Design and Development of an Autonomous Duct Inspection and Mapping Robot

Tracy Booysen

November 6, 2008

Abstract

Just a few years ago, the idea of having robots in factories and households was science fiction. But, as robotic technology develops, this is becoming reality. Nowadays, robots not only perform simple household chores, but are used in most production lines and are even employed by the army. Visual inspection robots are very common and are used in many industries, including inspecting the interior of duct systems.

Duct systems are in place in almost all large buildings and require ongoing maintenance and cleaning. Systems that are not properly maintained can pose a health risk as dust and mold form and are then blown throughout the building. In some cases, access holes have to be cut to allow access for inspection to occur. A robotic system, small enough to enter a duct through any existing access panel, would be advantageous. An autonomous robot would be even more useful as no operator would be needed thus reducing operating costs.

To this end, a robot was developed that could autonomously navigate through a duct system, recoding video images and mapping the internal profile. The development of which is discussed in this thesis, included the design of the robotic platform, the inclusion of appropriate sensors and accompanying circuitry, generation of a simulation to test the control algorithm and implementing embedded software to control the robot.

From the testing of the entire system the following conclusions were drawn. The robot as a whole performed well and navigated autonomously through the duct with a success rate of 90%. The system tests were repeatable and the odometry data closely matched the actual paths for straight line travel. The sonar data closly corresponded to the duct walls but was hard to interpret when the odometry and actual paths diverged. These paths diverged from each other due to wheel slip caused as the robot turned. The simulation developed showed that the control algorithm would ensure that the robot recursively inspected any duct system and provided information about the system as a whole.

Further work should concentrate on improving the correlation between the odometry path and the actual path, perhaps by adding in a bearing measurement system. Sensors with greater range and accuracy should be implemented and the entire system re-tested. The embedded controller allowed for expansion should additional requirements be needed and was more then adequate for the task.

Acknowledgments

I would like to thank the following people for all the support and help that they have provided during this project

My supervisor, Stephen Marais, for all his help and technical assistance, including his help in securing funding without which I would not have been able to complete this project

My research group, including Kate McWilliams, Sally Levesque and Hugo Krynauw for allowing me to bounce ideas off them and for entertaining me during tea time.

Ernesto Ismail, who's knowledge of vector use in Matlab made my life easier.

My parents, for their continued support during my extended studies.

The department of Mechanical Engineering, the University of Cape Town and the National Research Foundation for providing funding.

Opinion expressed and conclusions arrived at in this dissertation are those of the author and are not necessarily to be attributed to the NRF.

Declaration

I declare that all this dissertation contains only my work and that each significant contribution to, and quotation in this report from the work, or works, of other people has been attributed and has been cited and referenced. The IEEE convention for citation and referencing has been used.

I have not allowed and will not allow anyone to copy my work with the intention of passing it off as his or her own work.

Tracy Denise Booysen

Signed: Signed by candidate

Date:

Contents

1 Introdution **1**

2 Background Information **6**

2.1 The Need for Duct Inspection and Cleaning 6

2.2 Proximity Sensors 7

2.2.1 Ultrasonic Transducers 8

2.3 Localisation 9

2.3.1 GPS 10

2.3.2 Dead Reckoning and Optical Encoders 11

2.4 Recursive Exploration 11

2.4.1 Wall Following Algorithm 11

2.4.2 Recursive Exploration 12

2.5 The Embedded Controller 13

2.5.1 The Gumstix 13

2.5.2 The Roboaudiostix 14

2.6 Concluding Remarks 14

3 System Overview **16**

3.1 Introduction 16

3.2 Final System Specifications 19

3.3 Hardware 19

3.3.1 Platform 19

3.3.2 On-Board Cameras and Surrounding Equipment 22

3.3.3 Drive System 22

3.3.4 Ultrasonic Transducers 24

3.4 Power Distribution 25

3.4.1 Tether Power 25

3.4.2 Battery Power 26

3.5 Embedded Controller and Software Design 26

3.5.1 Motor Control 28

3.5.2 Ultrasound Readings 28

3.5.3 I^2C Communication 29

3.5.4 Data Storage and Transfer 29

3.5.5 External Control of the Robot 30

3.6 Summary 30

4 Simulation **31**

4.1 Introduction 31

4.2 Kinematic Simulation 32

4.3 Sensor Simulation 33

4.4 Wall Simulation 33

4.5 Control Algorithm Development 33

4.5.1 The Forward Control Algorithm 34

4.5.2 The Recursive, Reverse Control Algorithm 34

4.6 Simulation Results 36

4.7 Summary 42

5 Testing **43**

5.1 Speed Control 43

5.1.1 Verifying Accuracy of Runs 44

5.1.2 Choosing K_P 44

5.1.3 Matching K_P for both motors 45

5.2 Front Camera Test 46

5.3 Localisation and Ultrasound Testing 48

5.4 Stability of Speed Control and Repeatability of the System . . 51

5.5 Autonomous Branch Recognition and Turning 52

5.6 Map Generation and Branch Location 54

5.7 Autonomous Navigation and Mapping 55

5.8 Summary 59

6 Concluding Remarks and Recommendations 60

6.1 Concluding Remarks 60

6.2 Recommendations for Future Work 62

A Literature Survey A-1

B Simulation B-1

C Ultrasonic Transducer C-1

D Speed Control D-1

List of Figures

1.1 The Inside of a Duct 2

1.2 Two Commercially Available Duct Cleaning Robots 3

2.1 Hospital Duct Before and After Cleaning 7

2.2 Various Duct Cleaning and Inspection Methods 7

2.3 Regions of Constant Depth 9

2.4 The Gumstix and Roboaudiostix 13

3.1 Overview of the System 17

3.2 Interior Components 18

3.3 Front View of the Final System 18

3.4 External Structure of The Robot 21

3.5 ProE Screenshot of Base of the Robot 21

3.6 ProE Rendering of the Pan-Tilt Mechanism 23

3.7 A Schematic of the Circuitry and Hardware Incorporated in the Drive System 23

3.8 A Schematic of the Circuitry and Hardware Incorporated in the Ultrasonic System 24

3.9 Tether Power 25

3.10 Battery Power 26

3.11 Overview of the Control 27

3.12 The Roboaudiostix, Gumstix and NetMMC Stack 27

4.1 A Graphic Representation of the Robot's Variables 32

4.2 The Basic Algorithm Flowchart 35

4.3 The Recursive Algorithm Flowchart 37

4.4 The Forward and Reverse Algorithm 38

4.5 Reversing into the Recognised Duct and Exploring 39

4.6 Calling the Reverse Algorithm Recursively 39

4.7 Returning Home 40

4.8 The Actual Map Vs the Sonar Map 41

4.9 The Branch Centroids as Seen by the Platform 41

5.1 The Proportional Control Loop 44

5.2 Four Runs Using $K_P = 1$ 44

5.3 Altering K Using the Same Set Speed and Motor 45

5.4 Response of Both Motors Using $K_p = 0.25$ 46

5.5 Images of the Position of the Camera and the Robot 47

5.6 Images of the Duct Using the Forward Facing Camera 47

5.7 The Original and Processed Image 48

5.8 Video Stills Showing Actual Path 49

5.9 Comparison of Actual Path Vs Dead-Reckoning Path 50

5.10 Wheel Speeds 50

5.11 Sonar Data Using One Front Sonar 51

5.12 Three Straight Line Tests in the Same Duct 51

5.13 Left Sonar Readings While Navigating Around a Corner . . . 53

5.14 Right Sonar Readings While Navigating Around a Corner . . . 53

5.15 Left and Right Wheel Velocities While Navigating Around a Corner 54

5.16 Odometry and Sonar Readings Compared to the Actual System 55

5.17 The Paths During an Autonomous Test 56

5.18 Sonar Data Compared to the Actual Map and Odometry Path 57

5.19 Robot Position Before and After the Turn 58

5.20 Sonar Readings and Paths After 90° Fix 58

5.21 Sonar Readings and Paths After Positional Fix 59

A.1 Danduct's Range of Inspection Robots A-3

A.2 The Rovver Series A-5

A.3 Versatrax 100 . . . . . A-5

A.4 Nanomag A-6

A.5 CY-Bot Showing Air Pistol A-7

A.6 D-Link DCS-900 . . . . . A-8

B.1 A Graphic Representation of the Model's Variables B-7

B.2 A Graphic Representation of the Sonar Variables B-11

B.3 Sonar Setup B-12

B.4 Effects of Beam Angle on Visibility B-13

B.5 Pictoral Explanation of rap and ral B-14

B.6 Sonar Derivation Checks B-15

B.7 Explanation of the Distance Calculation B-16

B.8 Possible positions of *rap* B-17

B.9 The Sonar Definition B-19

B.10 The Position of the Bump Sensors B-21

B.11 The Basic Algorithm Flowchart B-26

B.12 Simulation Results for the initial algorithm B-27

B.13 The Recursive Algorithm Flowchart B-28

C.1 Sonar Connection Schematic C-4

C.2 Oscillator Circuit Diagram C-5

C.3 Gain Stage Circuit Diagram C-6

C.4 Peak Level Detector Circuit Diagram C-7

C.5 ADC Results for an Object at a Constant Distance C-8

C.6 Return Signal As the Angle of Incidence Increases C-9

C.7 Sonar Reading for a Moving Sonar C-10

C.8 Sonar Response of To Duct Branch C-10

D.1 The Control Algorithm D-3

D.2 Overview of the Drive System D-4

D.3 Optcoupler Circuit Diagram D-5

D.4 Optical Switch Circuit Diagram D-5

D.5 Four Runs Using $K_P = 1$ D-7

D.6 Altering K Ssing the Same Set Speed and Motor D-8

D.7 Response of Both Motors Using $K_p = 0.25$ D-9

D.8 Comparison of Other K_p Values D-10

Glossary of Terms

ADC	Analogue to Digital Conversion (Converter)
CNC	Computer Numerical Control
DAC	Digital to Analogue Conversion (Converter)
GPIO	General Purpose Input Output
HDPE	High Density Polyethylene
HVAC	Heating, Ventilation and Air Conditioning
I^2C	Inter-Integrated Circuit
Pose	Position and orientation
PWM	Pulse Width Modulation
RCD	Regions of Constant Depth
TOF	Time of Flight

List of Symbols

θ	The angle of the robot
Δt	The change in time
C	The vector containing the co-ordinates of the centroid
L	A vector containing the co-ordinates of the left wheel
R	A vector containing the co-ordinates of the right wheel
R_s	The range of the sonar
$\vec{ral}$	The line which is at 90° to the wall and bisects the sonar cone
rap	The point of intersection between $\vec{ral}$ and the wall
S_L	The vector containing the displacements of the left wheel
S_R	The vector containing the displacements of the right wheel
V_R	The vector containing the velocitie of the right wheel
V_L	The vector containing the velocitie of the left wheel
W	Width of the robot

Chapter 1

Introdution

As robotics components become more increasingly affordable, and as the technology becomes common place, the idea of using robotic platforms to perform everyday tasks is becoming a reality. What would once be considered science fiction is being implemented in the world today. Robots are not only being used to perform simple tasks such as household chores, but have found use in many situations where human involvement could result in the loss of life.

Visual inspection robots have been used in many applications, including during the September 11 attacks, where they were used to look for the injured in zones that were deemed unsafe for human rescuers. However, because navigating around obstacles and processing visual data seems simple to humans, we tend to take for granted that it is a simple task to implement in robots. It is not, in fact, an easy task to implement, especially when the object under consideration is composed of irregular, rounded surfaces.

The maintenance of air-conditioning ducting in buildings is an ongoing and complex task. The maintenance tasks that have to be performed include:

the removal of dust build up and foreign artifacts, the detection and repair of leaks and visual inspection of the condition of the duct. The internal size of most ducts makes direct human access impossible, creating a prime candidate for a robotic inspection system. *Figure* **??** shows how a duct may become clogged over extended periods. Clogged ducts not only reduce the efficiency of the ventilation system but also spread bacteria and other germs through the entire building. This can cause sick building syndrome and is particularly harmful in hospitals and other clean environments.

Figure 1.1: The Inside of a Duct **?**

There are many robots currently available for the inspection and cleaning of ducts. These robots are tethered and not autonomous, which severely restricts their performance. The lack of autonomy in these robots turns a relatively simple task into a time consuming one. An operator must be present at all times to direct the robot and to prevent damage to both the robot system and the duct. Two commercially available duct cleaning robots are shown in *Figure* **??**. The robot on the left is the *Rovver 400* and on the right is Cyclone Industry's *Cy-Bot*.

The proposal is to design and build an autonomous duct inspection robot which will steer itself through a duct, recording images as it moves along. It will recursively navigate through the duct until the entire system has been explored. The ducts' internal profile will be captured with cameras to allow for visual inspection, while the general outline and branches in the duct are recorded using a sensor system.

Figure 1.2: Two commercially available duct cleaning robots **??**

Autonomy is an ever growing field in robotics. Increased autonomy reduces operator hours thus reducing operating costs. For a robot to be called autonomous, it must be able to operate independently in an unstructured and possibly cluttered environment. Little or no *a priori* information should be available to the robot. J.Velagic *et al* **?** state: "To achieve this level of robustness, methods need to be developed to provide solutions to localization, map building,planning and control" with regard to autonomy.

For any of the above problems to be solved, sensors that allow for meaningful information to be extracted from the environment are required. In the system presented, mapping and navigation are made possible by the addition of six ultrasonic sensors. Sonar is affordable, relatively easy to control, cost effective and the technology has been known for a long time. However, several disadvantages do exist; they have poor directionality; are prone to misreadings and the quality of the reading depends heavily on the accuracy, range and cone angle of the sonar **??**.

Since mapping was not the primary function of the robot, and the main use of the sonars was to detect the presence and/or absence of the duct walls, many of these problems were not as severe or even present in this application. When considered with the fact that duct walls are smooth and uniform, the data generated using ultrasonic transducers was considered to be sufficient for the task.

To control the robot a suitable controller had to be selected. This controller needed to be sophisticated enough to perform high level tasks like navigation and path planning, but also be able to control the low level tasks like the movement of the robot and sensor readings. A Gumstix embedded controller with Roboaudiostix microcontroller **?** was chosen to perform the control. The system performed all the tasks required of it, but had the capability for expansion should further work or upgrades to the system be required.

Since the only method of localisation available to the robot was the dead-reckoning approach, speed control was added to the system. This allowed for odometry data to be collected and was used to ensure that the motion of the robot matched that required by the navigation algorithm.

Supporting circuitry for motor and sonar control had to be developed. Additional software to test and calibrate each of these subsystems was written.

Two cameras were included; a forward facing camera which included a pan and tilt mechanism designed around the camera housing and a rear camera. Both of these cameras streamed images over TCP/IP which could be stored or viewed in real time.

A control software verification simulation was developed to allow for minimisation of external errors. This simulation included realistic kinematic, sonar and wall interference models. These models could be modified to simulate various environmental factors the results in the performance were noted.

This report begins with background theory relevant to this work and then continues with a detailed discussion of the final solution. Further sections describe of all the on-board hardware, the design decisions made, the embedded controller and associated software design and finally an explanation of the simulation. This is followed by an explanation of the tests performed and concluding remarks and recommendations for future work are then drawn.

Technical details are provided in the appendices.

Chapter 2

Background Information

2.1 The Need for Duct Inspection and Cleaning

During air circulation, contaminants including dust, lint and dust mites enter duct systems through vents. Since ducts are often damp and warm, they make an ideal place for micro-organisms to breed. When **H**eating, **V**entilation and **A**ir **C**onditioning (HVAC) systems are contaminated, the bacteria and mold circulate around the building with the air. This can cause sick building syndrome, where many of the occupants of the building become sick at the same time. This is of particular concern in health care facilities where patients have compromised immune systems. Performance levels and efficiency decrease in a blocked duct, increasing operating costs. *Figure* **??** below shows the interior of a hospital duct before and after cleaning and the drastic effect of cleaning can be seen.

Ventilation duct inspection combines analysing air samples and visual in-

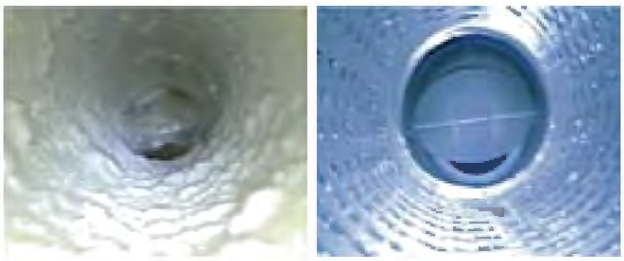

Figure 2.1: Hospital Duct Before (left) and After Cleaning (right) **?**

spections via any inspection hatches available. Inspections are also done at the incoming and outgoing air valves. In some cases the duct is inspected along the whole length. The ducts are cleaned by many methods, some of which include, using a brush on a flexible hose, blowing compressed air at high velocity through the system and by dry ice blasting. Some systems need to have permanent access doors installed to allow inspection to take place. *Figure* **??** shows a common cleaning brush (left) and a small camera attached to a long tether used for inspection (right).

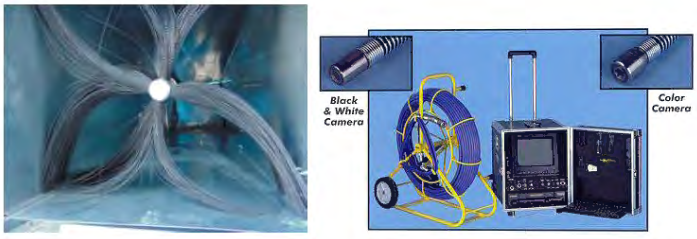

Figure 2.2: Various duct cleaning and inspection methods **??**

2.2 Proximity Sensors

Proximity sensors are needed to detect the presence or absence of duct walls, but could also be used to generate a map of the environment. This allows for appropriate robot behaviour at duct branches and terminations and avoided damage to the robot.

A minimum of three sets of sensors were required, for the left, right and front of the robot. A sensor set at the rear would not be required if all exploration was done while moving forwards. Many types of sensors are available, but only a few could be used in this application.

Light sensors are easy to implement and are economical. However, several problems with these sensors exist. Light sensors rely on reflected light and this reflection may be compromised should the duct widen or should the surface's reflectivity be affected by the dirt in ducts.

Laser scanners are the most accurate sensors available but are prohibitively expensive. Cameras have the advantage of high resolution but require a high level of computation to extract proximity data.

2.2.1 Ultrasonic Transducers

Ultrasonic transducers are widely used in mobile robotics as they provide range information and are affordable compared to other distance measuring devices such as laser range finders **??**. The technology is well developed and control is easy to implement. Sonars measure over a cone shaped area and are thus unlikely to miss small objects, which can be a common problem in laser based solutions. Measurement is rapid because sonar is governed by the speed in sound in air and not slower mechanical movement.

There are, however disadvantages when using sonars. Data can be inaccurate and circuitry has to be developed to compensate for this. They have poor directionality, frequent misreadings, are very sensitive to specular reflections **??** and have large amounts of noise superimposed on the data. The directionality is poor because when a return signal is interpreted, the simplest sonar model assumes that the object lies along the centre line of the cone,

regardless of the actual position. As a result, a single sonar cannot determine the orientation of an object. The reflected sound wave is dependent on the specularity of the surface. Almost perfect wave reflection will be achieved on a specular surface (one that reflects perfectly like a mirrors). A diffuse surface is uneven and will scatter sound at a number of angles.

The angle of incidence of the sound cone also plays an important part in the sensing of objects, too large an angle results in the beam reflecting away from the receiver and no return received. Another documented effect of the beam width is Regions of Constant Depth (RCD's) **??**. No matter what the angle of incidence to a flat wall, the return is always assumed to lie along the centre of the sonar cone. Because the distance to the wall remains constant, a series of curved surfaces are produced instead of a flat wall. The curved surfaces are called Regions of Constant Depth and can be seen in *Figure* **??**. Point A is the area that the sonar detects, but since the simplest sonar model assumes that the point seen lies along the centre line of the sonar cone, point B is produced. This creates the arc, generating the region of constant depth. Sonars are thus not usually used to create detailed maps, but rather to determine the presence of objects **?**.

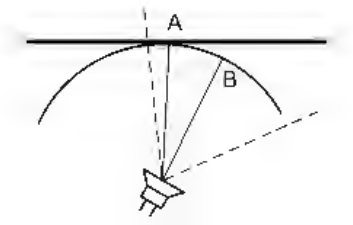

Figure 2.3: Regions of Constant Depth

2.3 Localisation

For any mobile robot that requires autonomous operation, some method of localisation is necessary. A robot vacuum cleaner needs to know where it has

been to avoid re-cleaning and a duct cleaning robot must be able to navigate through the duct, coming back to sections or regions that it has not yet explored. Most mobile robots need to know their position and orientation relative to their starting point. This is a simple problem if a of global beacon or sensor allowing localisation is available. A simple example of a global sensor is a number of beacons all sending out a sonar pulse at different frequencies. The robot determines the orientation and strength of each pulse and thus the position can be inferred. Since no objects can be placed in the duct, another form of localisation had to be found. The first method of obtaining localisation information to be investigated was GPS.

2.3.1 GPS

GPS or **G**lobal **P**ositioning **S**ystem is used to calculate latitude, longitude, altitude, speed and direction. There are more then 24 satellites in orbit providing information to GPS receivers. The position of the GPS receiver is determined by using at least three satellites. The calculation of latitude, longitude and altitude can be calculated from the distance between the ground receiver and each of the satellites. Once the receiver determines its position it can calculate the speed and heading of the object based on the change in position. GPS receivers are not completely accurate and a conventional signal may only be accurate within several meters.

Antaris **?** have developed a GPS receiver designed specifically for robotics to work in obstructed environments but the signal would not penetrate a duct wall, and building roof.

2.3.2 Dead Reckoning and Optical Encoders

Since it was not possible for any type of global sensor to be implemented, optical wheel encoders were investigated. Using an affordable sensor and slotted disk on the wheel, displacement can be calculated. Global position can be calculated by adding wheel displacements to the previous known position. Even though these sensors are affordable and easy to implement, this method, termed dead reckoning can be very inaccurate. The largest cause of error is wheel slip which cannot be measured by the encoders. The errors induced by optical encoders accumulate over time and orientation changes are particularly affected **??**. It is for this reason that some platforms have stationary beacons or areas of known position which allow it to update its pose.

2.4 Recursive Exploration

Maze exploration robots have been developed since the start of the Micro Mouse Contest **?** which began in 1977. The contest set out a maze with a defined goal that the robots had to reach. A ducting system is very similar to a maze in that they are both comprised of adjoining branches and straight edges. It is for this reason that maze exploration was considered.

2.4.1 Wall Following Algorithm

A very simple and very effective maze exploration algorithm is a simple wall following algorithm. The robot to senses intersections in the maze and always follows a particular wall. Consider the case where the robot first follows the leftmost wall [**?**, pp219-221]. It will attempt to do this until it is not possible

it may try to drive straight and finally to the right until it reaches the goal. Depending on the position of the goal, the algorithm may not explore the entire maze (therefore an incomplete map will be generated) but it can also develop problems when the goal is placed in the center of a room with wall on all sides **?**. The robot will then follow the wall continuously and never reach the goal.

2.4.2 Recursive Exploration

This algorithm requires that all regions that can be reached be visited, regardless of the maze construction [**?**, pp232-225]. To do this one must know which squares have already been visited and generate an internal representation of the maze. The algorithm is broken down into three stages.

- Explore the whole maze
- Compute the shortest distance from the start square to each reachable square
- Allow the user to enter the co-ordinates of any reachable area on the maze and have the shortest path to the goal calculated

The main difference between this algorithm and the wall following algorithm is the path followed. The wall follower will only take a single path, whereas the recursive algorithm will take each possible path to fully map the maze. The path taken by the wall following algorithm depends on the direction it prefers and this may lead to a longer path to the goal then necessary **?**.

The robot determines the presence of walls to the front, left or right of it and marks the square as visited on its internal map. If this is not done then

the robot may continually repeat a path. The number and position of the walls are recorded in the internal representation of the maze. If all three directions are open and the next square has not yet been visited then the robot will choose a direction and continue until the path terminates. It will then return to the square where the branch or junction occurred and choose the next direction. This is done recursively until the entire map is visited. Once each square has been visited and the internal map has been generated, the distance of each square from the start square can be calculated if desired.

2.5 The Embedded Controller

The operation of the robot is performed using an embedded controller, which was selected based on the required tasks. It needed to perform both the high-level tasks, such as localisation and path planning, but also the low-level tasks such as motor control and sensor manipulation.

2.5.1 The Gumstix

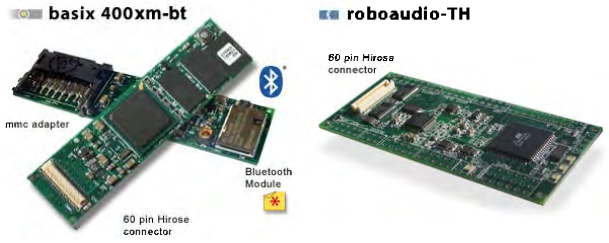

Figure 2.4: The Gumstix and Roboaudiostix **?**

The Gumstix is a relatively new product on the market. Its name derives from the size of the product as it is about the same size as a stick of gum.

It is similar to the PC-104 as modules can be stacked on top of it, adding to its functionality. Unlike the PC-104, the Gumstix can only have a limited amount of components connected to it. The Gumstix itself is a small computer that runs the Linux kernel and has many uses. It is available with either Bluetooth or ethernet to communicate with a remote host computer. The Gumstix itself cannot perform low level tasks such as motor control, however one of the external components that can be stacked onto it is the Roboaudiostix which was designed for robotic applications. Both these components can be seen in *Figure* **??**.

2.5.2 The Roboaudiostix

The Roboaudiostix has been designed for robotic control and has the correct specifications for this task. It has four **P**ulse **W**idth **M**odulation (PWM) channels, the capability to perform both analogue to digital conversions and digital to analogue conversions and has many general purpose input/output ports which can be used to read and control external sensors. Since the sonar transducers operate at 40kHz if the data is to be read into the Roboaudiostix it must have a conversion rate of at least 80kHz **?**. One of the advantages of both their Gumstix and the Roboaudiostix is the low power consumption and small size. The low power will increase the battery life of the system while the small size will reduce the overall size of the platform.

2.6 Concluding Remarks

Duct inspection is a task particularly suited to robots. This is even more so if the robot is autonomous. The use of a small robot would allow for inspection without the cutting of access holes. The robot designed for this task could be easily modified to include cleaning mechanisms.

Sonar was economical and easy to implement and provided information about the presence of duct walls, but also range information as well. This could be used in the control algorithm and in the generation of simple maps.

Since GPS was too unreliable to implement given the nature of the environment, dead reckoning was used as the only form of localisation implemented. Speed control was be implemented to reduce the errors as much as possible.

The small size, low power consumption and expansion capability of the Gumstix and Roboaudiostix combination make them well suited for this application. All required tasks can be performed, including easy transmission of commands and data via TCP/IP. Additionally, the system capability can be easily expanded.

Exploration had to be recursive to ensure the entire duct system was explored. The algorithm had to be verified in a simulation prior to being implemented on the robot.

Chapter 3

System Overview

3.1 Introduction

The development of the physical system followed top-down approach. Hardware was selected to achieve the required mobility, while the supporting software was designed concurrently. This was done to allow hardware choices to be guided by the requirements of the developing software. The software required inputs from the hardware to allow for autonomy, hence the concurrent design. An overview of the physical system can be seen in *Figure* **??** while the final system is shown in *Figures* **??** and **??**. Further details of the physical system are described below, with focus on the mechanical components, the power distribution and the embedded software design.

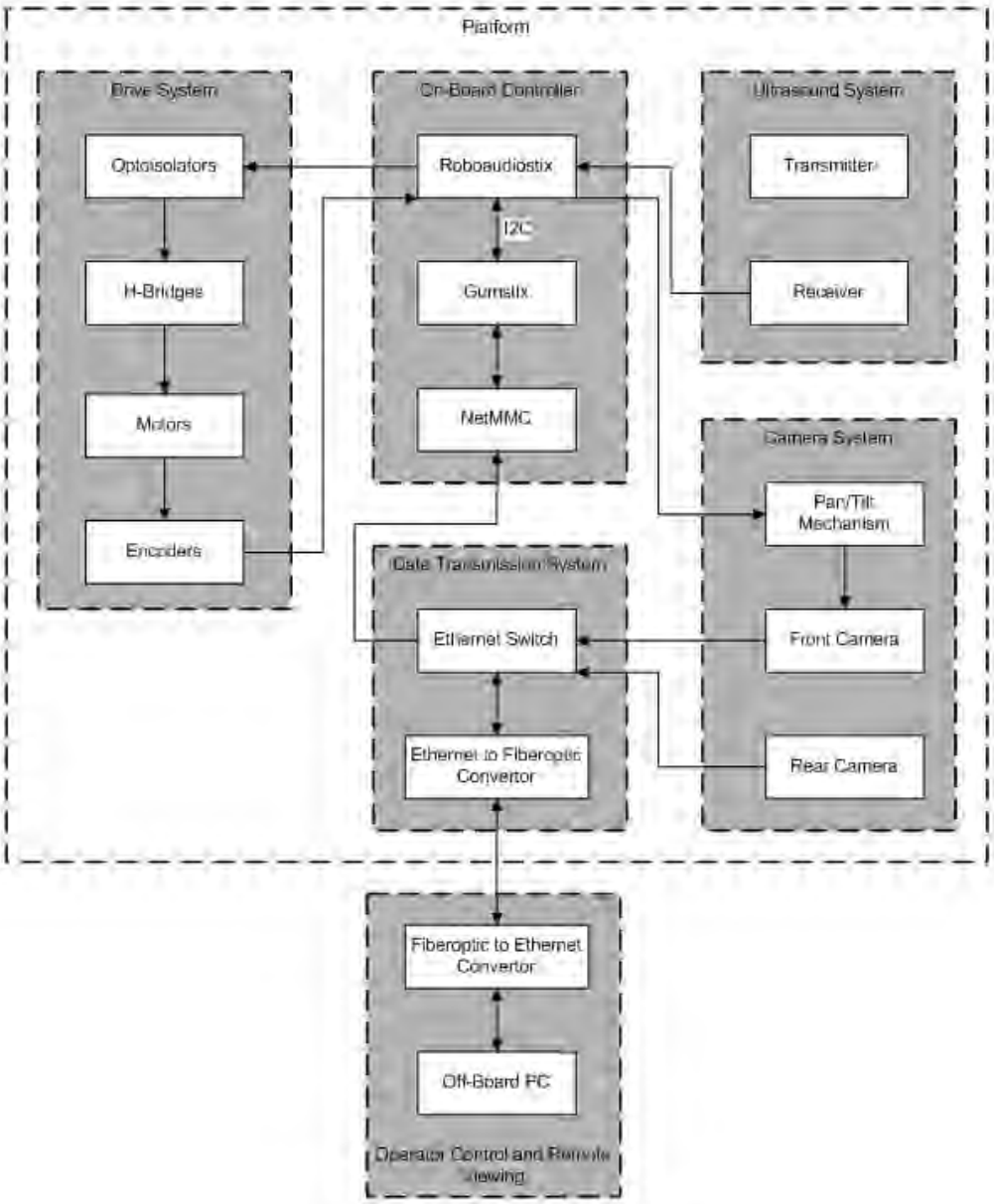

Figure 3.1: Overview of the System

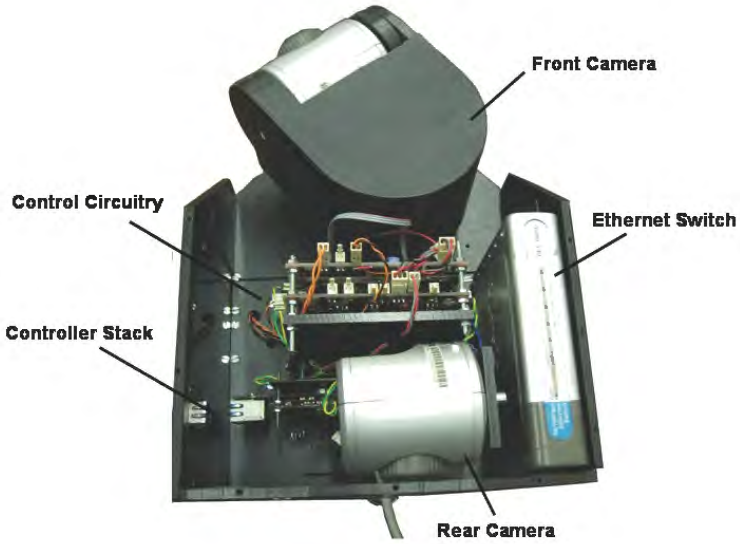

Figure 3.2: Interior Components

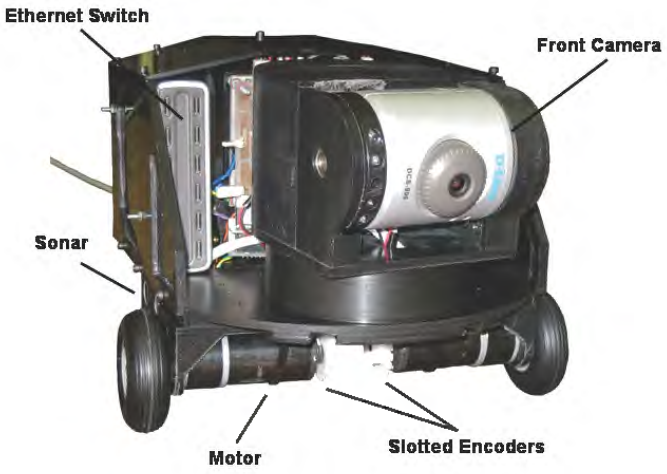

Figure 3.3: Front View of the Final System

3.2 Final System Specifications

- 200mm wide, 280mm long, 150mm high
- 2 independently controlled 15V geared motors, each controlling a single wheel
- Three independent sonar sets mounted on the front, left and right sides
- 20m fiberoptic tether
- Gumstix and Roboaudiostix controller with NetMMC for TCP/IP communication

3.3 Hardware

3.3.1 Platform

Design Constraints

The initial concept was to build a prototype which would carry all the sensors and control systems. This prototype would be able to navigate through a clean test duct in order to provide proof of concept for the final design. This was to be completed before any complex mechanical design to allow for operation in a more rough environment was carried out. Thus the main specifications for the mechanical design were size, weight and simplicity.

The base was designed with size minimisation as the main constraint. The smaller the platform the smaller the minimum duct size and the greater the variety of ducts that the robot could explore. The robot's chassis had to be

laid out in a manner that allowed for the required range of motion of the camera housing. The robot was dimensioned accordingly, with the chassis width determined by the movement envelope of the camera housing. This determined the minimum width of the platform.

Weight balance was crucial as the pan-tilt camera mechanism and the system's battery had a much greater mass when compared with the other components. Care had to be taken to ensure that the placement of these components with regards to the wheels was correct to prevent the platform from tipping or loading one wheel more then the other (which would result in unpredictable steering).

Since the operating environment was smooth and uniform, a complex design for the base was not needed. Instead, the robot was designed to be as simple as possible and thus made use of two independent motors for steering, with a jockey wheel for balance.

The platform was designed to be lightweight and modular in assembly and a cover was designed to reduce the amount of dust that entered the system.

Housing Design

Figure **??** shows the housing design. The two wheels with the encoders can be seen in the front while the jockey wheel was mounted at the rear end of the robot. The rear window was made out of clear perspex as the rear camera was placed behind the window and needed clear vision, while the side windows were made out of blue perspex for aesthetic reasons. All the windows were press fitted into grooves machined into the HDPE sides.

Figure **??** shows the underside of the base. For a platform of this size, a 5mm

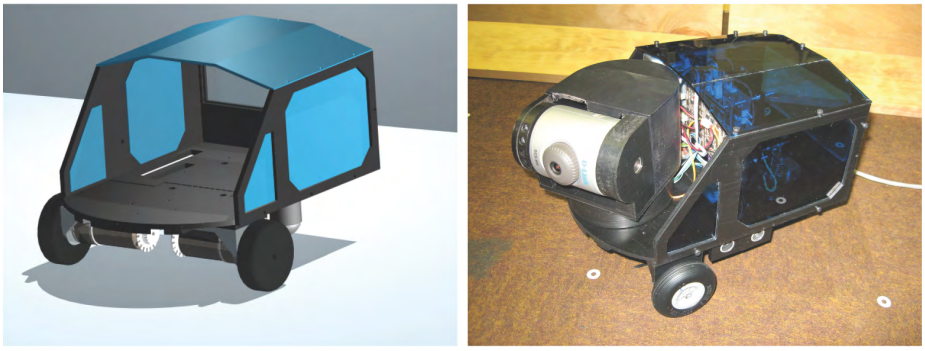

Figure 3.4: Exterior Structure: ProE Model, without Pan/Tilt Mechanism (left) Actual Design (right)

thick base would flex under load. A 5mm thick base with 5mm struts was machined using a CNC milling machine. This gave the base the structural integrity it required while decreasing the overall weight. Care had to be taken when placing the components to ensure that the mounting holes were placed either on or off the struts. The large solid strut in the front of the base was provided to support and mount both the motors. The openings in the base were to allow cable routing between the top and underside of the platform.

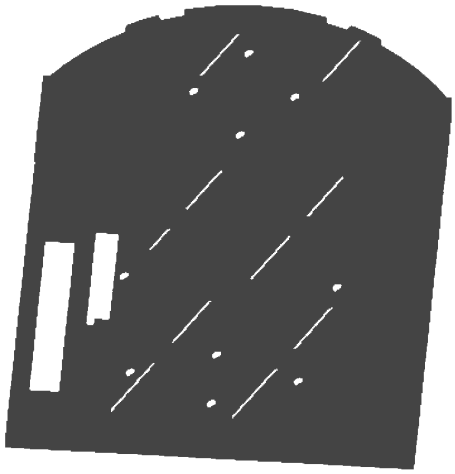

Figure 3.5: ProE Screenshot of Base of the Robot

3.3.2 On-Board Cameras and Surrounding Equipment

In *Figure* **??** most of the internal components can be seen. The fiberoptic to ethernet converter was mounted on the underside of the robot and is not visible in the figure. The battery enclosure, the fiberoptic enclosure and the circuitry mount were designed to be easily removed with the component it protects, which allowed for a modular assembly. The battery, which is also not visible in the picture, was mounted underneath the rear camera in a module designed for easy attachment should the battery need to be replaced.

Two cameras were mounted on the robot, one on the front, the other at the rear. A camera housing, shown in *Figure* **??**, allowing pan and tilt operation was designed for the front camera and was controlled using servo motors. The two servo motors and the four gears which drive the mechanism are shown in black and red respectively. Each of the cameras transmitted the images over an ethernet cable which was passed into a switch and then into an ethernet to fiberoptic converter.

The rear camera was mounted on the battery enclosure and the ethernet to fiberoptic converter was mounted inside an enclosure placed below the platform.

3.3.3 Drive System

Two independently controlled motors were mounted on the underside of the platform and were controlled using 12V PWM using an LMD18200 H-bridge chip **?**. The PWM signal was received from the Roboaudiostix and was passed through an optocoupler to ensure that the electronic noise from the motors would not affect to the more sensitive circuitry. A slotted encoder disk, mounted on the motor shaft was connected to a phototransistor optical

Figure 3.6: ProE Rendering of the Pan-Tilt Mechanism

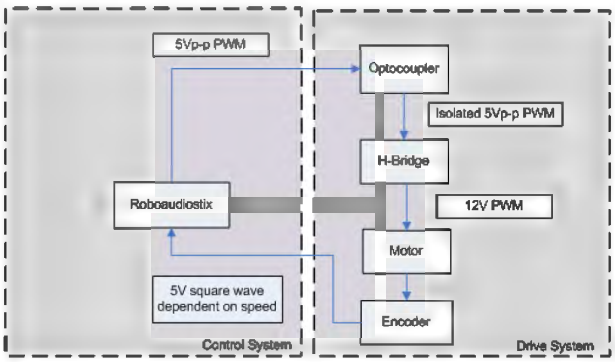

Figure 3.7: A Schematic of the Circuitry and Hardware Incorporated in the Drive System

interrupt switch **?** which was fed into the Roboaudiostix. An overview of the drive system can be seen *Figure* **??** while circuit diagrams and further information can be found in *appendix* **??**.

3.3.4 Ultrasonic Transducers

Ultrasonic transducers were added to the platform to detect the presence of duct walls and to provide information about the environment. *Murata* 40kHz ultrasonic transducers **?** were used, with separate transducers for transmitting and receiving the sonar signal. Each received signal was amplified, rectified and fed into the **A**nalogue to **D**igital **C**onvertor (ADC) port of the Roboaudiostix. Ultrasonic transducers were placed on the front, left and right of the platform.

Generally when sonars are implemented, Time of Flight is used. However, owing to the lack of timers on the Roboaudiostix and because great accuracy was not needed, a different technique was developed. This technique relied on the fact that the return of a sonar is a sin wave with varying amplitude which is dependent on range. This amplitude can be used as a range reading if it is amplified and rectified and sent into an ADC port. *Figure* **??** shows an overview of the signal manipulation for the sonars while all circuit diagrams can be found in *appendix* **??**.

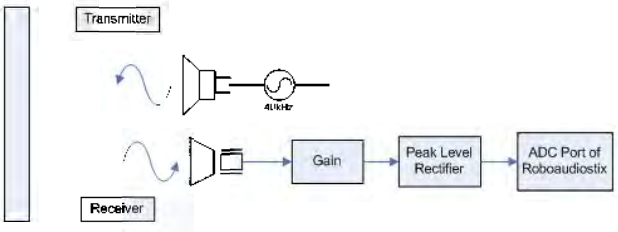

Figure 3.8: A Schematic of the Circuitry and Hardware Incorporated in the Ultrasonic System

3.4 Power Distribution

To completely isolate the motors from other electronics, two independent power sources were needed. The motors and H-bridges required 12V from the tether. All the other circuitry was run of the second power source, the on board battery. Since the noise from the motors could travel along power and ground lines, the ground and voltage lines for these sources were not allowed to come into contact.

3.4.1 Tether Power

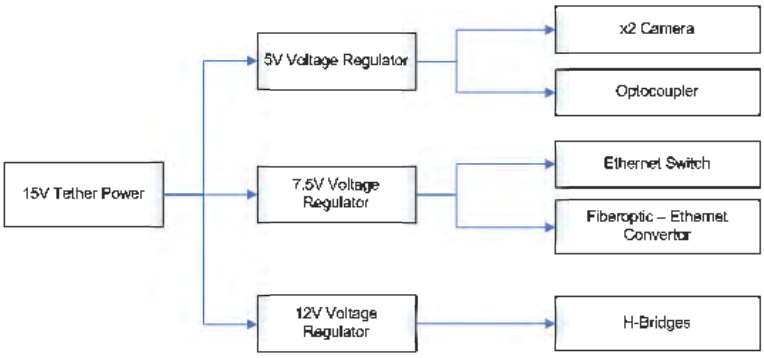

Figure 3.9: Tether Power

The H-bridges required a 12V supply, while the optocouplers and cameras needed 5V. Both the fiberoptic to ethernet converter and the ethernet switch needed 7.5V. Voltage regulators rated at 1A were used to provide all the required voltages and *Figure* **??** shows a power schematic for the tether.

3.4.2 Battery Power

The Gumstix, Roboaudiostix and NetMMC and sonar transmitter circuit all required 5V, while the sonar receiver circuit required +5V and -5V. For this reason the battery voltage was inverted using an ICL7660 **?** and then regulated to -5V. The optocouplers only required ground from the battery and a connection schematic can be seen in *Figure* **??**.

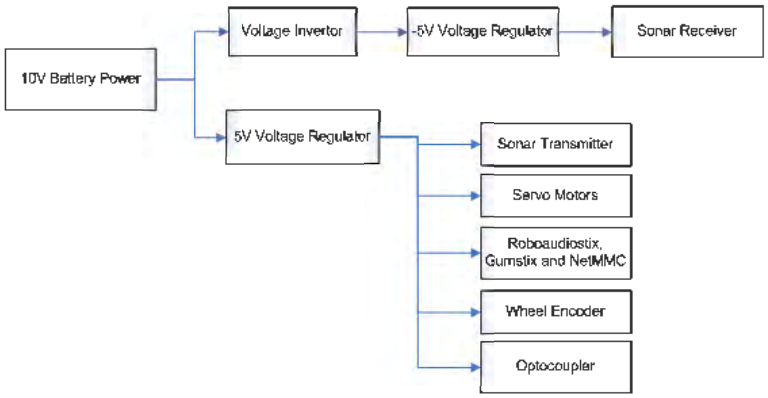

Figure 3.10: Battery Power

3.5 Embedded Controller and Software Design

There were three important sub-systems used in the control of the platform, the Roboaudiostix, the Gumstix and the off-board controller. The Roboaudiostix was responsible for the control of the motor drive and the sonar system described above. The Gumstix communicated with the Roboaudiostix and the off-board controller. It to performed the high-level control such as path planning, navigation and mapping. Attached between these two was the NetMMC which allowed communication between the Gumstix and off-board controller using the TCP/IP protocol. A schematic of the control system can

be seen in *Figure* **??**.

The control stack can be seen in *Figure* **??** below. The ethernet port of the NetMMC can clearly seen at the bottom of the stack, while the Roboaudiostix is on top. The Gumstix is not visible as it is in the centre of the stack.

The off-board controller was used in the initial testing phase to give instructions to and request data from the Roboaudiostix. The off-board controller also converted raw speed data into positional data to determine the path taken by the robot. Ultrasound data was transferred from the Roboaudiostix to the off-board computer and processed to create maps of the environment.

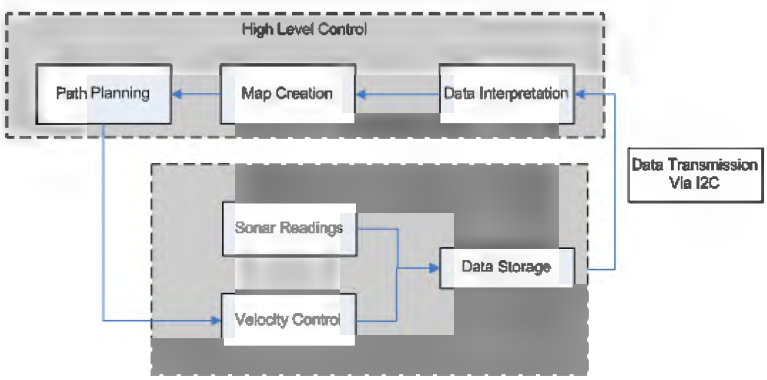

Figure 3.11: Overview of the Control

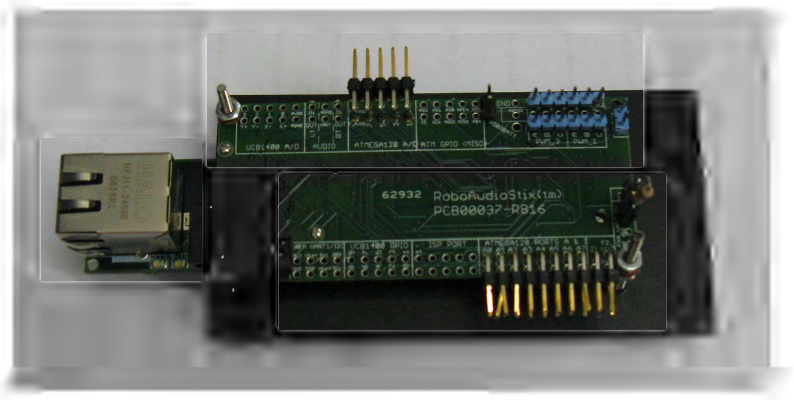

Figure 3.12: The Roboaudiostix, Gumstix and NetMMC Stack

3.5.1 Motor Control

A 16 bit dedicated PWM timer, set to 20kHz **?** was used for motor control, which provided independent PWM signals to each motor. These signals were fed through optocouplers to isolate the controller from any voltage or current fluctuations, and then into H-Bridges for current and voltage amplification. Further detail can be found in *appendix* **??**. To alter the motor speed and thus the duty cycle of the PWM, the value stored in the **O**utput **C**ompare **R**egister (OCR) was changed.

An additional 8 bit timer was used as an interrupt to perform the speed control. The signal from the encoders was fed into two external interrupt pins which allowed the total number of encoder slots that passed in a certain amount of time to be counted. Every 8 timer interrupts a proportional speed control loop was implemented and the wheel speeds recorded. This allowed for accurate speed control on the wheels.

3.5.2 Ultrasound Readings

The sonar system required two sets of circuitry, an oscillator circuit and a receiver circuit. The oscillator circuit was connected to every transmitter sonar and provided a 10V peak to peak, 40kHz signal. Each receiver signal was amplified and then rectified to provide a DC voltage which depended on the range of the object. Further information can be found in *appendix* **??**.

This ADC signal was fed into the ADC ports available on the Roboaudiostix and the ADC results were inversely proportional to the range, i.e. as the distance to the objects gets smaller, the ADC reading gets bigger. This was because the amplitude of the signal from the receiver became greater as the range decreased. The ADC data for each sonar differed slightly due to small

differences in components, so they were calibrated and then converted to a distance measurement.

Range readings were taken in two places in the software. They were taken inside the speed control interrupt routine as this is where all the data was stored. This had to occur simultaneously with the speed readings as the range data correlated directly to the pose of the robot. Readings were also taken more often inside the main control loop to increase the speed at which the robot responded to duct branches and terminations.

3.5.3 I^2C Communication

Communication between the Roboaudiostix and Gumstix allowed for data transfer, including sensor data and control commands. This was achieved by implementing the I^2C (**I**nter-**I**ntegrated **C**ircuit) protocol. Parts of the code was based on software written by Dave Hylands **??**, a Gumstix developer. Most of the I^2C and circular buffer code was left unchanged. Further code including some I^2C functions and the implementation of the circular buffer was implemented in collaboration with Kate McWilliams **?**. Additional I^2C functions for sensor data transfers and motion control were developed. All code used can be found on the CD that accompanies this report.

3.5.4 Data Storage and Transfer

For accurate mapping and navigation to occur, sensor data must be linked to the position and source of the reading. All of the sonar readings (front, left and right) and the wheel speeds are thus stored for later transmission to the Gumstix. The Roboaudiostix stored the speed and ultrasound readings in circular buffers, a data storage structure where the oldest data is progressively

overwritten with new data. The I^2C commands are uni-directional, with only the Gumstix able to define what is sent over the protocal. The Gumstix thus requests for data to be sent by the Roboaudiostix. This must be done more frequently then the data is added, to ensure that no data is lost. The data that was read was removed from the buffer. Once the Gumstix requested the data, a flag was set to signal that the buffer was empty and that the transfer was complete. The data was written to an external file for later processing.

3.5.5 External Control of the Robot

External commands could also be sent to the robot via the Gumstix during operation. These commands included; speed and directional information to the wheels, emergency stop commands and commands to control the pan/tilt mechanism of the front camera. These commands allowed the user to control the robot should a problem in the duct be spotted and a closer look needed.

3.6 Summary

This chapter has described the subsystems that make up the final solution. The mechanical, electronic and software implemented have been described and the operation of their components have been discussed. The chapter which follows deals with the simulation in greater detail and details the verification and results of a complex run.

Chapter 4

Simulation

4.1 Introduction

One of the objectives of this work was to gather experience in writing code and developing simulations. For this reason a custom simulation was developed instead of using an existing one.

In order to eliminate external factors that could cause unexpected behaviour, a control algorithm simulation of the ducting was developed using *MATLAB* **?**. The 'robot' was placed in the duct, which it interacted with. These interactions included contact with and sonar sensing of the duct walls. The robot was moved through the duct, by calculating displacement of the wheels. Parameters simulating various environmental conditions were included. A more detailed explanation of the simulation can be found in *appendix* **??**.

The behaviour of the robot in the environment was described in terms of pose. Time was discretised and at each time interval an updated pose was

calculated. This pose was determined by the control algorithm or by interaction with the environment. The robot's pose was described by its centroid C $\begin{bmatrix} x \\ y \end{bmatrix}$, and by its angle θ. *Figure* **??** shows the description of the robot. L and R represent the positions of the left and right wheel and W the width of the robot. There was a known map of the duct system which was represented by an array of the branch centroids. Error in the motion of the robot could be added into the system by an addition of a constant velocity component to one of the wheels. This would represent the worst case scenario as the real error would be of variable magnitude and be present on one or both of the wheels.

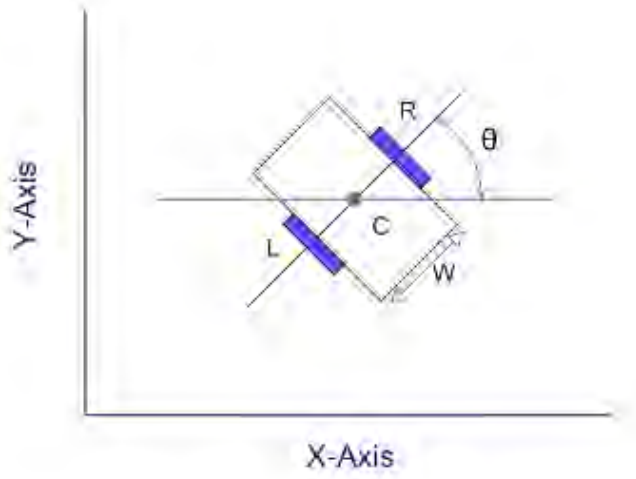

Figure 4.1: A Graphic Representation of the Robot's Variables

4.2 Kinematic Simulation

The path simulation uses the robot's start position and adds the displacement of each wheel to the robot at each time step. The new position and angle were calculated and stored to generate a map of the path taken.

4.3 Sensor Simulation

The sonar simulation provided range information about the closest object within the sonar cone. At each time step the sonars polled the objects (duct walls) and the distance to the closest one that was within an acceptable range of angle of incidence was returned. This data was the range measurement and was assumed to lie along the centre of the sonar cone. This point was stored to generate an internal map of the duct system.

4.4 Wall Simulation

The robot must not only not pass through walls, but must also react to them in a realistic way. The wall simulation assumed that should the robot come into contact with a wall it would push itself parallel to it and as such places it parallel and just off the wall.

4.5 Control Algorithm Development

For testing, a list of branch positions was provided to the algorithm for comparison with those detected by the sonar. Each position was was noted along with the branch type (left, right or t-junction) and an explored/unexplored flag. The algorithm updated this list as exploration occurred. Since the kinematic simulation was velocity driven, the only output the control algorithm had to provide was the velocity of each wheel.

Two separate algorithms were developed and merged to create the final algorithm. There was the forward moving algorithm, which drove the robot

forward, noting branches till the duct came to an end; and the reverse algorithm, which reversed the robot to the last seen branch and moved into it. When these two algorithms were combined, full, recursive exploration was achieved.

4.5.1 The Forward Control Algorithm

The forward algorithm advanced the robot through the duct and kept the robot in the centre of the duct by comparing the left and right sonar readings and moving away from the wall which was closer. As it moved through the duct, branches were noted. They were only noted as a branch if they were in approximately the expected position and of the expected type. A flowchart of this algorithm can be seen in *Figure* **??**. Since a particular branch would be seen through multiple time steps, the simulation considered them to be the same branch and stored the branch centroids, until it disappeared. Then the branch's actual centroid was calculated by taking an average of all the seen centroids. The forward algorithm came to an end when the front sonar's range reading went below a preset threshold.

4.5.2 The Recursive, Reverse Control Algorithm

After the forward algorithm was complete, the recursive, reverse algorithm, seen in *Figure* **??** takes over. This algorithm was recursive, simplifying the code and was required to move the robot back to an unexplored point. It reversed the robot, again using the sonars to keep it in the centre of the duct until the last seen branch was reached. It then turned into it and called the forward algorithm again. It had to be recursive as often there are multiple reverse and turning maneuvers required for the robot to reach the last remembered branch.

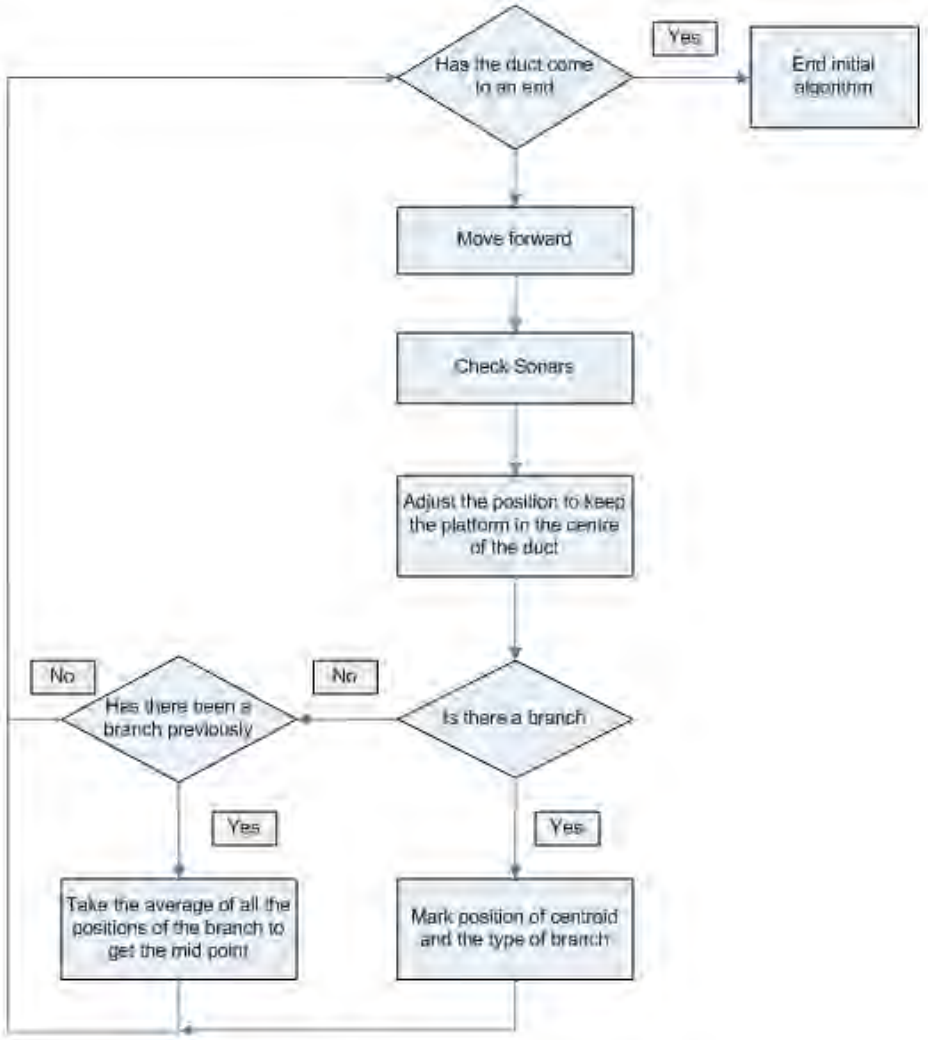

Figure 4.2: The Basic Algorithm Flowchart

The robot must always navigate through unexplored ducts in the forward position to allow the user to make use of the pan and tilt camera. When the robot navigates out of an explored duct it must do so in reverse to avoid tangling the tether. The direction the robot turns depends on whether it is moving into an unexplored duct or out of an explored one. If the robot is turning into a unexplored duct on the right, then it must turn right and face forwards. If it is coming out of that branch then it must turn left to continue navigating in reverse. This was decided based on a variable in the internal map. Initially all branches were given a value of 1, representing unexplored and as the robot turned into them this value was set to 0 or explored. The direction of the turn was dictated by the side of the branch and whether it held 0 or 1 in the array.

The entire control algorithm continued in this manner, gradually decreasing the list of branch centroids that were coded into it each time a branch was completely explored. This continued until the list was empty at which point the robot returned to its starting position.

Further detail on the control algorithms can be found in *appendix* **??**.

4.6 Simulation Results

The simulation was tested on maps of varying complexity with both left and right duct branches and with varying wheel velocity errors. A run with zero error in a complex duct is shown in the figures below.

The left of *Figure* **??** shows the forward algorithm where the platform moved forward into the duct, noting branches. Once the platform reached the end of the duct the reverse algorithm (shown in red) began and it traversed to the last branch seen and turned in. The robot can not turn immediately upon

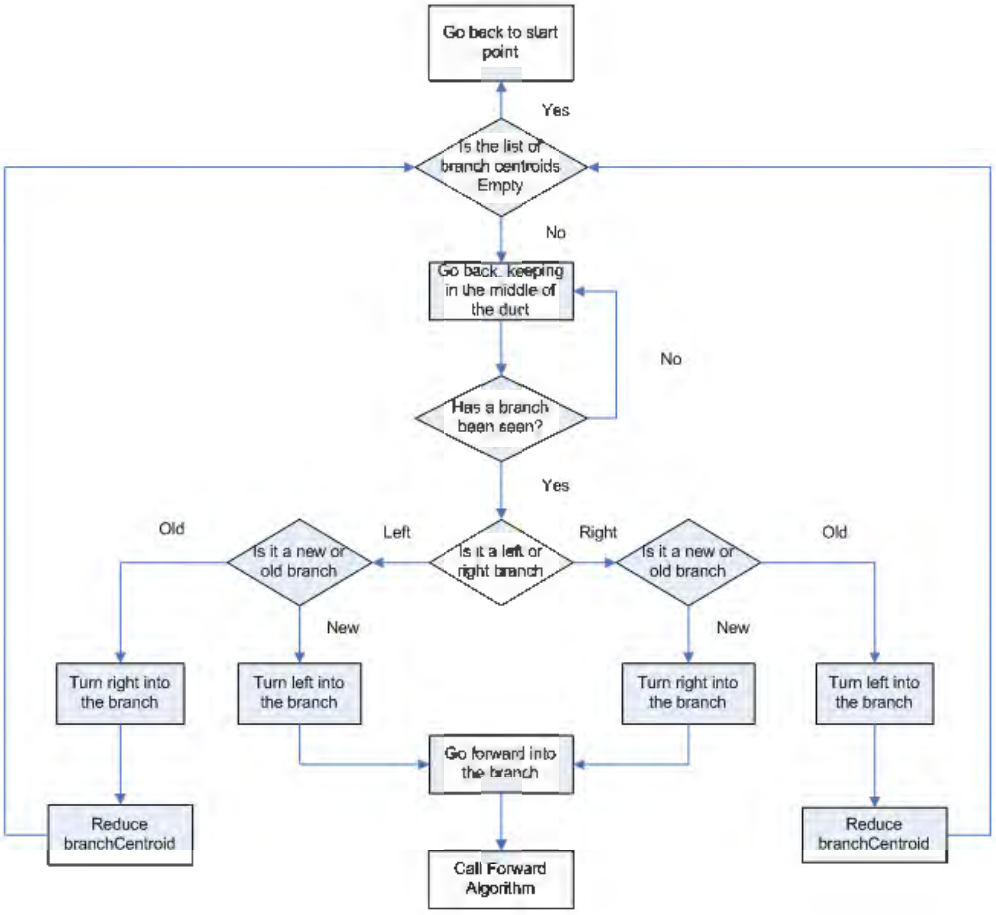

Figure 4.3: The Recursive Algorithm Flowchart

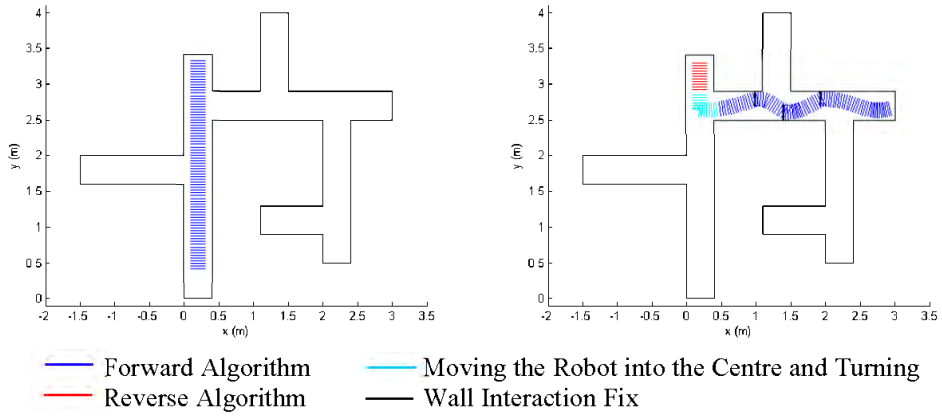

Figure 4.4: The Forward Algorithm (left) Complete Algorithm(right)

seeing the branch as it would not be in the centre of the duct. To account for this the robot reverses further backwards and turned. This movement is shown in cyan and can be seen on the right of *Figure* **??**. Once the robot was inside the duct, the forward algorithm (shown in blue) was called again.

It can be seen that even with zero error the platform began to oscillate back and forth along the width of the duct. This was because the turn into the duct did not put the platform exactly perpendicular to the duct walls. Even though the robot was not perpendicular, the distance between the walls and the left and right sonar was still the same. Thus the platform moved at an angle forward through the duct until the difference between the two sonars exceeded a preset threshold. Once this occurred, the velocities were set in such a manner as to turn the robot away from the closer duct wall. Depending on the magnitude of the angle of the robot, the time taken to change the angle from its current value to one where the robot begins to move away from the wall will vary. During this time, the robot moved closer to the wall. Once the robot was moving away from the closer wall, its angle will be biased toward the further wall and this process will be repeated. Thus an oscillating path along the duct was produced.

Should the robot hit a wall, the wall simulation will immediately correct the position and the orientation of the robot to be just off the wall hit and at 90° to it. This is shown in the figures in black.

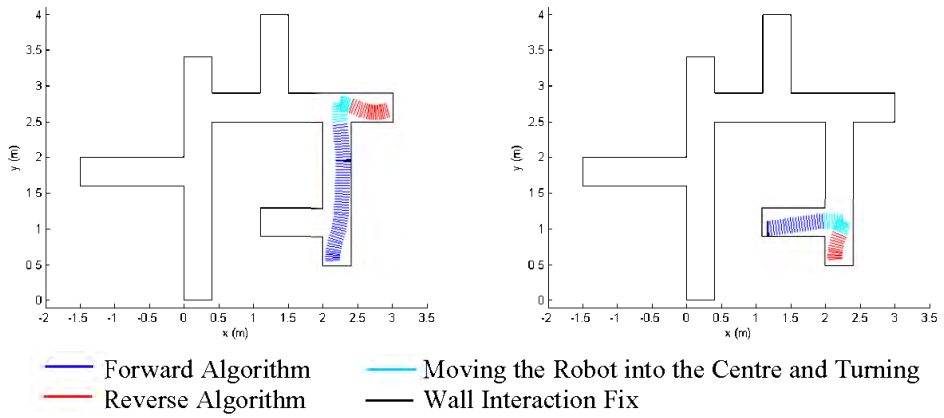

Figure 4.5: Reversing into the Recognised Duct and Exploring

In *Figure* **??** two sets of both the forward and reverse algorithm can be seen. In both, the platform began by reversing, identifying the duct branches, moving into them and then re-calling the forward algorithm.

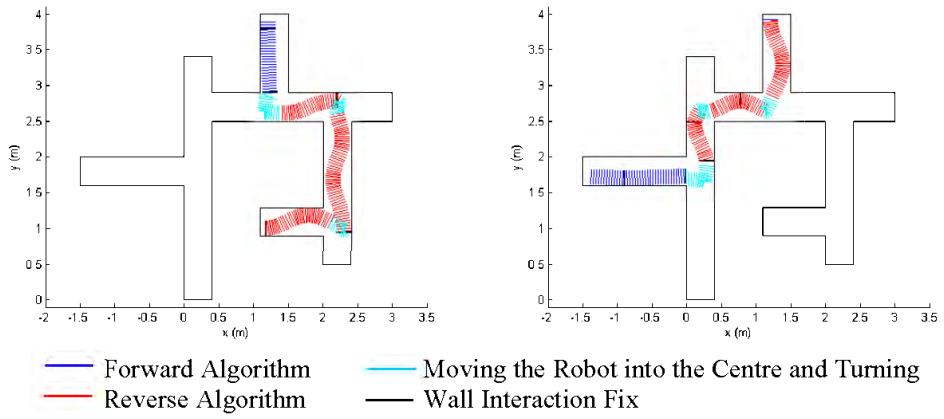

Figure 4.6: Calling the Reverse Algorithm Recursively

On the left of *Figure* **??** the recursive nature of the reverse algorithm can be seen. The robot reversed until it reached the branch in the duct. However,

since it is not a new branch to be explored, the platform turned into it in such a way that it will still be reversing when it entered the branch and then called itself recursively. This continued (reducing the list of branch centroids) until a new, unexplored duct was found. The robot turned into it in a manner that allows it to then move forward to explore the new duct. This same recursive behavior can also be seen on the right of *Figure* **??**

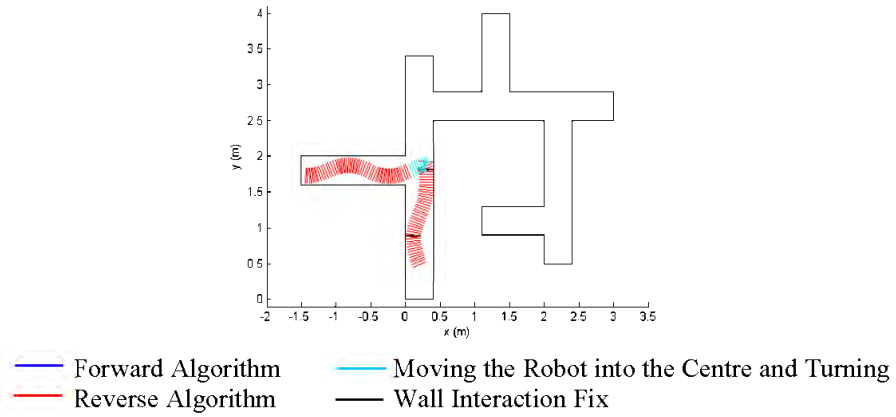

Figure 4.7: Returning Home

Once the platform reached the last branch and turned into it, the list of branch centroids was empty. The exploration was complete and the platform must return to its start point. *Figure* **??** shows it reversing, keeping itself in the centre of the duct until it is within a certain threshold of its start point. The simulation then ends.

The map that the robot explored can be seen in the left of *Figure* **??**. Since sonar returns only distance to the closest object and not any angular information, it is common practice to put the point in the centre of the sonar cone at the distance seen. The map as detected by the sonars can be seen on the right of *Figure* **??**. The regions of constant depth can clearly be seen on the ends of the ducts.

It was initially assumed that the platform would be able to detect the branch

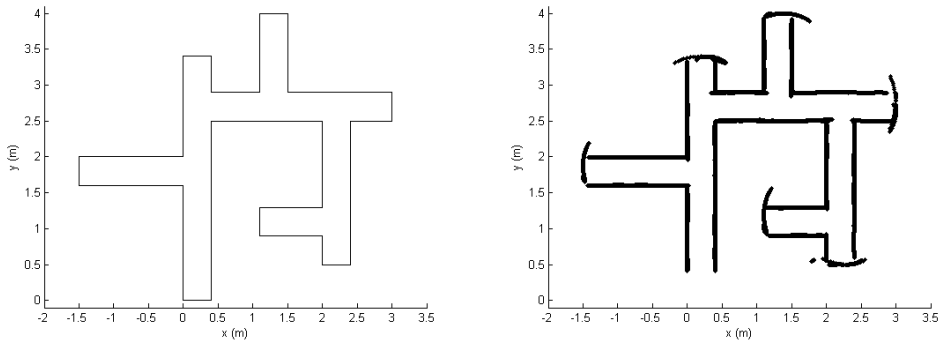

Figure 4.8: The Actual Map (left) and the Sonar Map (right)

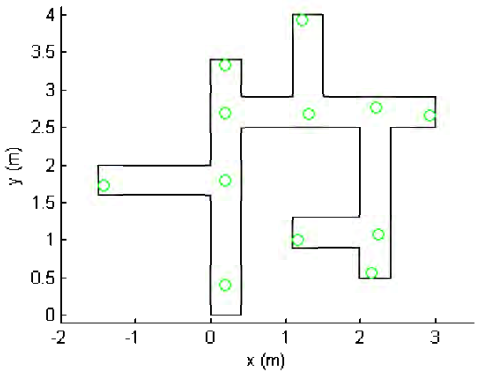

Figure 4.9: The Branch Centroids as Seen by the Platform

centroids, but that these centroids would contain error and that depending on the range and accuracy of the sonars, branches may be seen that did not exist. For this reason the simulation was initially given the branch centroids, rather then using centroids detected by the sonars. The centroids and the end points of the ducts that it detected were stored and can be seen in *Figure* **??**. Since there was no error in the system, the first two centroids have zero error. However, as the platform moves along the duct the subsequent centroids do

have a small error. There are no spurious branches detected and since the detected branches are in approximately the correct place, it indicates that the platform could have navigated using these centroids, proving that a map is not essential, but would reduce errors due to odometry in long ducts.

Runs with error up to 10% of the maximum velocity were performed on the system. It responded very similarily to the run shown above, however the osscillations began sooner.

4.7 Summary

In this chapter the setup of the simulation and its verification have been covered. This included descriptions of the various parts of the simulation, including simulating the kinematic motion, the ultrasonic transducers and the interaction between the walls and the robot. The operation of the control algorithm has been explained and the sonar data and robot path were plotted for a test run.

Since each subsystem has been manufactured and the accompanying circuitry and software designed, testing of the system can now take place and is detailed in the following chapter.

Chapter 5

Testing

Before tests could be conducted on the system as a whole, each subsystem had to be tested and its operation verified. Once that had taken place, the entire system was put through a series of tests of increasing complexity. The initial testing performed on the sonar and motor system can be found in *appendix* **??** and **??** respectively.

5.1 Speed Control

The speed control tests were run on a single motor under zero load. The proportional control loop used can be seen in *Figure* **??**.

It was not important that the robot strictly maintain a set speed, but rather that it maintained a constant speed. To this end only a proportional controller was implemented instead of a more complex one such as a PD of PID controller.

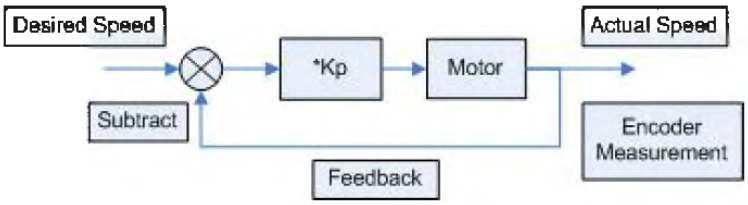

Figure 5.1: The Proportional Control Loop

5.1.1 Verifying Accuracy of Runs

Since errors could be present in the drive system, it was necessary to determine if any error was present before choosing a value for K_P. This was done by performing four runs using a K_P of 1 and then comparing the results. The results can be seen in *Figure* **??** and show that the system was stable.

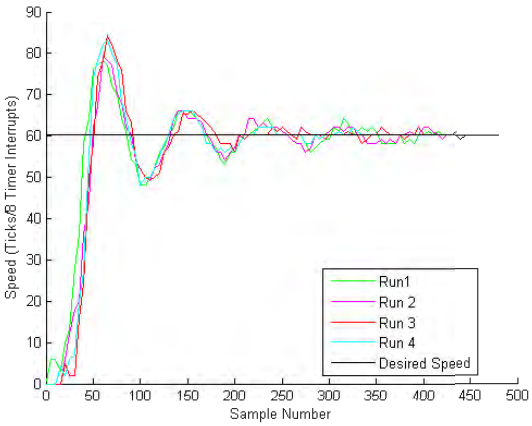

Figure 5.2: Four Runs Using $K_P = 1$

5.1.2 Choosing K_P

Since varying K_P has a huge impact on both the response time and steady state error of the system, tests were run on one motor using varying values of

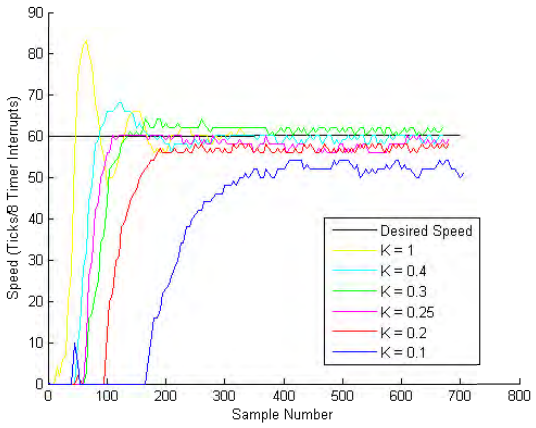

Figure 5.3: Altering K Using the Same Set Speed and Motor

K_P. As can be seen from *Figure* **??**, when K_P was either 1 or 0.4 oscillation occured. When K_P was reduced to 0.1 or 0.2 the response time was very slow and a large steady state error occurred. Since $K_P = 0.3$ overshot the desired speed a K_P of 0.25 was chosen.

5.1.3 Matching K_P for both motors

Since no two motors are the same a test was run to ensure that both motors would behave similarly if given the same speed. *Figure* **??** shows the response of both of the motors. As can be seen, they are almost identical and thus a K_P of 0.25 was implemented in both motors' speed control loop. The frequency of each reading occured at roughly 3kHz so the time to reach equilibrium was minimal.

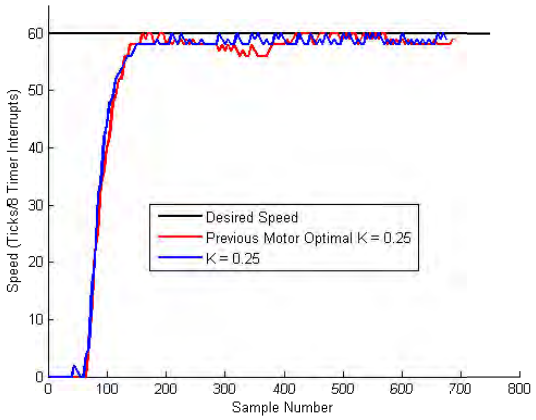

Figure 5.4: Response of Both Motors Using $K_p = 0.25$

5.2 Front Camera Test

To validate the operation of the front camera and pan/tilt mechanism, the robot was placed at a junction in a maze and various images while panning the camera. *Figure* **??** shows the actual positions of the robot and camera, while *Figure* **??** shows the snapshots from the streamed images. The robot drove toward the branch, looked to the end of the duct in front of it, then turned the camera to look down the right branch. With relative ease, the user can manipulate the camera to look down branches, or to scan the walls of the duct as they go past.

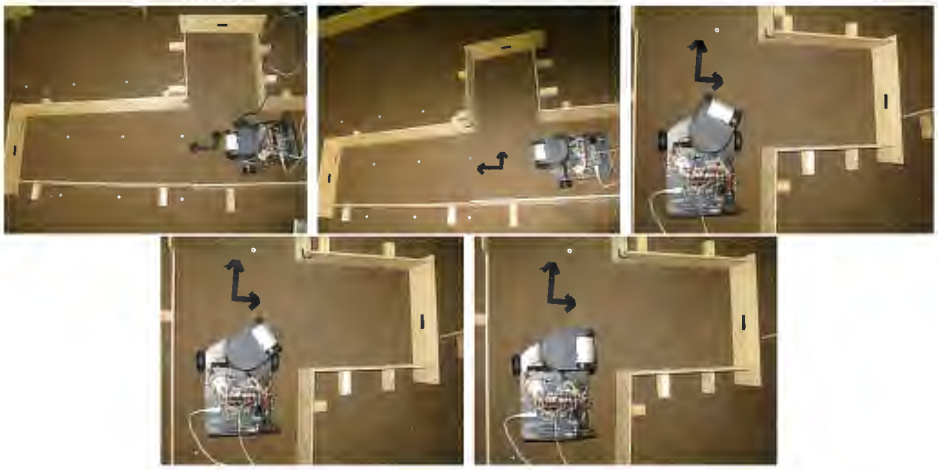

Figure 5.5: Images of the Position of the Camera and the Robot

Figure 5.6: Images of the Duct Using the Forward Facing Camera

5.3 Localisation and Ultrasound Testing

To test whether localisation using dead-reckoning was possible, the robot had to drive along a straight path while storing both sonar and odometry readings. Since the odometry readings did not include wheel slip or any other errors, the actual path that the robot took had to be recorded so that it could be compared with the dead-reckoning path. The testing method similar to that used by Kate McWilliams **?**, but with some modifications. To record and map out the robot's path, a camera was set up on a tripod and a grid was marked on the test area. The video of the test run was broken down into separate frames and post-processing was done to change the perspective of the image to compensate for the camera angle. This post-processing, done using ImageMagick **?**, was achieved by applying an affine transformation **?** to each image, forcing the grid back into a rectangular shape. The original camera shot can be seen on the left of *Figure* **??**, while the post-processed image is on the right.

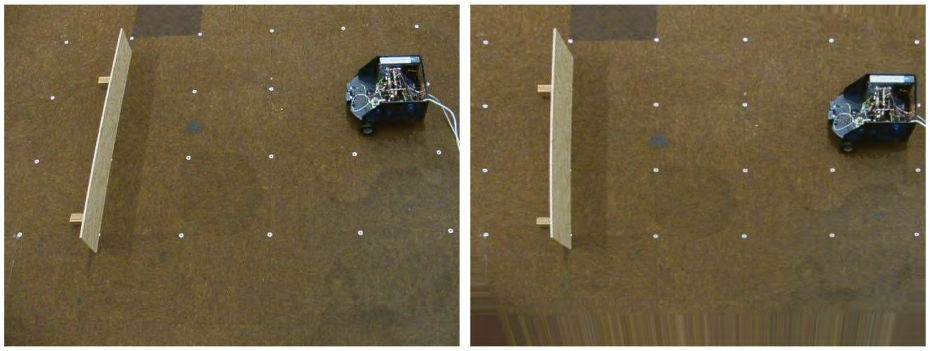

Figure 5.7: The Original Image (left) and the Processed Image (right)

The robot was given a constant forward velocity was driven toward a wall. The stills of the tests can be seen in *Figure* **??**.

The odometry received from the robot was converted from encoder ticks into displacement and was compared to the actual path taken, which had been found using the video stills. *Figure* **??** shows the difference between the

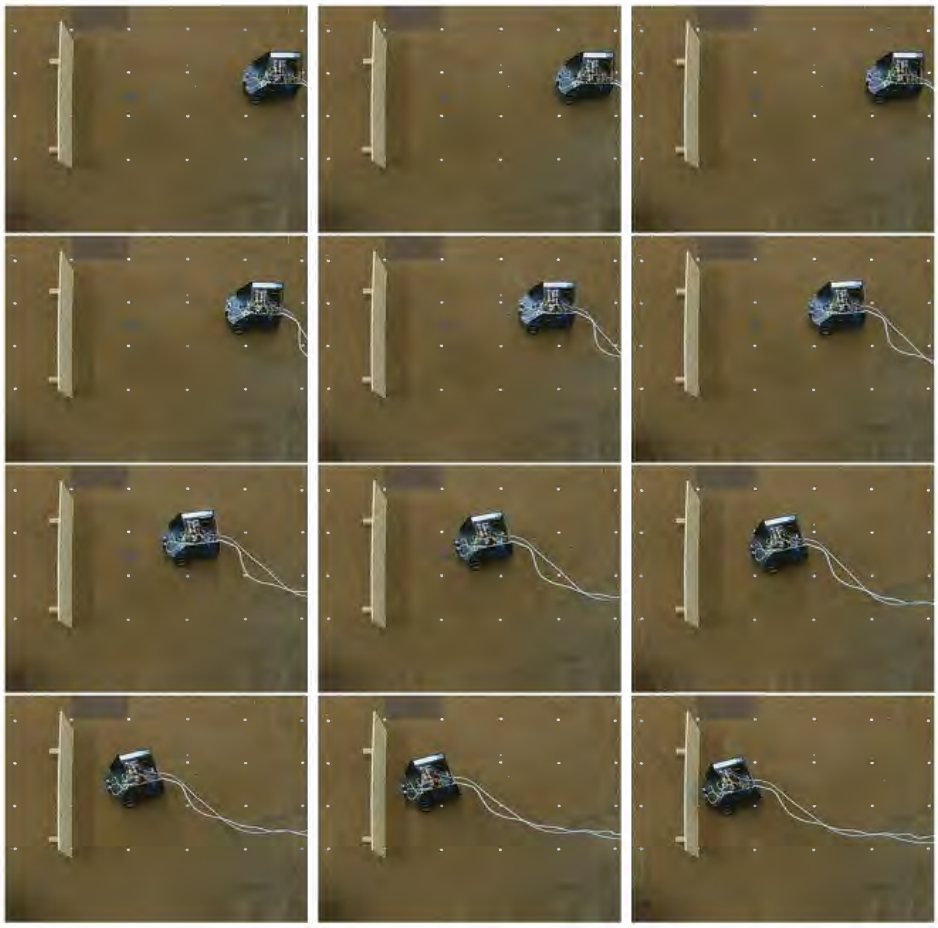

Figure 5.8: Video Stills Showing Actual Path

actual path taken and the dead-reckoning path. The raw speed data from each wheel can be seen in *Figure* **??**.

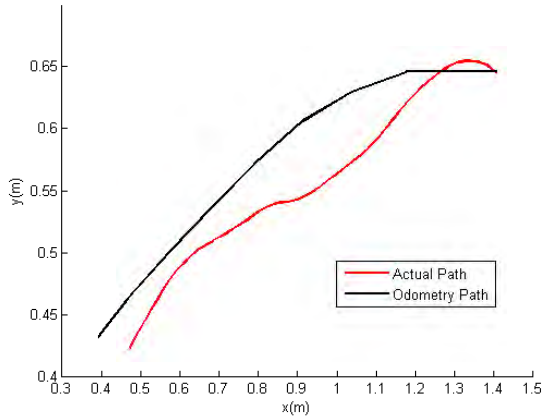

Figure 5.9: Comparison of Actual Path Vs Dead-Reckoning Path

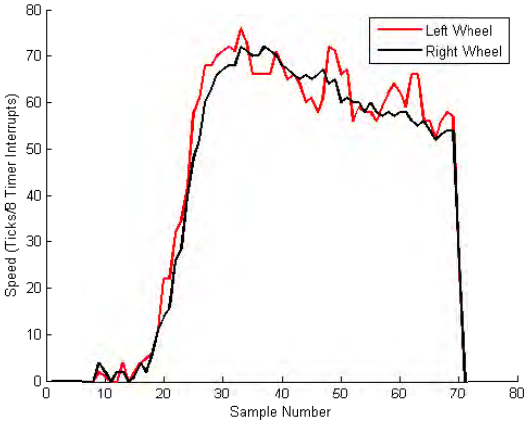

Figure 5.10: Wheel Speeds

The sonar data from the front sonar can be seen in *Figure* **??**. As expected, it increased as the robot became closer to the wall and then remained constant once the robot was stationary. The sonar value in approximately reading 26 was an outlier which the control algorithm would have rejected.

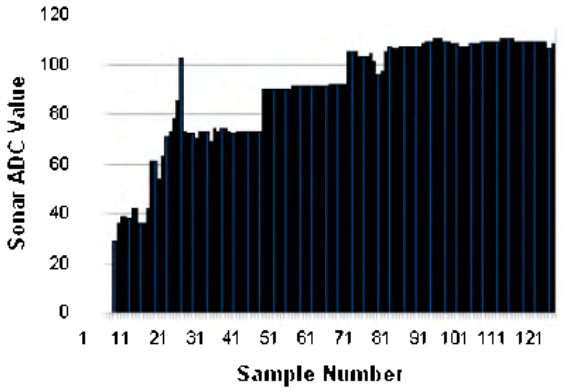

Figure 5.11: Sonar Data Using One Front Sonar

5.4 Stability of Speed Control and Repeatability of the System

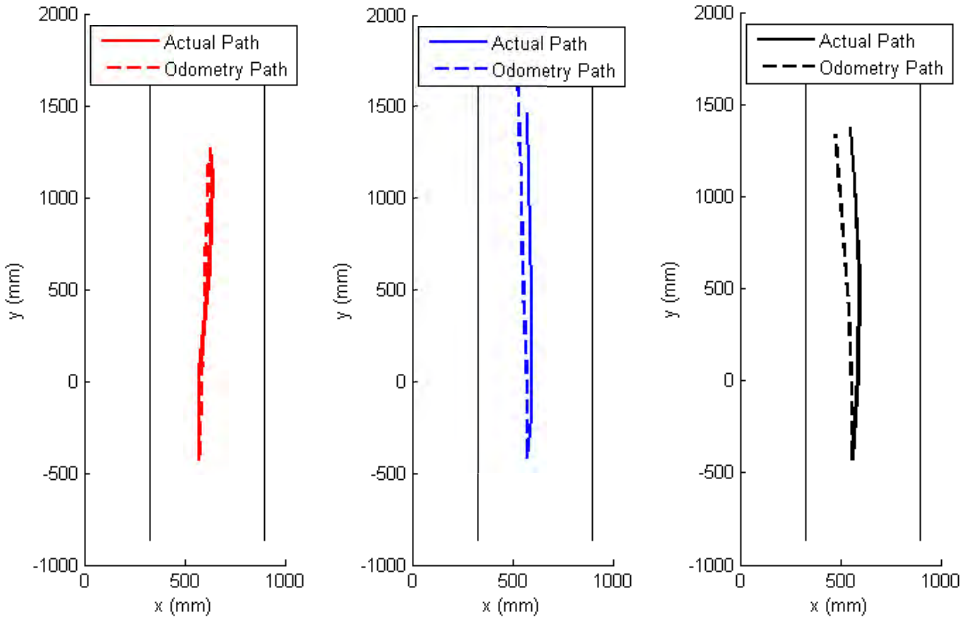

Figure 5.12: Three Straight Line Tests in the Same Duct

Since odometry error accumulates over time, a longer run should produce greater error. To test the system the robot was driven in a straight line through a duct of approximately 1m in length. Three separate tests were run, the odometry data stored and videos taken. Using the same system as described above, the odometry readings were processed and *Figure* **??** was generated. The dashed lines represent the odometry data while the solid lines represent the actual path taken. As can be seen the error goes from almost non-existent (leftmost run) to a maximum of roughly 50mm (centre run). In the worst case scenario this gives an error of 5% of the length of the duct. The figure also shows that the robot moves in a fairly consistant straight line.

5.5 Autonomous Branch Recognition and Turning

To navigate autonomously through a duct, the robot had to be able to recognise a duct branch and turn 90° into it. To verify this, a maze containing one right branch was constructed and the robot made to navigate toward it. Sonars were placed in the centre of the robot on both the left and right sides and no external input was given to the robot other then the initial wheel velocities. The wheel velocities and sonar readings were stored.

The sonar readings taken from the test can be seen in *Figures* **??** and **??**. *Figure* **??** shows the left sonar readings and *Figure* **??** the right. The robot sensed the duct on both sides until approximately reading 60 where a branch on the right was noted. The robot then performed a 90° spin so that it faced the branch, with no walls on either side. As can be seen in the both the sonar figures, the left and right sonar readings were effectively zero. The robot then moved forward, placing it back into the duct and walls are sensed by both the sonars. Once both a left and right wall were seen the robot stopped. The

drastic increase on the left sonar reading at reading 106 was attributed to the reflection caused by the corner as the robot turned and moved into the duct.

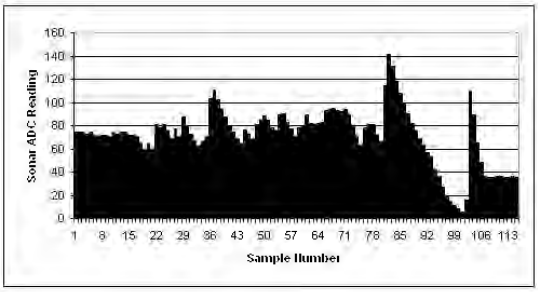

Figure 5.13: Left Sonar Readings While Navigating Around a Corner

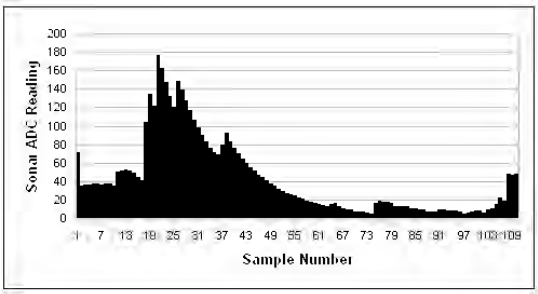

Figure 5.14: Right Sonar Readings While Navigating Around a Corner

The wheel velocities can be seen in *Figure* **??**. Both the left and right wheels ramped up from zero to the desired speed until the duct branch was sensed, at which time they both dropped to zero. During the turn the left wheel velocity was positive and the right wheel velocity was negative, creating a right spin. After the turn, both the velocities ramp back up to the desired speed until the robot came to a stop.

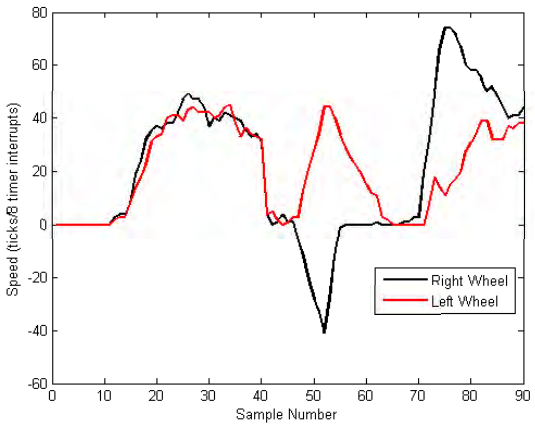

Figure 5.15: Left and Right Wheel Velocities While Navigating Around a Corner

5.6 Map Generation and Branch Location

To test the sonars further, the robot was placed in a duct with a single right branch. It was given a velocity and allowed to move until it came to the end of the duct. Speed and sonar readings were taken at the same time so that the sonar readings could be correlated with the position of the robot.

Before the sonar data could be accurately represented on a map it had to be calibrated. First, the left and right sonar were calibrated so that their range readings were the same when sensing objects at the same distances. The range measurements were then scaled from ADC units to millimeters and the position of the point seen was calculated. Since the range was stored as a voltage, when the robot approaches a duct, the range reading will not immediately become zero and will instead steadily decline, showing the duct getting further and further away, until a zero reading is reached. To reduce the time taken to identify the duct branch, a threshold was set which manually set the range reading to zero if it dropped below the preset value.

Once the sonar data was in the correct format, the map it generated was plotted along with the actual map. The positional data from the odometry was also plotted along with the actual path taken. This can be seen in *Figure* **??**. As expected, the error due to wheel slip and cable tension increases slightly as the distance traveled does. The generated map clearly shows the branch in the duct and also realistically maps the duct walls.

5.7 Autonomous Navigation and Mapping

Since achieving autonomy was a main objective of this thesis, the final test consisted of programming the robot to navigate autonomously through a

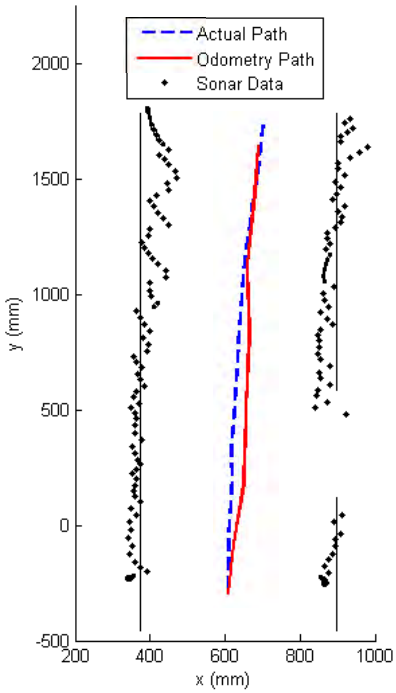

Figure 5.16: Odometry and Sonar Readings Compared to the Actual System

duct with one right branch. The robot must travel forward until the front sonar detects the end of the duct, then reverse, turn into the branch, explore it, reverse out and come back to its start position. The only input provided was the initial wheel velocities and the three sonar ADC readings and wheel speeds at each time step were recorded.

This test was repeated 20 times. The robot successfully navigated through the duct without error 18 times. The 2 failed runs were due to the tether catching on walls thus pulling the robot out of position.

Figure **??** shows the odometry and actual paths with regards to the duct system. As can be seen, the paths were almost identical until the robot performed the 90° turn. During the spin, the right wheel was pushed back by the left wheel, not producing any rotation of the wheel encoder. Because of this wheel slip no odometry readings were produced. It is documented that wheel slip is more prominent during turns **??**.

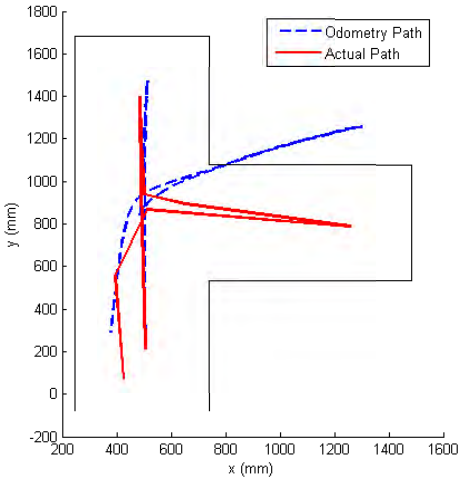

Figure 5.17: The Actual Path vs the Odometry Path During an Autonomous Test

The sonar readings are related to the actual path the robot took, but must be linked to the odometry path as this is the only information the robot has.

Thus when the two paths diverge, the sonar readings seem to map erroneous walls, but are actually mapping the correct walls just relative to an incorrect pose. The sonar readings with the actual map can be seen in *Figure* **??**.

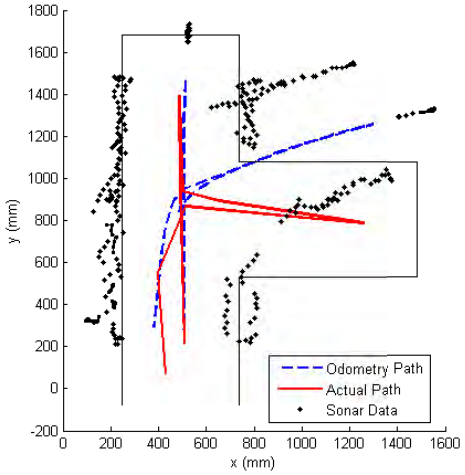

Figure 5.18: Sonar Data Compared to the Actual Map and Odometry Path

Figure **??** shows two stills of the robot, one before the turn and one after it. Since the turns were always roughly 90° and the wheel slip was always present, the embedded code was altered to update the odometry to ensure that the angle of the robot changed by 90°. Although this was not always perfectly accurate, the result was more accurate then the odometry readings before the alteration. *Figure* **??** shows the sonar readings and paths for the robot once the turns have been altered. As can be seen, the sonar data closely matches that of the actual map and the difference between the actual path and the odometry path has been greatly reduced.

If the robot had prior knowledge of the positions of the duct branches and duct ends, then this internal map could be used to update its position. In *Figure* **??** this prior knowledge has been used to update the robot's position. When the robot reached a duct branch or duct end, it updated its position with the position of the branch/end. Since only the position was changed and not the angle, the robot still had errors in odometry. This creates the

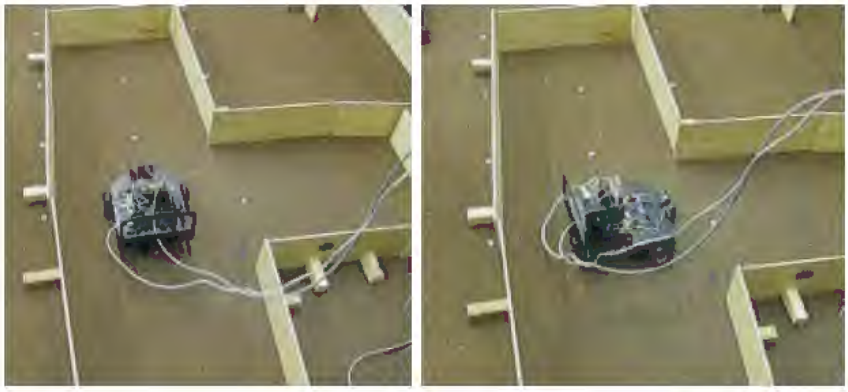

Figure 5.19: Robot Position Before (left) and After the Turn (right)

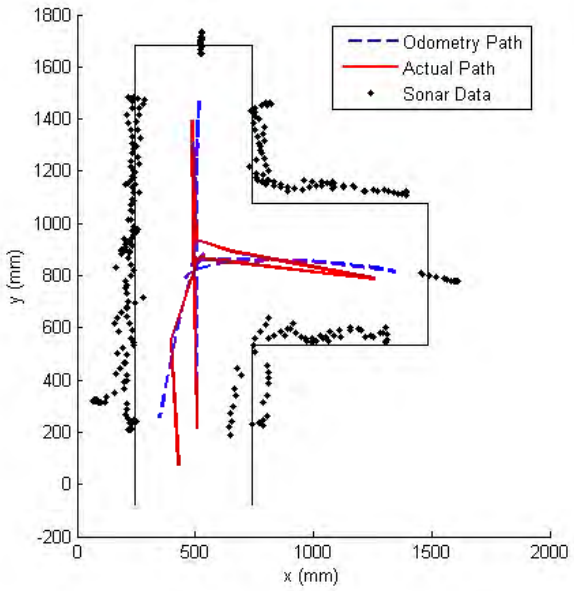

Figure 5.20: Sonar Readings and Paths After 90° Fix

trapezoidal shape shown in *Figure 5.21.* There are now two sets of walls present, however the robot now has a better understanding of its position which greatly increased the accuracy of the odometry data and improved the correlation of the sonar data.

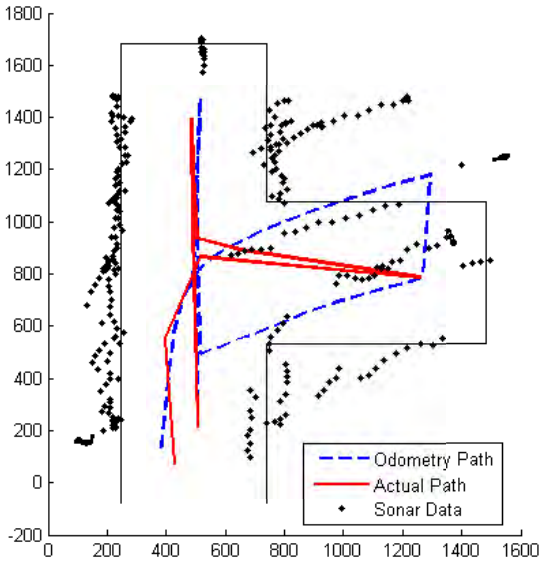

Figure 5.21: Sonar Readings and Paths After Positional Fix

5.8 Summary

This chapter has detailed the testing of the system as a whole. Tests have been run to demonstrate the operation of the speed control, dead reckoning, branch recognition and map generation. The actual path of the robot has been compared to the path generated using the dead reckoning approach. Odometry during turns was corrected to decrease the effect of wheel slip during orientation changes.

Conclusions and recommendations for further work are given in the next chapter.

Chapter 6

Concluding Remarks and Recommendations

6.1 Concluding Remarks

The system as a whole functioned well and performed as expected. Each subsystem was separately tested and calibrated before the final system was tested.

The ultrasonic transducers performed successfully and proved that time of flight does not have to be implemented for simple object recognition. The maps generated were accurate enough to see the general outline of the duct and the branches and duct ends were detected with a 100% success rate.

The odometry readings were accurate when the robot traveled in a straight line, but during turns, the large amount of wheel slip caused the odometry readings to be very inaccurate. However, since the robot turned roughly 90°

in every test, software was successfully implemented to force the change in angle to be 90°. Since the sonar data reflected what the robot saw at its actual position, the greater the difference between the odometry and actual path, the less the sonar data seemed to correlated with the duct.

The proportional speed control that was implemented on the system was repeatable and kept the robot going straight along the duct.

The Roboaudiostix/Gumstix combination implemented the control software and data transmission without fault. Although the full exploration algorithm was not implemented on the Gumstix, the system would be more then capable of doing so. Should further modifications need to be added to the system, this would be possible given that all the capabilities of the controller were not fully used. Further circuitry is available for the addition of a rear facing sonar and there are plenty of available GPIO, ADC and DAC ports available for expansion.

The simulation coded in *MATLAB* allowed for the generation of the control algorithm and was user friendly, should further duct systems need to be generated. The sonar simulation was too accurate when compared to the actual maps generated as it was free of errors, but enabled the control algorithm to be tested adequately.

The reliability of the sonar readings allowed autonomy to be implemented successfully. The robot mapped and explored a duct with a single branch and successfully navigated through the duct 90% of the time. Failure to complete 10% of the runs was due to catching the tether on walls and corners which pulled the robot out of position, thus not allowing it to turn correctly.

6.2 Recommendations for Future Work

Although the system functioned sufficiently well to implement in a complex duct, the following modifications are recommended to improve the operation of the system:

The sonars performed successfully to detect the presence and absence of duct walls, but the internal map generated would not be accurate enough should it need to be used. The sensors were also very sensitive to frequency, voltage and angle changes and should be replaced with more robust ones, with better specifications. Another option to effectively increase the accuracy is to mount sonars on a 360° ring above the robot.

Since the odometry data was fairly consistent to the actual movement of the robot while it was moving in straight lines and not so during changes of bearing, some system to detect bearing change would be needed. The entire system was designed to be cost effective and so expensive sensors such as digital compasses should be avoided. Instead an optical mouse wheel could be implemented simply to allow changes in bearing to be measured.

Since only three sets of sonars were implemented on the system, there were dead areas on the robot where no data was available. Thus it was possible for the robot to make contact with a wall should the sonar data not steer the robot away from it. Bump sensors placed on a ring around the robot would not only protect the robot, but would allow it to back away from any wall it had made contact with and continue on.

In a very long duct with multiple bends, tether management would become a problem. Either a tether with a self-recoiling mechanism would have to be developed or the robot would have to be made wireless. This may be possible with the addition of a wireless router board from Mikrotec **?** who

have developed directional aerials for use in pipes.

The full, autonomous algorithm should be implemented on the Gumstix and run in real-time in a complex duct environment and the performance monitored.

Further development of the chassis and wheel structure is required to allow the robot to navigate through a very dirty environment. The control systems should be enclosed and the castor wheel arrangement replaced for a more robust system.

References

Life Air - Ventilation hygiene inspection, air duct cleaning equipment. http://www.lifa.net/en/prod_ven.asp.

Everest VIT Robotic Crawlers. www.everestvit.com/rovver_index.html.

Cyclone Industries - Worlds Most Complete Robotics Offering. http://www.cyclone-industries.com/html/english/cy_bot.html.

J. Velagic, B. Lacevic, and B. Perunicic, "A 3-level autonomous mobile robot navigation system designed by using reasoning/system approaches," *Robotics and Autonomous Systems*, pp. 989–1004, 2006.

J. Fonseca, J. Martins, and C. Couto, "An experimental model for sonar sensors," *International Conference on Information Technology in Mechatronics*, 2001.

J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man, and Cybernetics*, 1989.

Gumstix - Dream it, Design it, Deliver it. http://www.gumstix.com/.

Air Care - Duct Cleaning Procedure. http://www.air-care.com.

Speedway - Video Inspection Systems. http://www.plumbingstore.com/speedway_pipeinspection.html.

J. D. Tardos, J. Neira, P. M. Newman, and J. J. Leonard, *Robust Mapping and Localization in Indoor Environments Using Sonar Data*, 2002.

G. D. Castillo, S. Skaar, A. Cardenas, and L. Fehr, "A sonar approach to obstacle detection for a vision-based autonomous wheelchair," *Robotics and Autonomous Systems*, vol. 54, pp. 967–981, 2006.

U-blox - SuperSense Indoor GPS. www.u-blox.com/technology/supersense.html.

J. L. Jones, B. A. Seiger, and A. M. Flynn, *Mobile Robots - Inspiration to Implementation*. A K Peters Ltd, 1999.

S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, 1997.

C. Christiansen, "Announcing the amazing micromouse maze contest," *IEEE Spectrum*, 1977.

T. Braunl, *Embedded Robotics*. Springer, 2006.

N. Morrison, *Introduction to Fourier Analysis*. John Wiley & Sons inc, 1994.

LMD18200 datasheet. http://www.national.com/pf/LM/LMD18200.html#Datasheet.

H21a1 datasheet. http://www.datasheetcatalog.org/datasheet/fairchild/H21A1.pdf.

Murata ma40 series datasheet. http://www.murata.com/.

ICL7667 datasheet. http://www.datasheetcatalog.org/datasheet/intersil/fn2853.pdf.

Atmel - Atmega128 datasheet. http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf.

D. Hylands, *Dave Hylands Home Page*. http://www.davehylands.com/.

D. Hylands, *Gumstix Sample Code*. http://docwiki.gumstix.org/index.php/Sample_code.

K. Mcwilliams, "Development of Localisation Capabilities for a Low-cost Robot," Master's thesis, University of Cape Town, 2008.

A. Gilat, *Matlab - An Introduction with Applications*. John Wiley & Sons inc, 2008.

ImageMagick Studio - ImageMagick. `http://www.imagemagick.org`.

Wolfram Mathworld - Affine Transformation. `http://mathworld.wolfram.com/AffineTransformation.html`.

Microtek Electronics - Superior Wireless Technology. `http://www.microtekelectronics.com`.

Danduct Clean Products. `http://www.danduct.com/`.

Inuktun - Modular Mobile Robotics. `http://www.inuktun.com/`.

D. Langer and C. Thorpe, *Sonar based Outdoor Vehicle Navigation and Collision Avoidance*.

D. B. Johnson, *A Note on Dijkstra's Shortest Path Algorithm*.

W. H. Huang, B. R. Fajen, J. R. Fink, and W. H. Warren, *Visual navigation and obstacle avoidance using a steering potential function*, 2006.

E. Kreyszig, *Advanced Engineering Mathematics*. John Wiley & Sons inc, 2006.

J. Stewart, *Calculus - Concepts and Contexts*. Brooks/Cole, 2001.

L. Kleeman and R. Kuc, *Mobile Robot Sonar for Target Localization and Classification*.

M. D. Adams, *Coaxial Range MeasurementCurrent Trends for Mobile Robotic Applications*, 2002.

CD4047 datasheet. `http://www.datasheetcatalog.org/datasheets/560/109076_DS.pdf`.

Appendix A

Literature Survey

Contents

A.1 Introduction A-3

A.2 Duct Inspection Robots A-3

A.2.1 Danduct Clean Family **?** A-3

A.2.2 The Rovver Family **?** A-4

A.2.3 Inuktun Family **?** A-5

A.2.4 CY-Bot **?** A-7

A.3 Inspection Cameras A-7

A.4 Embedded Systems A-8

A.5 Map Generation A-9

A.6 Navigation A-11

A.7 Concluding Remarks A-12

Figures

A.1 Danduct's Range of Inspection Robots A-3
A.2 The Rovver Series A-5
A.3 Versatrax 100 . . . . . A-5
A.4 Nanomag A-6
A.5 CY-Bot Showing Air Pistol A-7
A.6 D-Link DCS-900 . . . . . A-8

A.1 Introduction

This appendix provides information into robots used for duct inspection and cleaning. Included in this appendix is a resource of some of the most applicable robot systems as well as additional information on map generation and navigation.

A.2 Duct Inspection Robots

A.2.1 Danduct Clean Family ?

Danduct Clean is a company that has developed a complete range of equipment for the cleaning of ventilation systems in commercial, residential and industrial buildings. The robots also carry video cameras for visual inspection.

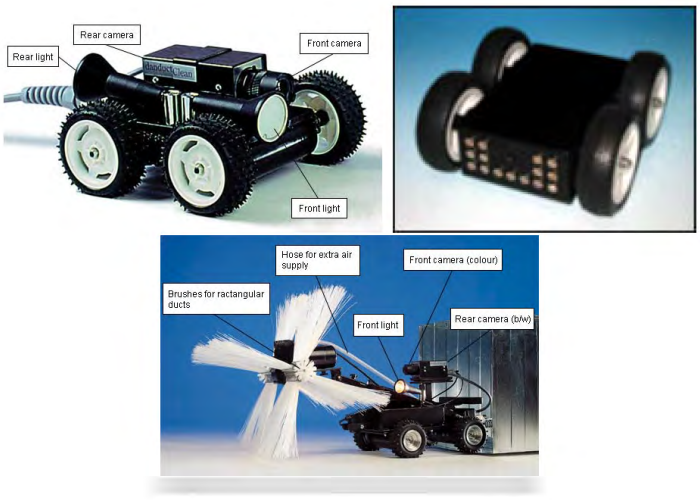

Figure A.1: Inspection Robot (left), Micro Inspection Robot (centre), Multi Purpose Robot (right)

The Danduct Inspection Robot, seen on the left of *Figure* **??**, has two cameras, a forward camera and a rear facing camera. These allow guided control during both forward and reverse motion. The robot is controlled using a joystick and the lighting is provided by halogen lamps which can be adjusted from the control box. The robot can be equipped with an air nozzle for cleaning if desired.

The Micro Inspection Robot, shown in the center of *Figure* **??**, is a smaller version of the Inspection Robot. It weighs 1.5 kg and its dimensions are 165mmx150mmx80mm. It also has two cameras and illumination is provided by a set of ultra-bright LED's. The Micro Inspector is controlled by a joystick and is configured to be driven upside down if it flips over.

The Danduct Multi Purpose Robot is a combination of an inspection and cleaning robot. It is controlled by a joystick, comes standard with a 10" colour monitor and 3 halogen lamps for illumination. It has interchangeable brushes for different size and shaped ducts and can carry a 30 meter compressed air hose. The brushes, lights and cameras can be seen on the right of *Figure* **??**. The Multi Purpose Robot has four separate engines, each controlling a wheel which gives the robot greater manoeuvrability.

A.2.2 The Rovver Family ?

The entire Rovver family is a series of robots developed by *Everest VIT*.

The Rovver 400, shown on the left of *Figure* **??**, is a compact and powerful inspection platform. It has a colour, pan and tilt, forward mounted camera and illumination is provided by halogen lighting. The camera has remote focus adjustment so that the image is clear at all times. It is powered using a tether and has a range of 200m. It is waterproof and some of its main

Figure A.2: The Rovver Series: Rovver 400 (left), Rovver 600 (centre), Rovver 900 (right)

applications are the inspection of air ducts, electrical conduits, small pressure vessels and tanks. Its dimensions are 247mmx94mmx80mm and it weighs 4.5kg.

The Rovver 600 (shown in the center *Figure* **??**) has the same application as the Rovver 400, but is slightly bigger and used for pipes with diameters ranging from 150mm to 900mm.

The largest of the Rovver family, shown on the right of *Figure* **??** is the Rovver 900, which is used for pipes ranging in diameter from 225mm to 1500mm.

A.2.3 Inuktun Family ?

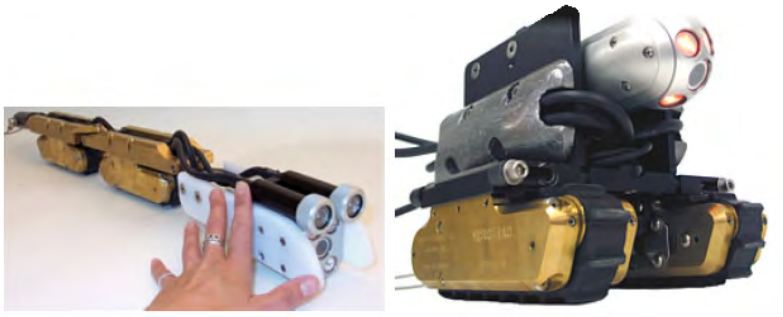

Figure A.3: Versatrax 100 In-line (left) and Parallel Configuration (right)

The Versatrax 100 is a pipe inspection robot designed by *Inuktun*. It can be configured for both pipe and flat surface operation as can be seen from *Figure* **??**. On the left, the Versatrax 100 is shown in the pipe inspection configuration and on the right it is shown in parallel configuration. Versatrax is designed to move through pipes of a minimum diameter of 100mm, is capable of submerged operation up to 30m and has a tether with a range of 90m. The forward facing camera is colour and has pan, tilt and zoom capability.

Versatrax also makes larger inspection robots with more features and for pipes of larger diameters.

Nanomag

Figure A.4: Nanomag

The Nanomag is designed to adhere to any metal surface, even upside down. It is designed for small spaces, its dimensions are 105mmx157mmx49mm and it weighs 2.2kg. Its small size can be seen in *Figure* **??** above. It has two cameras, one forward facing and one rear facing, with variable LED's for illumination. It has a 30m tether and was designed for the inspection of small spaces.

Figure A.5: CY-Bot Showing Air Pistol

A.2.4 CY-Bot ?

CY-Bot or *NATROLLER*, shown in *Figure* **??**, is a dedicated duct cleaning and inspection robot designed for HVAC ducts. It weighs only 6kg, but can carry a 50kg payload or tow a 45kg payload. CY-Bot comes equipped with a 100 foot tether, and a rotating module. This module is able to rotate 180° and contains the lights, camera and air pistol. Optional extras include a manipulator arm and brushes to clean the ducts.

A.3 Inspection Cameras

Video Cameras are necessary for the operator to do a visual inspection of the duct and determine if cleaning or repair was necessary.

Network Cameras

Network or IP cameras can be either analogue or digital and have an embedded video server with an IP address. They have higher resolutions then CCTV analogue cameras and a typical network camera has a VGA (640x480

pixels) resolution.

Network cameras are readily available and are reasonably priced. They commonly used for cheap surveillance applications.

Since the operator needs see the status of the duct in real-time, the camera data would need to be streamed live. Images cannot be easily streamed wirelessly in a duct and would have to be streamed over a tether. If desired the images may be stored for later viewing.

Figure A.6: D-Link DCS-900

The D-Link DCS-900 camera was chosen for this application because of it's low price, high frame rate and low mass. A pan and tilt mechanism was added to the front camera allowing 180° motion in the horizontal and vertical directions.

A.4 Embedded Systems

An embedded system is a computer designed to perform a specific task. The term embedded means that that particular system is contained wholly within

a larger system. Unlike personal computers, embedded systems are only able to perform specific tasks and may contain a task-specific mechanical device. Embedded systems can range in complexity from very simple, (containing only one microprocessor chip) to very complex (containing multiple control units).

Some of the simpler tasks that this particular embedded system would perform would be the capturing, storing and processing of the position data, control of the motors and performing the movement control algorithm. More complex tasks, such as receiving control information from the host computer, creating the map and storing the relevant data also needed to be performed. For this to occur a both a high-level and low-level controller were needed.

Using one controller can simplify the problem as no communication between controllers is needed. However, complexity is increased because the controller must perform all the tasks at once and scheduling both the simple and complex tasks in real-time can be difficult to implement. If two separate controllers are used, complexity is added because of the interfacing and communication that is needed. Having two controllers does add robustness to the system as if the high level controller fails, the robot can still operate in some manner and would be able to detect the failure and take appropriate action. Running two controllers in parallel means that specialised controllers can be chosen and that the principles of subsumption architecture can be implemented.

A.5 Map Generation

Once a method of detecting duct walls and a localisation technique have been developed, a map can be generated using the information gathered by the on-board sensors. Because accurate localisation is needed at each time

step to build to useful map, the positional error and the sensor error will contribute to an error in the map. If left unchecked these errors will grow bigger and bigger over time. Odometry errors effect the way that sensor errors are interpreted however recognisable features will increase the accuracy.

It is possible to use a probabalistic approach using either a Kalman filter or Dempster's expectation maximisation algorithm **?**. These would allow a map to be generated and would reduce odometry errors as sensor data is correlated with positional data to provide a pose extimate. However these approaches are often complex and have expensive computation times.

There are three simple approaches often used to generate maps: metric(geometric), topological and hybrid. Each of these have their own advantages and disadvantages **?**.

Metric or grid based maps **??** are created when the environment is broken down into discrete cells, each of which can be empty or occupied. The size of the cells must decrease for accuracy to increase. For a detailed environment, metric maps are complex and this inhibits path planning in large-scale environments. Since the resolution must be fine enough to capture all details, these maps can be memory intensive and slow. They are also memory intensive.

An occupancy grid can be used to increase accuracy. In this method the cells can be partially occupied **??**. As each measurement is taken, the probability of the cells at the returned range increase, while all that lie within the sonar cone below the returned range have their probability of occupation reduced. In this way a more accurate map can be generated, however it is slower as more measurements need to be taken to confirm occupancy.

A topographical map describes the connections between points or nodes **?**. Topographical maps are feature based and are best used in corridors, roads

etc. They are less sensitive to sensor error and are much less complex then metric maps. They permit very efficient path planning and require a smaller amount of memory. These maps need recognisable landmarks to operate correctly and cannot describe individual objects in great detail. Since this approach usually does not require the exact position of the robot in its environment, drift or wheel slip will have less of an effect **?**.

A hybrid of these approaches has been used successfully and combines the greater detail of the metric map with the global simplicity of the topographical **?**. The entire system is represented as a topographical map but each node on it is represented as a metric map.

A.6 Navigation

Once the platforms orientation and position are known at all times, navigation to a particular destination becomes possible. However, this is more complex then just driving to a specific location. In some cases there may be many possible paths to the goal with only one of them being the optimal route. A navigation algorithm can be used to solve this problem. These algorithms are theoretical and may not match the actual system, however they can often be modified for a specific application. Some navigation algorithms are shown below:

- Dijkstra's Algorithm **?**: Calculates the shortest path to all nodes from a given starting node.
- A* Algorithm [**?**, pp210]: Calculates the shortest path to all nodes from a given starting node while taking into account the cost to move from one square to another.
- Potential Field Method **???**: Gives each object a repulsive force and

the goal an attractive force. The platform will then be 'pulled' to the goal.

- DistBug Algorithm [**?**, pp211]: Go directly to the goal when possible, if an object is encountered then follow the wall until the platform can travel toward the goal.

A.7 Concluding Remarks

Many robots are currently available for inspection. They all have on-board cameras and most have a high quality forward facing camera and low quality rear facing camera. Illumination for the cameras was provided by either halogen lights, or by ultra-bright LEDs and most platforms were controlled and powered by a tether which was between 20-30m long.

The DCS-900 has adequate specifications for the task while being cost effective. The forward facing camera will be equipped with a pan and tilt mechanism while the rear facing camera will be mounted in place.

Two separate embedded controllers were chosen to perform the control tasks, allowing greater robustness. The Gumstix and Roboaudiostix were chosen for the task.

A map must be created while the platform explores the duct using the on-board sonars. This will test the accuracy of the transducers and provide some information about the duct. At first a grid based system will be implemented, but as the duct system to be explored becomes longer and more complex, a topographical method must be considered.

Whether the robot has been given the map or if the robot has generated

the map while exploring the duct, a navigation algorithm must be generated to allow travel from one point to another using the shortest possible path. Dijkstra's Algorithm calculates the shortest path from each node relative to a starting point and should be used for this application as it was considered to be superior to the other methods. The potential field method suffers from local minima and often gets stuck, the Dist Bug algorithm is slow, while the A* algorithm is computationally expensive since there is no real cost to movement for this application.

Appendix B

Simulation

Contents

B.1 Introduction B-6

B.2 Kinematic Model B-7

B.2.1 Description B-7

B.2.2 Assumptions B-8

B.2.3 Kinematic Derivation B-8

B.3 Sonar model B-11

B.3.1 Sonar Assumptions B-12

B.3.2 Sonar Derivation B-12

B.4 Wall Avoidance Algorithm B-21

B.4.1 Wall Avoidance Derivation B-21

B.5 Control Algorithm B-24

B.5.1 The Basic Forward Algorithm B-25

B.5.2 The Recursive, Reverse Algorithm B-27

Figures

B.1 A Graphic Representation of the Model's Variables B-7
B.2 A Graphic Representation of the Sonar Variables B-11
B.3 Sonar Setup B-12
B.4 Effects of Beam Angle on Visibility B-13
B.5 Pictoral Explanation of rap and ral B-14
B.6 Sonar Derivation Checks B-15
B.7 Explanation of the Distance Calculation B-16
B.8 Possible positions of *rap* B-17
B.9 The Sonar Definition B-19
B.10 The Position of the Bump Sensors B-21
B.11 The Basic Algorithm Flowchart B-26
B.12 Simulation Results for the initial algorithm B-27
B.13 The Recursive Algorithm Flowchart B-28

Symbol	Description
α	The angle between the cone centre and the cone edge
θ	The angle between the x-axis and the line joining the left and right wheels
Δt	The change in time
base A	The left point of the sonar cone
base B	The right point of the sonar cone
base C	The centre point of the sonar cone
C	The vector containing the x and y co-ordinates of the centroid
cb	The end point of the centre of the sonar cone
cl	The end point of the left side of the sonar cone
cr	The end point of the right side of the sonar cone
cL	The vector of the from base A to cl; the vector of the left sonar cone
cR	The vector of the from base B to cr; the vector of the right sonar cone
cV	The vector of the from base C to cb; the vector of the centre of the sonar cone

dist	The distance between the point that the sonar has detected and the robots centroid
E_L	The error added to the left wheel
inCone	1 When the right angled point is not in the sonar cone, 0 when it is
L	A vector containing the x and y co-ordinates of the left wheel
mapA	A matrix containing the initial x and y co-ordinates of each wall
mapB	A matrix containing the final x and y co-ordinates of each wall
mapL	A matrix containing the wall lengths
mapV	A matrix containing the wall vectors
onSide	1 when the wall is on the incorrect side, 0 when it is on the correct side
onWall	1 when the right angled point is not on the wall, 0 when it is
R	A vector containing the x and y co-ordinates of the right wheel
Rot	Rotation matrix
R_s	The maximum range of the sonar

$\vec{ral}$	The line which is at right angles to the wall and goes through the centre of the sonar cone
rap	The point of intersection between rap and the wall
S_L	The vector containing the x and y displacements of the left wheel
S_R	The vector containing the x and y displacements of the right wheel
V_R	The vector containing the x and y velocities of the right wheel
V_L	The vector containing the x and y velocities of the left wheel
W	Width of the robot

B.1 Introduction

There are many possible methods of interpreting the sensor data and controlling the robot that will enable it to navigate through the entire duct. There are also different methods of sensor placement and different sensor accuracies and range which will greatly affect the robots performance. Most of the localisation was performed using dead reckoning, position errors will greatly affect any control algorithm implemented.

A simulation of the robot would be useful to take into account position error and to analyse different navigation and mapping algorithms. This not only allows the algorithms to be perfected in an ideal system but will provide proof of concept. The real robot can be tested and compared to the model at a later stage and can then be updated to improve its similarity to the robot.

There were three main problems that had to be solved before any control algorithms could be used:

- The kinematic model which calculated the position of the robot at every time step given the wheel velocities.
- The sonar model which calculated what each sonar detected and how far the object was from the sonar.
- The wall model which ensured that the robot did not go through walls and instead reacted to them in a realistic manner.

Since each of these models were called at least once every time step care had to be taken to ensure that the code ran in the least time possible.

B.2 Kinematic Model

B.2.1 Description

The actual vehicle is controlled by two independent motors, one on each wheel. Each motor can be driven forwards, backwards and can produce different speeds.

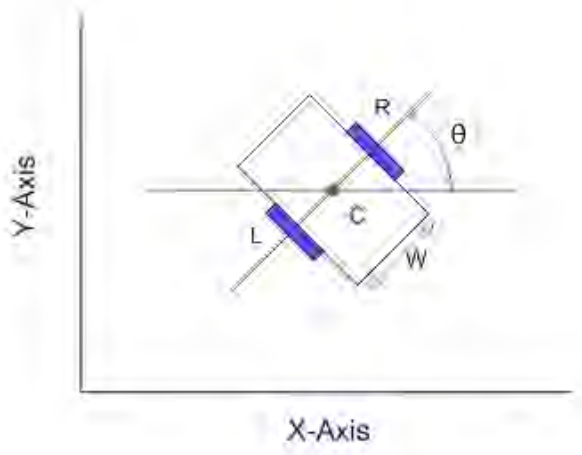

Figure B.1: A Graphic Representation of the Model's Variables

At any given time the pose of the robot can be uniquely defined by its position and orientation. The x and y co-ordinates of the centroid (C) define the robot's position in space. In this case the centroid was defined as a point half way on a straight line connecting the left wheel (L) and the right wheel (R). The orientation of the robot (θ) is the angle between the x-axis and the line joining the left and right wheels. If the position of the centroid and the orientation is known then the position of the left and right wheels can be found using the width of the robot (W). A graphic interpretation of the variables can be seen in *Figure* **??**.

B.2.2 Assumptions

To simplify the calculation of the position and orientation it was assumed that both the left and right wheels move independently in a straight line, perpendicular to the current orientation. To avoid distortion of the vehicle very small time steps must be used. After the left and right wheels have moved, the new orientation and centroid can be calculated. However, using this assumption will cause the robot's breadth to change over time. For this reason after the position of the centroid has been calculated, the position of the left and right wheel is re-calculated using the original breadth of the robot. It was also assumed that the wheels do not slip and that the robot had no length only breadth.

B.2.3 Kinematic Derivation

Given an independent velocity for each wheel, the new pose of the robot can easily be calculated. Error can be added to the system and the performance evaluated.

The rotation matrix (Rot) can be expressed as **?**:

$$Rot = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{B.1}$$

If C is situated at the origin and θ is zero, then the position of the left and right wheels can be described as follows, where W is the breadth of the robot.

$$L = \begin{bmatrix} -0.5W \\ 0 \end{bmatrix}$$
$$R = \begin{bmatrix} 0.5W \\ 0 \end{bmatrix} \tag{B.2}$$

However, if C is not at the origin and θ is not zero, then the wheel positions can be expressed for all θ in the following equations.

$$L = \left[Rot\right] \left(\begin{bmatrix} -0.5W \\ 0 \end{bmatrix} - \left[C\right] \right) + \left[C\right]$$
$$R = \left[Rot\right] \left(\begin{bmatrix} 0.5W \\ 0 \end{bmatrix} - \left[C\right] \right) + \left[C\right] \tag{B.3}$$

Given wheel velocities of V_R and V_R, the left and right wheels will move S_L and S_R respectively.

$$S_L = \left[Rot\right] (V_L \Delta t)$$
$$S_R = \left[Rot\right] (V_R \Delta t) \tag{B.4}$$

The wheel positions can then be updated:

$$L(\text{t}) = \left[L(\text{t-1})\right] + S_L$$
$$R(\text{t}) = \left[R(\text{t-1})\right] + S_R \tag{B.5}$$

Using the new wheel positions the new centroid and the orientation (θ) can be calculated

$$C_{(\mathrm{t})} = \frac{\Big[R(\mathrm{t})\Big] + \Big[L(\mathrm{t})\Big]}{2} \tag{B.6}$$

$$\theta = atan\left(\frac{R_y - C_y}{R_x - C_x}\right) \tag{B.7}$$

The assumption that each wheel acts completely independently causes a slight error in the left and right wheel position. Left uncorrected this will slowly increase the breadth of the robot. This is corrected by re-calculating the wheel positions after C has been calculated using equation **??** on page **??**.

To accurately simulate a real life robot error had to be added to the system. It was assumed that an error in wheel velocity was present on only one wheel (the left wheel was chosen) and that the error was constant. This represented the worst case scenario as in reality errors vary in magnitude and in direction. The wheel displacement with the error (E_L included can be seen in the following equation:

$$S_L = \Big[Rot\Big]\,(V_L + E_L)(\Delta t) \tag{B.8}$$

B.3 Sonar model

Given a perfect sonar transducer the ability to detect an object relies on the range of the given transducer and the beam angle. A sonar model was developed to ascertain what objects would be detected by each of the three sonars.

The setup of the front sonar is shown in *Figure* **??**. R_s was defined as the sonar range and *baseC* as the centre point of the sonar cone. Three vectors were defined that described the sonar cone; the vector of the left of the cone ($c\vec{L}$), the vector of the right of the cone ($c\vec{R}$) and the vector of the centre of the cone ($c\vec{V}$). The end points of $c\vec{L}$ and $c\vec{R}$ were defined as *cl* and *cr* respectively. Prior knowledge of the position of the centre of the sonar cone (*baseC*) and the angle at which *baseC* is positioned (*bear*) exists.

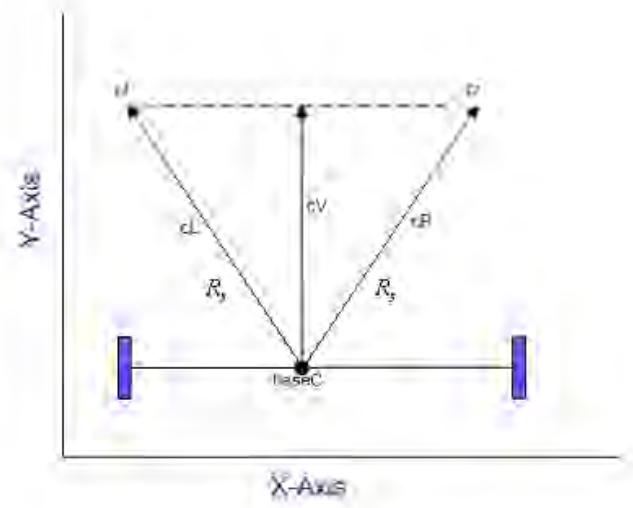

Figure B.2: A Graphic Representation of the Sonar Variables

Figure **??** shows the setup of the left and right sonars. They are positioned on the edge of the robot perpendicularly to ite . For the front sonar *bear* is θ, for the left sonar and right sonar it is $\theta + 90°$ and $\theta - 90°$ respectively. *baseC* is C for the front sonar and $C - 0.5W$ and $C + 0.5W$ for the left and

right sonars respectively.

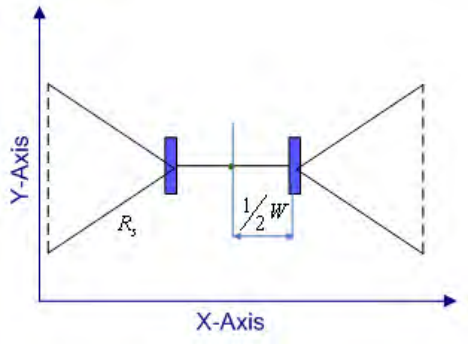

Figure B.3: Sonar Setup

B.3.1 Sonar Assumptions

It was assumed that the field of view was triangular when in fact it is of a more oval shape and that any point inside the triangle should be seen. Initially no error was added to the sonar system however objects were seen at the correct range along the sonar centre beam ($c\vec{V}$) no matter where they were placed inside the cone.

B.3.2 Sonar Derivation

Each wall was looked at by each sonar to determine which walls could be seen and which one was positioned at the minimum distance. For clarity the derivation is shown using only one wall. However when multiple walls are present the code must be vectorised to decrease computational time.

Each wall is defined as a line with a start and end point. $mapA$ and $mapB$ are matrices which contain one of the points of each wall. $mapV$, a matrix

of the wall vectors and *mapL* a matrix containing the length of each wall are created.

The effect of beam angle on visibility can be seen in *Figure* **??**. The point on the left of the figure should create a return, whereas the point on the right of the figure should not be seen as the angle of incidence is too great. It is because of this that only sonar beams with an angle of 90° to the wall will be considered.

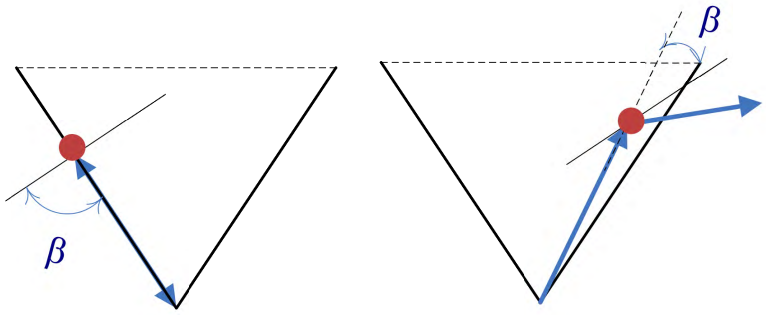

Figure B.4: Effects of Beam Angle on Visibility

The robot must not only not pass through walls, but must also react to them in a realistic way. The wall simulation assumed that should the robot come into contact with a wall it would push itself parallel to it and as such places it parallel and just off the wall.

To accomplish this the sonar algorithm creates a line which is at 90° to the wall and intersects the centre of the sonar cone. The point at which the right angled line ($\vec{ral}$) intersects with the wall is the right angled point (*rap*). An explanation of these variables can be seen in *Figure* **??**.

After $\vec{ral}$ and *rap* have been calculated a number of checks were done to ensure that the point can be seen. *rap* needs to fall in the sonar cone, on the wall under consideration and on the correct side of the robot. A diagram of how these checks work can be seen in *Figure* **??**. Every check resulted in a value of either 0 or 1. The minimum distance was calculated as the minimum distance plus all the checks multiplied by twice the sonar range. If all the checks are 0 then the distance to the point will fall within the range. Each

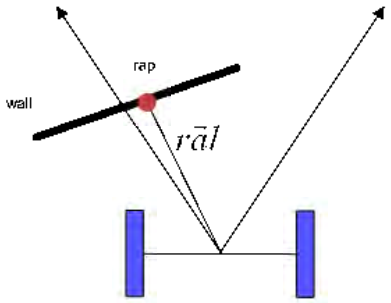

Figure B.5: Pictoral Explanation of rap and ral

check was multiplied by twice the maximum sonar range so that if any of them failed a result was created that was easily identifiable as an error.

Initial Calculation of the Right Angle Line and Right Angle Point

First the shortest path distance from the cone centre to the wall was calculated using the equation below and an explanation can be seen in *Figure* **??** [**?**, p675].

$$\begin{aligned} a &= \vec{QR} \\ b &= \vec{QP} \\ dist &= \frac{|a \times b|}{|a|} \end{aligned} \tag{B.9}$$

?

$$\vec{b} = \begin{bmatrix} mapA_x - baseC_x \\ mapA_y - baseC_y \end{bmatrix}$$
$$\vec{a} = \vec{mapV} \times \vec{b} \tag{B.10}$$
$$dist = \left| \frac{\vec{a_z}}{mapL} \right|$$

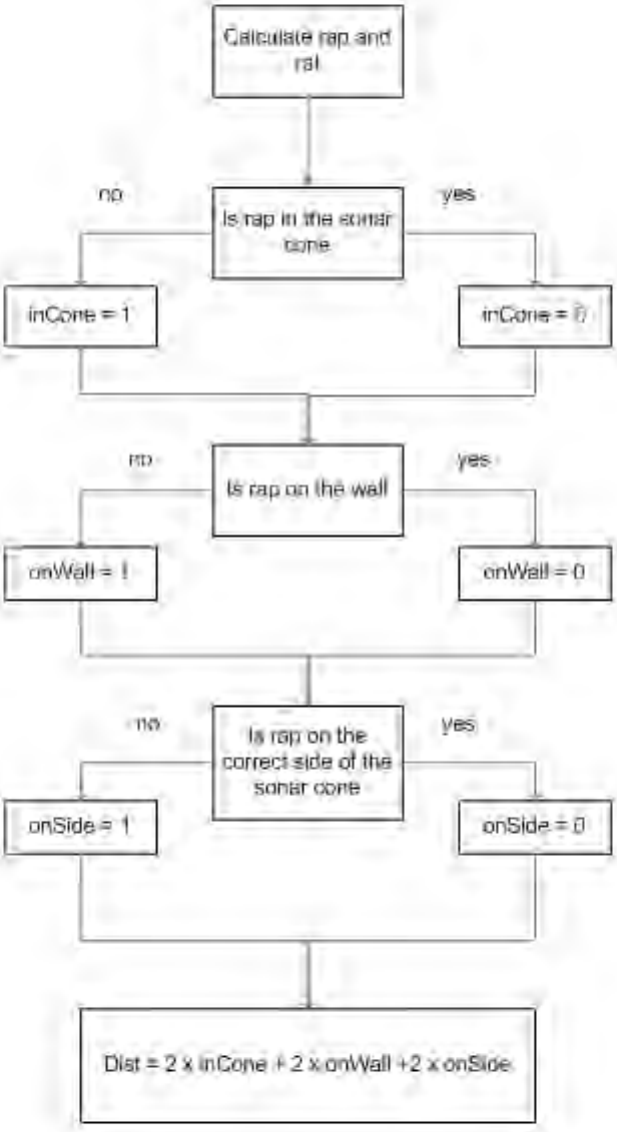

Figure B.6: Sonar Derivation Checks

Figure B.7: Explanation of the Distance Calculation

$\vec{ral}$ and *rap* are then calculated. *Figure* **??** shows a pictoral explanation of the two variables.

$$\vec{ral} = \begin{bmatrix} mapV_y \\ -mapV_x \end{bmatrix} \tag{B.11}$$

$\vec{ral}$ is then normalised

rap is merely along the vector $\vec{ral}$ at the distance calculated in equation **??** with regards to the position of the robot.

$$rap = (\vec{ral} \times dist) + baseC \tag{B.12}$$

When $\vec{rap}$ was calculated its position relied on the direction of $\vec{ral}$. Since $\vec{ral}$ could be in either direction there was a chance that $\vec{rap}$ could be placed off the wall at the distance *dist*. This can be seen in *Figure* **??**. Thus a check was done to ensure that $\vec{rap}$ was on the wall. If the distance between $\vec{rap}$ and the wall is non-zero then the direction of $\vec{ral}$ must be inverted and the position of $\vec{rap}$ recalculated.

First the vector between the wall and rap was calculated.

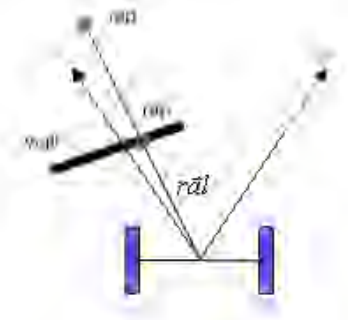

Figure B.8: Possible positions of *rap*

$$v\vec{RapMA} = mapA - r\vec{a}p \tag{B.13}$$

The distance from the wall to rap is calculated using the equation below **?**.

$$\begin{aligned} c\vec{Rap} &= \vec{mapV} \times v\vec{RapMA} \\ distRap &= \left| \frac{cRap_z}{mapL} \right| \end{aligned} \tag{B.14}$$

Once the distance from rap to the wall has been calculated a check is done to see if it is equal to zero. Either 1 and 0 is produced after this check where 1 means that *distRap* is zero and thus correct and zero means that it is non-zero and thus incorrect. This is then modified so that it is 1 for the correct *rap* and -1 for the incorrect *rap*.

The new $\vec{ral}$ is then inverted (multiplied either by 1 or -1) if need be and the new *rap* calculated.

Point within the Sonar Cone Check

A check is then performed to see whether the point is within the sonar cone. This is done by comparing the angle of $\vec{ral}$ to the cone angle.

First the angle between the centre of the cone and $\vec{ral}$ is calculated.

$$\gamma = \left|\arccos(c\vec{V}_x \times r\vec{al}_x + c\vec{V}_y \times r\vec{al}_y)\right| \tag{B.15}$$

γ is divided by the sonar cone angle. If the cone angle is greater then γ the result will be less than 1. This matrix of results is then scaled to be either 0 or 1.

This will produce a matrix, *incone*, which is 1 when the line does not fall within the cone and 0 when it does.

Point on the Wall Check

A check is then done to ensure that *rap* actually falls on the wall. (At the moment it is on the wall vector but not necessarily on the wall. The max and min values of the wall are compared with the x and y values of *rap* and a matrix of ones and zeros is then produced where 1 indicates that it is not on the wall and 0 indicates that it is.

If *rap* is less than the minimum point on the wall or greater then the maximum point on the wall it must be flagged as incorrect. A check was done to see whether it was incorrect or not. This check produced 1 when incorrect and 0 when correct.

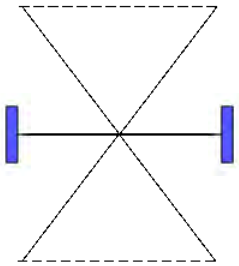

Figure B.9: The Sonar Definition

A check was then performed to see whether *rap* had the same x and y co-ordinates of the start or end points of the wall. The check was defined as 1 when *rap* was equal to the minimum of *mapA* or *mapB*.

The same check is performed to see if *rap* is greater then the maximum co-ordinates or if it also falls on the start/end point of the wall.

onWall was then produced by adding together the checks and was then scaled to produce zero if *rap* is on the wall and 1 if it is not.

Point on the Correct Side of the Sonar Cone

As can be seen in *Figure* **??** when the sonar cone was defined it extended in front of and behind the robot. Thus a check was performed to see whether *rap* was on the correct side of the sonar cone.

cl, *cr* and *baseC* were compared to *rap*. If *rap* was smaller then the minimum or greater then the maximum then pLT and pGT would be 1.

onSide was then created. If any part of pLT or pGT was 1 then the result of *onSide* would be greater then 1. *onSide* was then scaled to either 1 or 0, 1 when the wall was on the incorrect side and 0 when it was on the correct

side

$$onSide = pLT_x + pGT_x + pLT_y + pGT_y \tag{B.16}$$

Minimum Distance Calculation

To calculate the minimum distance to the wall the following equation was used:

$$distSeen = min(dist + 2R \times inCone + 2R \times onWall + 2R \times onSide) \tag{B.17}$$

If any of the checks failed then the distance to the object would be greater then the range by at least 200%. The distance was checked and any that were greater then twice the range were discarded.

Since sonar provides no directionality, the object was assumed to lie along the centre of the cone. A point was created which lay along the line cV at the distance seen which was used to generate the map.

$$mapP = baseC + cV \times distSeen \tag{B.18}$$

B.4 Wall Avoidance Algorithm

To accurately simulate the system, the robot must not be allowed to pass through walls. Not only must it stay within the boundaries of the maze, but it must also react realistically when coming into contact with a wall. The algorithm assumed that the robot would push itself until it was parallel with the wall and just touching it. As such the algorithm calculated which wall, if any, the robot had hit and then calculated its new orientation and position.

B.4.1 Wall Avoidance Derivation

Six variables were needed to perform this derivation: *L, R, direc, mapA* and *mapB*. *L* and *R* are the positions of the left and right wheel, *mapA* and *mapB* contain the co-orinates of the walls and *direc* is the direction of the robot, 1 indicating going forward, -1 indicating reverse.

Figure B.10: The Position of the Bump Sensors

To decide whether the robot had hit a wall or not, four areas where contact would be checked were defined. These areas or bumpers are represented by the dots in *Figure* **??**. Each bumper was defined as a circular area on the robot where wall contact was checked and since the robot had no length, bumpers were placed on each side, in front of the robot to simulate it. The function of the bumper simulation was to check each object for contact, as was done by the sonar simulation function described above in **??**. Because of this the sonar function was modified for this application. Instead of only

checking the small cone angle of the sonars for contact, a full 360° angle was used.

First check was done to determine if a wall had been seen, if it was on the left or right side and if it was close enough to take action. If the robot had hit a wall then the right angled line which needed to be considered was the right angled line of the wall seen.

The left and right wheel positions were updated to be just on the wall (plus a small margin) and the velocities were set so that the robot would pull away from the wall in the correct direction. The updating of the wheel positions when a left wall was hit is shown in the equation below.

$$\begin{aligned} L &= L - ral \times 0.005 \\ R &= L - ral \times W \\ vL &= direc \times \begin{bmatrix} 0 \\ 0.3 \end{bmatrix} \\ vR &= direc \times \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} \end{aligned} \tag{B.19}$$

There was a possibility that the robot hit an inside corner which means that it had hit two walls at the same time. Since only one wall was considered in the above algorithm a further check was done to prevent the robot from orientating itself incorrectly.

The inside corners were hard coded into the algorithm and the velocities reset to zero. *threshold* (the distance at which the robot was considered to have hit the wall) was set to 0.01 and *wallOffset* (how far the robot must be repositioned away from the wall) was set to 0.035.

The distance between the left wheel and all of the inside corners was then calculated and compared to the threshold. If the distance was less then the threshold then the corner that had been hit was stored in the variable, *cornerHitL*.

If cornerHitL was not equal to zero, a corner had been hit and a check was performed to decide which quadrant the corner is in. The following check assumed that all walls were at 90° or multiples of 90°.

If θ was between 45° and 135° or between 225° and 315° then the wall must be horizontal and the robot needed to be placed perpendicular to it. The robot's y co-ordinate is compared to the corner's y co-ordinate and a variable *above* was produced which either contains 1 or -1. 1 signifying the robot was above the corner and -1 signifying that it was below the corner (relative to the wall).

The left and right wheel positions are then calculated.

$$\begin{aligned} L &= \begin{bmatrix} insideCorners_x & (insideCorners_y + above \times wallOffset) \end{bmatrix} \\ R &= \begin{bmatrix} L(x) & (L_y + above \times W) \end{bmatrix} \end{aligned} \tag{B.20}$$

If the wall was not horizontal then it must be vertical and the robot must be placed perpendicular to it. The robot's x co-ordinate was compared to the corner's x co-ordinate to produce a variable called *right* which was 1 when the robot was to the right of the wall and -1 when it was to the left of the wall.

The left and right wheel positions were then updated.

$$\begin{aligned} L &= \begin{bmatrix} insideCorners_x + right \times wallOffset & insideCorners_x \end{bmatrix} \\ R &= \begin{bmatrix} L_x + right \times W & L_y \end{bmatrix} \end{aligned} \tag{B.21}$$

The wheel velocities were set so the the robot moved away from the corner it had hit.

$$\begin{aligned} vL &= \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} \\ vR &= \begin{bmatrix} 0 \\ 0.3 \end{bmatrix} \end{aligned} \tag{B.22}$$

The centroid was then recalculated using the new left and right wheel positions.

The same checks were done for the right wheel; whether it had hit a right wall or a right corner and the velocities were set accordingly.

B.5 Control Algorithm

To complete a full exploration of the duct, using the map provided a control algorithm was needed. There were two different sub-algorithms that together form the control algorithm. The basic algorithm moved the robot forward noting branches in the duct until the duct came to an end. After this occurred, the initial path has been mapped and all the branch positions were noted. The robot must return to the last branch seen, turn into it and

begin the initial algorithm again. This must continue until all branches have been mapped. Returning to the last seen branch and navigating into it while remembering where it has been and where it must still go required a more complex recursive algorithm.

The map was given to the robot using a list of the centroids and types of branches expected. This array would also serve as memory to the robot as to whether it was going into the branch or coming out of the branch. This array also served to ensure that the entire maze was visited. As a branch was completely explored, the branch entry was removed from the array. The entire control algorithm continued until this array was empty.

B.5.1 The Basic Forward Algorithm

The robot moved along and took sonar readings on each of the sonars. It calculated the difference between the left and right duct walls and altered the wheel velocities to move itself into the centre of the duct. It continued in this manner until it registered a branch in the duct. When a branch was detected, the centroid of the robot was noted and since it can no longer use the difference between the walls to keep itself in the centre it tracked the remining duct wall (if any). The type of branch (left, right or t-junction) was also noted. The robot continued to move forward, storing its centroid until the branch no longer exists. It then calculated the centroid of the duct branch by averaging all the stored centroids. Even though a map was provided it does this so that a comparison between the branches the robot saw and the actual branches can be made. These seen branches are stored in an array. A breakdown of this algorithm can be seen in *Figure* **??**.

In *Figure* **??** the results from the simulation can be seen. On the left of the figure the red line is the robot traversing through the duct. The green dots are the sensed branch centroids. On the right of the figure is the map as

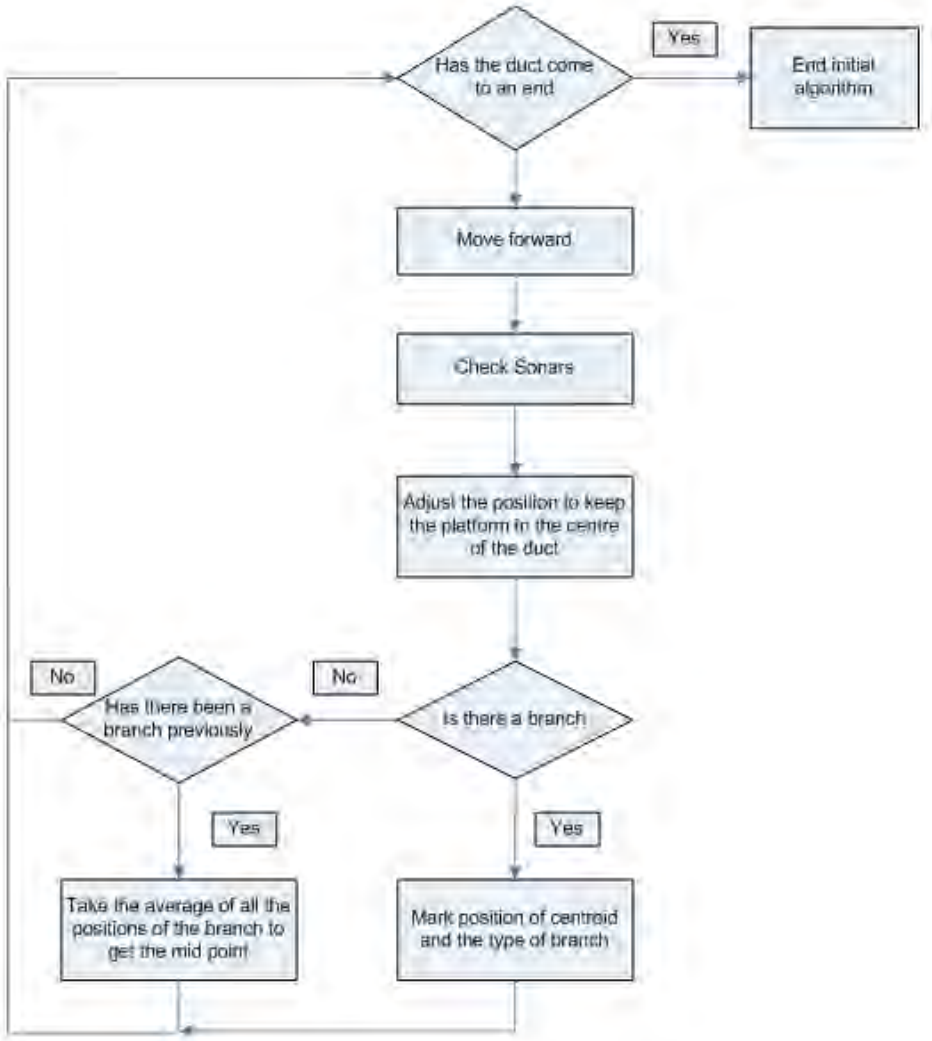

Figure B.11: The Basic Algorithm Flowchart

generated by the robot. This simulation was run with no errors in motion present, with a time step of 0.25s and a write time of 0.5s (How often the data is stored to be plotted). If the time step is increased the generated map will be less complete and the branch centroids will be less accurate. In this case the generation of the map is point based; each point seen is mapped. However, to save processing power on the robot, a topographical approach will be considered **??**.

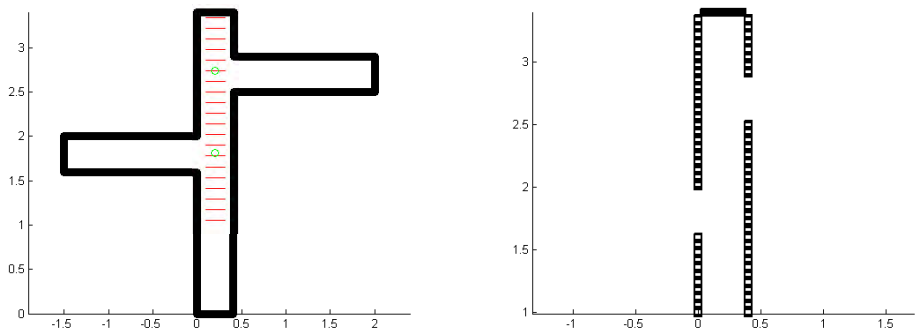

Figure B.12: Simulation Results for the initial algorithm

B.5.2 The Recursive, Reverse Algorithm

There are two important arrays populated in the recursive algorithm. The navigation array, *NavArray*, is an array of all important nodes and will be used to navigate around the duct and in the calculation of Djikstra's algorithm **?**. The branch centroid array, *branchCentroid* contains all the unexplored nodes in the system. Thus the algorithm must continue until branchCentroid is empty. Initially the branch centroids were given to the robot to eliminate errors. Thus a third array was introduced called *branchSeen*. *branchSeen* contains the centroids of all the branches as seen by the robot. If *branchSeen* is comparable to *branchCentroid* then that indicates that no prior knowledge of the branch positions is needed.

The robot must always go forward into the duct as the pan and tilt mechanism is only present on the front of the robot. However, when traversing through an already visited duct to get to a branch the robot must move backwards to prevent the tether from tangling.

The flowchart seen in *Figure* **??** shows the recursive algorithm. The robot moves backwards, using the sonar readings to keep itself in the centre of the duct until a branch was seen. If the branch matches the branch that it

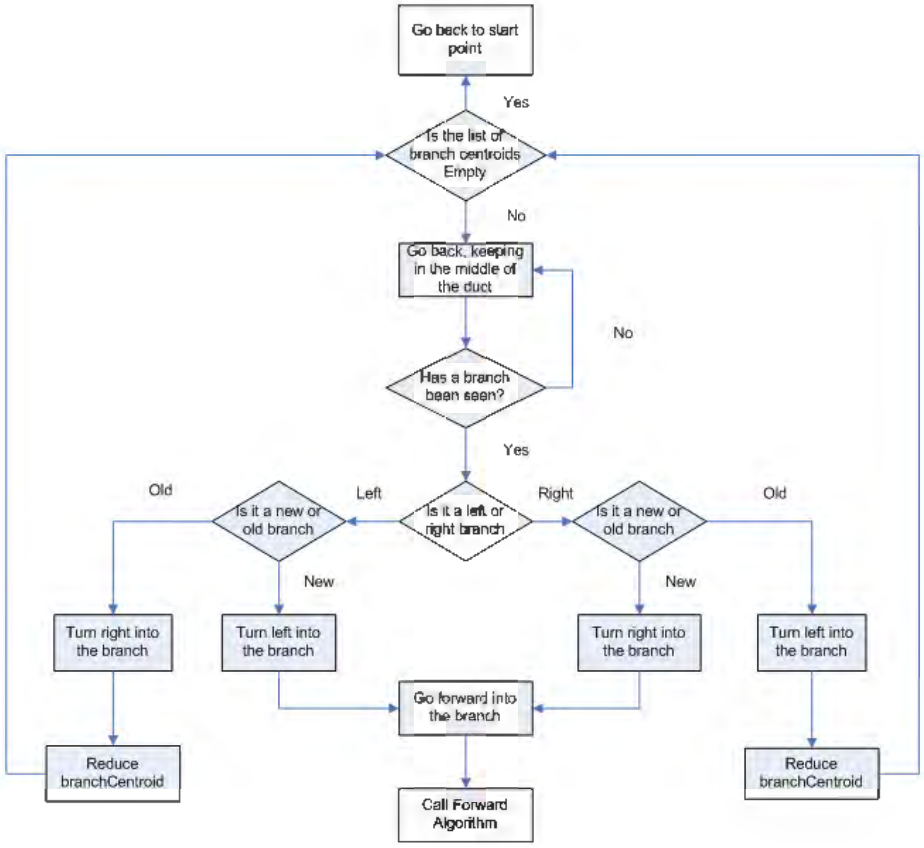

Figure B.13: The Recursive Algorithm Flowchart

expects to see (roughly the same co-ordinates and the same type of branch) then it must turn into it. However the direction of the turn depends on whether this branch must be turned into, or turned out of. If it must be turned into (which means it was unexplored and must be explored in the forwards direction) the robot must turn right if the branch is a right one, or left if it is a left branch. However, if the branch has already been explored and must be turned out of, then the robot must turn to position itself to reverse through the duct and so must turn left for a right branch and right for a left branch. It then called itself recursively, reducing the list of branch centroids until a new, unexplored duct has been or until the list of branch centroids was empty. An entry in the branch centroid array let the algorithm know if the branch was explored or unexplored. All branches were initially

given a value of 1, meaning unexplored. As the robot turned into them, this value was changed to 0, meaning explored. That variable and the side of the branch, decided the direction of the turn.

Once the list of branch centroids was empty, the algorithm was complete and the robot reversed through the duct, keeping itself in the middle using the sonars until it was roughly in its start position. The actual branch centroids were compared with the ones seen with the sonars and the map using the sonars was generated.

Appendix C

Ultrasonic Transducer

Contents

C.1 Introduction C-3

C.2 Time of Flight VS Varying Voltage C-3

C.3 Hardware C-4

C.3.1 Oscillator C-5

C.3.2 Receiver C-6

C.4 Testing C-7

C.4.1 ADC Resolution C-7

C.4.2 Return at Different Incidence Angle C-8

C.4.3 Sonar Return for Moving Objects C-9

Figures

C.1 Sonar Connection Schematic C-4
C.2 Oscillator Circuit Diagram C-5
C.3 Gain Stage Circuit Diagram C-6
C.4 Peak Level Detector Circuit Diagram C-7
C.5 ADC Results for an Object at a Constant Distance C-8
C.6 Return Signal As the Angle of Incidence Increases C-9
C.7 Sonar Reading for a Moving Sonar C-10
C.8 Sonar Response of To Duct Branch C-10

C.1 Introduction

To perform any autonomous algorithm the absence of duct walls or duct branches had to be detected. Ultrasonic transducers were chosen to perform this function as they are cost effective and can also provide range measurements. Further information about the reasons for sonar choice and the limitations involved can be found in the background theory **??**. The sonar that was implemented was the Murata MA40B8R/S **?** which is able to act as both a receiver and transmitter. Its nominal frequency is 40 kHz with a resolution of 9mm. To discriminate between planes, corners and edges at least two transmitters and two receivers are necessary. Having more then one transmitter with one receiver or more then one receiver with one transmitter makes corners and planes indistinguishable **?**. In this application only one transmitter and receiver were used since the ability to distinguish corners from planes was not deemed important when compared with the extra circuitry and processing power that would be needed.

C.2 Time of Flight VS Varying Voltage

Generally when sonars are used, **T**ime **o**f **F**light (TOF) is implemented. The transmitter sends out a chirp signal of a short duration, starts a timer and waits for a return signal. When a signal has been received the timer is stopped, the time taken to return is measured and the distance is calculated. This is the most accurate method of sonar use, however it requires an extra timer and more intense computation as the program must wait a specified amount of time for the return signal. As stated by Adams **?**:

"The first point to be noted with SONAR is that no time of flight range value can be produced if the detected signal amplitude does not exceed a preset

threshold value."

The return of a sonar signal is a sin wave where the amplitude is dependent on the distance to the target. This indicated that the amplitude of the wave could be used to calculate the distance to the target. The result would not be as accurate as TOF as it suffers from electronic noise and is not always linear. The resolution of the available ADC can cause even more errors and the fact that the transmitter is always on can increase the number of false returns. In this case accuracy was sacrificed for faster computational time and decreased computational complexity. The lack of timers available on the Roboaudiostix was also a factor in the decision to not use TOF. For this reason a varying voltage was considered preferable to time of flight.

C.3 Hardware

Figure **??** shows how the various parts of the sonar circuits were connected. A 40kHz oscillator was connected to the transmitter, while the signal coming from the receiver was gained, filtered and rectified.

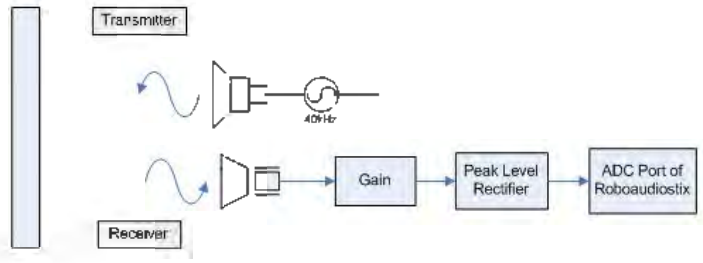

Figure C.1: Sonar Connection Schematic

C.3.1 Oscillator

To drive the transmitter a 40kHz oscillator was needed. A CMOS Low-power astable multivibrator 4047 **?** was used to create two antiphase square waves on Q and $\overline{Q}$. When the transducer was connected between them, a 40kHz square wave with double the supply voltage was produced. The frequency was set by R2, R3 and C2, using the formula below:

$$f = \frac{1}{4.4RC} \tag{C.1}$$

As the transducer was very sensitive to any frequency changes, a potentiometer (R3), was added to allow frequency adjustment. The 4047 has a tendency to start up and never time out. To prevent this, R1 and C1 were added and provided a positive pulse on the reset pin for a short period after startup. The final circuit diagram can be seen in *Figure* **??**.

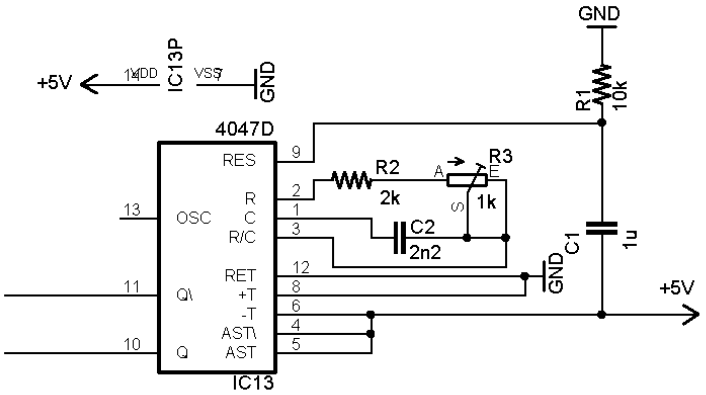

Figure C.2: Oscillator Circuit Diagram

C.3.2 Receiver

Gain Stage

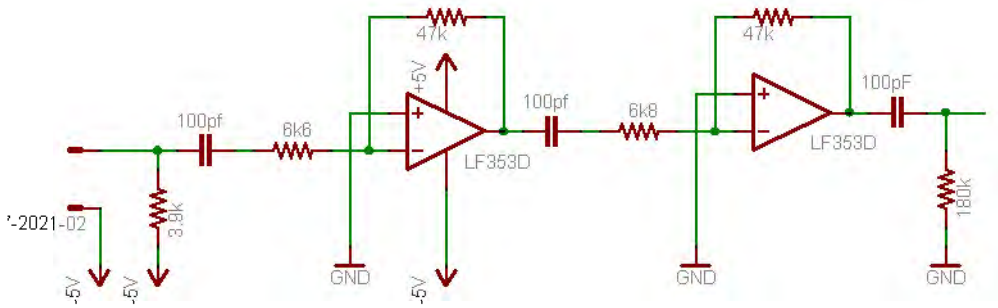

Figure C.3: Gain Stage Circuit Diagram

Once the sound wave bounced off an object it induced a 40kHz sin wave in the receiver. The closer the object the greater the amplitude of the wave. This amplitude was very small and needed to be gained in order to be interpreted. The circuit shown in *Figure* **??** used two LF353 opamps to provide the gain of approximately 48. Since the operating frequency was high, opamps with a high slew rate were needed. The capacitors were added to remove DC noise.

Rectification

Since an analogue voltage was to be used to interpret the distance, the signal needed to be rectified. After being gained, the signal was fed into a peak level detector, shown *Figure* **??**. This circuit followed the positive peak of the signal and held it in the capacitor, C1. To set the response time, a resistor (R1) was added across C1.

A comparator was added after the signal was rectified. This could be used should TOF need to be implemented by recognising the return signal as any signal above general noise or could be used as an interrupt to signal a particular distance.

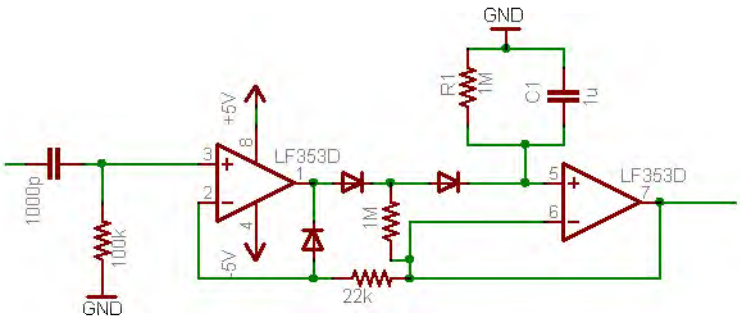

Figure C.4: Peak Level Detector Circuit Diagram

The rectified signal was then fed into an ADC port of the Roboaudiostix.

C.4 Testing

C.4.1 ADC Resolution

The ADC on the Roboaudiostix had a resolution of 10 bits. As can be seen in equation **??**, this amounted to 4.8mV per step over a 5V range.

$$10\,bits = 1023\,bytes$$
$$\frac{5}{1023} = 4.8 \times 10^{-3} mV/step \tag{C.2}$$

Because since the noise on the signal being fed to the ADC was greater then 4.8mV it would make the reading harder to interpret and so the 8 bit ADC was implemented. This not only reduced the noise and stabilised the system, but also decreased the total amount of data that had to be transferred. To

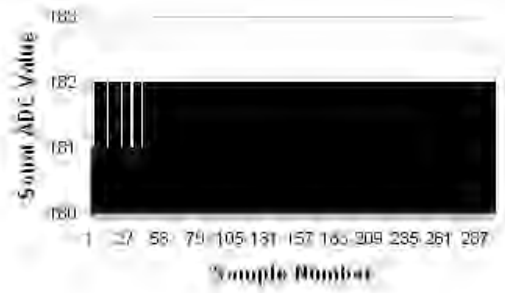

Figure C.5: ADC Results for an Object at a Constant Distance

test the stability of the reading, an object was placed in front of the robot and kept at a constant distance. *Figure* **??** shows that for a stationary object the varying voltage method was stable.

The minimum range of the sonars was 30mm ± 2mm. The sonars were set further back along the base so that walls closer then this could be detected if needed.

C.4.2 Return at Different Incidence Angle

The magnitude of a sonar return signal depends on the angle at which the sonar beam hits the object (angle of incidence). As this angle gets larger, the time taken to reach the object and return to the sonar will get longer until eventually the return will disappear entirely.

To test the sonar return at different angles of incidence, the platform was placed facing a wall. The wall was kept at the same distance to the platform and was rotated relative to the robot.

Figure **??** shows the return signal as the angle of incidence changes. As

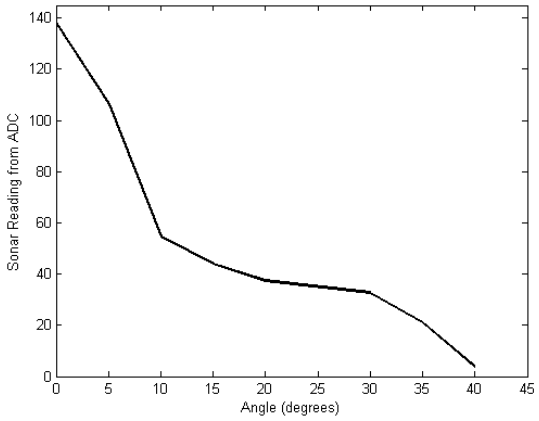

Figure C.6: Return Signal As the Angle of Incidence Increases

expected the return signal is indirectly proportional to the angle of incidence. At about 35° the return signal can be considered as absent and between 5 and 8° the signal was at its maximum. This was consistent with the results from Fonseca *et all* **?** which state:

"For small incidence angles (less then 8 degrees) the return was at its maximum".

C.4.3 Sonar Return for Moving Objects

Once the stability of the system had been tested using a stationary object, tests were run on moving objects to note the response of the sonar readings. First the platform was made to move toward an object stationed at roughly 90° and the sonar readings stored. The results can be seen in *Figure* **??**. As expected the sonar return increased as the object came closer. The irregularities in the graph are caused by small angle and reflectivity changes as the position of the robot changes.

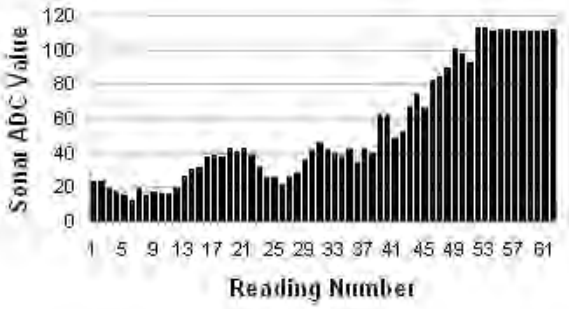

Figure C.7: Sonar Reading for a Moving Sonar

A test was also performed on the sonar's ability to recognise a duct branch. The platform was moved past a wall at a constant distance until it reached a branch and *Figure* **??** shows the results. Since the sonar circuitry stored the distance as a charge there is a finite discharging time. When the branch was detected, the value was effectively zero, but the finite discharging time created the slope seen on the figure.

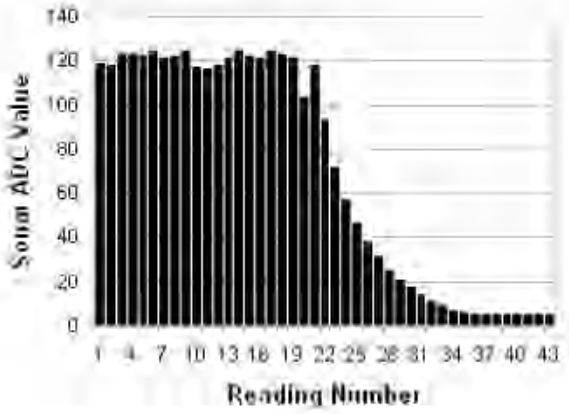

Figure C.8: Sonar Response of To Duct Branch

Appendix D

Speed Control

Contents

D.1 Introduction D-3

D.2 Proportional Control D-3

D.3 Drive System D-4

D.4 Software D-6

D.4.1 Floating Point Errors D-6

D.5 Step Tests to Optimise Kp D-6

D.5.1 Repeatability of the Motor Response D-7

D.5.2 Optimising K_p for One Motor D-8

D.5.3 Matching the Response of Both Motors D-9

Figures

D.1 The Control Algorithm D-3
D.2 Overview of the Drive System D-4
D.3 Optcoupler Circuit Diagram D-5
D.4 Optical Switch Circuit Diagram D-5
D.5 Four Runs Using $K_P = 1$ D-7
D.6 Altering K Ssing the Same Set Speed and Motor D-8
D.7 Response of Both Motors Using $K_p = 0.25$ D-9
D.8 Comparison of Other K_p Values D-10

D.1 Introduction

Given a constant PWM signal there is no guarantee that a motor will run at the same speed under all conditions. A motor under zero load will run faster then one under load even with the same PWM signal. If it is important that the desired speed is close to the actual speed and that the speed remains constant then some form of feedback or closed loop control is needed. Since odometry was to be used to calculate the position of the platform, constant speed was imperative.

D.2 Proportional Control

Variable	Description
R(t)	Motor Output Function Over Time t
$V_{act}(t)$	Actual Motor Speed At Time t
$V_{des}(t)$	Desired Motor Speed At Time t
K_p	Constant Control Value

Table D.1: Variables used

Proportional control makes use of the following equation [**?**, pp 57-65]:

$$R(t) = K_p \times (V_{des}(t) - V_{act}(t)) \tag{D.1}$$

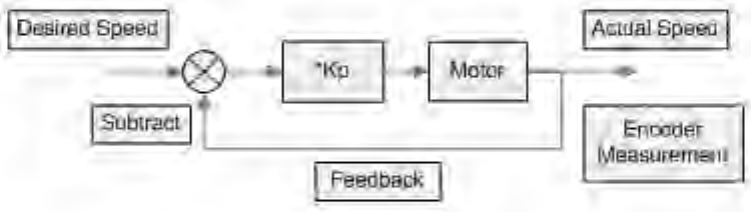

Figure D.1: The Control Algorithm

The difference between the desired velocity and the actual velocity is called the error function. K_p is a constant that can be varied, however, varying K_p will vary the behavior of the control loop. The higher K_p becomes the faster the system will respond, but too high a value will cause the system to oscillate. A low K_p will not only respond slowly, but will also maintain a large steady state error. The feedback loop is shown in *Figure* **??**.

D.3 Drive System

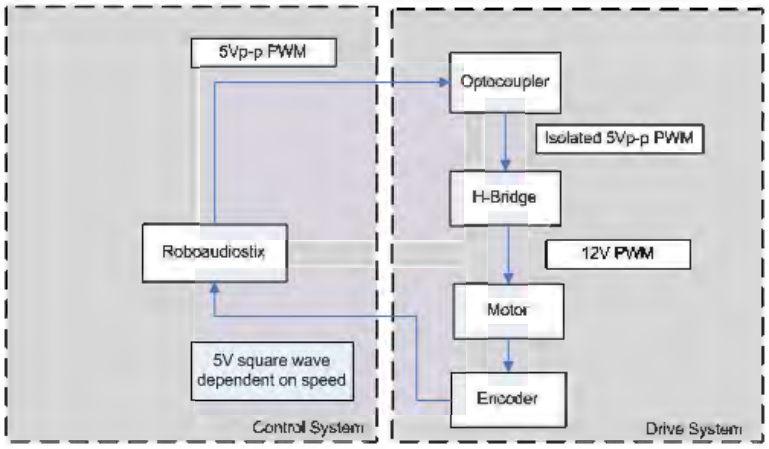

Figure D.2: Overview of the Drive System

Before speed control was implemented the motor control system had to be designed. An overview of this system can be seen in *Figure* **??**. The Roboaudiostix controlled the system using 5V peak to peak PWM of varying duty cycles. This signal was passed through optocouplers to ensure that none of the electronic noise from the motors was passed into the sensitive circuity and the circuit can be seen in *Figure* **??**. After being optoisolated, the signal was sent into an LMD18200 H-bridge **?** to provide the motors with a PWM signal of 12V and up to 3A.

Speed control requires some manner of measuring the actual speed and so a

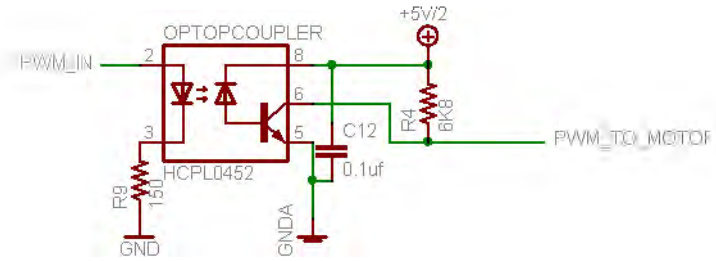

Figure D.3: Optcoupler Circuit Diagram

slotted disk or encoder was placed on the motor shaft and a phototransistor optical interrupter switch **?** was used to detect these slots as they passed. The more slots or ticks that went past in a fixed amount of time, the greater the speed. The number of slots that are machined into the encoder effect the accuracy of the system as more slots increase the resolution.

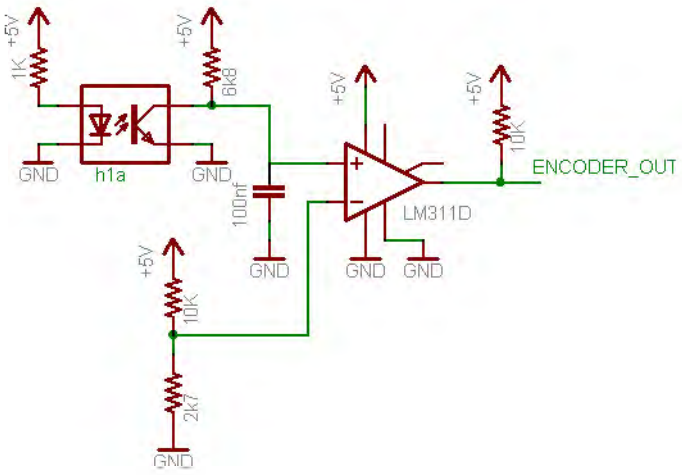

Figure D.4: Optical Switch Circuit Diagram

Figure **??** shows the circuitry connected to the optical switch. The LED and phototransistor were powered and the output was sent into a comparator to produce a 0-5V signal. This was fed into an interrupt pin on the Roboaudiostix.

D.4 Software

Speed control using a phototransistor optical interrupter switch can be done using input capture, where the rising or falling edges of the signal are counted. Because a limited number of timer modules were available, this method was not ideal. As such each of the two 0/5V signals from the encoders were fed into an interrupt pin of the Roboaudiostix. Each time a slot went past, the interrupt pin was triggered and a variable was increased. A timer was implemented as an interrupt so that every 8 timer interrupts the speed control algorithm could be performed. This involved comparing the actual speed (number of slots that went past or number stored in the variable generated by the external interrupt pin) to the desired speed which was set by the user or control algorithm. Once speed control had been performed the variables were set to zero and the process repeated. This allowed only one timer and two interrupt pins to be used.

D.4.1 Floating Point Errors

Since K_p could take on values with a decimal point and since the Roboaudiostix allows no floating point operations, some data manipulation had to take place. K_p was multiplied by 100 and the result from the control algorithm divided by 100. Any numbers beyond the decimal place were lost. This caused very small errors in the control algorithm.

D.5 Step Tests to Optimise Kp

It was necessary to perform tests on the system to ensure that the data was accurate and repeatable and to find the best K_p value for each motor. In

each of the following tests the motor was run unloaded and the desired speed was chosen to be 60.

D.5.1 Repeatability of the Motor Response

Before tests were run to determine the optimum K_p, it was necessary to establish whether the speed control was repeatable. For this reason four runs were performed using a K_p of 1. The speed of the motor is represented by the number of slots going past the phototransistor, or ticks, while time is represented by timer interrupts.

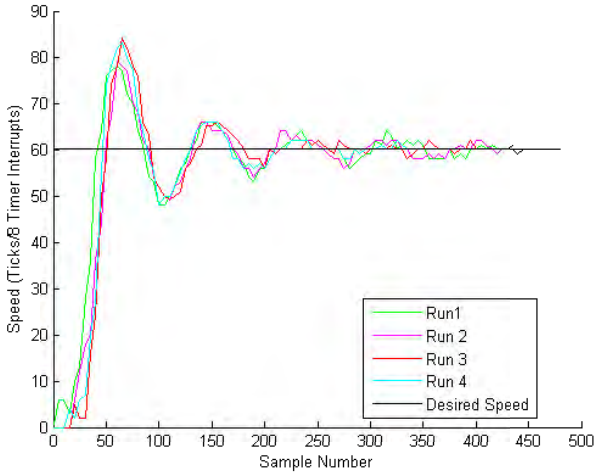

Figure D.5: Four Runs Using $K_P = 1$

Figure **??** shows the results of the four runs and as can be seen they are very similar. Thus it can be said that the speed control is repeatable enough to perform only one run per value of K_p without becoming inaccurate.

D.5.2 Optimising K_p for One Motor

Since the value of K_p has great impact on the response time of the system tests were done to optimise it. Different values of K_p were chosen and the speed control was performed until the system reached equilibrium.

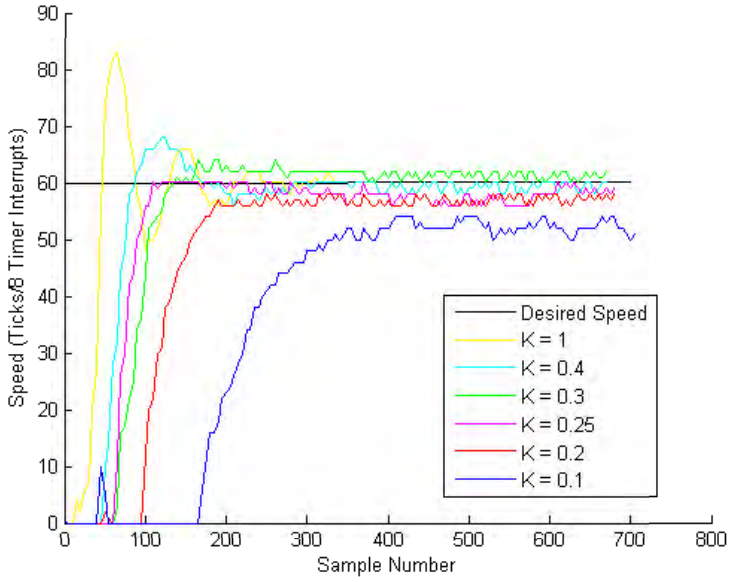

Figure D.6: Altering K Ssing the Same Set Speed and Motor

Figure **??** shows the results from the different test runs. At $K_p = 1$ the system began to oscillate. This means that 1 is too high. At $K_p = 0.1$ the system was very slow to respond and at equilibrium maintained a large steady state error and thus was too low. A K_p of between 0.25-0.3 could be chosen as neither oscillated or maintained a steady state error and both responded quickly. The frequency of each reading occured at roughly 3kHz so the time to reach equilibrium was minimal. $K_p = 0.25$ was chosen as it reached the required speed faster.

D.5.3 Matching the Response of Both Motors

Since no two motors are alike, it was necessary to make sure that the speed responses were the same for a given K_p. If not, the K_p matching would have to be done to find the correct value. *Figure* **??** shows the response of both of the motors with the same K_p. As can be seen both the motors responded almost identically with the same value of K_p.

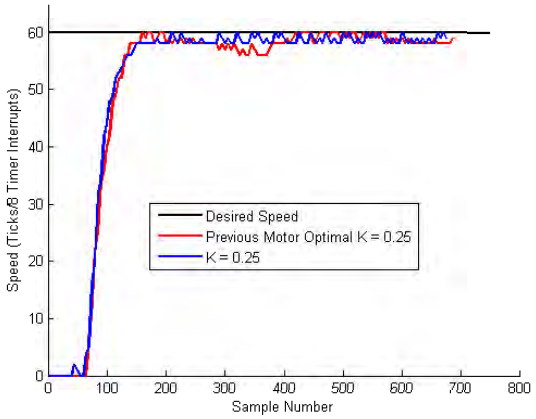

Figure D.7: Response of Both Motors Using $K_p = 0.25$

To confirm that the response using $K_p = 0.25$ for the second motor was really the closest match to the speed response of the first motor, two more runs were performed using different K_p values. These runs can be seen in *Figure* **??**, which shows that using a K_p of 0.25 for the second motor did indeed provide the closest match.

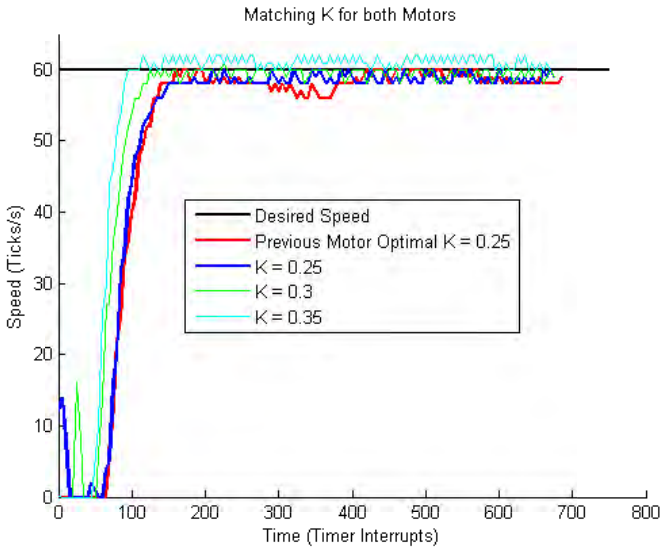

Figure D.8: Comparison of Other K_p Values