INTELLIGENT POTHOLE REPAIR VEHICLE

A Thesis

by

RUZBEH ADI MINOCHER HOMJI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2005

Major Subject: Mechanical Engineering

INTELLIGENT POTHOLE REPAIR VEHICLE

A Thesis

by

RUZBEH ADI MINOCHER HOMJI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Won-jong Kim
Committee Members,	Make McDermott
	Aniruddha Datta
Head of Department,	Dennis O'Neal

August 2005

Major Subject: Mechanical Engineering

ABSTRACT

Intelligent Pothole Repair Vehicle. (August 2005)

Ruzbeh Adi Minocher Homji, B.E., Marine Engineering & Research Institute

Chair of Advisory Committee: Dr. Won-jong Kim

This thesis presents an endeavor to design and construct a prototype of an automated road repair vehicle called the Intelligent Pothole Repair Vehicle (IPRV). The IPRV is capable of automatically detecting and filling potholes on road surfaces without operator assistance. An easy-to-construct mechanical means of pothole detection was employed to reduce costs and complexity that have thus far been the primary disadvantage of automated road repair vehicles. A network interface to an Ethernet was designed based on the transmission control protocol (TCP) to enable remote operability of the IPRV. A laptop computer was used onboard the IPRV for control and interfacing using a data-acquisition card installed on it. The Visual Basic® programming language and the Windows application programming interface (API) were used for all the programming requirements of this thesis. The IPRV employs feedback mechanisms for position control and path following. Operation has been designed to incorporate safety mechanisms that ensure that the IPRV automatically stops in the case of a loss of communication link or large network delays. Experiments were performed to test and calibrate the IPRV. The IPRV was designed to detect potholes that have a maximum depth greater than 2 cm.

To my wife Shweta

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere gratitude to my advisor, Dr. Won-jong Kim, for the immense confidence that he showed in my abilities. I would also like to thank him for the invaluable time and guidance that I received from him throughout the course of this thesis research.

I wish to extend my thanks to Dr. Aniruddha Datta and Dr. Make McDermott for serving on my thesis committee. I truly appreciate their time and effort in reviewing my thesis.

A special thanks goes to my good friend Yusuke Kawato for providing a helping hand whenever I needed it and for his invaluable suggestions. I would also like to thank Dr. Shobhit Verma, Adam Rogers, and Kun Ji for their help and support.

Most of all, I would like to thank my wife and best friend, Shweta, without whose love, understanding and encouragement this thesis would never have become a reality. I promise to make up for all the times I have neglected her for my thesis.

Lastly I would like to thank my mother, Harjeet Homji, who is solely responsible for my being where I am today. She has supported and encouraged me at every step of my life. Thank you.

TABLE OF CONTENTS

CHAPTER Page

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Introduction

Vehicular traffic has been rapidly growing over the recent years with more privately owned vehicles taking to the streets each day. Today, trucks weigh significantly more than ever before and are capable of carrying much larger payloads. The situation is further exacerbated by the decline of railroads. These factors in conjunction with inclement weather result in a major challenge that transportation departments throughout the country face – road damage in the form of potholes.

Potholes are not only the cause of significant damage to vehicle suspension systems but may, in severe cases, result in serious accidents and permanent injury. Year-round pothole repairs are also a major reason for the depletion of state funds. The United States alone spends billions of dollars every year on pavement maintenance. Thus there is an impending need for pothole repair techniques that are cost effective as well as long lasting.

This chapter begins with a description of the causes for pothole formation in Section 1.2. This is followed by a description of the types of materials and techniques commonly used for pothole repair. The spray injection technique of pothole repair is discussed in some detail leading to the need for an automated pothole repair vehicle. Section 1.3 delineates the research objectives of this thesis. Section 1.4 lists the major

---

This thesis follows the style of *IEEE Transactions on Automatic Control*.

contributions of this thesis. The final section provides an overview of the organization of the thesis.

1.2 Potholes – causes and repair methodology

"A pothole is any pavement defect involving the surface or the surface and base, to the extent that it causes significant noticeable impact on vehicle tires and vehicle handling. All potholes are the result of the interaction of water and traffic on pavement. Most are found on local road and street systems: 80% of the nation's roads are local roads and are more apt to have "just grown" rather than being planned from the start and are much more likely to have water, gas and other utilities underneath. [1]"

1.2.1 Pothole formation

The development of potholes is due to the simultaneous presence of two factors, water and traffic. These factors may cause potholes in two basic ways. Fatigue failure occurs due to excessive flexing of the pavement. Water due to melting snow, rainfall, or bad drainage weakens the soil below the pavement. In this weakened condition, traffic on the pavement causes the pavement to start flexing. This flexing eventually leads to cracks followed by breakage. Thinner pavements are more prone to this type of potholing [2]. Figure 1.1 shows a pothole cause by fatigue failure.

Raveling failure occurs when water on the pavement washes away the adhesive asphalt films that hold the stone aggregate together. Traffic on such pavements causes a gradual raveling away of the stone particles. Such a condition occurs when water has a

chance to permeate a pavement that lacks sufficient density to prevent water penetration [2].

The best way to minimize road damage is to follow a carefully planned preventive maintenance system. This includes the laying-out of well-planned roads, using proper resurfacing methods, ensuring adequate drainage facilities, regularly checking drains for blocks, and carrying out road repairs as soon as possible to prevent further deterioration.

Figure 1.1. A pothole caused by fatigue failure
(Source: dot.ci.tucson.az.us/streets/normal.htm)

Despite the best measures taken by state transport authorities, the development of potholes is inevitable. Preventive maintenance can at best delay their occurrence. It is thus essential to simultaneously focus on continuously improving pothole repair methods.

1.2.2 Pothole repair methodology

Current research on pothole repair can be divided into two broad categories.

1) *Repair Materials* – Typically the different types of mixes that are used for pothole patching are hot-mixes, cold-mixes, heated cold-mixes, and recycled mixes. Hot-mixes from an asphalt plant are the best material for patching potholes [2–3]. However the use of hot-mixes is limited due to their unavailability in the winter season as asphalt plants are closed at the time. Also, hot-mixes do not perform satisfactorily when used in wet potholes [3].

Most agencies make use of one or more of three types of cold asphalt mixes that are available to them – cold-mixes produced by local asphalt plants using locally available aggregate and binder, cold-mixes produced according to agency specifications including acceptable types of aggregate and asphalt, and proprietary cold-mixes that use specifically formulated binders [4–5]. The latter two types of cold-mixes have to be checked for the compatibility of the binder and the aggregate. Proprietary cold-mixes include high-performance mixes with anti-stripping and adhesive agents. While being more expensive, these high-performance mixes significantly increase the service life of the repair and are a better alternative for pothole repair [6–7].

2) *Repair Techniques* – Four types of repair techniques are commonly utilized for pothole patching as described in [4–5].

a) Throw-and-roll – This method consists of placing the patching material into the pothole and then compacting the patch using truck tires. The compacted patch must have

a crown between 3 mm and 6 mm. Figure 1.2 shows the filling stage of the throw-and-roll method.

Figure 1.2. The throw-and-roll procedure – material placement
(Source: [5])

b) Semi-permanent – This method consists of removing the water and debris from the pothole. The sides of the patch area are then squared-up and the mixture is placed into the pothole. This is followed by compacting the mixture.

c) Spray Injection – This method consists of blowing water and debris from the pothole. The sides and bottom of the pothole are then sprayed with a tack coat of binder. Next, aggregate is simultaneously premixed with heated asphalt emulsion and sprayed into the pothole, and finally the patched area is covered with a layer of aggregate. The spray injection method does not require compacting.

d) Edge Seal – Like throw-and-roll, this method consists of placing the mixture in the pothole and compacting it using truck tires. Once the patch has dried, a ribbon of asphaltic tack material is placed on the patch edge and a layer of sand is placed on the tack material.

Another method often used is the throw-and-go method [4–5]. This involves the placing of mixture into potholes followed by little or no compaction. While this is the most expedient way of pothole repair, it is also the least effective. The throw-and-go method can significantly increase long term expenditures and must not be used as a means for pothole repair.

The throw-and-roll method has proved to be very effective when performed using high-performance mixes. With high quality mixes the throw-and-roll method has been shown to be as effective as the semi-permanent method and is also comparatively less labor intensive [8]. The semi-permanent method also has higher equipment cost and lower productivity [6].

The spray injection method is a very effective and widely accepted method for pothole patching. It offers potential for greater productivity and efficiency and can operate in extreme cold weather [9]. Along with the throw-and-roll method it produces the highest quality repairs and is the most cost effective in the long run [6].

1.2.3 Spray injection patching

There are three types of units used for spray injection pothole patching as described in [9].

1) *Trailer-Type Unit* – In this unit, a dump truck pulls a trailer and feeds aggregate through a modified tailgate into the trailer unit. A minimum of two people are required. One person works behind the trailer to control a delivery hose suspended from a boom on the rear of the unit [9].

2) *Modified-Truck Unit* – Here the patching equipment is mounted on the chassis of an existing Department of Transportation (DOT) truck. The need for a trailer is eliminated; the spray injection hose and boom are still operated from the rear of the truck [9].

3) *Self-Contained Unit* – Only one person is required to patch the pothole. The spray injection equipment is built into the truck chassis. Patching is carried out by the truck operator using a joystick to remotely control the spraying operations. The boom and attached hose extend from the front of the truck [9]. Figure 1.3 shows a self-contained unit.

The self-contained unit has been found to perform extremely satisfactorily in all conditions. However, a major disadvantage of this type of unit is the initial capital expenditure in the range of $120,000 [9]. This is off-set if long term operational costs are taken into account. The "IDOT (Illinois Department of Transportation) has estimated that using one self-contained truck unit in seven maintenance districts would result for each district, in a labor savings of 53 person years over a 10-year cycle; material and equipment savings would be $1.05 million. [9]"

Figure 1.3. Spray injection device – self-contained unit
(Source: www.tfhrc.gov/focus/archives/focus399/0399cal.htm)

As is evident from the preceding discussion, over the years, advancement in technology has played a tremendous role in increasing the lifetime of repair patches while at the same time reducing costs. High-performance materials and equipment (like self-contained units) are replacing conventional repair methods. The next logical advancement in pothole repair techniques is the automation of road repair. Automation will eliminate the need for expensive labor and produce consistent results. Cost savings that can be derived by using an automated road repair process are estimated in [10]. Thus far, impediments in the success of automated road repair vehicles have been their high initial cost and the complexity of pothole detection.

This thesis describes the design and construction of a novel prototype road-repair vehicle that automates road repair providing an easy-to-construct, cost-effective mechanical means of pothole detection.

1.3 Thesis objectives

The key objective of this thesis is the design and construction of a semiautonomous mobile vehicle capable of automatically detecting and filling potholes on road surfaces. The vehicle is called the Intelligent Pothole Repair Vehicle (IPRV) and is capable of being maneuvered remotely over a wireless local-area network (LAN). The main objective of this thesis can further be broken down into the following objectives:

1. design and construction of the autonomous vehicle
2. design and construction of a pothole detection and filling mechanism
3. design of an interface to control the position and direction of the IPRV
4. development of a software platform to control the IPRV hardware
5. writing of an algorithm to automatically detect and fill potholes encountered by the IPRV
6. interfacing the IPRV with a LAN
7. selection of a transport protocol to be used for the network interface

1.4 Thesis contributions

The most significant contributions of this thesis are listed below.

1. The IPRV is remotely operable over a wireless LAN. It is also capable of semiautonomous operation wherein it detects and fills potholes without operator

assistance. Thus far, all existing automatic road repair vehicles have required an operator inside the vehicle.

2. The IPRV uses an easy-to-construct, mechanical means of pothole detection. This significantly reduces the processing requirements of the controller, which in turn reduces construction costs. Automated vehicles in the past used video image processing to detect potholes, which is a computationally intensive method requiring large processing capabilities.

1.5 Thesis organization

Chapter I describes the causes of pothole formation and the most common repair materials and techniques used in pothole repair. It provides a brief comparison among the various materials and techniques available in terms of pothole repair lifetime expectancy and cost effectiveness. The spray injection technique is described in some detail leading to the need for automation in pothole repair.

Chapter II presents all the relevant literature reviewed by the author. The literature review is divided into 3 categories, pothole repair materials and techniques, pothole detection, and automated road repair vehicles.

Chapter III describes in detail the mechanical design of the IPRV. The description is categorized according to the 3 stages of development of the IPRV: Semi-autonomous mobile vehicle design, pothole detection module design, and pothole-filling module design.

Data acquisition and interfacing is described in Chapter IV. This chapter begins with an introduction section followed by a description of the PCMDIO data acquisition

card being used. The next three sections describe the interfacing of the laptop computer with the motor controller, the filler valve, and the onboard sensors.

Chapter V describes in detail the software design. The choice of the programming language for the thesis is discussed followed by a detailed description of the algorithms employed for controlling the various components of the IPRV hardware. This is followed by a description of the network interface designed to remotely control the IPRV. The final section describes the client-side user interface.

Chapter VI starts with a description of the operation of the IPRV as a single unit integrating all the aspects of the design. The next section contains all the experiments conducted for the measurement, calibration, and testing of the IPRV.

Chapter VII summarizes the achievements of this thesis. The current limitations of the IPRV are provided and future work towards further developing the IPRV is proposed.

CHAPTER II

LITERATURE REVIEW

2.1 Introduction

Potholing presents a major challenge for all national and state agencies involved in the maintenance of roads and pavements. This has motivated a significant amount of research for the development of higher-quality materials and better techniques to combat road damage and increase road-repair life expectancy.

In 1987, the U.S. Congress established a 5-year applied research program called the Strategic Highway Research Program (SHRP). The SHRP functioned as a unit of the National Research Council, with its goal being to improve the performance, safety, and efficiency of the nation's highway system. Relevant projects of the SHRP are reviewed in this chapter. In addition, this chapter also reviews previous work done in the field of pothole detection and automated road repair vehicles.

2.2 Pothole repairing materials and techniques

The U.S. Army Corps of Engineers Cold Regions Research and Engineering Laboratory (CRREL) in 1981 sponsored the preparation of a manual to assist in the understanding and management of pothole problems in asphalt pavements. The manual by Eaton *et al.* was revised in 1989 [2]. This manual describes the factors that contribute to the increase in pothole occurrence. These include factors such as lack of financing, traffic growth, weather and insufficient drainage facilities. The two mechanisms, fatigue failure and raveling failure that lead to pothole development are described. The use of preventive-

maintenance programs and pavement inventories are recommended to ensure an organized and cost-effective way to preserve, repair, and restore roadway systems.

Smith *et al.* presented the research conducted under the SHRP Project H-105, Innovative Materials and Equipment for Pavement Surface Repairs [3]. This research effort was divided into five categories: asphalt concrete (AC) pothole repair, AC crack repair, Portland cement concrete (PCC) spall repair, PCC joint resealing, and PCC crack sealing. Proprietary bituminous mixes were found to have a life expectancy significantly longer than conventional cold-mixes. It was found that proprietary mixes are more advantageous over conventional mixes in colder conditions as compared with warmer conditions. The use of permanent hot-mixes in dry potholes was found to have the longest life expectancy; however, hot-mixes did not perform satisfactorily when placed in wet potholes.

Preliminary findings of the SHRP project H-106, Innovative Materials Development and Testing, are presented by Evans *et al*. in [8]. Four main areas were investigated, pothole repair in asphalt pavements, crack treatment in asphalt pavements, joint sealing in PCC pavements, and spall repair in PCC pavements. It was found that the throw-and-roll technique was as effective as the semi-permanent procedure when using high-performance cold-mixes. The spray injection method was found to be a viable method for pothole repair in asphalt pavements.

Wilson *et al*. conducted an extensive pothole-repair experiment as part of the SHRP project H-106 [6]. Tests were focused on cold-mix asphalt patching materials, the most commonly used materials for winter- and spring-time pothole repairs. The goal of

this project was to identify the most cost-effective materials and techniques. Twelve-hundred-and-fifty pothole patches were placed at eight test sites across the United States and Canada. These patches were placed using different types of cold-mixes and different installation techniques. Patches placed in the dry-freeze region performed better than those placed in the wet-freeze region. The throw-and-roll technique was found to be as effective as the semi-permanent technique for the same materials. The throw-and-roll and spray injection methods produced the highest-quality repairs in all cases and were found to be the most cost-effective. The choice of the material proved to have a dramatic effect on the life of the patch, and it was recommended using only high-performance cold-mixes.

Based on the SHRP research projects H-105 and H-106, Wilson *et al.* prepared a compendium of good practices for pothole repair [4]. The manual describes the different types of cold-mixes available and the procedures used for pothole repair in asphalt surfaced pavements. Guidelines have been provided to calculate the expected average life and overall cost effectiveness of the various repair materials and techniques.

Following the SHRP research projects, the Federal Highway Administration's (FHWA) Long Term Pavement Performance (LTPP) program conducted five years of additional research on pothole repair. Wilson *et al.* in [5] prepared an update to [4] that further validated the repair materials and techniques described in [4].

Griffith conducted a literature search and survey of nine transportation agencies to determine the kinds of specialized equipment being used to perform pothole repair [9]. The results showed that spray injection patching was a very effective and widely

accepted method of pothole repair. The report also described three types of spray injection patching equipment: trailer-type units, modified-truck units, and self-contained units.

2.3 Pothole detection

Karuppuswamy *et al.* used a non-contact vision approach to detect potholes [11]. In their approach, a histogram of the environment is used to determine a brightness threshold to determine if a pothole is within the field of view. Large white potholes more than 2 feet in diameter were detected.

Matthies *et al.* demonstrated the use of thermal signature for night-time negative obstacle (pothole) detection [12]. Their work is based on the fact that interiors of negative obstacles generally remain warmer than the surrounding terrain throughout the night.

2.4 Automated road repair vehicles

In 1992, the department of civil engineering at Carnegie Mellon University developed a prototype for an automatic crack-filling robot [13]. The robot utilized video imaging to identify areas of potential cracks and range sensing, with an infra-red laser range sensor, was used to confirm the location of the cracks. An onboard air lance was then used to clean the cracks, and a sealant wand was used to fill the cracks. In field trials, the located cracks were filled with an accuracy of less than 1 cm. However, the frame of view was narrow, thus requiring multiple runs over the same area. The robot was also very slow, requiring two minutes to complete a range scan of a captured frame.

The Advanced Highway Maintenance and Construction Technology Research Center (AHMCT) at UC Davis developed an Automated Crack Sealing Machine (ACSM) in 1993 as part of SHRP project 107A [14]. The ACSM is shown in Figure 2.1. According to the report by Velinsky, the machine comprised of two systems, one for longitudinal cracks and joints and the other for random or transverse cracks and joints. A vision system was used for crack detection. Once the crack was located, hot blowing and sealing were performed automatically. The ACSM was dismantled in 1998 due to complexities in running and maintenance.

Figure 2.1. The ACSM developed by the AHMCT at UC Davis
(Source: www.ahmct.ucdavis.edu)

The Basic Industries Research Laboratory (BIRL) of Northwestern University developed an Automated Pavement Repair Vehicle (APRV) in a 28-month research project as part of SHRP project 107B [10]. The final report by Blaha describes the

fabrication and testing of the APRV. The driver located the potholes to be repaired and used a pavement cutter operated by a joystick to cut and shape the holes if required. Next, a vision system scanned the area to be repaired and a telescoping robotic arm used a vacuum nozzle to clear the pothole of water and debris. The robotic arm then used a hot-air lance to heat the surface and bonding edges. This was followed by automatic spray patching of the pothole. The Northwestern University BIRL study did not achieve the anticipated results. "The prototype machine was not effective in field trials. It operated slowly and was costly to use. [9]"

Mara of Sandia National Laboratories patented the design of a Rapid Road Repair Vehicle in June 1998 [15]. According to his design, on-board image processing would be used to distinguish between holes, bumps, and manhole covers or cracks. Next, nozzles would pass over the pothole delivering the filling material. "The mixture would be tamped into place, dusted with grit to provide traction, and vacuumed. Finally, another row of scanners would check the quality of the repair. [15]" He estimated the cost of his vehicle to be between $300,000 and $325,000.

CHAPTER III

IPRV DESIGN

3.1 Introduction

The IPRV was designed in three stages. The first stage involved the design of a semi-autonomous mobile vehicle capable of being maneuvered remotely over a wireless LAN. Once this was achieved, the second stage involved the addition and testing of the pothole-detection module. The final stage involved the addition and testing of the pothole-filling module and overall system integration.

This chapter describes in detail the mechanical design of the IPRV. It also highlights the considerations that motivated the selection of various components of the IPRV. The description is organized according to the stages of development mentioned above.

3.2 Stage I – Semiautonomous mobile vehicle design

The first consideration in the design of the prototype was the determination of its size. Several factors influenced this decision. The IPRV must be robust to be able to operate satisfactorily in field tests. It must be able to carry significant loads (filler material). It should also be large enough to house a laptop computer to be used for control. Keeping these factors in mind, it was decided to use an electric powered wheelchair (EPW) as the building block for the IPRV.

The IPRV is built upon the base frame of an Invacare Ranger II™ electric powered wheelchair. The frame is 66-cm long, 46-cm wide, with a maximum height of

52.5 cm. It is capable of supporting a weight of approximately 100 kg. Figures 3.1 and 3.2 show the front and side views of the IPRV.

Figure 3.1. Front view of the IPRV

As depicted in the figures, motion is provided by two independently driven 24-V DC motors with built-in reduction gears that provide a maximum speed of 6 km/hr. Two 12-V DC deep-cycle marine gel batteries are housed within the entablature and provide 5

to 9 hours of typical operation. An Invacare MKIV RII motor controller is used for motion control and is mounted on the frame. The IPRV has pneumatic front wheels that are 31.75 cm in diameter. Two 18-cm-diameter caster wheels in the rear provide support to the vehicle. The vehicle is front-wheel driven to ensure that the traction wheels negotiate the terrain first.

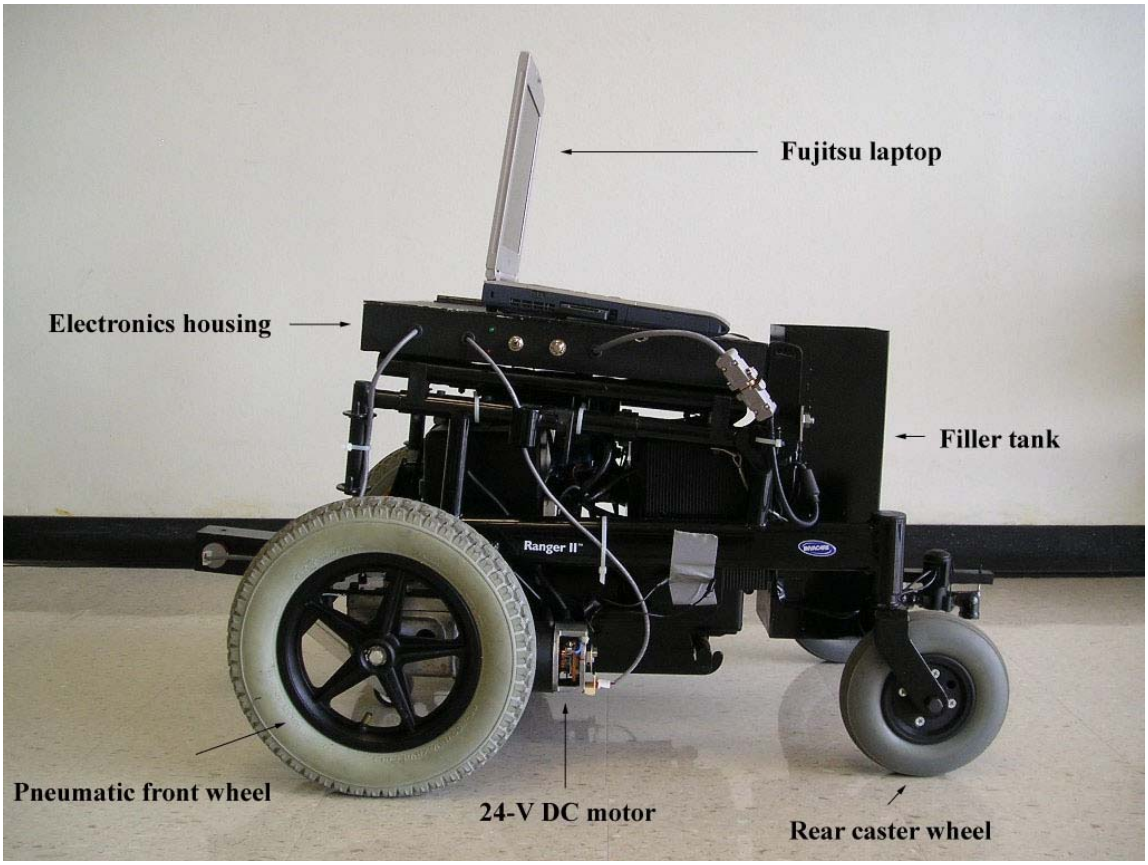

Figure 3.2. Side view of the IPRV

A sheet-metal box is mounted on the upper frame of the IPRV and forms a housing for the electronics. An on/off switch, a 1-kΩ potentiometer and two light

emitting diodes (LEDs) are mounted on one side of the housing. A Fujitsu Lifebook laptop with an AMD-K6™ 451-MHz processor and 192 MB of RAM serves as the controller for the IPRV and is placed on the metal box. The IPRV also has a mechanism to provide feedback for position control and to ensure that it proceeds in a straight path when being operated semi-autonomously. This mechanism is described below.

3.2.1 Position control

Once a pothole is detected during the operation of the IPRV, it must be capable of automatically positioning its filling-tank valve over the pothole. Thus there is a need for a real-time feedback mechanism that can provide closed-loop position control.

Also, the IPRV uses a differential drive system for steering. Turning is achieved by rotating one of the drive motors at a different speed than the other. Even the slightest difference in speeds between the two motors will result in the vehicle not following a straight course. These differences may arise due to dissimilar tire pressures, wear differences in the carbon brushes, bearings, or gears of the two motors. Thus a real-time feedback mechanism is also required to ensure that both motors rotate at the same rate when the IPRV moves in a straight direction.

To achieve closed-loop position control and to ensure an accurate drive system, a Hall-effect switch was mounted on the rear casing of both the motors as suggested in [16]. A disc with magnets attached to half of its periphery was mounted on the rotor shaft. This disc rotates with the rotor. Figure 3.3 shows the feedback assembly being used.

A pulse is generated by the Hall-effect switch on every rotation of the motor shaft and is input to a data acquisition card installed on the laptop. When the IPRV is following a straight path, the distance moved by it is proportional to the number of rotations of the motor shaft. Thus a feedback signal is available to position the IPRV with a resolution equal to half the distance moved by it in one rotation of the motor. This resolution was found to be 1.554 cm. It should be noted that the resolution can be improved by changing the distribution of the magnets. The algorithm used to achieve closed-loop position control is described in Chapter V.

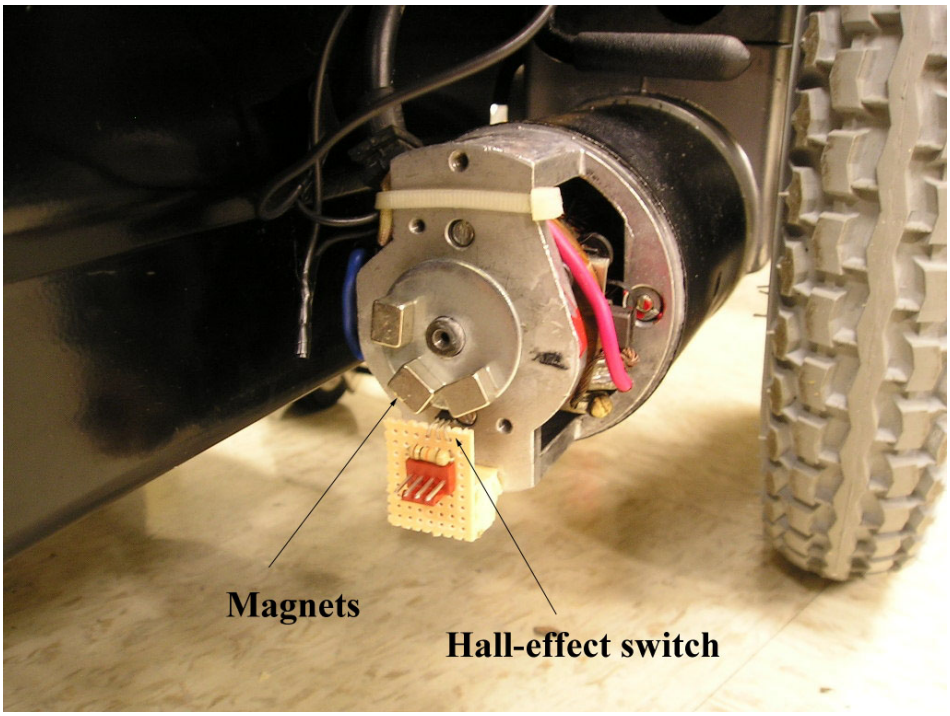

Figure 3.3. IPRV position control feedback mechanism

In order to mount the position-control feedback assembly, it was necessary to remove the brake-assembly installed on the rear casing of the motors. The procedure for its removal is described below.

3.2.2 Brake-assembly removal

As a safety feature, all electric powered wheelchairs are provided with braking mechanisms. In the Invacare wheelchair used for this thesis, the brake mechanism was located on the rear end of each motor. Figure 3.4 shows the original brake-assembly.

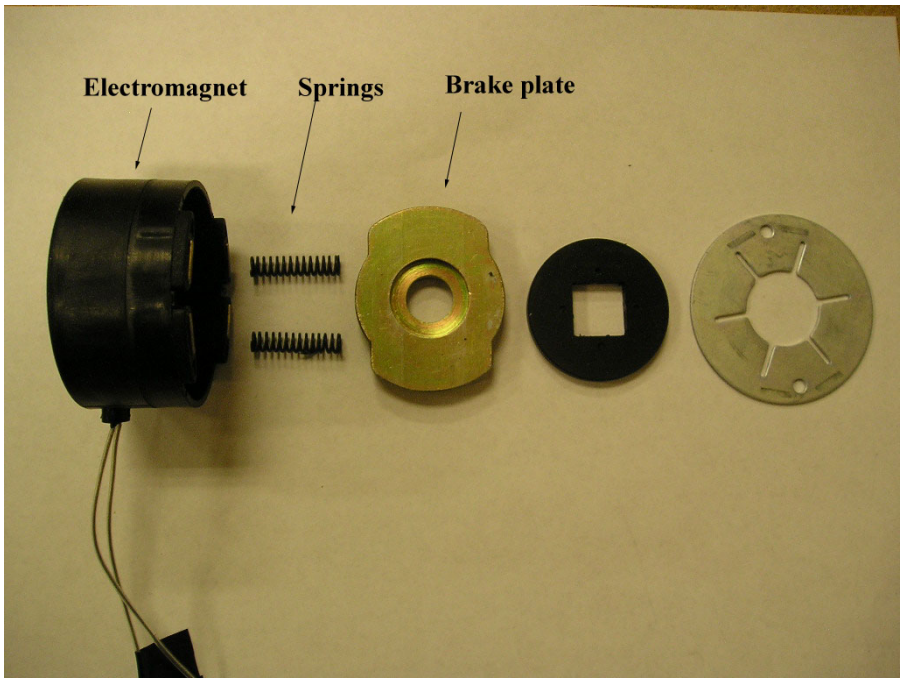

Figure 3.4. Electromagnetic brake-assembly

When the wheelchair power is off or the wheelchair joystick is in the neutral position, two springs force a plate against the motor shaft preventing the wheelchair

from rotating. On displacing the joystick from its neutral position, 24-V DC voltage is applied across the electromagnet to energize it. The plate is retracted by the electromagnet, and the brake is released.

This safety feature was not required for the operation of the IPRV and was removed, making room for mounting the feedback mechanism described in the previous section. However, it was found that the wheelchair motor controller prevented the wheelchair from being operated when the brake-assembly was disconnected. This problem was overcome by placing a resistor across the 24-V DC supply from the motor controller to make the motor controller 'believe' that the brake-assembly was still in place. The resistance and wattage of the resistor to be used was determined as follows.

Voltage across the electromagnet, V_{mag} = 24 V; Resistance of the electromagnet = 70 Ω

=> Power generated at the electromagnet = $\frac{V_{mag}^2}{R}$ = 8.23 W

Based on availability, two 200-Ω, 5-W resistors wired in parallel are being used in the IPRV to emulate each brake-assembly's impedance. This provides an equivalent 100-Ω, 10-W resistor that satisfies the power requirements calculated above.

3.3 Stage II – Pothole-detection module design

The IPRV employs a mechanical means for pothole detection as shown in Figure 3.5. The pothole detection module consists of a swinging link in the front end of the IPRV. One end of this link has a wheel attached to it that rolls along the road surface. The other end is fixed to an incremental optical shaft encoder. The encoder generates 256 pulses per revolution thus providing a resolution of 1.4°.

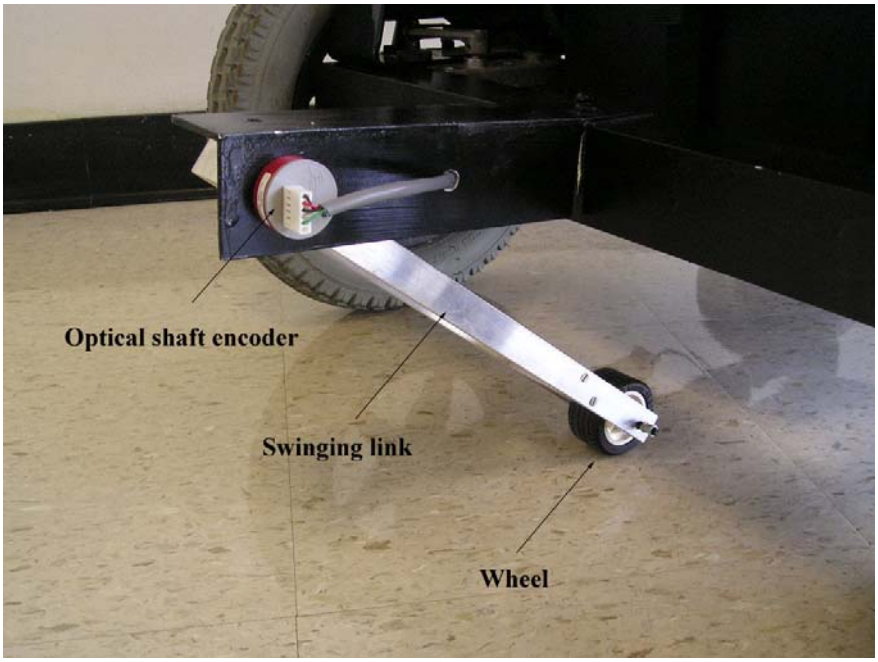

Figure 3.5. Pothole-detection module

During operation, the wheel follows the road surface contour. On encountering a pothole the swinging link rotates depending on the depth of the pothole. This rotation causes pulses to be generated by the encoder which are input to a data acquisition card installed on the laptop computer. The pulse count is used to determine the position of the deepest encountered part of the pothole. The IPRV is then positioned over the calculated location, and the filling operation begins.

The encoder uses quadrature encoding, which makes it possible to determine the direction of rotation. The direction-of-rotation information enables the IPRV to distinguish between potholes and bumps, and it ignores the bumps. A detailed description of the algorithm used to determine the direction of rotation of the encoder is provided in Chapter V.

3.4 Stage III – Pothole-filling module design

A filler tank is mounted on the rear end of the IPRV. This tank contains water that is used for simulation purposes, in place of filler material, to fill the potholes. A 12-V DC solenoid valve is mounted on the filler tank outlet pipe and is used to start and stop the pothole filling operation. Figure 3.6 shows the pothole-filling module being used.

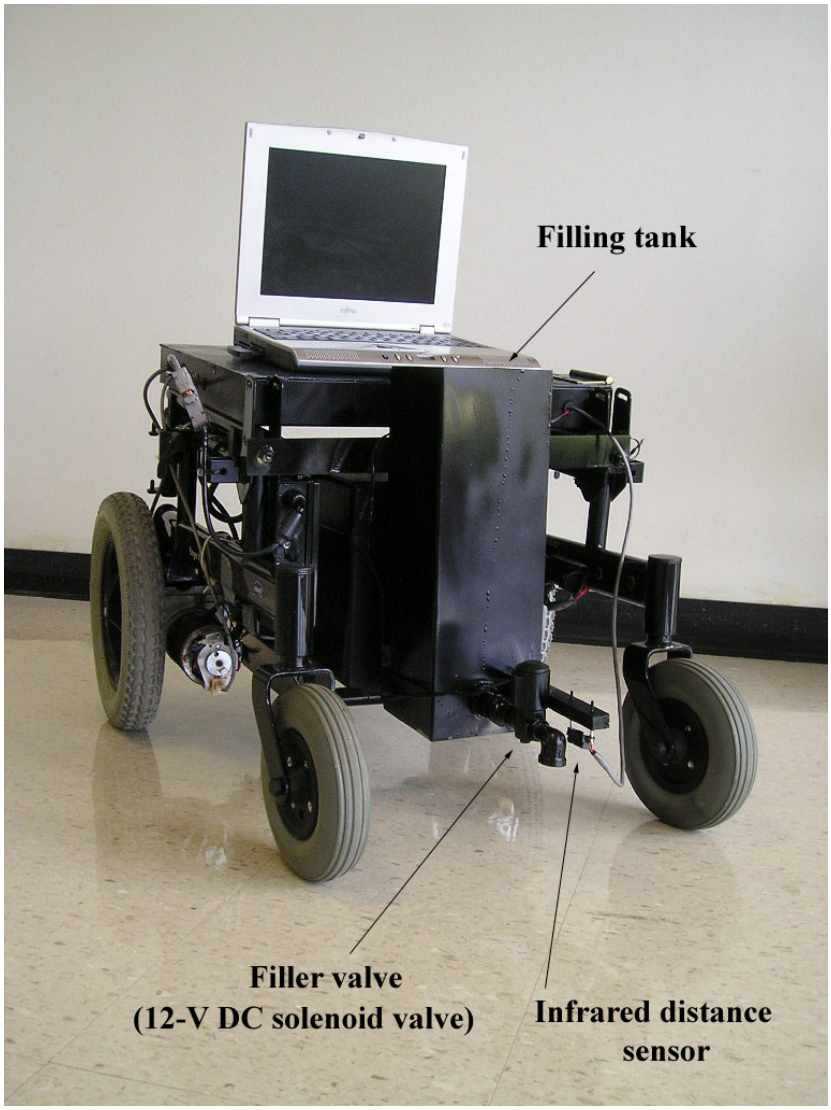

Figure 3.6. Pothole-filling module

Once a filling operation is in progress, additional information must be available to determine when to close the filler valve. This information may be in the form of the volume of the pothole or some other feedback mechanism. By their very nature, potholes are highly irregular in shape and size making volumetric analysis very complex. The IPRV uses an infrared distance sensor mounted on the filler tank to provide feedback indicating the end of a filling operation. The infrared sensor is positioned such that it undergoes a transition in state when the water level in the pothole reaches the road surface level. This state transition triggers the closing of the filler valve.

CHAPTER IV

DATA ACQUISITION AND INTERFACING

4.1 Introduction

In order to operate and control the IPRV, an interface was developed between the laptop and the individual components of the IPRV described in the last chapter. This chapter describes in detail the developed interface. Section 4.2 describes the PCMDIO data-acquisition card used for all input/output (I/O) functions, Section 4.3 describes the laptop/motor controller interface developed followed by a description of the filler valve interface in Section 4.4. The sensor interface is discussed in Section 4.5 and the final section describes the prototyping board used and the circuit designed to implement the interfaces.

4.2 The PCMDIO data-acquisition card

A Superlogics PCMDIO 24-channel digital I/O type II Personal Computer Memory Card International Association (PCMCIA) card is installed on the Fujitsu laptop and is used to perform all data acquisition and control functions. A CP-1037 adapter cable is used to convert the PCMDIO's 33-pin 0.8-mm I/O connector to an industry standard D-37 connector. Figure 4.1 shows the PCMDIO card with the CP-1037 adapter cable. The main features of the PCMDIO card are listed below.

1. The PCMDIO has 24 transistor-transistor-logic (TTL)-compatible buffered digital-I/O channels individually programmable as either input or output. These digital-I/O channels are grouped into three different ports with each port containing eight channels.

These ports are controlled via the Data Port A, Data Port B, and Data Port C control registers, respectively. In all three registers, each bit corresponds to one data line. A detailed description of the registers is provided in [17].

2. The eight Port C I/O channels may also be configured as interrupt sources. The interrupts may be configured in four ways: level-sensitive active-low interrupt, level-sensitive active-high interrupt, high-to-low transition-edge-sensitive interrupt, and low-to-high transition-edge-sensitive interrupt [17].

Figure 4.1. The PCMDIO card with the CP-1037 adapter cable

Individual channels of the PCMDIO are configured using the PCMDRIVE configuration utility described in Chapter V. The PCMDIO performs the following functions.

1. interfacing the laptop and the motor controller for IPRV speed and direction control
2. interfacing the laptop and the filler valve to start or stop the pothole filling operation
3. interfacing the laptop with all onboard sensors

These functions are described in the following sections.

4.3 Interfacing the motor controller

The IPRV has an onboard Invacare MKIV RII motor controller with a joystick for motion control. Power to the motor controller is supplied by two 12-V DC batteries. The motor controller uses pulse width modulation (PWM) to drive the two 24-V DC motors. The schematic wiring diagram of the Invacare wheelchair is shown in Figure 4.2.

To enable semiautonomous operation of the IPRV, an interface was required between the laptop and the DC motors. Two choices were available for the construction of this interface.

1. Develop an interface between the laptop and the motor controller. This interface would emulate the functioning of the joystick and would be used to provide speed and direction commands to the motor controller. The motor controller would drive the motors.
2. Develop a circuit that would use PWM signals generated by the laptop to drive the motors directly.

Option 1 was chosen as the preferred method as it allowed the use of the motor controller specifically designed to drive the DC motors. It also reduced the computational demands on the laptop's processor. To emulate the joystick, it was

necessary to understand the operation of the joystick and how it interfaced with the motor controller. This is described below.

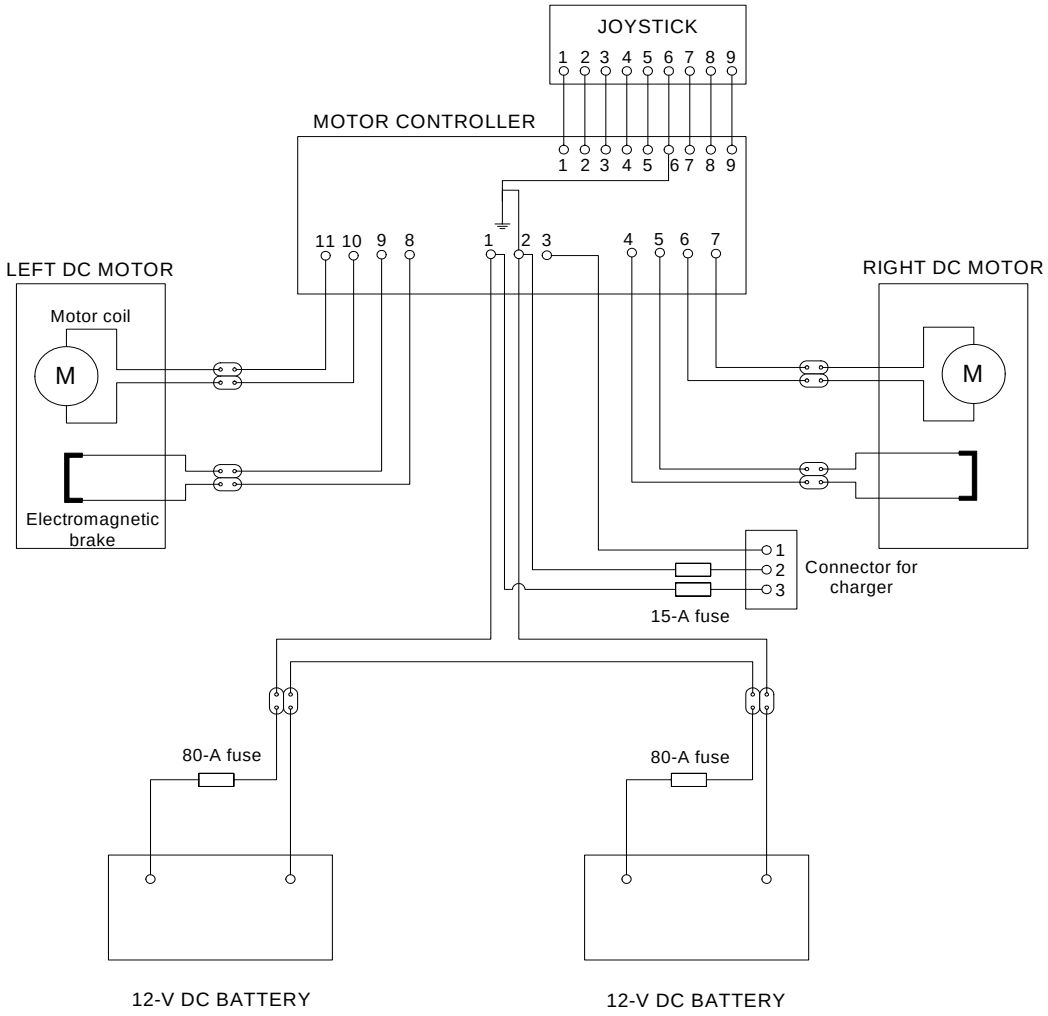

Figure 4.2. Invacare wheelchair schematic wiring diagram

4.3.1 Joystick operation and interface

The Invacare joystick is a two-axis, spring-return analog resistive joystick. It has two potentiometers mounted to spring-loaded bails that are moved on deflecting the joystick

handle. One potentiometer is used for the *x*-axis (Left-Right direction control) and the other for the *y*-axis (Forward-Reverse speed control). The voltage across each potentiometer is measured by the motor controller and is used to determine the required speed and direction. Figure 4.3 shows the joystick/motor controller interface circuit.

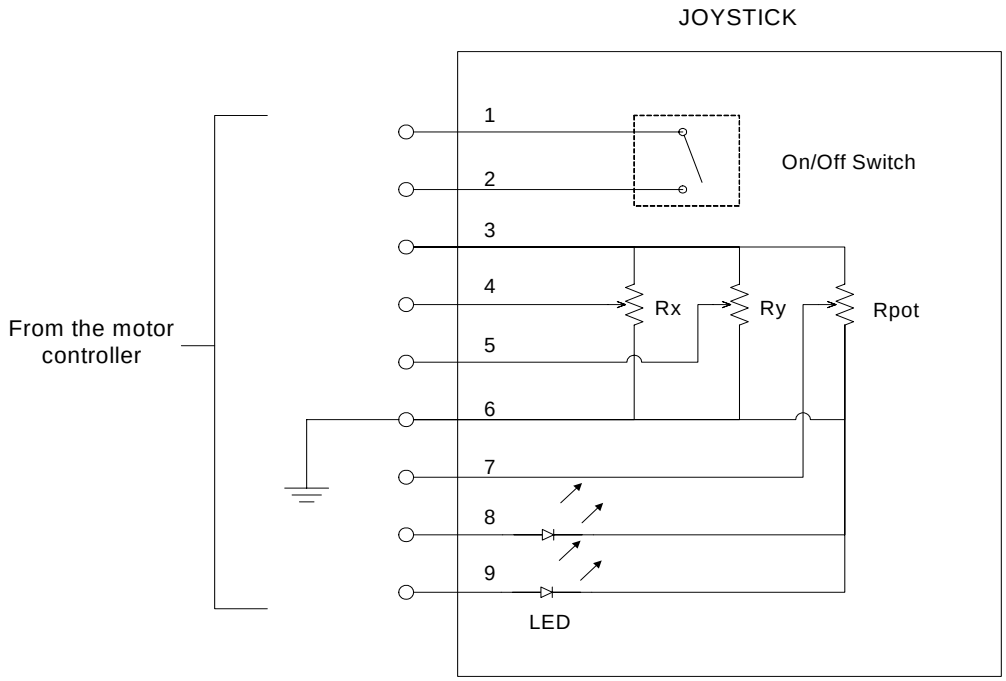

Figure 4.3. Joystick/motor controller interface

In the figure above, *Rx* and *Ry* are the two potentiometers of the joystick. These provide 2 analog input channels to the motor controller as described above. The joystick/motor controller interface also has a potentiometer, *Rpot*, that provides a third analog input channel to the motor controller. The input from this channel is used to set

the maximum speed of the wheelchair. Once the maximum speed is set, the joystick speed control channel is used to vary the speed of the wheelchair within the set bounds.

The interface also has the following components.

1. an on/off switch to power the wheelchair
2. an LED to indicate power to the wheelchair is on. The LED also flashes as a visual indication of low battery levels.
3. an LED to indicate an error detected by the motor controller

Voltages across the two analog input channels from the joystick were measured to determine the range of voltage change when the joystick handle was moved from one extremity to the other. The results are shown in Tables 4.1 and 4.2.

Table 4.1. Analog voltage range across Channel 1

Voltage across Channel 1	
Joystick handle position	Voltage across the channel (V)
Neutral	2.487
Maximum Forward	3.865
Maximum Reverse	0.995
Maximum Left	2.510
Maximum Right	2.477

Table 4.2. Analog voltage range across Channel 2

Voltage across Channel 2	
Joystick handle position	Voltage across the channel (V)
Neutral	2.489
Maximum Forward	2.490
Maximum Reverse	2.510
Maximum Left	0.800
Maximum Right	4.195

From the tables above it can be seen that analog input Channel 1 determines the speed of the wheelchair while Channel 2 determines the direction. An interface was thus developed to emulate the joystick and enable the laptop to control the speed and direction of the IPRV. This is described below.

4.3.2 Laptop/motor controller interface

Figure 4.4 shows the designed circuit that provides an interface between the laptop and the motor controller and replaces the joystick.

Channel 0 of the PCMDIO card is configured as an output channel. This channel provides an 8-bit output that is converted to an analog signal (0–5-V range) by the Quad 8-bit digital-to-analog converter (DAC). Channel 1 of the PCMDIO card is configured as a 2-bit output channel that determines which channel of the DAC is being set. In this way both speed and direction of the IPRV can be controlled by setting the appropriate

bits of Channels 0 and 1 of the PCMDIO. The description of individual components follows.

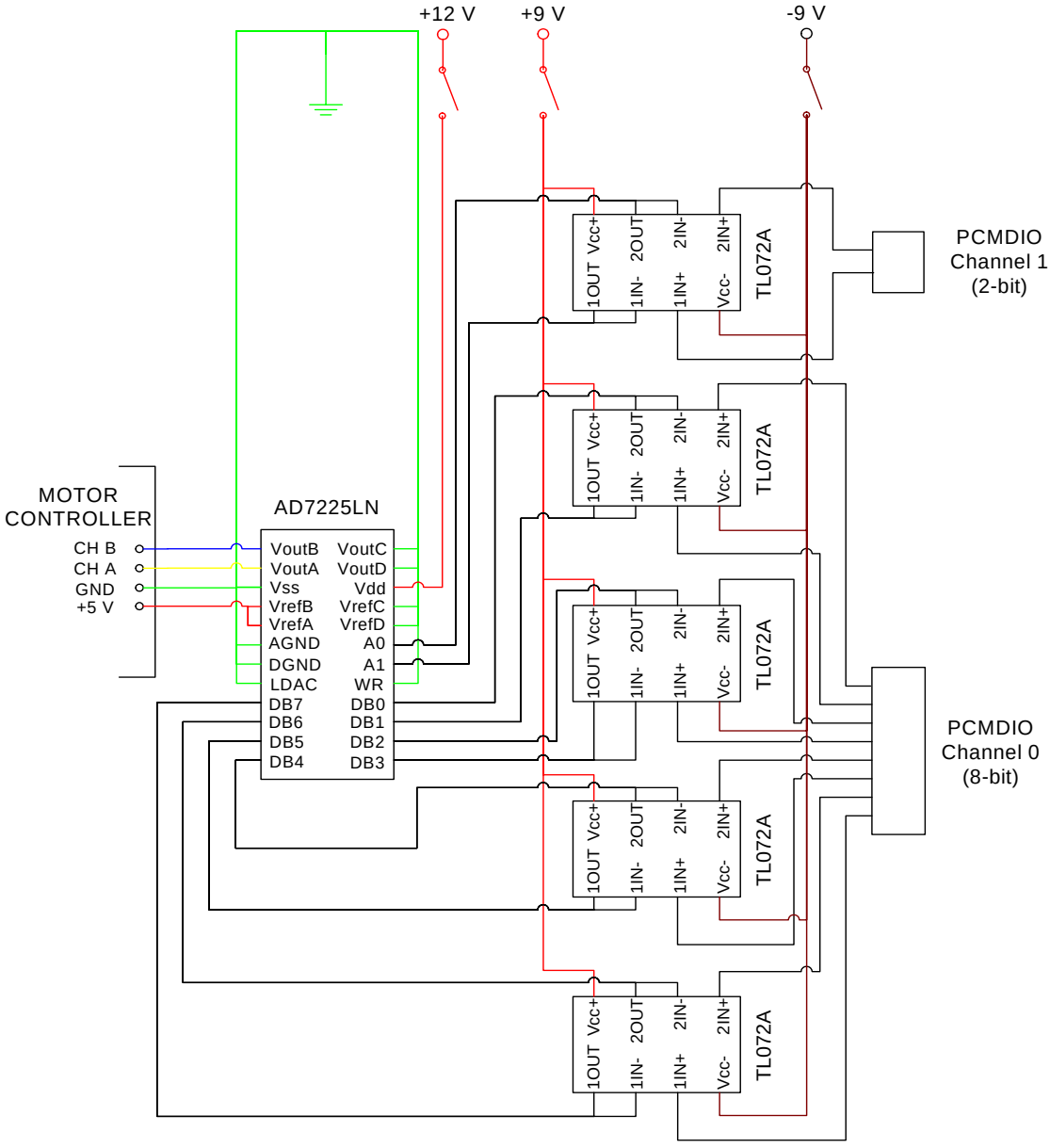

Figure 4.4 Circuit to interface the laptop with the motor controller

1) *TL072A operational amplifier (op-amp)* - Each TL072A comprises of two op-amps. Here, the op-amps are used as buffers or voltage followers as shown in Figure 4.5.

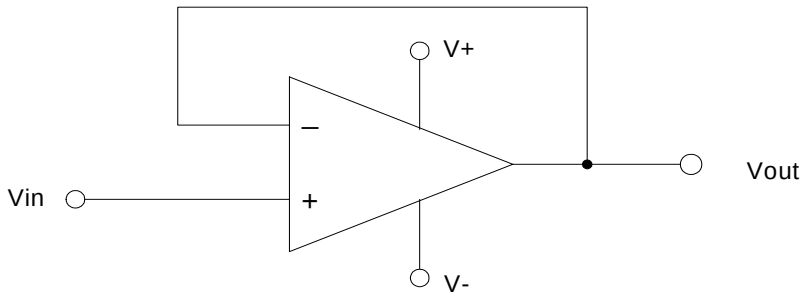

Figure 4.5 Voltage follower or buffer

The non-inverting (+) input is connected to the voltage source (*Vin* = 5 V) and the output (*Vout*) is connected to the inverting (–) input. Thus the gain is unity and *Vout* = *Vin*. The voltage follower is being used to isolate the PCMDIO from the output of the op-amp that is input to the DAC. Hence very little power is drawn from the PCMDIO avoiding 'loading' effects.

2) *AD7225LN Quad 8-bit DAC* - The AD7225LN has four 8-bit DACs with output amplifiers. Each of the 4 channels has two registers: an input register and a DAC register. Data held in the DAC registers determine the analog outputs of the converters. Only two channels of the DAC are being used, one for speed control and the other for direction control of the IPRV as described previously. Pins WR and LDAC are grounded rendering all the DAC registers and the selected input register transparent. This enables

the output to follow the input data for the selected channels. The latching capability of the DACs is not being used as the output of each channel of the PCMDIO is latched.

3) *Voltage supplies* - The laptop/motor controller interface circuit has three different voltage supplies. +5-V DC is supplied by the motor controller and is the input reference voltage to the DAC. +12-V DC is supplied from one of the wheelchair batteries and forms the supply voltage (*Vdd*) to the DAC. ±9-V DC is supplied by a set of four 9-V DC batteries as shown in Figure 4.6. This forms the supply voltage to the op-amps.

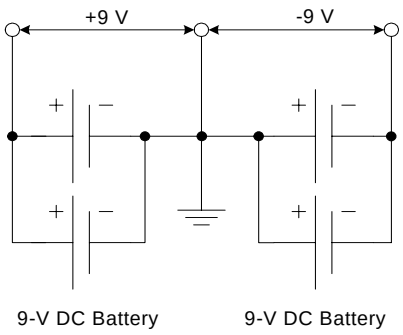

Figure 4.6. ±9-V DC supply

It was initially considered to supply the op-amps with ±12-V DC which could be made available using the two 12-V DC wheelchair batteries. This would have required the connection between the two batteries to serve as ground (zero potential). However, it was found that the motor controller ground was the same as the negative terminal of Battery 2 shown in Figure 4.1. Thus the two grounds would have had a potential difference of 12 V and hence an independent voltage supply had to be provided to operate the op-amps.

In addition to the components described above, 2 LEDs and a 1-KΩ potentiometer are mounted on one side of the electronics housing and perform the same functions as the joystick LEDs and potentiometer as described in Section 4.3.1.

4.4 Interfacing the filler valve

The IPRV filler valve is a Delrin direct-acting 12-V DC solenoid valve. The valve is normally-closed (NC) and does not require a minimum differential pressure to operate. When energized, the valve is open and draws a continuous current of 1.22 A. Figure 4.7 shows the circuit designed to operate the filler valve using the laptop computer.

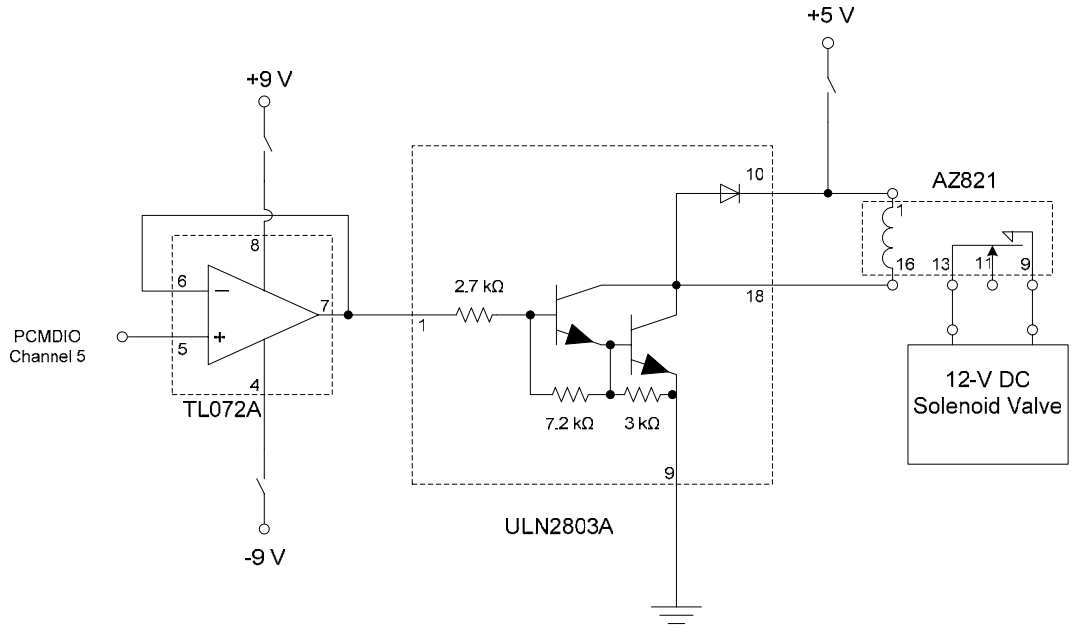

Figure 4.7. Filler valve interface

Output Channel 5 (1-bit) of the PCMDIO is used to open and close the filler valve. The filler valve is connected across the normally-open (NO) contacts of an AZ821

double-pole double-throw (DPDT) relay. One end of the relay coil is connected to a 5-V DC supply while the other end is connected to the open-collector (OC) output of a ULN2803 eight Darlington-transistor array. A high output from the PCMDIO Channel 5 drives the ULN2803 thus energizing the relay coil and opening the filler valve. The ULN2803 has a built-in flyback diode to be connected across the relay coil and prevents damage due to inductive kickback from the coil. A TL072A op-amp is configured as a voltage follower and is used to isolate the PCMDIO from the rest of the circuit.

4.5 Interfacing the sensors

An MC7805C three-terminal positive voltage regulator is used to convert 12-V DC from a wheelchair battery into a 5-V DC supply to all the sensors onboard the IPRV. These sensors include the following:

1. incremental optical shaft encoder
2. two Hall-effect switches
3. infrared distance sensor

Outputs of the sensors are connected to the input channels of the PCMDIO. The designed interface is shown in Figure 4.8. As can be seen from the figure, bypass capacitors are applied across the voltage regulator's input and output terminals to suppress any voltage transients and obtain a stable, fixed output. Each supply has a switch so that individual supplies may be switched on and off according to requirement without affecting the rest of the circuit.

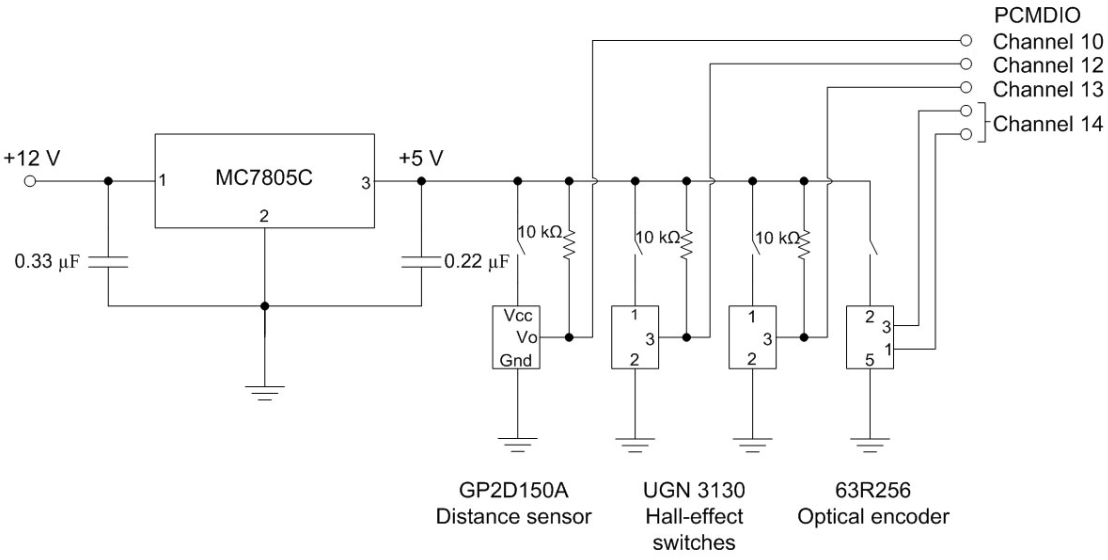

Figure 4.8. Sensor interface

4.6 Prototyping board and circuit design

An 8016 Vectorbord® Circbord® was used to design the circuit that implements all the interfaces described in this chapter. Figure 4.9 shows this circuit.

One side of the Circbord is used for all the outputs from the PCMDIO while the other side is used of all the sensor inputs. None of the sensors are permanently fixed to the prototyping board, but connectors are provided for all the interfaces facilitating removal or exchange of any circuit component. All power supplies have inline switches; this makes it possible to isolate any part of the circuit and is useful for fault detection.

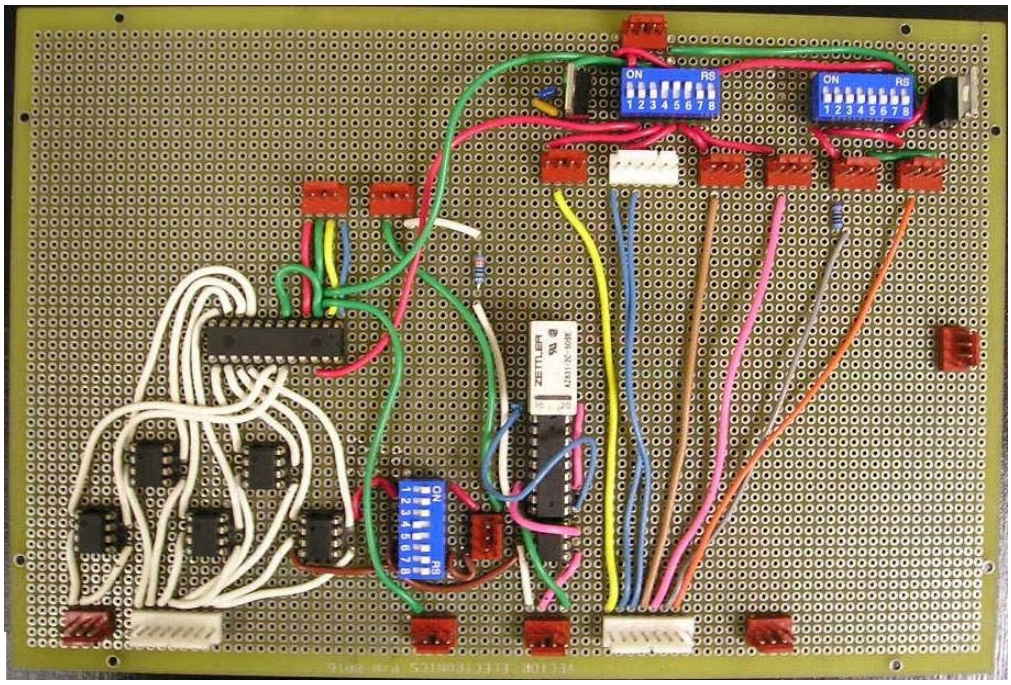

Figure 4.9 Circbord® with interface design

CHAPTER V

SOFTWARE DESIGN

5.1 Introduction

Programming the IPRV required the writing of diverse applications including hardware control, networking, and providing a user with a friendly graphical user interface (GUI). This chapter describes the various issues that arose during the programming phase along with their solutions. The first section describes the programming language being used, the next section describes the software designed to control the hardware, followed by a description of the software designed for networking. The final section describes the GUI.

5.2 Programming language

The Microsoft® Windows® Visual Basic® 6.0 – for 32-bit Windows Development (VB6) environment is being used for all the programming requirements of the IPRV. Visual Basic is an 'event-driven' programming language wherein the program is divided into procedures and functions executed in response to a stream of events. Visual Basic provides a powerful and flexible environment, enabling rapid Windows application development. It provided a single platform to write programs for all the diverse applications of the IPRV.

The Microsoft Windows application programming interface (API) was utilized to develop the application to control the PCMDIO digital-I/O card. The use of the Windows API provides direct access to the dynamic-link-library (DLL) files that contain

the functions, data structures, and constants required to operate the PCMDIO card. Further detail is provided in the next section.

Two programs were written to operate the IPRV. One is executed from the laptop and is called the 'server-side' program. The other can be executed on any computer on the same LAN as the IPRV and is called the 'client-side' program. Actual source code for the server- and client-side programs are available in Appendices A and B, respectively.

5.3 Hardware control

As described in the previous chapter, the PCMDIO digital-I/O card is used for all data acquisition and control functions within the IPRV. Along with the PCMDIO card, the vendor also provides the PCMDRIVE® Data Acquisition Software. The PCMDRIVE software includes the following components.

1. PCMDRIVE Configuration Utility - This utility is used to edit the PCMDIO hardware configuration file, pcmdio.dat. This file contains the setup of the 24 individual I/O channels of the PCMDIO card into logical channels. Using the configuration utility, the logical channels can be set as single-bit channels or multiple-bit channels by clicking on the current logical channel number [18]. Once all the logical channels have been assigned, each channel may be configured as either an input channel or an output channel. The PCMDRIVE configuration utility with the first 16 data lines is shown in Figure 5.1. For the IPRV, the PCMDIO was configured to have 14 logical channels. Table 5.1 lists the detailed channel configuration being used.

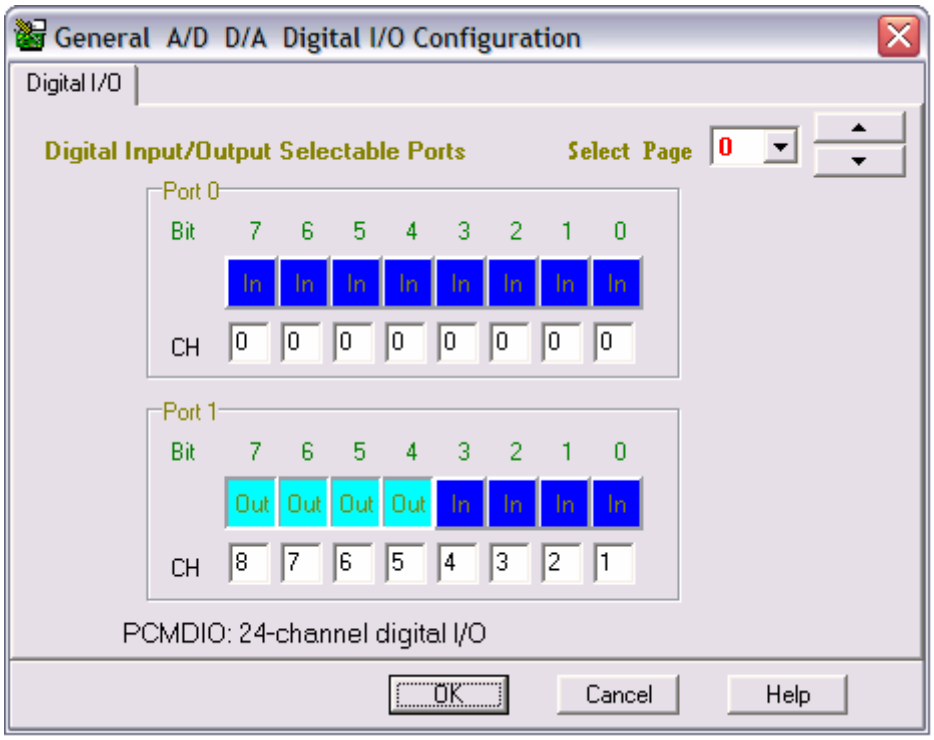

Figure 5.1. PCMDRIVE configuration utility

Table 5.1. IPRV PCMDIO channel configuration

Logical Channel (CH)	Number of bits/channel	Channel type (Input/Output)	Function
CH 0	8	Output	Data bits to the DAC
CH 1	2	Output	Logic bits to the DAC
CH 2	1	Output	Spare
CH 3	1	Output	Spare
CH 4	1	Output	To an LED for ‘Hole found’ indication
CH 5	1	Output	To open/close the filler valve
CH 6	1	Output	Spare
CH 7	1	Input	Spare
CH 8	1	Input	Spare
CH 9	1	Input	Spare
CH 10	1	Input	From the distance sensor
CH 11	1	Input	Spare
CH 12	1	Input	From the left-motor Hall-effect switch
CH 13	1	Input	From the right-motor Hall-effect switch
CH 14	2	Input	From the pothole-detection optical encoder

2. Device driver – The PCMDRIVE software provides a two-part driver. The first part contains the Window API and is also responsible for memory management, file I/O, and other hardware-independent functions. The second part of the driver is hardware-dependent and is responsible for implementing the requested operations on the PCMDIO.

The Windows API provides an interface among the pcmdio.dat file, the device driver, and the server-side program. The procedure to be used to perform data acquisition is described in [18]. Highlights of the procedure follow in the next section.

5.3.1 Performing data acquisition

"PCMDRIVE uses a 'data defined' rather than a 'function defined' interface. [18]" Each data-acquisition operation is defined by a series of configuration parameters. These parameters are contained in a data structure and are collectively referred to as a *request* or a *request structure*. In order to perform an input or output operation using the PCMDIO, the following sequence of steps is required [18].

1. Define the hardware configuration. – Explained in Section 5.3.

2. Open the hardware device. – Before an application program can use the PCMDIO, it must first 'open' it. If this is successful, PCMDRIVE assigns a logical device number used for all future references to the PCMDIO.

3. Allocate the request structure and data buffers. – Once the PCMDIO is open, memory is allocated and locked to be used to store the request structure and data buffers.

4. Define the request structure and data buffers. – The request structure contains information such as the channel number, trigger source, and a pointer to the data buffer. The data buffer contains the status of the buffer, the buffer size, and a pointer to the data storage area.

5. Request the operation. – This step is used to validate the contents of the request structure and to determine if the operation is supported by the hardware. If the request is valid and the operation supported, a *request handle* is issued to identify the configuration. Once the request handle is issued, the channel(s) specified in the request structure is allocated for use by this request.

6. Write data to the locked data buffer. – This step is used only if requesting an output and copies data from the Visual Basic array created in Step 4 to the locked PCMDRIVE data buffer created in Step 3.

7. Arm the request. – By arming the request, the hardware is programmed and any system resources required for the request are allocated and assigned to the request.

8. Trigger the request. – Triggering the request starts the requested operation

9. Wait for completion. – Once the operation has been started, the application waits for the request to be completed. Completion is indicated by the triggering of an event on the change of the request status to ‘complete.’

10. Read data from the locked data buffer. – This step is used only if requesting an input and copies data from the locked PCMDRIVE data buffer created in Step 3 to the Visual Basic array created in Step 4.

11. Release the configuration. – After the operation is completed, the channel(s) used by the request is freed.

12. Close the hardware device. – Once all required operations have been performed, the PCMDIO is closed. “System integrity can not be guaranteed if the application program exits without closing the hardware device. [18]”

For this thesis, 5 functions were specially created in order to simplify the use of the PCMDIO for digital I/O operations. The functions are briefly described below, and the source code for these functions is available in Appendix A

1. Function openDevice – used to define the hardware configuration and open the PCMDIO

2. Function singleDigitalInput – used to input a single value from a single input channel of the PCMDIO

3. Function multipleDigitalInput – used to input a single value from multiple input channels of the PCMDIO

4. Function singleDigitalOutput – used to output a single value to a single output channel of the PCMDIO

5. Function multipleDigitalOutput – used to output a single value to multiple output channels of the PCMDIO

The latter four functions perform Steps 3 through 11 of the above-mentioned steps. Since they do not close the hardware, these functions can be used a multiple number of times by the application program. The algorithm for the function singleDigitalInput is shown in Figure 5.2.

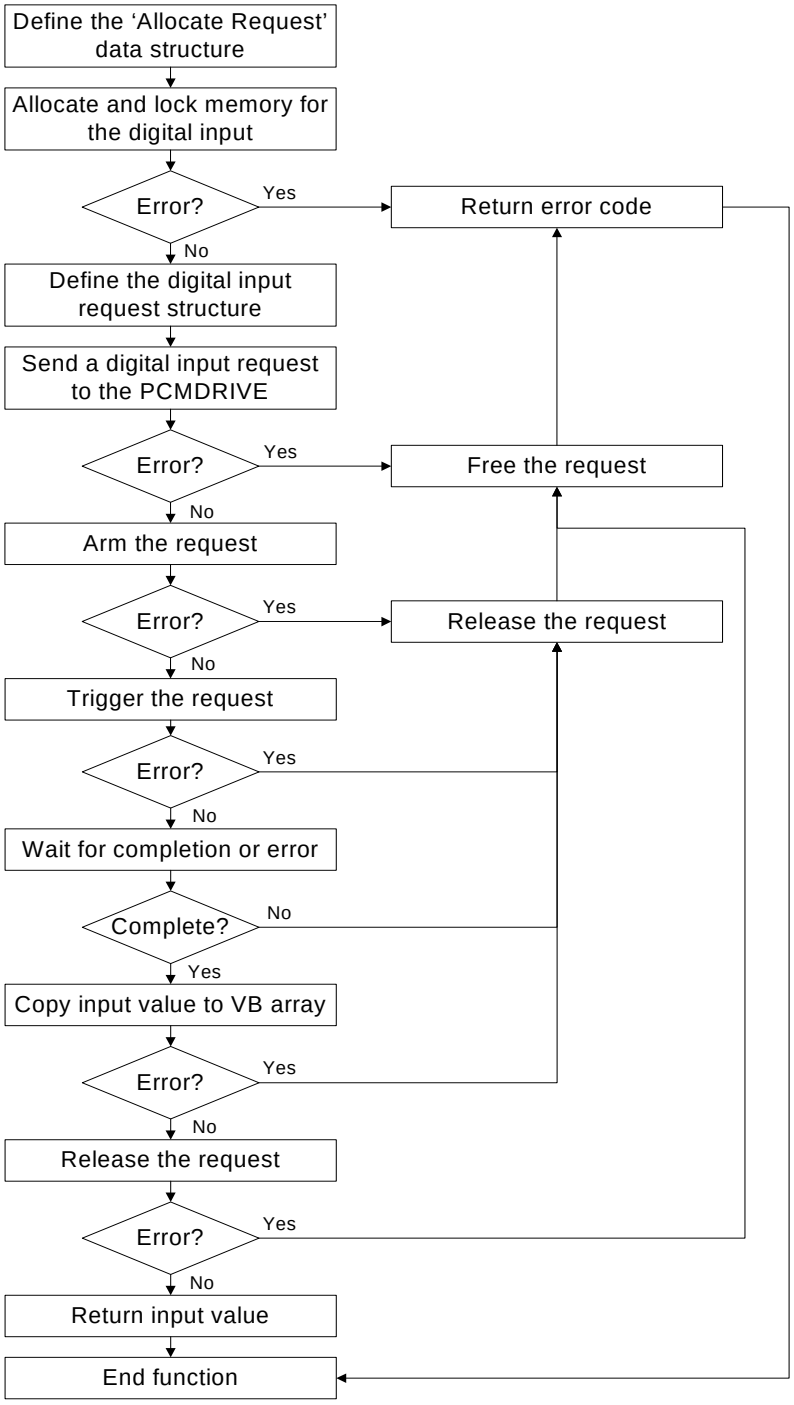

Figure 5.2. Algorithm of function singleDigitalInput

The IPRV application programs use the PCMDIO data acquisition functions described above to control the onboard hardware. Sections 5.3.2, 5.3.3, and 5.3.4 describe the key elements involved in the software design for controlling various aspects of the hardware.

5.3.2 Motion control

The IPRV can be operated in two distinct modes.

1) *Remote Maneuvering Mode* – In this mode a remote user can give direction and speed commands to be executed by the IPRV. The motion is open loop as no feedback mechanism is provided. Operation in this mode is similar to joystick control wherein the vehicle needs to be continuously controlled by the operator.

The IPRV can only be started in the remote maneuvering mode. When started, the voltage across the two analog channels to the motor controller must be 2.5 V indicating that the joystick is in the neutral position (Refer to Sections 4.3.1 and 4.3.2). This corresponds to an output value of 128 of the 8-bit PCMDIO channel 0.

The operator issues direction/speed commands using the client-side GUI (Section 5.5). These commands are sent as messages to the server-side program. Five motion-control messages may be sent: forward, reverse, left, right, and stop. On receiving a message, the server-side program first determines if the command requested is within the operational range of the IPRV (Refer to Tables 4.1 and 4.2). If out of range, the message is ignored; if the message is within the range, the program updates the speed/direction variable by incrementing/decrementing it by a fixed value. The updated variable

determines the new output across Channel 0. The complete algorithm for the remote maneuvering mode is provided in Chapter VI.

2) *Semiautonomous Mode* – In this mode, the IPRV follows a straight path looking for potholes. The remote user can start or stop the IPRV but has no control over its speed and direction. As described in Section 3.2.1, feedback from two Hall-effect switches is available and used to ensure that the IPRV moves in a straight direction. The algorithm for direction control is shown in Figure 5.3.

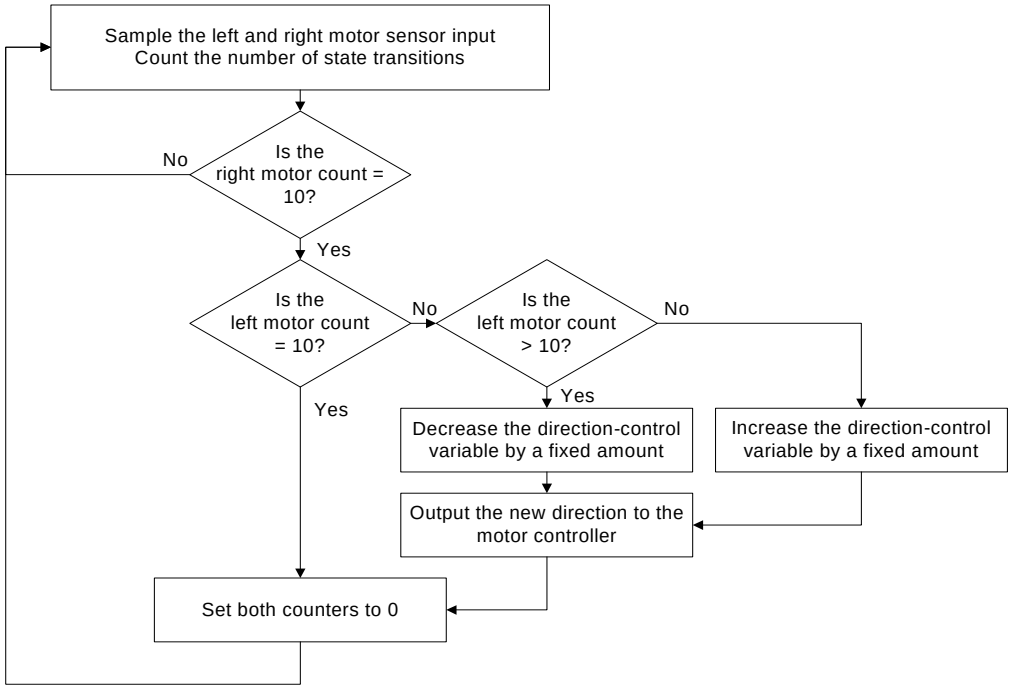

Figure 5.3. Algorithm used for direction control

Both the Hall-effect switches are continuously sampled at a sampling frequency of approximately 400 Hz. Changes of state of the inputs are counted, thus every rotation

of the motor shaft increments the count by 2. For the IPRV to move in a straight direction, both motors must rotate at the same rate. Hence, keeping the right-motor count as a reference, both counts are compared at regular intervals (every 10 right-motor counts) and the direction of the IPRV is changed by a fixed amount to compensate for the difference if any.

On switching the IPRV to the semiautonomous mode, the IPRV moves at a constant speed till a pothole is detected. Once a pothole is detected, it calculates the distance between the deepest encountered part of the pothole and its filler valve. The expression to calculate the distance is derived in Chapter VI. To position the filler valve over the calculated location, a real-time position control algorithm was implemented using feedback from the right-motor Hall-effect switch.

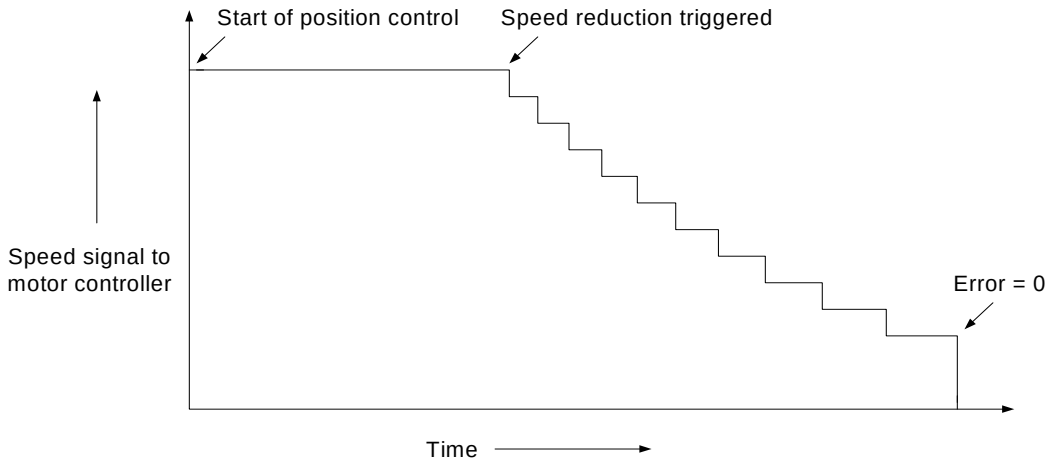

Figure 5.4 Position control speed signal versus time

The distance to be moved is proportional to the number of rotations of the motor shaft when the IPRV is moving in a straight direction. Thus the number of counts to be

moved is calculated and forms the set point for the position control. Figure 5.4 shows the speed signals generated by the position control algorithm. Initially the IPRV is moved at a constant speed till the error (difference between the set point and the current count) reaches a certain predetermined value. At this point the position control algorithm starts to reduce the speed signal proportional to the reduction in error such that the IPRV is brought to a halt when the error equals 0. Due to friction, the IPRV comes to rest before the speed signal is 0. Thus the reduction in speed signal is calibrated to ensure that the IPRV does not stop prematurely.

5.3.3 Pothole detection

The IPRV uses an incremental optical encoder in conjunction with a swinging link and roller to detect potholes as explained in Section 3.3. Rotation of the swinging link causes a series of pulses to be generated across the two channels of the encoder. These pulses will also be generated if the swinging link encounters a bump on the road. Thus an algorithm has been implemented to distinguish between potholes and bumps.

Figure 5.5 shows the pulses generated across the 2 channels of the encoder. Channel A leads Channel B by 90° in all rotations for clockwise rotation of the encoder shaft. Hence for clockwise rotation, the 2-bit input channel to the PCMDIO exhibits the state-transition pattern of 0-1-3-2 for every pulse cycle. Similarly for counter-clockwise rotation the state-transition pattern is 0-2-3-1 for every pulse cycle.

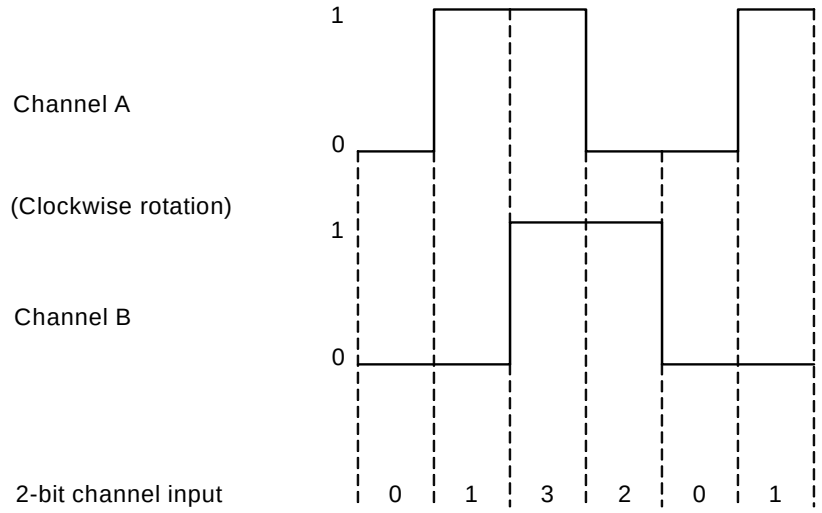

Figure 5.5. Encoder state-transition diagram when rotation is clockwise

An algorithm was implemented to distinguish between potholes and bumps by following this transition pattern as well as to calculate the change in the angle of the swinging link. The algorithm is shown in Figure 5.6. An additional variable is used by the server-side program in order to determine the deepest encountered part of the pothole. Due to its irregular nature, a single pothole may have several undulations each having a local maximum depth. The additional variable continuously monitors the swinging link angle and disregards the local maximum values that are below its value. The knowledge of the maximum depth also enables the IPRV to avoid initiating a pothole filling operation every time minor surface asperities are encountered. A minimum threshold value for the maximum depth is set below which all irregularities are ignored.

Figure 5.6 Algorithm used to distinguish between potholes and bumps

5.4 Networking

Remote operability of the IPRV is provided by interfacing the IPRV with a LAN using a wireless LAN card installed on the laptop. This is shown in Figure 5.7. The IPRV acts as

a server and executes the server-side program. Any remote terminal on the same LAN can be used to remotely operate the IPRV. The remote terminal forms the client and executes the client-side program.

The transport layer protocol used for sending and receiving data is the Transmission Control Protocol (TCP). The IPRV is operated remotely, so it is of utmost importance that when a command to 'stop' the vehicle is sent across the network, it must arrive at the IPRV. There is thus a need for a transport protocol that provides applications with a 'connection-oriented, reliable' service. TCP satisfies these requirements with a connection-oriented, reliable, in-sequence, byte-stream service [19]. The Microsoft Winsock Control 6.0 ActiveX control is used for the implementation of the TCP sockets within Visual Basic 6.0.

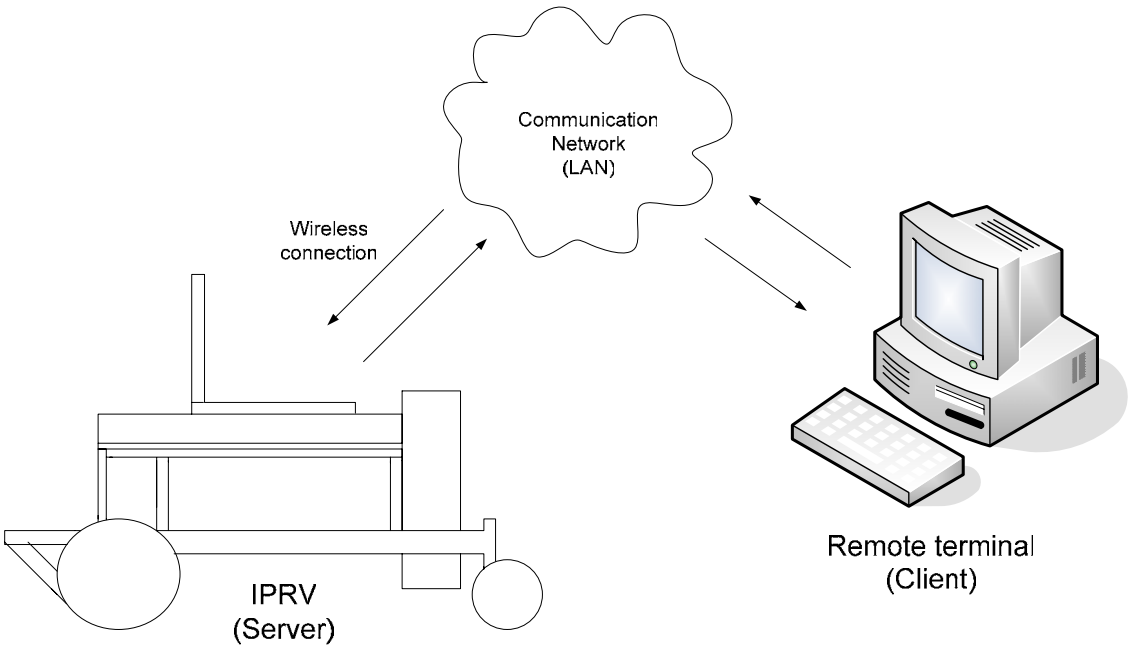

Figure 5.7. Network layout

All commands to the IPRV are sent as strings of data. On receiving data, the server-side program compares the data string with a list of messages that it recognizes. If there is a match then the code corresponding to the particular message is executed. The use of a network to control the IPRV raised two significant safety issues. The issues along with their solutions are discussed below.

5.4.1 Update of global variables

Two global variables are used on the server-side as well as the client-side program to keep track of the current speed and direction of the IPRV. While the variables in the server-side are responsible for the actual speed and direction of the IPRV, the variables on the client-side are used to provide the operator a visual indication of the current status of the IPRV. It is thus extremely important that the variables within both programs have the same value.

Initially the two programs were written to update the global variables based on the number of commands sent or received. However, a wireless network is inherently prone to data-packet losses, and it was found that the variables within the two programs would tend to have different values over a period of time. The rate of packet loss was found to increase on increasing the transmission rate.

In order to eliminate this problem, the IPRV application programs employ a feedback mechanism. The server-side program on receiving a message updates its global variables and returns an acknowledgment of the message received to the client-side program. This acknowledgment is appended with the current value of the 2 global variables. On receiving the acknowledgment, the client-side uses the values of the global

variables sent to update its own variables. Thus at all times the operator is provided with an accurate representation of the IPRV's current speed and direction.

5.4.2 IPRV going out of range

The other significant issue anticipated was that the IPRV may fall out of range of the wireless network during its operation. Another problem is excessive network delays. In both cases the IPRV goes beyond the control of the operator and becomes a serious threat to road safety. It is thus essential that the IPRV be brought to a stop as soon as one of the above conditions occurs.

To achieve this, the server-side program sends a 'check communication' message every 2 seconds to the client-side program. This message is appended with a random integer. As soon as the client-side receives this message, it returns the identical message along with the same random number. Before the next message is sent, the server-side program checks to ensure that the previous message was returned by the client-side. If the message has not been returned or the number received is incorrect then the server-side program stops the IPRV and displays a message indicating that the communication link is down. The use of random numbers ensures that the server-side program does not mistake delayed or resent data packets as valid replies.

5.5 Graphical user interface

In order to facilitate the operation of the IPRV, the client-side program provides an easy-to-use GUI for the operator. Figure 5.7 shows the GUI during the remote maneuvering mode. The GUI is composed of the following 5 elements.

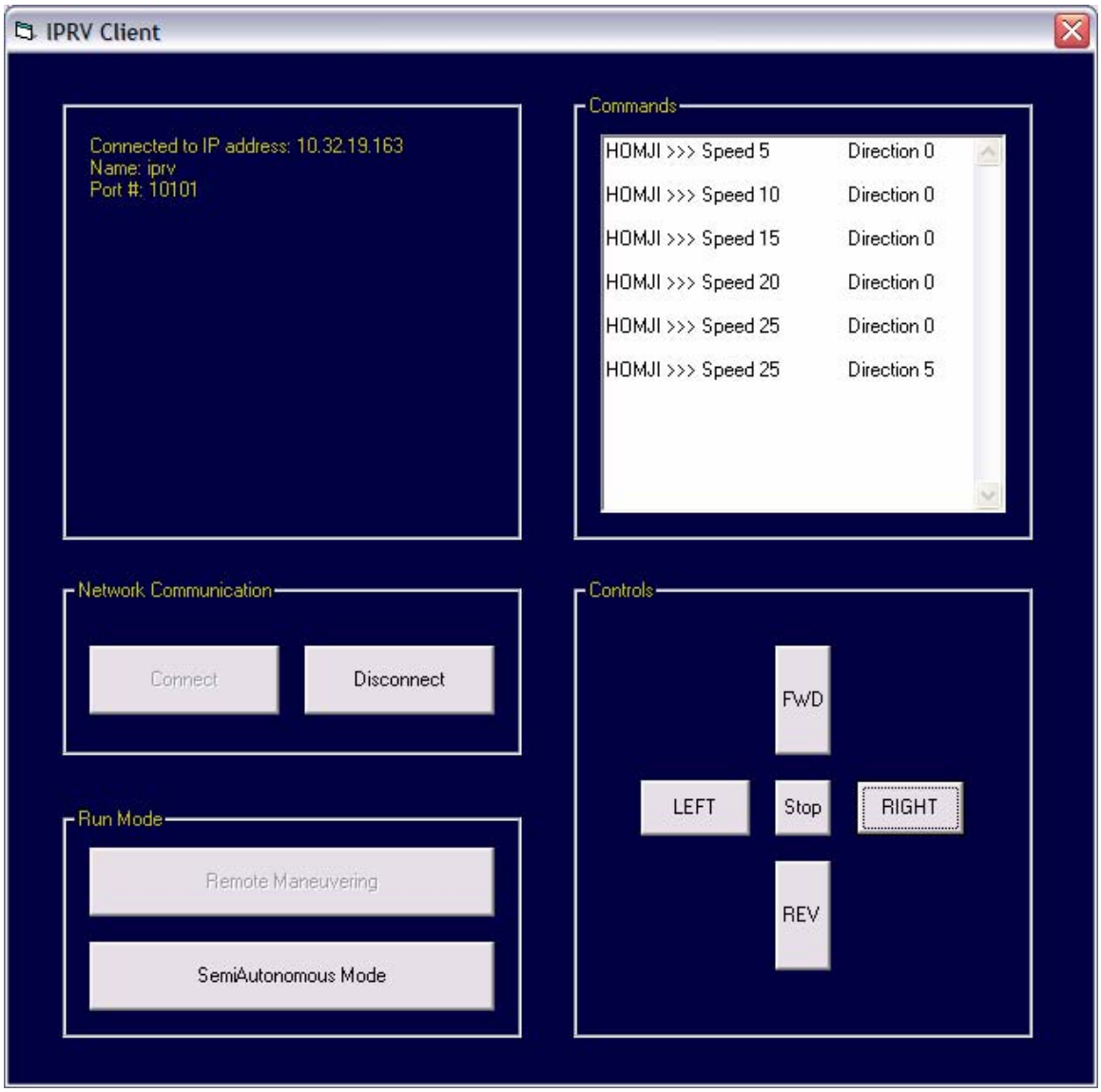

Figure 5.8 Client-side GUI during remote maneuvering

1. Message frame – This is used to inform the operator of the current status of the network connection. It also informs the operator if the run mode is semiautonomous or remote maneuvering.

2. Commands text box – This is used to provide the operator with a visual indication of the IPRV's current speed and direction. A positive speed indicates that the direction of motion is forward, and a positive direction indicates that the direction of motion is to the right. Reverse and left movements are indicated by negative speed and direction values, respectively.

3. Network communication frame – Once the server-side program has been initiated and the server is listening for an incoming connection, this frame can be used to connect or disconnect the client and the server. In order to prevent multiple connection requests from being initiated by a user, the 'connect' button is disabled once a connection has been established and is only enabled if this connection is terminated by either the client-side or the server-side.

4. Run-mode frame – Once a connection is established, the IPRV by default starts in the remote maneuvering mode. The user can use the buttons in this frame to switch between the remote maneuvering and semiautonomous modes. The button for the current run-mode is disabled to prevent multiple initiations.

5. Controls frame – This frame contains the buttons used to send speed and direction commands to the IPRV server-side program. All the buttons are enabled during the remote maneuvering mode. In the semiautonomous mode only the 'stop' and 'FWD' buttons are enabled as the operator can only stop or start the IPRV in this mode.

In addition to the GUI explained above, the client- and server-side programs also provide visual indications in the form of messages for any error or loss of communication that may occur.

CHAPTER VI

OPERATION AND TESTING

6.1 Introduction

The design of the IPRV involved work in three major areas: hardware design, interface design, and software design. These were explained in detail in the previous chapters. This chapter describes how these three designs come together to make the IPRV function as a single unit capable of automating the process of pothole repair. Section 6.2 describes the typical operation of the IPRV when deployed at a work site. Section 6.3 describes the complete set of experiments carried out for the measurement, calibration, and testing of the IPRV.

6.2 IPRV operation

The first step to starting the IPRV is executing the server-side program after switching on power to the motor controller and the individual sensors onboard. On executing the server-side program, the PCMDIO card is opened and the speed and direction of the IPRV are set to 0 (neutral position). The ‘Open Connection’ button is enabled on the server-side GUI. On clicking this button, a TCP socket is created and it ‘listens’ for an incoming connection request. The ‘Close Connection’ button can now be clicked at any time to close the TCP socket.

Next, the client-side program is executed at a workstation on the same LAN. On clicking the ‘Connect’ button on the client-side GUI, a connection request is sent to the server side. If no other connection is currently open, the server accepts the connection

request and a TCP connection is established between the client and server. The client can terminate the connection at any time by clicking the ‘Disconnect’ button.

On establishing a connection, the IPRV is by default in the remote maneuvering mode. In this mode, the operator controls the speed and direction of the IPRV using the control buttons on the client-side GUI. The speed and direction values available for output are restricted to a finite set of values. This is because digital output is being used to drive the IPRV. Hence, every control command increases or decreases the output signal by a fixed amount. The algorithm used for this mode is shown in Figure 6.1.

The operator uses the remote maneuvering mode to position the IPRV at the work site. Once this is done, the operator switches to the semiautonomous mode. The IPRV now runs at a constant speed in a straight direction looking for potholes. Feedback signals from the Hall-effect switches is used maintain the direction (Refer to Sections 3.2.1 and 5.3.2). The wheel at the lower end of the swinging link follows the road contour. If any bumps are encountered by the wheel, the swinging link rotates clockwise and no action is taken by the IPRV. On entering a pothole, the swinging link rotates counter-clockwise. The IPRV now tracks the maximum pothole depth encountered by counting the pulses generated by the optical encoder. A minimum threshold value for the count is set to prevent the pothole filling operation from being initiated for minor potholes that do not need immediate repair. This threshold was determined empirically and has a value of 8 corrosponding to a maximum depth of 2 cm.

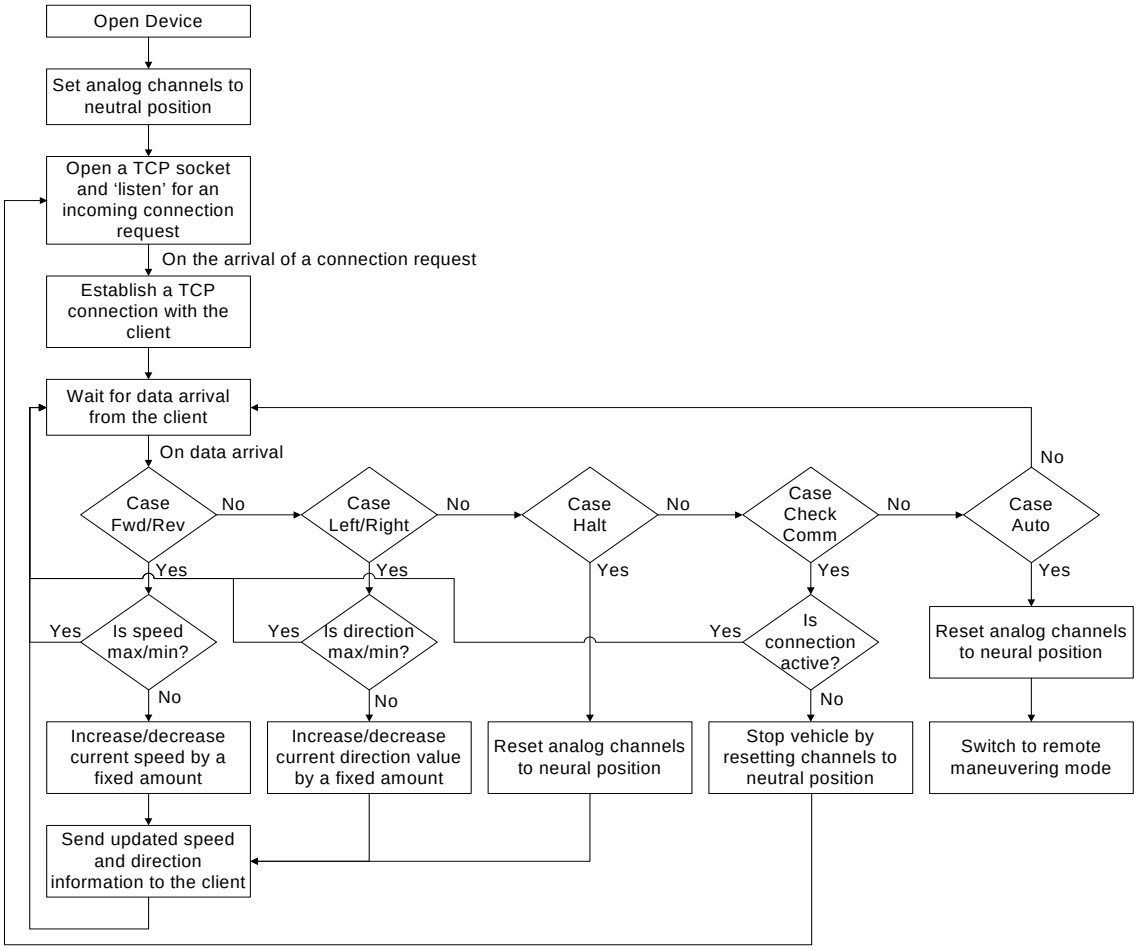

Figure 6.1. Server-side remote maneuvering mode algorithm

Once the swinging link is out of the pothole, the IPRV uses the maximum depth recorded and the number of motor rotations after the maximum depth to calculate the location of the pothole maximum depth. It then proceeds to position its filler valve over this location. Position control is achieved using feedback signals from the right-motor Hall-effect sensor (Refer to Sections 3.2.1 and 5.3.2). The IPRV comes to a halt once it has reached the desired location and the filling operation is initiated by opening the filler valve.

An infrared sensor mounted on the filling tank continuously monitors the filling operation. When the filler material (water used for simulation purposes in this thesis) reaches the road surface, the infrared sensor undergoes a transition in state (high-to-low). This transition is sensed by the server-side program and the filling operation is terminated by closing the filler valve. The IPRV now resumes running at a constant speed in a straight direction until another pothole is detected. Figure 6.2 shows the algorithm for the semiautonomous mode.

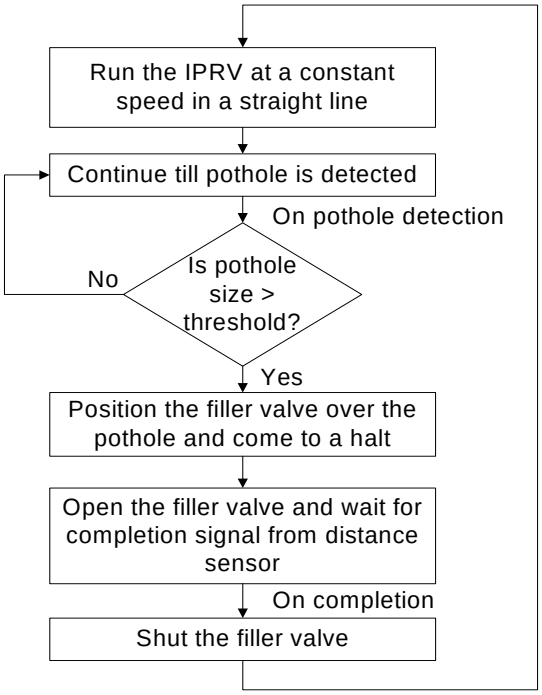

Figure 6.2. Server-side semiautonomous mode algorithm

If required, the operator can stop the IPRV at any stage during the semiautonomous operation. If the IPRV is stopped during a filling operation, the IPRV

will close the filler valve automatically before stopping. However, on resuming semiautonomous operation all the pothole detection variables within the server-side program are reset and the pothole will have to be detected again.

In addition to the operation explained above, the IPRV also implements the following safety features.

1. automatic stop on termination of the TCP socket by the client
2. automatic stop on losing connection with the wireless network
3. automatic stop on experiencing network delays longer than 2 seconds

6.3 IPRV testing and calibration

During the various stages of development of the IPRV, many experiments were conducted in order to test the individual components used onboard, determine any operational limitations, and calibrate the control algorithms being used within the application programs. This section describes these experiments along with the results obtained.

6.3.1 Determination of the maximum sampling frequency available

Knowledge of the maximum sampling frequency available is crucial to avoid ‘aliasing’ when sampling sensor input signals. For the IPRV a program was written to perform 100 consecutive input operations using a single channel. The start and stop time for each set of operations was recorded and used to calculate the average sampling time. The time interval between the operations was gradually reduced to obtain an upper-limit on the achievable sampling frequency. The experiment was performed thrice to check for

deviations in the average time for individual time periods. The results obtained are shown in Table 6.1. The maximum sampling frequency was found to be approximately 400 Hz corresponding to a sampling time of 2.5 ms. The experiment was also performed using multiple channels for output, and the results were found comparable.

Table 6.1. Programmed versus observed sampling times

Programmed sampling time (ms)	Observed sampling time (ms)		
	Set 1	Set 2	Set 3
20	20.5	20.4	20.4
10	10.2	10.5	10.5
1	2.8	2.8	3.21
0.5	2.1	2.4	3.0
0.1	2.1	2.2	2.41
0	2.2	2.4	2.2

Significant discrepancies were observed between average time values recorded for the same time intervals. These discrepancies are due to the use of the Visual Basic 'DoEvents' function within the timer function written by the author. The DoEvents function is used to transfer control to the operating system while a program is waiting for a change of status. This enables other events to be processed in the background. However, the time taken by the operating system to return control to the program is not fixed and hence the discrepancy. Use of the DoEvents function was necessary to ensure that the IPRV can be stopped even if it is in a wait loop.

6.3.2 Determination of the threshold value for pothole detection

As mentioned before, a minimum threshold value is set for the number of pulses generated by the optical encoder on entering a pothole. The purpose of this threshold is to prevent the initiation of a pothole-filling operation every time a small surface irregularity is encountered. There is no defined depth beyond which a pothole must be filled and state transport departments make their own rules with regards to the correct time for pothole repair. For this thesis, the IPRV has been set to repair potholes that have a maximum encountered depth of more than 2 cm. This depth corresponds to a threshold value of 8.

6.3.3 Test of the pothole-detection module

A program implementing the algorithm explained in Section 5.3.3 was written to test the pothole-detection module. A wooden platform, shown in Figure 6.3, was constructed for the experiment. Two potholes with the same maximum depth of 4 cm were used. One pothole had a gradual decline to the maximum depth while the other had a sharp fall. Multiple runs were conducted and the results are shown in Table 6.2.

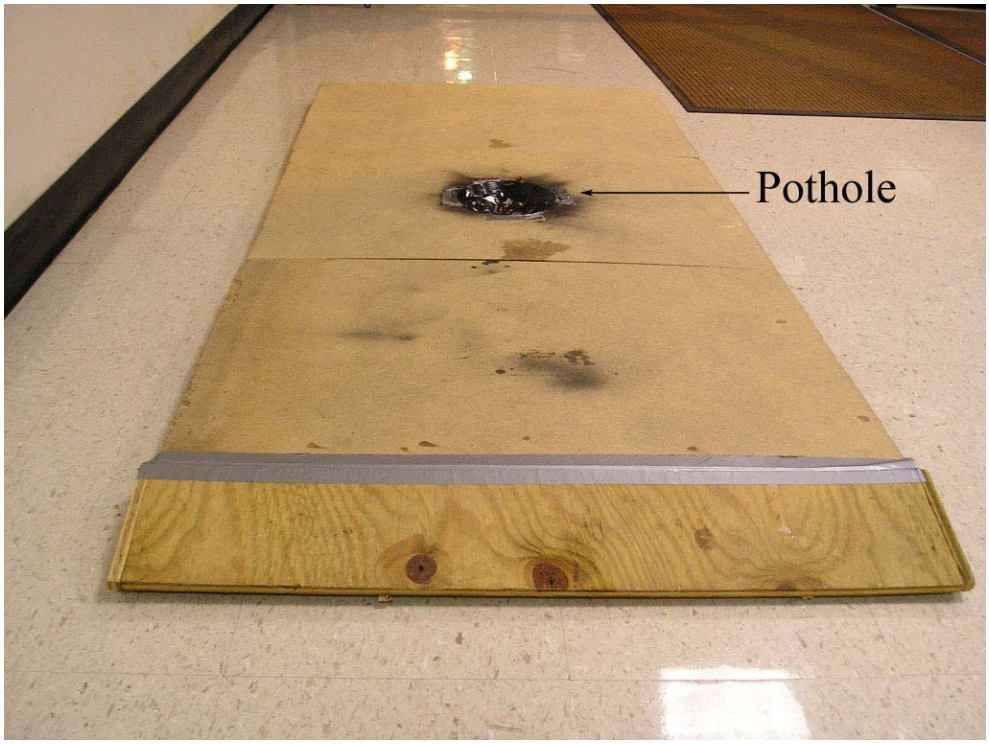

Figure 6.3. Experimental platform

Table 6.2. Pothole-detection module test results

	Pothole with gradual decline		Pothole with sharp fall	
	Max. count recorded	Count at pothole exit	Max. count recorded	Count at pothole exit
Test 1	16	0	9	1
Test 2	17	1	9	0
Test 3	16	0	10	1

As seen in the above table, a significant number of pulses from the optical encoder were lost when the swinging link dropped into a pothole with a steep edge. This

is because in the latter case the available sampling time of 2.5–3 ms was not fast enough to record all the encoder counts. However, so long as the maximum count recorded exceeds the minimum threshold, the primary function of pothole detection remains unaffected because an infrared distance sensor is used for end-point detection. Also, due to the very nature of pothole formation described in Chapter I, potholes tend to have gradually sloping edges and are usually bowl shaped.

6.3.4 Determination of the upper-limit of wheel rotations per minute

The number of rotations of the motor shaft for one rotation of the front wheel was found to be 32. Hence the gear ratio of the motor gear-reduction box is 32:1. The Hall-effect sensors are mounted on the motor shaft and provide two counts per shaft revolution. Three magnets were attached to the disc periphery (Refer to Figure 3.3) to increase the signal pulse duration so that the feedback signal would be in the low and high state for approximately half a revolution each. Let the angular speed of the wheel in rpm be R and the sampling period, T = 2.5 ms. Then,

1 wheel rotation = 64 counts

=> Sampling frequency, $\Omega_s = \frac{2\pi}{T} = 800\pi$ rad/s

From the Nyquist theorem, the Hall-effect switch signal frequency, $\Omega_h < \frac{\Omega_s}{2}$

=> $\Omega_h < 400\pi$ rad/s

Now the wheel frequency, $\Omega_w = \frac{\Omega_h}{64}$

=> $\Omega_w = \frac{2\pi R}{60} < \frac{400\pi}{64}$

=> $R < 187.5$ rpm

Hence the output shaft speed of the IPRV motor must be less than 187.5 rpm to avoid aliasing. The IPRV has a maximum output shaft speed of 180 rpm therefore aliasing is avoided at all speeds.

6.3.5 Determination of the position control parameters

At the moment that the swinging link comes out of a pothole (assuming that the threshold value has been exceeded), the following information in the form of two variables is available to the server-side program.

1) *intMaxAngle* – This variable stores the value of the maximum count generated by the optical encoder within the pothole and represents the maximum change in the angle of the swinging link.

2) *intRpmCount* – This variable stores twice the number of rotations taken by the motor shaft after the maximum depth was encountered and represents the distance of the point of maximum depth from the edge of the pothole.

The distance between the point of maximum depth and the filler valve is required to position the IPRV for the filling operations. This is calculated using the variables defined above as follows.

Number of counts per wheel rotation = 64

Distance moved by the IPRV in one wheel rotation = 99.5 cm

=> The number of counts/cm = $\frac{64}{99.5} = 0.6432$

Figure 6.4 shows the swinging link in three positions: (1) just before the pothole, (2) at the maximum depth of the pothole, and (3) just after the pothole.

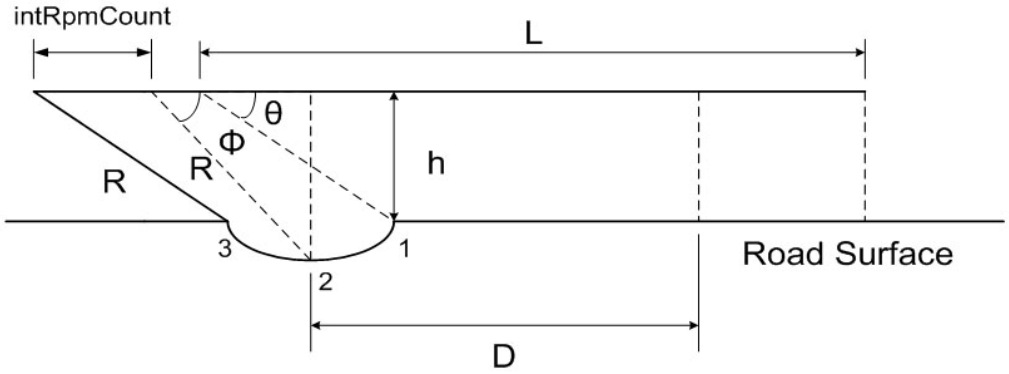

Figure 6.4. IPRV dimensions for position calculation

From the figure, *D* is the distance the IPRV has to be moved to position the filler valve over the maximum depth, *L* is the distance from the filler valve to the point of rotation of the swinging link, *R* is the length of the swinging link, *h* is the distance between a horizontal line passing through the center of the encoder and the road surface, θ is the initial angle of the swinging link from the horizontal axis, and φ is the angle of the swinging link when it is at the maximum depth.

For the IPRV it was found that:

L = 98 cm = 64 counts, *R* = 29 cm = 18.65 counts, and *h* = 18 cm.

Thus $\theta = \sin^{-1}\left(\frac{h}{R}\right) = 38.366^{\circ}$

Resolution of the optical encoder $= \frac{360}{256} = 1.40625^{\circ}$, therefore,

Change in swinging link angle = $\varphi - \theta$ = intMaxAngle * 1.40625°

$\Rightarrow \varphi = (38.366 + \text{intMaxAngle} * 1.40625)^{\circ}$

Now $D = L - R\cos\varphi - \text{intRpmCount}$

$\Rightarrow D = 63 - 18.65 * \cos(38.366 + \text{intMaxAngle} * 1.40625) - \text{intRpmCount}$

The equation above is used to position the IPRV once the swinging link comes out of the pothole. It should also be noted that the resolution of the position control is equal to the distance traveled by the IPRV in half a rotation of the motor shaft. Thus,

$$\text{Position control resolution} = \frac{99.5}{64} = 1.554 \text{ cm}$$

6.3.6 Determination of the position control algorithm

In the previous section, an expression was evaluated for *D,* the distance in counts that the IPRV has to move to position its filler valve over the maximum encountered depth of the pothole. The IPRV has significant inertia and to ensure that it stops exactly when the half-rotation count equals *D*, an algorithm has been implemented wherein the speed of the IPRV is continuously decreased as the desired location comes nearer (Refer to Section 5.3.2). However, due to friction, the IPRV comes to a complete halt at a commanded speed well above 0. Thus the algorithm was modified to reduce the speed up to a value slightly greater than the least speed required to keep the IPRV moving. This value was determined purely by observation of the IPRV at different running speeds. The minimum value was found to be 20, which corresponds to an analog voltage of 2.89 V across the speed control channel.

CHAPTER VII

CONCLUSIONS

7.1 Introduction

During the course of this thesis, the IPRV was successfully constructed and tested. Section 7.2 summarizes the accomplishments of the thesis. Section 7.3 discusses the current limitations of the IPRV, and in Section 7.4 future work is proposed to enhance the functionality of the IPRV and mitigate the current limitations.

7.2 Conclusions

The IPRV has been successful in achieving its key objective of automating the pothole repair process. It is capable of automatically detecting and filling potholes on road surfaces. It was designed to detect potholes greater than 2 cm in depth. The IPRV is also capable of distinguishing between potholes and bumps and ignores the bumps. A significant advantage of the IPRV is that it is remotely operable over a LAN thus eliminating the need for an operator onboard. Once in its semiautonomous mode, the operator is only required to start and stop the IPRV at the beginning and end of each run.

Road safety has been of primary importance throughout the development of the IPRV. In order to ensure a reliable means of network communication, TCP was chosen as the transport protocol for the network interface. In addition, the IPRV employs safety mechanisms that ensure that the vehicle is automatically stopped in the cases of connection termination, loss of communication, or network delays longer than 2 seconds.

Another feature unique to the IPRV is its use of a mechanical means for pothole detection. This provided an easy-to-construct and significantly cheaper solution for pothole detection. This pothole-detection method is also less computationally intensive when compared with the video image processing methods employed in the past by other researchers.

The IPRV employs a feedback mechanism to achieve position control when operating in the semiautonomous mode. The position resolution is 1.554 cm, significantly smaller than the potholes that require filling. In addition to position control, the IPRV also uses feedback to ensure that it follows a straight path during semiautonomous operation. If the need arises, the IPRV can be brought to an immediate stop at any stage of its operation. If the IPRV is stopped in the midst of a filling operation, it automatically shuts the filler valve first.

The IPRV was designed for possible future expansion. Thus the electrical interfaces have been designed such that none of the sensors are permanently mounted but rather interfaced using connectors. This allows the easy replacement of any faulty sensors as well as the provision of adding functionality to the IPRV with minimal of effort. Power supplies to each sensor are routed through a switch. This allows the isolation of any part of the circuit as well as aids in fault detection.

Visual Basic was found to be an appropriate choice for the development of all the IPRV software application programs. The Windows API was used for controlling the data-acquisition card within the Visual Basic environment.

7.3 Limitations

The IPRV in its current form has the following limitations.

1. Limited processor speed has prevented the use of a webcam to send live streaming images to the client side. Real-time video capability is a fundamental requirement for any remotely operable system. This limitation has restricted the use of the IPRV to within a visual distance of the operator.

2. The maximum sampling frequency available to the IPRV was found to be 400 Hz. This proves to be insufficient in cases where a pothole has steep edges or sharp falls. Thus in such circumstances, aliasing occurs and filler-valve positioning may be offset.

3. The motion of the IPRV is restricted to a straight line during the semiautonomous operation. This in turn implies that multiple runs are required to cover an area that is wider than the wheel base of the IPRV.

4. The size and location of the IPRV limits accurate detection. For a pothole to be detected, it must lie within the wheel base of the IPRV in such a way that the swinging link is deflected to a point beyond the minimum threshold value.

7.4 Future work

The following is proposed as future work to enhance the functionality of the IPRV and mitigate the current limitations.

1. Use of a second processor to send live streaming images to the client computer.

2. Use of a 1 MHz external clock to sample sensor signals. This can lead to a sampling period of 1 μs, which is a dramatic improvement to the current 2.5 ms.

3. Implementation of path-planning and collision-avoidance, artificial-intelligence, or other adaptive algorithms with an aim to fully automate the IPRV and eliminate the need for remote supervision.

REFERENCES

[1] *Pothole Patching Resources*, U.S. Army Engineer Research and Development Center (ERDC). http://www.erdc.usace.army.mil, Accessed on May 2005.

[2] R. A. Eaton, R. H. Joubert, and E. A. Wright, "Pothole Primer – A Public Administrator's Guide to Understanding and Managing the Pothole Problem," CRREL Special Report 81-21 (revised December 1989). Available at http://www.crrel.usace.army.mil/research/Pothole_Primer.pdf.

[3] K. L. Smith, D. G. Peshkin, E. H. Rmeili, T. V. Dam, K. D. Smith, and M. I. Darter, "Innovative Materials and Equipment for Pavement Surface Repairs – Volume I: Summary of Material Performance and Experimental Plans," Report No. SHRP-M/UFR-91-504, contract H-105, February 1991.

[4] T. P. Wilson and A. R. Romine, "Materials and Procedures for Repair of Potholes in Asphalt-surfaced Pavements – Manual of Practice," Report No. SHRP-H-348, August 1993.

[5] T. P. Wilson and A. R. Romine, "Materials and Procedures for Repair of Potholes in Asphalt-surfaced Pavements – Manual of Practice," Report No. FHWA-RD-99-168, February 2001.

[6] T. P. Wilson and A. R. Romine, "Innovative Materials Development and Testing – Volume 2: Pothole Repair," Report No. SHRP-H-353, contract H-106, October 1993.

[7] A. Heydorn, "The Future of Pothole Repair," *Pavement*, vol. 18, issue 7, pp. 34–36, October 2003.

[8] L. D. Evans, C. G. Mojab, A. J. Patel, A. R. Romine, K. L. Smith, and T. P. Wilson, "Innovative Materials Development and Testing – Volume 1: Project Overview," Report No. SHRP-H-352, contract H-106, October 1993.

[9] A. Griffith, "Improved Winter Pothole Patching," State Planning and Research, Project Number 538, Oregon Department of Transportation, August 1998.

[10] J. R. Blaha, "Fabrication and Testing of Automated Pothole Patching Machine," Report No. SHRP-H-674, contract H-107B, October 1993.

[11] J. Karuppuswamy, V. Selvaraj, M. M. Ganesh, and E. L. Hall, "Detection and Avoidance of Simulated Potholes in Autonomous Vehicle Navigation in an Unstructured Environment," *Proceeding of SPIE*, vol. 4197, *Intelligent Robots and Computer Vision XIX: Algorithms, Techniques and Active Vision*, pp. 70–80 October 2000.

[12] L. Matthies and A. Rankin, "Negative Obstacle Detection by Thermal Signature," *Proceedings of the 2003 IEEE International Conference on Intelligent Robots and Systems*, pp. 906–913, October 2003.

[13] "Investigation of a Pavement Crack Filling Robot," Department of Civil Engineering, Carnegie Mellon University, Report No. SHRP-ID/UFR-92-616, contract ID-017, November 1992.

[14] S. A. Velinsky, "Fabrication and Testing of an Automated Crack Sealing Machine," Report No. SHRP-H-659, contract H-107A, August 1993.

[15] "Sandian's Rapid Road Repair Vehicle Would Fix Potholes on the Fly," June 1998, http://www.sandia.gov/media/pothole.htm.

[16] B. Graham and K. McGowan, *Build Your Own All-Terrain Robot*, New York: McGrawHill, 2004.

[17] *Superlogics PCMDIO Users Manual*. Available at http://www.SuperLogics.com.

[18] *Superlogics PCMDRIVE® Data Acquisition Software User's Manual*. Available at http://www.SuperLogics.com.

[19] RFC 793, J. Postel (ed.), "Transmission Control Protocol: DARPA Internet Program Protocol Specification," September 1981.

APPENDIX A

SERVER-SIDE PROGRAM

The server-side program is a Visual Basic project consisting of the following forms and modules.

1. IprvMainForm.frm – This form provides the GUI of the server-side program. Additionally, the code within this form defines all the event-driven procedures used by the server-side program.

2. PCMMain.bas – Execution of the server-side program starts with this module. The IprvMainForm.frm is executed within this module. This module also contains the functions used for the semiautonomous mode.

3. PCMUserFunc.bas – This module contains the designed to simplify the operation of the PCMDIO data-acquisition card.

4. PCMDrvFunc.bas – This module was provided with the PCMDIO card and is freely downloadable at www.Superlogics.com. The module contains the API function declarations required for controlling the PCMDIO card.

5. PCMData.bas – This module was also provided with the PCMDIO card and is freely downloadable at www.Superlogics.com. The module contains the API data structures and constants required to control the PCMDIO card.

1. IprvMainForm.frm

```
Option Explicit

Private Sub Form_Load()
    Dim strInfo As String

    cmdListen.Enabled = True
    cmdDisconnect.Enabled = False

    'Disable the timer till a connection is established
    tmrCheckComm.Interval = 2000    ' 2 seconds
    tmrCheckComm.Enabled = False

    strInfo = vbCrLf & vbCrLf & "To start listening for an incoming
client request, " & _
"click on the Open Connection button" & vbCrLf & vbCrLf & _
"Click on the Disconnect button anytime to disconnect the TCP
connection"

    lblInfo.Caption = strInfo

End Sub

Private Sub cmdListen_Click()

    ' Set up the local port and wait for a connection request
    tcpIprvServer.LocalPort = 10101
    tcpIprvServer.Listen

    cmdListen.Enabled = False
    cmdDisconnect.Enabled = True
    txtCommands.Text = ""
    lblInfo.Caption = "The IPRV is now waiting to receive an incoming
connection request."

End Sub

Private Sub cmdDisconnect_Click()
    'Close the socket after checking the status
    If tcpIprvServer.State <> sckClosed Then
        Call tcpIprvServer.Close
    End If

    'Stop the IPRV and disable the timer
    Call resetDacChannels
    tmrCheckComm.Enabled = False

    cmdListen.Enabled = True
    cmdDisconnect.Enabled = False
    lblInfo.Caption = "Connection has been closed by the Server. " &
vbCrLf & vbCrLf & _
"Click on Open Connection to start listening for a new connection."
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    Call resetDacChannels

    If tcpIprvServer.State <> sckClosed Then
        Call tcpIprvServer.Close
    End If

End Sub

Private Sub tcpIprvServer_ConnectionRequest(ByVal requestID As Long)
    'First ensure that the tcpIprvServer is closed
    'If not, close the connection before accepting the new connection

    If tcpIprvServer.State <> sckClosed Then
        Call tcpIprvServer.Close
    End If

    Call tcpIprvServer.Accept(requestID)      'accept the incoming
connection

    'The connection will always start in the Remote Maneuvering Mode
    gblnRunModeAuto = False
    gblnRunIprv = False

    cmdDisconnect.Enabled = True
    txtCommands.Text = ""

    lblInfo.Caption = "Connection from IP address: " &
tcpIprvServer.RemoteHostIP & vbCrLf & _
                "Name: " & tcpIprvServer.RemoteHost & vbCrLf & _
                "Port #: " & tcpIprvServer.RemotePort & vbCrLf & vbCrLf

    'The connection is now established, the timer is now enabled
    tmrCheckComm.Enabled = True

End Sub

Private Sub tcpIprvServer_DataArrival(ByVal bytesTotal As Long)
    Dim strInputData As String
    Dim intStatus As Integer
    Dim strMessage As String
    Dim strCommand() As String
    Dim intCheckCommValue As Integer

    Call tcpIprvServer.GetData(strInputData)

    strCommand() = Split(strInputData, , -1)

    Select Case strCommand(0)
        Case "Forward"
            If gbytFwdRev < MAX_FWD_REV_CH_VALUE Then
                gbytFwdRev = gbytFwdRev + 5
```

```
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_FWD_REV)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, gbytFwdRev)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
            End If
        Case "Reverse"
            If gbytFwdRev > MIN_FWD_REV_CH_VALUE Then
                gbytFwdRev = gbytFwdRev - 5
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_FWD_REV)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, gbytFwdRev)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
            End If
        Case "Right"
            If gbytLeftRight < MAX_LEFT_RIGHT_CH_VALUE Then
                gbytLeftRight = gbytLeftRight + 5
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_LEFT_RIGHT)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, gbytLeftRight)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
            End If
        Case "Left"
            If gbytLeftRight > MIN_LEFT_RIGHT_CH_VALUE Then
                gbytLeftRight = gbytLeftRight - 5
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_LEFT_RIGHT)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
                intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, gbytLeftRight)
                If intStatus <> 0 Then
                    Call pcmdioError(gintLogicalDevice, intStatus)
                End If
            End If
        Case "Halt"
```

```
            Call resetDacChannels
        Case "Auto"
            'Start IPRV in SemiAutonomous mode after bringing it to a
complete halt.
            Call resetDacChannels
            gblnRunModeAuto = True
            gblnRunIprv = True
            lblInfo.Caption = "The IPRV is now in Semi Autonomous
mode." & vbCrLf & vbCrLf & _
                              "All operations will now run
automatically. Only Start/Stop can be performed remotely."
            Call runIPRV
        Case "Remote"
            'Stop the IPRV and switch to Remote Maneuvering mode
            Call resetDacChannels
            gblnRunModeAuto = False
            lblInfo.Caption = "The IPRV is now in the Remote
Maneuvering Mode." & vbCrLf & vbCrLf

        Case "CheckComm"
            'if the value of CheckComm does not match the expected
            'value, then stop the IPRV
            intCheckCommValue = Format(strCommand(1))
            If intCheckCommValue <> gintCheckComm Then
                Call resetDacChannels
                Exit Sub
            End If
            gintCheckComm = 0
            'Debug.Print "Communication link good"
            Exit Sub
        Case Else
            Exit Sub
    End Select

    strMessage = "Fwd " & gbytFwdRev & " Rev " & gbytLeftRight
    Call tcpIprvServer.SendData(strMessage)

    txtCommands.Text = txtCommands.Text & "Client" & " >>> " & _
                       "Speed " & gbytFwdRev - 128 & vbTab & "Direction
" & gbytLeftRight - 128 & vbCrLf & vbCrLf
    txtCommands.SelStart = Len(txtCommands.Text)


End Sub

Private Sub tcpIprvServer_Close()

    If tcpIprvServer.State <> sckClosed Then
        Call tcpIprvServer.Close
    End If

    'If the client closes the connection, stop the IPRV by resetting
    'the values of the Forward/Reverse and Left/Right channels
```

```
    Call resetDacChannels
    tmrCheckComm.Enabled = False

    lblInfo.Caption = "Client closed connection." & vbCrLf & vbCrLf & _
                      "Ready to receive a new connection."
    cmdListen.Enabled = True

    txtCommands.Text = txtCommands.Text & "Client" & " >>> " & _
                       "Speed " & gbytFwdRev - 128 & vbTab & "Direction
" & gbytLeftRight - 128 & vbCrLf & vbCrLf

    ' Set up local port and wait for a connection request
    tcpIprvServer.LocalPort = 10101
    tcpIprvServer.Listen

    cmdListen.Enabled = False
    cmdDisconnect.Enabled = True

End Sub

Private Sub tcpIprvServer_Error(ByVal Number As Integer, Description As
String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As
String, ByVal HelpContext As Long, CancelDisplay As Boolean)
    Call resetDacChannels

    MsgBox Source & ": " & Description, vbOKOnly, "IPRV Communication
Error"
    Unload Me
End Sub

Private Sub tmrCheckComm_Timer()
    Dim strCheckComm As String

    'Every 2 seconds the value of gintCheckComm is checked.
    'If the connection is still established then a new random number
    'is generated and sent to the client.
    'if the value of gintCheckComm is not 0, it implies that either
    'the connection is broken or there is a delay > 2 secs over the
network
    'In either case the IPRV is stopped.

    If gintCheckComm <> 0 Then
        Call resetDacChannels
        MsgBox "Lost Communication with the Client.", vbOKCancel +
vbExclamation
        If tcpIprvServer.State <> sckClosed Then
            tcpIprvServer.Close
        End If
        tmrCheckComm.Enabled = False
        gintCheckComm = 0

    'Wait for a new incoming connection
        tcpIprvServer.LocalPort = 10101
        tcpIprvServer.Listen
```

```
        lblInfo.Caption = "Connection Closed." & vbCrLf & vbCrLf & _
                          "Ready to receive a new connection."
    Else
        gintCheckComm = Round(100 * Rnd())
        strCheckComm = "CheckComm " & gintCheckComm
        tcpIprvServer.SendData (strCheckComm)

    End If
End Sub
```

2. PCMMain.bas

```
Option Explicit

'Define Output Channels

Global Const DAC_DATA_CHANNEL = 0               'Data0 - Data7
Global Const DAC_LOGIC_CHANNEL = 1              'Data8 - Data9
Global Const RESERVED_CHANNEL1 = 2              'Data10
Global Const RESERVED_CHANNEL2 = 3              'Data11
Global Const LED_HOLEFOUND_CHANNEL = 4          'Data12
Global Const SOLENOID_VALVE_CHANNEL = 5         'Data13

'Define Input Channels

Global Const DIST_SENSOR_CHANNEL = 10           'Data18
Global Const LEFT_MOTOR_SENSOR_CHANNEL = 12     'Data20
Global Const RIGHT_MOTOR_SENSOR_CHANNEL = 13    'Data21
Global Const INCREMENTAL_ENCODER_CHANNEL = 14   'Data22 - Data23

'Define DAC Control Logic bits

Global Const CHANNEL_FWD_REV = 0
Global Const CHANNEL_LEFT_RIGHT = 1
Global Const CHANNEL_SPARE1 = 2
Global Const CHANNEL_SPARE2 = 3

'Define min and max value of DAC channel outputs

Global Const MIN_FWD_REV_CH_VALUE = 51
Global Const MAX_FWD_REV_CH_VALUE = 195
Global Const MIN_LEFT_RIGHT_CH_VALUE = 40
Global Const MAX_LEFT_RIGHT_CH_VALUE = 215

Global Const IPRV_MAX_AUTO_RUN_VALUE = 190
Global Const HOLE_DETECTION_THRESHOLD = 4

Public gintLogicalDevice As Integer
Public gbytFwdRev As Byte
Public gbytLeftRight As Byte
Public gintCheckComm As Integer
Public gblnRunModeAuto As Boolean
Public gblnRunIprv As Boolean


Private Sub main()

    Dim frmMain As New frmIprvMain
    Dim intStatus As Integer

    On Error GoTo errUnknown

    gintLogicalDevice = 0
```

```
    'Open the PCMDIO digital I/O card and get a
    'logical device number

    gintLogicalDevice = openDevice()

    'Initialize the Forward/Reverse channel and the
    'Left/Right channel

    Call resetDacChannels

    'Display the main form that is used to communicate with a client

    frmMain.Show vbModal

    End

errPcmdioError:
    Call errorMessage(intStatus)
    Call PCMCloseDeviceVB(gintLogicalDevice)
    End

errUnknown:
    Call unknownErrorMessage
    End

End Sub


Public Sub pcmdioError(ByVal LogicalDevice As Integer, ByVal ErrorCode
As Integer)
    Call errorMessage(ErrorCode)
    Call PCMCloseDeviceVB(LogicalDevice)
End Sub
'Function used to stop the IPRV - it sets the speed and direction
channels to 128
'which corresponds to a value of 2.5V across each channel.
'Additionally, it also makes sure the solenoid valve is shut and the
hole
'found LED is off.

Public Sub resetDacChannels()
    Dim intStatus As Integer

    gbytFwdRev = 128
    gbytLeftRight = 128

    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_FWD_REV)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, gbytFwdRev)
```

```
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_LEFT_RIGHT)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, gbytLeftRight)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

    intStatus = singleDigitalOutput(gintLogicalDevice,
LED_HOLEFOUND_CHANNEL, 0)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

    If (gblnRunModeAuto = True) Then
        gblnRunIprv = False
        intStatus = singleDigitalOutput(gintLogicalDevice,
SOLENOID_VALVE_CHANNEL, 0)
        If intStatus <> 0 Then
            Call pcmdioError(gintLogicalDevice, intStatus)
        End If
    End If


End Sub

'This function is used to increase/decrease the speed of the IPRV

Public Sub iprvSpeed(ByVal bytSpeed As Byte)
    Dim intStatus As Integer
    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_FWD_REV)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, bytSpeed)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

End Sub

'This function is used to change the direction of the IPRV
```

```
Public Sub iprvDirection(ByVal bytDirection As Byte)
    Dim intStatus As Integer
    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_LOGIC_CHANNEL, CHANNEL_LEFT_RIGHT)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

    intStatus = singleDigitalOutput(gintLogicalDevice,
DAC_DATA_CHANNEL, bytDirection)
    If intStatus <> 0 Then
        Call pcmdioError(gintLogicalDevice, intStatus)
    End If

End Sub

'This is the function that controls the IPRV in the semiautonomous mode

Public Sub runIPRV()

    Dim intStatus As Integer
    Dim intAngleCounter As Integer
    Dim intMaxAngle As Integer
    Dim intRpmCounter As Integer
    Dim bytInputSensorNew(2) As Byte
    Dim bytInputSensorOld(2) As Byte
    Dim bytInputData(2) As Byte
    Dim intChannels(2) As Integer
    Dim intArrayLength As Integer
    Dim intArraySize As Integer
    Dim intHoleFull As Integer
    Dim blnCountRpm As Boolean
    Dim i As Integer
    Dim intPosition As Integer
    Dim lngPosition As Long
    Dim bytNewPosition As Byte
    Dim bytOldPosition As Byte
    Dim bytNewDist As Byte
    Dim intRightCounter As Integer
    Dim intLeftCounter As Integer
    Dim intChannel(1) As Integer
    Dim bytOldValue(1) As Byte
    Dim bytNewValue(1) As Byte
    Dim bytInputValue(1) As Byte


    intChannels(0) = LEFT_MOTOR_SENSOR_CHANNEL
    intChannels(1) = RIGHT_MOTOR_SENSOR_CHANNEL
    intChannels(2) = INCREMENTAL_ENCODER_CHANNEL

    intArrayLength = 3
    Debug.Print "Starts Here"

    While (gblnRunIprv = True)
```

```
        DoEvents
        If gblnRunIprv = False Then
            Exit Sub
        End If

        'Debug.Print "Running"

        '-----------------------------------------------------------------
        'Run the IPRV at a constant speed in a straight line
        '-----------------------------------------------------------------

        'the initial direction compensates for differences in tire
      pressures
        'and other motor components that make one motor turn at a
      different
        'rate than the other. This compensatory value was determined
      emperically

        gbytFwdRev = IPRV_MAX_AUTO_RUN_VALUE
        gbytLeftRight = 128

        Call iprvSpeed(gbytFwdRev)
        Call iprvDirection(gbytLeftRight)

        '-----------------------------------------------------------------
        'Look for potholes
        '-----------------------------------------------------------------

        intStatus = multipleDigitalInput(gintLogicalDevice,
intChannels(), intArrayLength, bytInputData())
        If intStatus <> 0 Then
            Call pcmdioError(gintLogicalDevice, intStatus)
        End If
        'Initialize values
        For i = 0 To 2
            bytInputSensorOld(i) = bytInputData(i)
        Next i
        intMaxAngle = 0
        intAngleCounter = 0
        intRpmCounter = 0

        'While looking for a pothole, interations should continue until
        'the pothole has been detected AND the swinging link is out of
      the pothole

        While (intMaxAngle < HOLE_DETECTION_THRESHOLD) Or
(intAngleCounter > 3)
            DoEvents
            If gblnRunIprv = False Then
                Exit Sub
            End If
```

```
            intStatus = multipleDigitalInput(gintLogicalDevice,
intChannels(), intArrayLength, bytInputData())
            If intStatus <> 0 Then
                Call pcmdioError(gintLogicalDevice, intStatus)
            End If
            For i = 0 To 2
                bytInputSensorNew(i) = bytInputData(i)
            Next i

            'Debug.Print bytInputSensorNew(i)

            If bytInputSensorOld(0) <> bytInputSensorNew(0) Then
                intLeftCounter = intLeftCounter + 1
            End If

            If bytInputSensorOld(1) <> bytInputSensorNew(1) Then
                intRightCounter = intRightCounter + 1
            'Count the motor rotations only after a hole has been
            detected
                If (intMaxAngle >= HOLE_DETECTION_THRESHOLD) Then
                    intRpmCounter = intRpmCounter + 1
                End If
            End If

            If intRightCounter = 5 Then
                If intLeftCounter > 5 Then
                    gbytLeftRight = gbytLeftRight - 2
                    Call iprvDirection(gbytLeftRight)
                    Debug.Print "Move to the left"
                    intLeftCounter = 0
                    intRightCounter = 0
                ElseIf intLeftCounter < 5 Then
                    gbytLeftRight = gbytLeftRight + 2
                    Call iprvDirection(gbytLeftRight)
                    Debug.Print "Move to the right"
                    intLeftCounter = 0
                    intRightCounter = 0
                Else
                    intLeftCounter = 0
                    intRightCounter = 0
                End If
            End If

            'Check if swinging link has rotated
            If (bytInputSensorNew(2) <> bytInputSensorOld(2)) Then
                Select Case bytInputSensorOld(2)
                    Case 0
                        If bytInputSensorNew(2) = 2 Then
                            intAngleCounter = intAngleCounter + 1
                        Else
                            intAngleCounter = intAngleCounter - 1
                        End If
                    Case 1
                        If bytInputSensorNew(2) = 0 Then
```

```
                            intAngleCounter = intAngleCounter + 1
                        Else
                            intAngleCounter = intAngleCounter - 1
                        End If
                    Case 2
                        If bytInputSensorNew(2) = 3 Then
                            intAngleCounter = intAngleCounter + 1
                        Else
                            intAngleCounter = intAngleCounter - 1
                        End If
                    Case Else
                        If bytInputSensorNew(2) = 1 Then
                            intAngleCounter = intAngleCounter + 1
                        Else
                            intAngleCounter = intAngleCounter - 1
                        End If

                End Select

                Debug.Print intAngleCounter

                'intMaxAngle will find the max depth encountered by the
                swinging
                'link. A pothole may have several local maximas but we
                need the
                'maximum value of all maximas. Everytime a new maxvalue
                is encountered
                'the rpm counter must be reset

                If intMaxAngle < intAngleCounter Then
                    intMaxAngle = intAngleCounter
                    intRpmCounter = 0
                End If
            End If
            For i = 0 To 2
                bytInputSensorOld(i) = bytInputSensorNew(i)
            Next i

        Wend

        'Out of the loop implies that a pothole has been located
        'The pothole holefound LED can now be lit

        intStatus = singleDigitalOutput(gintLogicalDevice,
LED_HOLEFOUND_CHANNEL, 1)
        If intStatus <> 0 Then
            Call pcmdioError(gintLogicalDevice, intStatus)
        End If

        'At this point, the swinging link is out of the pothole and
        'the value of intMaxAngle and intRpmCounter is known
        'Thus we have the max depth encountered and the distance from
      the
        'maximum depth
```

```
        'Debug.Print intMaxAngle
        'Debug.Print intAngleCounter

        lngPosition = 63 - 18.65 * Cos((38.36 + 1.40625 * intMaxAngle)
* 3.141 / 180) - intRpmCounter
        intPosition = Round(lngPosition)
        Debug.Print intPosition

        'once the location has been calculated, the right motor sensor
      signal is used
        'for position the IPRV over the calculated point

        intChannel(0) = LEFT_MOTOR_SENSOR_CHANNEL
        intChannel(1) = RIGHT_MOTOR_SENSOR_CHANNEL
        intArraySize = 2

        intStatus = multipleDigitalInput(gintLogicalDevice,
intChannel(), intArraySize, bytInputValue())
        If intStatus <> 0 Then
            Call pcmdioError(gintLogicalDevice, intStatus)
        End If
        bytOldValue(0) = bytInputValue(0)
        bytOldValue(1) = bytInputValue(1)
        intLeftCounter = 0
        intRightCounter = 0

        While (intPosition >= 1)

            DoEvents
            If gblnRunIprv = False Then
                Exit Sub
            End If
            intStatus = multipleDigitalInput(gintLogicalDevice,
intChannel(), intArraySize, bytInputValue())
            If intStatus <> 0 Then
                Call pcmdioError(gintLogicalDevice, intStatus)
            End If
            bytNewValue(0) = bytInputValue(0)
            bytNewValue(1) = bytInputValue(1)

            If bytNewValue(0) <> bytOldValue(0) Then
                intLeftCounter = intLeftCounter + 1
                bytOldValue(0) = bytNewValue(0)
            End If

            If bytNewValue(1) <> bytOldValue(1) Then
                intRightCounter = intRightCounter + 1
                intPosition = intPosition - 1
                Debug.Print "right counter" & intRightCounter
                Debug.Print "position counter" & intPosition
                If (intPosition < 25) Then
                    gbytFwdRev = gbytFwdRev - 1
                    Call iprvSpeed(gbytFwdRev)
```

```
                End If
                bytOldValue(1) = bytNewValue(1)
            End If

            If intRightCounter = 5 Then
                If intLeftCounter > 5 Then
                    gbytLeftRight = gbytLeftRight - 2
                    Call iprvDirection(gbytLeftRight)
                    Debug.Print "Move to the left"
                    intLeftCounter = 0
                    intRightCounter = 0
                ElseIf intLeftCounter < 5 Then
                    gbytLeftRight = gbytLeftRight + 2
                    Call iprvDirection(gbytLeftRight)
                    Debug.Print "Move to the right"
                    intLeftCounter = 0
                    intRightCounter = 0
                Else
                    intLeftCounter = 0
                    intRightCounter = 0
                End If
            End If
        Wend

        'Once the error has reached zero, implies that the filler valve
      is over the pothole maximum depth. The IPRV is now stopped and
      the filling operation now begins

        gbytFwdRev = 128
        gbytLeftRight = 128

        Call iprvSpeed(gbytFwdRev)
        Call iprvDirection(gbytLeftRight)
        Call iprvSpeed(gbytFwdRev)

        intStatus = singleDigitalOutput(gintLogicalDevice,
SOLENOID_VALVE_CHANNEL, 1)
        If intStatus <> 0 Then
            Call pcmdioError(gintLogicalDevice, intStatus)
        End If

        intHoleFull = 0

        While (intHoleFull <= 10)
            DoEvents
            If gblnRunIprv = False Then
                Exit Sub
            End If

            intStatus = singleDigitalInput(gintLogicalDevice,
DIST_SENSOR_CHANNEL, bytNewDist)
            If intStatus <> 0 Then
                Call pcmdioError(gintLogicalDevice, intStatus)
            End If
```

```
            Debug.Print bytNewDist
            If bytNewDist = 1 Then
                intHoleFull = intHoleFull + 1
            Else
                intHoleFull = 0
            End If
            Call waitTime(100)
        Wend

        'Coming out of the loop implies that the hole has been filled
        'the value of intHoleFull is checked 3 times to ensure that
      splashing water was not the reason for a 'full' indication by the
      distance sensor

        'Now the solenoid valve and the LED can be shut

        intStatus = singleDigitalOutput(gintLogicalDevice,
SOLENOID_VALVE_CHANNEL, 0)
        If intStatus <> 0 Then
            Call pcmdioError(gintLogicalDevice, intStatus)
        End If

        intStatus = singleDigitalOutput(gintLogicalDevice,
LED_HOLEFOUND_CHANNEL, 0)
        If intStatus <> 0 Then
            Call pcmdioError(gintLogicalDevice, intStatus)
        End If

    Wend

End Sub
```

3. **PCMUserFunc.bas**

```
Option Explicit

Public gstrDeviceType As String
Public gstrDllName As String
Public gstrConfigFile As String

'*********************************************************************
' Function: openDevice
' Purpose: Define the hardware configuration and open the
'          hardware device
' Returns: The Logical Device number
'*********************************************************************


Public Function openDevice() As Integer
    Dim intStatus As Integer
    Dim intLogicalDevice As Integer

    On Error GoTo errUnknown

    intLogicalDevice = 0

    'Define the device type, dll and config files

    gstrDeviceType = "PCMDIO"
    gstrDllName = "pcmdio32.dll"
    gstrConfigFile = "C:\Program Files\SuperLogics\PCMCfg\PCMDIO.dat"

    intStatus = PCMOpenDeviceVB(gstrDllName, intLogicalDevice,
gstrDeviceType, gstrConfigFile)

    'Debug.Print "Open Device status = " & intStatus

    If intStatus <> 0 Then
        Call errorMessage(intStatus)
        End
    End If

    openDevice = intLogicalDevice

    Exit Function

errUnknown:
    Call unknownErrorMessage
    End

End Function
```

```
'*********************************************************************
' Function: singleDigitalInput
' Purpose: Input a single value from a single input channel
'          of the PCMDIO digital I/O card
' Inputs:
'   LogicalDevice:  the Logical Device number
'   Channel: The channel variable
'   InputValue: pointer to the variable that will contain the
'               input value
'Returns: the status of the input request
'*********************************************************************


Public Function singleDigitalInput(ByVal LogicalDevice As Integer, _
                                   ByVal Channel As Integer, _
                                   InputValue As Byte) As Integer
    Dim intStatus As Integer
    Dim intRequestHandle As Integer
    Dim udtDigioRequest As DigioRequest
    Dim udtDataBuffer As PCMDriveBuffer
    Dim udtAllocateRequest As allocate_request

    Dim lngRetChannelAdd As Long
    Dim lngRetBufferAdd As Long

    Dim blnCompleteStatus As Boolean
    Dim lngEventMask As Long
    Dim errorcode As Integer

    On Error GoTo errUnknown

    intRequestHandle = 0
    blnCompleteStatus = False

    '-------------------------------------------------------------------
    'Allocate and lock memory for the Digital Input
    '-------------------------------------------------------------------

    With udtAllocateRequest
        .request_type = DIGIN_TYPE_REQUEST
        .channel_array_length = 1
        .number_of_buffers = 1
        .buffer_size = 1
        .buffer_attributes = RING_BUFFER
    End With

    intStatus = PCMAllocateRequestVB(LogicalDevice, udtAllocateRequest)

    If intStatus <> 0 Then
        singleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Allocate Request Status = " & intStatus
```

```
    '-------------------------------------------------------------------
    'Prepare the Digital Input Request Structure
    '-------------------------------------------------------------------

    lngRetChannelAdd = PCMGetAddressOfVB(Channel)
    lngRetBufferAdd = PCMGetAddressOfVB(udtDataBuffer)

    With udtDigioRequest
        .ChannelArrayPtr = lngRetChannelAdd
        .ArrayLength = 1
        .DigioBufferptr = lngRetBufferAdd
        .NumberOfScans = 1
        .IOMode = ForegroundCPU
        .TriggerSource = InternalTrigger
        .ScanEventLevel = 0
        .RequestStatus = NoEvents
    End With

    '-------------------------------------------------------------------
    'Send a digital input request to the PCMDrive
    '-------------------------------------------------------------------

    intStatus = PCMDigitalInputVB(LogicalDevice, udtDigioRequest,
intRequestHandle)

    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Handle request status = " & intStatus
    'Debug.Print "Request Handle = " & intRequestHandle


    '-------------------------------------------------------------------
    'Arm the request
    '-------------------------------------------------------------------

    intStatus = PCMArmRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Arm Request Status = " & intStatus
```

```
    '-------------------------------------------------------------------
    'Trigger the Request
    '-------------------------------------------------------------------

    intStatus = PCMTriggerRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Trigger Request Status = " & intStatus

    '-------------------------------------------------------------------
    'Wait for completion or error
    '-------------------------------------------------------------------

    lngEventMask = CompleteEvent Or RuntimeErrorEvent
    While (blnCompleteStatus = False)
        If (udtDigioRequest.RequestStatus And lngEventMask) <> 0 Then
            If (udtDigioRequest.RequestStatus And CompleteEvent) <> 0
Then
                intStatus = PCMReadBufferVB(intRequestHandle, 0,
InputValue)
                If intStatus <> 0 Then
                    Call PCMReleaseRequestVB(intRequestHandle)
                    Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
                    singleDigitalInput = intStatus
                    Exit Function
                End If
                blnCompleteStatus = True
            End If
        Else
            intStatus = PCMGetRuntimeErrorVB(intRequestHandle,
errorcode)
            MsgBox "Run Time Error." & vbCrLf & "Error code " &
errorcode, vbOKOnly, "PCMDIO Run Time Error"
            Call PCMReleaseRequestVB(intRequestHandle)
            Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
            Call PCMCloseDeviceVB(LogicalDevice)
            End
        End If
    Wend

    '-------------------------------------------------------------------
    'Release the Request if operation is completed
    '-------------------------------------------------------------------

    intStatus = PCMReleaseRequestVB(intRequestHandle)
```

```
    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Release Request Status = " & intStatus

    singleDigitalInput = intStatus

    Exit Function

errUnknown:
    Call unknownErrorMessage
    Call PCMCloseDeviceVB(LogicalDevice)
    End

End Function


'*********************************************************************
' Function: multipleDigitalInput
' Purpose: Input a single value from multiple input channels
'          of the PCMDIO digital I/O card
' Inputs:
'   LogicalDevice:  the Logical Device number
'   Channel: pointer to the channel array
'   ArrayLength:  the length of the channel array
'   InputValue: pointer to the array that will contain the
'               input values
'Returns: the status of the input request
'*********************************************************************


Public Function multipleDigitalInput(ByVal LogicalDevice As Integer, _
                                     Channel() As Integer, _
                                     ByVal ArrayLength As Integer, _
                                     InputValue() As Byte) As Integer
    Dim intStatus As Integer
    Dim intRequestHandle As Integer
    Dim udtDigioRequest As DigioRequest
    Dim udtDataBuffer As PCMDriveBuffer
    Dim udtAllocateRequest As allocate_request

    Dim lngRetChannelAdd As Long
    Dim lngRetBufferAdd As Long

    Dim blnCompleteStatus As Boolean
    Dim lngEventMask As Long
    Dim errorcode As Integer

    On Error GoTo errUnknown
```

```
    intRequestHandle = 0
    blnCompleteStatus = False

    '---------------------------------------------------------------------
    'Allocate and lock memory for the Digital Input
    '---------------------------------------------------------------------

    With udtAllocateRequest
        .request_type = DIGIN_TYPE_REQUEST
        .channel_array_length = ArrayLength
        .number_of_buffers = 1
        .buffer_size = ArrayLength
        .buffer_attributes = RING_BUFFER
    End With

    intStatus = PCMAllocateRequestVB(LogicalDevice, udtAllocateRequest)

    If intStatus <> 0 Then
        multipleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Allocate Request Status = " & intStatus

    '---------------------------------------------------------------------
    'Prepare the Digital Input Request Structure
    '---------------------------------------------------------------------

    lngRetChannelAdd = PCMGetAddressOfVB(Channel(0))
    lngRetBufferAdd = PCMGetAddressOfVB(udtDataBuffer)

    With udtDigioRequest
        .ChannelArrayPtr = lngRetChannelAdd
        .ArrayLength = ArrayLength
        .DigioBufferptr = lngRetBufferAdd
        .NumberOfScans = 1
        .IOMode = ForegroundCPU
        .TriggerSource = InternalTrigger
        .ScanEventLevel = 0
        .RequestStatus = NoEvents
    End With

    '---------------------------------------------------------------------
    'Send a digital input request to the PCMDrive
    '---------------------------------------------------------------------

    intStatus = PCMDigitalInputVB(LogicalDevice, udtDigioRequest,
intRequestHandle)

    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalInput = intStatus
```

```
        Exit Function
    End If

    'Debug.Print "Handle request status = " & intStatus
    'Debug.Print "Request Handle = " & intRequestHandle

    '-------------------------------------------------------------------
    'Arm the request
    '-------------------------------------------------------------------

    intStatus = PCMArmRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Arm Request Status = " & intStatus

    '-------------------------------------------------------------------
    'Trigger the Request
    '-------------------------------------------------------------------

    intStatus = PCMTriggerRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Trigger Request Status = " & intStatus

    '-------------------------------------------------------------------
    'Wait for completion or error
    '-------------------------------------------------------------------

    lngEventMask = CompleteEvent Or RuntimeErrorEvent
    While (blnCompleteStatus = False)
        If (udtDigioRequest.RequestStatus And lngEventMask) <> 0 Then
            If (udtDigioRequest.RequestStatus And CompleteEvent) <> 0
Then
                intStatus = PCMReadBufferVB(intRequestHandle, 0,
InputValue(0))
                If intStatus <> 0 Then
                    Call PCMReleaseRequestVB(intRequestHandle)
                    Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
                    multipleDigitalInput = intStatus
```

```
                    Exit Function
                End If
                blnCompleteStatus = True
            End If
        Else
            intStatus = PCMGetRuntimeErrorVB(intRequestHandle,
errorcode)
            MsgBox "Run Time Error." & vbCrLf & "Error code " &
errorcode, vbOKOnly, "PCMDIO Run Time Error"
            Call PCMReleaseRequestVB(intRequestHandle)
            Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
            Call PCMCloseDeviceVB(LogicalDevice)
            End
        End If
    Wend

    '---------------------------------------------------------------------
    'Release the Request if operation is completed
    '---------------------------------------------------------------------

    intStatus = PCMReleaseRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalInput = intStatus
        Exit Function
    End If

    'Debug.Print "Release Request Status = " & intStatus

    multipleDigitalInput = intStatus

    Exit Function

errUnknown:
    Call unknownErrorMessage
    Call PCMCloseDeviceVB(LogicalDevice)
    End

End Function

'*********************************************************************
' Function: singleDigitalOutput
' Purpose: Output a single value to an output channel
'          of the PCMDIO digital I/O card
' Inputs:
'   LogicalDevice:  the Logical Device number
'   Channel: The channel variable
'   OutputValue: pointer to the variable that contains the output
'                value
'Returns: the status of the output request
'*********************************************************************
```

```
Public Function singleDigitalOutput(ByVal LogicalDevice As Integer, _
                                    ByVal Channel As Integer, _
                                    ByVal OutputValue As Byte) As
Integer
    Dim intStatus As Integer
    Dim intRequestHandle As Integer
    Dim udtDigioRequest As DigioRequest
    Dim udtDataBuffer As PCMDriveBuffer
    Dim udtAllocateRequest As allocate_request

    Dim lngRetChannelAdd As Long
    Dim lngRetBufferAdd As Long

    Dim lngEventMask As Long
    Dim errorcode As Integer

    On Error GoTo errUnknown

    intRequestHandle = 0

    '---------------------------------------------------------------------
    'Allocate and lock memory for the Digital Input
    '---------------------------------------------------------------------

    With udtAllocateRequest
        .request_type = DIGOUT_TYPE_REQUEST
        .channel_array_length = 1
        .number_of_buffers = 1
        .buffer_size = 1
        .buffer_attributes = RING_BUFFER
    End With

    intStatus = PCMAllocateRequestVB(LogicalDevice, udtAllocateRequest)

    If intStatus <> 0 Then
        singleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Allocate Request Status = " & intStatus

    '---------------------------------------------------------------------
    'Prepare the Digital Output Request Structure
    '---------------------------------------------------------------------

    lngRetChannelAdd = PCMGetAddressOfVB(Channel)
    lngRetBufferAdd = PCMGetAddressOfVB(udtDataBuffer)

    With udtDigioRequest
        .ChannelArrayPtr = lngRetChannelAdd
        .ArrayLength = 1
        .DigioBufferptr = lngRetBufferAdd
        .NumberOfScans = 1
```

```
        .IOMode = ForegroundCPU
        .TriggerSource = InternalTrigger
        .ScanEventLevel = 0
        .RequestStatus = NoEvents
    End With

    '-------------------------------------------------------------------
    'Send a digital output request to the PCMDrive
    '-------------------------------------------------------------------

    intStatus = PCMDigitalOutputVB(LogicalDevice, udtDigioRequest,
intRequestHandle)

    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Handle request status = " & intStatus
    'Debug.print "Request Handle = " & intRequestHandle

    '-------------------------------------------------------------------
    'Copy output data to the PCM drive allocated buffer
    '-------------------------------------------------------------------

    intStatus = PCMWriteBufferVB(intRequestHandle, 0, 1, OutputValue)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalOutput = intStatus
        Exit Function
    End If

    '-------------------------------------------------------------------
    'Arm the request
    '-------------------------------------------------------------------

    intStatus = PCMArmRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Arm Request Status = " & intStatus
```

```
    '-------------------------------------------------------------------
    'Trigger the Request
    '-------------------------------------------------------------------

    intStatus = PCMTriggerRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Trigger Request Status = " & intStatus

    '-------------------------------------------------------------------
    'Wait for completion or error
    '-------------------------------------------------------------------

    lngEventMask = CompleteEvent Or RuntimeErrorEvent
    While (udtDigioRequest.RequestStatus And lngEventMask) = 0
        DoEvents
    Wend

    If (udtDigioRequest.RequestStatus And RuntimeErrorEvent) <> 0 Then
        intStatus = PCMGetRuntimeErrorVB(intRequestHandle, errorcode)
        MsgBox "Run Time Error." & vbCrLf & "Error code " & errorcode,
vbOKOnly, "PCMDIO Run Time Error"
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        Call PCMCloseDeviceVB(LogicalDevice)
        End
    End If

    '-------------------------------------------------------------------
    'Release the Request if operation is completed
    '-------------------------------------------------------------------

    intStatus = PCMReleaseRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        singleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Release Request Status = " & intStatus

    singleDigitalOutput = intStatus

    Exit Function
```

```
errUnknown:
    Call unknownErrorMessage
    Call PCMCloseDeviceVB(LogicalDevice)
    End

End Function


'*********************************************************************
' Function: multipleDigitalOutput
' Purpose: Output a single value to multiple output channels
'          of the PCMDIO digital I/O card
' Inputs:
'   LogicalDevice:  the Logical Device number
'   Channel: pointer to the channel array
'   ArrayLength:  the length of the channel array
'   OutputValue: pointer to the array that contains the output
'                values
'Returns: the status of the output request
'*********************************************************************


Public Function multipleDigitalOutput(ByVal LogicalDevice As Integer, _
                                       Channel() As Integer, _
                                       ByVal ArrayLength As Integer, _
                                       OutputValue() As Byte) As Integer
    Dim intStatus As Integer
    Dim intRequestHandle As Integer
    Dim udtDigioRequest As DigioRequest
    Dim udtDataBuffer As PCMDriveBuffer
    Dim udtAllocateRequest As allocate_request

    Dim lngRetChannelAdd As Long
    Dim lngRetBufferAdd As Long

    Dim lngEventMask As Long
    Dim errorcode As Integer

    On Error GoTo errUnknown

    intRequestHandle = 0

    '-------------------------------------------------------------------
    'Allocate and lock memory for the Digital Input
    '-------------------------------------------------------------------

    With udtAllocateRequest
        .request_type = DIGOUT_TYPE_REQUEST
        .channel_array_length = ArrayLength
        .number_of_buffers = 1
        .buffer_size = 2 * ArrayLength
        .buffer_attributes = RING_BUFFER
    End With
```

```
    intStatus = PCMAllocateRequestVB(LogicalDevice, udtAllocateRequest)

    If intStatus <> 0 Then
        multipleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Allocate Request Status = " & intStatus

    '-------------------------------------------------------------------
    'Prepare the Digital Output Request Structure
    '-------------------------------------------------------------------

    lngRetChannelAdd = PCMGetAddressOfVB(Channel(0))
    lngRetBufferAdd = PCMGetAddressOfVB(udtDataBuffer)

    With udtDigioRequest
        .ChannelArrayPtr = lngRetChannelAdd
        .ArrayLength = ArrayLength
        .DigioBufferptr = lngRetBufferAdd
        .NumberOfScans = 1
        .IOMode = ForegroundCPU
        .TriggerSource = InternalTrigger
        .ScanEventLevel = 0
        .RequestStatus = NoEvents
    End With

    '-------------------------------------------------------------------
    'Send a digital output request to the PCMDrive
    '-------------------------------------------------------------------

    intStatus = PCMDigitalOutputVB(LogicalDevice, udtDigioRequest,
intRequestHandle)

    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Handle request status = " & intStatus
    'Debug.print "Request Handle = " & intRequestHandle

    '-------------------------------------------------------------------
    'Copy output data to the PCM drive allocated buffer
    '-------------------------------------------------------------------

    intStatus = PCMWriteBufferVB(intRequestHandle, 0, 1,
OutputValue(0))

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
```

```
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalOutput = intStatus
        Exit Function
    End If

    '-------------------------------------------------------------------
    'Arm the request
    '-------------------------------------------------------------------

    intStatus = PCMArmRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Arm Request Status = " & intStatus

    '-------------------------------------------------------------------
    'Trigger the Request
    '-------------------------------------------------------------------

    intStatus = PCMTriggerRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMReleaseRequestVB(intRequestHandle)
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Trigger Request Status = " & intStatus


    '-------------------------------------------------------------------
    'Wait for completion or error
    '-------------------------------------------------------------------

    lngEventMask = CompleteEvent Or RuntimeErrorEvent
    While (udtDigioRequest.RequestStatus And lngEventMask) = 0
        DoEvents
    Wend

    If (udtDigioRequest.RequestStatus And RuntimeErrorEvent) <> 0 Then
        intStatus = PCMGetRuntimeErrorVB(intRequestHandle, errorcode)
        MsgBox "Run Time Error." & vbCrLf & "Error code " & errorcode,
vbOKOnly, "PCMDIO Run Time Error"
        Call PCMReleaseRequestVB(intRequestHandle)
```

```
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        Call PCMCloseDeviceVB(LogicalDevice)
        End
    End If


    '-----------------------------------------------------------------
    'Release the Request if operation is completed
    '-----------------------------------------------------------------

    intStatus = PCMReleaseRequestVB(intRequestHandle)

    If intStatus <> 0 Then
        Call PCMFreeRequestVB(LogicalDevice,
udtAllocateRequest.memory_handle, udtAllocateRequest.memory_pointer)
        multipleDigitalOutput = intStatus
        Exit Function
    End If

    'Debug.print "Release Request Status = " & intStatus

    multipleDigitalOutput = intStatus

    Exit Function

errUnknown:
    Call unknownErrorMessage
    Call PCMCloseDeviceVB(LogicalDevice)
    End

End Function


'*****************************************************************
' Procedure: waitTime
' Purpose: Waits for a time specified in milliseconds
' Inputs:
'   MilliSec: wait time in milliseconds
'*****************************************************************


Public Sub waitTime(ByVal MilliSec As Long)
    Dim oldTime As Long

    oldTime = Timer() * 1000
    Do
        DoEvents
    Loop Until ((Timer() * 1000 - oldTime) >= MilliSec)

End Sub
```

```
'*************************************************************
' Procedure: unknownErrorMessage
' Purpose: In the case of a non PCMDIO error the procedure
'          displays a message box that describes the nature
'          of the error
'*************************************************************


Public Sub unknownErrorMessage()
    Dim errmsg As String

    errmsg = "Error # " & Str(Err.Number) & " was generated by " _
            & Err.Source & vbCrLf & vbCrLf & Err.Description
    MsgBox errmsg, vbOKOnly + vbExclamation, "PCMDIO Error"

End Sub


'*************************************************************
'Procedure: errorMessage
'Purpose: In the case of a PCMDIO error the procedure displays
'         a message box that describes the nature of the error
'*************************************************************


Public Sub errorMessage(ByVal errorcode As Integer)
    Dim errmsg As String
    Dim errDesc As String

    Select Case errorcode
        Case 10
            errmsg = "Error opening configuration file."
        Case 11
            errmsg = "File is not a valid PCMDRIVE configuration file."
        Case 12
            errmsg = "Configuration file invalid for specified adapter
type."
        Case 13
            errmsg = "Error reading configuration file."
        Case 14
            errmsg = "End-Of-File encountered reading configuration
file."
        Case 15
            errmsg = "Invalid configuration file version."
        Case 18
            errmsg = "Unable to write device parameters to registry."
        Case 30
            errmsg = "Error loading DLL."
        Case 31
            errmsg = "Cannot locate the PCMDRIVE DLL open command."
        Case 35
            errmsg = "Cannot locate the PCMDRIVE TSR driver."
        Case 39
            errmsg = "PCMDRIVE is out of date."
```

```
        Case 50
            errmsg = "Auto-configuration support not available."
        Case 51
            errmsg = "Invalid device type."
        Case 52
            errmsg = "Auto Configuration Failure."
        Case 60
            errmsg = "Configuration file error."
        Case 70
            errmsg = "Configuration file error."
        Case 71
            errmsg = "Configuration file error."
        Case 72
            errmsg = "Configuration file error."
        Case 73
            errmsg = "Configuration file error."
        Case 74
            errmsg = "Configuration file error."
        Case 100
            errmsg = "Invalid logical device number."
        Case 120
            errmsg = "No logical devices defined."
        Case 150
            errmsg = "Invalid request handle."
        Case 200
            errmsg = "No interrupt level defined for adapter."
        Case 201
            errmsg = "Interrupt in-use by another device."
        Case 205
            errmsg = "Internal interrupt error."
        Case 250
            errmsg = "No DMA channedl defined for adapter."
        Case 251
            errmsg = "DMA channel in-use by another device."
        Case 255
            errmsg = "Internal DMA error."
        Case 300
            errmsg = "Memory allocation error."
        Case 310
            errmsg = "Memory release error."
        Case 400
            errmsg = "Channel in-use by another request."
        Case 410
            errmsg = "Timer in-use by another request."
        Case 450
            errmsg = "Hardware dependent resource in-use by another
device."
        Case 500
            errmsg = "Invalid procedure call for a request that is not
configured."
        Case 600
            errmsg = "Invalid procedure call for a request that is not
armed."
        Case 650
```

```
            errmsg = "Invalid procedure call for a request that is
armed."
        Case 700
            errmsg = "Trigger command invalid with specified trigger
source."
        Case 800
            errmsg = "Invalid re-configuration request."
        Case 1000
            errmsg = "Requested function not supported by targer
hardware."
        Case 1050
            errmsg = "Invalid operation in multi-user mode."
        Case 1100
            errmsg = "Invalid channel number"
        Case 1101
            errmsg = "Invalid array length."
        Case 1150
            errmsg = "Duplicate entries in channel list."
        Case 1160
            errmsg = "Invalid channel sequence."
        Case 1280
            errmsg = "Invalid gain."
        Case 1300
            errmsg = "Invalid data buffer length."
        Case 1320
            errmsg = "Invalid output value."
        Case 1350
            errmsg = "DMA mode data buffer crosses page boundary."
        Case 1351
            errmsg = "DMA mode data buffer defined on odd address."
        Case 1352
            errmsg = "Internal DMA error."
        Case 1400
            errmsg = "Invalid trigger source."
        Case 1401
            errmsg = "Trigger source not supported."
        Case 1410
            errmsg = "Invalid trigger slope."
        Case 1411
            errmsg = "Trigger slope not supported."
        Case 1420
            errmsg = "Invalid trigger channel."
        Case 1430
            errmsg = "Invalid analog trigger voltage."
        Case 1500
            errmsg = "Invalid data transfer mode."
        Case 1600
            errmsg = "Invalid clock source."
        Case 1601
            errmsg = "Clock source not supported."
        Case 1650
            errmsg = "Invalid sampling rate."
        Case 1700
            errmsg = "Invalid calibration mode."
```

```
        Case 1710
            errmsg = "Adapter does not support auto-calibration."
        Case 1720
            errmsg = "Adapter does not support auto-zero."
        Case 3000
            errmsg = "Hardware failure."
        Case 3010
            errmsg = "A/D converter failure."
        Case 3020
            errmsg = "D/A converter failure."
        Case 3030
            errmsg = "Digital I/O failure."
        Case 3040
            errmsg = "Counter/timer failure."
        Case 5000
            errmsg = "Buffer overrun."
        Case 5010
            errmsg = "Buffer under-run."
        Case 5100
            errmsg = "FIFO overrun."
        Case 5110
            errmsg = "FIFO under-run."
        Case 5200
            errmsg = "Request time-out."
        Case 5300
            errmsg = "User break."
        Case Else
            errmsg = "Unknown Error."
        End Select

        errmsg = "Error code " & errorcode & vbCrLf & vbCrLf & errmsg
        MsgBox errmsg, vbOKOnly + vbExclamation, "PCMDIO Error"

End Sub
```

4.PCMDrvFunc.bas

```
'PCMdrive operation
Declare Function PCMOpenDeviceVB Lib "PCMDrvVB.DLL" ( _
        ByVal dllname As String, _
        logical_device As Integer, _
        ByVal DeviceType As String, _
        ByVal ConfigFile As String) As Integer

Declare Function PCMResetDeviceVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer) As Integer

Declare Function PCMCloseDeviceVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer) As Integer

Declare Function PCMGetAddressOfVB Lib "PCMDrvVB.DLL" ( _
        pointer As Any) As Long

Declare Function PCMAllocateRequestVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        pAllocReqs As allocate_request) As Integer

Declare Function PCMFreeRequestVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        ByVal memory_handle As Long, _
        ByVal memory_ptr As Long) As Integer

Declare Function PCMReadBufferVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer, _
        ByVal curBuf As Integer, _
        databuf As Any) As Integer

Declare Function PCMWriteBufferVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer, _
        ByVal curBuf As Integer, _
        ByVal cycles As Long, _
        databuf As Any) As Integer

Declare Function PCMReadBufferFlagVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer, _
        ByVal curBuf As Integer, _
        pBufferStatus As Integer) As Integer

Declare Function PCMWriteBufferFlagVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer, _
        ByVal curBuf As Integer, _
        ByVal BufferStatus As Integer) As Integer

'I/O operation
Declare Function PCMDigitalInputVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        ByRef Request As DigioRequest, _
        ByRef handle As Integer) As Integer
```

```
Declare Function PCMDigitalOutputVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        Request As DigioRequest, _
        handle As Integer) As Integer

Declare Function PCMSingleDigitalInputVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        ByVal Channel As Integer, _
        Value As Any) As Integer

Declare Function PCMSingleDigitalInputScanVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        Channel As Integer, _
        ByVal Length As Integer, _
        Value As Any) As Integer

Declare Function PCMSingleDigitalOutputVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        ByVal Channel As Integer, _
        Value As Any) As Integer

Declare Function PCMSingleDigitalOutputScanVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        Channel As Integer, _
        ByVal Length As Integer, _
        Value As Any) As Integer

Declare Function PCMArmRequestVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer) As Integer

Declare Function PCMReleaseRequestVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer) As Integer

Declare Function PCMStopRequestVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer) As Integer

Declare Function PCMTriggerRequestVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer) As Integer

Declare Function PCMGetRuntimeErrorVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer, _
        errorcode As Integer) As Integer

Declare Function PCMNotifyEventVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer, _
        ByVal NotifyProc As Long, _
        ByVal EventMask As Long) As Integer

Declare Function PCMUserBreakVB Lib "PCMDrvVB.DLL" ( _
        ByVal request_handle As Integer, _
        BreakProc As Any)


Declare Function PCMGetDeviceCfgInfoVB Lib "PCMDrvVB.DLL" ( _
```

```
        ByVal logical_device As Integer, _
        CfgInfo As DeviceConfiguration) As Integer

Declare Function PCMGetDigioCfgInfoVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        ByVal Digio As Integer, _
        CfgInfo As DigioConfiguration) As Integer

Declare Function PCMVersionNumberVB Lib "PCMDrvVB.DLL" ( _
        ByVal logical_device As Integer, _
        PCMVersion As Single, _
        SoftVersion As Single, _
        FirmVersion As Single) As Integer


Declare Sub PCMWordsToBytesVB Lib "PCMDrvVB.DLL" ( _
        Words As Integer, _
        Bytes As Integer, _
        ByVal Length As Long)

Declare Sub PCMBytesToWordsVB Lib "PCMDrvVB.DLL" ( _
        Bytes As Integer, _
        Words As Integer, _
        ByVal Length As Long)
```

5. PCMData.bas

```
'Attribute VB_Name = "PCMData"


'************************************************************************
'*  PCMDRIVE data buffer structure
'************************************************************************


Option Base 0

Type PCMDriveBuffer
   BufferStatus         As Integer       ' current buffer status
   DataBufferptr        As Long          ' offset of data points
   BufferLength         As Long          ' data_buffer length in
                                         '   number of points
   BufferCycles         As Long          ' number of cycles of this
                                         '   buffer (output only)
   NextStructureptr     As Long          ' address of next structure
End Type


'************************************************************************
'*  Digital I/O request structure
'************************************************************************


Type DigioRequest
   ChannelArrayPtr        As Long        ' address of channel scan list
   ReservedArray0(3)      As Integer     ' reserved for future
                                         ' expansion
   ArrayLength            As Integer     ' length of chan & gain arrays
   DigioBufferptr         As Long        ' address of PCMDRIVE_buffer
   ReservedArray1(3)      As Integer     ' reserved for future
                                         ' expansion
   TriggerSource          As Integer     ' trigger source
   TriggerMode            As Integer     ' continuous / one-shot
                                         ' trigger
   TriggerSlope           As Integer     ' rising / falling edge
                                         ' trigger
   TriggerChannel         As Integer     ' trigger channel number
                                         ' analog or digital trigger)
   TriggerVoltage         As Double      ' trigger voltage (analog
                                         ' trigger)
   TriggerValue           As Long        ' value for trigger (digital
                                         ' trigger)
   ReservedArray2(3)      As Integer     ' reserved for future
                                         ' expansion
   IOMode                 As Integer     ' input mode
                                         '    = 0 poll
                                         '    = 1 IRQ
                                         '    = 2 DMA with CPU status
                                         '    = 3 DMA with IRQ status
```

```
   ClockSource            As Integer     ' clock source (0 = internal)
   ClockRate              As Double      ' clock rate (if not internal)
   SampleRate             As Double      ' input sampling rate (Hz)
   ReservedArray3(3)      As Integer     ' reserved for future
                                         ' expansion
   NumberOfScans          As Long        ' number of channel scans
   ScanEventLevel         As Long        ' generate event each
                                         ' scan_event_level
                                         '    scans ( 0 = disable )
   ReservedArray4(7)      As Integer     ' reserved for future
                                         ' expansion
   TimeoutInterval        As Integer     ' timeout interval (in sec)
   RequestStatus          As Long        ' request event status
End Type


'*************************************************************************
'*  Allocate Request Structure
'*************************************************************************


Type allocate_request
    request_type            As Long
    channel_array_length    As Integer
    number_of_buffers       As Integer
    buffer_size             As Long
    buffer_attributes       As Long
    memory_pointer          As Long
    memory_handle           As Long
End Type


'*************************************************************************
'*  Device configuration structure
'*************************************************************************


Type DeviceConfiguration
   BaseAddress            As Integer     ' base I/O address
   IRQ                    As Integer     ' IRQ level     (-1 = none)
   DMA1                   As Integer     ' primary DMA   (-1 = none)
   DMA2                   As Integer     ' secondary DMA (-1 = none)
   ADCDevices             As Integer     ' number of A/D devices
   DACDevices             As Integer     ' number of D/A devices
   DigioDevices           As Integer     ' number of digital I/O chans
   TimerDevices           As Integer     ' number of C/T channels
End Type


'*************************************************************************
'*  Digital I/O configuration structure
'*************************************************************************
```

```
Type DigioConfiguration
   DataSize               As Integer     ' digital I/O width (in bits)
   IOMode                 As Integer     ' input / output mode
End Type


'====================================================================
'*  Counter / timer configuration structure
'====================================================================


Type TimerConfiguration
   DataSize               As Integer     ' timer data width (in bits)
   InternalClockRate      As Double      ' rate of internal clock (Hz)
   MinRate                As Double      ' minimum output rate
   MaxRate                As Double      ' maximum output rate
End Type


'====================================================================
'*  Define channel types
'====================================================================


Global Const ADC_TYPE_REQUEST = &H0
Global Const DAC_TYPE_REQUEST = &H1
Global Const DIGIN_TYPE_REQUEST = &H2
Global Const DIGOUT_TYPE_REQUEST = &H3


'====================================================================
'*  Define flags
'====================================================================


Global Const SEQUENTIAL_BUFFER = &H0
Global Const RING_BUFFER = &H1


'====================================================================
'*  Define buffer status constants
'====================================================================


Global Const BufferFull = &H1             ' data buffer full status
Global Const BufferEmpty = &H2            ' data buffer empty status



'====================================================================
'*  Define calibration constants
'====================================================================
```

```
Global Const NoCalibration = &H0            ' disable calibration
Global Const AutoCalibrate = &H1            ' enable auto-calibration
Global Const AutoZero = &H2                 ' enable auto-zero


'====================================================================
'*  Define trigger constants
'====================================================================


Global Const InternalTrigger = 0        ' selects internal trigger
Global Const TTLTrigger = 1             ' selects TTL trigger
Global Const AnalogTrigger = 2          ' selects analog trigger
Global Const DigitalTrigger = 3         ' selects digital value trigger

Global Const ContinuousTrigger = 0      ' selects cont. trigger mode
Global Const OneShotTrigger = 1         ' selects one-shot trigger mode

Global Const RisingEdge = 0             ' selects rising edge trigger
Global Const FallingEdge = 1            ' selects falling edge trigger


'====================================================================
'*  Define acquisition modes
'====================================================================


Global Const ForegroundCPU = 0         ' perform task using CPU polling
Global Const BackgroundIRQ = 1         ' perform task using interrupts
Global Const ForegroundDMA = 2         ' perform task using DMA
                                       ' transfers
                                       ' CPU monitors DMA (foreground)
Global Const BackgroundDMA = 3         ' perform task using DMA
transfers
                                       ' IRQ monitors DMA (background)

'====================================================================
'  Define timer clock sources
'====================================================================

Global Const InternalClock = 0         ' selects internal clock source
Global Const ExternalClock = 1         ' selects external clock source


'====================================================================
'*  Define event mask / status bits
'====================================================================


Global Const NoEvents = &H0
Global Const TriggerEvent = &H1
Global Const CompleteEvent = &H2
Global Const BufferEmptyEvent = &H4
Global Const BufferFullEvent = &H8
```

```
Global Const ScanEvent = &H10
Global Const UserBreakEvent = &H20000000
Global Const TimeoutEvent = &H40000000
Global Const RuntimeErrorEvent = &H80000000


'*********************************************************************
'*  Define event types
'*********************************************************************


Global Const EventTypeTrigger = 0
Global Const EventTypeComplete = 1
Global Const EventTypeBufferEmpty = 2
Global Const EventTypeBufferFull = 3
Global Const EventTypeScan = 4
Global Const EventTypeUserBreak = 29
Global Const EventTypeTimeout = 30
Global Const EventTypeRuntimeError = 31
```

APPENDIX B

CLIENT-SIDE PROGRAM

The client-side program is a Visual Basic project consisting of the following forms and modules.

1. modIprvClientMain.bas - Execution of the server-side program starts with this module. All the other forms in this project are executed within this module.
2. IprvClientMain.frm – This form contains the client-side GUI described in Section 5.5.

The client-side program also has three other forms. One is used as a splash screen, the second is used for password protection, and the last is used to input control keys that the operator may prefer to use for IPRV speed and direction control instead of the control buttons provided on the GUI.

1. ModIprvClientMain.bas

```
Option Explicit

Public gintKeyFwd As Integer
Public gintKeyRev As Integer
Public gintKeyLeft As Integer
Public gintKeyRight As Integer

Global Const MIN_FWD_REV_VALUE = 51
Global Const MAX_FWD_REV_VALUE = 195
Global Const MIN_LEFT_RIGHT_VALUE = 40
Global Const MAX_LEFT_RIGHT_VALUE = 215



Private Sub main()

    Dim dblTimeDelay As Double
    Dim dblCounter As Double

    Dim frmLogin1 As frmLogin
    Dim frmDirection1 As frmDirection
    Dim frmMain1  As frmIprvClientMain

    Load frmSplash
    Call frmSplash.Show

    ' Creating a delay of 1 second
    dblTimeDelay = Timer()
    Do
        dblCounter = Timer() - dblTimeDelay
        DoEvents
    Loop While (dblCounter < 1)

    Set frmLogin1 = New frmLogin
    With frmLogin1
        .Show vbModal
        If .LoginSucceeded = False Then
            Set frmLogin1 = Nothing
            Unload frmSplash
            End
        End If
    End With

    Set frmLogin1 = Nothing
    Unload frmSplash

    Set frmDirection1 = New frmDirection
    With frmDirection1
        .Show vbModal
        If .DirectionSucceeded = False Then
```

```
            Set frmDirection1 = Nothing
            End
        End If
    End With

    Set frmDirection1 = Nothing

    Set frmMain1 = New frmIprvClientMain
    frmMain1.Show vbModal


End Sub
```

2. IPRVClientMain.frm

```
Option Explicit

Public mintFwdRev As Integer
Public mintLeftRight As Integer
Public mintCheckComm As Integer
Public mblnRunModeAuto As Boolean


Private Sub Form_Load()
    Dim i As Integer
    Dim strInfo As String

    mintFwdRev = 128
    mintLeftRight = 128

    For i = 1 To 2
        cmdFwdRev(i).Enabled = False
        cmdLeftRight(i).Enabled = False
    Next i
    cmdHalt.Enabled = False
    cmdConnect.Enabled = True
    cmdDisconnect.Enabled = False
    cmdAuto.Enabled = False
    cmdRemote.Enabled = False

    mblnRunModeAuto = False 'Initialize to start in Remote Maneuvering
Mode

    txtCommands.Text = ""

    strInfo = "Click on the connect button to connect to the IPRV" &
vbCrLf & vbCrLf & _
              "Use the Fwd,Rev,Left,Right buttons to remotely control
the IPRV" & vbCrLf & vbCrLf & _
              "To stop the IPRV at any time press Disconnect."

    lblInfo.Caption = strInfo

End Sub

Private Sub Form_Unload(Cancel As Integer)
    'Ensure that the TCP connection is closed before unloading
    If tcpIprvClient.State <> sckClosed Then
        Call tcpIprvClient.Close
    End If

End Sub

Private Sub cmdConnect_Click()
    'Before connecting, ensure current state of connection is closed
    If tcpIprvClient.State <> sckClosed Then
        Call tcpIprvClient.Close
```

```
    End If
    'Define connection parameters and send a connection request
    tcpIprvClient.RemoteHost = "iprv"
    tcpIprvClient.RemotePort = 10101 'server port

    Call tcpIprvClient.Connect ' connect to remotehost address
    cmdFwdRev(1).TabIndex = 0

End Sub

Private Sub cmdDisconnect_Click()

    'Close the connection
    If tcpIprvClient.State <> sckClosed Then
        Call tcpIprvClient.Close
    End If

    Dim i As Integer
    For i = 1 To 2
        cmdFwdRev(i).Enabled = False
        cmdLeftRight(i).Enabled = False
    Next i
    cmdHalt.Enabled = False
    cmdConnect.Enabled = True
    cmdDisconnect.Enabled = False
    cmdRemote.Enabled = False
    cmdAuto.Enabled = True

    mintFwdRev = 128
    mintLeftRight = 128

    txtCommands.Text = txtCommands.Text & tcpIprvClient.LocalHostName &
" >>> " & "Speed " & mintFwdRev - 128 & vbTab & _
                    "Direction " & mintLeftRight - 128 & vbCrLf &
vbCrLf
    txtCommands.SelStart = Len(txtCommands.Text)
    lblInfo.Caption = "Connection has been closed by the Client" &
vbCrLf & vbCrLf


End Sub

Private Sub tcpIprvClient_Connect()
    Dim i As Integer
    For i = 1 To 2
        cmdFwdRev(i).Enabled = True
        cmdLeftRight(i).Enabled = True
    Next i
    cmdHalt.Enabled = True
    cmdConnect.Enabled = False
    cmdDisconnect.Enabled = True
    cmdAuto.Enabled = True
```

```
    mblnRunModeAuto = False 'On establishing connection, mode should be
Remote Maneuvering

    txtCommands.Text = ""

    ' when connection occurs, display a message
    lblInfo.Caption = "Connected to IP address: " &
tcpIprvClient.RemoteHostIP & vbCrLf & _
                        "Name: " & tcpIprvClient.RemoteHost & vbCrLf &
_
                        "Port #: " & tcpIprvClient.RemotePort & vbCrLf
& vbCrLf
End Sub

Private Sub tcpIprvClient_DataArrival(ByVal bytesTotal As Long)
    Dim strInputData As String
    Dim strMessage As String
    Dim strCommand() As String

    Call tcpIprvClient.GetData(strInputData)
    'Split the received data into commands and values
    strCommand = Split(strInputData, , -1)

    Select Case strCommand(0)
        Case "Fwd"
            'Check the integrity of the received data and assign values
to variables
            If strCommand(2) = "Rev" Then
                If strCommand(1) > MIN_FWD_REV_VALUE And strCommand(1)
< MAX_FWD_REV_VALUE Then
                    If strCommand(3) > MIN_LEFT_RIGHT_VALUE And
strCommand(3) < MAX_LEFT_RIGHT_VALUE Then
                        mintFwdRev = strCommand(1)
                        mintLeftRight = strCommand(3)
                    End If
                End If
            End If
        Case "CheckComm"
            'Return the CheckComm message with the random number
received
            strMessage = "CheckComm " & strCommand(1)
            Call tcpIprvClient.SendData(strMessage)
            Exit Sub
        Case Else   'for completion of code only
    End Select

    txtCommands.Text = txtCommands.Text & tcpIprvClient.LocalHostName &
" >>> " & "Speed " & mintFwdRev - 128 & vbTab & _
                    "Direction " & mintLeftRight - 128 & vbCrLf &
vbCrLf
    txtCommands.SelStart = Len(txtCommands.Text)

End Sub
```

```
Private Sub tcpIprvClient_Close()

    'Close the connection
    Call tcpIprvClient.Close

    Dim i As Integer
    For i = 1 To 2
        cmdFwdRev(i).Enabled = False
        cmdLeftRight(i).Enabled = False
    Next i
    cmdHalt.Enabled = False
    cmdConnect.Enabled = True
    cmdDisconnect.Enabled = False
    cmdRemote.Enabled = False
    cmdAuto.Enabled = True

    mintFwdRev = 128
    mintLeftRight = 128

    txtCommands.Text = txtCommands.Text & tcpIprvClient.LocalHostName &
" >>> " & "Speed " & mintFwdRev - 128 & vbTab & _
                    "Direction " & mintLeftRight - 128 & vbCrLf &
vbCrLf
    txtCommands.SelStart = Len(txtCommands.Text)

    lblInfo.Caption = "Server Closed Connection"

End Sub

Private Sub tcpIprvClient_Error(ByVal Number As Integer, Description As
String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As
String, ByVal HelpContext As Long, CancelDisplay As Boolean)
    MsgBox Source & ": " & Description, vbOKOnly, "TCP/IP Error"
    Unload Me
End Sub

Private Sub cmdFwdRev_Click(Index As Integer)
    Dim strCommand As String

    Select Case Index
        Case 1
            'The Fwd button performs different functions in different
running modes
            'In the Remote Maneuvering mode, Fwd is used to increase
the speed of the IPRV
            'In the Semiautonomous mode, Fwd is used to start the IPRV
            If mblnRunModeAuto = False Then
                If mintFwdRev < MAX_FWD_REV_VALUE Then
                    strCommand = "Forward"
                End If
            Else
                strCommand = "Auto"
                cmdFwdRev(1).Enabled = False
                cmdHalt.Enabled = True
```

```
            End If
        Case 2
            If mintFwdRev > MIN_FWD_REV_VALUE Then
                strCommand = "Reverse"
            End If
    End Select

    Call tcpIprvClient.SendData(strCommand)

End Sub

Private Sub cmdFwdRev_KeyPress(Index As Integer, KeyAscii As Integer)
    Dim strCommand As String
    Select Case KeyAscii
        Case gintKeyFwd
            If mblnRunModeAuto = False Then
                If mintFwdRev < MAX_FWD_REV_VALUE Then
                    strCommand = "Forward"
                End If
            Else
                strCommand = "Auto"
                cmdFwdRev(1).Enabled = False
                cmdHalt.Enabled = True
            End If
            cmdFwdRev(1).TabIndex = 0
        Case gintKeyRev
            If mintFwdRev > MIN_FWD_REV_VALUE Then
                strCommand = "Reverse"
            End If
            cmdFwdRev(2).TabIndex = 0
        Case gintKeyRight
            If mintLeftRight < MAX_LEFT_RIGHT_VALUE Then
                strCommand = "Right"
            End If
            cmdLeftRight(2).TabIndex = 0
        Case gintKeyLeft
            If mintLeftRight > MIN_LEFT_RIGHT_VALUE Then
                strCommand = "Left"
            End If
            cmdLeftRight(1).TabIndex = 0
        Case Else
            Exit Sub
        End Select

    ' send data to server
    Call tcpIprvClient.SendData(strCommand)

End Sub

Private Sub cmdLeftRight_Click(Index As Integer)
    Dim strCommand As String
    Select Case Index
        Case 1
            If mintLeftRight > MIN_LEFT_RIGHT_VALUE Then
```

```
                strCommand = "Left"
            End If
        Case 2
            If mintLeftRight < MAX_LEFT_RIGHT_VALUE Then
                strCommand = "Right"
            End If

        End Select
    ' send data to server
    Call tcpIprvClient.SendData(strCommand)

End Sub

Private Sub cmdLeftRight_KeyPress(Index As Integer, KeyAscii As
Integer)
    Dim strCommand As String
    Select Case KeyAscii
        Case gintKeyFwd
            If mblnRunModeAuto = False Then
                If mintFwdRev < MAX_FWD_REV_VALUE Then
                    strCommand = "Forward"
                End If
            Else
                strCommand = "Auto"
                cmdFwdRev(1).Enabled = False
                cmdHalt.Enabled = True
            End If
            cmdFwdRev(1).TabIndex = 0
        Case gintKeyRev
            If mintFwdRev > MIN_FWD_REV_VALUE Then
                strCommand = "Reverse"
            End If
            cmdFwdRev(2).TabIndex = 0
        Case gintKeyRight
            If mintLeftRight < MAX_LEFT_RIGHT_VALUE Then
                strCommand = "Right"
            End If
            cmdLeftRight(2).TabIndex = 0
        Case gintKeyLeft
            If mintLeftRight > MIN_LEFT_RIGHT_VALUE Then
                strCommand = "Left"
            End If
            cmdLeftRight(1).TabIndex = 0
        Case Else
            Exit Sub
        End Select

    ' send data to server
    Call tcpIprvClient.SendData(strCommand)

End Sub

Private Sub cmdHalt_Click()
```

```
    If mblnRunModeAuto = True Then
        cmdFwdRev(1).Enabled = True
        cmdHalt.Enabled = False
    End If

    ' send data to server
    Call tcpIprvClient.SendData("Halt")

End Sub

Private Sub cmdAuto_Click()
    'On clicking the SemiAutonomous mode only remote start and stop is
possible
    'The IPRV will run automatically in all other aspects

    mblnRunModeAuto = True

    cmdAuto.Enabled = False
    cmdRemote.Enabled = True
    cmdFwdRev(1).Enabled = False
    cmdFwdRev(2).Enabled = False
    cmdLeftRight(1).Enabled = False
    cmdLeftRight(2).Enabled = False
    cmdHalt.Enabled = True

    ' send data to server
    Call tcpIprvClient.SendData("Auto")

End Sub

Private Sub cmdRemote_Click()
    'On clicking the Remote Maneuvering mode the IPRV stops and waits
for commands

    mblnRunModeAuto = False

    cmdAuto.Enabled = True
    cmdRemote.Enabled = False
    cmdFwdRev(1).Enabled = True
    cmdFwdRev(2).Enabled = True
    cmdLeftRight(1).Enabled = True
    cmdLeftRight(2).Enabled = True
    cmdHalt.Enabled = True

    ' send data to server
    Call tcpIprvClient.SendData("Remote")
End Sub
```

VITA

Name: Ruzbeh Adi Minocher Homji

Address: Department of Mechanical Engineering, 3123 TAMU, College Station, TX 77843-3123

Email Address: rhomji@yahoo.com

Education: B.E. Marine Engineering, Marine Engineering & Research Institute, Kolkata, India, 1999
M.E. Mechanical Engineering, Texas A&M University, 2005