the 'common travel situations' and the number that reported that their travel aid made a difference

Common travel situation	% that reported a difference
Avoiding obstacles at body level	77%
Avoiding obstacles at head level	68%
Assessing distance to and direction of objects	61%
Avoiding pedestrians in a crowd	64%
Locating a door to a building	50%
Avoiding objects at foot level, shore-lining, maintaining a straight line of travel, squaring off at kerbs, locating a place in line, crossing streets, locating up and down curbs and determining the size of objects	<50%

Table 2.2

Another question asked was "What is the one thing you like best about using an ETA?"

The top responses to this question are tabulated below

Response	Percentage who responded in this way
Object detection	24%
Increased independence	10%
Mobility (safety, ease of travel)	9%

Table 2.3

The former ETA users reported gradual declines in the use of their device When asked why they no longer used their aid, the responses varied greatly, the main ones of which are shown below

Reason for discontinued use of aid	Percentage who responded in this way
Personal or health reasons	14%
Preferred sole use of primary aid	13%
Change in environment that decreased the need for mobility aids	12%
Design problems such as weight	11%
General unfavourable responses such as "I just didn't like it"	13%

Table 2.4

Design change recommendations for the various ETAs as reported by both ETA and former ETA users are as follows,

Laser Cane

- Lower cost
- Improve reliability
- Reduce weight

Mowat Sensor

- Improve reliability
- Improve information provided

Sonic Guide

- Reduce size
- Improve reliability
- Improve information
- Improve appearance
- Reduce cost

2.5. End User Feedback

This section is based on the responses received from many blind and visually disabled people when asked their opinions and expectations of a navigation aid It must be pointed out at this stage that this survey was conducted over the Internet, which means that the average participant was not the statistically average visually disabled person in terms of age and income However it can be argued that the largest number of potential users of a navigation aid would be included in this (younger) group since their income is more likely to allow the purchase of travel aids and they also tend to be more aware of modern technology and its possibilities Responses varied greatly, from very negative to very positive In all about 250 responses were received A compilation of the responses is presented below The first part deals with the potential disadvantages of a fully functioning navigation aid and goes as far as to argue against the development of such an aid The number of people actually against the development of a navigation aid was very small (less than 10) and can be regarded as a minority

2.5.1 Why bother with a Navigation Aid?

A common assumption made by many sighted people is that cane and guide dog users cannot travel independently without a great deal of difficulty and stress Given proper training in the use of a primary mobility aid, the blind *can* travel wherever they choose to go without a great deal of difficulty and without the benefit of a navigation aid Learning new areas should not take a great deal of time, information may be easily acquired from other people and/or maps Obtaining information from other people is a very reliable method of supporting navigation in the sense that it is almost always available Tactile and Braille maps are inexpensive to produce, and are adequate for many areas

Navigation systems can enforce the popularly held opinion that travel for blind people is an ordeal by implying that they are helpless without such a system Negative images of the blind may also lead to reduced employment prospects since employers may envisage blind employees as being dependent and hence not as productive

People who use navigation systems may foster dependency and hence avoid areas not fitted with navigation systems, which is contradictory to the intended purpose of a navigation system, namely to increase the overall mobility of people with a visual disability The motivation to learn how to correctly use primary mobility aids may be eroded if navigation systems reduce the need for having these skills

2 5 2 Arguments in favour of the development of a navigation aid

While canes and guide dogs are adequate mobility aids in most everyday situations, they offer a limited amount of information about the environment Canes are mobility aids, essential for travel, but they do not provide information about the layout of an area other than the immediate area within reach of the cane Getting information about one's position within an area and the location of objects in relation to each other can be done by consulting maps, human guides or navigational aids Tactile maps are good at relating fine details but are less adequate when relating an overall view of an area Braille maps are also available but require a good knowledge of Braille Both human guides and navigational aids are superior in this respect In large, complicated environments, a navigation aid would allow a person to immediately access information without the need for tactile maps and without constantly having to ask for directions Asking for directions in itself is not an ordeal, but offering an alternative to having to ask questions would make life a little easier, especially if one suffers from speech problems

Navigation aids would be especially useful in complicated, unfamiliar environments not likely to be frequented which would make familiarisation with the area impractical, such as airports, railway stations and so on They could also be beneficial on familiar routes as best described below by a respondent,

> " anything that could augment the reception of stimuli garnered by the cane should be welcomed I disagree that a familiar route presents no problem to the traveller The fact is that conditions change--wind currents, snow, rain, personal attentiveness--and a navigation aid could surely supplement the meagre information which one can get from the cane alone In my travels, I

have to deal with rounded corners, and how many times have I gone too far, and had to retrace "

Snow especially can lead to problems even on familiar routes, covering familiar landmarks and surface details that would usually be used to ascertain orientation and position

Large environments can benefit from navigation systems not just for people unfamiliar with the area, but as an *optional* aid that can be consulted if desired For instance if one is visiting a less well known section of say, a university campus a navigation device can help in locating such sections and provide information about them in relation to other well known sections An added benefit is that people who use a navigation aid can easily acquaint themselves with all of the services and facilities available in a particular area and with which of these services and facilities are located conveniently

The letter shown below is from Greg Goodrich, a research psychologist at Stanford, who has been involved in the field of assistive technology for many years (see also follow-up surveys of ETAs) It is in response to the summary presented above of the various views and opinions expressed on the development of a navigation aid

"I thought the summary was fair to most positions I do, of course, take exception to some of the positions various people proposed, but then your goal was to present a balanced summary so they needed to be included

I do think the overall conclusion from the debate was that there is a need to pursue the topic of navigational aids for visually impaired travellers One could take the position that if such a navigational aid were put in place and that it was easy to use and inexpensive that it would be a potential service to other groups as well (i e , retarded citizens, various people with cognitive impairments, etc) and that it would be more cost effective in that it helped a larger number of people In any event, there seems to be sufficient support to warrant a funding request for a pilot project or initial demonstration project By the way the other way to look at the statistics that I've given you is that existing travel aids are inadequate[4] For example, in the

[4]That only 10% of the legally blind in the USA use canes

US the National Eye Institute estimates there are 3 million severely visually impaired individuals[5] The American Foundation for the Blind reports that there are 100,000 long cane users and less than 10,000 guide dog users That means that the two most frequent travel aids used by visually impaired people are used by only three and a third percent That is, 96% of the severely visually impaired do not use long canes or guide dogs Clearly, an easily used, inexpensive and widely available travel aid would have widespread appeal "

2 6. Audio and tactile interfaces

What would be the best method of relating navigational information to a person using a navigation aid? The most common interface types are audio and tactile Both have their advantages and disadvantages

Auditory perception can be very versatile and used for identifying and locating objects at considerable distances, in contrast to tactile perception which can only be used to detect and locate objects within the immediate vicinity Relating navigational information through tactile interfaces has the advantage that it leaves one's auditory senses free to concentrate on gathering the information required for basic mobility as well as listening to announcements, conversations and so on Other advantages that tactile interfaces have over audio interfaces is that audio interfaces usually consist of loudspeakers or headphones, both of which are not as discreet as tactile sensors Up to 35% of people with a visual disability also suffer from light to severe hearing loss [1], for whom tactile interfaces are a better option

Interpreting tactile information other than locating and identifying objects may require training, whereas audio information tends to be more self explanatory Many people cannot read Braille and as such Braille based information should not be essential for correct use of the device Some diabetics may also suffer from reduced tactual perception This would favour the use of audio interfaces, especially if the information presented is not continuous but no more than an occasional signal to indicate correct or incorrect directions of travel Tactile interfaces may also be quite bulky Each pin in a vibrotactile display must be separated by at least 2 mm in order for a person to be able to differentiate between them

[5]As opposed to the 1 million *legally* blind

Based on the above advantages and disadvantages of audio and tactile interfaces, offering a choice of interface types would seem essential in order to make the device practical for as wide as possible a range of people It may be possible to allow a person to select the degree of information coming from the tactile and audio information sources to suit his or her needs according to their environments The user can thus not only choose their preferred interface type, but also the degree of information being relayed by the device, for instance, the device may be consulted if required or set so as to give continuous navigation signals

Complex information related to a person through the sense of touch can best be done through Braille or Moon, however this not only requires knowledge of one or more of these alphabets, but a high level of concentration as well Verbally communicating complex information is very effective but also requires concentration In short, very detailed information should be avoided unless explicitly requested by the user The selection of the preferred destination may however require a relatively complicated interface such as a verbal audio or Braille tactile display Since the time that a user will have to spend selecting destinations will be very much less than the time spent receiving navigation signals, the effects of using a more complicated interface for this purpose is negligible Destinations need not be selected whilst on the move This further validates the use of more complicated displays when selecting destinations A problem arises here if a person has a hearing deficiency as well as not being able to read Braille A limited choice of destinations may be presented to such a user through means of symbols

Ideally, the user interface to be used with a navigational device should accommodate personal preference as to the degree of information offered and as to how the information is conveyed This would accommodate people who prefer either audio or tactile and those who prefer a mixture For example, the selection of the desired destination can be done by means of an audio interface, while the navigation signals may be conveyed through a tactile interface The degree of information may consist of two or more settings, such as a continuous indication of a persons position in relation to the selected destination, timed updates of the information or more simply the options available at junctions and so on

2.7. Design Considerations

Electronic navigation aids must be used in conjunction with existing travelling techniques if they are to provide a good service. They must in no way interfere with the travel techniques employed by blind travellers and must in no way limit the options available to them. In other words, a navigation aid must complement existing travel techniques and should *not* attempt to replace them.

A navigation system must not limit the traveller in any way. Systems that limit the traveller to fixed, pre-defined routes curtail the independence of the user in the sense that they exert a certain amount of control over the route options available to them. Individual preferences vary dramatically. What may be suitable for one may be less than adequate for another. Fixed routes do not allow for this. For example, guide dog users and cane dog users may rely on different aspects of the environment and may therefore choose different routes. Having to adhere to rigid routes may also demand concentration and effect one's ability to give sufficient attention to the various audio and tactile queues needed for basic mobility.

The position of the user interface is a very important factor. Hand held electronic devices may be useful but they have the disadvantage that any user will have both hands in use at the one time. Fitting a navigation aid to a cane has the advantage that one hand is left free. Further the handle of the cane is a good position for a tactual interface. Integrating a navigation aid with a cane excludes guide dog users as using both a cane and a guide dog is impractical. It is also important not to interfere with the information obtained from the cane. Fitting a navigation aid may affect the cane's weight, rigidity and may also reduce its collapsibility. From a design point of view, mounting sensors on a helmet can have many potential benefits. However, head mounted systems are conspicuous and, referring to the point made earlier about spreading the image of helplessness and dependency associated with blind people, are not desirable.

Expense is a major consideration. Developing a system that is costly, keeping in mind the large percentage of unemployed among the blind, will limit the numbers of people that will benefit from the device. To give a sense of the importance of cost, consider that only about 10% of the legally blind people in the US use canes, which are primary mobility aids and cost the relatively small sum of $35. While there are

many factors contributing to this low percentage[6], it puts into perspective the value placed on navigation and mobility aids by many blind people The finished product must therefore be inexpensive or subsidised Alternatively, locally managed areas such as airports and railway stations could provide a navigational system as a service This would require that potential users be alerted as to the availability of the system

2.8. Conclusions

2.8 1 General requirements of a navigation aid

The points listed below should be kept in mind when designing a navigational aid,

1 A device designed to assist with navigation for people with visual disabilities must be designed in such a way as to complement existing navigational techniques

2 It must be affordable and/or subsidised

3 It must be as inconspicuous as possible so as not to make the user unnecessarily self aware and convey an image of dependency and helplessness

4 It should give clear and easy to understand navigation signals that do not contain on excess of information

5 It should be user friendly and require a minimum of training for correct use Navigation aids that are difficult to use and that require training will both limit the scope of potential users and require training instructors

6 No limits should be placed on the options and travel preferences of the user, which means that the system should be flexible enough to allow a user to travel as they see fit along the route of their choosing while providing navigational information to the traveller if it is required

[6]One of the main reasons is that despite being legally blind, many (approximately 75%) have enough vision to allow unaided mobility

2.8.2 The User Interface[7]

1 The device should be able to display destinations which the user can select in some suitable way and should also be able to convey simpler navigation information

2 For tactile displays, it would be preferable if the use of a hand is not (always) required, i e the user should be able to position the display on their person rather than carry it in the hand

3 The user interface should be able to accommodate a wide variety of user preferences

4 The information displayed should not interfere with audio or tactile information necessary for basic mobility

[7]Electrocutaneous displays which use electrodes to stimulate the skin are not considered

3. Proposals

This section looks at some possible navigation aid and user interface design Each subheading deals with a different aspect of the overall design Several solutions are outlines as responses to these design issues

3.1 Selecting a destination

The problem of how the user can inform a navigation aid of the destination they wish to go is addressed here This suggested interface could be used with any of the systems proposed in this section Simply typing in a desired destination is not practical since it requires a prior knowledge of the names that the navigation aid has assigned to this particular destination The user should have the option of choosing a destination This has the added advantage of bringing to the attention of the user all the available facilities and services within a complex One possible way of solving this problem is to have a list of all available destinations, divided up into consecutively more detailed categories, for instance a university's list of destinations could be divided up as shown in Figure 3 1

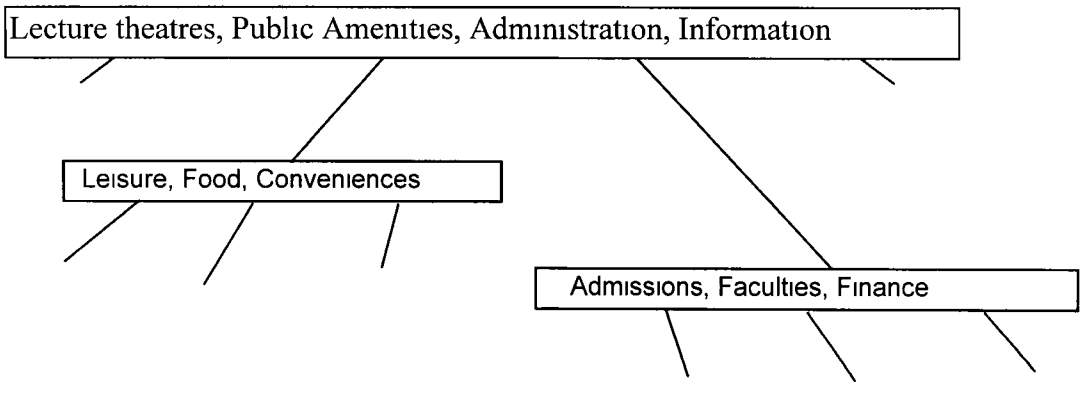

Figure 3.1

This information can be conveyed to the user by audio or tactile information For each category, the user may select his or her preference by either pressing an 'OK' button when the preferred option is heard or read, or by using a numbered keypad, each selection being associated with a number The former method is preferable from

the point of view that it requires fewer keys, however a repeat last message key would be essential. Accessing preferred destinations would be relatively quick, depending on the number of destinations available. Arranging the information in this format would be useful to all people entering an unfamiliar environment, sighted or not.

3.2. Establishing position [17]

This is a very complicated part of the development process. Accuracy, range, structure of environment, hardware and software as well as installation and maintenance costs are all significant factors. Very accurate systems tend to be more costly and complex. If the system is to be used solely outside, less accuracy is needed but greater range is required. It is possible to divide the proposed navigation aids up into two main categories; the first is where the navigational aid is capable of establishing its whereabouts independently, the second is where the position of the aid is established by a separate system. The latter type of navigational systems do not need to have their own on-board circuitry capable of establishing their positions which means they are less complicated, however it also means that either the established position or the required navigation signals would have to be transmitted back to the device. If the navigation signals are to be transmitted back, the intended destination must firstly be transmitted from the navigation device to the external controller. Three dimensional positioning is essential for navigation in multi-story complexes. Some of the proposed systems below only allow two dimensional navigation, making them more suitable for navigation between building unless used in conjunction with an other technique such as user feedback to establish the third dimension.

3.2.1. Independent systems

Independent tracking systems rely on using 'smart' hand held devices that are capable of establishing position and calculating any required information themselves.

3.2.1.1. Local transmitters

The installation of a series of low power transmitters in an area would allow a receiver with prior knowledge of the location of each transmitter to establish its approximate

position Transmitters may also be mounted on obstacles to give warning to people within the vicinity Both infra red and radio transmission could be used for this This method is limited in accuracy, however it is sufficient for a system that does not guide the user every step of the way As noted before, guiding a user every step of the way along a route is not desirable An advantage of radio transmitters is that they can easily be adjusted to cover different sized areas, thus making them suitable for both inside and outside Infra red has the disadvantage that detectors capable of detecting an infra red signal must be positioned so as to have line of sight with the transmitters The above method allows three dimensional positioning within a complex since transmitters can be located wherever desired

3.2.1.2. Global positioning

Global positioning (GPS) technology is advancing rapidly, and is becoming increasingly accurate and inexpensive The output of a portable GPS is in latitude and longitude The major drawback that GPS has is that it requires line of sight with overhead satellites for correct and reliable operation This means that it cannot be used inside or in the vicinity of very tall buildings Another drawback is that only two dimensional positioning can be achieved

3.2.1.3. Phased Arrays

It is possible to direct a radio signal in a particular direction quite accurately by cancelling out the transmissions to all other directions using other, out of phase, transmitters Antennas working together in this way are knows as antenna arrays Many different types of array exist, varying in size, cost and accuracy Two or three such arrays can 'sweep' an area with a signal that may be used to establish the position of the receiver A synchronisation signal is emitted in all directions The time between this pulse and the time it takes a particular transmitter to transmit in the direction of the receiver is directly proportional to the direction that that particular signal is being transmitted at Three such signals would thus establish the position of the receiver in two dimensions

3.2.1.4. Inductively coupled sensors

Inductively excited sensors can be used to transmit data and hence establish position Such sensors are cheap but have a very limited range

3.2.2. Systems with external control

Navigation systems with external control consist of a portable 'dumb' aid which relies on an external control system to establish position and compute any requested information

An example of such a system is shown in the diagram below,

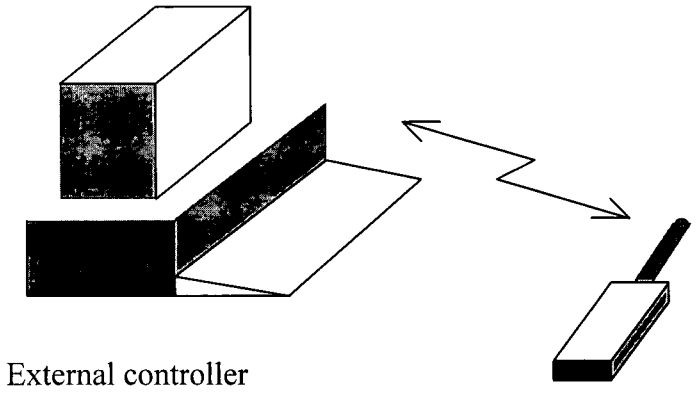

Figure 3.2

3 2 2 1 Direction finding

Transmitted signals can easily be sourced If a navigational device emits radio signals, triangulation techniques can be used to establish its position quite accurately This is similar to the method described in the independent systems category (Section3 2 1), the only difference is that there is no need for a synchronisation pulse and the navigational signals or positions have to be transmitted to the device

3 2.2.2 Ultrasonic transmitters

Ultrasonic receivers can be used to establish the position of an ultrasonic transmitter Due to the relatively slow speed of ultrasound in comparison with radio waves, the difference between the times a particular ultrasonic signal is received by two or more receivers can be used to establish the position of the ultrasonic transmitter Three ultrasonic receivers within view of the transmitter are required to prevent inaccuracies due to reflected signals The transmitter must first transmit a signal in order to

synchronise the receivers This method can be very accurate depending on the proximity to the receivers Since many receivers are required to cover an area, three dimensional positioning is possible

3.3. Storing spatial information

The independent tracking systems described in 3 2 1 can be used in conjunction with navigation aids that have the circuitry required to provide navigational signals on-board In order to provide these signals, a knowledge of the spatial layout of the area is required Independent systems must therefore also have stored spatial information or in other words a built in Geographical Information System (GIS) is needed This may not be practical if the area to be covered is complex and requires a large amount of detail to be stored The spatial information can be transmitted to devices in an area instead of having the information on board This would make the device more versatile In order to transmit such information, a method similar to the way teletext information is transmitted could be used In other words information could be 'piggy-backed' on signals used for direction finding This problem need not be considered with systems with external control as described in 3 2 2

3 4 Relating navigational information to the user

3.4.1 Compass based interface

Once the position of a person with a navigation aid has been established, data can be relayed to that person that will allow him or her to move along an appropriate route with respect to the intended destination The need to keep this information simple and concise was discussed earlier One viable option of relating this type of data is by means of a programmable, non-magnetic compass The suggested direction of travel can easily be conveyed by this method The user interface can be as simple as left, right or straight ahead The use of a compass will not restrict conventional travel techniques or limit the options of the traveller If, for example, a user decides to ignore the advice given by the device and travel in a different direction, the suggested direction will simply change Systems that have predefined routes would not be able to cope with such situations as easily and therefore limit the user options An added

bonus is that the compass can be used for ordinary travel outside areas fitted with navigation systems. Such a device can be worn around the waist and can easily be fitted with tactile and audio interfaces. What makes this method attractive is not only its versatility, but also the fact that signals that the user is receiving allows them to easily keep track of the direction in which they are facing. Flux-gate compasses and gyroscopic compasses would be suitable for this type of system.

3.4.2 Virtual map

This option is particularly versatile. It does not require very accurate tracking. An area can be divided into a grid, each section of the grid appearing on the users tactile map as they move into it. A compass is an optional extra that could assist orientation. A less detailed display of the overall area covered could be made available which would make the selection of a particular destination optional. Routes can be decided by the traveller as with ordinary maps which in no way limits any travel options. The main problem with this option is the likelihood that such a system would be quite bulky, use a considerable amount of power, and be quite expensive.

3.5. Options

3.5.1 Learning capability

It is possible to add a 'learn' option to a navigation device, in other words allow it to automatically keep track of the route so far travelled so that the route can easily be retraced in the future. This could be especially useful if the end destination of the route is not available on the navigation system itself or if the recommended route is not suitable. Alternatively, such a device could be used to create custom made electronic strip maps for areas not fitted with the necessary equipment for keeping track of the device. The user could update the program whenever they wish. No information about the distances between objects on the 'map' would be recorded if the device was outside the range of the tracking system. A typical program could run; left, left, straight, stairs etc., while for areas within range of the tracking device the navigation signals would be given as before, i.e. if the user is in a certain position, the device recommends the direction of travel.

3.5.2 Obstacle detection

A navigation aid will indirectly provide information to a traveller about static objects by suggesting routes that are free of static obstacles, however movable obstacles such as cars, pedestrians, temporary obstructions and so on may present a problem to the visually disabled pedestrian Despite the fact that navigation aids as described above should complement primary mobility aid use, electronic detection of temporary obstacles may prove beneficial It may be possible to include temporary obstacle detection by adding extra sensors Information about these obstacles could be presented to the traveller in exactly the same fashion as for static objects, i e a route could be suggested away from the obstacle An extra indicator could be used to identify whether or not a temporary obstacle has been detected Such an option could also be beneficial to the user outside the area in which the navigation aid is functional It is important that such an addition should be optional to the user as the information received may be more complicated and thus lead to confusion

3 5.3 Audible clicks

As described in section 2 2 6, sound can be used for echolocation Adding a feature onto a navigation aid system that allows a user to hear clicks emitted at chosen intervals may be useful for some individuals, such as guide dog users who do not have the benefit of the sounds created by the cane when it strikes the ground This option would probably be more useful in areas outside the range of the intended use of the navigation aid

4. Control Software

4.1. Technical summary

All the system proposals outlined in chapter 3 be split up into the sections shown below

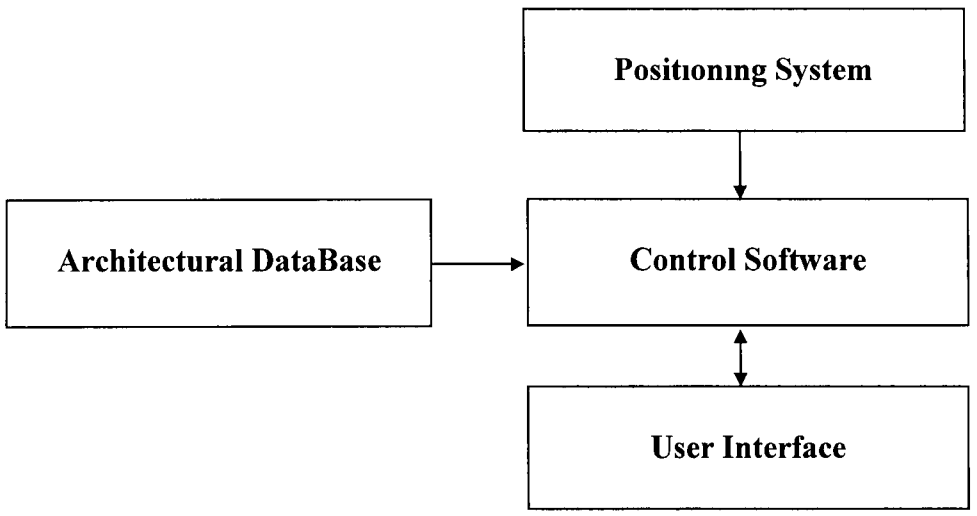

Figure 4.1 Overall System Block Diagram

The system shown above consists of a central controller whose job it is to interface with the positioning system, architectural database and user interface and to calculate the required information For example in response to a user's request for information on how to get from a location A to B, the controller would have to

- retrieve the user data from the user interface
- request user location data from the Positioning System
- obtain the appropriate drawings from the architectural database
- calculate the required navigational information
- supply the information to the user via the user interface

The above procedure seems quite straightforward The greatest challenge is the actual calculation of navigational information The control software must be able to access and extract useful information from architectural drawings Ideally it should also allow these drawing to be edited to include temporary obstacles or structural changes

If these functions are implemented on a central controller its functionality may be extended

As previously mentioned, extensive field testing of any navigational aid will be required before a design can be finalised The function of the controller can therefore be extended to act as a development tool Typical applications could include

1 Feasibility testing for various types of Positioning System (PS)
2 Many PSs require the installation of a network (e g localised transmitter PS) Such networks could be modelled on the control software and tested for suitability before actual installation of the networks
3 Various navigation systems could be tested for suitability for existing and yet to be constructed building complexes Work can begin as soon as architectural drawings are available
4 The software could also be used as an automated stand-alone information point such as the type commonly seen in shopping complexes
5 The software could be used for pre-journey route planning
6 A standardised interfacing system could allow the software to control many forms of user interface, ranging from virtual maps to Braille and audio interfaces

As the controller software is central to the operation of a PS, the project will concentrate on the development of the software required for the controller The following sections deal with the design and implementation of this software

4.2. Software Design specifications

The software must be able to

- Access drawings in an architectural database
- Display these drawings and allow features to be inserted and deleted
- Identify any given part of a drawing
- Locate any requested position or feature
- Understand the relationships between drawings and features in the drawing, for example it must understand how two particular floors of the same building are interconnected or how two adjacent rooms on a floor are connected

- Calculate routes between any two points in a complex of buildings (inside and outside) and choose the one that best fits the user defined criteria
- Output various detail levels, ranging from step by step guidance to general directions

One of the first considerations to be taken into account is the choice of language and platform In the last few years a new programming style has gained immense popularity, both for its versatility and adaptability This technique is known as Object Orientated Design OOD allows code to be updated with much greater ease than traditional design methods This feature is especially important in this application, where system specifications may well be required Object orientated designs can be carried out on several different programming languages, the most popular Object Orientated Programming (OOP) platform being C++

The choice of platform must take into account several factors, popularity, ease of use, and suitability for the given program specifications Since the program must produce results in the shortest possible time, speed is a major consideration UNIX workstations would be ideally suited but are not in wide spread use For PC based systems, the two most popular operating systems are DOS and Windows The major advantage that Windows has over DOS is its memory management system Sixteen bit Windows programs can access up to 16Mbytes of memory, while ordinary DOS programs are still limited to 64 Mbytes DOS extender programs allow multi megabyte programs to be written on DOS, but do not have the virtual memory system incorporated into Windows Many Windows programs have what is known as MDIs or Multiple Document Interfaces This means that a program can open documents (i e access data) without have to discard or close any documents that may already be open This feature is especially useful for a program that may have to access several drawings in order to calculate the required information Windows thus has the advantage of a superior memory management system, ease of use and tremendous popularity The implementation choice is thus a Windows program to be written using the Microsoft Visual C++ package This package contains a library of classes which greatly eases the development of Windows based applications

4 3. The Drawing Interchange File

One of the most fundamental features of the software is the accessing of drawings stored in digital format Existing CAD programs such as AutoCAD, Sketch, Insite etc all allow a user to save information in a standardised format that enables drawings created on such packages to be accessible in other packages Such files are known as Drawing Interchange Files or DXF files If the software to be developed is capable of reading these DXF files, communication with dedicated architectural databases, such as Insite is possible

Each drawing may consist of any combination of lines, circles, arcs and solids on one or more of the layers Any program capable of translating a DXF file into an actual drawing must be able to understand the DXF protocols and translate them to actual, 'on screen' images Each DXF file includes information about the dimensions used on the drawings, zoom and scroll information, text styles, line types, layer names and so on

Drawings made with CAD packages have the ability to split up the drawing into different layers This can be a very useful attribute It not only allows different features to be separated, such as elevators and stairs, but also allows a limited amount of three dimensional information to be stored For example, a layer may be reserved for information about head-height obstacles and other potential hazards

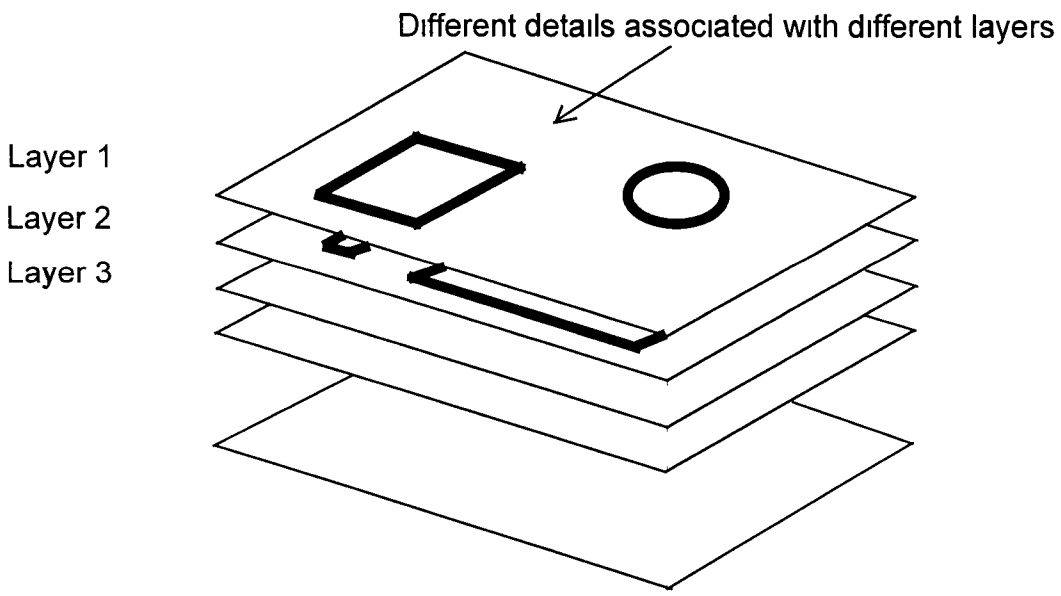

Figure 4.1

4.4. Object Orientated Design

Much of the following chapter deals with terms such as objects, classes, instances, virtual functions etc A short introduction to OOD is given here [18]

The underlying principle to an object orientated design is that it consists of a series of inter-related objects, each object being modelled on a real world entity For example, if we wish to write a program to regulate the rental of say, bicycles, we could create a software object representing a bicycle Each bicycle will thus have a software representative

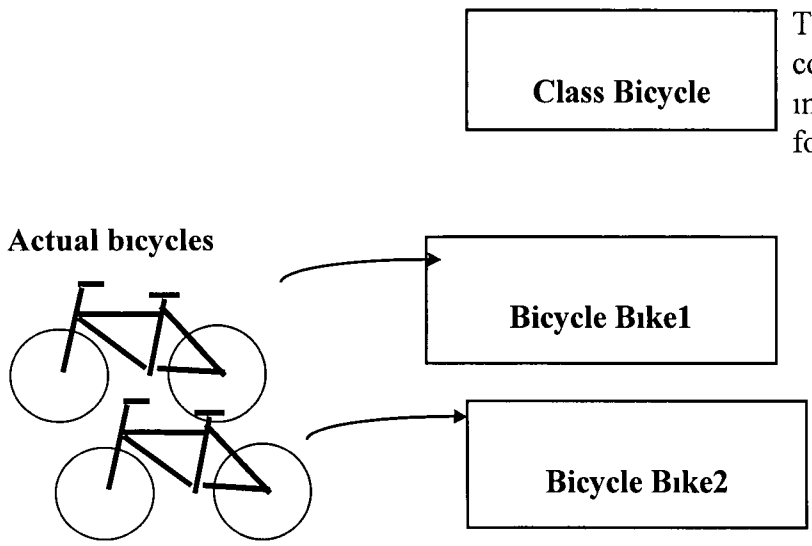

Figure 4.2

When implementing this in code, we would create a software *class* called Bicycle[8] Within this class we could store such parameters as, date of purchase, rental status, date of last check-up and so on We can classify this information as either *public* or *private* Private data refers to data that can only be accessed from within the class or through interface functions Public data can readily be accessed For instance, we could make the rental status public, which means that we can easily set or check its status The date of purchase data on the other hand, will only need to be set once and will never need to be changed We could therefore make this data private and add a function to the Bicycle class which returns the date of purchase

[8] Frequent references to objects and classes will be made throughout most of this chapter A capital letter will be used when referring to specific object or classes such as the Bicycle class

If we wish to distinguish between say ordinary bicycles and mountain bikes, we do not need to write a completely new class for the mountain bicycle Instead, we can use a feature called *inheritance* A new class called, say, MountainBike can be created that inherits all the features of the Bicycle class and can have features of its own, for instance it may contain information about its tyre size

Suppose we wish to add functions to calculate the cost of rental The cost is based on the amount of hours the bicycle has been rented for If the mountain bikes are more expensive to rent, we could add separate functions to calculate cost to both the Bicycle and MountainBike classes, however, since the MountamBike class is based on the Bicycle class, this would not be very efficient A better solution is to use what is known as *polymorphism*, or put simply, to over-ride an inherited function with a new one Thus, if we put a function called say, ReturnBike( ) in the Bicycle class, we can override this function in the MountamBike class to calculate the appropriate value for mountain bikes (see example code below)

There is another feature that makes Object Orientated Programming very powerful It is called the *virtual function* To continue with the example of the bicycle shop, let us suppose that the mountain bikes are only a recent addition to the rental shop and that a considerable amount of code has been written using the Bicycle class That same code must now be made compatible with the MountainBike class If we define the ReturnBike( ) function in the Bicycle class as virtual however, we may not have to rewrite any code The following example is intended to illustrate how the virtual function works

```
Class Bicycle
{
private
        BOOL rented,                    // True if the bicycle been rented out
        int time,
public
        Bicycle( ),                     // Call this to construct a Bicycle object
        ~Bicycle( ),                    // Call this to destroy a Bicycle object
        void   Rent(int),               // Rent bike
        virtual mt ReturnBike(int),     // Calculate rental


},
```

```
Class MountainBike  public Bicycle     // MountamBike inherits all of class Bicycle's
                                        // data and functions
{
private
        int tyre_size,                  // can add extra data to derived classes
public
        int ReturnBike(int),            // MountainBikes version of ReturnBike
},

////////////////////////////////////////////////////////////////////////
// This section implements the functions declared above
Bicycle  Bicycle( )
{
rented = FALSE,
hours = 0,                              // Every new Bicycle has 0 hours usage initially
}

void Bicycle  Rent(int start_time)
{
rented = TRUE,
tiine = start_time,
}

int Bicycle  ReturnBike(int return_time)
{
rented = FALSE,
return((time - return_time) * 5),                   // 5 pounds an hour
}

MountainBike  MountainBike( ) Bicycle( )  // When we create a MountainBike we
                                          //  also  call  the  Bicycle
constructor, since                                  // MountainBike is based
on Bicycle
{
}

int MoutainBike  ReturnBike(int return_time)
{
rented = FALSE,
return((time - return_time) * 7),           // 7 pounds an hour for inountain bikes
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
// SomeFunction( ) is intended as an arbitrary function that uses a Bicycle pointer

void SomeFunction( Bicycle *pBike )   // A function that takes a Bicycle pointer as a
                                      // parameter
{

int cost = pBike->ReturnBike( return_time ),      // Call to the ReturnBike() function

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//The Main program

void Main( void )
{
Bicycle Bike1,              // Create a Bicycle object
MountainBike Mbike,         // Create a MountainBike object,

SomeFunction( &Bike1 ),     // Pass a pointer to Bike1 as a parameter
                            // to SomeFucntion

SomeFunction( &Mbike ),     // Note that a pointer to a MountainBike is being passed
                            // to the function  This is allowed since MountainBike
                                  // is based on Bicycle  Because the ReturnBike
function                          // was declared as virtual, the MountainBike
version                           // will still be called within SomeFunction
                                  // There is therefore no need to create another
version of
                            // SomeFunction( ) for MountainBikes

}
```

Having introduced some of the main concepts of OOD, we can now consider the implementation of the control software using these methods

4.5 The OOD for the control software

It is the software's job to provide information to a user The content of this information will be based on actual, existing objects For example, information about a users current whereabouts or information of how to get from one point to another cannot be calculated without prior knowledge of the layout of an area The method by

which the software must interface with real world objects is through architectural drawings

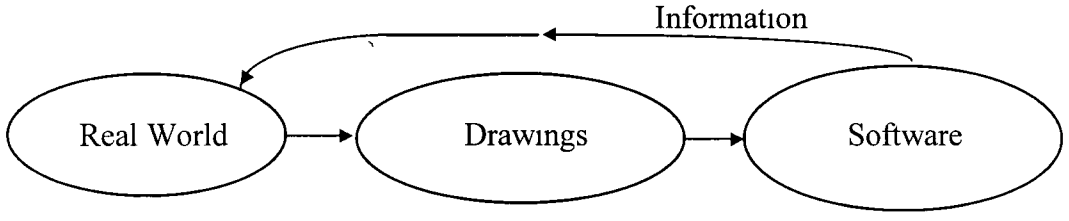

Figure 4.3

4.5.1. The Drawings

A method must be chosen whereby the drawing retrieved from the architectural database can be stored in memory Each drawing will have a different number of drawing objects such as lines or circles associated with it and can have a unique number of layers and so on This brings the question of memory management to mind It is not good enough to set aside say ten layers for each drawing with a fixed number of lines on each layer Dynamic memory allocation is required Consider the diagram below

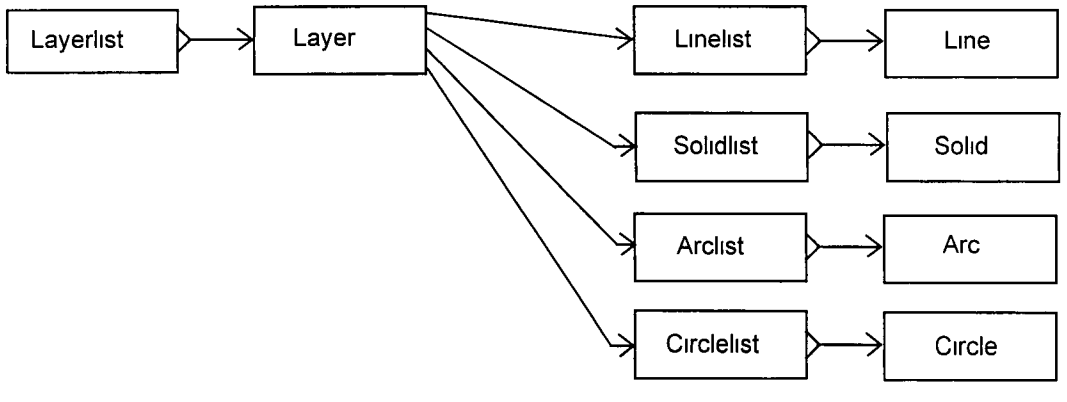

Figure 4.4

The above diagram is known as an entity relationship diagram There are two types of arrow used in such diagrams, the single tailed arrow and the double tailed arrow A single tailed arrow implies that only one of the entities being pointed at is contained within the entity from which the arrow originates A double tailed arrow means that one or more of the entities being pointed at may be stored

The Layerlist entity or object shown above can store any number of Layer objects using linked list techniques The Layer object contains information such as the layer name, the default colour of any drawing objects associated with it and whether the layer is active The Layer object also contains one Linelist, one Solidlist, one Arclist and one Circlelist Each of these list can contain any number of elements The list objects are thus responsible for allocating and de-allocating memory as objects are added or deleted Each Line object contains information about the start and end co-ordinates of the line and can contain other information such as colour etc The Circle objects store a centre point and a radius and so on Other features such as text can also be added

When the above diagram is implemented in code, it can be used to store a complete drawing Firstly a Layerlist object is created An appropriate amount of Layers can then be added to this object Individual Line and Arc objects etc can be added to each of these layers Every object stored in the lists can called on to draw itself This gives full control over which layers to display and which features on that layer to display

It is important not to confuse the above diagram with a diagram showing which classes inherit properties from another The hierarchical relationship between the drawing objects is shown below

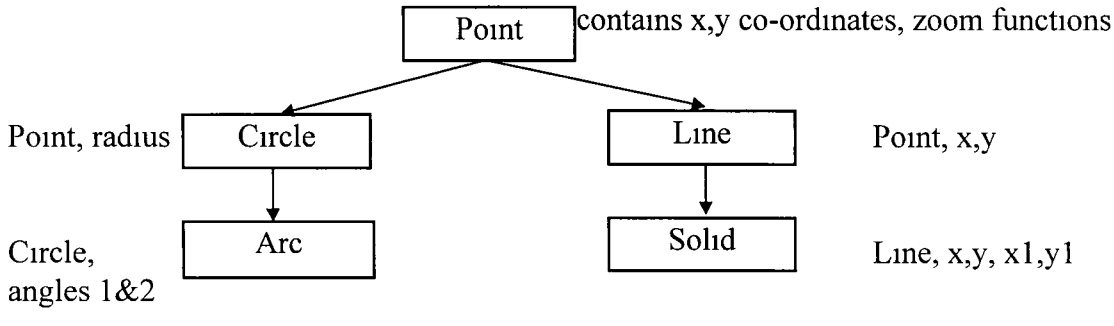

Figure 4.5

The base class, Point, contains functions common to all the objects, including zoom tools It also contains a single x,y co-ordinate The Circle class thus inherits the x,y co-ordinate from Point and also has radius information The Arc class inherits all this plus it contains start and stop angles The Line class has the Point class's x and y, and

also has an x,y of its own It would be possible to substitute this x,y for another Point There is however little sense in doing this, since this second point would only be required to store a simple x and y co-ordinate and does not need to have access to the functions stored in Point as these functions are already inherited The Solid class contains a Line and has another two sets of x,y co-ordinates, which can be used to draw a four sided figure

4.5.2. The Real World Objects

The next design problem that needs to be tackled is the question of how to teach the computer to interpret architectural drawings A drawing is in essence merely a collection of lines and text The problem is further complicated by the fact that no single drawing can show all the necessary information for navigating around a complex of buildings For example it is not possible to show the details of all the floors in a building on just one drawing The computer must therefore also have an understanding of the relationships between drawings To overcome this problem, a solution was developed in which a set of drawings representing a complex of buildings is reduced into a single model A simplified entity relationship diagram is show below

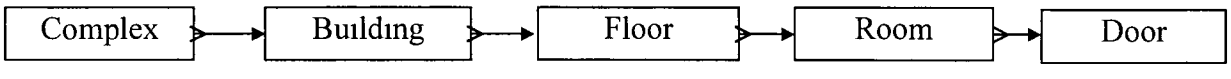

Figure 4.6

This diagram is intended as an overview of the relationships between the entities in the model The overall model is considerably more complex Each entity will be explained in turn in the following sections

4.5.3. The Complex Class

This class is representative of an actual complex of buildings Like its real world counterpart, it contains a set of buildings Its purpose is to keep track of the relationships between these buildings in terms of position and orientation The

Complex class can thus be used to co-ordinate the route finding algorithms that calculate paths between buildings

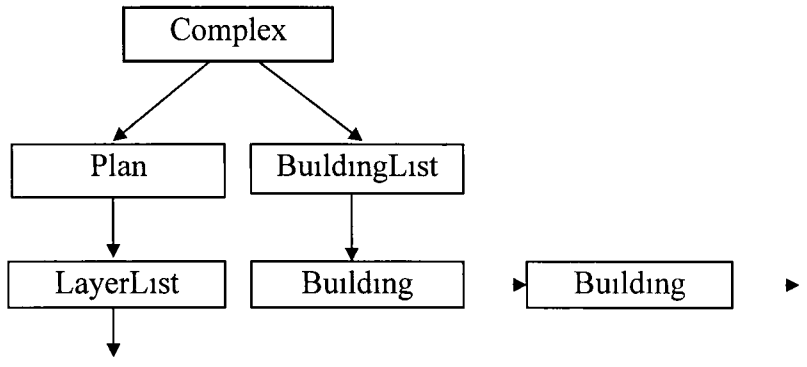

Figure 4.6

The diagram above shows the Complex class and its dependencies It inherits the function necessary for storing a list of buildings from a class called BuildingList A Plan object (see Section 4 4 4) is part of the Complex class's data The Plan is essentially an overall plan drawing of the actual complex of buildings from which the individual buildings in the BuildingList can be identified and located It should be noted that all the objects contained within the model can be accessed either directly or indirectly through the Complex class Individual buildings may be accessed directly which in turn gives access to any objects contained within these buildings

The Complex class can save any information stored in it in file format The files used have the extension CMP The CMP format is given in Appendix A

4.5.4. The Building Class

Each instance of the Building Class represents an actual building in the real world Each building thus has a unique name (and/or identification code) and may contain any number of floors, stairs, rooms and so on

The main components of the Building class are thus a list of the Floors it contains, plus a record of the Stairs and Lifts that connect these Floors The FloorList class acts as a base class from which the Building class inherits the ability to add, delete and sort any number of Floors The Floor class is dealt with in more detail in Section 4 5 5 The Stair and Lift objects contained in the Building class require a little more explanation A building can have any number of staircases and lifts Each of these

can be connected to one or all of the floors in a building Both the Stair and Lift classes are based on the RoomCapsule class This class keeps a record of which floors the staircase leads to and also the name of the Room or stairwell that contains the stairs on each of these floors The reason why the relevant Rooms (or pointers to them) cannot be stored directly in the Stair class but use the RoomCapsule class instead is that each Room object is already contained within a linked list (called RoomList, see Section 4 4 3) It cannot therefore be inserted in another linked list directly The use of the RoomCapsule class overcomes this problem The Building class can thus contain any number of Stair and Lift objects, each of which can in turn contain any number of connecting Floors and Rooms

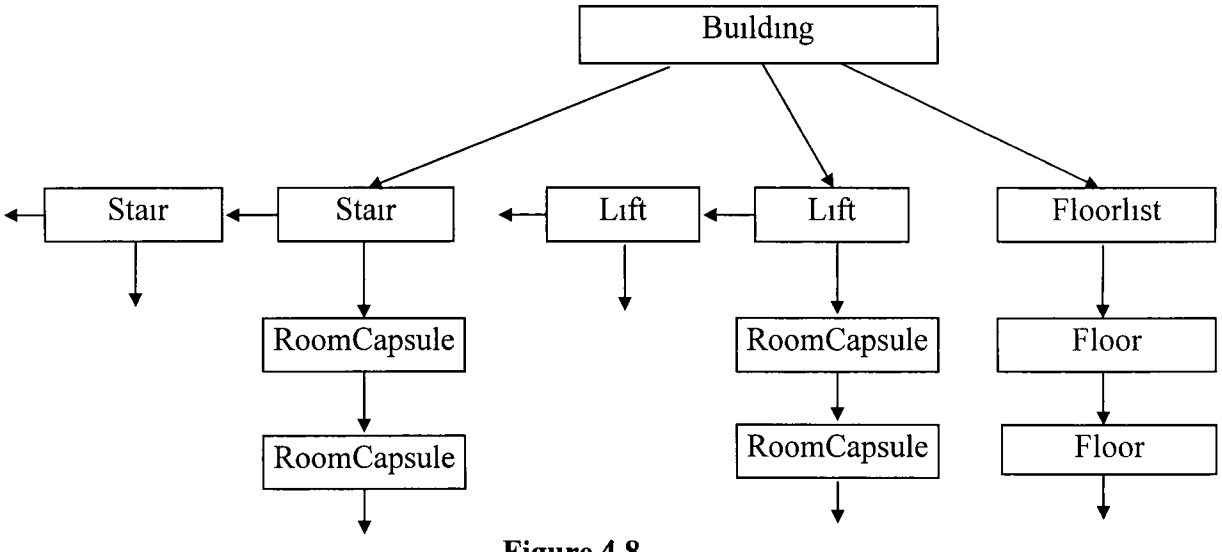

Figure 4.8

4.5.5. The Floor Class

One of the most fundamental classes is the Floor class This class is responsible for directly loading and saving the architectural drawings It contains the identification algorithm tools (see Section 4 6) as well as the routing tools It is through this class and its derivative, the Plan class, that most of the required navigational information is calculated The diagram below shows the Floor class and the objects contained within it

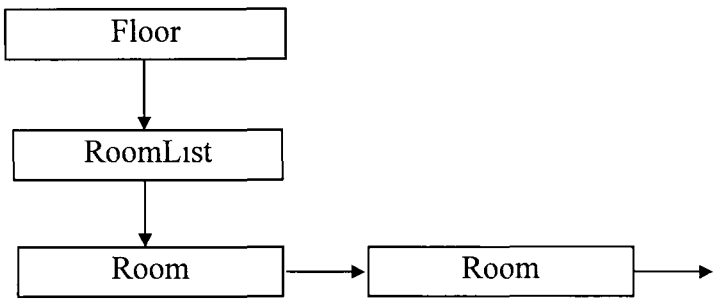

Figure 4.9

The class contains information about

- Layer assignments (i e which drawing layers contain the walls, doors etc)
- Colours, such as fill colours, wall colour and so on
- Drawing limits and scaling factors This data along with colour and layer information is required by the identification algorithm
- Names, such as room, stair and lift labels as well as the overall floor name
- All the rooms on the floor

And contains tools to

- Load files in DXF format
- Load and save files in FLR format (see Appendix B)
- Load and save the TRE file containing the RoomList contents
- Find and identify given locations (Section 4 8)
- Call the route finding algorithms (Section 4 9)
- Automatically identify objects in a given drawing and build up the resulting RoomList

4.5.6. The Plan Class

The Plan class is derived directly from the Floor class It is used to store a plan drawing of the overall complex of buildings Whereas the Floor class is used to calculate navigational information for inside buildings, the Plan class is used for outside of buildings The route-finding algorithms in the Floor class are defined as virtual functions The Plan class overrides these to calculate outside paths and routes

The same sections of code responsible for calling the Floor route finding tools can call the Plan route finding tools

4.5.7. The Room and Door classes

These classes are used to build up a tree or list whose elements are connected in exactly the same way as the rooms and doors on a given floor The entity relationship diagram for these classes is show below The RoomList is contained within a Floor object (Section 4 5 5)

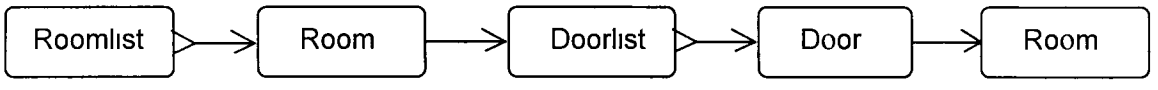

Figure 4.10

A fully interconnected tree can have many connections To illustrate how these connections are made, consider the diagram of a very simple floor drawing given below

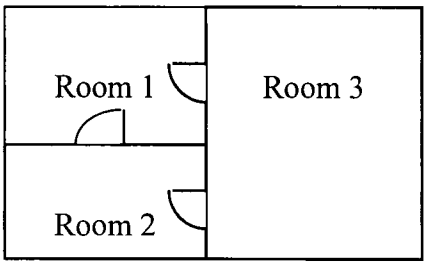

Figure 4.11

The RoomList for the above floor would look like

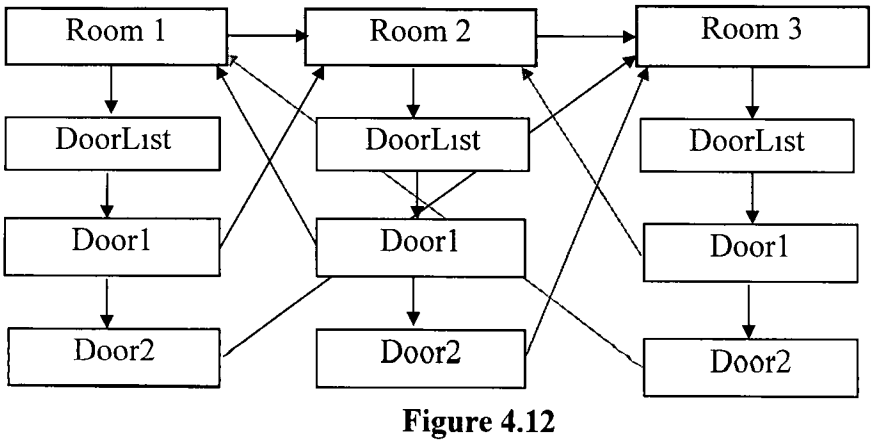

Figure 4.12

The first door in Room1 thus leads to Room2, the second door to Room3 and so on At this stage, it is possible to see the advantages of constructing such a model Once the model is made, it is no longer necessary to consult the drawing for information about the location or identification of objects It is also possible to determine the relationships between objects from the model

The Room class contains an x,y position and a name and number It can be 'locked' to route finding algorithms by setting the Locked flag The complete software package contains an editor that allows the Room information to be edited, for example a Room may be locked or unlocked, or Doors may be removed or added at will Every Room object also contains a list of Aliases, or names that the Room may also be known as The reason for this is explained in the section on the identification algorithms (see Section 4 8)

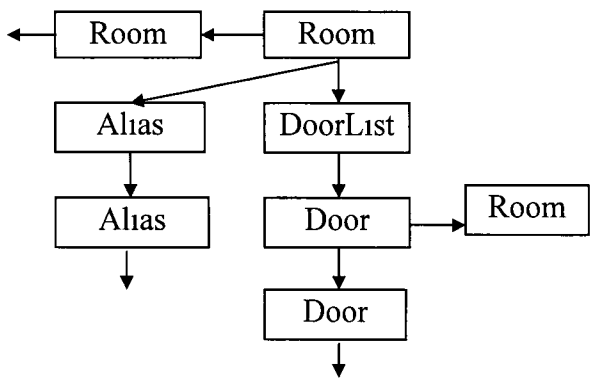

Figure 4.13

The Door class contains information about its position and has x,y co-ordinates for both inside and outside the actual door A Door object can also be locked The difference between locking a Door and locking a Room, is that a locked Room cannot be accessed by any door, while a room containing a locked door may still be accessed through another door

4.5.8. The Complete Model

Figure 4.14 shows the class structure of the complete design. It gives an overview of how each class is related to the others.

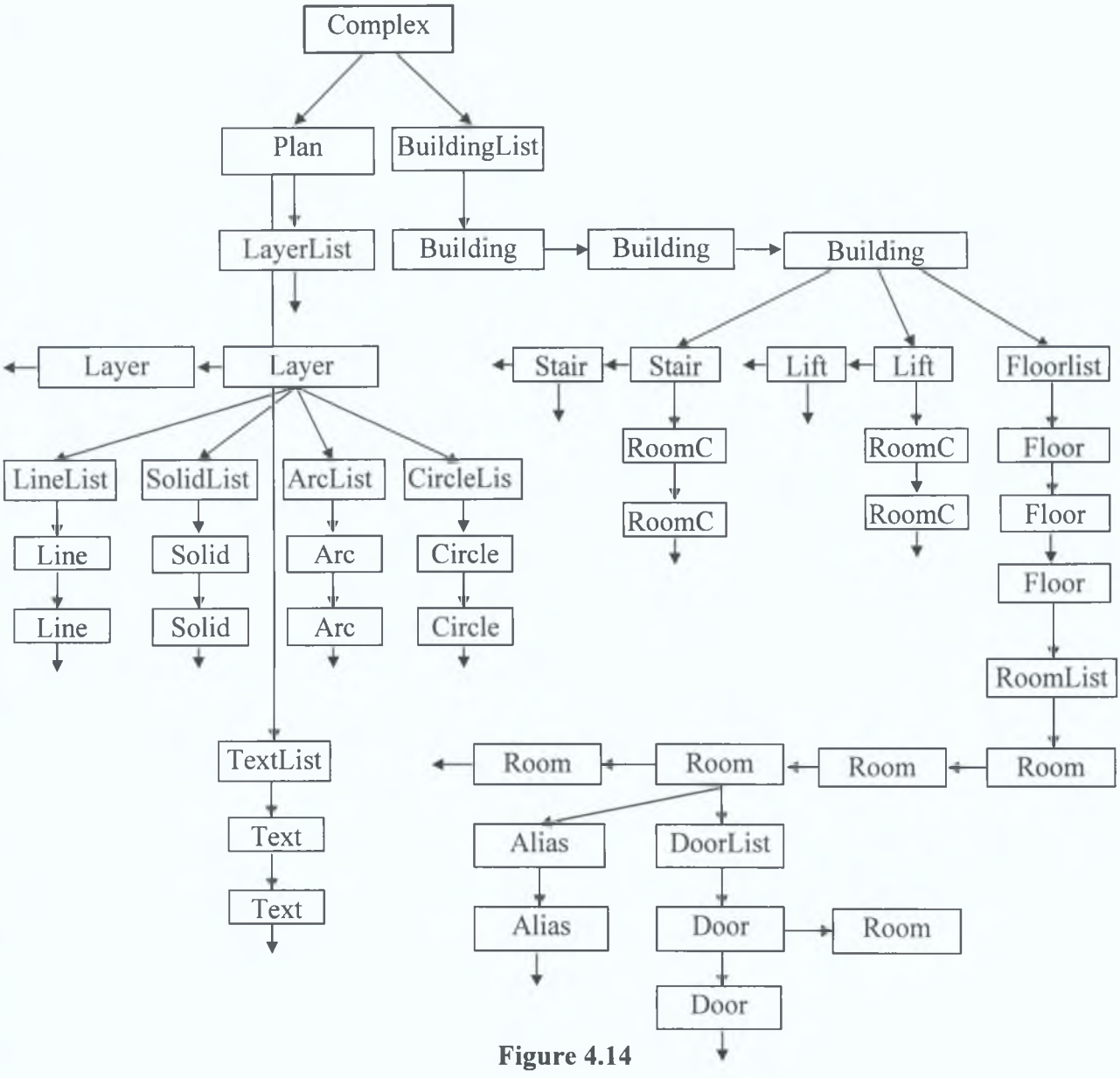

Figure 4.14

4.6. Overall Structure

This section describes the basic structure of the program. Windows programming is not as straightforward as DOS programming. The program must comply with Windows protocols, such as calls to update the displayed information. The MS Visual C++ package provides a basic structure for a Windows application and takes care of basic window functions and message passing. This structure, or skeleton application,

is based on the Microsoft Foundation Classes (MFC) [18] When starting a new application, the user is provided with an Application file, a Frame Window file, a Document file a View file, as well as resource and project files A brief introduction to the purposes of each of these files follows

4.6.1. The Application File

This file contains a class derived from the MFC CWinApp class Each application can have only one CWinApp derived class This class is used to initialise and run an application It responsible for such tasks as setting background colours, registering the different types of Documents available (see 4 6 3), as well as creating and displaying initial Frame windows

4.6.2. The Frame Window File

The frame window 'frames' a view It can also contains tool and status bars It is the main application window Client windows are contained within the frame window

4.6.3. The Document and View classes

The Windows document is the part of a program responsible for the programs main data storage Each document can have one or more views associated with it, each view showing some or all of the documents data

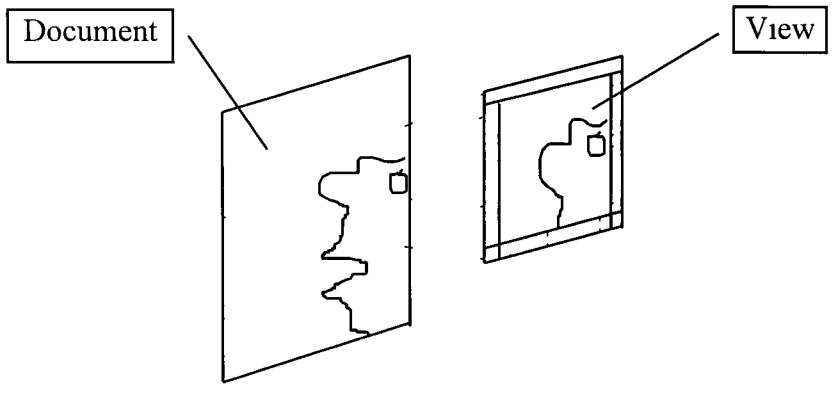

Figure 4.15

Each Windows application can have a number of different document types, each designed for a specific purpose The different documents with their associated views can have their own menus, giving the user the opportunity to call the functions and

menus appropriate to the document Three document types will be used in this application, the DXF, Floor and Complex documents

4.6.4. The DXF and Floor documents

These two documents have similar functions The DXF document's primary purpose is to store and maintain data from DXF files, while the Floor document deals with FLR files The menus associated with both documents are the same The only difference between the two documents is that the DXF document will not allow the file to be saved in DXF format, but only in FLR form The DXF document will thus mainly be used to take in drawings from a database, where the user can edit them as appropriate and then save them in FLR form The Floor document is thus used with files prepared by the DXF document Both document types' primary storage is a Floor object The DXF document is defined by the CDXFDoc class and the Floor document by the CFloorDoc class

The main functions associated with these documents are

- create, edit or delete drawing objects or layers
- format the drawing colours and layers
- set user options for building the RoomList tree
- build and edit the tree* (Section 4 9)
- display the tree
- identify the object under the mouse* (Section 4 9 1)
- locate a user specified object* (Section 4 8)
- calculate the routes between two mouse specified points* (Section 4 10)
- calculate the routes between two rooms* (Section 4 10)

The functions with an asterix are functions are member function of the Floor class

4 7. The Complex Document (CComplexDoc)

The complex document stores a Complex object All the data needed to describe a complex of buildings can thus be stored in this document When first loaded, a complex document will display the Plan object associated with the document's Complex object The document contains most of the functionality of the DXF and floor documents with the notable exception of the edit features Upon creation, the complex editor is automatically displayed This editor allows a Complex object to be

created A plan drawing, buildings and their associated floors can be attached to the Complex Special tools in the editor allow buildings to be located on the plan drawing and stored in memory List of stairs and lifts connecting the floors in a building may automatically be compiled after the creation of the Complex Another function of the complex document is to map the entry and exit points of a building onto the plan drawing (see Chapter 5 for user instructions)

The complex document can be used to display any floor within the complex and may also be used to calculate routes from any point within the complex to another Section 4 10 will deal with the specifics of the routing algorithm For now, it is enough to know that a routing algorithm is associated with each Floor object The return value for calling these functions is a list of lines representing the route from any point on that floor to another It is thus the job of the complex document to calculate which floors (and/or plan) need to be accessed for a particular route, switching to or creating the views for these floors, collecting (and scaling) all the returned path values, deciding which paths need to be displayed on which floors and finally calling on all the floors involved to display these paths

When a user tells the program that he or she wants the route between, say, a room on the first floor of one building to a room on the ground floor of another building, the following steps are necessary to calculate this route

- locate the origin and destination buildings and floors
- Since two separate buildings are involved, the route is split up into three parts,
  - from the origin to the outside
  - from outside building one to outside building two
  - from outside building two to the destination
- In order to access the outside from the origin, the ground floor needs to be accessed The route for the origin building is therefore further divided down into the following parts,
  - from the origin to all the stairs and lifts leading to the ground floor,
  - from all the stairs and lifts on the ground floor that have access to the first floor to all the exits on the ground floor

- All open views must now be checked to see if any of the required floors are already being displayed If any of the required floor views are found, they are instructed as to what path needs to be calculated
- Any floors not already being displayed must have views created for them must be informed of the required path calculations
- The plan view must calculate the routes between the two buildings
- Once all the route calculations are complete, all the route data is collected from the appropriate views The data collected from each view is scaled using the view's drawing scale Individual routes may now be compared regardless of which view created them
- The overall routes are constructed and the one best fitting the user output type specifications is selected
- Each view is now informed which of their calculated routes they must display

4.8. Identifying and Locating objects on a drawing

This is one of the simplest yet most essential functions of the program If the program cannot correctly identify or locate a feature or area it cannot provide the user with information about it, similarly it cannot calculate a route from A to B if these points cannot be found

The identification of areas such as rooms is based purely on the text provided with the text layer of the drawing Every drawing should contain textual information about the areas shown, split into four categories as show below,

Category Number	Category
1	number
2	description
3	Assigned department
4	Area

Figure 4.16

Category 2 would consist of a description of the function of the area or room, for example OFF-AD could mean that that particular room is an OFFice used for Administration This type of textual description is very common, however allowances have been made for different nomenclature systems Rooms can have similar functions and belong to the same departments, however each room must have a unique number The accepted standard for numbering is that each room is assigned a number that starts with the number of the floor they are on The ground floor rooms are prefixed by the letter 'g' This floor number or letter is stored in a Floor class data member called roomlabel

The function Find is a member function of the Floor class Its declaration is as follows

coords Floor Find(resize, char*, CDC*, int),

'coord' is a simple data structure that contains an x and a y co-ordinate The textual description of the object to find is pointed at by the char* If the object is found, it is highlighted on the screen with the colour specified by the int and its co-ordinates are returned The resize parameter is a structure containing information about the current zoom status of the drawing This information is necessary since co-ordinates of objects change (relative to the screen) for different magnification factors The CDC* parameter is a DeviceContext class, necessary for all graphical functions

The implementation of the algorithm is straightforward All the available text is simply scanned for the given description If a match is found, the next task to be undertaken is to ascertain if the text is written directly in the area it describes or if it has an associated arrow or line pointing to the area in question Any such arrows should be contained on the textlayer of the drawing The immediate vicinity of the text in question is scanned to see if any lines originate there If a line is found, it is followed and the co-ordinates are deemed to be at the end of this line If no lines are found, the co-ordinates are simply given as the co-ordinates of the text

All items matching the description are highlighted Only the co-ordinates of the last match are returned The function is also able to take wild cards, for example id_items("OFF*") would identify all the rooms used as offices on the screen (by highlighting them)

The Identify function is also a member of the Floor class Its declaration is

CString Floor Identify(int, int, CClientDC*, resize),

The function returns a textual description of the area under the xy co-ordinates given by the two integer parameters The algorithm used is simply the reverse of the Find tool Firstly, the area is highlighted in the current fillcolour The screen colour given by the co-ordinates of each text group is subsequently checked Those text groups that point at or are in an area filled by the highlighted colour must describe that area Both of the above functions were thoroughly tested on a variety of drawings The results were satisfactory One error occurred where an arrow did not terminate in the area it described but merely pointed at it This situation can only be rectified by editing the drawing to extend the arrow to terminate in the correct area Two errors occurred when arrows did not originate close enough to its associated text This led to experimentation with the area around a text group that can be termed as 'immediate vicinity' If this range is extended too far, arrows originating in nearby text groups may be mistaken as originating from the current text group The original choice (12 pixels with zoom factor 1) was found to be the best When creating or editing the drawings, care must thus be taken to position any arrows within a reasonable distance to the text and to terminate the arrows in the area in question It must be pointed out that the error rate was very low given that the over all number of tests conducted was over 250

4.9 Automatic Construction of a RoomList

For even a medium is sized building, the amount of rooms in a building can be quite large The full RoomList can be thus be very complex The Floor class contains a tool that automatically builds up its associated RoomList Its prototype is given by

void Floor IdRooms(resize, CDC*, BOOL, BOOL),

The two Boolean variables are option parameters, the first tells the algorithm whether to automatically lock double doors (see Section 4 9 6) and the second indicates whether the aliasing function should be activated or not (see Section 4 9 4)

4.9.1. The IdRooms algorithm

In Section 4 8 we saw how each room in a building is has a textual description associated with it The IdRooms algorithm starts by calling up each text group in turn If the text describes a room (as determined by the roomlabel, user definable include and ignore strings and a check to see if the co-ordinates of the text lies within the walls of the building), a Room object is constructed for it This object is constructed with the room number, description and x,y co-ordinates as a parameter Each Room contains a DoorList Constructing this is the next step undertaken

Firstly, the Find( ) function is called with the Room co-ordinates as parameters This highlights the room in question with the current fill colour Next the ID_Doors function is called The principle behind this function is that doors are represented on drawings by arcs By checking which arcs are affected by the fillcolour, the doors associated with a room can be identified It is not enough to simple know whether a door is associated with a particular room however, since each Door object also stores the positions of the co-ordinates of the insides and outsides of the doors In order to find these co-ordinates, the call to IdRooms( ) is made immediately after instructing the View class to draw the floor with the doors closed, as shown in the figure below

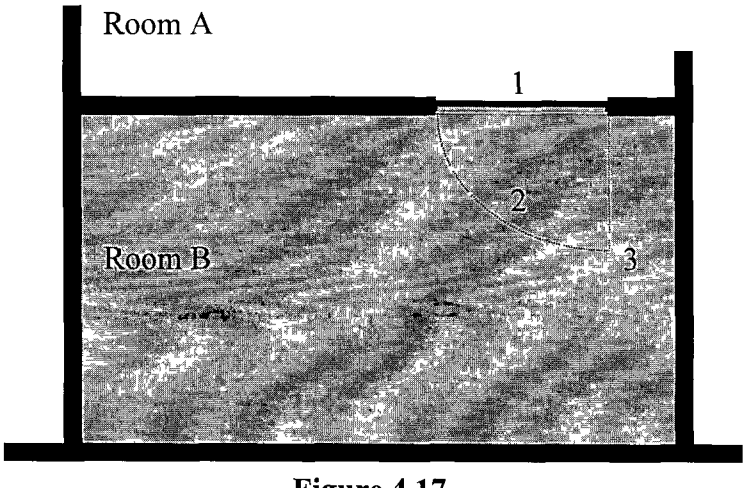

Figure 4.17

This allows the ID_Doors( ) function to determine the inside and outside co-ordinates of the door by taking test points at the three positions shown In this example, test points 2 and 3 are shaded, which tells us that the vertical arm of the arc represents the open door and the horizontal arm the closed door The remaining test point must

therefore be on the outside of the door In this manner the complete DoorList can be constructed The algorithm then continues adding Room objects to the RoomList Once this task is completed, we have a complete list of available rooms and doors, however, we still do not know where each door from a given room leads

Consider the example above, we have two Rooms, A and B Say each of these Rooms has one Door object associated with it The only difference between these objects is the inside, outside set of co-ordinates (and perhaps the Lock status) Both Door objects are based on the same arc The algorithm thus scans the RoomList and compares all the Door objects in each Room to the others When two Door objects are found to share the same arc, they are each assigned pointers to the Rooms to which they lead (i e in the example, a pointer to Room B would be added to the Door in Room A and vice versa)

An important aspect has not been considered in the above explanation however, that is doors that lead to the outside of the building In fact, the entire outside has been ignored

4.9.2. The Outside

Each Room object in the RoomList represents a dimensioned space on a drawing There is no reason why this concept cannot be extended to let a Room object represent the outside The outside has access to rooms inside in a similar way to which a room inside has access to other rooms in the building, i e through a series of doors A Room object called Outside can therefore be created with a DoorList containing all the doors which have access to the inside of the building It is not quite as easy to assign specific co-ordinates to the outside In fact, the co-ordinates are set at (-1,-1) for the outside Room since the actual co-ordinates are never explicitly required Each ground floor is assigned a single Room representing the outside which is linked into the RoomList in exactly the same manner as all other Room objects

4.9.3 The Sealed Room

After all the Room objects have been linked together, a verification check of all the Doors are made If any Doors do not contain Room pointers, i e if they do not lead anywhere, an error of some sort has occurred Consider the example given below (taken from an actual drawing)

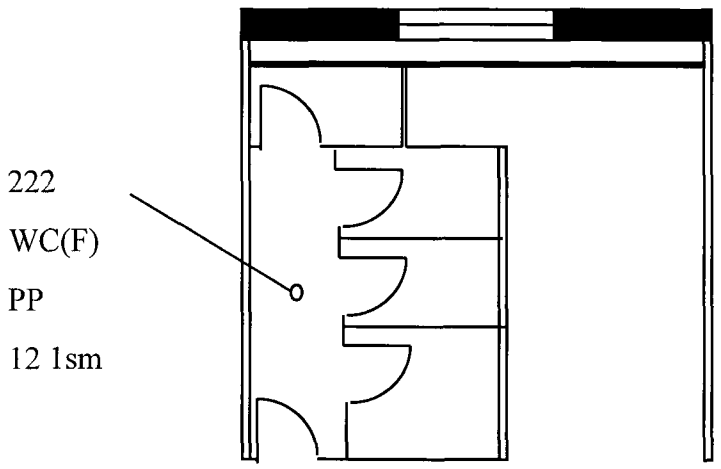

Figure 4.18

The section shown above shows a typical ladies toilet in a public building Only one room is identified, yet this room leads to four other rooms A Room object representing room 222 will thus be added to the Floors RoomList This object will have five doors, but only four will contain pointers to other rooms It is unreasonable to expect each toilet cubicle to be individually named, labelled and added to the RoomList Instead, a Room object labelled as Sealed is created This object is the default Room to which all unconnected Doors are pointed to The Room is labelled as Sealed because it is locked In other words, it cannot be used as a throughway when calculating a route from A to B No fixed co-ordinates are provided as the sealed Room is never accessed directly A user cannot ask to be directed to a location accessible only through a Door object pointing to the sealed Room since these locations are not uniquely identified Other examples of where Door objects with sealed Room pointers would occur are walk-in wardrobes and unlabeled storage rooms

4.9.4. Aliasing

Aliasing occurs when a room is not uniquely identified but can be associated with several names The drawing below is taken from an actual floor drawing

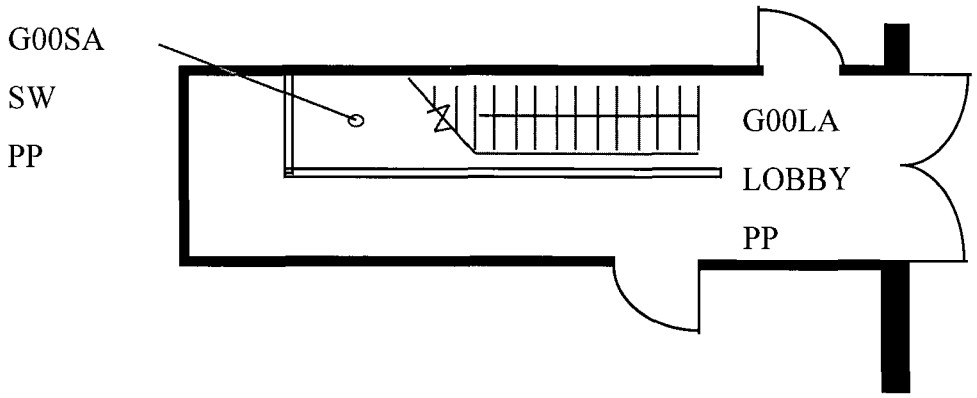

Figure 4.19

It can be seen that both G00SA and G00LA are connected without any doors This situation can also occur in large rooms such as laboratories where different sections may be individually labelled The problem that arises due to this is that the IdRooms algorithm as it stands would create two separate, yet completely identical Room objects (except for name) for both areas This leads to many complications The most significant problem would occur with the route finding algorithms, which would attempt to create routes through both of these Rooms despite the fact that they are the same Further complications occur with Door objects leading to the area The Door class only contains one Room pointer, not two

If we refer back to Section 554 5 7, we can see that the Room class contains a list of objects called Aliases The Alias class is the solution to the problem outlined above If the aliasing option is selected, the IdRooms( ) algorithm will not create two separate Rooms for areas not uniquely named, but will create one Room with an Alias list After each Room object created by the algorithm, the FindAliases( ) function is called This function is similar to the Identify( ) function in operation and scans all text groups for text positioned in the highlighted area Checking for potential aliases is computationally intensive, the more rooms associated with a floor, the longer the full IdRooms( ) algorithm will require to run For this reason, the user may specify not to check for aliases

4.9.5. Virtual Doors

Virtual doors are used to give access to an area that does not have access in the normal way As before, a description of the problem to be discussed can best be accompanied by a drawing

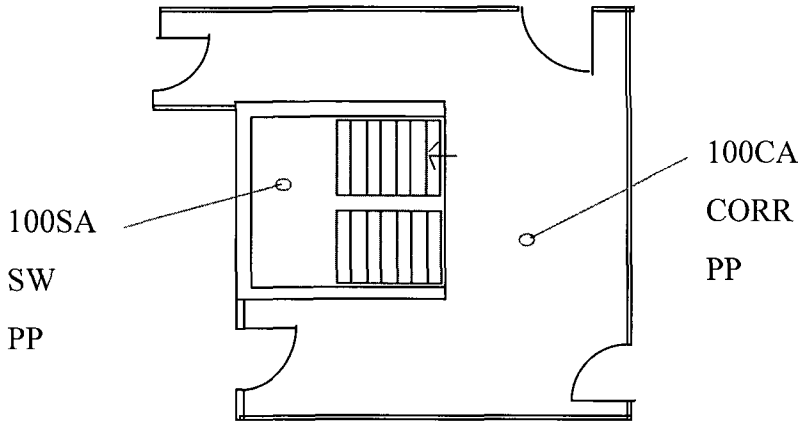

Figure 4.20

The drawing shows a corridor (100CA) connected to a stairwell (100SA) No doors have been specified but a line has been drawn over the mouth of the stairwell The IdRooms( ) algorithm will not be able to associate the stairwell with the corridor in this case The virtual door is used to overcome this problem A virtual Door is a user created Door object that does not have a real world representative It does not appear on the drawing and is used only by the model to identify which rooms are connected to each other

4.9.6. Locking double doors

Many public areas have access through one or more sets of double or swinging doors Such a set of doors will be treated as two separate doors in the model, each leading to the same area This is not a problem in itself, but causes the route finding algorithms to calculate separate routes for the individual doors where one route may be sufficient An option exists in the IdRooms( ) function that recognises when double doors occur and automatically locks one of these doors Doing this prevents the route finding algorithms from using one of the doors in a set of double doors and can speed up the overall routing process considerably

4.10 The Route Finding Functions

Route finding is by far the most complex task required In order to produce a step by step route from a room in a building to another room in a different building a huge amount of calculations are required It must be kept in mind that in order to calculate say, the shortest route, all the other routes need to be calculated for comparison Due to the complexity of the problem, the route finding tools have many private helper functions Both the Floor and Plan classes contain route finding tools The tools in the Floor class are primarily for calculating routes between the rooms of a particular floor, while the tools in the Plan class deal with routes between buildings See Section 4 5 for details about the route finding management functions The prototypes defined in the Floor class are as follows (see Appendix D for listings)

virtual RouteList get_path_from (char *, char *),

virtual DataList Move_Along(Routelist*,Linelistlist*,resize,float,BOOL, CClientDC*),

The first function calculates a RouteList (see Section 4 10 1) from the origin (as given by the first char *) to the destination (given by the second char *) The second function can take the returned RouteList and use it to calculate paths of higher accuracy than described by the RouteList Both of these functions are overridden in the Plan class

4.10.1. The RouteList

The RouteList is used exclusively to store a list or tree of Door and Room objects which can be used to travel from one point to another on a given floor Such a list can consist of no more than say, an origin Room, a Door and a destination Room In most cases however, the routes are not quite as simple The elements contained within the RouteList are of the Route class Each Route contains a pointer to a Door and a Room as well as containing a RouteList of its own, resulting in the structure shown in Figure 4 21 below

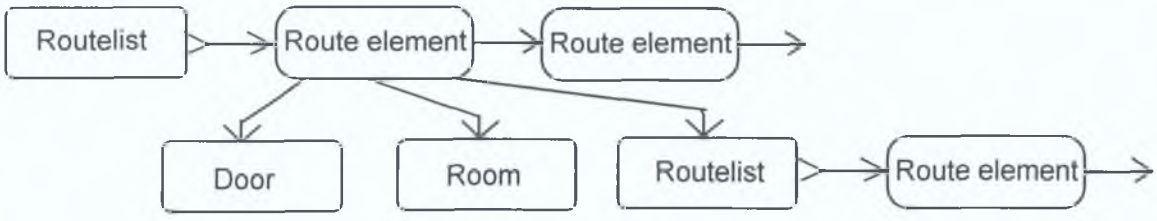

Figure 4.21

Each RouteList can thus contain any number of Route elements and each Route element in turn can contain any number of Route elements through its own RouteList. It is thus possible to construct a multi-branched tree using the Route and RouteList classes. Each branch of the tree can split up into any number of other branches which can split up in turn as shown in the diagram below.

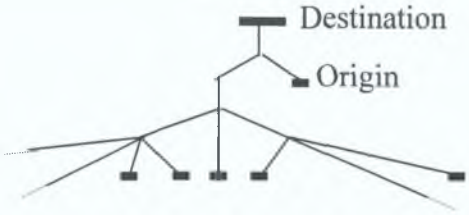

Figure 4.22

At the top of the tree is the destination, while branch extremities represent the origin. The branches connecting the origin and destination represent all the possible routes that can be taken when travelling from one to another.

4.10.2. Constructing the RouteList (Floor class only)

A RouteList from one point to another is constructed by calling the get_path_from(char *origin, char *destination) tool. The first task undertaken by this function is the identification of the areas described by the origin and destination strings, which are passed in as parameters. This is done simply by scanning the RoomList until Room objects with names or numbers matching the origin and destination strings are found. The challenge now is to find all the Room and Door objects that will lead us from the origin Room to the destination Room. The basic principle behind the algorithm is to check all the doors in a room to see if they lead to the destination. All the Door and Room elements that are found to lead to the destination are added to the RouteList. Every unlocked door that does not lead to the

destination is 'entered' and the algorithm called recursively This process is repeated until all possibilities are explored

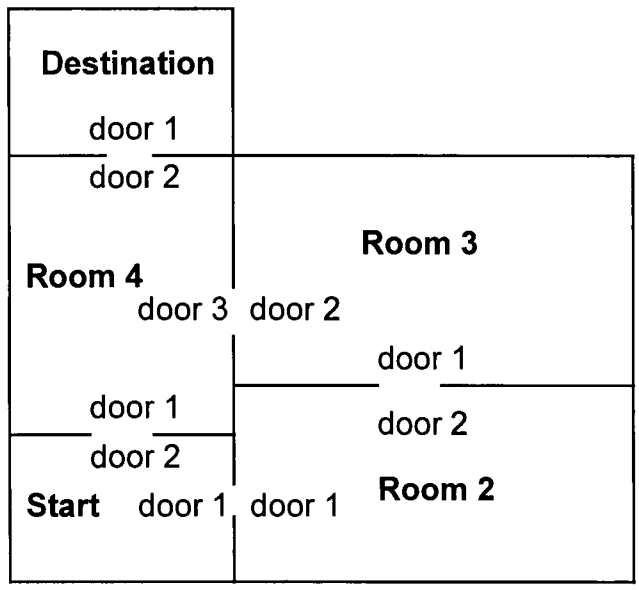

Figure 4.23

Consider the above simplified drawing of a building In order to travel from the start room to the destination room, the get_path_from("start","destination") function would operate as follows,

Step 1

The start and destination rooms are identified from the RoomList,

Step 2

Each Room object contains a special flag, called the 'checked' flag This flag is set in the origin and destination Rooms This is done to prevent the search algorithm from searching rooms that have already been explored

Step 3

Doors 1 and 2 in room Start are checked to see if they lead to the destination room

Step 4

Since neither door leads to the destination, get_path_from("Room2","destination") is called Both of Room 2's doors are checked for the destination The room leading from door 2 is

entered since the checked flag in the room leading from the first door is set to 1

Step 5

The Room 2 checked flag is set to 1 and build_path("Room 3","destination") is called which after the appropriate checks sets the checked flag in Room 3 to 1 and calls build_path("Room 4","destination")

Step 6

Door 2 in Room 4 leads to the destination The destination room and door 2 are added to the RouteList Door 3 is subsequently checked to see if that leads to an unexplored room or also to the destination The checked flag in this room is now set to 2, indicating that this room leads either directly or indirectly to the destination A pointer to the current RouteList element is also stored in the room The reason for this is that if the room is again entered, the algorithm will be able to join the RouteList at this point without having to actually search for the destination again

Step 7

Control is handed back to the build_path("Room 3","destination") call Since the search has been successful, Room 3 and door 2 are added to the RouteList The checked flag is set to 2 and the current RouteList element pointer is set for Room 3 as was done for Room 4

Step 8

Room 2 is re-entered where the checked flag and current RouteList element pointer are set as for Rooms 3 and 4 The RouteList is updated and control is returned to build_path("Start","destination")

Step 9

The RouteList is updated to contain door 1

Step 10

The second door of the Start room is checked to see if it is unexplored The checked flag is seen to be 2, which implies that this room leads to the destination The second door is therefore also added to the RouteList *at the point pointed to by the RouteList element pointer contained in the room leading from door 2*

Step 11

The start room is added to the RouteList and the completed list is returned

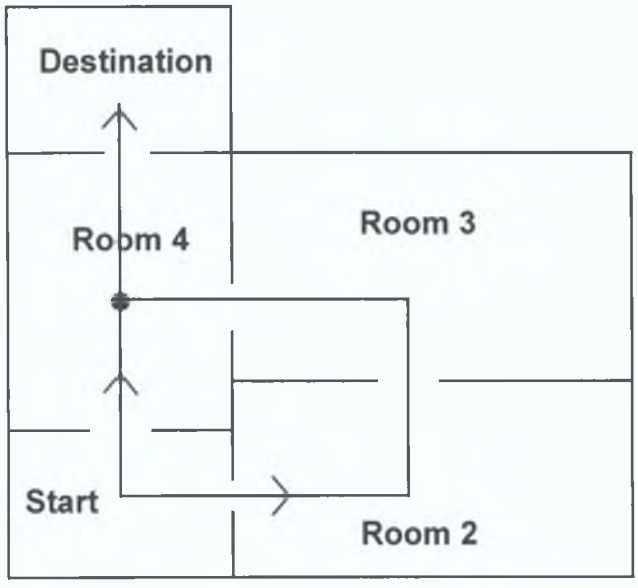

Figure 4.24

The above diagram shows the routes established. The point at which the two routes join up is shown in Room 4.

It must be pointed out that to get from the inside of a building to the outside or vice versa, a RouteList can be build up in exactly the same way. Remember that the outside is simply looked at as another Room object and can be dealt with in exactly the same way. The finished RouteList can provide us with basic instructions when attempting to get from A to B. The algorithm takes a negligible amount of time to produce results even for large RoomList and is thus highly efficient. For more detailed navigational information however, the Move_Along( ) function described in Section 4.10.3 below must be used.

4.10.3. Calculating step by step navigational information

Calculating step by step navigational information is very difficult. Consider what is required. Given any two points in a complex of buildings, calculate *all* the possible routes between these two points, avoiding all obstacles in the way. Looking at the overall task, it would seem close to impossible. However, we have already made considerable progress in reducing the overall problem to a series of smaller problems. Instead of attempting to calculate the overall routes, we can calculate all the routes inside separately to the outside routes. Inside the building we can calculate the routes

on each floor separately, again simplifying the problem By using the get_path_from() function we can quickly build up an overall picture of potential routes It is a logical progression to calculate the step by step information on the smallest possible scale available to us, that is, instead of graphically calculating step by step information for the overall A to B route, we can simply calculate step by step information for each Room in the RouteList and join all the results together
It is the task of the Move_Along( ) function to extract useful information from the RouteList and to call the appropriate functions for calculating the step by step routes The prototype for Move_Along( ) is as follows

DataList Move_Along(RouteList *, Linelisthst *, resize, float, int, CClientDC *dc)

The return value is a DataList A DataList is used to extract useful information from the RouteList The DataList can be used to construct a simple room to room route that has no obstacle avoidance information The Lmelisthst is a list of Linelists and contains a full list of step by step paths They are calculated with a user defined offset (given by the float parameter) and using either Algorithm A or B (given by the int parameter) These will be explained in Section 4 10 3 2 and 4 10 3 2 1 Let us first take a closer look to see how the DataList is constructed

4.10.3.1. The DataList

This MakeDataList( ) function called by Move_Along( ) takes in the RouteList created previously The RouteList is processed in order to discard unnecessary information All the rooms (with the exception of the start and destination rooms) are not needed for plotting the actual routes within each of the rooms, it is the position of the relevant doors within these rooms that is required A DataList is thus constructed with each element in the list representing the entry and exit door for each room in the RouteList If all the elements in this list were drawn, a rudimentary path from A to B would be produced, however no regard would be given to any potential obstacles in any of the rooms For the above example, the DataList would look like the figure given below In this simple case, the DataList would return the same paths as the Lmelistlist Note that each node has an assigned number These node numbers are

also stored in the DataList Each line segment can be uniquely described by the node it originates from and the node it leads to

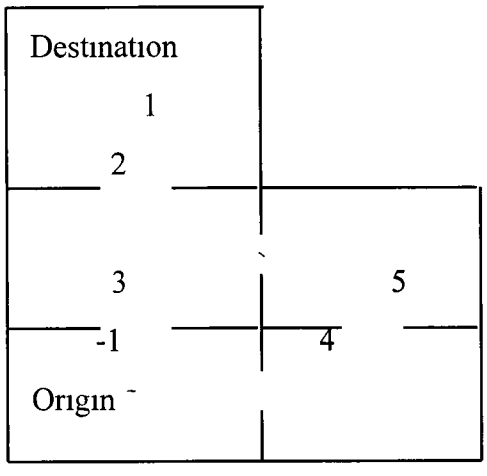

Figure 4.25

The construction of the DataList is not overly complex The algorithm starts by taking the first element in the RouteList If the RouteList in this contains elements, it calls itself until a Route element is found with no elements in its RouteList At this stage, it starts adding elements to the DataList If a Room is passed through (i e all the rooms with the exception of the start and end rooms), the co-ordinates of the entrance Doors are stored along with the co-ordinates of the exit Doors in Data members The actual co-ordinates of the Room are of no concern Node numbers are assigned as the list is being constructed

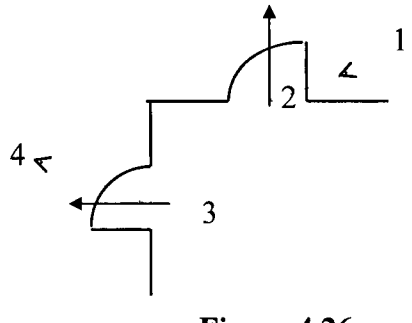

Figure 4.26

In the diagram above we can see a DataList with 4 nodes crossing a room from door to door The arrows at the door indicate the direction to the outside of the room This information is also stored in the DataList It is used by Algorithm A to calculate the

best direction of travel. The finished DataList may be returned if that is all that is required or it may be used by Algorithms A or B to construct an even more detailed output. Each element in the DataList is taken in turn and the appropriate call is made, depending on whether Algorithm A or B is to be used.

4.10.3.2. The Search Algorithms

Several different types of algorithm were experimented with. Two were eventually selected for implementation in the program. A brief description of the algorithms used is given below.

4.10.3.2.1. Algorithm 1

This algorithm is in essence a trial and error technique. It attempts to travel in a straight line from its origin to its destination. When an obstacle is encountered, the algorithm must make a decision on which direction to take in order to negotiate the obstacle. This decision is based on the direction of the destination with respect to the current position. Consider the simple room scenario depicted below.

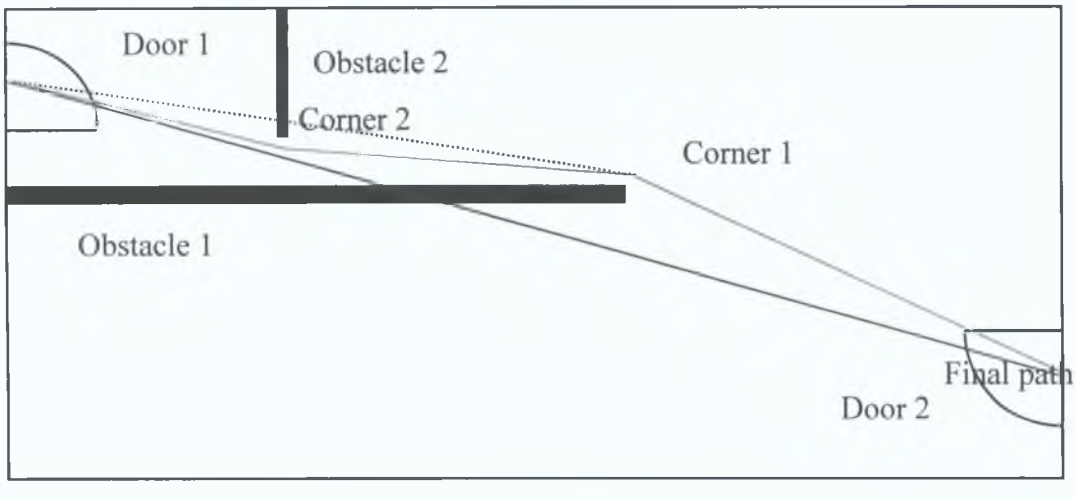

Figure 5.27

Suppose the algorithm is required to travel from Door 1 to Door 2. The diagonal straight line represents the most direct route possible. The algorithm will attempt to travel along this line until it hits the obstacle. At this point, it must make a decision as to how to negotiate the obstacle. The destination, Door 2, is below and to the right. Based on this, a path around the obstacle is looked for to the right of the point where the direct path hits the obstacle. At the point labelled Corner 1, the obstacle

disappears. The path is now split into two parts, from Door 1 to Corner 1 and from Corner 1 to Door 2. Both of these sections must now be checked for obstacles. The algorithm is consequently called twice recursively, once to calculate the path from Door 1 to Corner 1 and once for Corner 1 to Door 2. The former call will see the path split up once again when it encounters the second obstacle. The second recursive calls will thus check the paths from Door 1 to Corner 2 and Corner 2 to Corner 1 for obstacles. The final path is solid line shown avoiding the obstacles.

While this algorithm is fast, it cannot deal with complex environments. Consider the diagram below.

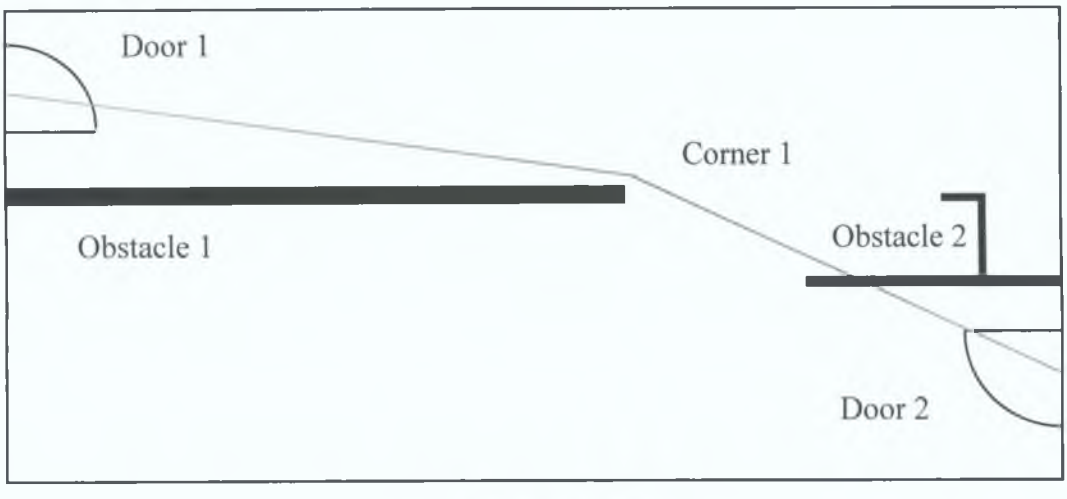

Figure 4.28

This environment is not much more complex than the first one. However, consider what would happen when the path between Corner 1 and Door 2 is checked for obstacles. At this point, the first real problem is encountered. The conditions under which the obstacle is hit are the same as for the previous obstacle, i.e. the destination is the bottom right, therefore the algorithm moves right instead of left as it obviously should. Upon encountering the vertical part of the object, the algorithm can be called on to move up, left, up, right and subsequently continue on a direct path, however, it is evident that no matter what it does at this point, it will not reach the destination. It would be possible to move up along the wall and back-track left along the top wall, however, this would not result in anything positive. The error resulted when the algorithm travelled right instead of left at the second obstacle. One solution to this problem is to use recursion at every point where two direction are possible. If one

direction proves fruitless, the other direction is chosen The main problem with this approach is that it is quite difficult to detect at what point the search has proved fruitless The example above shows an outcrop on the lower obstacle, which can successfully be negotiated, however the wall is reached after this obstacle, not the destination The shape of the far wall and upper wall is similar to the shape of the obstacle, but it can not be negotiated This shows the difficulty in deciding at what point the search has proved fruitless

The only points at which the search can positively be said to have failed are the start point and any point in an area previously searched

It can be argued that for every obstacle encountered, all the possibilities should be explored in order to achieve an optimum result However, implementing this would exasperate the problem of identifying the point at which a search proves fruitless

4 10 3 2 2 Algorithm 2

This algorithm is based on Algorithm 1 One of the difficulties with improving Algorithm 1 was deciding on a point at which the search can be deemed to have failed If certain assumptions are made about the complexity of obstacles, situations may occur at which we can say that a fault has occurred, or that the search has failed Consider the obstacles shown below The arrows indicate the direction of travel

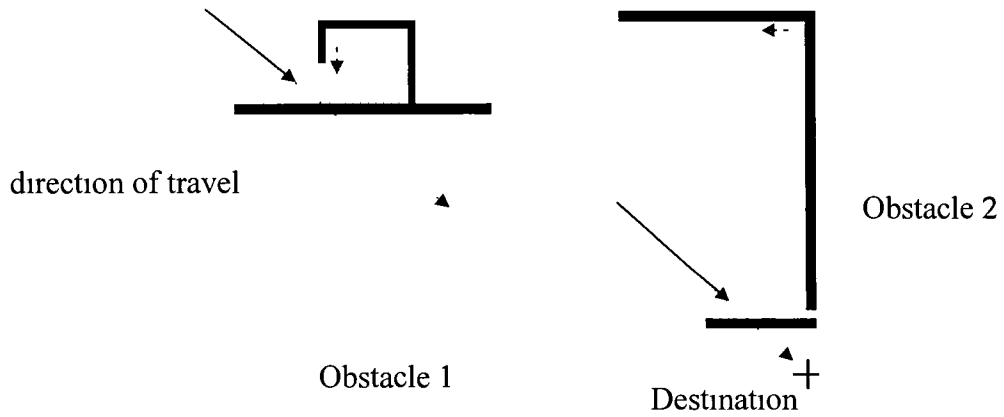

Figure 4.29

In the first scenario shown, the obstacle encountered comes upward and subsequently comes down again If we assume that an obstacle such as this cannot lead to the destination, we can determine that a fault occurs at the point where the obstacle turns down again Similarly for obstacle two we can determine a fault point In this case the destination co-ordinates has the same vertical component as the vertical part of the

obstacle At the top of the obstacle, the path is forced to turn away from the destination A fault may also be said to occur at this point Many other such scenarios may be formed to determine faults Note that errors occur for these obstacles only for the approximate directions of travel shown If this direction of travel is reversed, a fault need not occur Remember from Algorithm 1 that the path may get split up into various sections through recursive calls If any of these sections thus encounter a fault, the direction of travel may be reversed for that section and the fault may be bypassed Similarly if the overall path fails, the direction of travel may be reversed and a second attempt made to construct a path The room shown below can be used to demonstrate practical examples of some of the obstacles described above

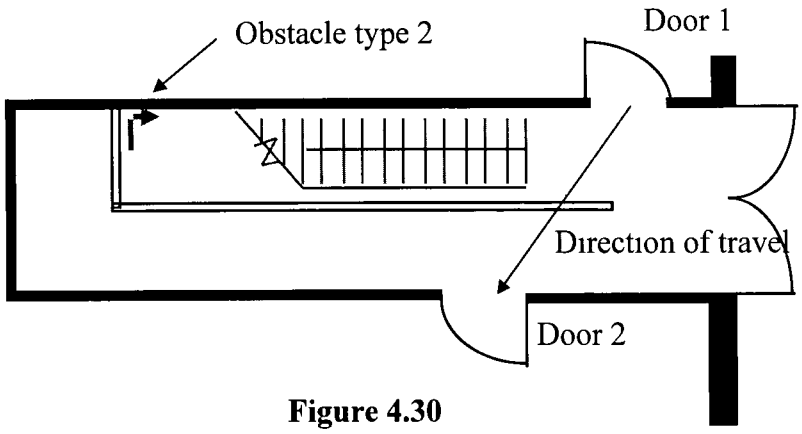

Figure 4.30

The original direction of travel is from Door 1 to Door 2 (the straight diagonal line) which hits the obstacle shown In an effort to avoid the obstacle, the path follows the dotted line, taking it past the destination At the point where the path is forced to turn back again (the solid arrow), an obstacle similar to the second obstacle shown above is deemed to have been met and a fault occurs Reversing the direction of travel will result in the scenario shown below

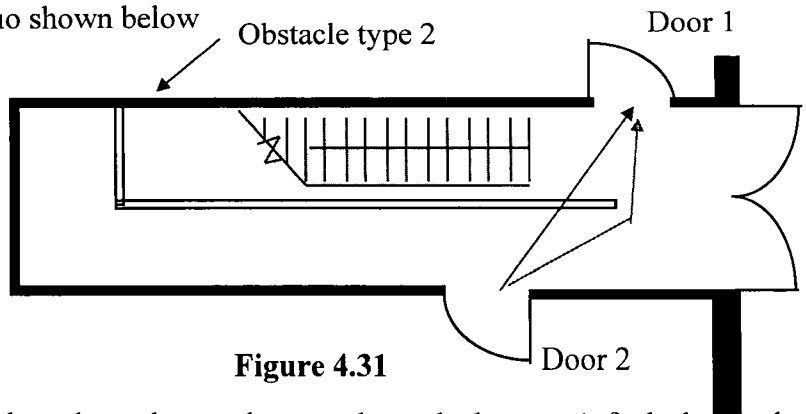

Figure 4.31

The obstacle is dealt with easily, resulting in the path shown A fault due to the first type of obstacle can also occur in this room

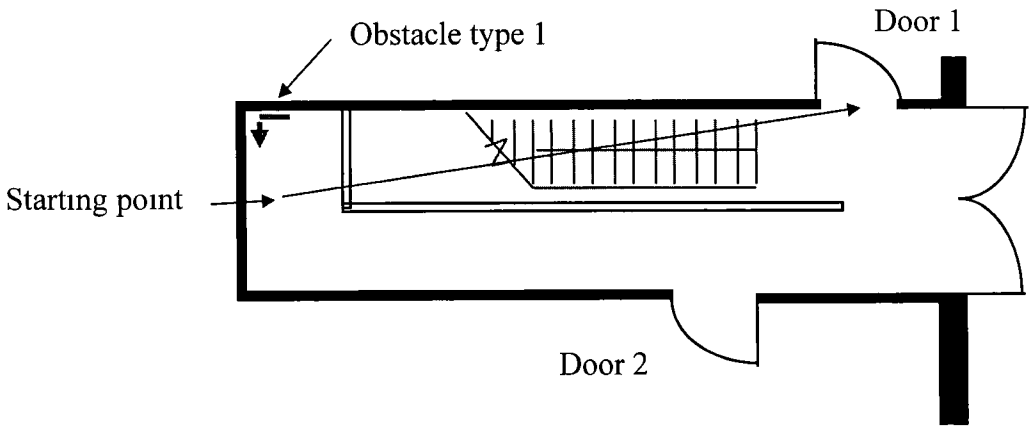

Figure 4.32

In Figure 4 32, the starting point is a given point in the room The obstacle forces the path up, left and down, at which point the fault occurs The direction of travel is reversed, resulting in the obstacle being dealt with as shown

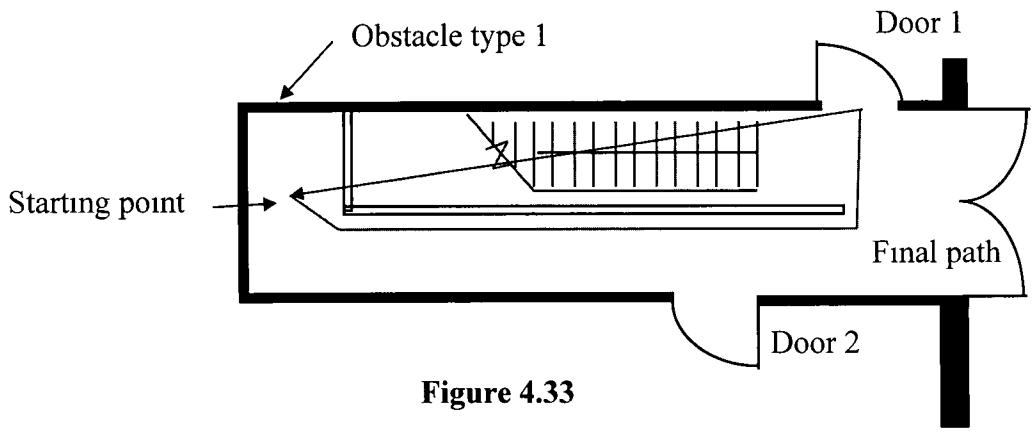

Figure 4.33

In the scenario shown in Figure 4 33, two obstacles are met The first one is where the direct diagonal line hits the vertical wall and forces the algorithm to negotiate the obstacle by going down and across At this point the path is split into two The first part is between Door 1 and the corner of the obstacle while the second part is between the corner and the starting point The second part encounters a further obstacle but this can easily be dealt with, resulting in the final path shown Further improvements may be made to the algorithm in an effort to predict the direction of travel considered most likely to yield results

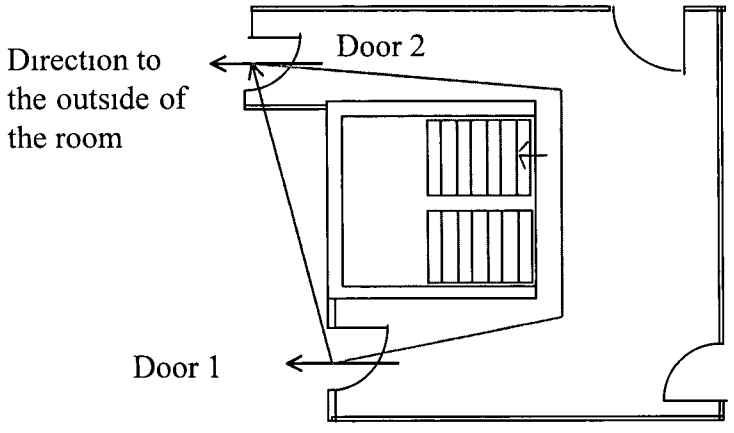

Figure 4.34

In Figure 4 34, the direction of travel is from Door 1 to Door 2 Remember that the DataList contains information about the direction to the outside for each door If we compare the direction of travel to Door 1's direction to the outside, we see that they are within 90^0 of each other In other words, the direction of travel is towards the outside Knowing this, we can expect to encounter obstacles the moment the algorithm attempts to start routing from Door 1 to Door 2 However if we reverse the direction of travel, this is not necessarily true Therefore, if we find the direction of travel to be in the same direction as the outside, we can reverse the direction of travel and prevent the routing algorithm from starting at an obstacle

To conclude, Algorithm 2 is quite effective in dealing with rooms that are not overly complex The vast majority of rooms will fall into this category The algorithm may be continuously improved by teaching it to recognise more fault situations However, the resulting path is not necessarily the optimum path and the algorithm is not guaranteed to be able to deal with all situations Despite this, it was implemented as Algorithm A Tests with the finished algorithm showed it to be fast and free of errors It must be pointed out though, that none of the test maps contained rooms of excessive complexity

4 10 3 2 3 Algorithm 3

In order to overcome the shortcomings of previous algorithms this algorithm was developed It is intended to be a robust routing algorithm able to deal with highly complex environments The main problem with the algorithms above is that they deal with obstacles as they meet them In other words, the algorithm has no knowledge of

the layout of the room through which the route is to be constructed This can lead the algorithm to explore sections of the room which can never lead to the required destination This algorithm is capable of learning about the layout of a room, and mathematically calculate a route through the room

The algorithm consist of three parts, listed below,

LineList Floor Traverse_RoomB(Data *lin, float offset, CClientDC *dc)

void Floor AlgorithmBX(Linehstlist *ret, Data *lin, float offset, CClientDC *dc)

void Floor AlgorithmBY(Linelistlist *ret, Data *lin, float offset, CClientDC *dc)

The first member function shown here is responsible for calling the other two functions when required, collecting their data and processing this data to produce the final LineList for the path

The Linelisthst parameter in the Algorithm B member functions is used to return data The offset parameter is used to determine both the distance that the eventual route will keep clear off corners and the minimum size of room sections to be explored The Data parameter is used to pass in start and end co-ordinate values to the algorithm and also contains directional information

In order to understand the Traverse_RoomB( ) algorithm it is easier to examine the other member functions first The workings of the AlgorithmBX can best be explained by means of example

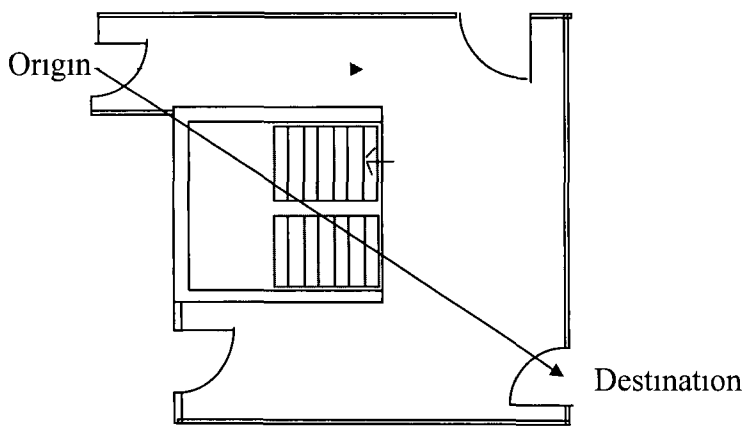

Figure 4.35

The diagonal line in Figure 4 35 represents the direct route from the origin to the destination The algorithms previously described would have attempted to follow this line until an obstacle is encountered This algorithm however, does not do that

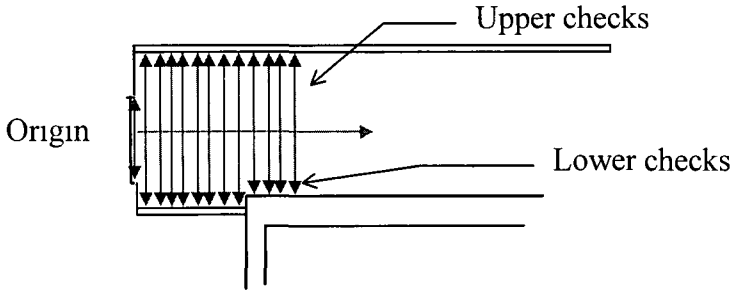

Figure 4.36

The algorithm starts by travelling in the **horizontal** direction of the destination (the horizontal arrow), checking above and below it as it travels along (represented by the vertical arrows in Figure 4 36) The upper and lower checks not only detect when an obstacle is hit, but can also detect whether they are travelling beside an obstacle or empty space The points at which the checks hit an obstacle is recorded in the Linelistlist parameter, while the wall detection is used to find new sections of the room as shown below

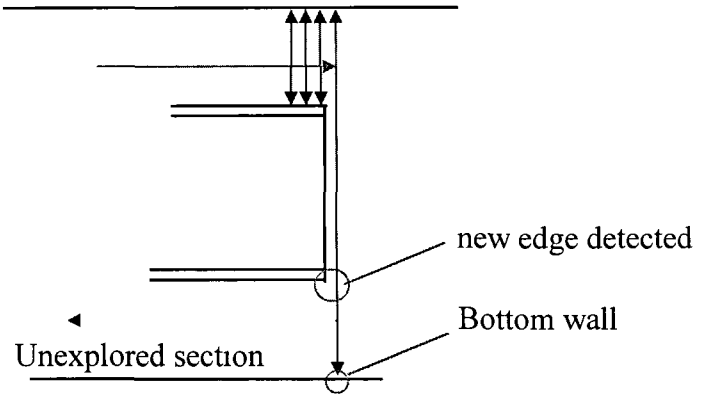

Figure 4.37

The last lower check in Figure 4 37 travels beside a wall for some distance before detecting the bottom wall Since the wall it travels beside does not continue all the way to the bottom wall, an unexplored section of the room lies to the left It can be argued that the destination lies to the right of the check, while the new section lies to the left therefore this new section need not be explored However this assumption is not true for all cases which means that exploration of this new section is required

Exploring new sections of a room can be done by calling the algorithm recursively The start point for this recursive call will be the point half way between the detected edge and the bottom wall There is one major difference between this call of the algorithm and the previous one That is that the destination must now lie in the opposite direction to the direction of travel The Data parameter passed into the algorithm contains information about start and end co-ordinates as well as information about the direction of the destination in relation to the start co-ordinates This directional information is adjusted for the above example so that the algorithm knows that the direction of exploration must be in the opposite direction to the direction of the destination

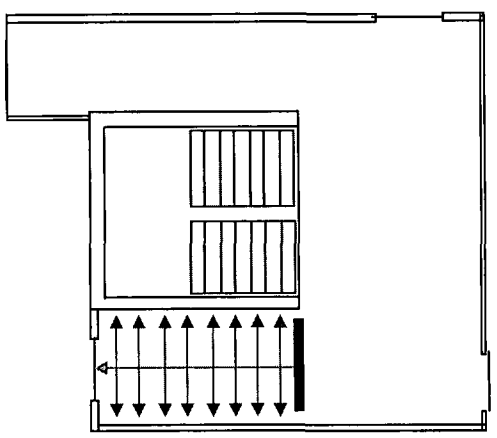

Figure 4.38

Figure 4 38 shows the recursive call exploring the bottom section of the room The solid vertical line shown is used to seal off the explored sections of the room from the unexplored sections Without this seal, it would be possible for the search to circle around indefinitely searching areas explored previously

When a search reaches a blind end as in Figure 4 38, the algorithm returns an empty LineListList to the function that called it That calling function now proceeds as before, travelling in a horizontal direction towards the destination When the right hand wall is met, the lower check detects the doorway to the right

Figure 4.39

This results in a second recursive call (Figure 4 39), which successfully finds the destination A Linelistlist is returned with information about the layout of the doorway This information is added to the information already gathered about the layout of the room and passed back to the calling Traverse_RoomB( ) function This Linelistlist thus consists of separate LineList, each containing information about a specific section of the room explored The Linelistlist for the above example would thus consist of only two LineList The first list would cover most of the room, while the second would cover the doorway leading to the destination If it were to be displayed, it would look like this

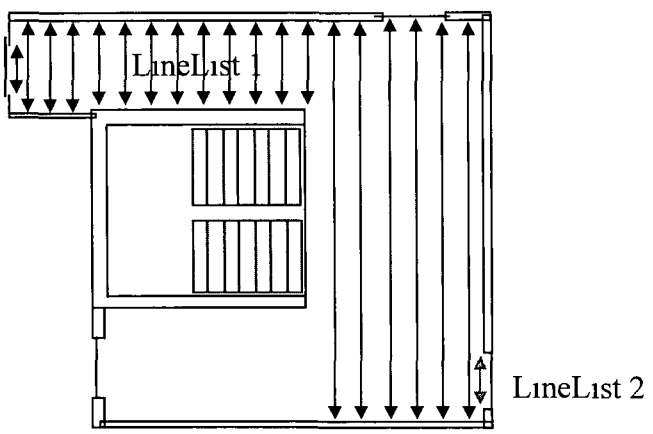

Figure 4.40

Note that the Linelistlist is never actually used for display purposes The only visible attributes of the search algorithm are the lines used to seal off explored sections before recursive calls

AlgorithBY( ) is virtually identical to AlgorithmBX( ) with the notable exception that the direction of travel is in the vertical direction The reason for the two versions will become clear in the next section which will discuss the Traverse_RoomB( ) algorithm The Linelistlist resulting from an AlgorithmBY( ) call for the above example would look like the figure below

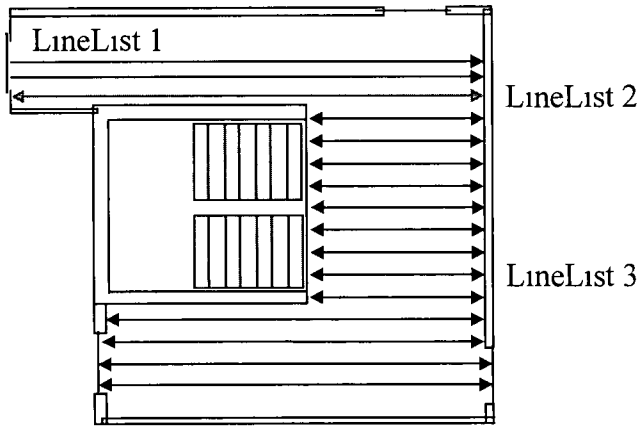

Figure 4.41

The Lmelistlist now contains three sections as opposed to only two for the algorighmBX( ) call Note also that the search started by travelling vertically down towards the destination This left the top section of the room completely unexplored If the search for the destination in the downward vertical direction were to have proved fruitless, the upper section would have been explored

4 10 3 2 4 The Traverse_RoomB( ) algorithm

This algorithm is responsible for calling both the X and Y versions of algorithm B and processing the data The result is a LineList which contains the path from origin to destination Both X and Y versions are processed in a similar way As before, the method used to process this data can best be explained by example Consider the room shown in the figure below

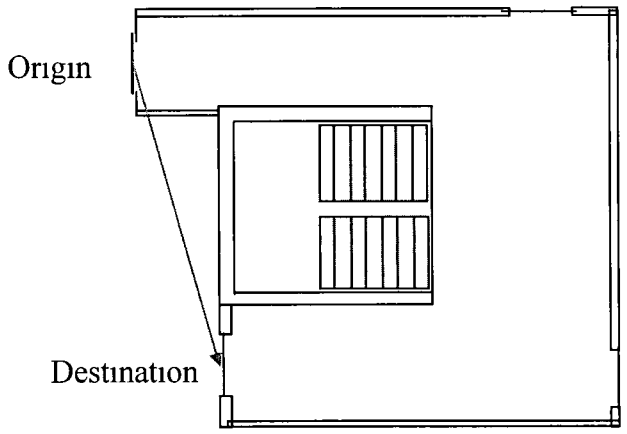

Figure 4.42

By visual inspection it can be seen that the path from the origin to the destination is blocked by an obstacle We can see the layout of the obstacle and with little conscious effort can see how the route must be altered to accommodate the obstacle An obstacle can be any shape or size, such as the one shown below This obstacle, though much more complex in shape, can still be easily negotiated with little thought How does the brain accomplish this?

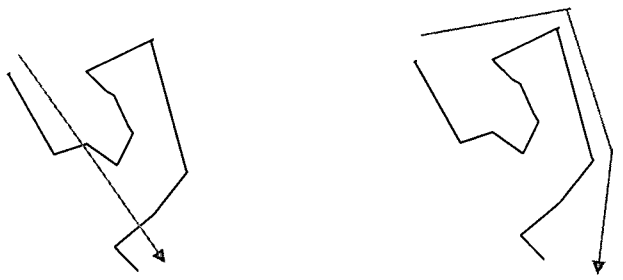

Figure 4.43

The obstacle shown above has many points and planes, most of which can be ignored when considering a path around the obstacle The only point that must always be dealt with, no matter what shape or size the obstacle, it the point that is farthest away from the direct route Identifying this point is therefore the most fundamental step in calculating a route around an obstacle

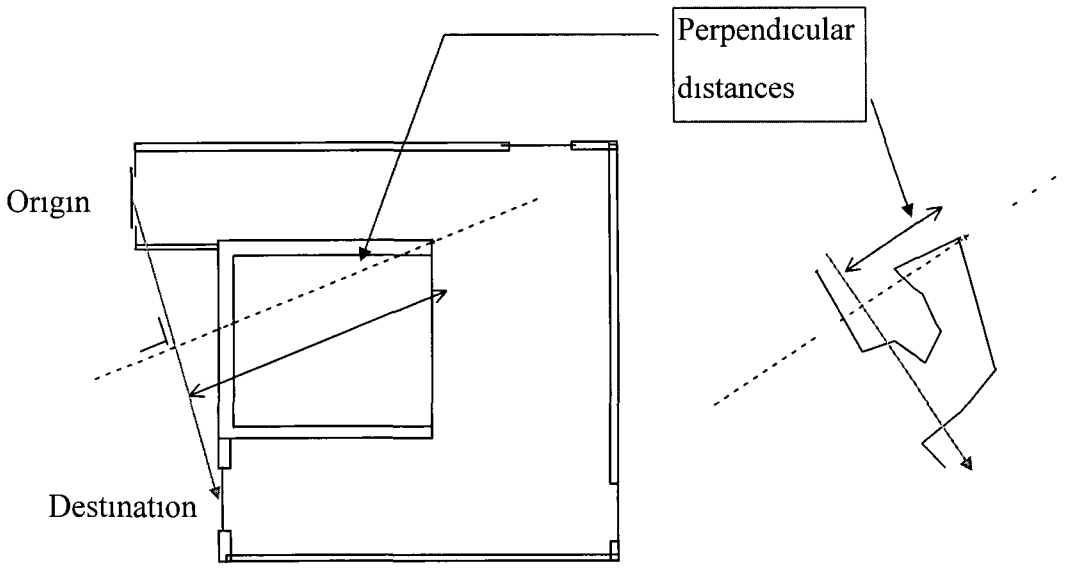

Figure 4.44

Once this point has been identified, the first step in plotting a path around the obstacle can be taken The path may now be split up into two, from the origin to this point on the obstacle, and from this point to the destination It is now a simple matter of repeating the above process of identifying the most distant point on the obstacles (if any) and splitting the path up into further sections

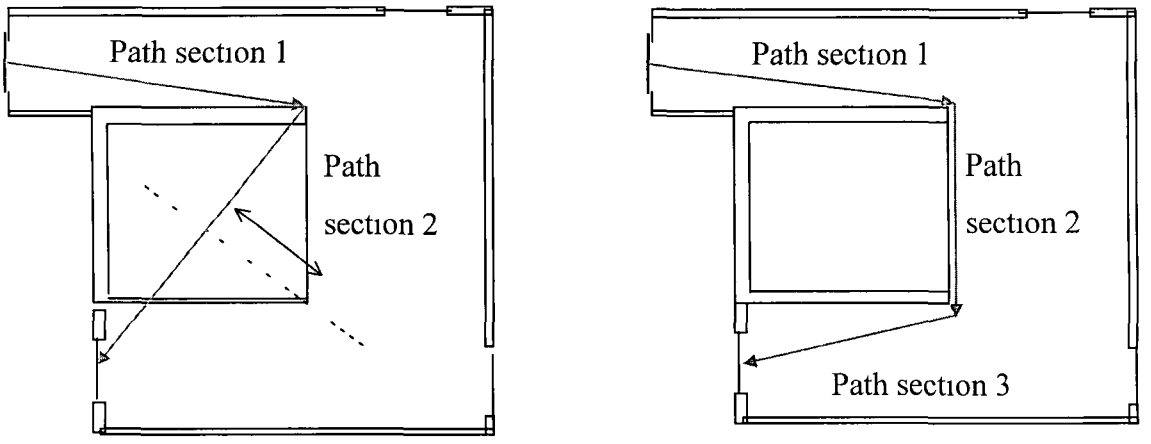

Figure 4.45

The question now remains how to identify the furthest points on any given obstacle. The Linelistlists returned by Algorithms BX and BY can be used for this. The Linelistlist below was calculated by the algorithBY( ) function.

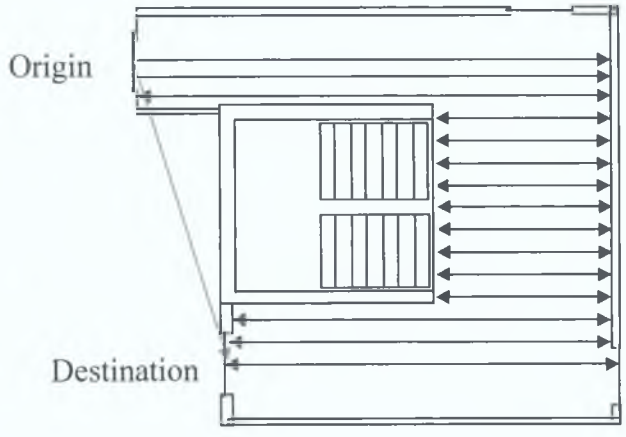

Figure 4.46

The diagonal arrow represents the direct path from origin to destination. If the path was not blocked by any obstacles, this line would lie between the start and end points of all the Line elements in the Linelistlist. The problem is thus to check all the Line elements to see if they meet this criteria, and if they do not, to calculate the perpendicular distance between the diagonal line and the nearest point on the Line element. The best method for achieving this is to:

- Calculate the line equation for the line representing the direct route.
- Each Line element returned by the AlgorithmBY( ) has a fixed y co-ordinate. The x co-ordinate of the point on the direct line for this y co-ordinate can now be calculated using the equation of the direct line.
- The start and end x co-ordinates of the Line element may now be compared to the x co-ordinate of the direct line.
- If the Line element does not intersect the direct line, its perpendicular distance from the line must be calculated.
- If this distance is greater than any previously calculated, record the position of the Line element within the Linelistlist.

The method for processing a Linelistlist returned by AlgorithmBX( ) is similar except that the roles of the x and y co-ordinates are reversed. Once the most distant point has

been established in this manner, the path can be split into two and the algorithm called recursively

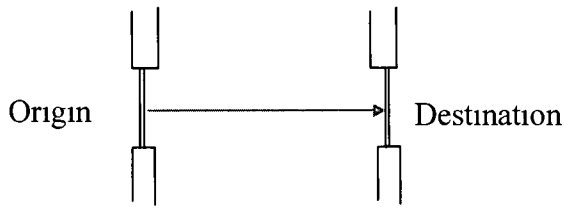

Figure 4.47

Consider the situation above A call to algorithBY( ) would result in a single line being returned, identical to the line shown that also represents the direct path The method described above for using the Linelistlist to detect and negotiate obstacles cannot be used in this case The direct line shown is parallel to the element(s) in the Linelistlist and therefore does not have a single point of intersection The above case is an extreme form of this situation In fact the closer slope of the direct line approaches that of the slopes of the Linelistlist elements the more inaccurate the data processing becomes until the situation above arises where processing becomes completely impossible To maintain a high degree of accuracy and prevent the above situation from occurring, the Traverse_RoomB( ) algorithm makes use of both algorithms BX( ) and BY( )

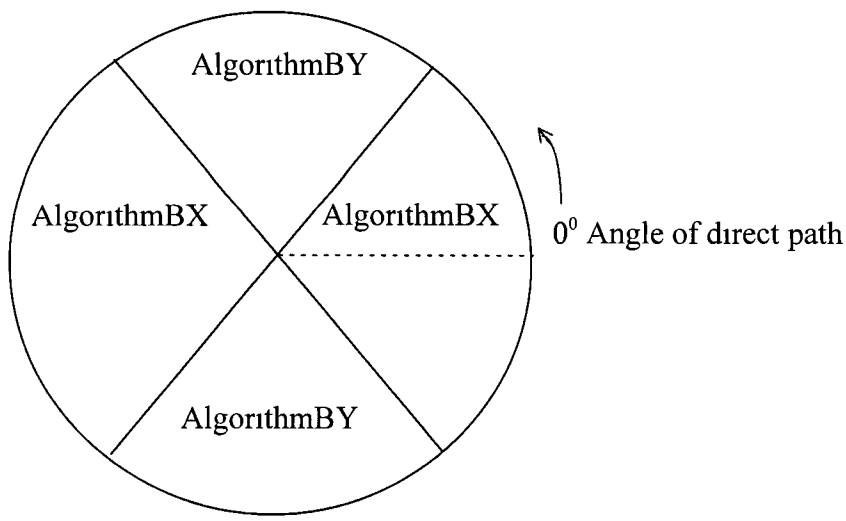

Figure 4.48

The figure above shows how the Traverse_RoomB( ) algorithm decides whether to use the BX or BY algorithms

4.11 Conclusions and recommendations for Algorithm 3

4.11.1. Problem 1

Algorithm 3 is much more thorough than both algorithms 1 and 2 Its main draw back is the amount of processing required in order to calculate paths, this can make the algorithm relatively slow One potential solution is to make the algorithm operate in two stages The first stage would operate as before, with Algorithms BX and BY being used to explore the room as before The returned Linelistlists can be translated into a simple LineList that describes the layout of the room In essence it is a method for identifying which lines of the drawing represent the walls of the room Subsequent calls to calculate paths through the room can then use this LineList instead of relying on the relatively slow Algorithms BX and BY If speed is essential, it is not inconceivable that each room is 'explored' and a LineList describing its interior stored prior to the system coming on line In this way, all future calls will be fast

4.12 Problem 2

Another potential problem with algorithm 3 is how it deals with obstacles of the type shown below

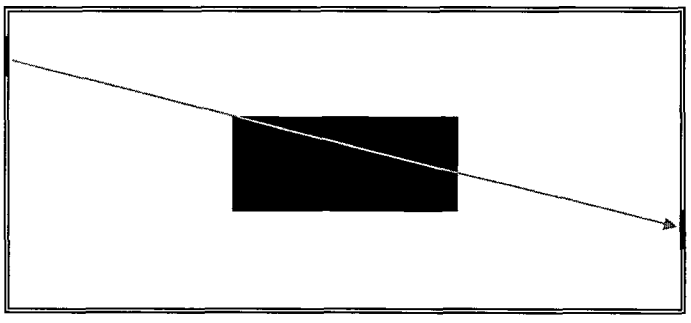

Figure 4.49

While the algorithm will deal with the obstacle shown in Figure 4 49, it will not do so efficiently and may not produce optimum results

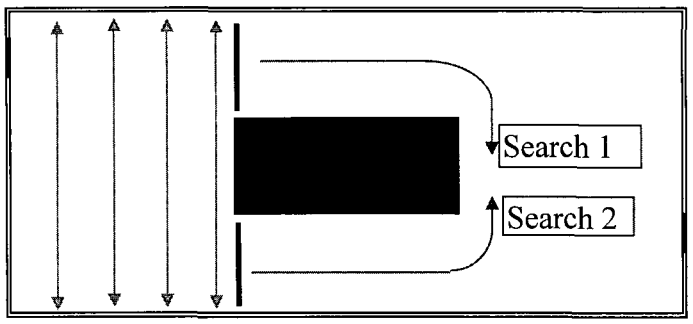

Figure 4.50

Figure 4 50 shows what happens when the algorithm meets the obstacle The two thick vertical lines shown seal of the explored area from the unexplored area The curved arrows show the future direction of the search As it stands, the search that finds the destination first, i e the search that is started first, will cause the algorithm to terminate and report success Search 2 will therefore never happen even though it may produce a shorter path While this is not a difficult problem to overcome, it is evident that if search two is instigated, it will explore an area partially explored by search 1 The total speed of the algorithm will therefore be reduced further This problem will not arise if the suggested solution to problem 1 is implemented successfully

5. User Manual

The program, named 'SiteAid exe', requires a Pentium PC with 8Mbytes of memory

This chapter will describe how to set up and run the program

5.1. Menu layout and functionality

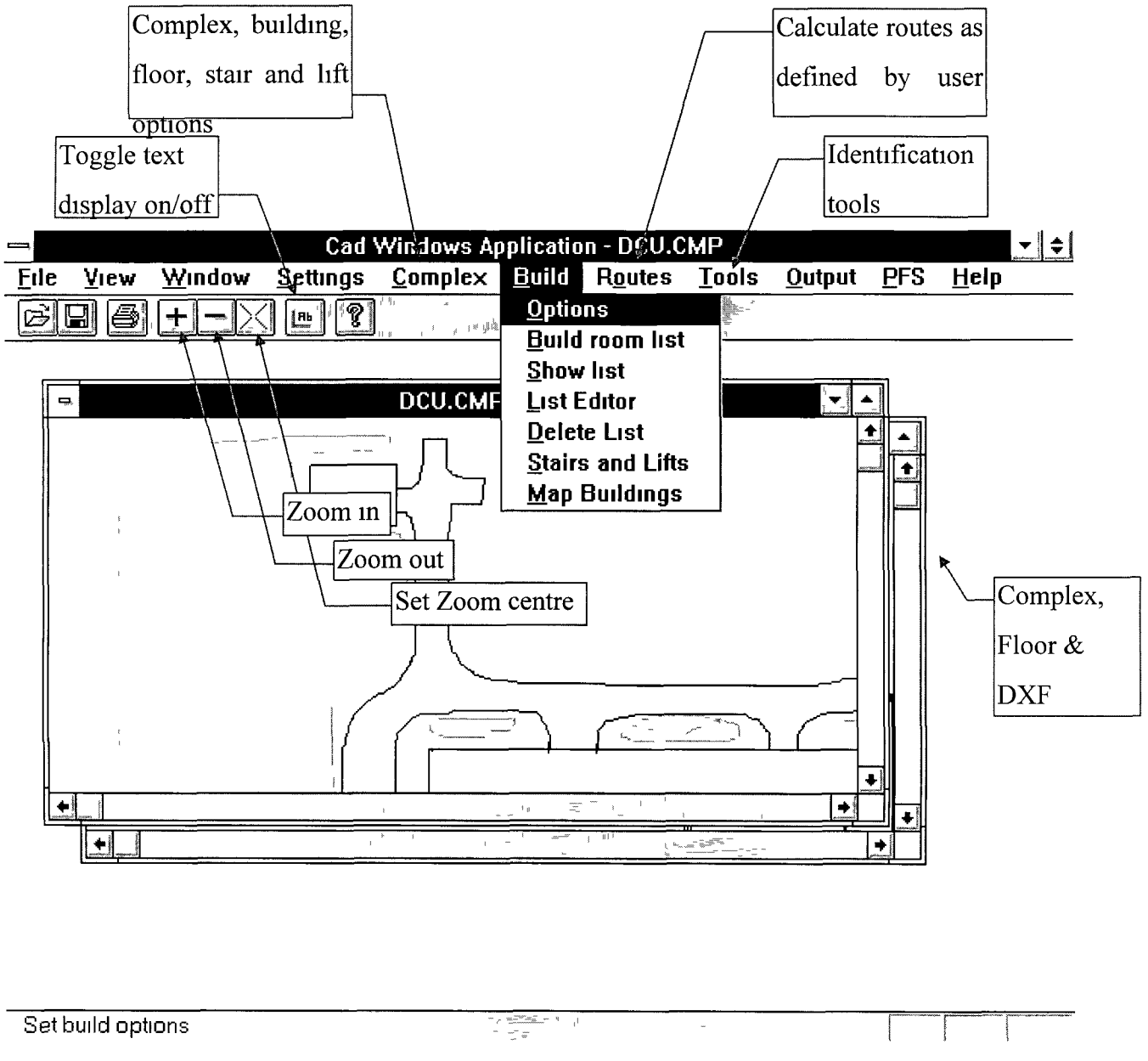

Figure 5.1

5.2 Initial Setup

It is common practice to split a drawing up into layers of specific objects, such as doors or walls Four main object types need to be identified, walls, doors, stairs and lifts The program will automatically examine the layer names on load up time in an effort to establish what type of objects reside on what layer These layer assignments may be examined and edited by selecting the 'Layers' option in the settings menu The diagram below shows this dialogue box

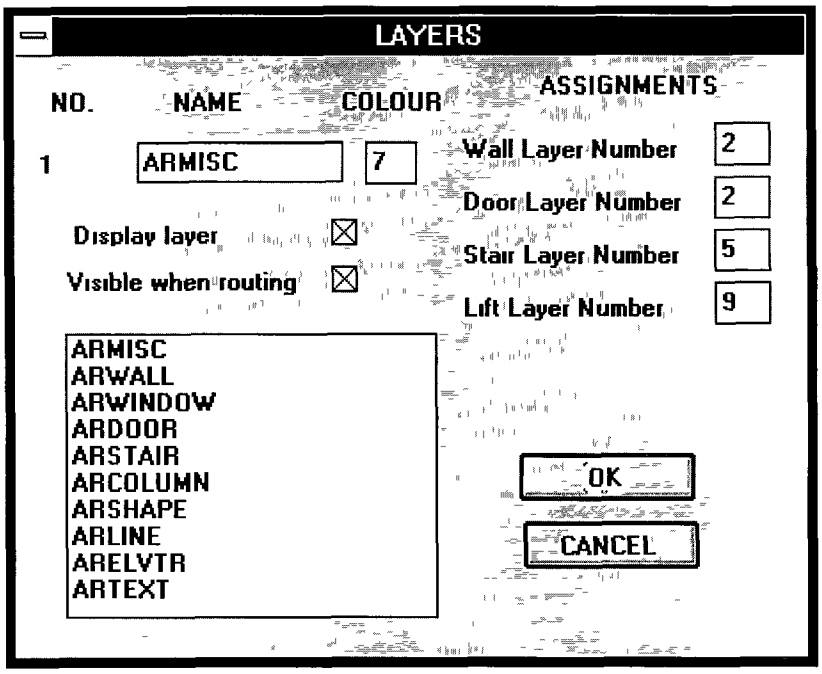

Figure 5.2

In this example, all the layers in the drawing are listed, along with the assigned layers for each object type Initially, the program assumes that all doors are drawn on the ARDOOR layer This in not true for this drawing however, since the doors are actually drawn on the same layer as the walls The door layer assignment is therefore changed to the ARWALL layer, which is layer number 2 The layer dialogue box also allows a user to set which layers are normally visible and which layers are visible during route calculation For example, the ARMISC layer above contains information such as the arrow indicating the direction of the stairs as well as the drawing scale This layer will normally be visible, but should not interfere with route calculations

5 3. Editing a drawing

Selecting the Edit option on the main menu will toggle the Edit menu on and off The edit menu allows a user to add or edit drawing objects New drawing layers may be added or existing layers may be selected

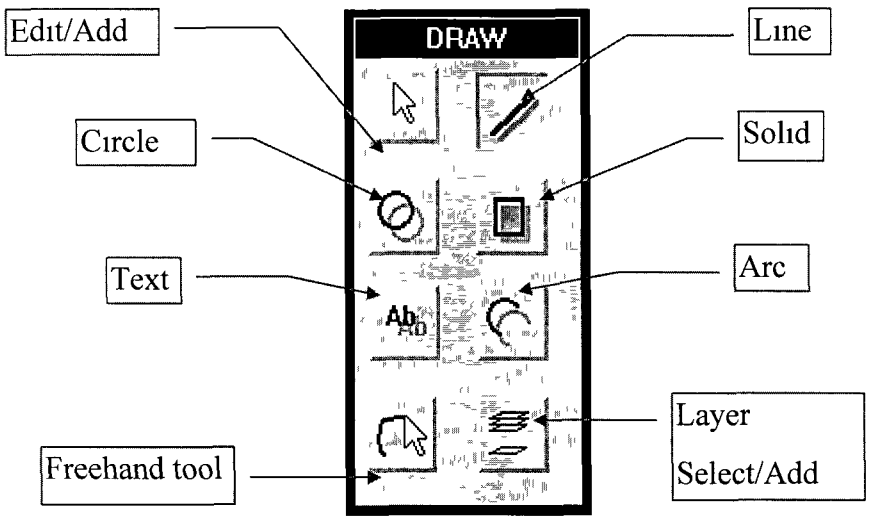

Figure 5.3

5.4 Building and editing a tree

A tree may automatically be built up for a drawing by selecting the 'Build room list' option from the 'Build' menu Build options may be specified in 'Build options' dialogue box These options include aliasing and double-door locking (see Section 4 9 6) The 'Labels' section of this dialogue box allows the user to specify any labels needed for the construction of the list The floor label is the letter used to prefix all the room names This label is automatically chosen but may be changed by the user The stair and lift names are used to identify stairwells and lifts and to establish the inter-connections between floors The ignore and include filter strings are used to identify areas that do not follow the standard room naming procedure (see Section 4 9 5)

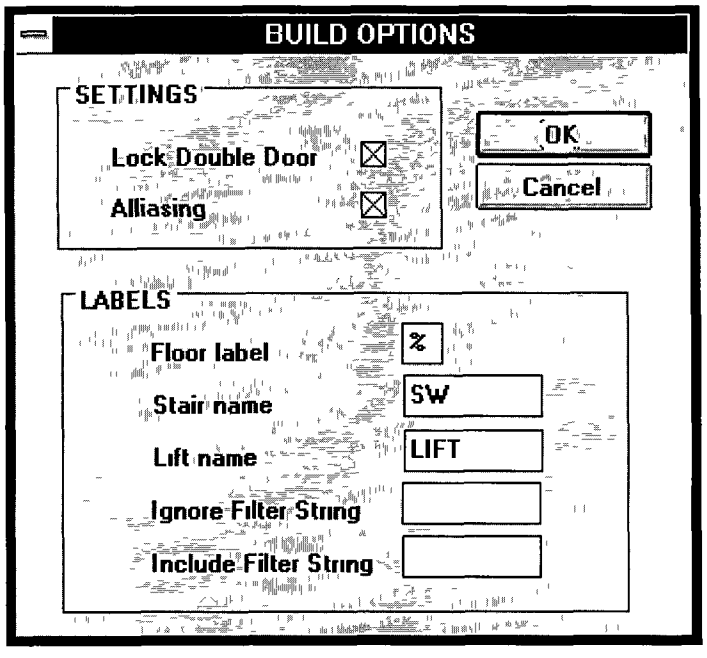

Figure 5.4

The tree produced by the 'Build room list' option may be seen by selecting 'View Tree' from the 'Build' menu This allows the user to visually inspect the tree for any errors that may have occurred as a result of incorrectly placed labels The diagrams below show a simple floor plan and its schematic equivalent

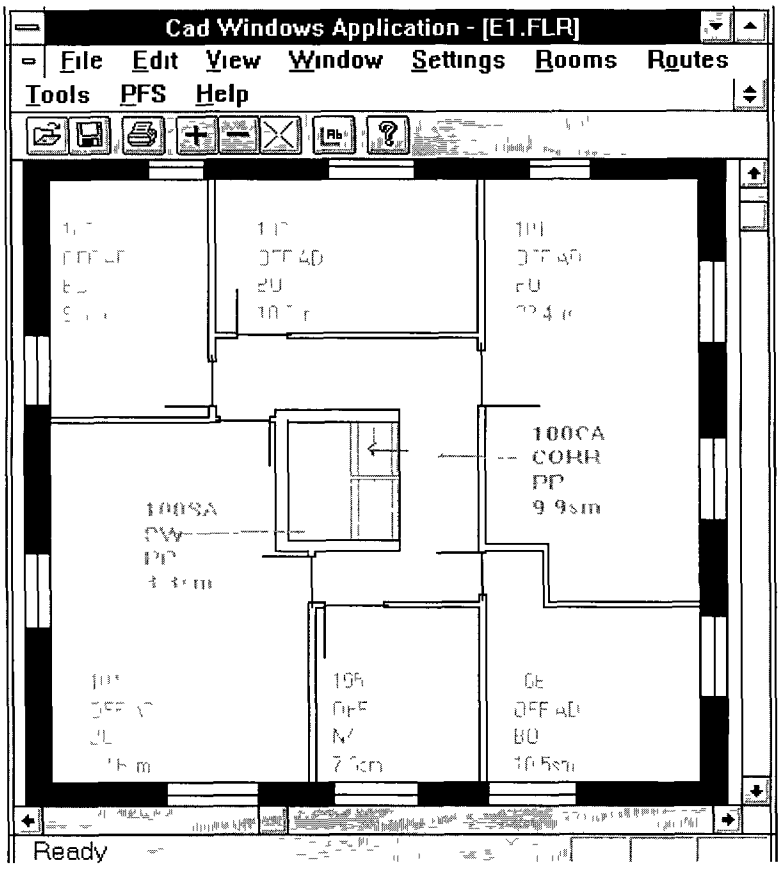

Figure 5.4 A

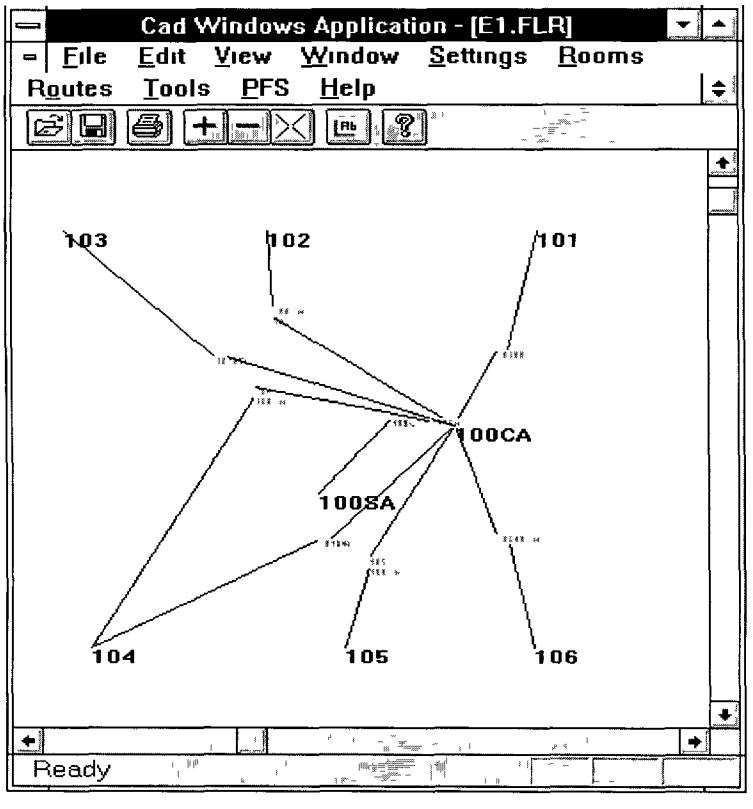

Figure 5.4 B

If the Build algorithm discovers rooms with no access, the user is alerted as to their existence At this stage, a 'virtual door' may be added or alternatively, the drawing may be edited to amend the problem A virtual door is a door that is added to the tree without having a physical representation on the drawing The user is prompted as to the position of the virtual door and its inside and outside co-ordinates

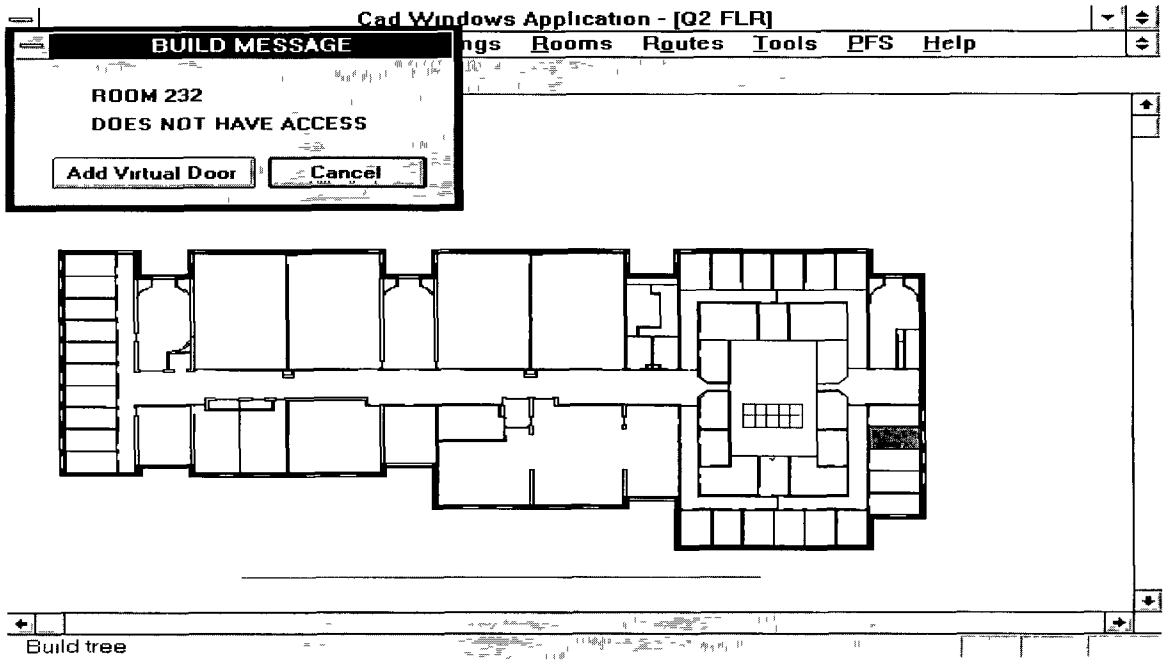

Figure 5.5

5.5. The Room List Editor

A room list may be manually edited by using the Room List Editor The editor displays all the available rooms in a selection list Details of each room may be examined by selecting one of the rooms in the list All aliases and connecting rooms will be displayed The rooms co-ordinates and lock status are also displayed These may be altered by the user as required Door positions and locks may also be updated The editor also allows rooms and doors to be created or deleted

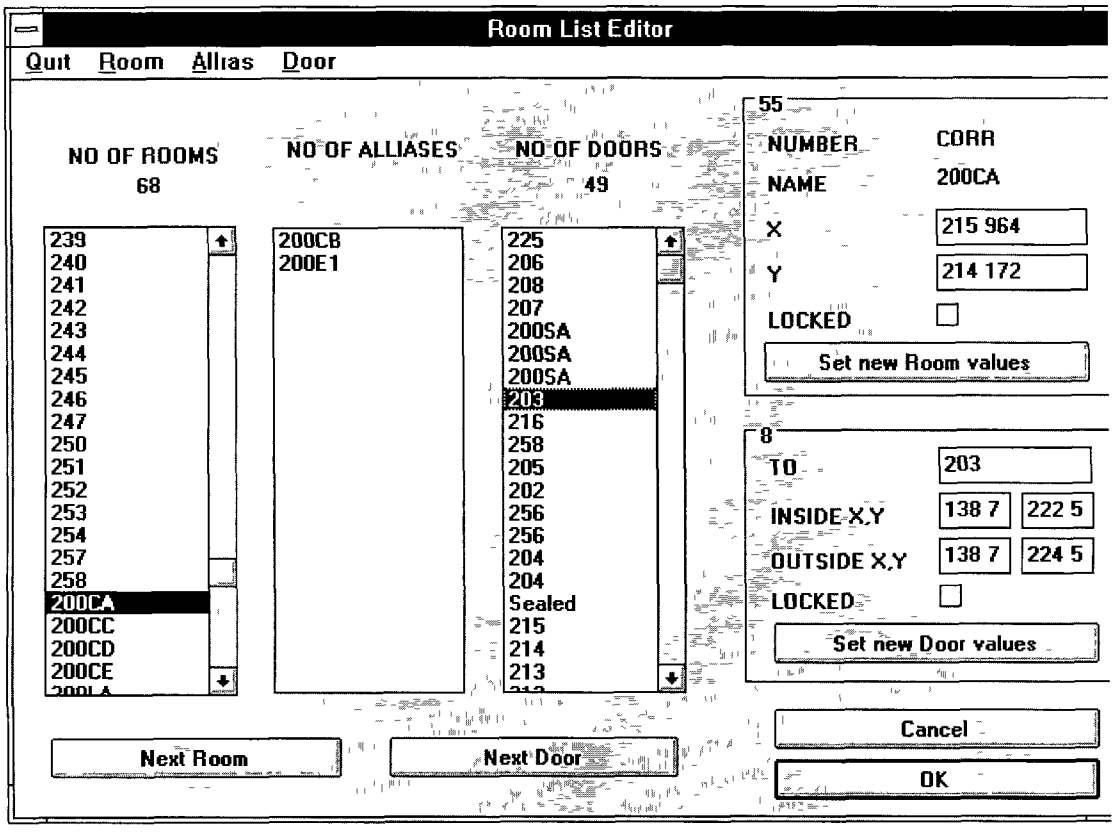

Figure 5.6

5.6. Basic routing

Plotting routes from one room to another may be done in either of two ways The mouse may be used to select specific origin and destination points, or the user may simply select an origin and destination room

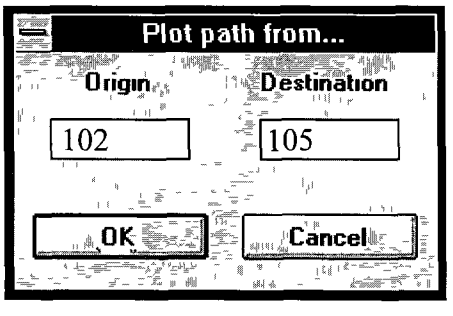

Figure 5.7

The user can select the route type to be displayed by choosing the Route options dialogue box Route types include Shortest, Simplest, Longest, and All The Simplest route option displays the route with the least amount of turns Note that the type of route that is being displayed can be changed after the routes have been calculated

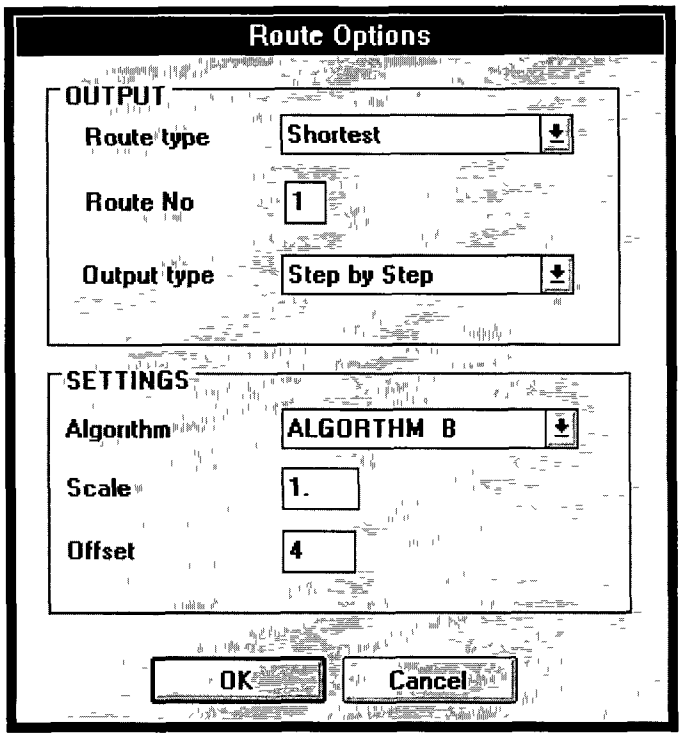

Figure 5.8

A choice of output types is also presented to the user in this dialogue box, step-by-step, door-to-door and room-to-room The step-by-step output is the full point to point route type, the door-to-door output displays a simple door to door route that does not include information about obstacles within the any of the rooms crossed The room-to-room output simply consists of a list of rooms that can be used to get from one room to another The last two output options do not require the use of either Algorithm A or B and are therefore very fast

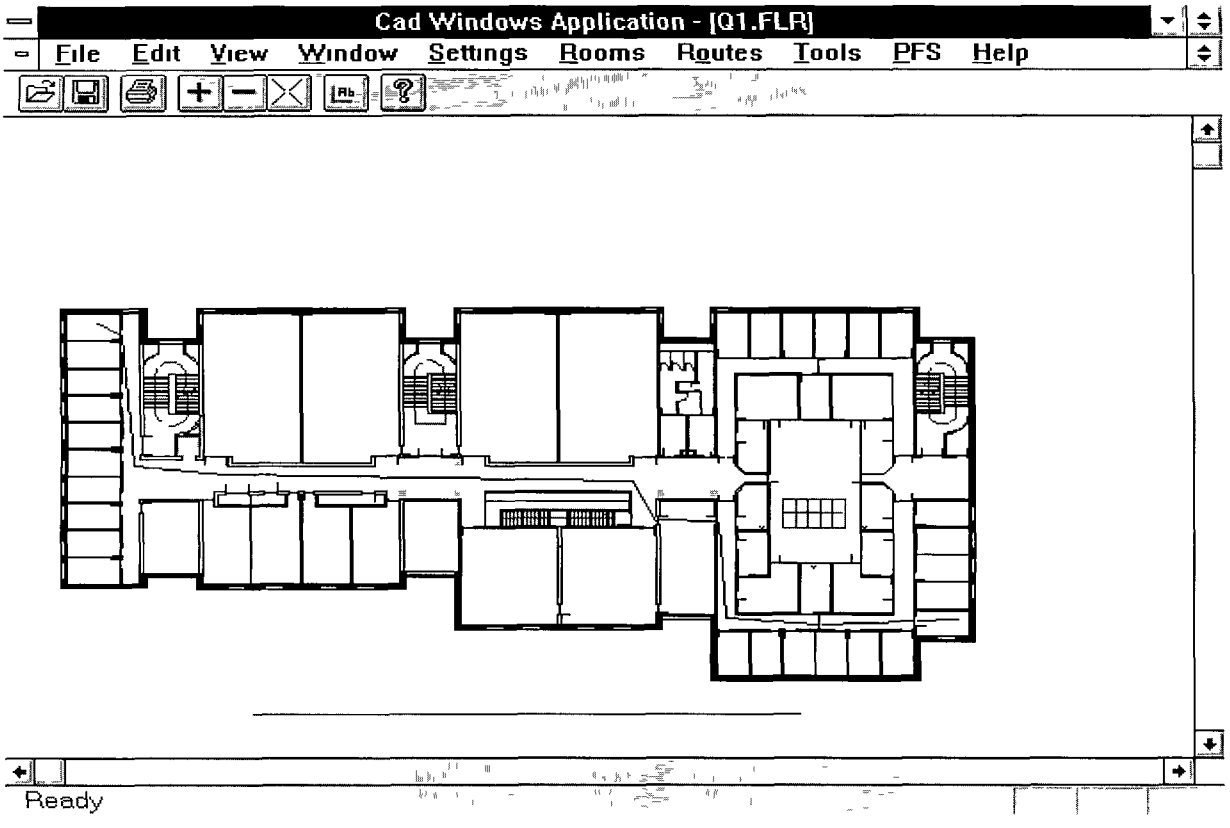

Figure 5.9 Shortest Route

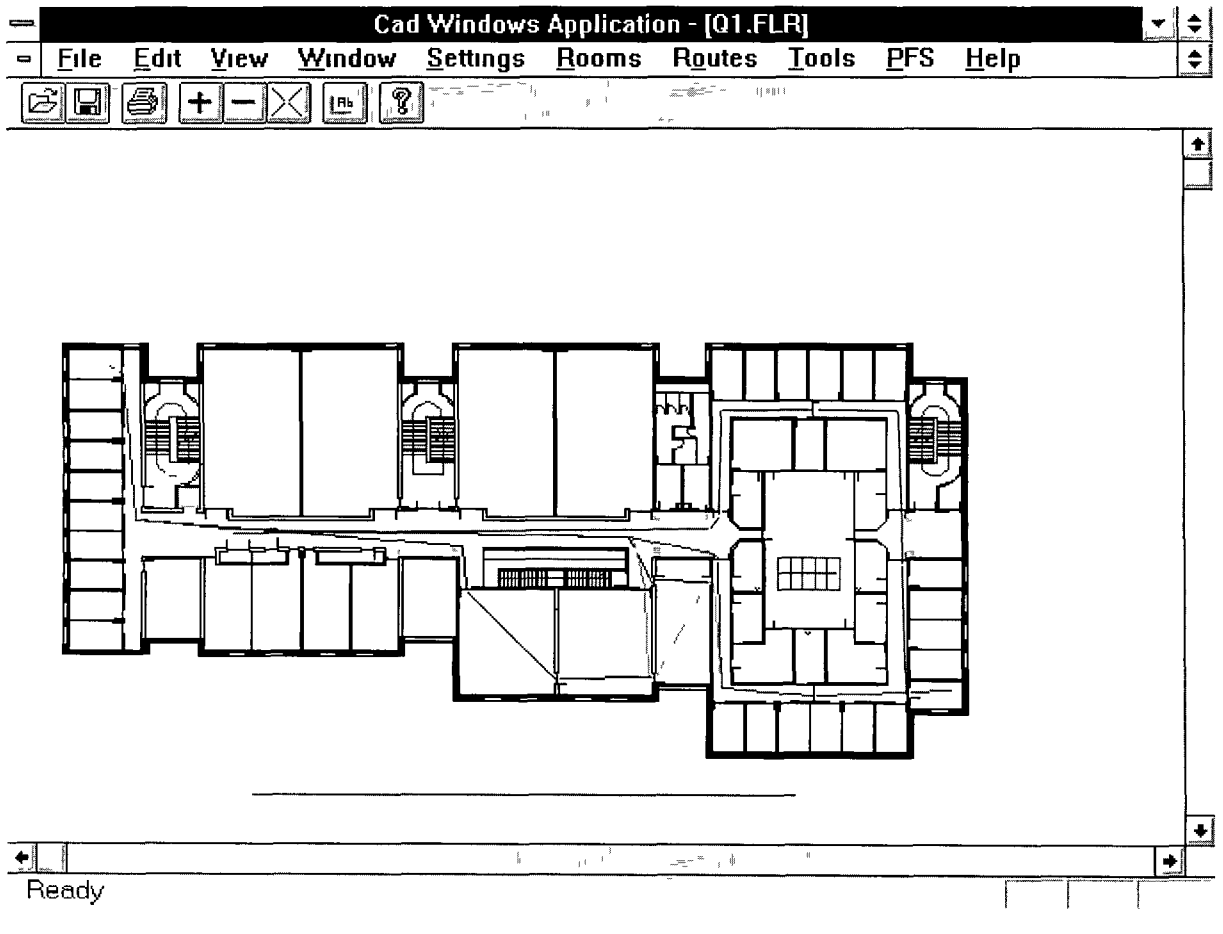

Figure 5.10 All Routes

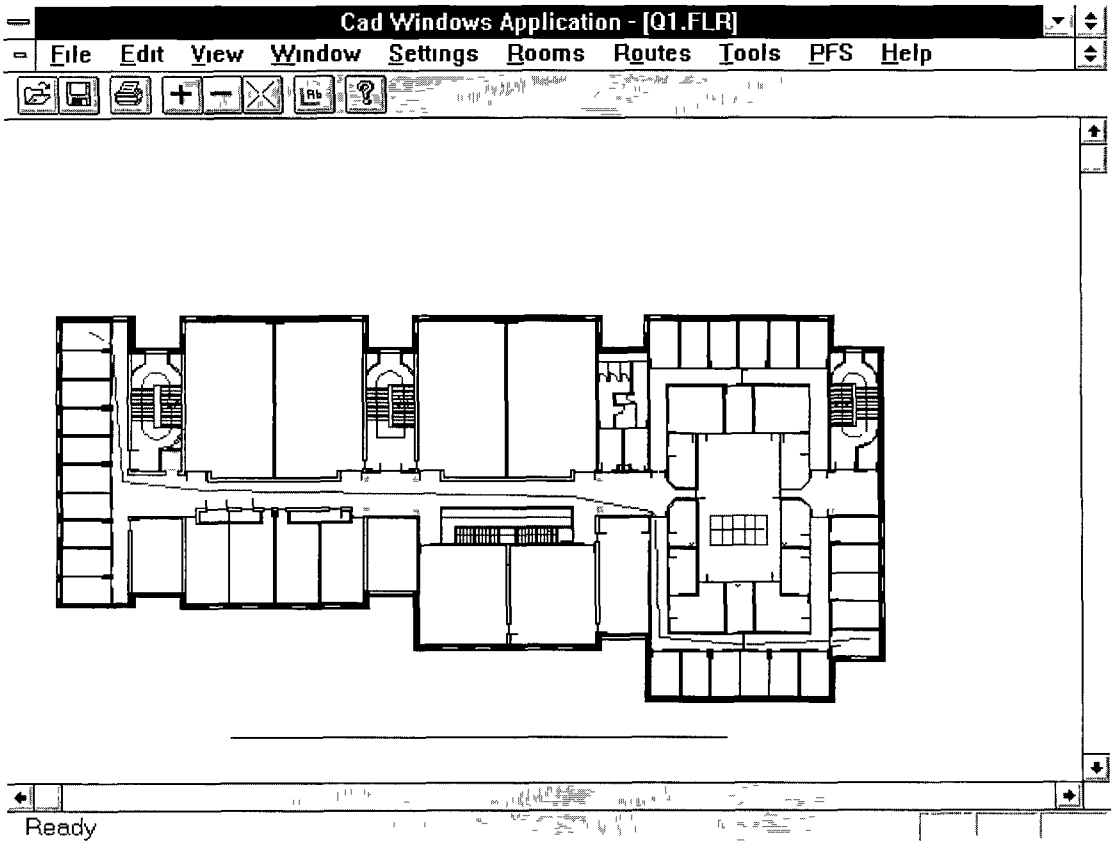

Figure 5.11 Simplest Route

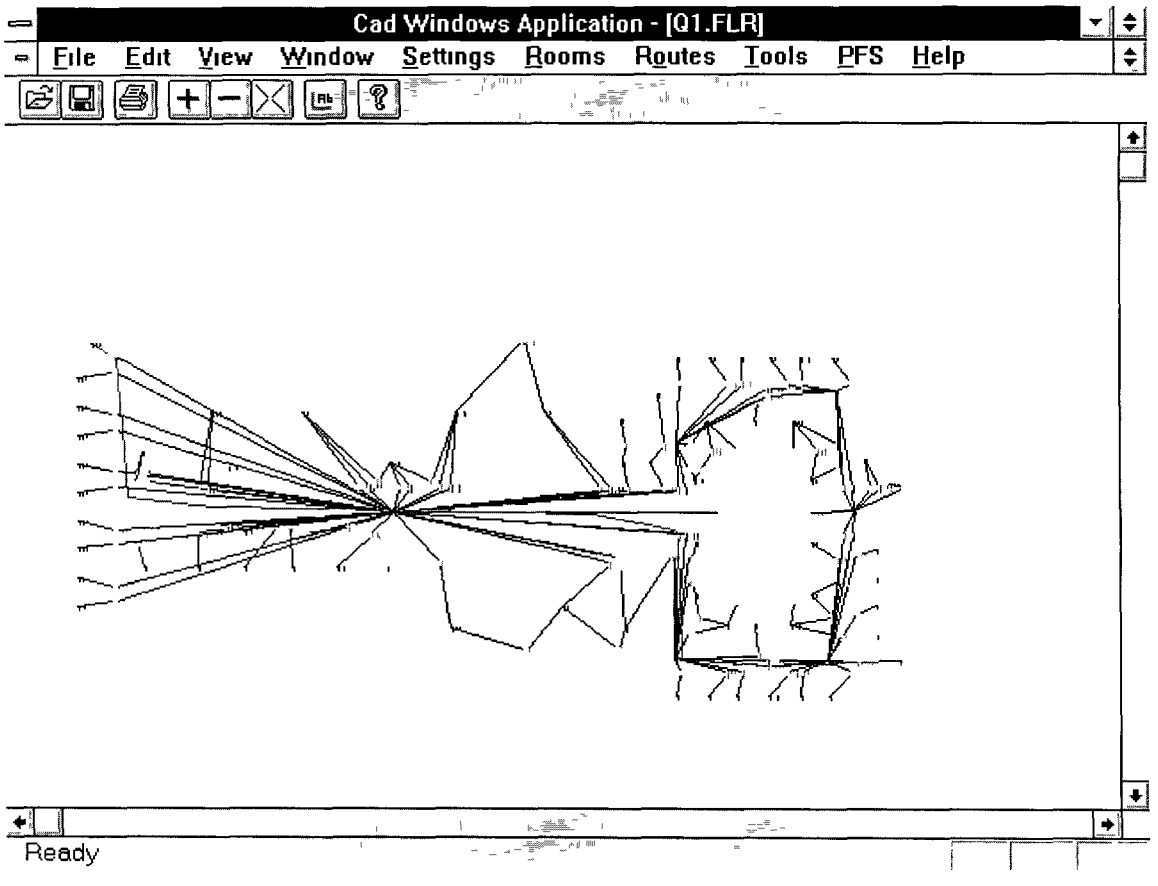

Figure 5.12 A route displayed on the room schematic

5 7. Creating a Complex

When creating or updating a complex, the complex editor is used It is a tool by which buildings may be added and mapped on an overall map of the complex The user specifies the name of the complex, the name of the plan file and any buildings which are contained in the complex The floors of each building are listed in the floors selection list

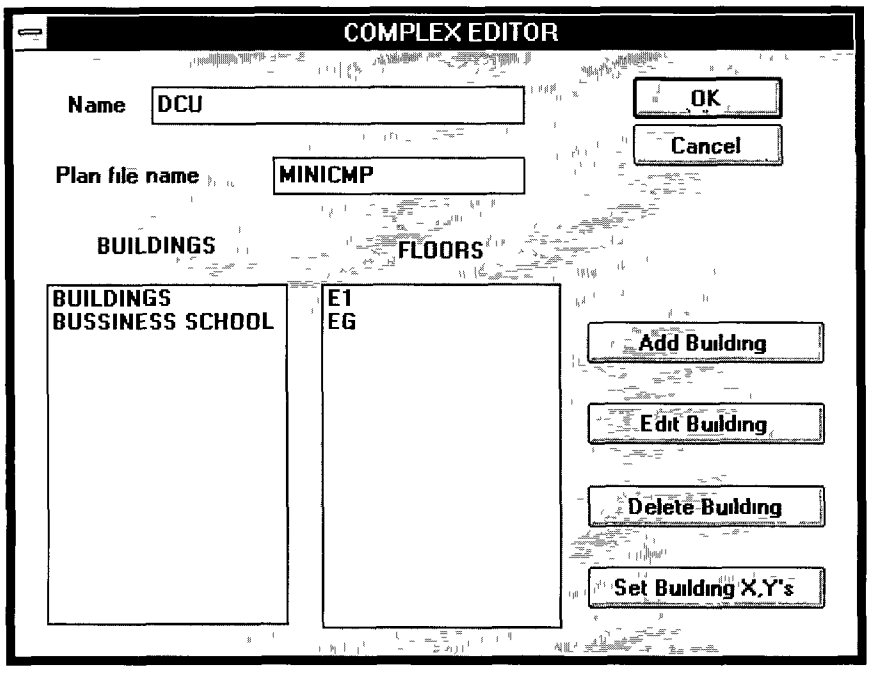

Figure 5.12

Individual buildings may deleted or edited by selecting them in the buildings selection list and clicking the appropriate button The 'Set Building X,Y's' button automatically links the currently selected building to the complex and assigns co-ordinates to the building These co-ordinates may be viewed or edited in the building editor tool The building editor also allows floors to be added or deleted from buildings and building name and label to be set The building label is a unique alphabetic identification code that must precede the names of all the floors in a building For example, the ground floor of the 'business school' building shown below is labelled 'Q' and has three floors, QG, Q1 and Q2 By using this method,

every room in the complex may be uniquely identified, e g QG02 is a room on the ground floor of the business school

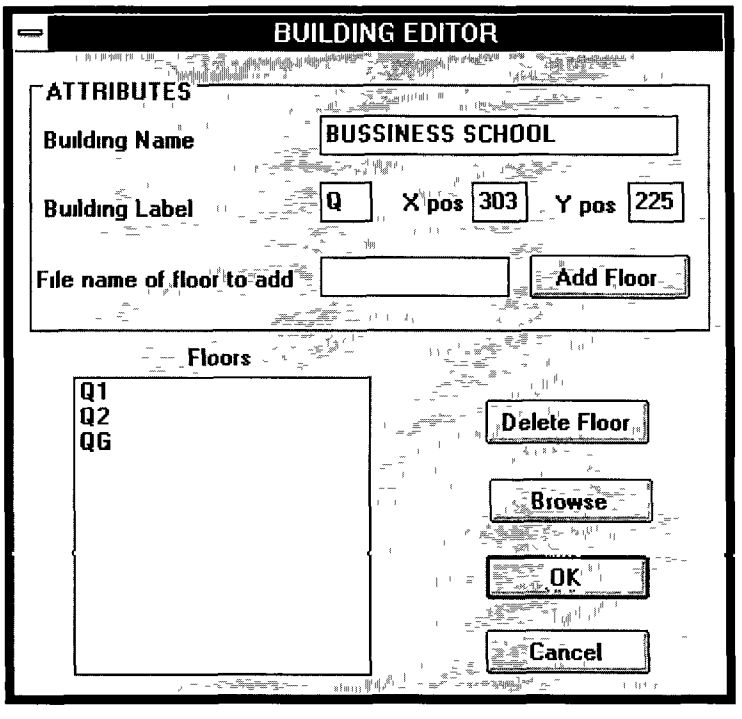

Figure 5.13

After all the buildings have been added to the complex, it is necessary to map the entry and exit points of each building onto the plan drawing of the complex To update or add the entry and exit points for a building, load the ground floor map for that building and select the 'Map buildings' option from the 'Build' menu

5.8. Lifts and Stairs

Once a building has been added to a complex the stair and lift connections between the floors are automatically constructed These connections may be examined or edited using the stair and lift editor shown in Figure 5 14 below

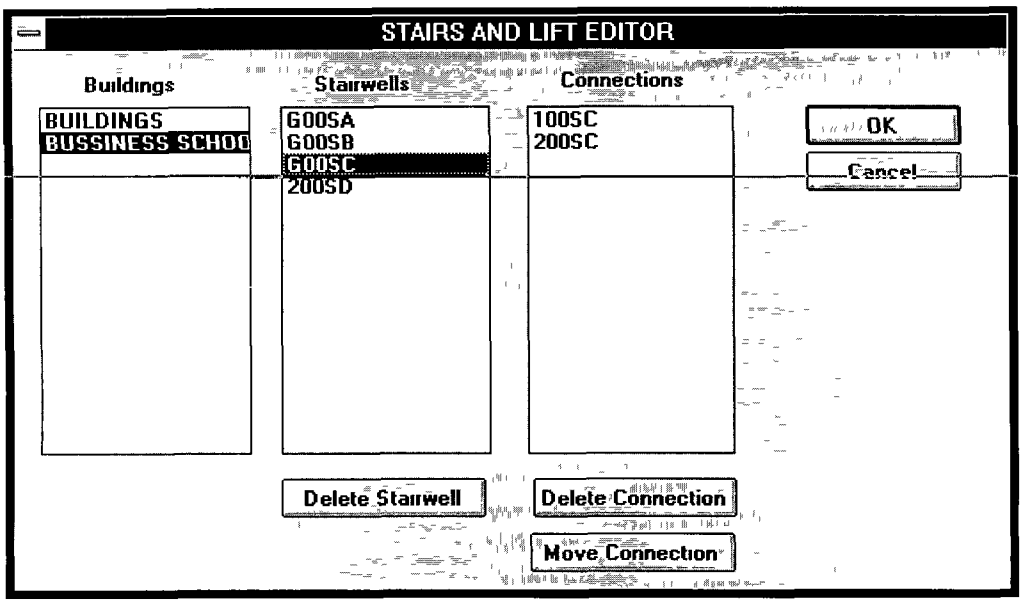

Figure 5.14

All the buildings within a complex are listed in the 'Buildings' selection list The user may select any of these buildings and examine the connections between the floors of that building The 'Stairwells' selection list shows the lowest sections of all the lifts and stairs in the building Selecting an entry in the 'Stairwells' selection list will result m all the connections of that entry being shown in the 'Connections' selection list In figure 5 14, we can see that the business school has three staircases starting on the ground floor, and one on the second floor The stairwell 'G00SC''s connections are shown It can be seen that this staircase connects all three floors The last staircase shown in the 'Stairwells' list has no listed connections, this is because it leads to the roof and therefore does not connect any floors together

5 9. Complex Routing

When plotting the route between two buildings in a complex, all the relevant maps are automatically displayed with the routes shown on them The diagram below shows a very simple complex with two buildings

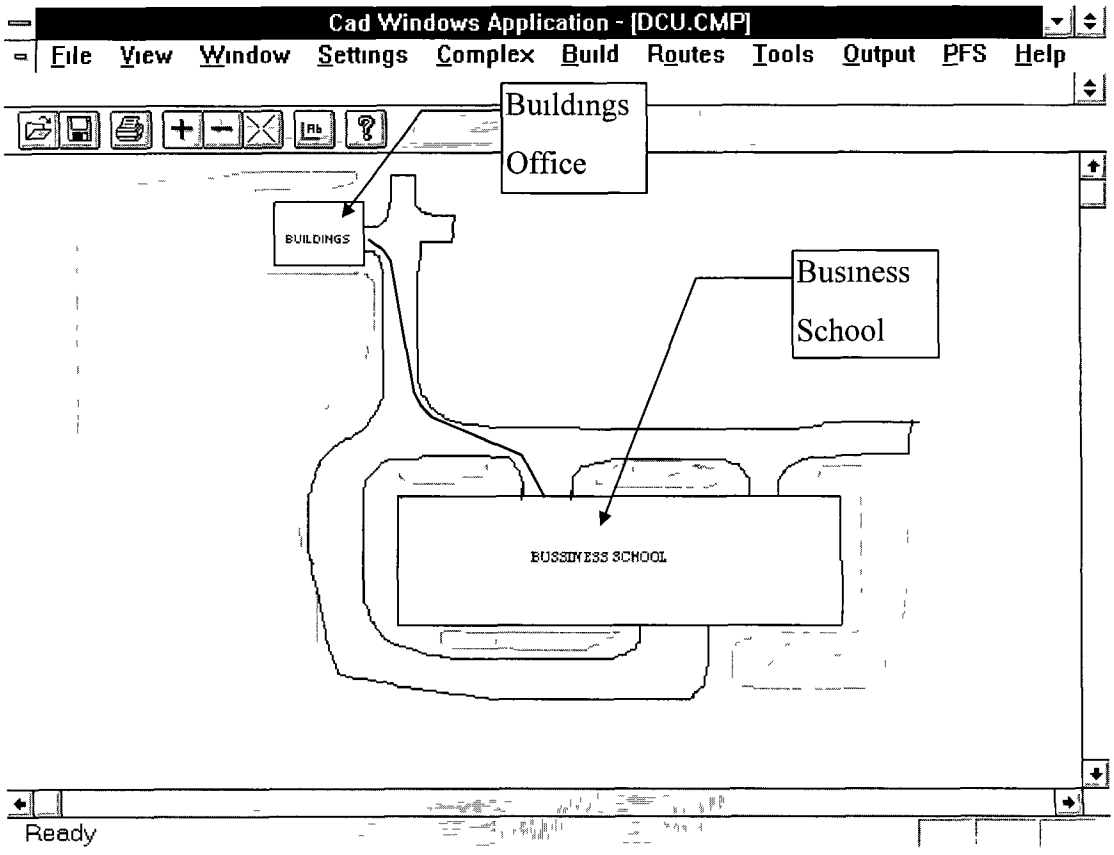

Figure 5.15

If for example a route needs to be calculated from room 'A' on the first floor of the 'Buildings office' building to room 'B' on the ground floor of the 'Business School', then the following routes have to be calculated and displayed

1 From room 'A' to all the stairs and lifts with access to the ground floor
2 From all the stairs and lifts on the ground floor with access to the first floor to all the exit points
3 From all the exit points from the 'Buildings office' to all the entry points on the ground floor of the 'Business school'
4 From all the entry points to room 'B'

Figure 5 15 shows the shortest route between the two buildings

6. Conclusions

While the design of a practical navigation aid is still very complex, it is an achievable goal with the help of modern technological advancements The many factors involved in the design, manufacture and testing of a practical navigation aid necessitate a large investment in man-hours The purpose of this work has been to establish a solid foundation upon which to base further research and development Two fundamental areas were covered that of establishing reliable design criteria and specifications, and the development of software which can be used to yield any required navigational information These achievements can serve as a test bed for further research The investigation into the requirements of a navigation aid carried out yielded some very important points that must be taken into consideration The strength and reliability of these results do not rest solely on the opinions of people who have been involved in the design of assistive technologies for a significant number of years, but more importantly, on the opinions of the people whom the technology is designed to benefit All too often well meaning designers develop products that while technically excellent, are simply not practical for reasons as simple and mundane as comfort or cost The mam results condensed from the end-user survey are repeated here A navigation device must complement existing travel aids, be affordable, inconspicuous, provide clear and concise information, be easy to use and learn and last but not least, the device should not limit the personal freedom of the user in any way

The package developed can prove to be very beneficial for any future work carried out It can be used for feasibility testing and modelling of positioning systems, advance planning and testing of actual navigation aid systems within building complexes, as well as stand alone applications such as pre-journey route planners and information points Whilst the package works well as it stands, its object orientated architecture allows for easy expansion and modification The design model may be added to without restriction For instance, the model currently consists of a complex containing buildings which in turn contain such objects as rooms, stairs Extra abstract objects representing such things as streets, parks, roads and so forth could

easily be integrated into the model Larger abstractions can also be added, for example the complex of building could become part of an area, town or city

Even though the work carried out for this project is not inconsiderable, it only scratches the surface of the total amount of work that needs to be carried out to bring about a fully functioning, practical navigation aid The software should be field tested by utilising it in stand-alone information points Such points could be used by sighted and visually impaired people alike, provided the correct interfaces are supplied This would in turn prompt further research to be carried out into interfacing the software with the real world through practical user-interfaces The user interface is responsible for relating information to the user from the navigation device as well as notifying the device of user requests The communication between the user interface and the device is thus also a major design consideration If the navigation aid's control hardware is external, the use of terrestrial radio direction finding techniques for establishing position may be utilised as a means of communication

Proposals for user interfaces have been forwarded in this thesis but a considerable amount of research is still required, such work requires extensive field testing in order for truly suitable interfaces to be developed

Another area which requires more extensive research, is the development of a practical, reliable and inexpensive positioning system capable of operating in a complex of buildings, both outside and inside the buildings The accuracy and reliability of this system will dictate the practical usage of the navigation aid For example, it would be pointless for a user to request directions to a destination from their current location if their current location cannot be established with a reasonable degree of accuracy

The field testing of designs is of extreme importance, bringing everything from major design errors to tiny flaws to light Test results may be used to improve existing designs or lead to new innovations until a viable, practical navigation aid is perfected'

Appendix A: The .CMP File

The following is the format used in the CMP file The indentation is used to illustrate that the indented information will be repeated the number of times indicated on the line preceding the indentation

Name of Complex
No of Buildings contained in it
    Building name
    Building label
    No of floors in Building
        Floor filename
    No of Stairs
        No of flights
            Stairwell name
    No of Lifts
        No of possible stops
            Names of connecting rooms
Plan filename

Appendix B: The .FLR File

This file type replaces the DXF file, which is much slower to load and save Its format is as follows

Tree information flag - this is 0 if no TRE file exists

limmaxx - limminx - limit information

limmaxy - limminy

topx

topy

bottomx

bottomy

wallno - the layer on which the walls are drawn

doorno

textno

stairno

liftno

No of Layers

- layer name
- frozen - this is 1 if the layer should not be displayed
- colour
- No of Lines
  - x
  - y
  - x1
  - y1
- No of Solids
  - x
  - y
  - x1
  - y1
  - x2
  - y2
  - x3

y3

No of Arcs

x

y

r

a1

a2

No of Circles

x

y

r

TextLayer name

frozen

colour

contd /

No of Text items

x - Text x,y height and contents

y

h

text string

No of Lines

x

y

x1

y1

No of Circles

x

y

r

No of Virtual Arcs - virtual arcs are for rooms with no apparent access They are -not normally displayed

x

y

r

a1

a2

Appendix C: Function Reference

coords Find(resize r, char*Area, CDC* pDC, int nColour);

Finds the coordinates of the object with given label or description as identified by the Area parameter The area will be filled with the colour specified by nColour The first parameter, r, tells the function at what zoom level the drawing should be refreshed This parameter and the ccurrent device context is common to most funcions

CString Identify(int x, int y, CClientDC* pCDC, resize r);

Identifies area with given coordinates x,y The function returns the label associated with the area

void IdRooms(resize, CDC*, BOOL, BOOL);

Automatically identifies all the rooms on a floor and builds up the RoomList

virtual RouteList get_path_from (char *Area1, char *Area2);

Calculates a Routelist from the origin identified by Area1 to Area2 This fucntion is used in conjunction with the move_along function

virtual DataList move_along

(Routelist* route, Linelistlist* list,resize r,float,BOOL, CClientDC* pCDC);

This function is used in conjunction with get_path_from function It takes the returned Routelist from this function

LineList Floor::traverse_roomB(Data *lin, float offset, CClientDC *dc)

This member function is responsible for implememting the third search algorithm It depends on two helper functions, algorithmBX and algorithmBY, to return vertical and horizontal search data which is then processed by this function to produce the final LineList for the overal path

void Floor::algorithmBX(Linelistlist *ret, Data *lin, float offset, CClientDC *dc)

Helper function of traverse_roomB It implements a **horizontal** search in any given space and returns the data to its calling function

void Floor::algorithmBY(Linelistlist *ret, Data *lin, float offset, CClientDC *dc)

Helper function of traverse_roomB It implements a vertical search in any given space and returns the data to its calling function

Appendix D: Listings

The Get_Path_From Algorithm

```
//The function 'get_path_from' creates a routelist (if possible) containing
//the route elements (i.e. rooms or doors etc.) which must be passed in
//order to get from the origin to the destination.  The function firstly
//checks if the two inputted names (here and there) correspond to two existing
//rooms.  If so, the doors in the origin room are checked to see if they lead
//directly to the destination, in which case the Routelist consists simply of the
//origin, the door(s) leading to the source and the source.  If, as will be the
//case in most situations, to doors from the origin room do not lead directly
//to the destination, the room leading from the first door is entered and
//control is passed to the 'find_dest' algorith which takes three parameters,
//a pointer to the current Route element, the current room and the
//the destination room.  The current Route element is passed in so that new
//Route elements may be added to the Routelist.  All the doors from the "new"
//room are subsequently checked and if the destination is not found, the room
//leading from the first door is entered and the algorith is called
//recursively.  Each room entered is given a status as described by the
//parameter checked contained in each room.  A zero flag indicates that the
//room has not been entered before, while a 1 indicates that the room has
//been checked before and may or may not lead to the destination.  This is
//done to prevent the algorithm from re-entering rooms already being checked
//by previous calls to 'find_dest' or 'get_path_from' and thus prevents the
//algorithm from going around in circles.  A status of 2 is assigned to any
//room that is know to lead to the destinaion.  When such rooms are
//re-encountered, the current Route element is made to point to the Route
//element for the current room.  By joining routes in this manner, the total
//search time for all possible roots is decreased.  Routes are joined together
//by means of the Room member function 'join()' which together with 'link()'
//allow the algorithm to remember the address of the Route element associated
//with a particular room.  The last operation carried out is the reseting of
//all the rooms 'link' and 'visited' values.



Routelist Floor::get_path_from(char *here, char *there)
{
    Room *pRoom=NULL;
    Room *origin=NULL;
    Room *destination=NULL;
    Door *thru=NULL;
    Route *position=NULL;
    Route *pRoute=NULL;
    Route *pRoute2=NULL;
    Routelist path;
    RouteCapsule *pRt = NULL;
    RouteCapsule *pLast;
    CString Here = here;
    CString There = there;

    int nw=1;       //Flag used by get_next functions to indicate first call
    int chk=1;
```

```
    BOOL success = 0, //Has the destination been found?

//Firstly, check to see if the roomlist is available
    if ( rmlist no_of_rooms() == 0 )
        {
        AfxMessageBox("No rooms in list", MB_OK, 0),
        return( path ),
        }

     if ( Loaded == 0 )
        {
        AfxMessageBox( "Floor not loaded", MB_OK, 0 ),
        return( path ),
        }

//Find the origin and destination rooms
     do
        {
        pRoom=rmlist get_next_room( nw ),
        nw=0,
        if ( pRoom != NULL )
                {
                CString RoomName = pRoom->number,
                if ( RoomName CompareNoCase(here) == 0 ) origin=pRoom,
                        else if ( RoomName CompareNoCase(there) == 0 ) destination=pRoom,

                Allias *alls = pRoom->GetAllias(), //also check any allias names
                while ( alls != NULL )
                        {
                        RoomName = alls->numberval(),
                        if ( RoomName CompareNoCase(here) == 0 ) origin=pRoom,
                        else if ( RoomName CompareNoCase(there) == 0 ) destination=pRoom,
                        alls = alls->newmemval(),
                        }
                }
        }
  while ((pRoom!=NULL)&&((origin==NULL)||(destination==NULL))),

  // Do some preliminary check to see if the origin and destination selections are valid
  if (origin==NULL)
        {
        AfxMessageBox("Origin not found",MB_OK,0),
        return(path),
        }
  else if (destination==NULL)
        {
        AfxMessageBox("Destination not found",MB_OK,0),
        return(path),
        }

  if ( origin == destination )
        {
        AfxMessageBox( "Select different Origin and Destination rooms", MB_OK, 0),
        return(path),
        }

if ( strcmp( here, "Outside" ) == 0 ) //Its easier to go from a room to the Outside
        {
```

```
        pRoom = origin,
        origin = destination,
        destination = pRoom,
        }

//Start at the origin and search for the destination
origin->visited(1),

//Add the destination to the routelist
position=(path addroute(destination)), //the addRoute function returns a pointer to the element added

//Enter each of the unlocked doors in a room in turn to see if it leads to the destination  If it
//does not, call the 'find_dest' function
if (chk)
        {
        nw=1,
        for (int i=0, i<origin->no_of_doors(), ++i)
                {
                thru=origm->get_next_door(nw),          // Get doors in room one by one
                nw=0,

                if ( (thru->roomval() == destination) && (thru->GetLock() == 0) )      // Check to
                                                        //see if the destination has been found
                        {
                        success = 1,
                        thru->connect_no(-1),
                        pRoute=position->addRoute(Route(thru)),
                        pRoute->addRoute(Route(origin)),
                        chk = 1,
                        }
                else
                        {
                        chk=(thru->roomval())->checkval(), //Has the room leading from the door
                                                        //been checked
                        if ((thru->roomval())->GetLock() == 1) chk=3,
                                                        // is the room or the door locked?
                        else if (thru->GetLock() == 1) chk=3,
                        }

        if (chk==0)
                {
                pRt=find_dest(position,thru->roomval(),destination), //Enter the room
                pLast=pRt,
                if (pRt->GetContents()!=NULL)
                        {
                        success = 1,
                        pRoute=pRt->GetContents(),

//If the destination is found, add the Room through wich it was found(1)  Set the link value in that
//room so any subsequent routes discovered that pass through that room can join directly to that
//route(2)
//Next, add the door that leads to the room(3) and the origin(4)  If more than one route was
//dicovered(5&6),
//then the room, door and origin must also be added to these routes (7)

                        pRoute2=pRoute->addRoute(Route(thru->roomval())),       //1
                        (thru->roomval())->link(pRoute2),                       //2
```

```
                    thru->connect_no(-1),
                    //Increase connect no by 1 (for delete purposes)
                    pRoute2=pRoute2->addRoute(Route(thru)),          //3
                    pRoute2->addRoute(Route(origin)),                //4
                    pRt=pRt->newmemval(),                            //5
                    if (pRt!=NULL) pRoute2=pRt->GetContents(),
                    else pRoute2=NULL,
                    if (pRoute2!=NULL)                               //6
                    do
                            {
                            pRoute2->change_newroute(pRoute->newrouteval()), //7
                            if (pRt!=NULL) pRt=pRt->newmemval(),
                            if (pRt!=NULL) pRoute2=pRt->GetContents(),
                            else pRoute2=NULL,
                            }
                    while (pRoute2!=NULL),
                    }
            pLast->Delete_all(),
            delete(pLast),
            }
            else if (chk==2)   //If the room has a check value of 2, it leads to the destination
                               //Join the routes at this point and add the door and origin as before
            {
            pRoute2=(thru->roomval())->join(),
            thru->connect_no(-1),
            pRoute2=pRoute2->addRoute(Route(thru)),
            pRoute2->addRoute(Route(origin)),
            }
        }
    }
    nw=1,
    do
    {
            pRoom=rmlist get_next_room(nw),
            nw=0,
            if (pRoom!=NULL)
                    {
                    pRoom->visited(0),
                    pRoom->link(NULL),
                    }
            }
    while (pRoom!=NULL),

    if ( !success ) path delete_all(),      //If the destination wasn't found, return an empty routelist

    return( path),
}
```

The AlgorithmB listing

```
void Floor  algorithmBX(Linelistlist *ret, Data *lin, float offset, CClientDC *dc)
{
Linelistlist recurs,
Linelist store,
int x=(int)lin->xval(),
int y=(int)lin->yval(),
int x1=(int)lin->x1val(),
int y1=(int)lin->y1val(),
int uptempy,downtempy,tempx,
int newfronty,newbacky,
int swop=0,swop2,direction,oldswop,oldswop2,truedirection,success=0,
COLORREF test,
COLORREF test2,
COLORREF front,
COLORREF back,
COLORREF backcolour=dc->GetBkColor(),
COLORREF oldfront,
COLORREF oldback,
int limit=0,                                    //debug only
CPen pen,
CPen pen2,

pen2 CreatePen(PS_SOLID, 1, backcolour),
pen CreatePen(PS_SOLID, 1, wallcolour),
CPen *pOldPen=dc->SelectObject(&pen),
if (x1>x) direction=1,
        else if (x1<x) direction=-1,
                else direction=(int)lin->desc2val(),
truedirection=direction,
if (lin->descval()==500) direction=-direction,
tempx=x+(direction*(int)(-1)),

//The front and back test points are used to find other possible routes  If a test point encounters
//a wall, followed by more blank space, followed by another wall, an opening has been found  This
//opening must also be checked  Thus when a test point encounters a wall, it uses the swop variables
//to check if this is the first or second time a wall is encountered  If it is the first, the co-ordinates
//are stored, if it is the second, a recursive call is made to explore the opening


do
        {
        tempx+=direction,
        uptempy=y+1,
        downtempy=y-1,
        front=dc->GetPixel(tempx+1,y),          //Find the colours in front and behind the current point
        back=dc->GetPixel(tempx-1,y),
        oldfront=front,
        oldback=back,
        if (front==wallcolour)      swop=1, else swop=0,
        if (back==wallcolour)       swop2=1, else swop2=0,
        oldswop=swop,
        oldswop2=swop2,

        do    //Move up from temp x,y loop
                {
                --uptempy,
```

```
test=dc->GetPixel(tempx,uptempy);                    //Test new spot
test2=dc->GetPixel(tempx+1,uptempy);                 //Test in front (test2)
if ((test2!=front)||((swop==2)&&(test==wallcolour))) //If the test is different to the
                                                     //last test or wall is reached...
        {
        ++swop;          //then add 1 to the swop variable and
        front=test2;                 //set last test colour =to current test colour
        if (swop==2)                 //If this is the first difference detected...
                {
                        newfronty=uptempy;      //then store the current x and y

                }
                else  if (swop==3)                   //else if this is the 2nd change...
                        {
                        if (abs(uptempy-newfronty)>(offset))   //check to see how
                                                               //big the gap is
                                {
                                Data pass;
                                if ((truedirection==1) ||
                                        ((truedirection==(-1)) &&
                                                (tempx<x1)))
                                        pass = Data(tempx+1,
                                                (uptempy+newfronty)/(float)2,
                                                0,x1,y1,direction,0,0);
                                        else pass=Data(tempx+1,
                                                (uptempy+newfronty)/(float)2,
                                                500,x1,y1,direction,0,0);
                                dc->MoveTo(tempx,uptempy+1);
                                dc->LineTo(tempx,newfronty-1);
                                algorithmBX(&recurs,&pass,offset,dc);
                                dc->SelectObject(&pen2);
                                dc->MoveTo(tempx,uptempy+1);
                                dc->LineTo(tempx,newfronty-1);
                                dc->SelectObject(&pen);
                                if (recurs.list_pointer()!=0) success=2;
                                }
                        if (front==wallcolour)      swop=1; else swop=0;
                        }
        }

test2=dc->GetPixel(tempx-1,uptempy);                 //Test behind
if ((test2!=back)||((swop2==2)&&(test==wallcolour)))
        {
        ++swop2;
        back=test2;
        if (swop2==2) newbacky=uptempy;
                else
                        if (swop2==3)
                        {
                        if (abs(uptempy-newbacky)>(offset))
                                {
                                Data pass;
                                if  ((truedirection==(-1)) ||
                                        ((truedirection==1)&&(tempx>x1)))
                                        pass=Data(tempx-1,
                                        (uptempy+newbacky)/(float)2,
                                        0,x1,y1,direction,0,0);
                                else pass=Data(tempx-1,
```

```
                                        (uptempy+newbacky)/(float)2,
                                        500,x1,y1,direction,0,0),
                                dc->MoveTo(tempx,uptempy+1),
                                dc->LineTo(tempx,newbacky-1),
                                algorithmBX(&recurs,&pass,offset,dc),
                                dc->SelectObject(&pen2),
                                dc->MoveTo(tempx,uptempy+1),
                                dc->LineTo(tempx,newbacky-1),
                                dc->SelectObject(&pen),

                                if (recurs list_pointer()!=0) success=3,
                                }
                        if (back==wallcolour)       swop2=1, else swop2=0,
                        }
                }
        }
while ((test==backcolour)&&(!((tempx==x1)&&(uptempy==y1)))&&(success==0)),
if ((tempx==x1)&&(uptempy==y1)) //If end found then
        {
        success+=1,
        uptempy-=2,
        goto stop,
        }
front=oldfront,                 //Reset these values for down check
back=oldback,
swop=oldswop,
swop2=oldswop2,

do    //Move down from temp x,y loop
        {
        ++downtempy,
        test=dc->GetPixel(tempx,downtempy),
        test2=dc->GetPixel(tempx+1,downtempy),
        if ((test2!=front)||((swop==2)&&(test==wallcolour)))
                {
                ++swop,
                front=test2,
                if (swop==2) newfronty=downtempy,               //test front
                        else
                                if (swop==3)
                                {
                                if (abs(downtempy-newfronty)>(offset))
                                        {
                                        Data pass,
                                        if ((truedirection==1) ||
                                                ((truedirection==(-1))&&(tempx<x1)))
                                        pass=Data(tempx+1,
                                                (downtempy+newfronty)/(float)2,
                                                0,x1,y1,direction,0,0),
                                        else pass=Data(tempx+1,
                                                (downtempy+newfronty)/(float)2,
                                                500,x1,y1,direction,0,0),
                                        dc->MoveTo(tempx,downtempy-1),
                                        dc->LineTo(tempx,newfronty+1),
                                        algorithmBX(&recurs,&pass,offset,dc);
                                        dc->SelectObject(&pen2),
                                        dc->MoveTo(tempx,downtempy-1),
                                        dc->LineTo(tempx,newfronty+1),
```

```
                                        dc->SelectObject(&pen);
                                        if (recurs.list_pointer()!=0) success=2;
                                        }
                                if (front==wallcolour)      swop=1; else swop=0;
                                }
                        }

                test2=dc->GetPixel(tempx-1,downtempy);          //Test behind
                if ((test2!=back)||((swop2==2)&&(test==wallcolour)))
                        {
                        ++swop2;
                        back=test2;
                        if (swop2==2) newbacky=downtempy;
                                else
                                        if (swop2==3)
                                        {
                                        if (abs(downtempy-newbacky)>(offset))
                                                {

                                                Data pass;
                                                if ((truedirection==(-1)) ||
                                                        ((truedirection==1)&&(tempx>x1)))
                                                        pass=Data(tempx-1,
                                                        (downtempy+newbacky)/(float)2,
                                                        0,x1,y1,direction,0,0);
                                                else pass=Data(tempx-1,
                                                        (downtempy+newbacky)/(float)2,
                                                        500,x1,y1,direction,0,0);
                                                dc->MoveTo(tempx,downtempy-1);
                                                dc->LineTo(tempx,newbacky);
                                                algorithmBX(&recurs,&pass,offset,dc);
                                                dc->SelectObject(&pen2);
                                                dc->MoveTo(tempx,downtempy-1);
                                                dc->LineTo(tempx,newbacky);
                                                dc->SelectObject(&pen);

                                                if (recurs.list_pointer()!=0) success=3;
                                                }
                                        if (back==wallcolour)       swop2=1; else swop2=0;
                                        }
                        }
                }
        while ((test==backcolour)&&(!((tempx==x1)&&(downtempy==y1)))&&(success==0));
        if ((tempx==x1)&&(downtempy==y1))
                {
                success+=1;
                downtempy+=2;
                }
stop:   ++limit;
                if (!(((direction==1)&&(success==3))||((direction==(-1))&&(success==2))))
                        {
                        if (success==1) swop=0; else swop=2;          //Set success flag
                        if (direction==1) store.addline(
                                Line(tempx,downtempy-swop,tempx,uptempy+swop));  //Add data
                        else store.addline(Line(tempx,uptempy+swop,tempx,downtempy-swop));
                                //swop up and down data for opposite direction
                        if (success>=2)
                                {
```

```
                                store join_lists(recurs list_no(0)),     //join up the lists & add to list
                                ret->addlist(store),
                                for (int i=1, i<recurs list_pointer(),++i)
                                        ret->addlist(recurs list_no(i)),
                                }
                        }
                        else
                                {
                                ret->addlist(store),
                                for (int i=0, i<recurs list_pointer(),++i)
                                        ret->addlist(recurs list_no(i)),
                                }
        }
while ((limit<1000)&&(dc->GetPixel((tempx+direction),y)!=wallcolour)
                &&(success==0)),
if (success==1) ret->addlist(store),
if (limit>=1000)
        {
        AfxMessageBox("Limit error, check Algobx cpp",MB_OK,0),
        ret->delete_all(),
        exit(0),
        }
dc->SelectObject(pOldPen),
if (success==0)
        {
        ret->delete_all(),                      //If the end is not found, do not return data
        }
}
```

References

1, Gill, J M (1986) *International Survey of Visual Aids for the blind* Royal National Institute for the Blind

2, Dodds, Allan (1992) *Rehabilitating blind & visually disabled people, a psychological approach*

3, Dobree, John H , Boulter, Eric (1982) *Blindness and visual handicap, the facts*

4, Herbert L Pick, Jr *Perception, Locomotion, and Orientation* , pp 73-89, Welsh, R L , Blasch, B B editors (1980) *Foundations of Orientation and Mobility*

5, Herbert L Pick, Jr *Tactual and Haptic Perception,* pp 89-115, Welsh, R L , Blasch, B B editors (1980) *Foundations of Orientation and Mobility*

6, William R Wiener *Audition,* pp 115-187, Welsh, R L , Blasch, B B editors (1980) *Foundations of Orientation and Mobility*

7, Billie Louise Bentzen *Orientation Aids,* pp 291-357, Welsh, R L , Blasch, B B editors (1980) *Foundations of Orientation and Mobility*

8, Leicester W Farmer *Mobility Devices* pp 357-412, Welsh, R L , Blasch, B B editors (1980) *Foundations of Orientation and Mobility*

9, Rieser, Lockman and Pick, (1976) *The role of visual experience in spatial representation* Perception and Psychophysics , 19, *117-121*

10, Brabyn, J A , Strelow, E R (1977) *Computer-analysed measures of human locomotion and mobility* Behaviour Res Methods & Instrumentation, 9

11, Barth, J L (1978) *The effects of preview constraint on perceptual motor behaviour and stress level in a mobility task*

12 , Airasian, P (1973) *Evaluation of the binaural sensory aid* AFB research bulletin, 26, 51-71

13, Darling, N , Goodrich, G , Wiley, J (1977) *A follow-up survey of electronic travel aid users* Bulletin of Prosthetics Research, 10, 82-91

14, Morrisette, D , Goodrich, G , Hennessey, J (1981) *A follow-up survey of the Mowat Sensor's applications, frequency of use, and maintenance reliability,* Journal of Visual Impairment & Blindness, 75, 244-247

15, Simon, E (1984) *A report on electronic travel users Three to five years later,* Journal of Visual Impairment & Blindness, 78, 478-480

16, Blasch, B B , Long, R G , Griffin-Shirley, N (Nov 1989) *Results of a National Survey of Electronic Travel Aid use* Journal of Visual Impairment & Blindness

17, Kallewaard, L (1993) *Direction finding using Antenna Arrays* D C U Internal Report

18, Microsoft Visual C++ Development Books

MFC reference guide,

C++ reference guide