Distributing Intelligence in the Wireless Control of a Mobile Robot using a Personal Digital Assistant

by

Madri Ophoff

Distributing Intelligence in the Wireless Control of a Mobile Robot using a Personal Digital Assistant

by

Madri Ophoff

Dissertation

submitted in fulfillment of the requirements for the degree

Magister Technologiae

in

Electrical Engineering

in the

Faculty of Engineering, the Built Environment and Information Technology

of the

Nelson Mandela Metropolitan University

Promoter: Prof. Theo I. van Niekerk

January 2011

Declaration

I, Madri Ophoff, hereby declare that:

- The work in this dissertation is my own work.
- All sources used or referred to have been documented and recognised.
- This dissertation has not previously been submitted in full or partial fulfillment of the requirements for an equivalent or higher qualification at any other recognised educational institution.

Madri Ophoff
January 2011

Abstract

Personal Digital Assistants (PDAs) have recently become a popular component in mobile robots. This compact processing device with its touch screen, variety of built-in features, wireless technologies and affordability can perform various roles within a robotic system. Applications include low-cost prototype development, rapid prototyping, low-cost humanoid robots, robot control, robot vision systems, algorithm development, human-robot interaction, mobile user interfaces as well as wireless robot communication schemes.

Limits on processing power, memory, battery life and screen size impact the usefulness of a PDA in some applications. In addition various implementation strategies exist, each with its own strengths and weaknesses. No comparison of the advantages and disadvantages of the different strategies and resulting architectures exist. This makes it difficult for designers to decide on the best use of a PDA within their mobile robot system.

This dissertation examines and compares the available mobile robot architectures. A thorough literature study identifies robot projects using a PDA and examines how the designs incorporate a PDA and what purpose it fulfils within the system it forms part of. The dissertation categorises the architectures according to the role of the PDA within the robot system.

The hypothesis is made that using a distributed control system architecture makes optimal use of the rich feature set gained from including a PDA in a robot system's design and simultaneously overcomes the device's inherent shortcomings. This architecture is developed into a novel distributed intelligence framework that is supported by a hybrid communications architecture, using two wireless connection schemes.

A prototype implementation illustrates the framework and communications architecture in action. Various performance measurements are taken in a test scenario for an office robot. The results indicate that the proposed framework does deliver performance gains and is a viable alternative for future projects in this area.

Acknowledgements

This work could not have been possible alone. I would like to thank the following people for their valued and much appreciated contribution to this study and time in my life:

- My loving husband and life-long friend, Jacques. For your unending love, motivation and encouragement to see this work through to the end - thank you! For the interest you've shown, discussions, ideas and reading every word I will always be grateful.
- My parents, Japie and Elza Coetzee, for encouragement in all I undertake and especially for carrying me in their prayers.
- Chris and Marie Lues, my "Cape Town-parents", for adopting Jacques and myself and making the move to Cape Town so much easier. Thank you for your friendship and support.
- Morne van der Westhuizen, for being such an understanding boss. And for giving me time off from work to present the paper and to put the finishing touches to this dissertation.
- My promoter, Prof Theo van Niekerk, and the School of Engineering at the Nelson Mandela Metropolitan University for giving me the opportunity to do and complete this work.

> *Beg as loud as you can for good common sense. Search for wisdom as you would search for silver or hidden treasure. Then you will understand what is means to respect and to know the Lord God.*
> Proverbs 2:3-5 cev

To my wonderful husband, Jacques, my partner, who walks every road next to me - I love you.
To my parents, Japie and Elza Coetzee, two of the best people I have the honour of knowing - my role models in this life.
And to my little miracle, I cannot wait to meet you!

Contents

List of Figures

List of Tables

Part I

Background

Chapter 1

Introduction

Mobile robots are robots that have the ability to move around in their environment, other than industrial robots which are usually attached to a fixed surface. The first mobile robots, Elmer and Elsie, created by Mr G. Walter between 1948 and 1949, could follow a light source using a light sensor and were able to move and avoid obstacles in their way (Holland, 2011).

Today the field of mobile robotics is enjoying tremendous scientific, practical and popular success. Mobile robots give museum tours, play soccer (RoboCup), map abandoned coal mines, defuse bombs, drive autonomously through the desert (DARPA Grand Challenge), assist the elderly and even gather data from the surface of Mars (Oates, 2005).

There are a number of methods by which to classify a mobile robot. Mobile robots may be classified by the environment in which the robot moves around in (indoor and outdoor on land, aerial, underwater), the type of locomotion (legged, wheeled, tracked) or their application (entertainment, education, service, exploration).

Mobile robots are described as complex systems due to the large number of competencies needed in order to function. The complexity of an autonomous robot depends primarily on the number of tasks or functions it can perform in response to various stimuli (Miller, 2004). A simple robot may only perform one task, such as stopping when it encounters an object. A more complex robot may be able to stop when it reaches an obstacle and then use various sensors to correct its course and continue. While the manoeuvrability of a robot depends on the mechanical design, the ability to respond to different stimuli depends on the capabilities of the processing system.

Mobile robots are also processing intensive devices (Miller, 2004). The more intelligent a robot is designed to be, the greater the amount of data it will need to store and/or process. For this reason many mobile robot systems incorporate multiple processing elements. Often the main control device is not attached to the actual robot body, but connects wirelessly with an on-board lower level driver unit, accessing the sensors and actuators.

From the release of the first Personal Digital Assistant (PDA), the Psion Organiser by Psion in 1984 featuring an 8-bit Hitachi 6301-family processor, with 4 K of ROM and 2 K of battery-backed RAM and a single-row monochrome LCD screen, PDAs have become much more versatile and popular over the years (Boerner, 2010). Today's traditional PDAs are descendants of the original PalmPilot (Pilot 1000 in March 1996 from Palm Inc.) and Microsoft Handheld PC (by manufacturers like HP, Compaq and Casio in November 1996) which originally used the Windows CE operating system (Palm Inc., 2007a; Tilley, 2001). Palm devices run the Palm Operating System and modern Microsoft Pocket PCs run Windows Mobile. These operating systems are less complex and have fewer instructions to those running on desktop PCs.

The original and main purpose of a PDA is to act as an electronic organiser or day planner that is portable, easy to use and capable of sharing information with your PC. As they developed, many other features and technologies have been integrated into these devices. These include integrated cell phone, Internet and network connectivity through WLAN, short-range wireless connectivity using Infra Red (IR) or Bluetooth technology, acting as a Global Positioning System (GPS) device, running specialised software, memory card expansion slot, FM radio and digital camera.

They can now be described as being miniature versions of typical desktop PC systems. However, space and power consumption constraints have limited the processing power, storage space, and available memory. Even this drawback is set to diminish in the near future, with low power processors being developed for mobile devices such as the ATOM from Intel, set to be comparable to their current Intel Core 2 Duo processor (Nicolo, 2008). Clock speeds of up to 624MHz are possible in current leading models, but PDAs also have some fundamental hardware differences when compared to standard desktop and laptop PCs e.g., they generally do not incorporate a hard drive, but store all basic programs in ROM and any additional programs and data in RAM or Flash memory (PDA Phone Blog, 2010).

Jensen et al. (2005) describe the PDA as a powerful device compared to other robot controllers, such as microcontrollers, thus allowing a control mechanism to integrate high-level planners that implement computationally expensive algorithms.

1.1 Motivation for this Study

Mobile robotics continue to grow in popularity within the research and hobbyist communities. Mobile robotics is itself a hot research topic but they are also valuable research tools for overlapping fields such as intelligent manufacturing, artificial intelligence and human-robot interaction to name but a few (Jensen et al., 2005).

The task of roaming in a dynamically changing environment, safely and accurately, while carrying out meaningful tasks requires a mobile robot to have at least three basic components in its hardware architecture:

1. A hardware platform, or body, that houses all the other robot components.

2. A drive system that allows the robot to move from point A to point B. It usually consists of a combination of motors and either wheels, tracks or legs.

3. Several actuators and sensors that enable the robot to act on its environment as well as gain information from it.

Advances in technology inadvertently provide robot designers with an ever-expanding range of choices to make their robots smaller, more capable and cheaper to produce. One example is how the PDA evolved into what is today a powerful palm-sized processing and user interface (UI) tool. Ever resourceful, robotisists have used this device in unique ways to improve their robot designs. Making them more capable (intelligent), remotely accessible, smaller and lighter, easier to communicate with and more cost-effective.

Deciding to use a PDA within a mobile robot system is not a new idea. Many researchers have incorporated PDAs within their robot designs. It is enticing to think of the possibility of adding so much capability to a robot system by simply adding a PDA. Instantly the robot has a powerful processing device, more available memory, integrated colour screen and built-in wireless technologies such as IR, Bluetooth, WLAN and GPS. And all of this does not take up much valuable platform real estate, as PDAs come in a comparatively small form factor. **This leads to the question of how this addition of a PDA to a mobile robot has been done in the past and how it can be done to the greatest advantage of the robot system.**

In reviewing past projects that also decided to make use of a PDA, it soon became clear that although the robots use the PDA for different purposes within the respective projects, few utilise any of its processing power. Those that do make use of the PDA's processing capabilities, try to implement memory and processing algorithms too expensive for the PDA's available resources. This left researches with incomplete implementations.

In order to design a framework for PDA-integrated robot control, the following must be considered:

1. The physically distributed hardware components of PDA, PC and mobile platform.
2. What control architecture would suite the distributed hardware and how it could be implemented.
3. How these hardware components connect and share information while still supporting the chosen control architecture.

Most robot systems require processing to be distributed to multiple processing devices. This becomes of greater value as the capability of a mobile robot is increased and the overall system complexity also increases, for example the number of sensors, motors and processing components that must be integrated and coordinated (Yasuda, 2003). Therefore, a distributed computing architecture offers a number of advantages to aid in coping with the significant design and implementation complexity inherent in sophisticated mobile robot systems. Multiple processors provide the opportunity to take advantage of parallelism for improved throughput (Hu & Brandy, 1996).

Distributed computing is a type of segmented or parallel computing, but the latter term is most commonly used to refer to processing in which different parts of a program run simultaneously on two or more processors that are part of the same computer. While both types of processing require that a program be segmented, distributed computing also requires that the division of the program take into account the different environments on which the different sections of the program will be running.

Selvatici & Costa (2005) describe a control architecture as a framework for determining the robot actuation and Mataric (1997) explains it as a set of principles for organising control systems, supplying a structure as well as constraints in the way control problems can be solved.

Stasse & Kuniyoshi (2000) state that in the case of an architecture that is implemented as a distributed architecture, it will include the handling of communications. The communication infrastructure establishes how architectural levels interact, how architectural components communicate, how they access distributed data and how behaviours are executed (Posadas et al., 2007).

An improvement in performance cannot be achieved by solely increasing the number of processing units because the time necessary for communication or additional data administration may increase simultaneously (Heinrich & Honiger, 1997).

1.2 Problem Statement

The problem addressed by this study is whether it is a viable option to distribute intelligence in a mobile robot system using a PDA between the PC and microcontroller-based control board, instead of using a PDA as the only interface and processing device or only as a data relay device between the control and PC interfaces.

In order to address this problem successfully a number of sub-problems need to be addressed:

- Can intelligence be distributed in a wireless mobile robot system?
- What are the considerations for distributing software components onto a PDA?
- Will distributing some intelligence from the PC to the PDA provide an increase in system performance?

Effectively answering the above questions would ensure that the following objectives are achieved.

1.3 Research Objectives

The main aim of this study is to determine if distributing intelligence from a PC to a wirelessly connected PDA (as used to wirelessly control a mobile robot) is possible and whether such an implementation improves system performance.

To successfully implement such a distributed system the following subobjectives need to be addressed:

- Identify the advantages and limiting factors associated with using a PDA through a detailed literature review.
- Use the identified limiting factors of a PDA to identify which software control components could possibly be implemented on the PDA.
- Determine whether implementing the identified tasks on the PDA improves system performance, considering response time, battery life and any other quantifiable measures.

These objectives will be achieved through the following methodology.

1.4 Research Methodology

A thorough literature study will form the basis of the project. First, an extensive study of mobile robots will be done. This study will consider control system components and architectures used in distributed robotics and will review how other researchers have implemented distributed intelligence in their projects. Second, a study on PDAs, their development, their past use as wireless processing devices in mobile robotics, the different PDAs used in past projects, current PDA technology and development software available will be done. This includes identifying the advantages and limiting factors of PDAs used as control devices.

Using the knowledge gathered through the literature study a framework will be presented as a possible solution for using a PDA in a flexible and optimum way within a distributed mobile robot system. The design will show which control components can potentially be implemented on a PDA as part of a distributed wireless mobile robot system.

The framework is demonstrated by implementing a prototype PDA-based robot system. The system will comprise three distributed processing elements: a PDA, PC and microcontroller-based mobile platform. System

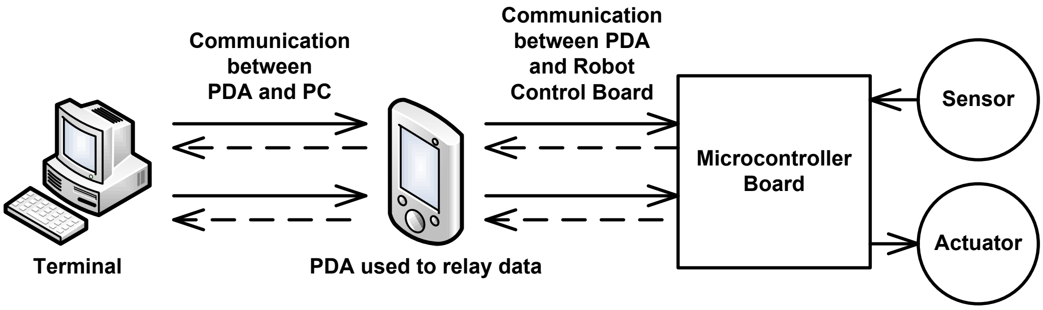

Communication between PDA and PC
Communication between PDA and Robot Control Board
Microcontroller Board
Sensor
Actuator
Terminal
PDA used to relay data

Figure 1.1: Experimental set-up with no Intelligence on the PDA

performance is judged according to response time, battery life and implementation flexibility. Measurement data will be obtained through the implementation and experimental results from a test scenario in two different processing configurations, shown in Figures 1.1 and 1.2.

The first (Figure 1.1) will use the PC, as done in many projects, as the main/only processing device, using the PDA only to relay data between the microcontroller board and the PC. The second (Figure 1.2) will use a PDA with intelligent components implemented according to the proposed framework.

The results of this study will be reported in the form of a dissertation. In addition, relevant conferences will also be targeted (see Appendix A).

1.5 Hypothesis

This study hypothesises that it is feasible to distribute control intelligence from a PC to a PDA and that this will result in an increase in system

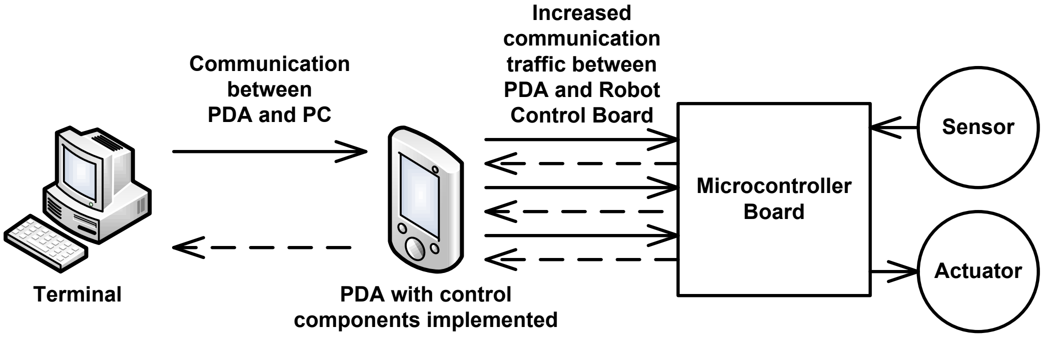

Increased communication traffic between PDA and Robot Control Board
Communication between PDA and PC
Microcontroller Board
Sensor
Actuator
Terminal
PDA with control components implemented

Figure 1.2: Experimental set-up using the PDA as an Intelligent Agent

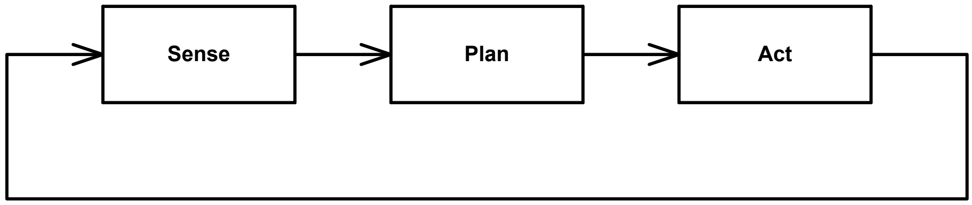


Figure 1.3: Sense, Plan, Act Organisation in Hierarchical Paradigm (Murphy, 2000)

performance.

A well-known software architecture for wireless mobile robot control is the Sense, Plan, Act paradigm as shown in Figure 1.3.

According to this control architecture the robot gathers sensory data, followed by a planning stage that determines the next action and the robot subsequently does the action. There is no direct link between sensing and acting and because the planning stage can be a lengthy process, robots designed using this architecture are often too slow to react in a dynamically changing environment (Murphy, 2000).

A hybrid approach, combining low-level reactive behaviours with higher level deliberation and reasoning, has since gained favour among researchers. Hybrid systems are usually modeled as having three layers; one deliberative, one reactive and one middle layer. Figure 1.4 shows the general layered architecture representing a hybrid approach.

The lowest level, the world interface level, interfaces directly with the robot's sensors and actuators. All time-critical tasks are handled by the behaviour based layer such as keeping the robot moving in the desired direction and avoiding static and dynamic obstacles. All subsequently higher layers will also receive sensory information at increasing levels of abstraction and can direct relevant robot responses through the inter-process communication and synchronisation mechanism. The local planning layer is concerned with local navigation needs such as getting from one location to another. The global planning involves strategic planning needs such as path planning, re-planning and multi-robot coordination.

This project proposes an implementation of the hybrid architecture, with the global layer implemented on a PC and local planning layer on a PDA, and the behaviour based functions located on the mobile platform's controller. It hypothesises that this, together with a suitable communications architecture,

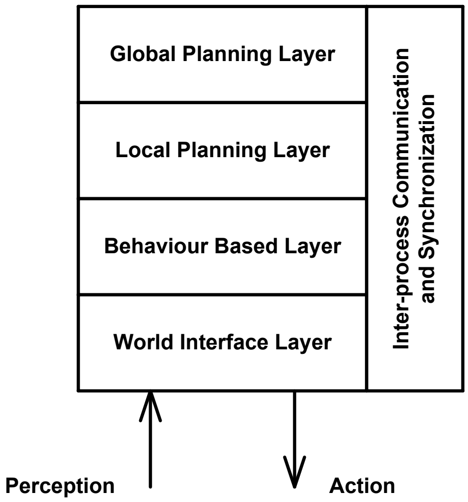


Figure 1.4: General Hybrid Architecture

would improve performance.

1.6 Delimitation

The software control components may vary with the robot application area. A mobile robot designed for space exploration may have different software requirements, and therefore components, than a mobile robot used for automatic vacuuming. The application focus for this project will be mobile robots used for human assistance in office environments. The software components described will be limited to this field only.

A number of control components will be identified that could potentially benefit from implementation on a PDA. The focus of this study is not to implement all possible components on the PDA and PC, or to develop a fully functioning robot controller, but only as would prove or disprove whether such a distribution in processing and intelligence will have an increase in system performance. Performance is measured with respect to response time, PDA battery life and other quantifiable measures.

This project will not include developing a fully functional navigation, localising and mapping scheme for the robot.

The prototype will be based on an ad-hoc connection between the PC and the PDA.

The software will be developed for a PDA running the Windows CE

(Pocket PC) operating system, which is a variation of the Microsoft Windows operating system and offers the familiar "MS Windows" look and feel.

1.7 Chapter Layout

The proposed layout of the dissertation is depicted in Figure 1.5 and is divided into four parts.

Part I introduces the domain of discourse to the reader and is divided into 3 chapters. **Chapter 1** provides some background into the problem area in order to describe the problem. In **Chapter 2** PDAs, their development, current PDA technology and the advantages and limiting factors of a PDA used as a control device is discussed. Next, **Chapter 3** provides an overview of PDA mobile robotics, hardware architectures used and the implementation of distributed intelligence in existing mobile robots.

Part II is dedicated to the formulation of the framework and its implementation in an experimental set-up. In **Chapter 4** the conceptual foundation for the proposed framework is provided. **Chapter 5** develops the mobile platform. In **Chapter 6** the control software is described. **Chapter 7** describes the communications software.

Part III describes the rationale and design of the proposed experimental set-up. The viability of the developed framework is demonstrated through a number of experimental tests done using the prototype and experimental set-ups. **Chapter 8** discusses the tests done to assess the system performance and the results obtained.

Part IV with **Chapter 9** concludes the dissertation and suggest areas for further research.

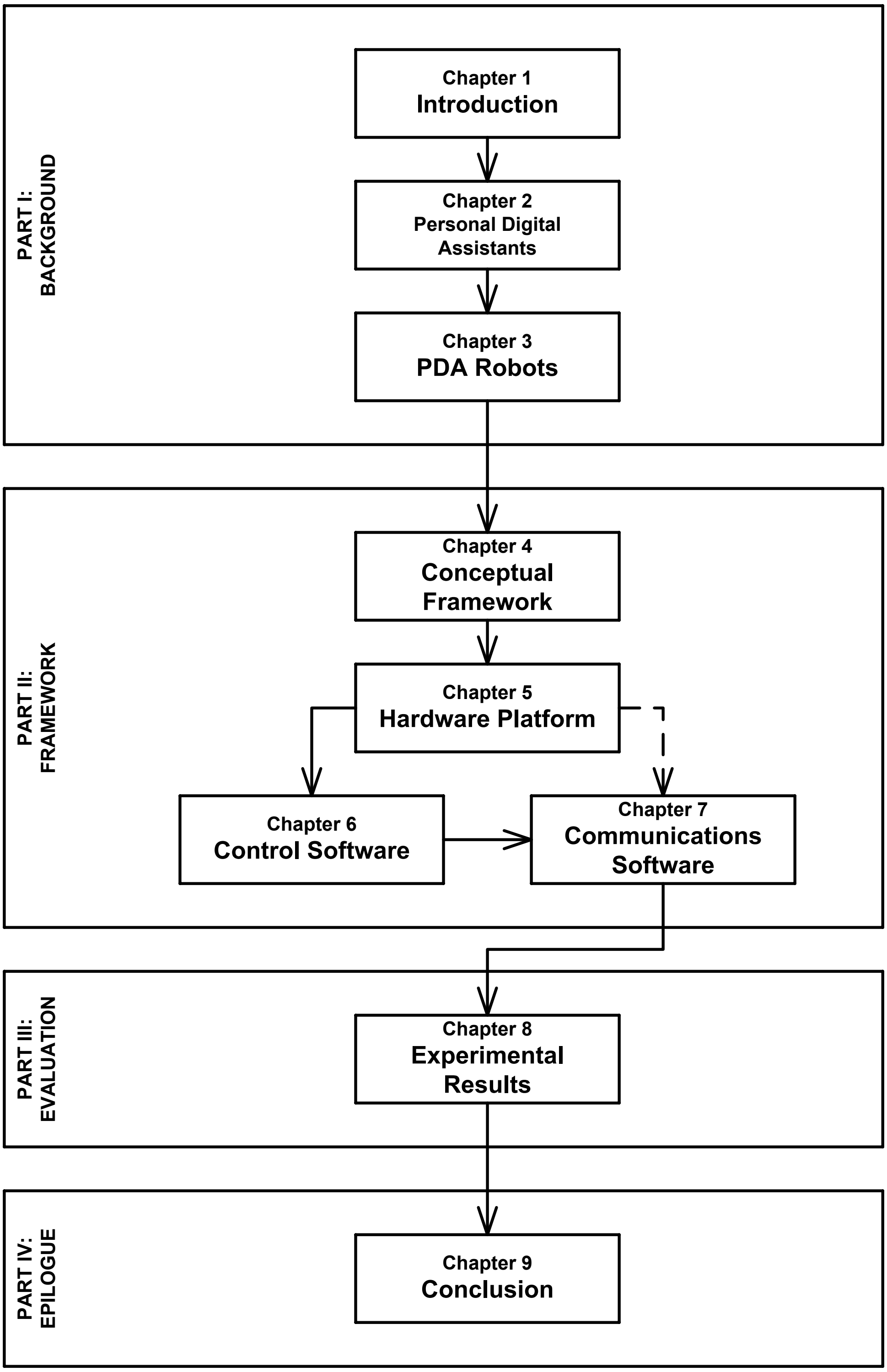

PART I: BACKGROUND
Chapter 1 Introduction
Chapter 2 Personal Digital Assistants
Chapter 3 PDA Robots
PART II: FRAMEWORK
Chapter 4 Conceptual Framework
Chapter 5 Hardware Platform
Chapter 6 Control Software
Chapter 7 Communications Software
PART III: EVALUATION
Chapter 8 Experimental Results
PART IV: EPILOGUE
Chapter 9 Conclusion

Figure 1.5: Chapter Layout

Chapter 2

Personal Digital Assistants

This chapter describe the origin of the PDA (Personal Digital Assistant), what its original purpose was, its development and what features are common to modern PDAs. The definition of a PDA as used in this study in given and also how this device differs from other mobile computers.

The PDA's history shows how this device has developed into a device which today has many rich features built into them. Today PDAs enable users to have electronic calenders, phone- and address books, connect to other devices and networks, make a phone call and connect to the Internet. This chapter elaborates on the most common PDA features and gives an overview of the specification for the PDA used within this study – the HP iPAQ 614c.

2.1 What is a PDA?

A PDA (also called handheld) is a mobile or portable PC, meaning that it can be easily or conveniently transported. A device's portability increases inversely proportional to its relative size and weight. For example desktop PCs, being the largest, is the least portable, laptop PCs are more portable but less so than palmtops and PDAs.

PDAs are not the only form of mobile computers. Others include Laptops, Subnetworks, Ultra-Mobile PCs, Portable and Mobile Data Terminals, Electronic Organisers, Pocket Computers, Handheld Gaming Consoles, Wearable Computers, Handheld PCs, Portable Media Players and Digital Audio Players. Figure 2.1 shows a PDA as well as some other forms of mobile computers.

Many people use the name of one of the popular PDA products as a

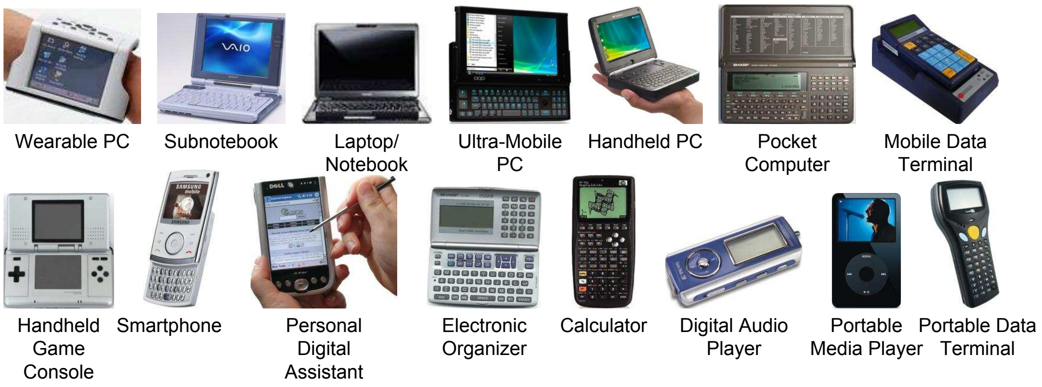


Figure 2.1: Mobile Computers

generic term. These include Hewlett-Packard's Palmtop and 3Com's PalmPilot (Williams, 2003).

According to NCCW (2008) there are three defining points that makes a PDA. First, the device should have a "one-handed design". This limits the size of the device roughly to palm-size. To consider a device as a PDA it must therefore not require two hands to handle it properly. Secondly the device must be able to "function independently". This requires that the device does not rely on an external power source and that it must not require the user to carry extra components. And lastly the device's application set is "non-appliance and non-mathematical" meaning that it does more than a calculator and does not just allow playing of games or translate words. Such a device would also not fall into the category of a PDA. Besides these three criteria, this study identifies a fourth measure to distinguish a PDA from other mobile devices such as ultra-portable PC's. This study distinguishes a handheld from others in its use of a specially designed and optimised operating system. This means that devices that use versions of standard operating systems is not considered a PDA as this study groups these devices as ultra-mobile pc's.

A PDA or handheld device is intended to be a portable, self-contained information management and communication device. It is a small hand-held device that has computing power and can store and retrieve information such as schedules, calendars and address book information (Williams, 2003). A PDA does not only manage personal information such as contacts, appointments and to-do lists. They can also connect to the Internet, have a GPS receiver built-in, run multimedia software and have a built-in mobile phone (Carmack & Freudenrich, 2008b). Other functionality might include an added interface such as a miniature QWERTY keyboard, a touch screen, a built-in camera, an accelerometer, the ability to read business documents in various formats, software for playing music, browsing photos and viewing video clips.

The line between what classifies as a mobile phone or PDA is becoming less clear. Phone-PDAs (also called Smartphones) are mobile phones that incorporates accepted PDA features. This includes software like an appointment calendar, a to-do list, an address book for contacts and a note-taking program. Smartphones typically also include e-mail and Web support. Some distinguish between PDAs and PDA-phones (a PDA with a built-in mobile phone), but the classification between PDAs, phone-PDAs (Smartphones) and PDA-phones is debatable and they are all grouped under the term “PDA” in this study.

2.2 Related Devices

All the devices shown in Figure 2.1 are mobile computers, but they each preform a slightly different function or have a slightly different form factor. The lines between the different types of mobile computers are becoming more blurred as mobile devices incorporate ever larger sets of functions and built-in hardware. For example manufacturers combine the features of different mobile computers into single devices such as combining PDAs with mobile phones, multimedia players and other electronics (Carmack & Freudenrich, 2008b). Also some multimedia players combine the functions of a PDA with multimedia features, such as a digital camera, an MP3 player and a video player (Carmack & Freudenrich, 2008a). The names representing types of mobile computers other that “PDA” are often mistakenly used as synonyms for PDAs. This section describes some of these related, yet different mobile computing devices.

Subnotebook

A subnotebook (also called a netbook, ultraportable and minilaptop) is a small and lightweight portable computer, which has most of the features of a standard laptop computer but on a smaller scale. Subnetworks are smaller than laptops but larger than handheld computers. They often have smaller-sized screens, usually measuring from about 17 cm to 34 cm, and weigh up to about 2 kg. The term often applies to systems that run full versions of desktop operating systems, rather than specialised software such as Windows Mobile or Palm OS as used on PDAs. The essential configuration reduces the standard notebook to having only a display, a keyboard, a hard drive, and a few critical data ports resulting in a compact, lightweight and more portable form of a standard laptop computer. Extra drives in the form of external drive can connect to the subnotebook through a USB link (Page, 2005).

Handheld PC

A Handheld PC (also called a Palmtop) is a computer built around a form factor which is smaller than any standard laptop computer. Handheld PC differs from related devices (such as the Palm-Size PC, Pocket PC, or Smart-Phone) in that the specification provides for larger screen sizes as well as a keyboard.

Portable Media Player

A Portable Media Player is a mobile computer related to the PDA, but focuses on integration with Microsoft's Windows Media Centre and Windows Media Player and can play digital audio, images, and video. Introduced in 2004 it runs an adapted version of the Windows Mobile operating system. Some players include readers for memory cards, emulates PDA features and gives support for games.

Pocket Computer

A Pocket Computer is a small calculator-sized handheld computer programmable in BASIC. This specific category of computers existed mainly in the 1980s. It differs from modern PDAs in that they are self-contained units with their own outputs for example to print a document and did not link with a desktop PC.

Ultra-mobile PC

A UMPC (Ultra-mobile PC) is the term used to describe the platform for small form-factor tablet PCs. The UMPC normally has a touch sensitive display of 4 to 7 inches and weighs less than 2 pounds. A UMPC runs the same versions of operating systems as a standard notebook or tablet PC including Windows XP Tablet PC Edition, Windows Vista and Windows XP. Others run on specially adapted versions of Linux (Microsoft, 2008).

Ultra Low-Cost PC

A ULPC (Ultra low-cost PC) is a laptop PC with limited hardware capabilities. This automatically lowers their cost and manufacturers qualify for discounts on operating system software. These PC's have screens of no bigger than 10.2 inches and hard drives up to 80 GB and do not come with touch screens (Shah, 2008).

Palm-Size PC

The Palm-Size PC is the name Microsoft gave to their first PDAs after Palm Inc contested the name "Palm PC". Palm-Size PCs uses the Windows CE 2.01 and 2.11 operating system (Wikipedia, 2008a). The name "Palm-Size PC" later became "Pocket PC" to refer to PDAs using the Windows Mobile operating system.

Wearable PC

A wearable computer is a computer that one wears on the body and is especially useful for applications that require computational support while the user attention cannot focus solely on the PC.

Some smart watches offer some PDA functions in a wristwatch form factor. These watches can receive weather and news, receive calendar information and personal messages.

2.3 History

The earliest methods used to keep information such as appointments, addresses and telephone numbers were the notepad, diary organiser and Rolodex. Though launched in 1921 by the Norman & Hill Ltd company, the ring-bound organiser (commonly called a FiloFax) only became popular in the late 1970s and early 1980s (Schrödinger & co, 1996). The FiloFax is the size of an A5 page and contains sheets for diary entry, notes, addresses and telephone numbers and tasks. Users could also buy new sheets as well as new sections for organising for example projects, meetings, minutes and travel information.

Electronic Organisers were the first electronic devices to replace the FiloFax which had a built-in diary application, an address book and calendar. They had the problem of compatibility and many devices were incapable of communicating with other devices, and those able to connect to a PC would do so using different formats. Upgrading to a newer device often meant loss of data (H2G2, 2004).

A PIM (personal information manager) is considered a direct ancestor of the modern PDA. Their features included the ability to link with a desktop PC, standard functions in the form of programs to organise information such as contacts, appointments, tasks, and notes. They were user-friendly and some models included a stylus for input. PIMs also allowed users to upgrade with new software via a PC link.

Today all PDAs come with some kind of PIM software (Carmack & Freudenrich, 2008b).

The first PDA was arguably the Psion Organiser released in 1984 (Psion, 2008). But the fist device to carry the name "PDA" was Apple's "MessagePad" also known as the "Newton" in 1993 (shown on right of Figure 2.2). The Newton was very ambitious for its time, featuring handwriting recognition software, plug-in memory cards, fax, e-mail and IR communications (Zeldes, 2005).

PDAs have become much more versatile and popular over the years. Today's traditional PDAs are descendants of the original PalmPilot (Pilot 1000 in March 1996 from Palm Inc) shown in Figure 2.2 (left) (Palm Inc., 2007b) and Microsoft Handheld PC (by manufacturers like HP, Compaq and Casio), who use Windows CE (HPC, 2001). Rather than attempting to stand alone as a computer, the Pilot was designed to easily and quickly exchange information with a PC. It sat in a cradle that was plugged into the desktop computer. The basic Pilot 1000 retailed for $299, half the price of a Newton. It could hold 500 addresses and 600 appointments. The Pilot 5000 had four to five times the memory and sold for $369 (Palm Inc., 2007b). The first Windows CE PDA competed with the Pilot 1000 in November 1996. The idea of a mobile office was introduced with the release of a BlackBerry PDA with synchronised e-mail capabilities in 1999 (PC MAG.COM Encyclopedia, 2008).

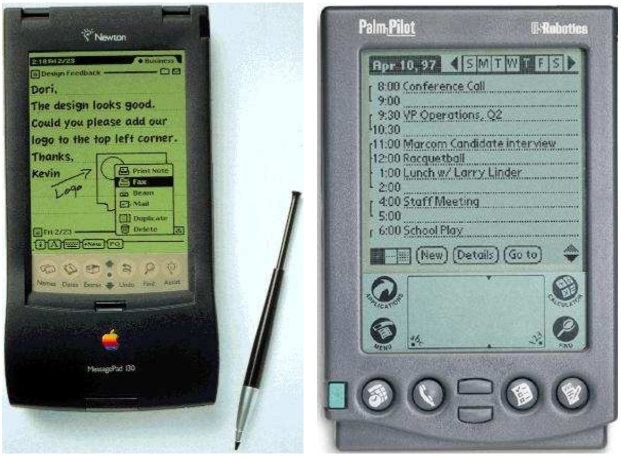


Figure 2.2: Apple's MessagePad – "The Newton" (left) and the first Palm Pilot (right)

PDAs can now be described as being miniature versions of typical desktop PC systems. However, space and power consumption constraints have limited the processing power, storage space, and available memory (Advanced System Technologies Ltd., 1999).

Even this drawback is set to diminish in the near future with low power processors being developed for mobile devices such as the ATOM from Intel set to be comparable to their current Intel Core 2 Duo processor (GADGETS, 2008). Clock speeds of up to 624 MHz, are possible in current leading models, but PDAs also have some fundamental hardware differences when compared to standard desktop and laptop PCs. For example few currently incorporate a hard drive, but store all basic programs in ROM and any additional programs and data in RAM or Flash memory (Advanced System Technologies Ltd., 1999).

PDAs have developed from their earliest counterparts, the electronic organiser and personal information manager (PIM) to a highly capable, integrated device. Many features have become common in these devices and the following section describes the basic components of a standard PDA system. PDAs are becoming all-in-one devices and features such as wireless networking, built-in sensors and mobile phones also come included with some PDAs.

2.4 Modern PDA Features

Today's PDAs have several features that are now common to many devices in this class of mobile computer. Figure 2.3 shows the basic parts that form a general PDA system (Carmack & Freudenrich, 2008b). Figure 2.3 shows the PDA's processor as central to the device. Other components include memory (RAM, ROM and Flash), wireless connectivity through Wi-Fi, Bluetooth and IR. The PDA also has several different input methods that include the touch sensitive display, buttons and keyboard. Other key parts include the PDA's power supply and its ability to synchronise with a standard PC.

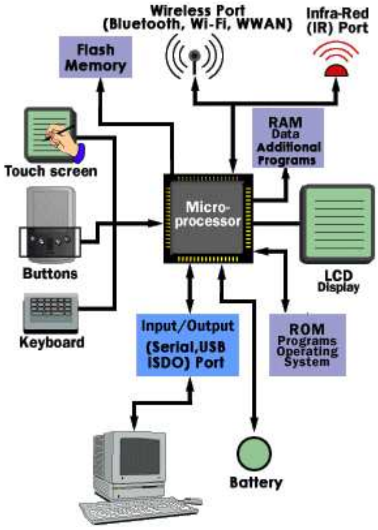


Figure 2.3: Components of a PDA

High-end PDAs may also offer multimedia, security and add-on features not found on less expensive devices. These features include an SDIO card slot to add peripherals to the PDA. Cards can extend a PDA's capabilities by for example adding Bluetooth, Wi-Fi or GPS functionality. A built-in digital camera and GPS capabilities and even security features such as an incorporated fingerprint reader can also come included (Carmack & Freudenrich, 2008a). Many PDAs now incorporate mobile phone technologies combining the features of a standard mobile phone with that of a PDA.

PDAs usually have a docking cradle. This is a device used to connect a PDA to a PC for synchronisation and application downloads. The two connect via the PDA's communication port using a serial or USB cable. The cradle often doubles as a battery charger.

The following sections highlights some of the common features of a PDA with a short description of each.

2.4.1 Processor

Microprocessors, just like in standard desktop and laptop computer, powers a PDA. The microprocessor is the brain of the PDA and coordinates all the functions according to programmed instructions. PDAs mostly use smaller, cheaper microprocessors when compared with desktop and laptop PCs. Different processors work at different clocking speeds and this (for PDAs) today range between 200 MHz and 624 MHz (Mobile Tech Review, 2008). Although these microprocessors are often much slower than their PC counterparts, this may not be the case for long as newer devices start to incorporate Intel's new Atom processor into their handhelds. The Atom works from 800 MHz to 1.87 GHz and based on the x86 and x86-64 instruction sets, use less power and has a smaller footprint. Intel currently targets ultra-mobile PCs and MIDs (Mobile Internet Devices) specifically with this processor. Gigabyte's M528 (left of Figure 2.4) is one of the first MID's based on Intel's Atom. The M528 use an 800 MHz Atom processor, has 512 MB of RAM, and a 8 GB SSD. Sharp's D4 mobile phone (right of Figure 2.4) uses Intel's Centrino Atom platform to run Windows Vista Home Premium SP1. It has a 1.33 GHz Atom Z520 onboard with 1 GB of 533 MHz DDR2 memory and a 40 GB hard drive (Smith, 2008).

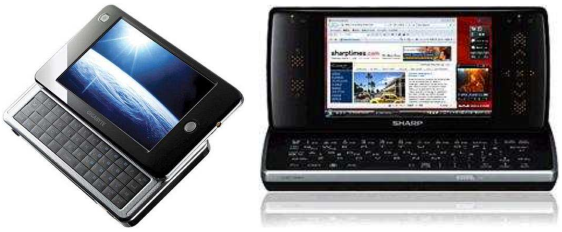


Figure 2.4: Atom-based Gigabyte M528 and Sharp D4

Undoubtedly the most popular embedded CPU architecture today is the ARM architecture and most handheld processors today base their design on it. ARM CPUs find themselves in most corners of consumer electronics, from portable devices (such as PDAs, mobile phones, media players, handheld gaming units, and calculators) to computer peripherals (including hard drives and desktop routers). ARM processors featured in industry firsts such as Apple Computer's Newton. About processor development for today's market ARM says: "In a competitive and changing market place where digital entertainment and communications devices are converging, the need for a high performance processor that can meet both the demanding performance requirements of leading-edge consumer entertainment devices and the tight power requirements for advanced mobile products is clear" (ARM, 2008).

Unlike other functions of a PDA, processor specifications cannot always simply be compared between different products. If devices run different operating systems there could be a variation in speeds because of their different designs (Conger, D., 2003).

2.4.2 Operating System

A mobile operating system (also known as a mobile platfrom or a handheld operating system) contains the pre-programmed instructions that tell the microprocessor what to do. The operating systems used by PDAs are not as complex as those used by PCs. They have fewer instructions, which requires less memory (Carmack & Freudenrich, 2008a). The operating system governs what software can be used on the PDA, and (except for memory) what expansions can be used with the included expansion slots. The operating system also determines the UI and general feel, or user experience with a specific PDA (Conger, D., 2005).

The Gartner report for the second quarter (2008) sales for smartphones shows the Symbian platform to have 57.1% of the market share. Though this is down from 65.6% of the market for the second quarter in 2007, it still makes Symbian OS the market leader. Second is RIM (BlackBerry) with 17.4% then Windows Mobile with 12%, Linux with 7.3%, Max OS X (iPhone) with 2.8%, Palm OS with 2.8% and other platforms sharing 1.1% of the market (phoneArena.com, 2008).

This following sections describes some of the most popular mobile operating systems. These include the Windows Mobile platform by Microsoft, the Palm OS from PalmSource (now ACCESS), RIM Blackberry owned by Research In Motion, Symbian OS, iPhone OS and Linux based operating systems.

Windows Mobile

Windows CE is based on the Microsoft Windows operating system but designed for embedding in mobile and other space-constrained devices of different shapes, sizes and degrees of ruggedness. Being a compact operating system combined with a suite of basic applications for mobile devices based on the Microsoft Win32 API, it manages the interaction between application software and the hardware on the physical units. Its design is similar to desktop versions of Windows in its features and aesthetics. Windows CE is used in several types of mobile computers (ranging from Pocket PCs, Smartphones, Portable Media Centres, on-board computers for certain automobiles, TV set-top boxes and other rugged, custom devices) (Laberge & Vujosevic, 2003).

Like the full-scale Windows systems, Windows CE is a 32-bit multitasking, multithreading operating system (Franklin, 2006) but has been optimised for devices that have limited storage. A Windows CE kernel may run in under a megabyte of memory. Windows CE conforms to the definition of a real-time operating system, with a deterministic interrupt latency. It supports 256 priority levels and uses priority inheritance for dealing with priority inversion. The fundamental unit of execution is the thread.

Windows Mobile is a platform based on the Windows CE operating system. Windows Mobile is similar to Windows, but is not Windows and does not run Windows applications. Its design, performance and user experience is all similar to that of a Windows PC (Conger, D., 2005).

Devices without an integrated phone is called “Windows Mobile Classic” instead of “Pocket PC”. Devices with an integrated phone and a touch screen is called “Windows Mobile Professional” and devices without a touch screen are called “Windows Mobile Standard” (Hall, 2007). Currently, Pocket PC (now called Windows Mobile Classic), SmartPhone (Windows Mobile Standard), and PocketPC Phone Edition (Windows Mobile Professional) are the three main platforms under the Windows Mobile umbrella. Each platform utilises different components of Windows CE, as well as supplemental features and applications suited for the respective devices.

Palm OS

One of the major operating systems for mobile devices is the Palm OS by PalmSource (now a subsidiary of ACCESS). Palm OS is an embedded operating system designed for mobile devices. It was originally designed for the Pilot series of PDA's launched in 1996 and has since been implemented into an array of mobile devices including smartphones, wrist watches, game consoles, barcode readers and GPS devices. Palm OS is designed for ease of use with a touchscreen-based GUI and to provide PIM related applications (Wikipedia, 2008).

Palm OS is known for its speedy navigation when compared with Pocket PCs. Palm's VersaMail program has the ability to fetch e-mail over a variety of connections including Bluetooth, Wi-Fi, and desktop synchronisation, which relies on your PC's Internet connection (CNET Networks Inc., 2008b).

Devices running the Palm OS will be similar to that of personal organisers, but will have additional features for more advanced uses (Conger, D., 2005). Palm OS therefor has a personal information manager style. The user interface is modeled more after personal organisers than standard PCs. And the main focus of the Palm operating system is the management of your personal information like contacts, calendar, and tasks (Conger, D., 2005).

A feature admired of the Palm OS is that a program does not need a formal close or exit. Only one program can execute at a time and opening a new one will close the other automatically. And unlike some operating systems like Windows Mobile, Palm OS never gets slowed down by multiple applications running simultaneously in the background (CNET Networks Inc., 2008b).

RIM BlackBerry OS

BlackBerry OS is the proprietary software platform made by Research In Motion for their BlackBerry line of handhelds. It provides multitasking, and makes heavy use of the device's specialised input devices, particularly the thumbwheel. The lack of stylus and touch screen slows down some operations, but the built-in keyboard that is a signature feature of BlackBerries accelerate others such as composing e-mails (CNET Networks Inc., 2008a). Among mobile devices the BlackBerry is considered the best at handling e-mail.

Symbian OS

Symbian OS (a descendant from Psion's EPOC) is an open operating system, designed for mobile devices (mainly on mobile phones and smartphones). Its associated libraries, user interface frameworks and reference implementations of common tools are all produced by Symbian Ltd. The Symbian OS runs only on ARM processors.

The Symbian operating system offers a broad array of PIM features, including contact and calendar management and a robust library of third-party applications. But the Symbian operating system is usually tailored to individual hardware so that it will look and act differently depending on the device that it's running on. It incorporates full support for Word, Excel, and PowerPoint documents, though the ability to create and edit these type of documents or just view them depends on the hardware (CNET Networks Inc., 2008c). 9,834 third-party Symbian applications have been released for Symbian OS up to the second quarter of 2008 (Symbian, 2008).

Symbian OS technology has been designed with several key points in mind including power, memory and I/O resource management, complies with global telecommunications and Internet standards and to facilitate wireless connectivity for various networks (Nokia, 2008). Mobile phone manufacturers that shipped Symbian phones in the first half of 2008 are Fujitsu, LG, Mitsubishi, Motorola, Nokia, Samsung, Sharp and Sony Ericsson (Symbian, 2008).

iPhone OS

The iPhone OS (also called OS x iPhone) is the operating system developed by Apple Inc. for the iPhone and iPod touch. The four main applications are Phone, Mail, Safari (for web browsing), and iPod (for listening to music). Other applications included are: SMS (Text messaging), Calendar, Photos, Camera, YouTube, Stocks, Maps (Google Maps), Weather, Clock, Calculator, Notes, Settings, iTunes , App Store and Contacts. The CPU used in the iPhone and iPod Touch is an ARM-based processor (Apple Inc., 2008b).

Linux

Rather than being a platform in its own right, Linux is the basis for various different platforms developed by several vendors which are mostly incompatible.

A new player in the mobile OS market is Google (now the Open Handset Alliance) with its Android OS based on the Linux kernel. Android allows developers to write managed code in a Java-like language that uses Google-developed Java libraries but does not support programs developed in native code (Wikipedia, 2008). The first handheld using the Android platform is the T-Mobile G1 (previously HTC Dream) (T-Mobile, 2008).

2.4.3 Memory

Unlike PCs, most PDA's do not have hard drives because they can be too large, too slow and consume too much power. In recent years hard drive technology has improved and iPods and other personal media players have tiny MicroDrive hard drives (miniature hard drives designed for use in mobile devices). Like any hard drive, data stored there is persistent, which means that it requires no power to preserve its contents.

The first PDA to have a hard drive was Sharp's Zaurus SL-C3000 in November 2004 (Smith, 2004). The Zaurus SL-C3000 (figure 2.5) has a 4 GB hard drive. It uses a 416 MHz Intel XScale PXA270 processor with 64 MB of SDRAM and 16 MB of Flash ROM. The Palm LifeDrive Mobile Manager also has a 4 GB hard drive.

Figure 2.5: Sharp Zaurus SL-C3000, the First PDA with a Hard Drive

Other than those with hard drives, PDAs instead have RAM and ROM memory, a portion of which is used to store programs and data. The ROM area is non-volatile, which means it isn't erased even if you wipe out the PDA via a hard reset. ROM is where the operating system and basic programs that come with the unit is installed, which remains intact even when the machine shuts down. If you install add-in commercial and shareware programs, they will be stored in the device's RAM. Information in RAM is only available when the device is on. Some PDAs allow you to store programs and data in ROM as well (Mobile Tech Review, 2008). Due to their design, PDAs keep data in RAM safe because they continue to draw a small amount of power from the batteries even when you turn the device off (Carmack & Freudenrich, 2008a). Some newer PDAs use flash memory instead of RAM. Flash memory is non-volatile, which means that no power is needed to preserve the information stored in the chip. In addition, flash memory offers fast read access times (although not as fast as volatile DRAM memory used for main memory in PCs) and better shock resistance than hard disks.

To provide additional memory, many PDAs accept additional memory through removable flash media cards into expansion slots. These typically provide storage for large files with multimedia content, such as digital photos.

2.4.4 Expansion Slots

It is difficult to change or upgrade hardware (such as the processor, memory and other parts) of a handheld when compared to a standard PC. Expansion slots gives the user an easy and inexpensive way a to add memory to the PDA. (SearchMobileComputing.com, 2007) Expansion slots can also accept expansion units such as a modem, network card, software ROM or digital camera. Popular slots include CF (Compact Flash), SD (Secure Digital), MMC (Multi Media Card) and Palm Universal (a manufacturer-specific slot for Palm devices).

CF and SD slots are also compatible with other devices such as different PDA makes, digital cameras and MP3 players that also make use of these slots. CF expansion slots can accept memory or expansion cards, SD slots can only use memory while SDIO slots can use both memory and expansions (SearchMobileComputing.com, 2007). SDIO is an SD card with input/output functionality. Input/output functionality allows cards to feature expansions like network adapters and cameras such as the FlyCAM camera from LifeView in Figure 2.6.

To keep the device's size down, many PDAs now have mini-SD or micro-SD slots instead of full-sized slots.

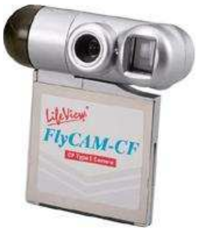


Figure 2.6: A FlyCAM Camera that Connects to the CF Slot

2.4.5 Power Supply

Most PDAs use battery power. Older models use alkaline batteries, while other, newer PDAs use rechargeable batteries such as lithium, nickel-cadmium or nickel-metal hydride batteries. The battery life depends on what PDA you have and how you use it. There is several causes influencing power consumption (or battery drain) of a PDA's battery including the operating system used. Reportedly Windows CE (Windows Mobile) devices use more power than for example Palm devices (SearchMobileComputing.com, 2007). The amount of memory also affect the power consumption as a PDA with more memory also using more power. Using the wireless connections such as WiFi and Bluetooth and using the display's backlight will all drain extra power.

The battery life (the time between charges) vary from hours to months depending on the model and make as well as the features included. PDAs usually have a power management system to extend battery life. Exhausting all power result in the loss of data stored within RAM. Even if the PDA can no longer switch on the PDA there will usually still be enough power to keep the RAM refreshed (Carmack & Freudenrich, 2008a). Most PDAs also have an internal backup battery to provide short-term power for cases such as when the battery is removed for replacement.

Besides battery power, many PDAs come with AC adaptors to run off household electrical supplies. A car adaptor is often available as an accessory (Carmack & Freudenrich, 2008a).

2.4.6 Display

PDAs use an LCD (liquid-crystal display) screen with a backlight for reading in low light conditions. Unlike the LCD screens for desktop or laptop computers, where it's solely an output devices, PDAs use their screens for output and input. The LCD screens of PDAs are smaller than laptop screens, but vary in size. Almost all modern PDAs now come with colour displays (Carmack & Freudenrich, 2008a).

2.4.7 Input Methods

PDAs vary in how you input data and commands. Many PDAs give users a choice between several input methods. Some devices use a stylus and touch screen exclusively in combination with a handwriting recognition program. A stylus is a pen-like device for navigation and data input. Usually a handheld will have a secure slot for storing its stylus. The user applies a plastic stylus to the touch sensitive display to draw characters on it or on a dedicated writing area (separate from the display area). Software inside the PDA converts the characters to letters and numbers.

A miniature on screen keyboard (virtual keyboard) can be used for inputting text. The virtual keyboard has the same layout as a regular PC keyboard but requires the user to tap on the letters with the stylus. Many PDAs include a small QWERTY (as used in mobile phones) or full QWERTY keyboard (often sliding out). These allow users to enter data using their fingers. A lightweight, full-size keyboard can also connect to the PDA through Bluetooth, USB or IR. Figure 2.7 shows a Snap N Type keyboard (MobileTechReview, 2002). Silicon keyboards that roll up and attach to the PDA's sync port are also available. They are lightweight and water- resistant. Figure 2.7 shows a silicon keyboard made by Flexis (Zhang, 2002). An example of a Bluetooth-based keyboard extension is the Stowaway Universal Bluetooth Keyboard (Gade, 2004) and the IR keyboard from PocketTop Computer Corporation (Zhang, 2004) shown in Figure 2.7.

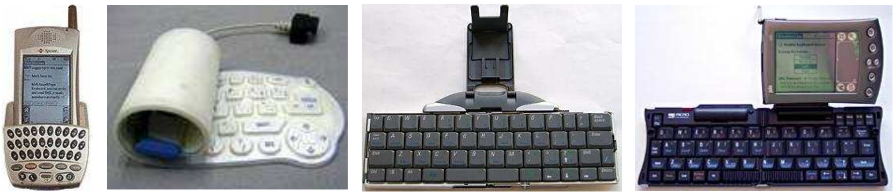


Figure 2.7: From left to right: Snap N Type, Silicon, Bluetooth and IR keyboards

2.4.8 Synchronisation

PDAs are designed to work with a standard PC. The information such as contact lists, appointments and e-mail should be the same on both the PDA and PC. Synchronisation software allows this sharing between a PDA and PC and is one of the key features of a PDA and one that most PDAs have. Having a backup of information prevents data loss due to a PDA being lost, stolen or damaged.

Different operating systems make use of the different synchronising software. Palm OS provides HotSync Manager and Windows Mobile has ActiveSync and its successor Windows Mobile Device Center. Palm's Hotsync can synchronise with Palm Desktop software as well as Microsoft Outlook and Microsoft's ActiveSync and Windows Mobile Device Center can only synchronise with Microsoft Outlook or a Microsoft exchange server. There are also third-party software available to synchronise a PDA with PIMs not supported by the PDA's manufacturer. An example is the program *The Missing Sync* that allows synchronisation for Windows Mobile PDAs with Macintosh systems (CNET Networks Inc., 2008d).

2.4.9 Wireless Connectivity

PDAs usually have a combination of IR (Infrared), Bluetooth and WiFi for wireless connectivity to other devices. Short-range wireless connectivity can be through IR or Bluetooth. IR requires clear line of sight between the two devices and is commonly used to sync with a notebook computer that has an IR port (Carmack & Freudenrich, 2008b).

Bluetooth is the wireless equivalent of USB (Universal Serial Bus). It is defined as short-range because it only operates at less then 10 m from another device. This means that Bluetooth enables compatible mobile devices, peripherals and computers that are close to one another to communicate direct with one another without wires. Bluetooth has the advantage (being RF) that it does not require line of site between connected devices (Conger, D., 2003).

WiFi is a midrange wireless solution and works at a maximum distance between devices of between 33 and 50 metres. WiFi is the wireless equivalent of Ethernet or a LAN (Local Area Network). WiFi networks are setup similar to wired Ethernet networks and wireless networks can be accessed through an access point or wireless router. WiFi allows for connection to the Internet if the network (or computer) the WiFi card is connected to has an Internet connection (Conger, D., 2003). PDAs commonly implement WiFi version 802.11b and 802.11g. 802.11b operates in the 2.4 GHz band and has a top transfer speed is 11 Mbps. 802.11g also operates at 2.4 GHz but supports data transfer over improved and faster protocols at up to 54 Mbps.

2.4.10 Sensors

Modern PDA's can include several sensors. Sensors give PDAs added functionality, improve the user experience and save battery power. Examples of built-in sensors include ambient light sensors that enables the PDA to automatically adjust the display's brightness level according to the current environment.

Proximity sensors is incorporated into some PDAs as part of a power conservation strategy. When the user lifts the handheld to his or her ear, the proximity sensor will trigger the device to turn its display off.

An accelerometer is a sensor that measures movement through aspect and velocity. Having this sensor built into a PDA allows the PDA to respond to the user's movement. The accelerometer can signal a change in the screen layout of the display between portrait and landscape for example when viewing photos and websites. It can also be used as control of a game through moving the handheld (Apple Inc., 2008a).

GPS (Global Positioning Sensor) technology is a satellite-based navigation system that uses information from earth-orbiting satellites to find locations (El-Rabbany, 2002). A receiver estimates the distance to GPS satellites based on the time it takes for signals to reach it and then uses that information to identify its location. A-GPS (Assisted GPS) finds the closest satellites to more quickly identify (or fix) a location than regular GPS. Figure 2.8 shows an example of a map used with GPS technology on a PDA.

A built-in digital compass can be used to provide an automatic North reverence and is used to assist with the GPS and accelerometer features (Wikipedia, 2008b).

2.5 HP iPAQ 614c

The Hewlett Packard iPAQ 614c shown in Figure 2.9 is the chosen PDA for this study. This section describes the many features of this device.

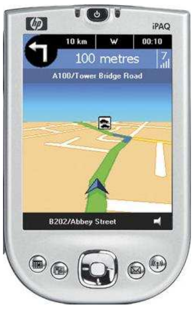


Figure 2.8: GPS Map on a PDA

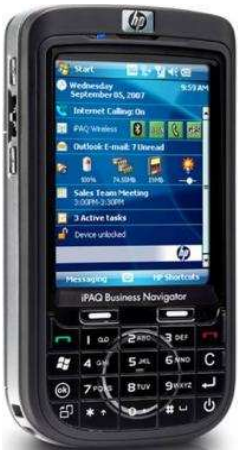


Figure 2.9: The HP IPAQ 614c

The iPAQ 614c uses the Windows Mobile 6 Professional operating system. It uses the Marvell PXA270 Processor which runs at 520 MHz. The memory included consists of 128MB SDRAM main memory for running applications and 256 MB flash ROM. The PDA has a 64-bit micro-SD card slot for memory expansion. An added 2 GB micro-SD memory card expands the PDA's memory.

The display is a TFT type display, 2.8" big and has a resolution of 240 x 320 pixels. The display is in the form of a touch panel, lit by an LED backlight. The touch-sensitive display allow for finger or stylus operation. The iPAQ 614 c measures 6.03 x 1.75 x 11.7 cm and weighs 145 g.

The PDA comes with an integrated 12-button numeric keypad. Other integrated buttons include a smart touch wheel, a three-way thumb wheel, two soft key buttons, send and end buttons, a reset button, a volume control button, a voice command button, and a landscape/portrait key. The PDA also come with several messaging services including a Phone Dialer application, SMS support, MMS composer, Microsoft Outlook Push e-Mail, Pull e-Mail via ActiveSync, HP Voice Reply, Predictive Text tool, and Microsoft Live Messenger

The IPAQ 614c has a built in mobile phone which is an integrated Quad-band GSM/GPRS/EDGE phone and has Tri-band HSDPA 3.6/7.2 Mpbs.

It has a built in GPS receiver for assisted GPS navigation.

Software that came with the device include: HP iPAQ Mobile Broadband Connect, HP VoiceReply, Voice Commander, MMS Composer, Bluetooth Manager, Certificate Enroller, HP Photosmart Mobile, HP Help and Support, HP QuickStart Tour, and HP Enterprise Mobility Agent. Outlook Mobile, Office Mobile, Internet Explorer Mobile, Windows Media Player Mobile, Microsoft ActiveSync, Phone Dialer, Voice Notes, Calculator, Solitaire, Bubble Breaker, Microsoft Live Messenger and Microsoft Live Search all come pre-installed on the device.

The 614c comes with a 3.0 Megapixel autofocus integrated digital camera with 4X digital zoom, 640 x 480 VGA resolution, and 1280 x 1024 SXGA resolution. The iPAQ also come with an integrated microphone, receiver, speaker and stereo headphone jack.

Wireless technologies built into the device are WLAN 802.11b/g with WPA2 security and Bluetooth 2.0 with EDR.

Besides the micro-SD card slot there is also one mini-USB connector available for synchronisation and charging.

A removable and rechargeable Lithium-Polymer 1590mAh battery powers this PDA. The included AC adapter takes an AC power input of between 100 and 240 Vac of 50/60 Hz and gives an output voltage of 5 Vdc (typical) and an output current of 1 A (typical). According to the device's specifications the talk time can be up to four hours and the standby time up to 10 days or 250 hours.

2.6 Conclusions

PDAs have been around since the late 1980s and has since developed into devices that provide users with a rich feature set that converges the different technologies of PIMs, mobile phones and wireless connectivity into a single portable unit.

This chapter looked at the term “Personal Digital Assistant” (or “PDA” for short) as used to describe small, mobile and hand-held computing devices. Here a hand-held computer classifies as a PDA using the four key measures of: design, independent functioning, application and type of operating system used. Section 2.3 gives a brief description of the origin of the PDA. And Section 2.4 described modern PDA features such as input methods, operating systems, processors and memory in more detail followed by a description of the PDA chosen for this study– the HP iPAQ 614c from Hewlett Packard.

Chapter 3

PDA Robots

This chapter describes robots that make use of a PDA in some way. Some projects use a PDA as a means to add features easily to their design such as wireless technologies. Other use the built-in touch sensitive screen as a means to provide communications with their existing robot control system. The PDA-robots described in this chapter clearly show the PDA can perform various roles within a robotic system. The focused application or research area also spans wide-ranging topics. These include low-cost prototype development, low-cost humanoid robots, robot control, vision systems and algorithm development, human-robot interaction (HRI), mobile user interfaces as well as wireless robot communication schemes.

From the many existing PDA robots it is possible to identify five general ways in which PDAs work across the different application areas. The five sections are:

1. The PDA as the controlling device, doing all the needed processing.
2. The PDA as message forwarder, providing wireless connectivity to the robot base.
3. The PDA as teleoperation device, allowing remote operation of the robot.
4. The PDA as part of a multimodal interface, a user interface alternative.
5. The PDA as part of a distributed control system

This section groups descriptions of PDA robots and the different projects according to the purpose the PDA fulfils within the system they form part of.

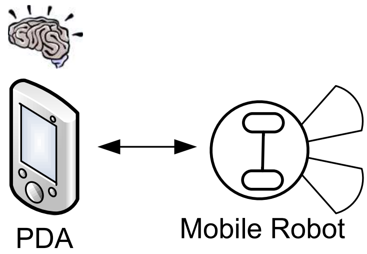


Figure 3.1: The PDA as the Main Controlling Device

3.1 The PDA as the Controlling Device

This section describe robots using a PDA to do all the needed processing. This can include image processing and calculating motor commands to send to the onboard controller on the robot platform. Figure 3.1 shows a general representation of the architecture used by robots in this section. The robot platform has a controller which is either a microcontroller-based board, an embedded PC or an LEGO RCX brick. The onboard controller interfaces with the robot's sensors (if there is any) and its motors. It also links with the PDA – usually through a direct link such as RS232 or IR.

3.1.1 PDA Robot

The PDA Robot, in the book *PDA Robotics: Using Your Personal Digital Assistant to Control Your Robot* (Williams, 2003) has a PDA brain which connects to a PIC microcontroller-based control board through an IR link. An infrared transceiver allows the microcontroller board to interface with the IR port of the PDA, handling the IrDA handshaking and data exchange between the Robot and the PDA. The microcontroller interfaces with the motors and distance sensor. The PDA implements all control software. A wireless camera, visible in Figure 3.2 on the mobile robot's platform, allows remote viewing of what the robot sees on a PC screen. The PC acts as a remote control station for the robot.

3.1.2 Robota

Robota (Figure 3.3), a humanoid type robot, uses a PDA to do the needed image and speech processing for the research in HRI with gesture and speech

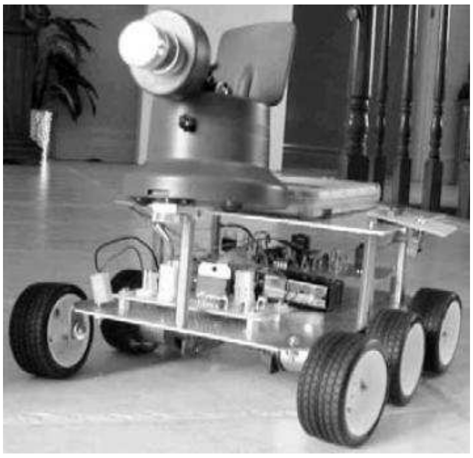


Figure 3.2: PDA Robot

as medium (Calinon & Billard, 2003). Robota is not mobile, but has 5 degrees of freedom with which the robot mimics and learns from the human interacting with it. The authors created a PDA-based language learning program. The microcontroller based sensor and motor control cards connects to the PDA through an RS232 serial link. The PDA calculates motor commands to send to the motor control card. The main constraint to their work is the limited processing power of the iPAQ 3850 PDA they used, limiting them to implement only simple vision and speech processing applications.

3.1.3 Toni

Toni, shown in Figure 3.3 (middle), as well as its successors Jupp, Sepp, and Max are all soccer playing robots designed and built at the Albert-Ludwigs-University of Freiburg (Behnke et al., 2005b). Toni, a fully autonomous humanoid robot, uses an FSC Pocket Loox 720 PDA as its main controller. The PDA sends target positions for the servos attached to three microcontroller-based boards every 6 ms. The microcontroller boards sends processed sensory data back to the PDA for behavioural control and controls the 18 servomotors used. The PDA connects to the microcontrollers through an RS232 serial link and communicates with a remote PC for debugging through WLAN. The added camera gives the only information about the robot's environment.

3.1.4 NimbRo RS

The NimbRo RS project (Behnke et al., 2004) uses a modified RoboSapien (a commercially available remote controlled humanoid robot) as its robot base to create a low-cost platform for humanoid robot research, especially multi-agent research using humanoid robots. The Toshiba e755 PDA replaces the head of RoboSapien (right of Figure 3.3) and does all the image processing and motion control needed to make this an autonomous robot as well as communicating through WLAN with other robots. The PDA connects to the on-board controller through an IR link. The RoboSapien has some sensors built in, including: bumper switches on its feet, contact sensors on its fingers and a sonic sensor for reacting to clapping hands. The current modification cannot use these sensors to provide information about the robot's environment because of the different and incompatible IR interfaces used by the PDA and RoboSapien respectively. This led to setting up a unidirectional IR link and adding a small colour camera to the PDA through its CF slot. The low precision in walking and the limited DOF provided by the RoboSapien platform makes reliable navigation unfeasible as well as limiting the number of possible movements. A remote PC does visualisation and debugging of the performed behaviours and coordinates team behaviours as needed for soccer playing robots. The frame rate of the camera and the rate at which the RoboSapien's controller can accept commands limits the rate of behaviour decisions. The rate is about 4 Hz (Behnke et al., 2005a).

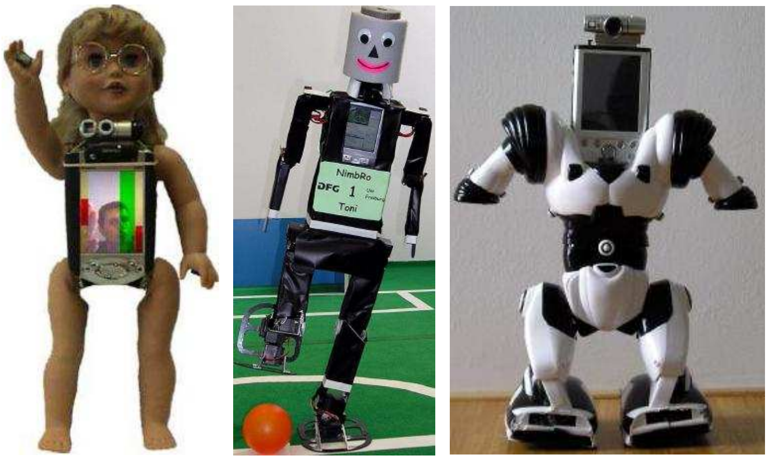


Figure 3.3: Robota (left), Toni (middle) and NimbRo RS (right)

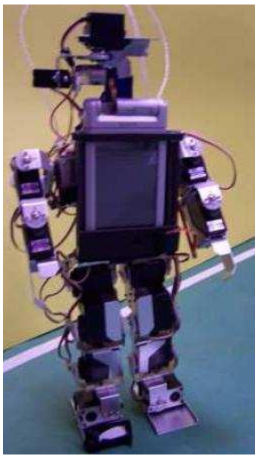


Figure 3.4: Abarenbou

3.1.5 Abarenbou and DaoDan

Built using a modified Kondo KHR-1 humanoid fighting robots (Figure 3.4), Abarenbou and DaoDan (Baltes et al., 2006) where developed to be low-cost humanoid robot platforms. A Sony Clie NR70V PDA and Palm Zire 72 PDA does all the image processing and motion control needed to make these robots autonomous soccer players. Two microcontroller boards control the servomotors and connects to the PDA through an RS232 serial link. The camera provides all the environmental information to the robot with no other sensors implemented. A remote PC does the motion development using XML based meta language. The result is converted to C before being loaded on to the PDA. They found full image processing to be too slow.

3.1.6 PDA Robot Vision System

Jantz & Doty (2002) created a robot vision system with the use of a PDA and a digital camera. The PDA links to the TJ Pro robot platform through an RS232 serial link (Figure 3.5). The limited processing power of the Handspring PDA forced the use of efficient algorithms and limited their choice of implementation. The greatest limit imposed by using a PDA was the constraint on memory. The PDA controls the motor speeds and reads the sensor data. They concluded that using a PDA is an efficient choice for vision research and for small mobile robots that have limited processing power onboard the platform itself.

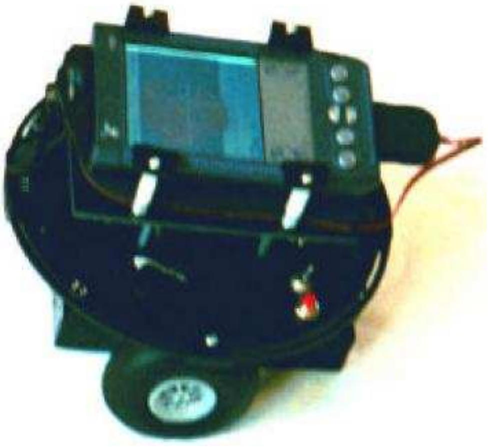


Figure 3.5: PDA-based Robot Vision System

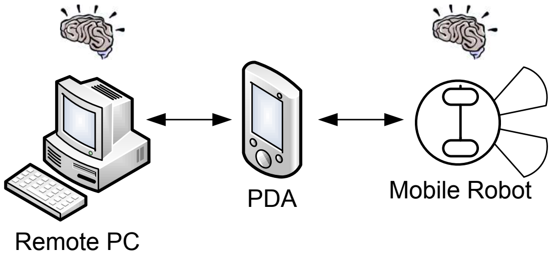


Figure 3.6: The PDA as a Message Forwarding Device

3.2 The PDA as Message Forwarder

This section describe robots that use a PDA to provide a wireless link between the main processing device and the robot body. This can be a WLAN or Bluetooth communications link. The remote device (a PC or another PDA) runs the needed control and processing software. There is a direct, physical link between the PDA and the robot base in the form of an RS232 serial or IR link. The controller onboard the robot platform interfaces with the robot hardware and links with the PDA. Figure 3.6 shows a general representation of the architecture used by robots in this section. Usually no control software runs on the PDA itself.

3.2.1 Mini-Whegs Robot

This project (Joshi, 2003) wanted to answer the question of whether a PDA could provide a small mobile robot such as the mini-Whegs with efficient motor control over a wireless link. A PDA attaches to the mini-Whegs robot as shown in Figure 3.7, which incorporates a microcontroller-based board to interface with the motors as well as providing the necessary serial RS232 interface. The lack of sensors in the design means there is no feedback from the robot's environment. The PDA wirelessly links the robot base to the remote PC. The PC provide a GUI with which the user can set the speed and angle of a specific motor. The PDA is essentially a message forwarding device within this robot system.

3.2.2 WiMo

The WiMo robot is the creation of Brian Cross, a Windows Mobile software engineer (Cross, 2007). It has a Windows Mobile 5.0 Smartphone that connects through Bluetooth to the microcontroller board on the robot base (Figure 3.8). A remote PC (or Pocket PC) connects to the Smartphone through WLAN, giving basic commands to the robot base via the Smartphone. WiMo has a camera function which sends streaming video to the remote PC. It can respond to voice commands as well as SMS-based commands. Microsoft Robotic Studio (MSRS) runs on the remote PC. WiMo essentially uses the Smartphone as a message forwarding device and to capture video with its

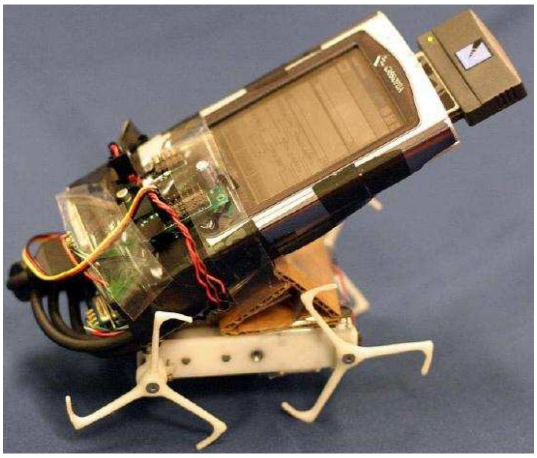


Figure 3.7: PDA-Controlled Mini-Whegs Robot

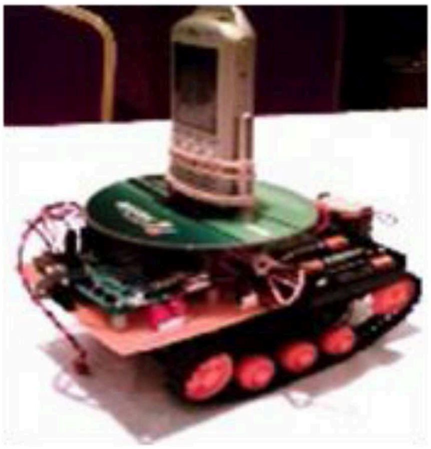


Figure 3.8: WiMO

built-in camera. It forwards standard movement commands such as forward, back, left, and right to the microcontroller. A LEGO-based WiMo has been built using the LEGO Mindstorms NXT set and is named WiMo NXT.

3.2.3 NiVek J.D.

The NiVek J.D. (Figure 3.9) by Kevin Wolf is a robot similar to the WiMo robot (Wolf, 2007). The Nivek J.D. uses a PDA to connect wirelessly to the robot base through Bluetooth and to a remote PC through WLAN. The added GPS receiver module (right of Figure 3.9) allows showing the path travelled by the robot. Just like the WiMo robot it also makes use of MSRS on the remote PC, showing captured video as well as allowing the user to send basic direction commands to the robot. The PDA acts as a "Repeater that allows for communications from the NiVek embedded computer to a PC" (Wolf, 2007).

3.2.4 PEWIR Robot

The PEWIR Robot in (Langeveld et al., 2005) use a PDA to provide a means of wireless communication with their robot platform. The platform consists of a Lynxmotion chassis (Figure 3.10). The PDA (or laptop) connects to the microcontroller-based control board on the robot through an RS232 serial link. The user teleoperates the robot through a web server running on a remote PC via the Internet with the help of the feed from a wireless camera

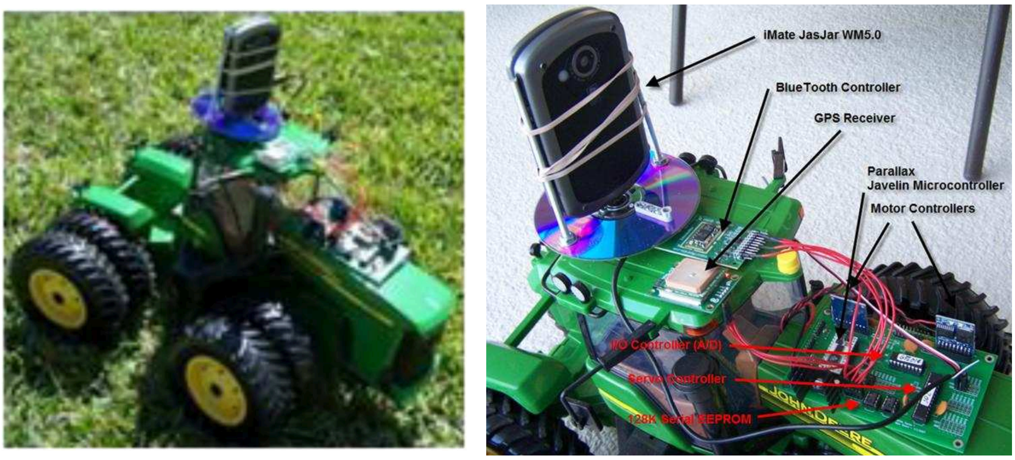


Figure 3.9: NiVek J.D. (left) and its Components (right)

found on the robot body.

3.2.5 Calligrapher Robot

The Calligrapher Robot described in (Kovan Research Lab, 2006) is a PDA controlled mobile robot that navigates according to a path drawn on to the screen of the PDA. The robot can be autonomous or teleoperated and uses two HP Jornanda 548 PDAs in its hardware design as well as the Palm Pilot Robotic Kit (PPRK) (The Robotics Institute at Carnegie Mellon University, 2001) as its base (right of Figure 3.10). One PDA is the remote user in-

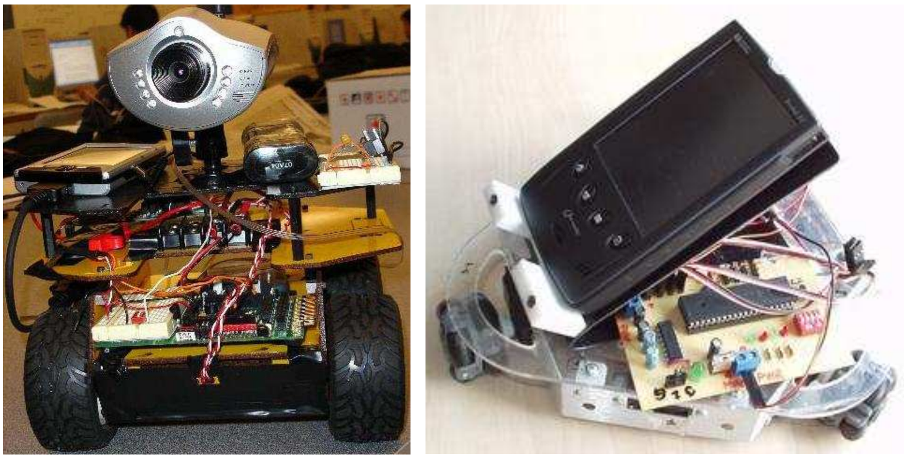


Figure 3.10: PEWIR Robot (left) and Calligrapher Robot (right)

terface, allowing the user to draw a path for the robot to follow using the PDA's stylus and screen. This PDA calculates the necessary motor velocities and communicates through Bluetooth with the second PDA found on the robot base. This PDA is "significant only because of its Bluetooth capabilities" (Kovan Research Lab, 2006). It receives the motor commands and sends it to the microcontroller on the robot base through an RS232 serial link.

3.2.6 A PDA LEGO robot

The robot shown in Figure 3.11 uses two light sensors, the RCX brick from the LEGO Mindstorms Robotic Invention System, a grid pattern on the floor and a PDA. This LEGO-based robot (Buschmann et al., 2003) navigates from one grid position to the next. Goal positions are sent wirelessly from a remote PC that connects to the PDA though WLAN. The PDA connects to the RCX brick through an IR link and is "solely to provide a wireless communications gateway to the remote PC" (Buschmann et al., 2003). The RCX brick carries out the algorithm for path planning and navigation. The RCX brick has limited resources (memory, processing power) available and because of this the algorithm employed is not ideal. The inaccurate drive system of the LEGO platform created another problem, resulting in the robot occupying the wrong grid position.

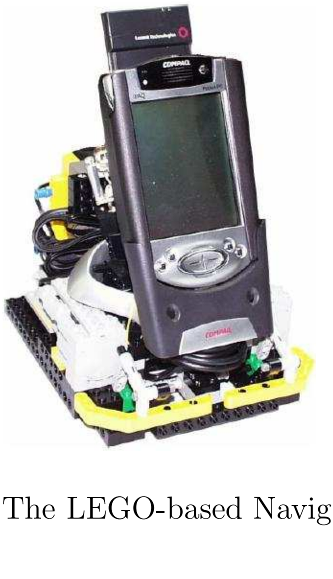


Figure 3.11: The LEGO-based Navigation Robot

3.3 The PDA as Teleoperation Device

This section describe robots that use a PDA to provide wireless teleoperation for a mobile robot. There is no direct, physical link between the PDA and the robot base. Usually a user interface is present on the PDA, allowing the user to send control commands remotely to the robot. Sometimes sensor data is also displayed on the screen. Figure 3.12 shows a general representation of the architecture used by robots in this category. Usually no processing is done on the teleoperation device (PDA). All the processing is done by the robot platform's onboard controller.

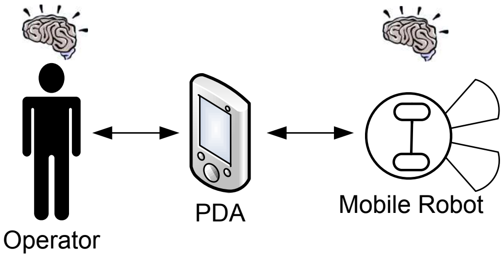


Figure 3.12: PDA as a Teleoperation Device

3.3.1 PdaDriver

The PdaDriver as discussed in (Fong, Cabrol, et al., 2001; Fong, Conti, et al., 2001; Fong et al., 2003) is a PDA teleoperation interface for a mobile robot. The PDA connects wirelessly to a Pioneer2-AT robot through WLAN. It shows camera and sensor data and allows directional control through several interface screens. The UI on the PDA uses PersonalJava and includes screens for video, map, command and sensors (Figure 3.13). In Fong, Cabrol, et al. (2001) the PdaDriver also includes human-robot collaborative dialogue, supporting various query-to-user messages and giving the robot the ability to ask the human operator for help. The robot system is safeguard-teleoperated, with all the needed autonomous features carried out by the mobile robot's onboard controller. The focus in the PdaDriver project is human-robot interaction through a mobile user interface. The latency measured between the capture of an image and the time the image displays on the PDA screen is 800ms. The control latency (the time between the operator giving the

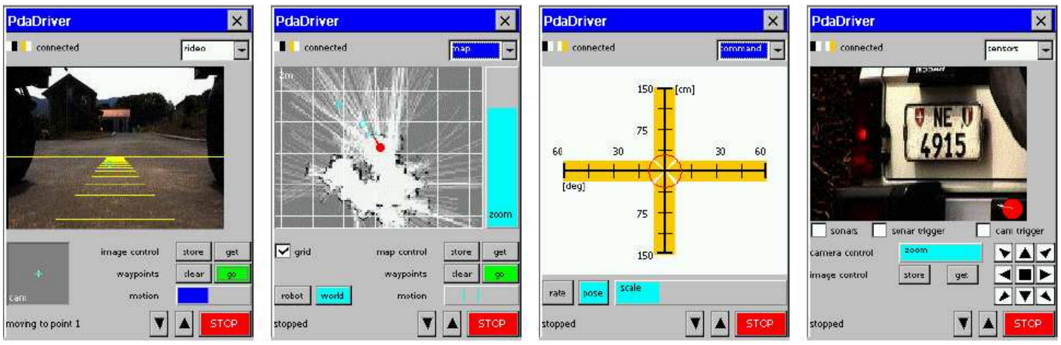


Figure 3.13: PdaDriver User Interface Screens (video, map, command and sensors)

command on the PDA and robot execution of it) is about 500ms. Operators need training to be able to handle these delays properly.

3.3.2 Touch Screen Teleoperation Interface

The project described by Keskinpala et al. (2003) is a PDA teleoperation interface for a ATRV-Jr Robot. Three screens are developed for the Toshiba E740 PDA. They include a screen that is an image-only screen, a sonar and laser range finder based screen and an image and sensor data overlay screen. The input is designed to allow the user the freedom not to use the stylus with the focus of their research being the development of human-robot interaction through mobile devices. Their design incorporates large buttons to accommodate human fingers, but due to the limited screen size the buttons were made to be semi-transparent. This allows the user to give commands and view environment information at the same time. The user is able to give basic directional commands to the robot remotely. Figure 3.14 shows the image only screen with the semi-transparent direction buttons.

3.3.3 EMG-based Modified Morse Code

Nilas (2005) describe EMG signals (electromyography or tiny electrical impulses produced by muscle contraction) carry out an adapted Morse code interface for a mobile robot. A PDA provides a portable user interface allowing both teleoperation of the robot and has the ability to receive the user's commands from EMG signals via biosensors. These signals form Morse code-based high-level task commands. The PC then decomposes the high-

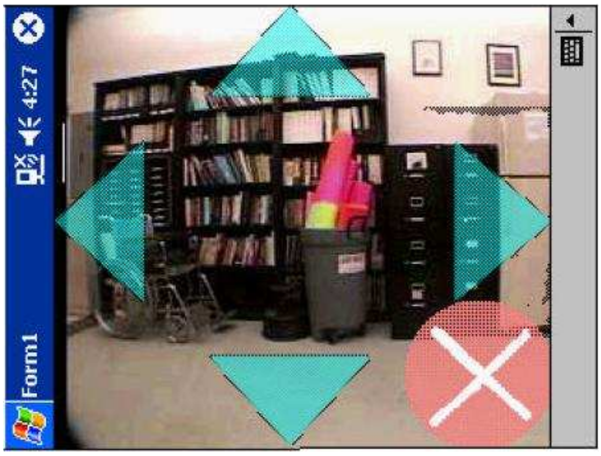


Figure 3.14: Image Only Screen Interface

level commands to low-level primitive tasks and creates a task plan for the robot to carry out. Here the PDA provided a small and lightweight mobile interaction device allowing disabled people to control a robotic aid. Their focus is on providing disabled people with a mobile and physically practical human-robot interaction device.

3.3.4 COMMBOTS

COMMBOTS by De Ipiña et al. (2005) describe the control of a fleet of small mobile robots through a GPRS data network. Their focus is remote monitoring and control. The PDA (PC or mobile phone) becomes the control station in the architecture employed. Each COMMBOT is equipped with a GSM/GPRS communication module which connects to the onboard microcontroller-based controller through an RS232 serial link. The communication module connects wirelessly with the COMMBOTS proxy. The proxy is the intermediate device between the mobile robot and the control station.

3.4 The PDA as Part of a Multimodal Interface

This section describe robots that use a PDA as part of a multimodal interface. This means the PDA provides the user an optional interface, where it is one of several interface choices. Often the PDA does not connect directly to the robot's onboard controller, but rather to a user interface module which may be running on the robot's onboard controller or on a remote PC. This module first consolidates the user inputs through the multiple possible sources such as touch, gesture and speech before giving the command to the subsequent responsible module. Figure 3.15 shows a typical representation of the architecture used by robots in this category.

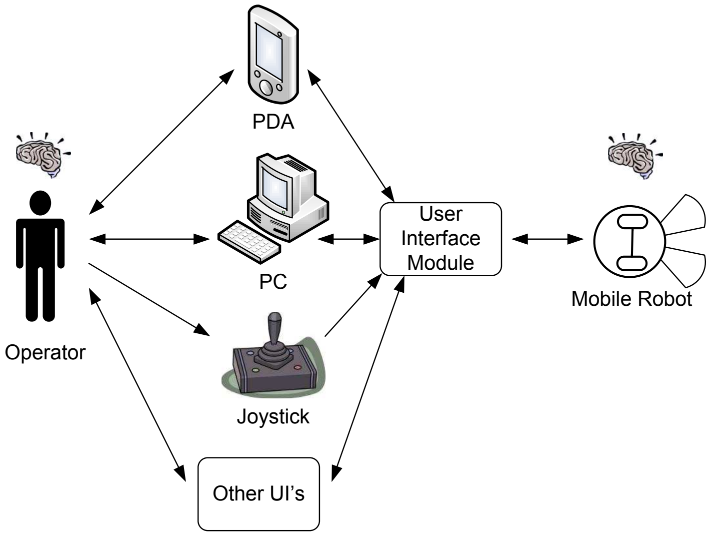


Figure 3.15: PDA as Part of a Multimodal Interface

Many robots include as part of their multimodal interface architecture a PDA or Smartphone as an optional or alternative means of providing user input to the robot. The SCOUT robot described by Rybski et al. (2001) offers the user a choice between a laptop, joystick or PDA as interfaces to their wearable-PC robot control device. In their research on human-robot interfaces, Sofge et al. (2004) include a PDA as a user interface tool in their multimodal interface design, allowing communication through touch on the PDA's touch screen. The architecture used is described in more detail in (Perzanowski et al., 2001). Here the PDA screen is used to show a map produced by the mobile robot's laser scanner, allowing the user to point on the map to command the robot in that direction. Figure 3.16 shows the PDA screen with a map representing the robot's environment. The system can use both the Nomad 200 and RWI ATRV-JR robot platforms. The user is able to give a limited set of commands to the robots through this interface.

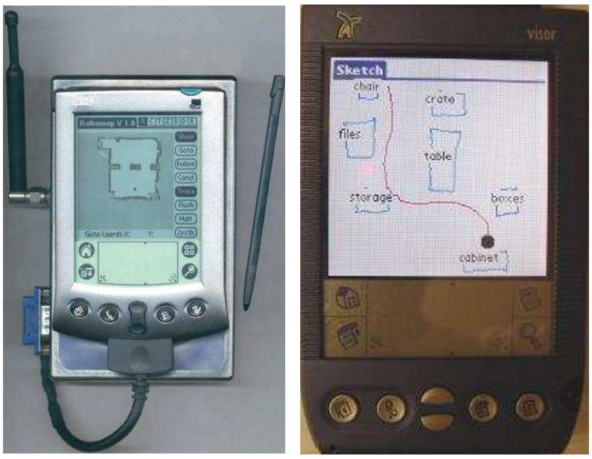


Figure 3.16: PDA screen presenting a laser scanner-formed map of the environment (left) and a scene sketched on the PDA (right)

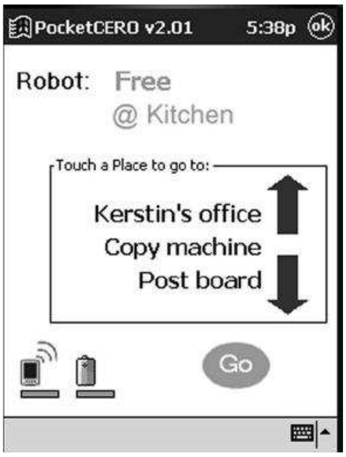
3.4.1 PDA Sketch Interface

In (Skubic et al., 2002, 2003) the authors describe a sketch-based interface for a mobile robot. The user draws a map on the PDA screen using the stylus (left of figure 3.16). The captured sketch is a sequence of x-y coordinates. The path information extracted from the sketch can help navigate a mobile robot with the route represented as a sequence of steps including landmark's relative positions. Their research concentrates on capturing route data from a sketched map that can help guide a mobile robot. The sketch client on the PDA connects to an existing robot system and a sketch server through WLAN. Here the x-y coordinates, object positions and paths are processed further, ready for use in guiding the mobile robot.

3.4.2 PocketCERO

PocketCERO is another robot that makes use of a PDA as part of its semi-autonomous and teleoperation control schemes (Hüttenrauch & Norman, 2001; Hüttenrauch & Eklundh, 2003). The PDA gives the user a means to send teleoperation commands wirelessly to the robot (a Nomadic Super Scout). Communication with the robot's on-board controller is through WLAN, but the PDA does not connect straight to the robot controller, but through a separate GUI module. They found the PDA especially useful as an interface device when both the user and the robot is on the move (roaming).

They focused on providing PDA-based HRI because they felt that a mobile robot should have a mobile UI. Their interface provides three screens; two needs stylus-based interaction and a third allows touch-based interaction. Figure 3.17 shows the prototype interface screen for the PDA that allows for single hand usage (touch-based).



PocketCERO v2.01 5:38p ok

Robot: Free
@ Kitchen

Touch a Place to go to:

Kerstin's office
Copy machine
Post board

Figure 3.17: One-hand GUI for PocketCERO

3.5 The PDA as Part of a Distributed Control System

A distributed control system, divides the control software between all the available processing elements within the system as shown in Figure 3.18.

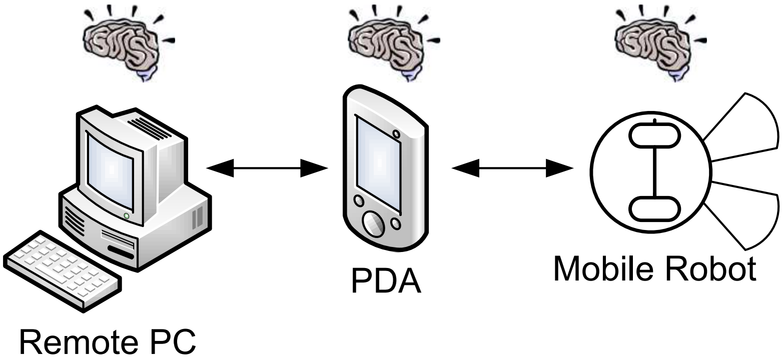


Figure 3.18: PDA as Part of a Distributed Control System

3.5.1 FLIP

FLIP is a small, low-cost prototyping robot for use within intelligent manufacturing system design (Jensen et al., 2005). FLIP is a step between the simulation environment and a real-life and life-size (usually expensive) prototype. The robot platform uses the LEGO RCX brick (left of Figure 3.19) to interface with the motors and sensors. A PDA is the brain of the robot and connects to the RCX brick through an IR link. It performs all the necessary processing for navigation and local planning. A remote PC, and an overhead camera, provides navigation support and connects to the PDA through WLAN. The PC is also responsible for multirobot coordination. Different FLIP robots can communicate their whereabouts and missions to one another through WLAN. Control intelligence distributes through the system by implementing an adapted version of the InterRRaP architecture. The incompatible IR interfaces of the PDA and RCX brick, forced development of a platform specific IR library.

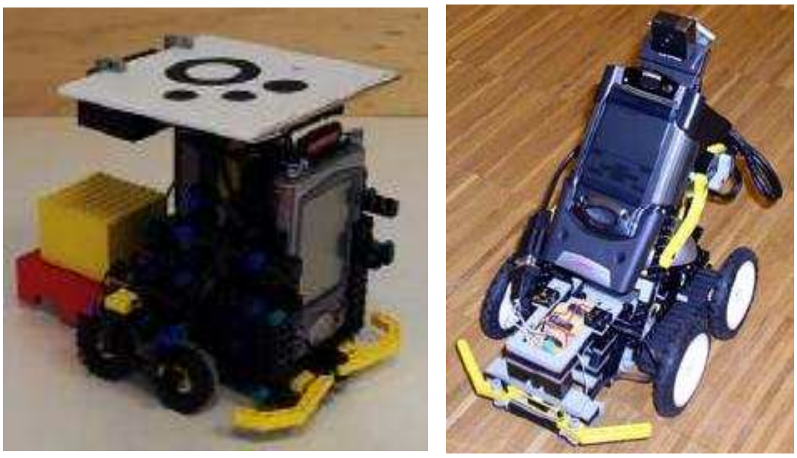


Figure 3.19: FLIP Prototyping Robot (left) and in the Caddie Paradigm (right)

3.5.2 Caddie Paradigm

The "caddie" is a mobile robot that moves like a super market trolley (Lemoine et al., 2004). A caddie is easy to drive and a user only needs to push and orient it with the hands while walking. This is implemented using a third person view of a virtual robot as well as live video from a webcam on the robot.

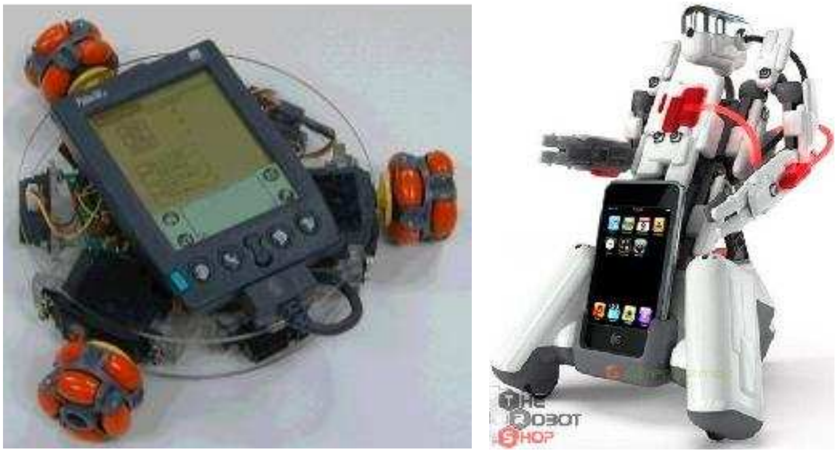
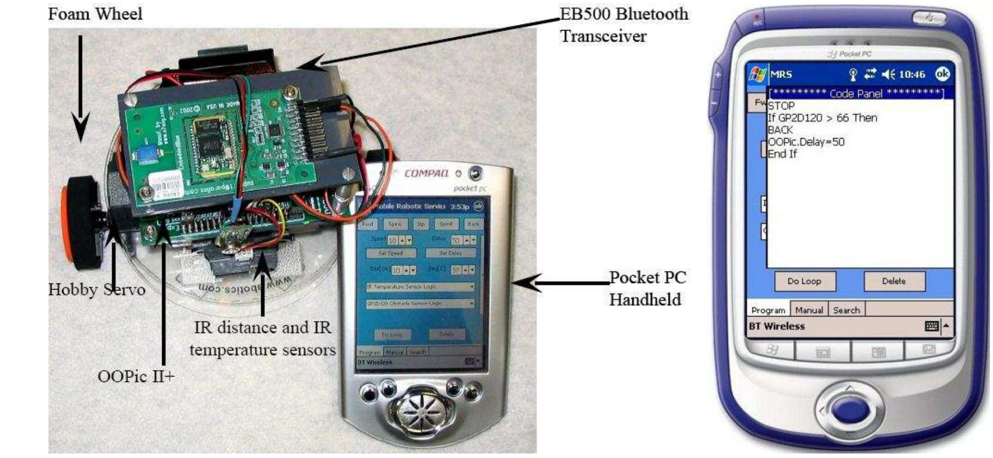


Figure 3.20: Palm Pilot Robotic Kit (left) and Spykee Cell (right)

Hand gestures (orientation) and walking on a treadmill (pushing) gives the user control of the remote robot. The walking speed of the user is mapped to the robot's speed. The mobile robot uses the LEGO Mindstorms Robotic Invention System kit as shown in Figure 3.19 (right). A PDA connects to the LEGO RCX brick through a IR link. An IR library was needed to be able to interface between the different IR protocols of the PDA and RCX brick. This project best fits this category due to the PDA running basic collision detection and response software. The PDA also accepts control messages to be sent to the RCX brick from the master, remote PC through WLAN. The remote PC does the needed image processing for the gesture commands as well as mapping the speed of the treadmill to that of he mobile robot.

3.6 Commercial PDA Robots

Some robot platforms are commercially available that specifically cater for the use of a PDA as its brain. The most famous example is the Palm Pilot Robotic Kit (PPRK) shown in Figure 3.20 and designed by Carnegie Mellon University (The Robotics Institute at Carnegie Mellon University, 2001). The book, *The Ultimate Palm Robot* (Mukhar & Johnson, 2003) describes the use of this platform. The PPRK currently retails at $325.00 from Acroname Robotics. Another example is Spykee Cell (Figure 3.20) which is available at £169.99 (The Robot Shop, 2008).



```
********* Code Panel *********
STOP
If GP2D120 > 66 Then
BACK
OOPic.Delay=50
End If
```

Figure 3.21: Pocket PC-controlled educational robot (right) and its code development screen (left)

3.7 PDA Robots in Education

Howell & Sersen (2004) describe a low-cost mobile robot for use as an educational tool. They integrate a low-cost Bluetooth enabled robot with a PDA and a web service on a PC as shown in Figure 3.21. The web service compiles the software for the microcontroller on the robot platform, stores programs within a database and allows searching through saved programs. The NewCDBot (built around the OOPic II+ microcontroller board) is the mobile platform and provides users with the choice to download a compiled program to an onboard EEPROM from where it executes. The user writes the robot program using the PDA (right of Figure 3.21). The web service then compiles the program and sends the compiled hex code back to the PDA. Once the hex is on the PDA it can be loaded onto the NewCDBot through the Bluetooth interface. The PDA can also act as a remote control device.

3.8 Mobile Phone Robots

This section describes a number of projects that uses a mobile phone in their robot designs in stead of a PDA. Modern mobile phones are powerful devices in their own right and has much the same technologies available to connect to a mobile robot as PDAs.

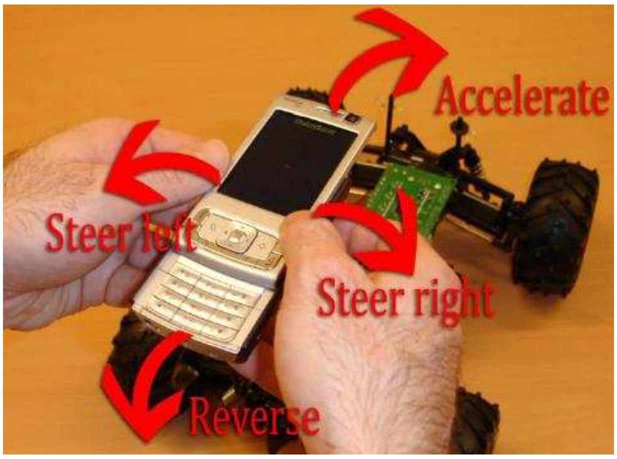


Figure 3.22: ShakerRacer

3.8.1 Robot control through a WAP enabled phone

A WAP enabled phone controls a hydraulic robot arm by d'Angelo & Corke (2001). Here the robot arm and mobile phone (via a WAP gateway) connects to the save server. The result was an even greater time delay between giving the commands using the mobile device and robot execution than using the internet with its already known latency problems. Other problems include that mobile phones do not handle floating point inputs well and the overall system was expensive due to the data and call costs associated with the connections.

3.8.2 ShakerRacer

The ShakerRacer makes use of the acceleration sensor built into the Nokia N95. The acceleration sensor together with the software on the mobile phone controls the direction of movement as well as speed of the ShakerRacer (Selinger & Jakl, 2007). Tilting the phone, moves the robot (a modified remote control car) in the desired direction (figure 3.22). The RC car is controlled by a microcontroller board equipped with a Bluetooth module. The mobile phone sends the control commands to the robot through Bluetooth.

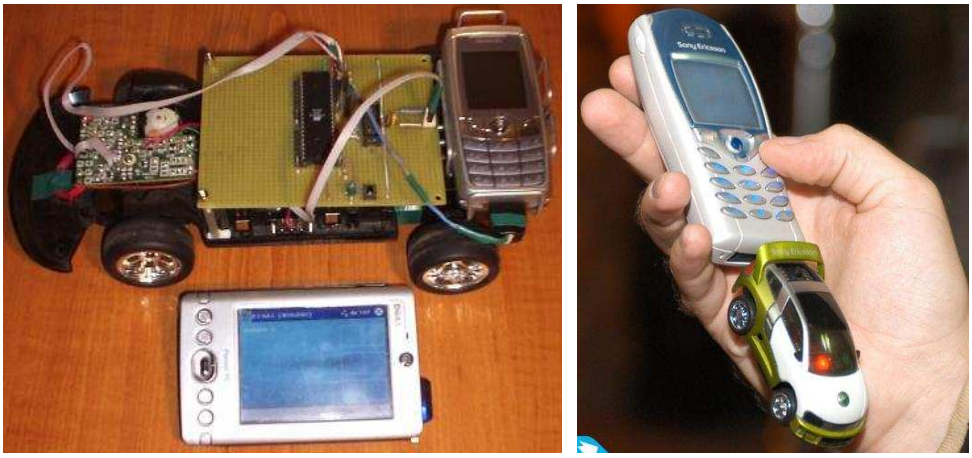


Figure 3.23: The Z-1 Robot (left) and the Ericsson Bluetooth Car (right)

3.8.3 Nokia 6020

In Patra & Ray (2007) two mobile phones provide remote and wireless connectivity to a mobile robot platform. They present a mobile robot that gets commands in the form of SMS's. The user sends the instructions from his mobile phone to a second mobile phone (a Nokia 6020) that's connected to a PC. The PC translates the sms message into a control message and forwards it to the mobile robot using an RF module connected to PC's serial port. A microcontroller-based board, which also has an RF module, controls the robot base. It sends and receives messages to and from the remote PC.

3.8.4 Z-1 Robot

The Z-1 Robot shown in Figure 3.23 is a mobile robot similar to the one described in (Patra & Ray, 2007). A mobile phone with Java support attaches to a microcontroller board on a modified RC car through an RS232 serial link. The mobile phone provide a wireless link to a remote PDA through Bluetooth. The PDA acts as a remote control device allowing the user to send control commands and view sensor data received from the mobile robot.

3.8.5 Sony Ericsson Bluetooth Car

The Sony Ericsson Bluetooth Car is a small car (see Figure 3.23) charged and controlled with a Sony Ericsson Bluetooth-enabled model phone. A Sony

Ericsson standard slot connects the car to the phone and ensures that this car can only be controlled with original handsets. The user controls the car with the phone's joystick or navigation keys. The phone then sends the commands to the car via Bluetooth. This car accessory sells for about $100 (Murtazin, 2003).

3.9 PDA Advantages

The choice of using a PDA within their robot designs were seldom made without consideration of the benefits the PDA could provide. Rather its use in their designs is a considered decision. This section mentions some of the benefits noted by the developers that they hoped to exploit in their applications.

To make their robots autonomous, most researchers turn to industrial computers, such as PC104 or other single-board computers. These computers lack a display and possibilities for user I/O, need a power supply, wireless communication, housing, and cooling. Except with a laptop, power consumption is large and to provide AC-power on a mobile unit is bulky, needing heavy batteries and an inverter.

The PDA gives users a small, lightweight interaction device (Keskinpala et al., 2003; Nilas, 2005) that is also portable, robust and affordable. The touch sensitive screen combined with a stylus allow users to interact using touch (Perzanowski et al., 2001; Skubic et al., 2002). And the built-in user interface allow novice users to control a robot (Skubic et al., 2002) through a familiar environment, minimising the need for training (Fong et al., 2003). The PDAs UI also give powerful debugging and development advantages over embedded devices (Jantz & Doty, 2002). This allows for rapid development of multimedia applications for robots (Calinon & Billard, 2003).

The built-in wireless technologies such as Bluetooth and WLAN (Buschmann et al., 2003; Kovan Research Lab, 2006) allow for remote operation "anywhere and any time" (Fong, Conti, et al., 2001; De Ipiña et al., 2005) and gives "seamless network coverage" to mobile agents and operators (Hüttenrauch & Norman, 2001).

It is an ideal device for use in cost and size limited platforms (Calinon & Billard, 2003; Baltes et al., 2006). And is easy to attach to small robots through one of the multiple interface options available (Behnke et al., 2005c). This leads to powerful small robots that are safer and cheeper (Jantz & Doty, 2002).

The PDA is a powerful device in processing power and memory, allowing integration of high-level planners and implementation of computationally expensive algorithms (Jensen et al., 2005). It can give simple robots that usually only consist of one or more microcontroller based boards the processing power and vision sensor capacity needed to be used as autonomous robots (Behnke et al., 2004, 2005c,a).

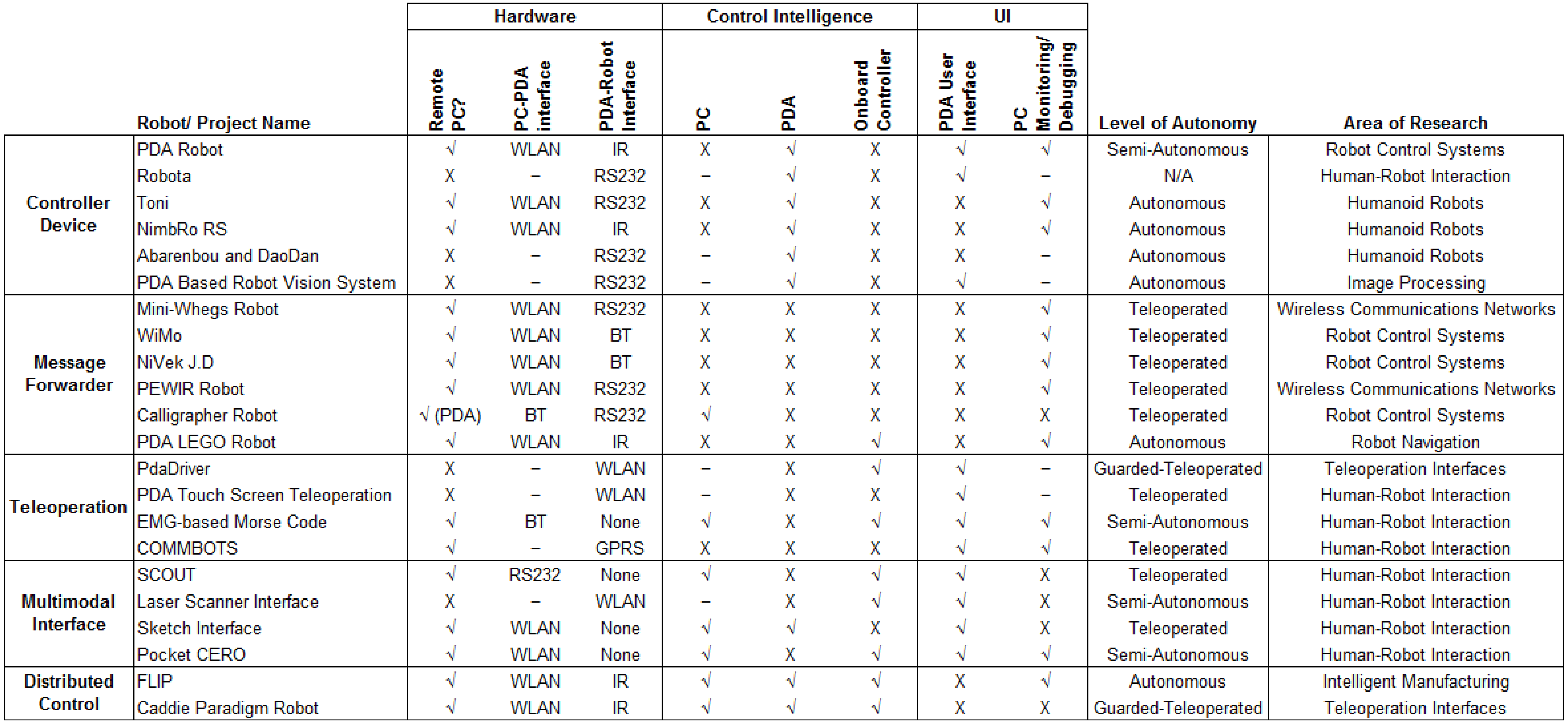
3.10 PDA Disadvantages

Having decided to use a PDA, the PDA is incorporated into the robot design. Many problems associated with using the PDA in its capacity within a project only becomes evident after the fact. This section summarises the different problems using a PDA as mentioned by researchers in past projects.

When sending control messages to the robot there is an inherent latency between the time sensing the command via a PDA interface and the robot carrying out the command. This means operators need special training to cope with the delays (Fong et al., 2003). The PDA screen has a low contrast when in power saving mode making it difficult to read the object on the screen without first activating the screen again by tapping somewhere on the screen (Hüttenrauch & Norman, 2001; Keskinpala et al., 2003).

Several projects make use of the LEGO RCX brick and its available consumer IR port for communicating with the PDA. Using the LEGO RCX brick forces developing a platform-dependent IR library to allow the IR ports of the PDA and RCX brick to communicate. This entails creating a library for every PDA used. The choice of PDA should preferably be open and interchangeable (Jensen et al., 2005).

The limited resources (memory, processing power, processing speed) of the PDA when compared with a PC forced the authors in (Calinon & Billard, 2003) to implement only simple vision and speech recognition algorithms and those in (Jantz & Doty, 2002) to develop optimised algorithms for use on the PDA.



		Hardware			Control Intelligence			UI			
	Robot/ Project Name	Remote PC?	PC-PDA interface	PDA-Robot Interface	PC	PDA	Onboard Controller	PDA User Interface	PC Monitoring/ Debugging	Level of Autonomy	Area of Research
Controller Device	PDA Robot	√	WLAN	IR	X	√	X	√	√	Semi-Autonomous	Robot Control Systems
	Robota	X	–	RS232	–	√	X	√	–	N/A	Human-Robot Interaction
	Toni	√	WLAN	RS232	X	√	X	X	√	Autonomous	Humanoid Robots
	NimbRo RS	√	WLAN	IR	X	√	X	X	√	Autonomous	Humanoid Robots
	Abarenbou and DaoDan	X	–	RS232	–	√	X	X	–	Autonomous	Humanoid Robots
	PDA Based Robot Vision System	X	–	RS232	–	√	X	√	–	Autonomous	Image Processing
Message Forwarder	Mini-Whegs Robot	√	WLAN	RS232	X	X	X	X	√	Teleoperated	Wireless Communications Networks
	WiMo	√	WLAN	BT	X	X	X	X	√	Teleoperated	Robot Control Systems
	NiVek J.D	√	WLAN	BT	X	X	X	X	√	Teleoperated	Robot Control Systems
	PEWIR Robot	√	WLAN	RS232	X	X	X	X	√	Teleoperated	Wireless Communications Networks
	Calligrapher Robot	√ (PDA)	BT	RS232	√	X	X	X	X	Teleoperated	Robot Control Systems
	PDA LEGO Robot	√	WLAN	IR	X	X	√	X	√	Autonomous	Robot Navigation
Teleoperation	PdaDriver	X	–	WLAN	–	X	√	√	–	Guarded-Teleoperated	Teleoperation Interfaces
	PDA Touch Screen Teleoperation	X	–	WLAN	–	X	X	√	–	Teleoperated	Human-Robot Interaction
	EMG-based Morse Code	√	BT	None	√	X	√	√	√	Semi-Autonomous	Human-Robot Interaction
	COMMBOTS	√	–	GPRS	X	X	X	√	√	Teleoperated	Human-Robot Interaction
Multimodal Interface	SCOUT	√	RS232	None	√	X	√	√	X	Teleoperated	Human-Robot Interaction
	Laser Scanner Interface	X	–	WLAN	–	X	√	√	X	Semi-Autonomous	Human-Robot Interaction
	Sketch Interface	√	WLAN	None	√	√	X	√	X	Teleoperated	Human-Robot Interaction
	Pocket CERO	√	WLAN	None	√	X	√	√	√	Semi-Autonomous	Human-Robot Interaction
Distributed Control	FLIP	√	WLAN	IR	√	√	√	X	√	Autonomous	Intelligent Manufacturing
	Caddie Paradigm Robot	√	WLAN	IR	√	√	√	X	X	Guarded-Teleoperated	Teleoperation Interfaces

Table 3.1: Caparison of PDA-incorporated Robot Systems

3.11 Comparison of PDA Robots

Table 3.1 describes the PDA robots mentioned in the text, their use of technology and the division of processing within their respective designs.

In this table, WiMo is the only robot that doesn't use a PDA, but rather a Windows Mobile Smartphone. And the SCOUT robot makes use of a wearable PC.

Robota is the only robot listed that does not make use of a mobile platform in its design, therefore it has no *level of autonomy*.

From the literature reviewed it is interesting to see that much of the work in mobile robotics using a PDA fall into the human-robot interaction category. In this area some use the PDA to provide a novel remote control (teleoperation) device. Others use the PDA as a means of giving the user feedback from the robot and allowing user response and direction. And some use it as part of a multimodal set of I/O devices, where the PDA is an optional input device. Using the PDA as a means of getting user input and showing sensory information focus only on the available I/O built into the device. These designs do not make full use of the powerful processing capability of the PDA itself.

In humanoid robot research PDAs are becoming a lightweight and powerful processing alternative for the control of these robots. The PDA links directly to the base, becoming part of the robot platform. The link between the PDA and the robot base is either serial or IR depending on the platform controller used.

It is interesting to see how few projects make use of the PDA as part of a distributed control system. The PDA's processing capability is underutilized when it forwards control messages from the main (often remote) control station to the platform controller. Often these messages are motor specific commands for the individual motors and servos on the robot base, increasing the latency between sending a command and its execution.

The FLIP robot is one of only two examples of projects that use distributed processing in their designs. FLIP makes use of three processing elements in a distributed fashion. FLIP has intelligent processing implemented on all three (the RCX brick on the LEGO-based platform, the PDA itself as well as on the remote PC). Their reason for imposing this distribution was not a conscious effort but forced for two reasons. First the LEGO Mindstorms RCX brick is unsuitable to carry out multi-robot systems research, with severe limits on both memory and processing power and better equipped for purely reactive robot development. Secondly, the infrared port of the RCX brick only allows line of sight communication between robots restraining the possibility of using cooperative planning. Adding the PDA to their LEGO robot increased the memory and processing power of their platform as well as providing WLAN for long-range, out of sight, communications. Though the Caddie Paradigm project does not concern multi-robot systems, but rather teleoperation, it may be that they implemented some collision detection software on the PDA due to the limitations of using the LEGO RCX brick.

The WiMo and NiVek J.D robots have a wireless connection scheme between the robot base, the PDA (or Smartphone) and the remote PC. The use of Bluetooth to connect the robot base to the PDA allows the PDA to be portable or separate from the base. It does not have the need for line of sight such as the FLIP robot. Bluetooth allows omnidirectional communications up to about 30 m and is immune to objects such as walls. However, these two robots do not use the PDA as anything more than “a repeater” (Wolf, 2007). It forwards the basic directional control commands it receives from the remote PC to the microcontroller on the base of the respective robots.

On the other hand some projects suffer from trying to run computational and memory intensive routines solely on the PDA. This lack of resources leads to implementing incomplete systems, or settling for a less desirable output that can successfully be implement on the PDA.

Incorporating a wirelessly connected, remote PC is common to many projects, but with many only using the PC to give goal directives or motor commands. Some make use of a remote PC to preform debugging and logging of important parameters, while others implement a UI on the PC. Few use the PC’s processing power, speed and memory, implementing all the control software either on the onboard controller or sometimes only on the PDA.

3.12 Conclusions

This chapter showed the different uses PDAs fulfil within robot research. That it is not limited to a specific robot application or research area. Robots were grouped according to the PDA’s purpose within the different projects. Table 3.1 shows a comparison of the mobile robots reviewed, showing the different use and combination of technology and the use of secondary processing elements such as a remote PC. A discussion of the advantages and disadvantages of using a PDA as experienced by developers is in Section 3.9 and 3.10.

No related work was found on whether using a PDA to distribute processing and control is a viable proposition in terms of system performance. Neither on how such distribution can or should be done.

Part II

Framework

Chapter 4

Conceptual Framework

The phrase “power is nothing without control” applies to many areas in life and mobile robotics is no exception. Mobile robot control is structured according to control architectures. These govern the way in which a robot will react to its environment and more specifically the processes it will follow in order to react in a predefined way to its environment.

In Chapter 3 five different control strategies for robot control with the inclusion of a PDA was identified. These practical examples of PDA robots shows how the PDA can be used without harnessing any of its processing ability (as message forwarding device) and also cases where the PDA’s memory and processing power could not match up with the demands of the system.

This chapter seeks to answer the question of how a PDA should be incorporated within the robot system to be of greatest use and improve, if possible, system performance. In answering this question, a new distributed intelligence control framework is proposed for a wireless mobile robot, consisting of three processing elements – PC, PDA and mobile platform controller.

The new framework’s design is underpinned by four components as shown in Figure 4.1. First each of these components are discussed. Thereafter they are combined into the proposed framework. This framework aims to combine a practical and optimum use case for a PDA within a mobile robot system using a suitable control architecture.

4.1 Distributed Hardware

The three components chosen for the robot design is a PC, PDA and mobile platform. These devices are physically distributed in that there is no direct

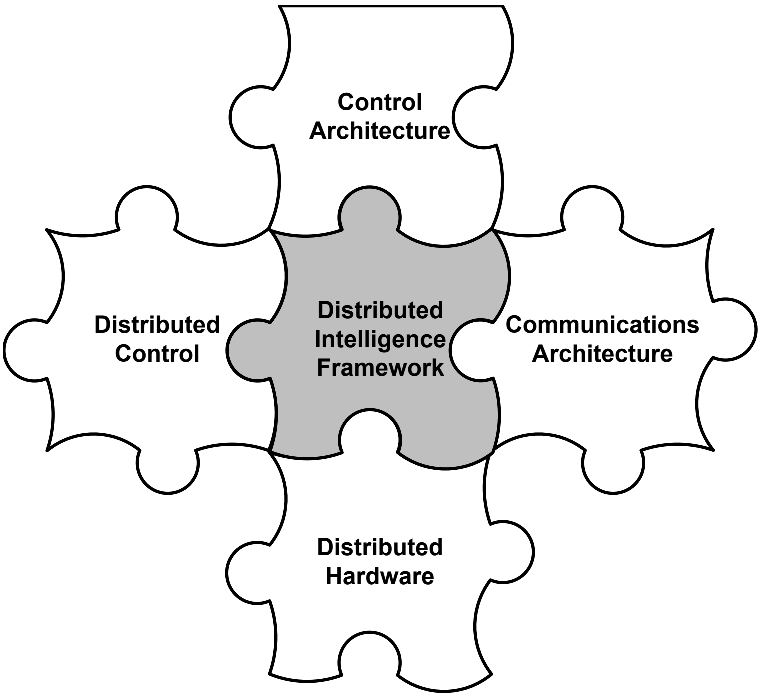

Control Architecture
Distributed Control
Distributed Intelligence Framework
Communications Architecture
Distributed Hardware

Figure 4.1: Building a Distributed Intelligence Framework

link between them. It is an aim not to have any direct, physical link as to provide the mobile robot the most flexibility to move in its environment and allow not only the test scenario described in Chapter 8 to be implemented, but scenarios in different fields using more or less capable mobile platforms.

Many projects described in Chapter 3 connected the PDA to the mobile platform through a direct if not always physical link. This would make the same architecture unusable for systems wanting to use the user interface of the PDA. The idea of the proposed framework is to be as flexible in application area as possible. Thus, the link between the PC and PDA as well as the PDA and mobile platform should be wireless.

4.2 Distributed Control

Most complex or more capable mobile robots require their processing to be distributed across multiple processing elements. This becomes of greater value as the capability of a mobile robot is increased and the overall system complexity increases, in other words the number of sensors, motors and processing components that must be integrated and coordinated (Yasuda, 2003). As an example, RHINO is an indoor mobile robot designed for entertainment and as a museum tour guide (Bugard et al., 1999). Its control tasks are distributed over three internal computers. It handles all safety and time-critical software components on board, with higher level software implemented on stationary computers connected to the mobile robot through a radio link.

The modular robot described by Firmansyah et al. (2007) implements each logic component within the system (main control, data acquisition and data processing) on a separate microcontroller based board, with the main control board connected wirelessly to a desktop PC.

Distributed computing is a type of segmented or parallel computing, but the latter term is most commonly used to refer to processing in which different parts of a program run simultaneously on two or more processors that are part of the same computer. While both types of processing require that a program be segmented, distributed computing also requires that the division of the program take into account the different environments on which the different sections of the program will be running.

The processing requirements of a mobile robot increases as the intelligence increases. Most robotic systems employ some form of processing, though purely reactive robots may or may not include a central processor. Distributed computing is the process of dividing the processing requirements of a system over multiple processing devices and/or threads. This can be done using multi-processors, multi-core or multi-computer systems.

Multiple processors provide the opportunity to take advantage of parallelism for improved throughput (Hu & Brandy, 1996). Heinrich & Honiger (1997) state that the solution methods from different applications can be parallelised in various ways. Due to parallel processing being applied in single areas in a robotic system, they distinguish between eight levels of parallel processing in robot control architectures: multi-robot level, robot level, kinematics level, control level, functions level, behaviours level, abstraction level and algorithm level.

Parallel processing along the abstraction level is closely related to how the control architecture is implemented within the system hardware. The functions level describes parallelism where planning and perception are distributed and carried out in parallel across multiple processing devices (Heinrich & Honiger, 1997).

Functional decomposition is the classical top-down approach to building systems. In this approach, the entire control task of a mobile robot is divided into subtasks which are then implemented by separate modules (Hu &

Brandy, 1996). This is the most commonly encountered area of distributed implementation.

4.3 Control Architecture

Selvatici & Costa (2005) describe a control architecture as a framework for determining the robot actuation and Mataric (1997) explains it as a set of principles for organising control systems, supplying a structure as well as constraints in the way control problems can be solved.

There are three schools of thought behind the design of the control architecture: hierarchical (or purely deliberative), purely reactive and a combination of the two, or hybrid architecture.

The implementation of any of these control architectures depend on how the system will be implemented and encompass control, software, communications, processing and hardware. Laengle & Lueth (1994) argue that the control of complex systems, such as a mobile robot, that consist of several executive subsystems/components can be divided into three different design classes or planning systems: centralised, distributed and decentralised.

Centralised control is the traditional means of directing robots. Here a single computer, robot or operator issues commands to a group of robots to follow, and as the number increases these can be subdivided into smaller groups, resembling a tree or pyramid in organisational structure. The major instability in this type of control structure is apparent in the event of a missing or damaged supervisor element. It is also difficult to handle abrupt changes. According to Laengle & Lueth (1994), distributed control implements a negotiation process between the executive components and execution by them. Decentralised control allows individuals to use local information to complete goals. Organisation in this case results from each collective action contributing to the overall goal. Here it is common for each executive component to assume larger responsibility and have some sensing and processing power, though not in all cases (Ranky, 2007).

4.3.1 Hierarchical

Figure 1.3 showed a typical representation of the hierarchical control architecture. According to this control architecture the robot gathers sensory data, followed by a planning stage that determines the next action which the robot subsequently performs. There is no direct link between sensing and acting and because the planning stage can be a lengthy process, robots designed using this architecture are often too slow to react in a dynamically changing environment (Murphy, 2000).

4.3.2 Reactive

An architecture introduced by Brooks (1986), the reactive control architecture does not make use of any deliberative components. Instead actions are coupled directly to sensor data. An arbitration scheme is employed, called subsumption, so that simultaneous and often conflicting sensor data can be governed in such a way as to achieve high-level goals. The reactive architecture produces a highly responsive robot. However, disadvantages include:

1. Difficulty to define all possible sensor combinations and reactions. This also makes the final design rigid against any change or addition to the system.

2. It is mostly used for lower-level intelligent processes/robots.

3. There is no representation of the operating environment.

4.3.3 Hybrid

The hybrid control architecture consists of three basic layers: a control layer (or reactive layer), a sequence layer and a deliberative layer. It facilitates the design of efficient low-level control with a connection to high-level reasoning. Figure 4.2 shows a traditional hybrid architecture.

A host of architectural designs based on the hybrid theory exists including 3T (Bonasso et al., 1997), AuRA (Arkin & Balch, 1997), Saphira (Konolige et al., 1997), DAMN (Rosenblatt, 1997), RHINO (Bugard et al., 1999), TCA (Simmons, 1994) and ATLANTIS (Gat, 1992). Each design defines the layered structure and interaction between layers slightly different.

Information flows up and down between the control layers. The deliberative layer sends plans to the sequence layer, which decomposes the tasks into subtasks and dispatches them, based on timing constraints, to the behavioural layer. The behaviours act to control the robot, sending sensor and status information back to the sequence layer. The sequencer informs the

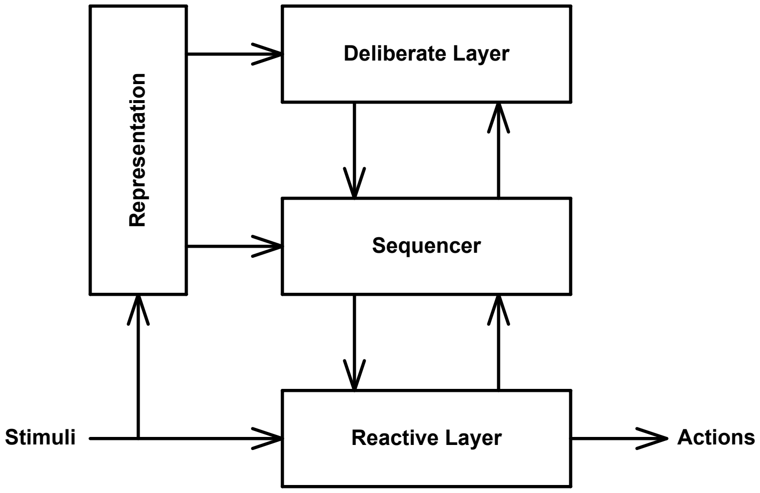


Figure 4.2: Traditional Hybrid Architecture (Santana, 2005)

deliberative layer when tasks are completed, and possibly abstracts sensor data for use by the deliberative layer.

4.4 Communications Architecture

Stasse & Kuniyoshi (2000) state that in the case of an architecture that is implemented as a distributed architecture, it will include the handling of communications. The communication infrastructure establishes how architectural levels interact, how architectural components communicate, how they access distributed data and how behaviours are executed (Posadas et al., 2007).

An improvement in performance cannot be achieved by solely increasing the number of processing units because the time necessary for communication or additional data administration may increase simultaneously (Heinrich & Honiger, 1997).

Distributing the hardware according to the proposed hardware architecture will lead to processing being distributed, but will also influence the communication architecture, which involves inter-layer communication as well as communications between distributed system components.

A hybrid architecture defines a model for interaction between soft real-time deliberation processes and hard real-time reactive control loops. Distributed implementation of hybrid systems have to cope with sensory information distribution among reactive and deliberative tasks. The different temporal requirements between reactive and deliberative tasks must be reflected in the communications system design (Coste-Maniere & Simmons, 2000). Reactive tasks have to cope with strict deadlines and have to lay over a predictable communication infrastructure. Deliberative tasks are related to soft deadlines and can deal with high performance and an unpredictable communication infrastructure.

Hybrid communication structures often make use of more than one communication medium and/or bus. Posadas et al. (2002) used a CAN bus for the reactive tasks and radio Ethernet for external IP communications. In such cases both communication infrastructures have to work together so as to ensure predictability at the reactive level as well as offering the needed information overlap between levels (Posadas et al., 2002).

4.5 Defining an Optimum Distributed Intelligence Framework

Implementing all intelligence on a single processing unit, such as a PDA, limits the functionality of the overall system. This puts restraints on navigation and planning capabilities due to memory and processing limits. Distributing the required processing of a mobile robot across multiple PEs would increase overall throughput, and using multiple processors provides the robot the opportunity to take advantage of parallelism for improved throughput (Hu & Brandy, 1996). Few projects have made use of a distributed control architecture.

Most projects that use this architecture for integrating a PDA will give the resulting architectural structure as shown in Figure 4.3, similar to the architecture modification needed for the FLIP project (Jensen et al., 2005).

Figure 4.3 shows the general architecture, shown in Figure 1.4, distributed over a robot platform, a PDA and a PC. The PDA is used to provide the wireless link between the behaviour based layer on the PC and the world interface layer on the robot platform.

The framework developed is an implementation of the three-layered approach of the hybrid control architecture, which provides event handling at different layers of abstraction through the use of behavioural, local and global planning layers. The framework allows for flexibility to take advantage of the resources available in a PDA robotic system. The distributed control

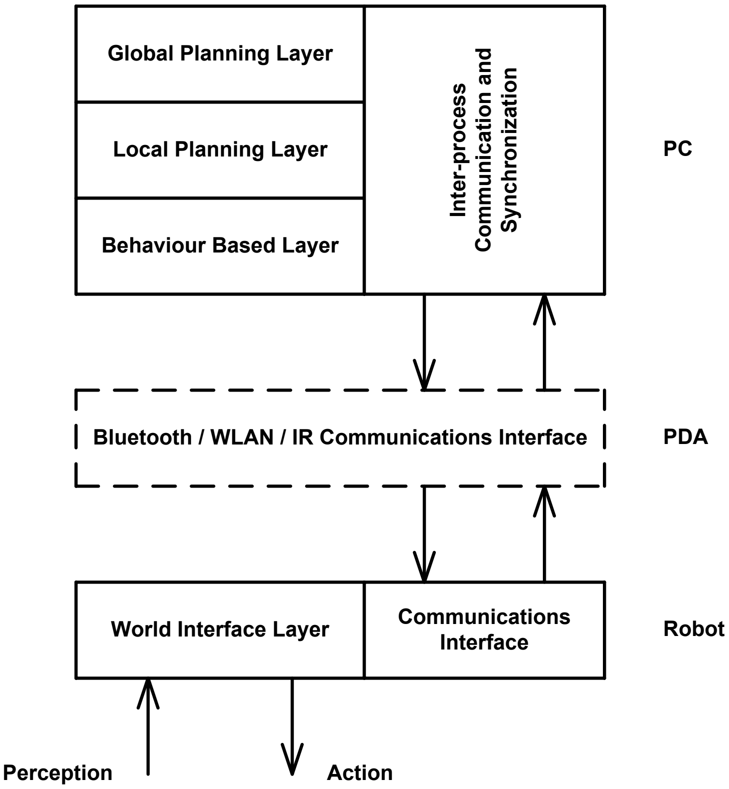


Figure 4.3: Typical Wireless Distributed Hybrid Architecture

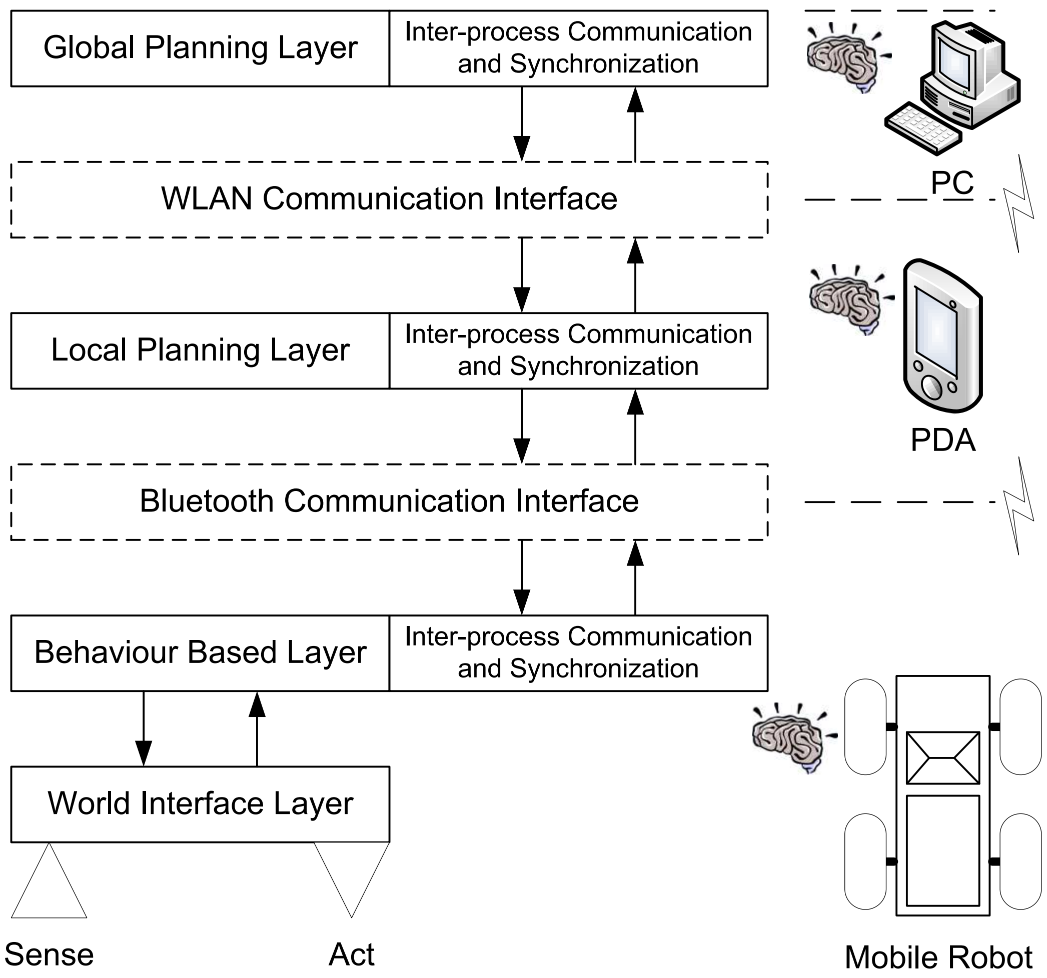

Global Planning Layer
Inter-process Communication and Synchronization
WLAN Communication Interface
Local Planning Layer
Inter-process Communication and Synchronization
Bluetooth Communication Interface
Behaviour Based Layer
Inter-process Communication and Synchronization
World Interface Layer
Sense
Act
PC
PDA
Mobile Robot

Figure 4.4: Distributed Control Framework

allows the robot to respond quickly to changing dynamics within the environment. The framework combines the four architectures: hardware, control, distributed control and communications using a PC, PDA and mobile platform with communication strategies suited to the three components.

4.5.1 Parallelism and the Hybrid Control Architecture

Parallelism is the process of doing multiple tasks simultaneously. Heinrich & Honiger (1997) show that there is no robot architecture that is perfectly parallel, with designers choosing single areas to implement in parallel. They distinguish between eight ways in which parallel processing can be implemented within robotics. One such way is parallelism on the abstraction level. The control architecture of robots is often divided into levels according to the degree of abstraction of processed data and response time. The layers, performing tasks which have different response times, are considered to work simultaneously and semi-independent from each other, and can be implemented on different PEs.

This hybrid control architecture promotes efficient low-level control together with high-level reasoning. Many architectures use the hybrid theory, and although the definition of the layered structure and the inter-layer mechanisms differ, the basic role of each layer is similar.

Global Planning Layer

The global planning layer, sometimes referred to as the planning or deliberative layer, decides how to achieve high-level goals by breaking them down to task level. The layer is implemented on a laptop PC and is responsible for tasks such as:

1. Synchronisation between tasks.
2. Monitoring task execution.
3. Interfacing with databases (local and distributed).

Local Planning Layer

The local planning layer, also called the sequencer or executive layer in some designs, sequences and monitors task execution. This layer is implemented on the PDA and is responsible for functions such as:

1. Decompose tasks from global planning layer into sub-tasks.
2. Construct control messages for the reactive layer.
3. Receive abstracted sensory data from the reactive layer.
4. Monitor and modify sub-tasks, according to data received, to achieve the goal.
5. Abstracting sensor information received from the reactive layer further, before passing it to the global planning layer.

Reactive Layer

The reactive or behavioural layer is implemented on the mobile platform. It forms direct links between what is sensed in the environment and relative actions or reactions. For example, if the robot senses an object it avoids it by acting immediately, without any authorisation from higher control layers. Reactions may also be governed by tasks received from the local planning layer. For example, the 'drive forward, 5m' command may cause the mobile platform to come to a halt when the sensors have indicated that the distance has been completed.

4.5.2 Communications

Wireless technologies such as WLAN and Bluetooth now come built-in in many PDA devices and PCs, making these technologies convenient to implement within a wireless control architecture.

The combination of the two wireless connection technologies was chosen for the proposed project due to their relative strengths. Bluetooth has the advantage over WLAN of having less overhead, making it computationally less expensive and using less power to transmit. WLAN on the other hand, though using more power, provides a means of securely connecting to networked devices through the TCP/IP protocol. It also has a much faster up and download speed than Bluetooth which is advantageous when transmitting video data.

This project will focus on an indoor mobile robot applied as an office assistant. It is important to define the application area for a robot because it influences the control tasks needed to be performed as well as the processing and communication requirements of the robot.

4.6 Conclusions

This chapter considered four building blocks in defining a new framework for distributing intelligence using the distributed hardware of PC, PDA and mobile platform. First, the different control architecture methodologies were examined with the hybrid control architecture ultimately chosen as the base of the new framework's design. Second, the theory of distributed control as well as supporting communications architecture is applied to the hybrid control architecture to enable it to support the chosen hardware components.

This also assists the distributed hardware devices to employ the idea of parallelism through abstraction across the layers of the control architecture in a wireless fashion.

The following chapters implement and test the proposed distributed intelligence framework. Chapters 5 and 6 develop a supporting mobile platform. Chapter 7 implements the chosen hybrid communications architecture. Chapter 8 describes a test scenario and describes how the distribution of intelligence is implemented and to what extent the new framework influenced performance.

Chapter 5

Mobile Platform

Mobile robots are robots that have the ability to move around in their environment, other than industrial robots which usually attach to a fixed surface. The task of roaming in a dynamically changing environment safely and accurately while carrying out meaningful tasks requires a mobile robot to have at least four basic components in its hardware architecture:

1. A hardware platform, or body, that houses all the other robot components.

2. A drive system that allows the robot to move from point A to point B. It usually consists of a combination of motors and either wheels, tracks or legs.

3. Several actuators and sensors that enable the robot to act on its environment as well as gain information from it.

4. A brain that interprets sensory information, navigates the robot within the working environment, controls actuator actions and monitors robot health. The robot's brain is often one or more PEs (processing elements) and can combine both onboard and remote PEs. Popular choices for robot PEs include desktop PCs, laptop PCs, embedded PCs, microcontrollers, and LEGO RX bricks.

The following section will discuss the attributes required for the mobile platform that is to form part of the PDA-based robot system as shown in Figure 3.18. Discussing each of the sub-systems in detail, the basic requirements are elaborated into the hardware and software design of the mobile platform. The chapter concludes with the completed mobile hardware platform.

5.1 Design Requirements

The design of the mobile platform must support the project's aim at investigating the use of a PDA within a mobile robot system. With this goal in mind, this section defines the design requirements, considerations and design for the mobile platform in terms of hardware and software. The software requirements and implementation being discussed in greater detail in Chapter 6.

The mobile platform is to be a student-built robot platform to meet the test requirements as discussed in Chapter 8.

Attributes that are important to consider in the design of the mobile platform include:

1. Simple construction
   - (a) Use a commercially available RC (remote control) car to provide an instant hardware platform
   - (b) Also provides drive motor and rechargeable battery
2. Mobile
   - (a) The mobile platform should not need to be attached to a stationary power supply or to the PDA
3. Safe
   - (a) Keep the mobile platform as small and as light-weight as possible to reduce chance of injury to people and/or property
   - (b) Make use of double sensing to ensure that if one sensory system fails the secondary system can still ensure save operation
4. Low cost
   - (a) Only implement basic drive, steering and sensing systems
   - (b) Make use of available components and use as few parts as possible
5. Collect as many types of operational data as possible
   - (a) Robot speed
   - (b) Distance to objects
   - (c) If robot has bumped into an object

   (d) Battery level

6. Communicate wirelessly with a PDA
   (a) Using Bluetooth as communications mechanism
   (b) Convey operational data through a custom communication protocol to a PDA

5.1.1 Operational Specification

With the design attributes in mind, the following operational specification can be defined for the mobile platform:

1. Measure the velocity in RPM.
2. Measure distance traveled in cm using an encoder.
3. Measure the battery voltages.
4. PWM drive for the drive motor.
5. Simulated PWM for the steering motor.
6. If the mobile platform bumps into an object, it should stop immediately.
7. Communicate through Bluetooth with a PDA.
8. Continuously gather operational data.
9. Package operational data into packets of information.
10. Interpret command messages from a PDA.

Figure 5.1 shows the basic block diagram of the mobile platform. Indicating the various subsystem needed to fulfil the operational specifications listed.

5.2 Implementation

Using a commercially available RC car, provides an instant robot platform. It also forces the choice of drive system and requires several modifications to transform it into a true mobile robot. This section describe these changes and hardware additions and how the brain of the mobile robot (a PIC18F4620)

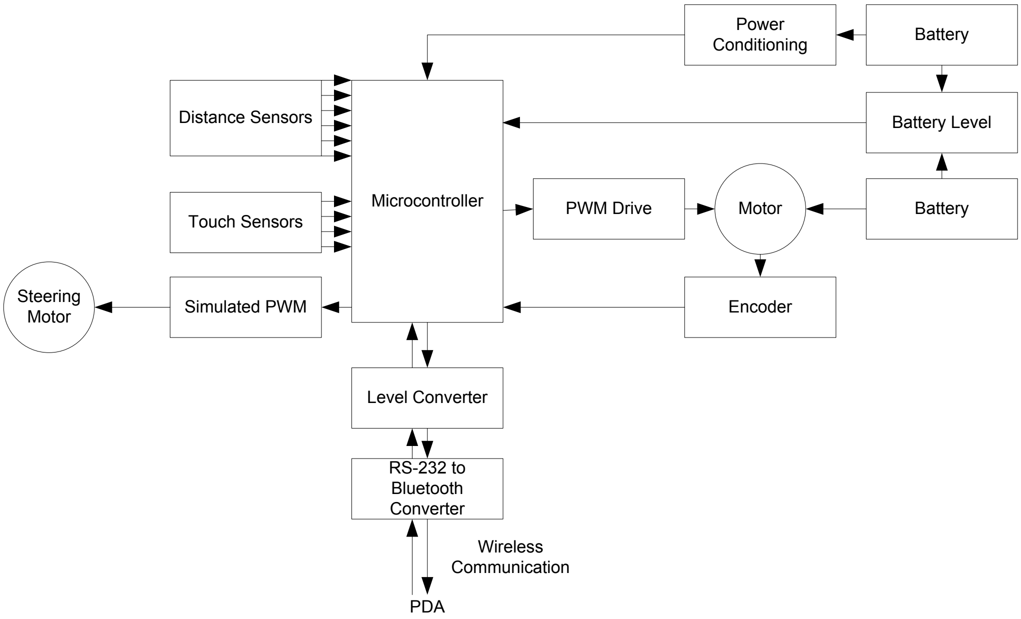


Figure 5.1: Basic Block Diagram

interfaces with the various subsystems. The basic system diagram in Figure 5.1 is expanded so that the subsystems include:

1. Distance sensing through IR (Infra Red) sensors.
2. Touch sensing through contact switches.
3. Steering through the use of a servo motor.
4. PWM drive of DC drive motor.
5. IR shaft encoder with encoder wheel for determining speed and distance traveled.
6. Power supply and battery monitoring.
7. Wireless Bluetooth communications.

5.2.1 Chosen Microcontroller

The PIC18F4620 microcontroller was chosen as the brain of the mobile platform primarily for:

1. its availability,

2. familiarity and,

3. availability of development tools.

The PIC18F4620 has the following features as taken from its datasheet (Microchip, 2007):

- High-current sink/source 25 mA/25 mA
- Three programmable external interrupts
- Four input on change interrupts
- Up to two Capture/Compare/PWM (CCP) modules
- Enhanced Capture/Compare/PWM module
- Master Synchronous Serial Port (MSSP) module that supports 3-wire SPI and I2C in master and slave modes.
- Advanced addressable USART module
- 10-bit, up to 13-channel analogue-to-digital converter module (A/D)
- Dual analogue comparators with input multiplexing
- Programmable 16-level High/Low-voltage detection module
- Priority levels for interrupts
- In-Circuit serial programming via two pins
- In-Circuit debugging via two pins
- Wide operating voltage range of between 2.0 V and 5.5 V

5.2.2 Motor Drive Unit

The RC car is driven forwards and backwards through a DC motor connected through a gearbox to the rear shaft. In its unmodified state, the RC car would move forward and backward at top speed, as is desirable for RC cars. But this feature is less desirable for mobile robots that must be able to execute more precise manoeuvres. To enable this, it is important to be able to control the speed of the robot. A PWM (Pulse Width Modulated) signal from the microcontroller provides an average voltage supply to the drive motor, which in turn regulates the motor's speed. But the microcontroller cannot interface directly to the drive motor, necessitating the use of a drive circuit that can handle the large currents drawn by the motor.

The old drive circuitry was removed from the RC car and replaced with a L298 Dual Full-bridge driver circuit. The L298 (Figure 5.2) is a popular motor driver IC that operates from 6 to 50 V, at up to 4 A total output current.

Figure 5.2: L298 Motor Driver IC

The Compact L298 Motor Driver kit from Solarbotics (Figure 5.3) makes this IC convenient to use and interface with (Solarbotics, 2008). It features:

- 6 to 26 V operation with 4 A total drive current
- Onboard user-accessible 5 V low-dropout regulated voltage
- Four motor direction indicator LEDs
- Schottky EMF protection diodes
- Small 4 cm square footprint
- Terminals for power and motor connections
- Socket pin connectors for logic interfacing

Table 5.1 shows a logic table for the three input signals (Enable, L1 (direction 1) and L2 (direction 2)).

Schematic Diagram

The schematic diagram in Figure 5.4 shows the use of one H-bridge of the dual H-bridge L298N IC as well as the auxiliary 5 V supply that is included in the kit.

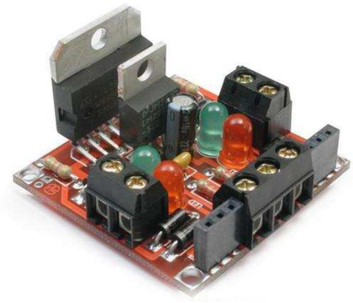


Figure 5.3: L298 Motor Driver Kit

Enable	L1	L2	Result
L	L	L	OFF
L	L	H	OFF
L	H	L	OFF
L	H	H	OFF
H	L	L	BRAKE
H	L	H	FORWARD
H	H	L	BACKWARD
H	H	H	BRAKE
PWM	L	L	PULSE BRAKE
PWM	L	H	FORWARD @ SPEED
PWM	H	L	BACKWARD @ SPEED
PWM	H	H	PULSE BRAKE

Table 5.1: Logic Table for the L298 Motor Driver Kit

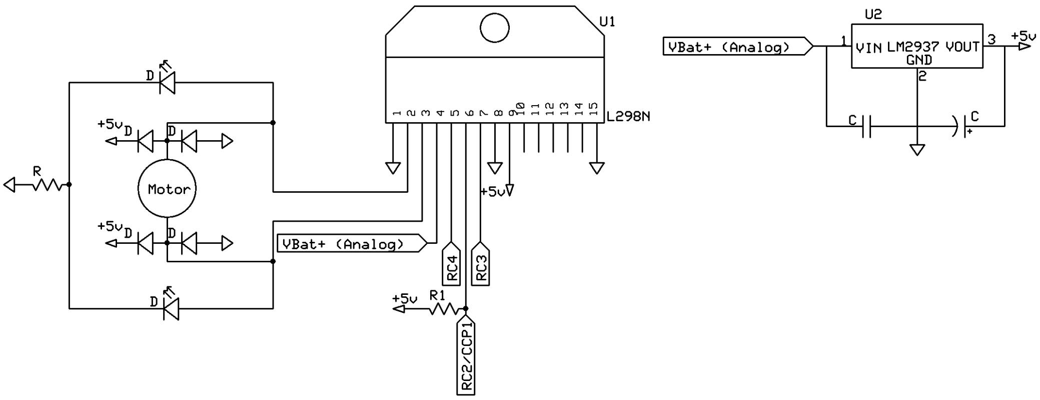


Figure 5.4: L298N Motor Driver Circuit and Auxiliary 5 V Supply

Software Requirements

The software for the drive system should control the three inputs of the L298N drive circuit in such a way to:

1. Drive the motor in a forward and backward direction.

2. Stop/brake the motor.

3. Control the speed of the motor.

Motor Power Control and Steering

PWM is a method used to convert a digital signal to an analogue signal. The duty cycle (proportion that a signal is high) of a square wave as output from the microcontroller is varied. This has the effect of producing a varying DC output by filtering the actual output waveform to get the average DC signal.

The diagram in Figure 5.5 shows the different average levels of DC output obtained from a 10%, 50% and 90% applied duty cycle.

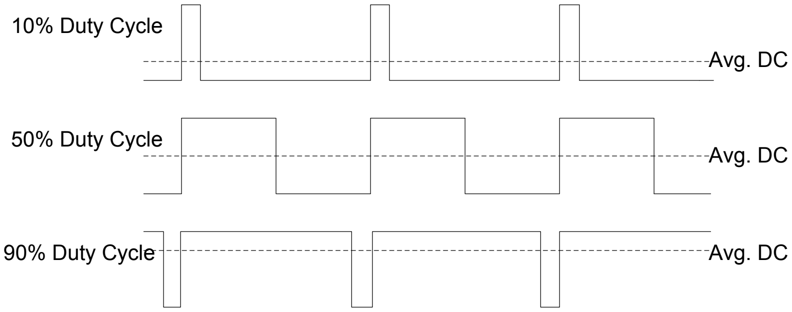


Figure 5.5: Pulse Width Modulation

The microcontroller produces PWM signals to control the drive motor speed as well as the servo motor's rotation.

5.2.3 Shaft Encoder

An optical shaft encoder is used to determine both the distance traveled as well as the speed in RPM of the robot. The encoder is made up of three parts as shown in the block diagram in Figure 5.6.

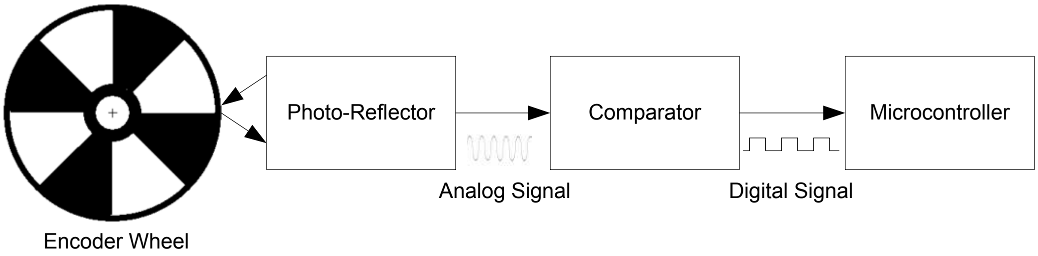


Figure 5.6: Encoder Block Diagram

The photo-reflector, the QRB1134 (Figure 5.7), is attached to the wheel shaft. A printed encoder wheel with black and white segments is placed on the inside of the wheel so that it turns with the wheel at the same speed. As both back wheels are presumed to turn at the same rate, only one of the wheels is fitted with an encoder.

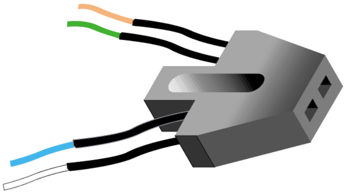


Figure 5.7: QRB1134 Photo-reflective Sensor

Figure 5.8 shows the placement of the QRB1134 sensor on the robot chassis as well as the encoder wheel.

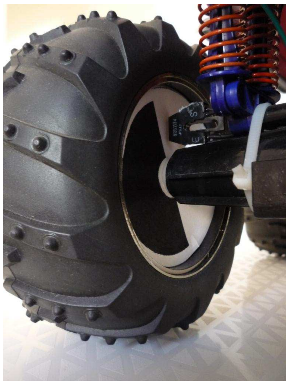


Figure 5.8: QRB1134 and Encoder Wheel Placement

If the photo-reflector is placed in front of a reflective surface (such as white paper), the output of the sensor is pulled low. When placed in front of a non-reflective surface (such as black paper) the output is high. Ideally, the microcontroller would react/trigger on the change between a high and low signal. But the QRB1134 is an analogue sensor that provides sensor data as indicated in Figure 5.9 with slow and irregular rising edges. The LM339 comparator is added to the design to produce the desired square wave signal.

As the wheel of the mobile platform turns, the encoder wheel turns with the same speed as the wheel. The black and white segments pass closely in front of the photo-reflector which is stationary on the wheel shaft. The square

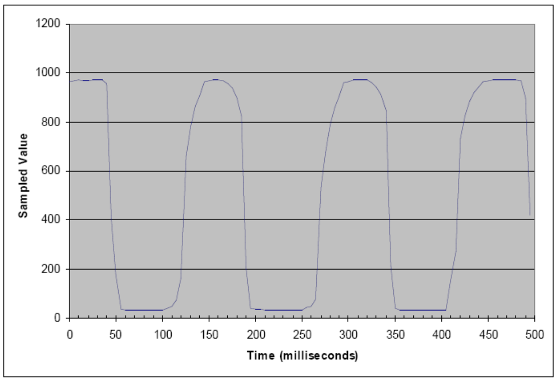


Figure 5.9: Sampled QRB1134 Sensor Data

wave signal produced can now be used by the microcontroller to determine the period of the square wave signal and therefore also the speed (inverse of period) of the mobile platform. The number of alternating black and white segments on the encoder wheel determines the number of pulses per revolution which is used in determining the rotational speed (RPM) of the robot as shown in equation 5.1.

$$RPM = \frac{60}{T \times pulses\ per\ revolution} \tag{5.1}$$

Schematic Diagram

Figure 5.10 shows the schematic diagram for the encoder subsystem.

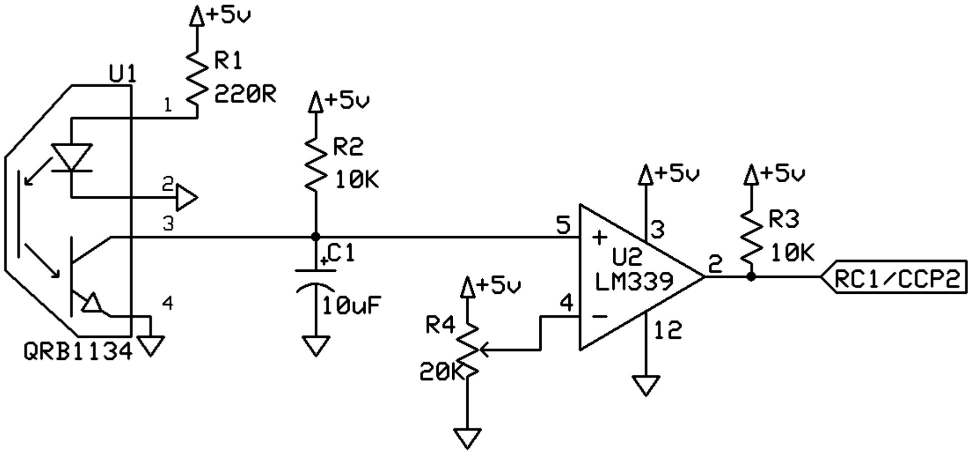


Figure 5.10: QRB1134 Encoder Circuit

Software Requirements

The software for the encoder subsystem must fulfill the following:

1. React to each rising edge of the square wave signal.

2. Evaluate the speed in RPM.

3. Calculate the distance traveled in cm.

5.2.4 Steering Motor

The RC car came with a 5-wire servo. This is a simpler type of servo motor in that the intelligence handled by standard servo's internally is done here by circuitry outside of the servo's casing. With two wires for the motor control and three for the potentiometer that determines its position. This servo was removed and replaced by a standard servo. With this type of servo, the motor will attempt to reach and hold a position as specified by the PWM control signal. The PWM signal's period is to be 20 ms. The neutral position has a pulse length of 1500 us and varies with +/-500 us to produce the minimum to maximum operation angle.

The chosen servo is a GSW S03T standard servo (Figure 5.11) and comes with a motor geared for extra torque. At 6v it is specified to deliver 111 oz-in torque at 0.27 sec/60 degrees.

Figure 5.11: GSW S03T Standard Servo

Schematic Diagram

Figure 5.12 shows how easy it is to connect the servo motor.

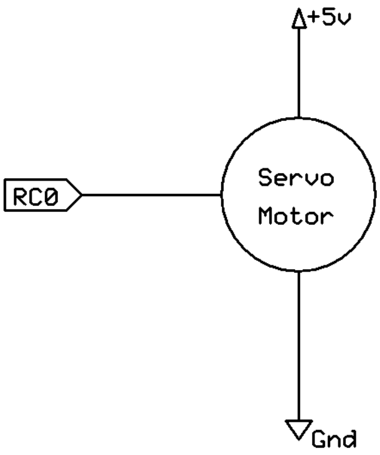


Figure 5.12: Servo Motor Circuit

Software Requirements

The software for the steering subsystem must fulfill the following:

1. Provide a 20 ms period.
2. Provide a suitably accurate PWM signal.
3. Utilise the maximum turning radius of the mobile platform.
4. Accept degree angle values as input.

The PWM generator is designed to generate an accurate pulse between 0% and 100% duty cycle, but the servo motor requires a range of duty cycles between 5% and 10% (1 ms/20 ms minimum and 2 ms/20 ms). With the typical PWM generator 8 or 10 bits, only a fraction of the bits to calculate the pulse width. This means that a lot of accuracy will be lost. The microcontroller must produce a PWM signal that does not cause such a loss in accuracy.

5.2.5 Power Supply

The mobile platform is powered by two battery packs each with eight 1.2 V cells. One battery pack is used to power the drive motor and is connected directly to the DC motor through the drive circuit as shown in 5.4. No feedback has been implemented so that as the voltage of the battery pack depletes, the motor will drive effectively slower at the same PWM output by the microcontroller. The auxiliary 5 V output provided by the L298 driver kit is used to power the steering servo motor. The second battery pack supplies the digital circuits.

Schematic Diagram

Figure 5.13 shows the digital power supply, reset and oscillator circuit as used with the PIC18F4620.

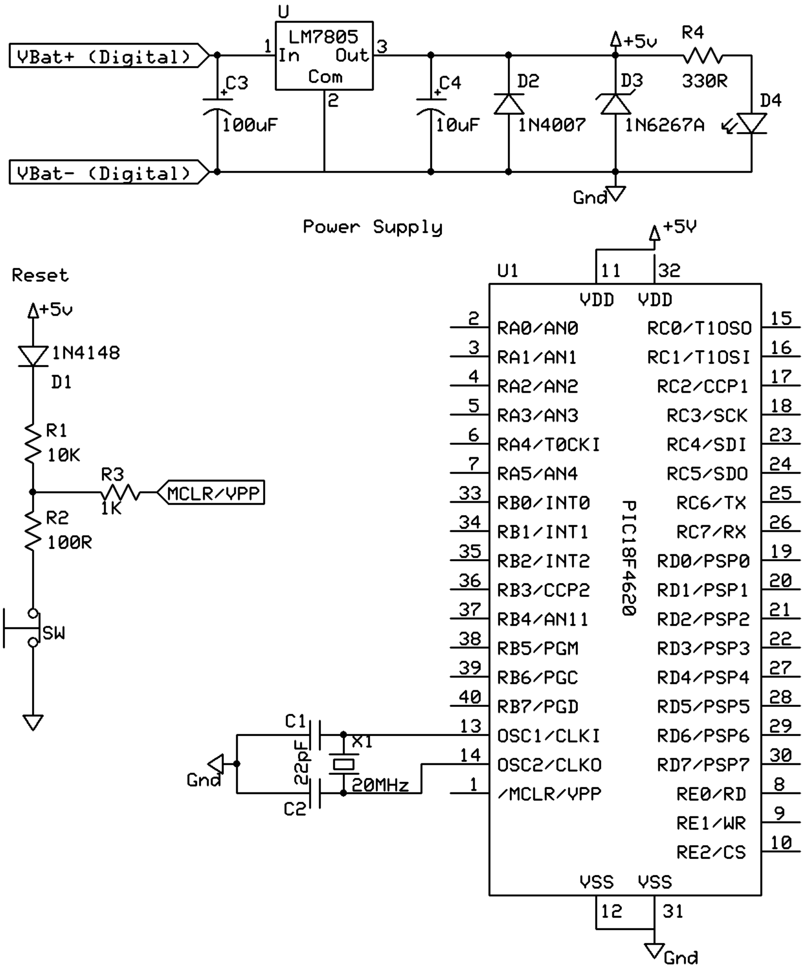


Figure 5.13: Power Supply, Reset and Oscillator Circuit

5.2.6 Battery Level Monitoring

Having a means to check the voltage levels of the two battery packs employed on the mobile platform is useful to:

1. Eliminate the need to check the voltage levels with a multi-meter.
2. Add possibility to programmatically determine when the power level is below safe operating levels which may cause erratic and unsafe behaviour by the software and thus the robot.

Battery levels are monitored by using voltage divider circuits as shown in Figure 5.14. This effectively scales the input voltage from the battery down to a range that the microcontroller can accept. Due to each cell with in the battery pack capable of being overcharged by as much as 25%. With each cell rather 1.2 V in the eight cell battery packs, this means that the maximum total voltage of either battery pack is 12 V.

$$V_o = V_i \times \frac{R_L}{R_L + R_i} \tag{5.2}$$

$$V_o = 12 \times \frac{220}{330 + 220} \tag{5.3}$$

$$V_o = 4.8V \tag{5.4}$$

The resistors are chosen to have a tolerance of 1%. The less tolerance in resistor values, the more accurate the voltage readings would be. The diodes prevent the circuit from shorting.

Schematic Diagram

The power levels of the two battery packs is monitored by using the circuits shown in Figure 5.14. The resistors chosen has tolerances of +/-1%.

Software Requirements

The software for the battery level measurement subsystem should take the analogue voltage readings from the two battery packs and convert the value to a range of 0 V to 12 V.

5.2.7 Distance Sensors

For a mobile robot to be able to move within its environment, it must be able to avoid colliding with stationary and moving obstacles in its path. Detecting obstacles at a relative distance from the robot body, allows the

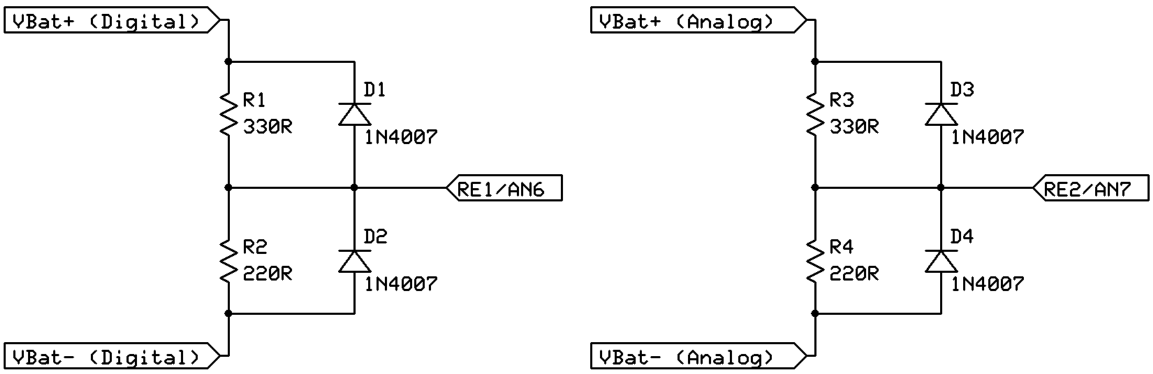


Figure 5.14: Battery Level Monitoring Circuit

robot the time to calculate its current position according to its knowledge of the environment it works in. Also whether a recalculation in path is needed to avoid a collision in the near future. For small robots with limited sensing ability, detecting obstacles is usually done by detecting distance to a reflective plane rather than having an exact knowledge of what type of object it senses.

The mobile platform makes uses six Sharp GP2D12 IR distance measuring sensors (Figure 5.15). From the GP2D12/GP2Y0A21YK datasheet, Figure 5.16 shows the internal block diagram of the GP2D12. From this diagram it is clear why no external components is required for operation. These sensors also do not require an external clock, but rather provide continuous readings at a fixed interval for as long as power is supplied to the sensor's power lines. It is important to note that a new output reading is only available every 40 ms, which is slow in microcontroller terms and that the first available reading after power up is an unstable reading and should be discarded by software.

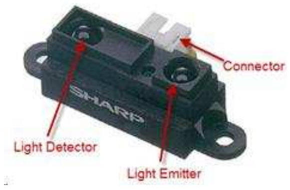


Figure 5.15: Sharp GP2D12 Distance Sensor

Three sensors is placed on the front, one on each side of the robot, but towards the front, and one on the back (Figure 5.17).

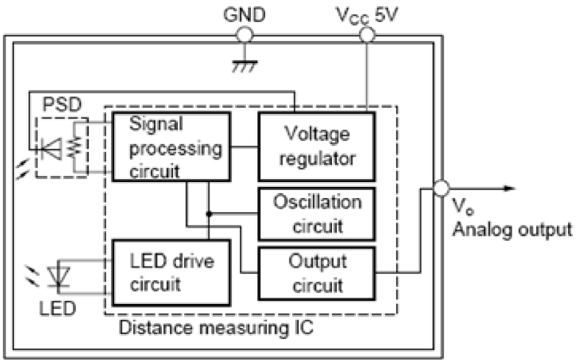


Figure 5.16: Internal Block Diagram of the GP2D12/GP2Y0A21YK

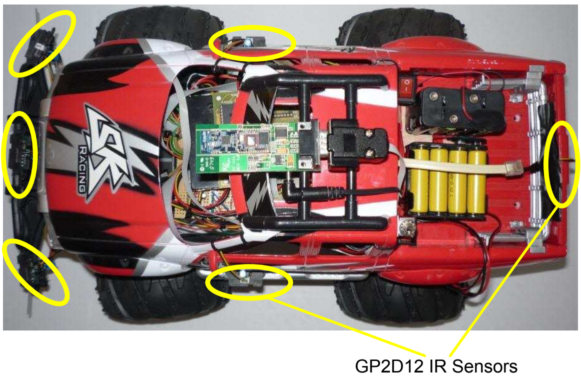


Figure 5.17: GP2D12 Sensor Placement

These sensors allow the robot to measure the non-electrical quantity, distance by transforming it into a relative voltage that the microcontroller can measure. The relationship between the voltage measured from the sensor and the measured distance is non-linear (a change in output voltage does not indicate the same change in distance) which makes the conversion more complicated.

The characteristic curve of five (the sixth sensor was found to be faulty and replaced) of the IR sensors were evaluated as shown in Figure 5.18.

From Figure 5.18 the operating range of the sensors can be seen to be between 10 cm and 80 cm.

The data used to create the curves in Figure 5.18 was obtained by obtaining the average of 1000 samples at distances ranging from 0 to 100 cm with increments of 0.5 cm at a rate of 10 kHz. Figure 5.19 shows the test setup used.

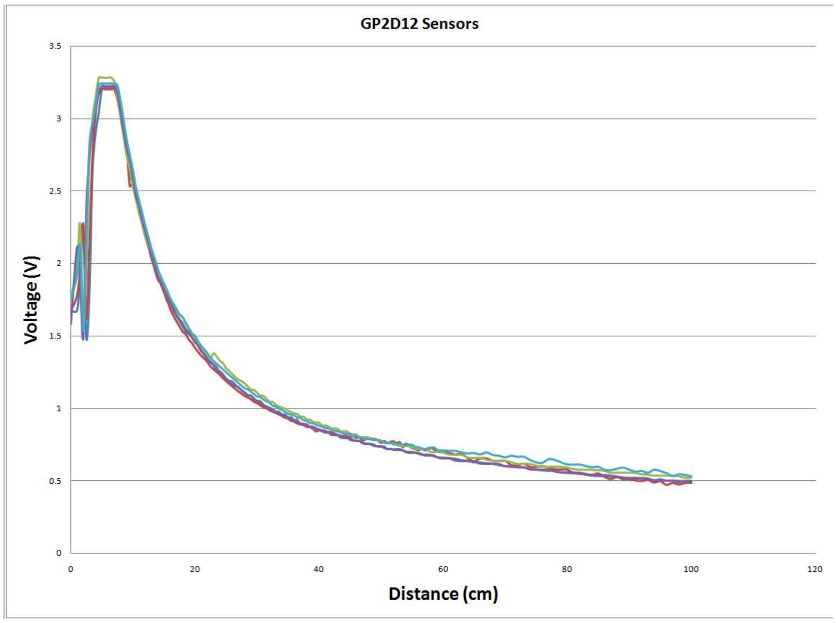


Figure 5.18: Characteristic Curves of 5 GP2D12 Sensors

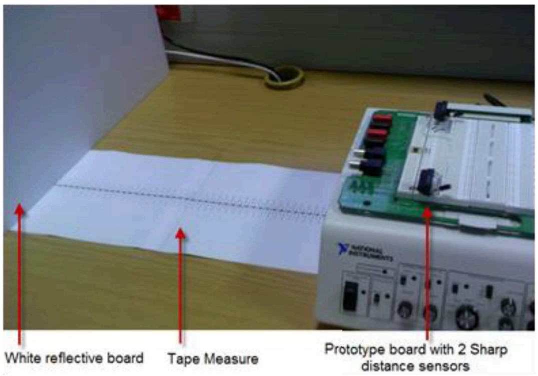


Figure 5.19: GP2D12 Experimental Setup

Two possible strategies can be followed to produce an adequate means of relating the measured voltage to distance namely an approximation function (described in Appendix C) and a lookup table. The latter was chosen for this implementation due to it requiring minimal processing power from the microcontroller, making it faster.

Schematic Diagram

The six SHARP GP2D12 IR sensors, their connections and placement on the mobile platform is shown in Figure 5.20.

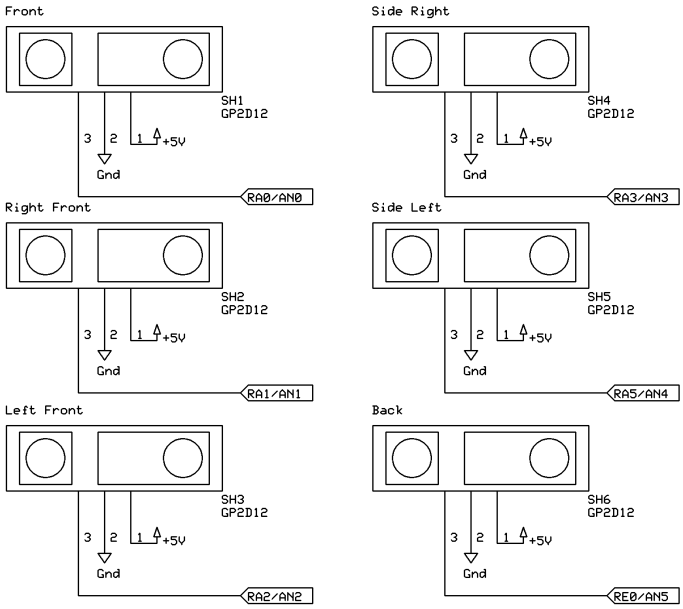


Figure 5.20: GP2D12 Distance Sensors Circuit

Software Requirements

The software should:

1. Limit the effect of noise by taking the average of multiple samples.

2. Implement a suitable lookup table for readings between 10 cm and 80 cm.

5.2.8 Touch Sensors

One of the requirements is for the robot to move safely in its environment. Contact switches with long "whiskers" is used to act as touch sensors for the mobile platform. Figure 5.21 shows the two touch sensors on the mobile platform's "bumper".

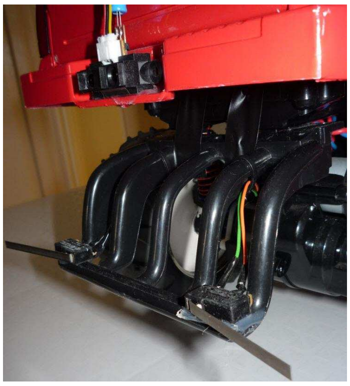


Figure 5.21: Rear Touch Sensor Placement

Schematic Diagram

The 4 contact switches, their connections and placement on the mobile platform is shown in Figure 5.22.

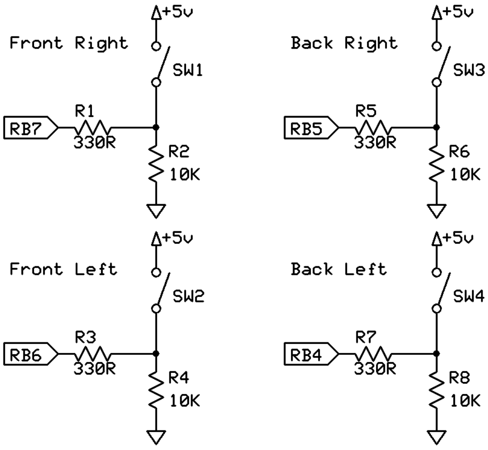


Figure 5.22: Touch Sensors Circuit

Software Requirements

The touch sensors form part of the safety system of the robot as well as the reactive control component. If the touch sensors bump against an object, the robot should stop immediately. This reactive control acts as a safeguard for any malfunction in more sensitive components such as the IR distance sensors as well as a communication loss with the PDA/PC.

Table 5.2: KC111 Bluetooth Serial Adapter Specifications

Hardware	**Specification**
Bluetooth	Version 1.2
Frequency	2.4 GHz
Bandwidth	1 MHz, 79 Channels
Frequency Hopping	1600 hops/sec
Typical RF Power	+18dB
Operating Temperature	-20 to +85 C
Input Voltages	4 to 10 VDC
Dimensions	32 mm width, 86 mm length
Power Consumption	typical 50 mA, max 200 mA
Chipset	Zeevo ZV4002

To enable the reactive response, contact on one of the touch sensors should trigger an interrupt routine. The software should make sure that the switches are properly debounced. The software must also be able to distinguish which touch sensor(s) is making contact.

5.2.9 Bluetooth Communication

The microcontroller on the mobile platform must be able to communicate in a bi-directional manner with a remote device. The distributed intelligence framework described in Chapter 4 requires the use of a Bluetooth communications link for this purpose. This link should enable the PDA to send control messages to the mobile platform to which the required actions can be preformed. The PDA must also be able to obtain sensory information through data request messages to which the mobile platform can respond with the requested data.

The PDA needs to communicate with the robot (or microcontroller-based platform of the robot system) in a bi-directional manner. The KC111 Wirefree Bluetooth module was chosen to Bluetooth-enable the PIC18F4620 microcontroller. It allows the microcontroller to use the SSP (Serial Port Profile) which defines how to set up virtual serial ports and connect two Bluetooth enabled devices. Table 5.2 lists the KC111 features.

The KC111 application has two modes, a “command” mode and a “bypass” mode. In the command mode, the host (microcontroller in this case) can issue specially formatted text strings called commands. These command strings can be used to configure the Bluetooth module or to manage a connection with a remote device. The KC111 supports a vendor-specific AT command set for this purpose. The KC111 can be the master or initiator of the Bluetooth binding process between itself and a remote device. The following has led to the KC111 together with the microcontroller-host to be implemented as the Bluetooth slave with respect to the PDA:

1. A Bluetooth master node may have several salve nodes. The mobile platform is not envisioned to communicate directly to any peer robots.

2. The PDA is more likely to interface with multiple or "swarm" robots.

3. Being the slave in the binding process means that the microcontroller only needs to handle a small set of the communication setup AT commands.

4. The mobile platform acts as the slave during the "bypass mode", where it can only respond to control or request messages, but never initiate a communication session itself. This conforms to the aim of the framework discussed in Chapter 4 with its hierarchial control strategy.

Once a connection is established, the application transitions to the bypass mode. In the bypass mode, bytes sent from the host will be sent over the Bluetooth link to the remote device (PDA). Any data received from the remote device will also be delivered to the host (microcontroller).

Schematic Diagram

Figure 5.23 shows the MAX232 serial driver circuit. The female DB9 connects to the male DB9 of the KC Bluetooth adapter module.

Software Requirements

The Bluetooth communications link between the PDA and microcontroller is far less stable than wired RS-232. With the addition of possible "junk" characters as well as unexpected breaks in communication. The software that interfaces with the KC111 Bluetooth Serial Adapter module should:

1. Be able to send and receive the needed AT command sequence for communication link setup.

2. Handle connection down and up events.

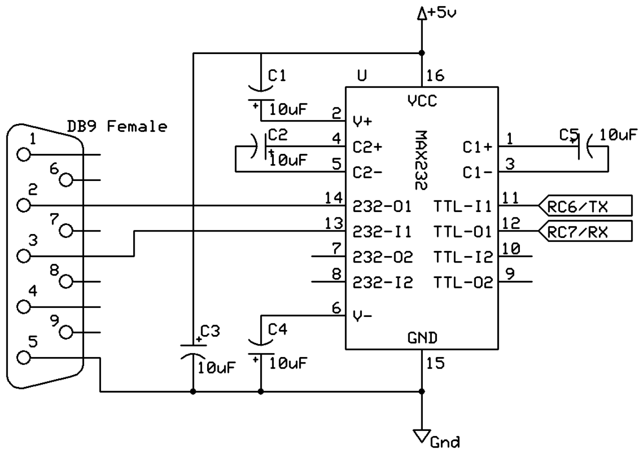


Figure 5.23: MAX232 Serial Driver Circuit

3. Distinguish and handle both KC111 AT commands and commands from the PDA.

4. Handle incomplete AT and normal control commands. By not allowing the software to hang waiting to receive complete message.

5. Handle "junk" characters especially right after initial communication link setup.

6. Fast response to allow for smallest possible robot response time.

5.3 Conclusions

The mobile platform consists of two parts namely the hardware as described in this chapter and the software needed to interface with the hardware (discussed in Chapter 6). Several design requirements were identified and developed into a list of needed attributes for the mobile platform's hardware. Simple construction, mobility, safety and cost was set as requirements. In order to be able to fulfill the needs of the reactive layer in the distributed intelligence framework designed in Chapter 4, the mobile platfrom also needed to communicate through Bluetooth and collect as many types of operational data as possible. Figures 5.24 and 5.25 shows a front and side view of the completed mobile platform hardware. The microcontroller and interface circuitry is all located inside the chassis of the RC car.

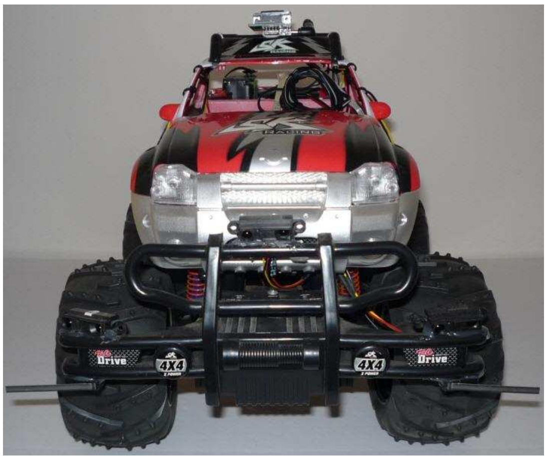


Figure 5.24: Front View of the Completed Mobile Platform

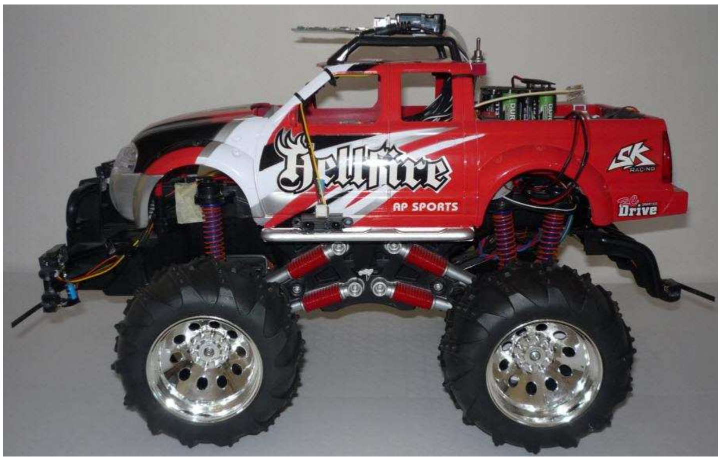


Figure 5.25: Side View of the Completed Mobile Platform

Chapter 6

Control Software

Most intelligent robots will have a brain. The mobile platform designed for this study in Chapter 5 uses the PIC18F4620 microcontroller as its processing element. This chapter discusses the software implemented on the PIC18F4620 microcontroller to integrate the hardware subsystems described in Chapter 5 into a mobile robot capable of interpreting sensory information, control actuator actions and monitor robot health.

Each subsystem in Chapter 5 described specific software requirements. Here the requirements are matched with the capabilities of the PIC18F4620 (the chosen microcontroller). The full source is included as Appendix B.

6.1 Software Considerations

The following has to be taken into account when implementing the routines needed to interface with the mobile platform hardware:

1. The PIC18F4620's limitations in terms of ROM and RAM, stack space, available IO, timers and interrupts.

2. The sensors and actuators have timing requirements

3. The microcontroller has necessary delays in sampling analogue inputs.

6.2 System Integration

Each of the hardware subsystems described in Chapter 5 interfaces with the PIC18F4620 microcontroller, the robot's brain. Thus each subsystem requires sensing and interpreting or control signals of some sort from the microcontroller. The following sections describes the software implemented to integrate each section with the microcontroller through hardware specific software routines.

6.2.1 Motor Drive Unit

The L298 driver requires a PWM signal from the microcontroller. TIMER 2, an 8-bit timer on the PIC18F4620 microcontroller, is used in CCP (Capture and Compare) PWM Mode with PR2 set to 255 and prescaler of 1.

The microcontroller internally handles the interrupt to toggle the output on pin RC2. TIMER 2 will count up to the value matching PR2 and then reset again to the value of PR2.

The following shows how the time base, the period, for the PWM signal is calculated and how the PR2 and prescaler values were chosen:

$$TIMER\ 2\ Clock\ Period = \frac{4 \times prescaler}{Fosc} \tag{6.1}$$

therefore:

$$TIMER\ 2\ Clock\ Period = 50ns \tag{6.2}$$

The time taken to interrupt TIMER 2 and toggle the output is dependant on the value of 'PR2' as shown in the following equation:

$$period = \frac{4 \times prescaler \times post\ scaler \times PR2}{Fosc} \tag{6.3}$$

which gives:

$$PR2 = \frac{period \times Fosc}{4 \times prescaler \times post\ scaler} \tag{6.4}$$

The diagram in Figure 6.1 shows the flow of the "driveMotor(int1 direction, int pwm_duty)" routine which sets the desired direction and duty cycle. Due to the DC motor drawing in excess of 4 times as much current when switching directly from forward to backwards direction, the motor is first stopped before a change in direction is made.

Robot Speed

The robot speed is controlled through setting the duty cycle for the PWM control. Slow, medium and fast speed correlates to 60%, 70% and 80% duty cycles. At this time no feedback is implemented so that as the voltage of the supply decreases or the robot encounters sloping terrain, the speed will vary

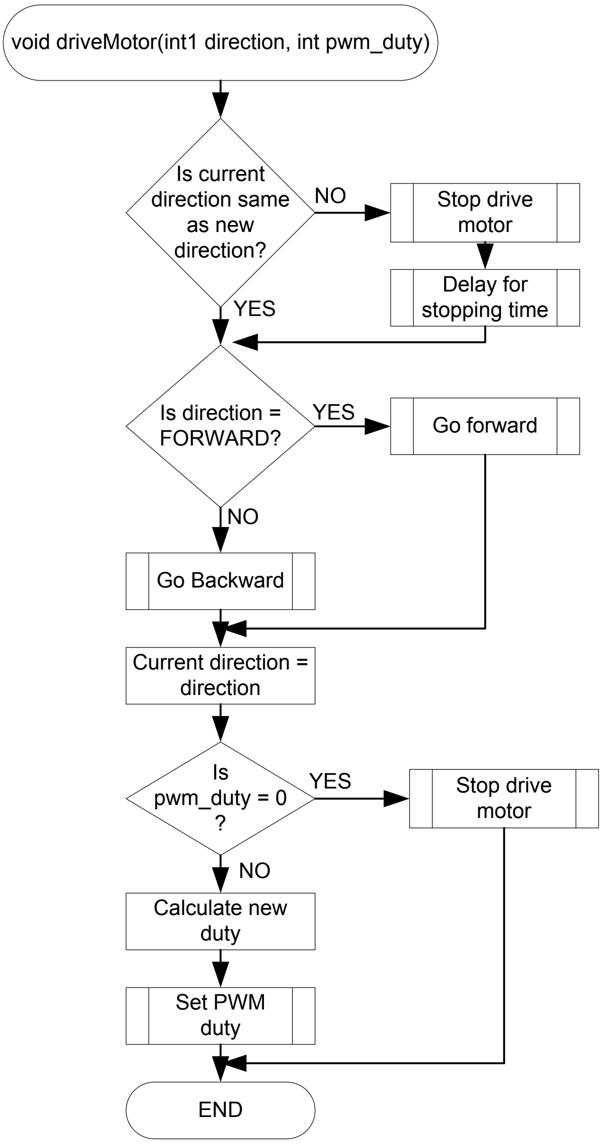


Figure 6.1: Drive Motor Control Flow Diagram

even though the duty cycle applied to the driving circuit stays constant. Because the mobile platform cannot come instantaneously to a complete stop, a stopping delay is used. The stopping delay will effectively be more than the set delay value due to the other interrupts interrupting the delay "Delay_ms(delay)" routine. The value of STOP_TIME has been modified to incorporate these induced delays through a trial and error basis.

6.2.2 Shaft Encoder

TIMER 1 together with the CCP2 module in CCP and “Capture on Rising Edge Mode” is used to determine both the speed and distance traveled by the robot. TIMER 1 is set to have a period of 1.6us per tick by setting the prescaler to 8. TIMER 1 provides the time base for the speed and related calculations of which distance is also implemented. TIMER 1 interrupt keeps track of the number of TIMER 1 rollover conditions that have occurred. The associated CCP2 interrupt will occur when the sensor on pin RC1/CCP2 of the microcontroller goes high (rising edge). On the interrupt the value of TIMER 1 is ”captured” to the CCP2 variable. When the interrupt is triggered, the routine saves the value of CCP2. On the next interrupt the two values CCP2-old and CCP2-new is used to calculate the time in timer-ticks that has passed between the first and second reading. The actual calculation is not done within the interrupt to save time. The calculation also only needs to be preformed when the microcontroller is asked to provide the distance and/or speed reading. Thus the CCP2 interrupt routine only sets a flag to indicate that a reading has been done and that the speed and distance calculations can be preformed with the data in the global buffers.

Figure 6.2 shows the flow diagram of the CCP2 interrupt routine.

6.2.3 Steering Motor

The servo motor used for steering requires a PWM signal with a 20ms period and a high time of between 1ms and 2ms. To obtain better accuracy TIMER 0 is used to produce the period signal and TIMER 3 is used to provide the high time of the signal, thus creating a TIMER-based PWM with TIMER 3 determining the duty cycle of the signal. Figure 6.3 shows the flow between the two timers. This gives the resulting wave form as shown in Figure 6.4. The “SetServo(signed int angle)” routine is used to make sure that the angle falls within the operating angle of the mobile platform as well as converting the angle received into a time delay to use with TIMER 3. A lookup table is used for this purpose.

6.2.4 Battery Level Monitoring

The battery levels are analogue measurements just like the distance sensor measurements. The sensors are connected to PORT E pins 1 and 2. See

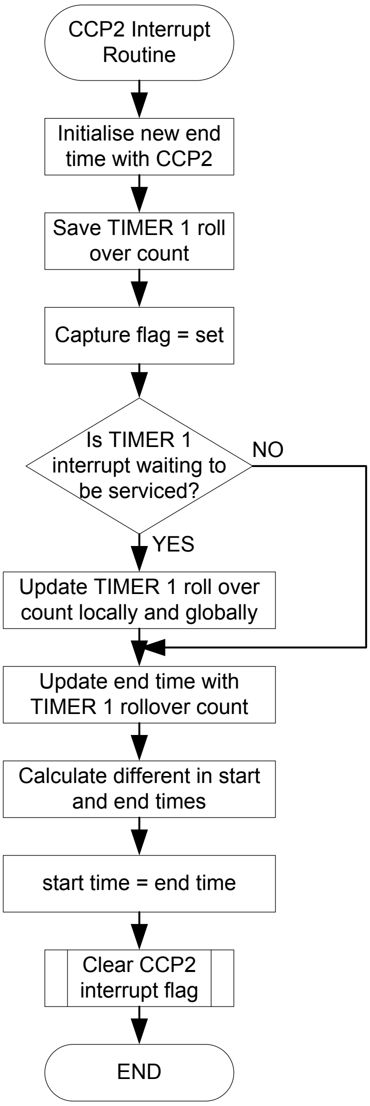

CCP2 Interrupt Routine
Initialise new end time with CCP2
Save TIMER 1 roll over count
Capture flag = set
Is TIMER 1 interrupt waiting to be serviced?
NO
YES
Update TIMER 1 roll over count locally and globally
Update end time with TIMER 1 rollover count
Calculate different in start and end times
start time = end time
Clear CCP2 interrupt flag
END

Figure 6.2: CCP2 Interrupt Routine Flow Diagram

Section 6.2.5 for more details on the analogue sampling procedure.

6.2.5 Distance Sensors

The Sharp GP2D12 IR sensor gives an analogue output. The analogue output of the six distance sensors are sampled by the microcontroller on PORT A pins 0 to 3 and 5 and PORT E pin 0. The A/D acquisition requirements for the PIC18F4620 needs the charge holding capacitor to be allowed to fully charge to the input channel voltage's level. This means that two input readings cannot immediately follow on one another and that some delay should

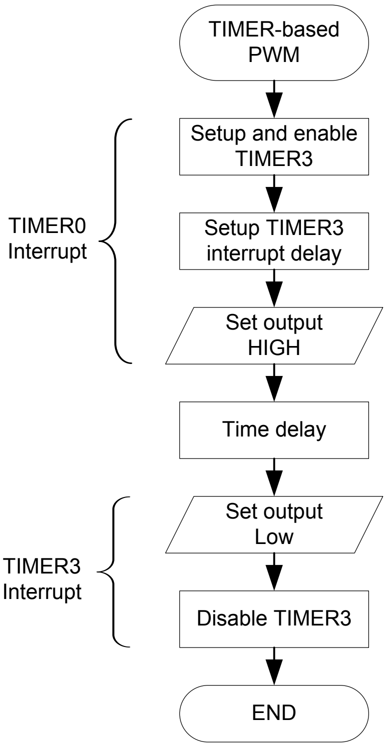


Figure 6.3: Timer-based PWM using TIMER0 and TIMER3

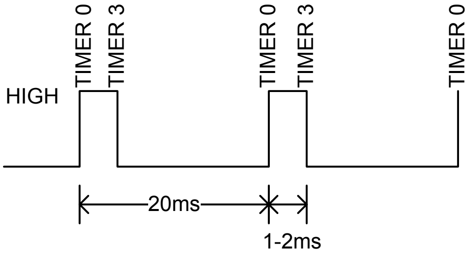


Figure 6.4: Timing diagram of Timer-based PWM

be introduced. A common method for implementing analogue measurement is shown in Figure 6.5.

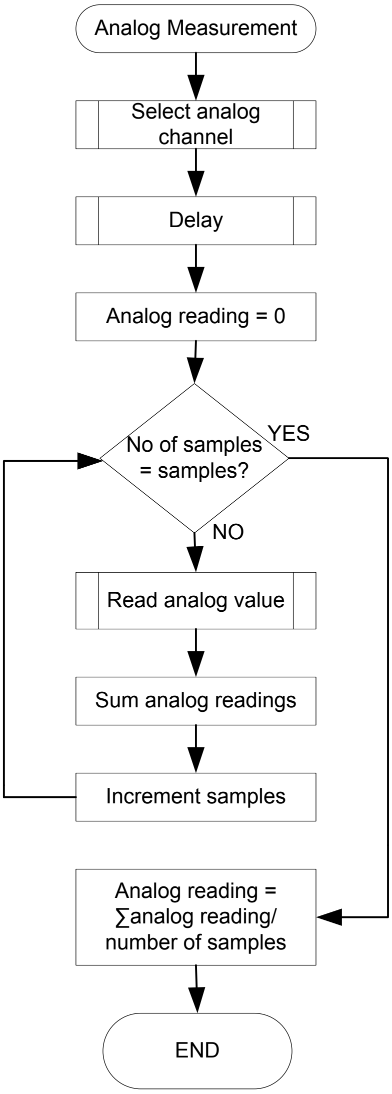


Figure 6.5: Typical Analogue Sampling Method

The nature of the GP2D12 sensors is such that a new reading is only available every 40 ms, which is slow in microcontroller terms. TIMER 0 is already used to provide a 20 ms time base for the timer-based PWM used for the steering motor (see Section 6.2.3). This delay can be made up of a number of smaller delays. The 20 ms delay is thus implemented to be a 20 times 1ms delay and can thus be used to also provide a time base for the analogue sampling routine. The routine incorporates filtering and the needed sampling delay without calling a delay function within the interrupt.

The flow diagram in Figure 6.6 explains the operation of the A2D sampling functionality as implemented as part of the TIMER 0 interrupt routine. Each interrupt is a "tick" on which part of the A2D process takes place. Once a series of "ticks" is completed, the process start again for the next analogue channel in the list.

For each analogue channel specified, ten samples is taken. The highest and lowest value (potential spikes) is eliminated from the data series and the remaining eight readings is averaged. With the current implementation, each distance sensor reading takes 13 ms. Thus new readings for all six sensors is available after 78 ms. This time would decrease if TIMER 0 was set to interrupt more frequently than 1 ms. As the distance sensors themselves are only able to produce a new output every 40 ms, the current implementation would pick up at least every second output with the averaging being done well within the 40 ms time at 1 sample every 1 ms.

One way of calibrating this type of sensor is by measuring the voltage output of the GP2D12/ GP2Y0A21YK at given fixed distances. Once this information has been experimentally obtained it can be placed within a constant lookup table. The resultant table of data is used by a routine in the program to calculate the distances according to the measured voltage. This is the method currently implemented.

The second way is to create a representative equation to describe the relationship between the voltage read and distance measured. This method is described in more detail in Appendix C.

6.2.6 Touch Sensors

The touch sensor code is implemented using the PORT B interrupt on change feature of PORT B. This will cause the routine linked with the interrupt to execute every time a sensor makes or unmakes contact that is connected to the upper nibble of PORT B (bits 4 to 7). If any of the sensors is touching an object, the drive motor is stopped immediately. Without properly debouncing the sensors, each make and unmake will be detected multiple times. The flowchart in Figure 6.7 shows the software design of this subsystem.

The above code, placing a delay within the interrupt routine, would not be good practice and would interfere with the effective functioning of the other interrupts used. The delay, which is used to debounce the touch sensors needs to be implemented without delaying the exit out of the "ContactSensors"

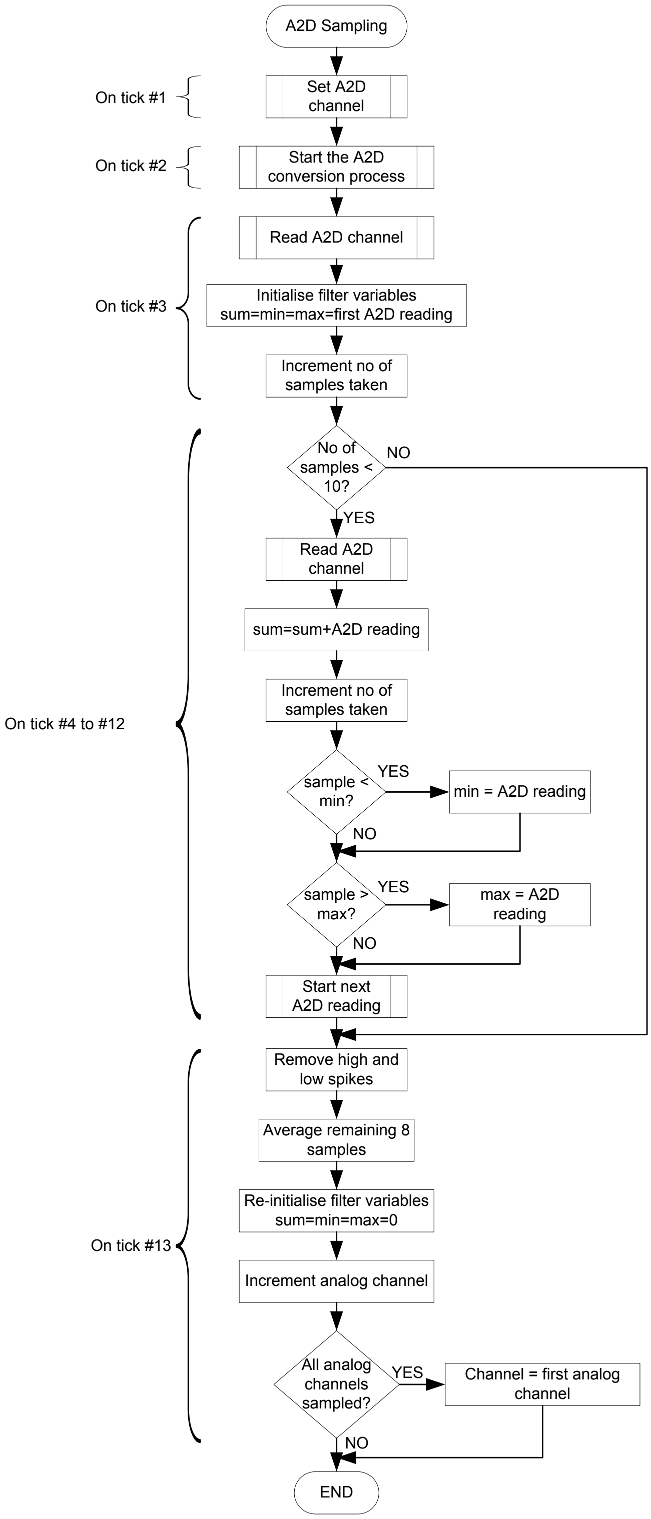

A2D Sampling
On tick #1
Set A2D channel
On tick #2
Start the A2D conversion process
Read A2D channel
On tick #3
Initialise filter variables sum=min=max=first A2D reading
Increment no of samples taken
No of samples < 10?
NO
YES
Read A2D channel
sum=sum+A2D reading
On tick #4 to #12
Increment no of samples taken
sample < min?
YES
min = A2D reading
NO
sample > max?
YES
max = A2D reading
NO
Start next A2D reading
Remove high and low spikes
Average remaining 8 samples
Re-initialise filter variables sum=min=max=0
On tick #13
Increment analog channel
All analog channels sampled?
YES
Channel = first analog channel
NO
END

Figure 6.6: Analogue Sampling through TIMER0 Flow Diagram

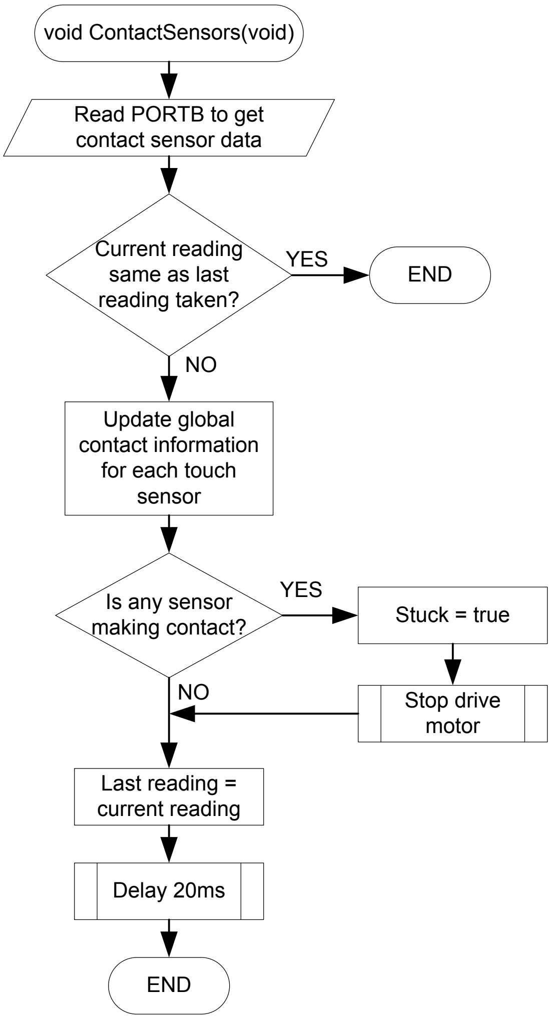

void ContactSensors(void)
Read PORTB to get contact sensor data
Current reading same as last reading taken?
YES
END
NO
Update global contact information for each touch sensor
Is any sensor making contact?
YES
Stuck = true
Stop drive motor
NO
Last reading = current reading
Delay 20ms
END

Figure 6.7: Touch Sensors Flow Diagram

interrupt routine. To accomplish this, the delay itself is also implemented using an interrupt where a global variable is set. The "ContactSensors" routine can then interrogate this variable for whether contact debouncing has been completed or not. Thus, the "ContactSensors" routine will be called more times than needed due to switch bouncing, but the reading will only be updated if the debouncing time has elapsed. For this purpose TIMER 0 interrupt is used. TIMER 0 is already providing a 20 ms base for the steering servo motor which is suitable for the switch debouncing timing as well.

6.2.7 Bluetooth Communication

On top of the communications link, a custom communications protocol is implemented as described in Chapter 7.

To ensure the fastest possible response, both transmit and receive functionality is implemented on the microcontroller using interrupts. This means that the program is never waiting to send or receive data.

The communications link preforms the following functions to transmit and receive data:

1. Use circular buffers to store incoming and outgoing characters.
2. Transmit a single character at a time using the transmit interrupt, INT TBE.
3. Buffer outgoing characters in a transmit buffer.
4. Send a message (also called packet) of a specified length.
5. Receive a single character at a time using the receive interrupt, INT RDA.
6. Buffer incoming characters in a receive buffer.
7. Receive packet data until the specified terminating sequence is encountered.
8. Interpret and respond to AT commands from the Bluetooth module.

Transmit and Receive Buffers

Two independent buffers is implemented. One to hold incoming characters and the other to store data ready to be sent to the remote device or KC111 Bluetooth adapter module. Circular buffers is used so that characters are placed in increasing buffer positions and, if a character is placed in the final buffer slot, the next character will be placed in the first position. When the buffer wraps round to the start, any data that was previously stored in those slots are overwritten. It is therefore critical for the buffer to be large enough not to wrap around in a single message. Variables is used to keep track of the next available position in the buffer into which or from which data is to be stored/taken. The receive buffer is implemented as an array that can hold up to 90 bytes. The circular buffer must be able to hold three consecutive incoming messages from the KC111 module. 90 bytes accommodates for this worse case condition. The transmit buffer is implemented as an array of 64 bytes.

TX Interrupt (INT TBE)

The “SerialTransmitISR()” routine is executed on a transmit interrupt. The routine will transmit a single character from the transmit buffer. If a character has been transmitted, the interrupt flag is set so that the next character can be transmitted when the interrupt routine is serviced. Once there is no more characters in the transmit buffer, the transmit interrupt is disabled.

Transmit Buffering

The “BufferedPutc(char c)” function places a single character in the transmit buffer and enables the transmit interrupt, INT TBE.

Sending Packets

The “SendPacket(int packet_len, byte *packet_ptr)” routine is used to send a packet (a complete message that includes header, payload and termination characters) to either the PDA (when in “bypass” mode) or to the KC111 Bluetooth adapter module (when in “command” mode). The function puts a packet with length equal to packet_len into the transmit buffer using the “BufferedPutc(char c)” function which in turn makes use of the serial interrupt service routine “SerialTransmitISR()”.

RX Interrupt (INT RDA)

The serial interrupt handler, “SerialReceiveISR()”, places incoming bytes into the receive buffer. The serial interrupt, INT RDA, is triggered by a character in the USART buffer. The routine reads a single character and places it in the receive buffer.

Receive Buffer

The “BufferedGetc()” function returns a single character from the receive buffer. It will wait for a certain time for an incoming character to become available in the receive buffer, but implements a time out so that the microcontroller program will not hang in waiting for a character to be received.

This feature is important due to Bluetooth carrier loss possible at any time, even in the middle of receiving a packet. The watch dog timer would also be an option to use for this purpose, but as the implementation responds to incomplete packets with a corresponding error codes, the CPU should not be reset.

Receiving Packets

The “ReceivePacket(byte *packet_ptr, int *length_ptr)” routine is used to receive a packet. A start character is used to distinguish between packets from the Bluetooth module and those originating from the PDA. It is also used to eliminate the “junk” characters received after the communication link has been set up or any random ”noise” received at other times from being processed as the start of a packet, causing an error packet to be returned. A pre-amble could also be used for this purpose, but as the start character is already useful in determining the origin of the packet, it is used instead. The routine will wait for a valid start character of either a PDA command or KC111 command. The routine attempt to receive a complete packet (ending with a CR and a LF) by retrieving bytes from the receive buffer and copying them to ‘packet_ptr’.

AT Command Interpretation

The commands received from the Bluetooth adapter module is in the form of AT command strings. Two start character options and terminates with a CR followed by a LF. An example is ”AT-ZV -CommandMode-”. Two methods for determining which command string was received from the KC111 Bluetooth adapter module was investigated namely string comparison and using a hash function.

The first uses a number of constant string arrays for each of the possible commands that can be received from the KC111. These values must then be copied from ROM to RAM in order to be compared with the received command string. This means that one would potentially do as many comparisons as there are command types possible before the actual command can be identified.

A hash function transform a variable sized input and returns a fixed sized value, called the hash value. This is why a hash function was implemented rather than a string comparison or variant. The hash function takes the variable length command string as received from the KC111 adapter and calculates a hash value that is unique for the input value, or at least unique with the commands used. These hash values can then be used in a simple switch statement to determine which command was issued and to react accordingly.

Hash Function

"DjbHash(char *str, unsigned int len)" implements the chosen hash algorithm, the DJB hash:

```
unsigned long djb_hash(char* str, unsigned int len)
{
   unsigned long hash = 5381;
   unsigned int i    = 0;

   for(i = 0; i < len; str++, i++)
   {
      hash = ((hash << 5) + hash) + (*str);
   }
   return hash;
}/* DJB Hash Function */
```

6.2.8 Initialisation

A number of initialisation functions is implemented to initialise:

1. PIC18F4620 pins.
2. The analogue ports and associated clock.
3. The timers.
4. The interrupt sources.

PIN Initialisation

Each port on the PIC18F4620 microcontroller has an associated TRIS register used to set a pin as either an output or input. All the pins on PORT A is configures to be inputs as they are used as analogue inputs. PORT B is configured to be all inputs, with the high nibble is used with the interrupt on change for the touch sensors. Pins 0 to 6 of PORT C is outputs, with pin 7 as the RX input. PORT D is all outputs. PORT E pins 0, 1 and 2 is inputs as they are used for analogue inputs. The following code shows the use of the "set_tris_x" routine to initialises the pins:

```
void InitPins(void)
{
    set_tris_a(0b11111111); /* Port A is all analogue inputs. */
    set_tris_b(0b11111111); /* Port B is all inputs */
    set_tris_c(0b10000000); /* Pins 0-6 are outputs. C7 (RX) is input. */
    set_tris_d(0b00000000); /* Port D is all outputs. */
    set_tris_e(0b00000111); /* Port E is analogue inputs. */
} /* InitPins */
```

Analogue Initialisation

Eight analogue inputs is used with reference voltage of +5V. A prescaler is added to the clock used for the A/D conversions. The needed analogue initialisation is done using the "setup_adc_ports" and "setup_adc" routines as demonstrated in the following routine:

```
void InitAnalog(void)
{
    setup_adc_ports(AN0_TO_AN7|VSS_VDD);
    setup_adc(ADC_CLOCK_DIV_32);
}/* InitAnalog */
```

Timer Initialisation

Four timers is used. TIMER 0, used in conjunction with the TIMER 0 interrupt is initialised to interrupt every 1ms. The CCP1 module is set to PWM mode and the associated timer, TIMER 2, is initialised with no prescaler and PR2 value of 255. The CCP2 module is configured in the Capture and Compare on Rising Edge mode. The associated timer, TIMER 1, is set to have a divide by eight prescaler. TIMER 3 is initialised within the TIMER 0 interrupt routine as part of the timer-based PWM implemented to control the steering servo motor. The following routine implements the needed timer initialisation:

```
void InitTimers(void)
{
    //set_timer0(40536); //set up timer 0 to interrupt every 10ms
    set_timer0(63036); //set up timer 0 to interrupt every 1ms
    setup_counters(RTCC_INTERNAL,RTCC_DIV_2);

    setup_ccp1(CCP_PWM);
    setup_timer_2(T2_DIV_BY_1, PR2_VALUE, 1);

    setup_ccp2(CCP_CAPTURE_RE);
    setup_timer_1(T1_INTERNAL|T1_DIV_BY_8); //period of timer 1 = 1.6us per tick
    set_timer1(0);
}/* initTimers */
```

Interrupt Initialisation

The serial receive interrupt, TIMER 0 and TIMER 1 interrupts as well as the CCP module interrupts is initialised and cleared on reset as follows:

```
void InitInts(void)
{
    enable_interrupts(int_rda); //serial receive interrupt
    enable_interrupts(INT_RB); //contact sensor interrupt
    enable_interrupts(INT_TIMER0); //servo and contact sensor time base

    clear_interrupt(INT_TIMER1); //ensure interrupt flag bit is cleared
    clear_interrupt(INT_CCP2); //ensure interrupt flag bit is cleared
    enable_interrupts(INT_CCP2);
    enable_interrupts(INT_TIMER1); //used to provide time base for speed related calculations

    enable_interrupts(GLOBAL);
}/* InitInts */
```

6.3 Conclusions

This chapter looked at the software considerations in implementing the sensory and actuator requirements laid out in Chapter 5. The different hardware systems described is integrated into a complete functioning mobile robot platform through the control software implemented for each system component. The mobile platform being complete, Chapter 7 can build on top of the basic communications implemented by specifying a custom communications protocol to enable data transfer between the PDA and mobile platform.

Chapter 7

Communications Software

The communications platform, or architecture, is designed to compliment the framework discussed in Chapter 4. To enable this the platform has to support not only the distributed nature of the hardware involved in the design, but also the distributed control components, as well as the hybrid control architecture it is based upon.

The hybrid control architecture requires that information be passed in various levels of abstraction between each of the three control layers. The opposite must also be supported, where high level decisions made by the highest, deliberative layer, must be communicated down the layers in more detailed control instructions to the bottom, world interface layer. This means that the communications platform must support communications in a bi-directional manner. With each control layer implemented on a separate and physically distributed hardware device, bi-directional as well as wireless communication links must be established between the three devices: PC, PDA and mobile platform. Figure 7.1 shows the three sections of the communications platform and how this relates to the three hardware devices.

This chapter describes the communications protocol used to transfer system data to and from the mobile platform. First, the WLAN link between the PC and PDA is described. After this the Bluetooth communications link between the PDA and mobile platform is discussed. A custom protocol is employed between the PDA and mobile platform. It is used to deliver actuator commands to the mobile platform's controller while also enabling the request for specific sensory information to be returned to the PDA. The chapter concludes by discussing the various types of packets defined in the custom protocol.

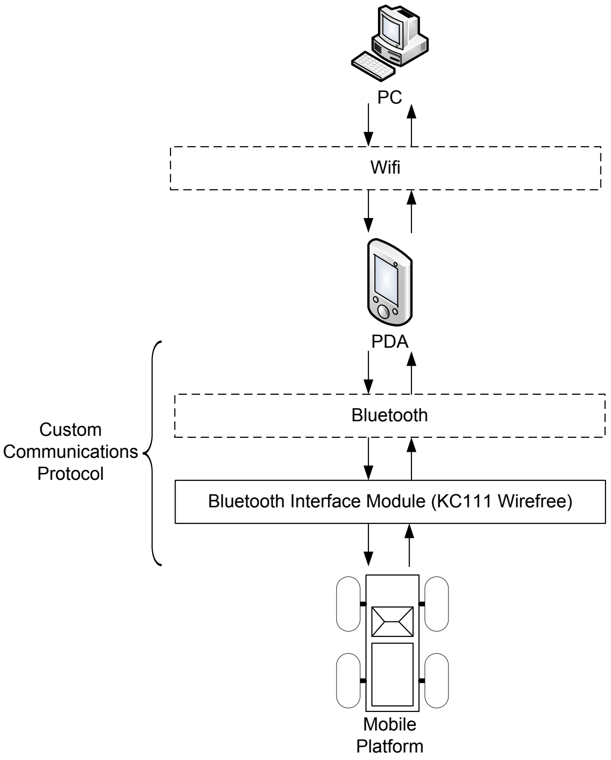

PC
Wifi
PDA
Custom Communications Protocol
Bluetooth
Bluetooth Interface Module (KC111 Wirefree)
Mobile Platform

Figure 7.1: Communications Platform with Chosen Hardware

7.1 Communicating over WLAN

Communication between the PDA and PC is implemented using the TCP protocol over a socket-based network connection. The network connection uses WLAN to connect the two devices together. This allows the PDA to move around freely while still being able to communicate with the PC. An application on each device enables communication to take place – a TCP server on the PC and a TCP client on the PDA.

The TCP server is a desktop application running on the .NET Framework. It has a user interface to display messages from clients, while the underlying code handles network connections using a TCP server. To increase the application's performance it uses three types of threads, as illustrated in Figure 7.2.

The main thread handles the update of the user interface. The TCP server's operations, including starting the server and listening for incoming connection requests, are performed in another thread. Once a connection request arrives a worker thread is created to handle data communications and

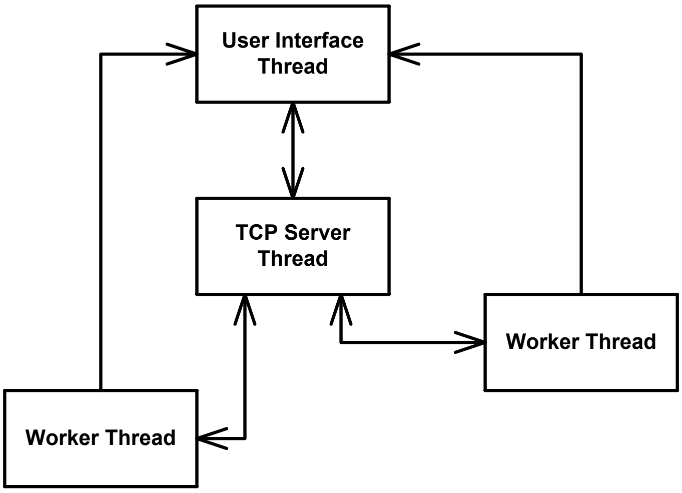


Figure 7.2: TCP Server Threads (Yang et al., 2007, p. 183)

any user interface updates for this specific request. Many worker threads can handle connection requests simultaneously, thus improving the performance of the application.

To create a TCP server which listens for incoming connection requests the following code is used:

```
1  private TcpListener tcpListener;
2  private TcpClient tcpClient;
3  private Thread listenThread;
4
5  private void StartServer()
6  {
7      tcpListener = new TcpListener(IPAddress.Any, 4400);
8      listenThread = new Thread(new ThreadStart(ListenForClients));
9      listenThread.Start();
10 }
11
12 private void ListenForClients()
13 {
14     tcpListener.Start();
15     while (true)
16     {
17         tcpClient = tcpListener.AcceptTcpClient();
18         Thread tcpClientThread = new Thread(new ParameterizedThreadStart(ReceiveData));
19         tcpClientThread.Start(tcpClient);
20     }
21 }
```

The “TcpListener” class contains the functionality to listen for incoming connections from remove devices. An instance of this class is created which runs in its own thread and waits for incoming connections on port 4400 any IP address on the PC (lines 5–10).

The thread continuously waits for incoming connections (line 15). When a connection is detected a worker thread is started to handle communications between the PC and remote device (lines 17–19). To display the messages being received the code below is used.

```
1  private void ReceiveData(object client)
2  {
3      TcpClient tcpLocalClient = (TcpClient)client;
4      NetworkStream stream = tcpLocalClient.GetStream();
5      byte[] message = new byte[1024];
6      int bytesRead = 0;
7      while (true)
8      {
9          try
10         {
11             bytesRead = stream.Read(message, 0, 1024);
12         }
13         catch
14         {
15             break; // connection lost
16         }
17         if (bytesRead > 0)
18         {
19             ASCIIEncoding encoder = new ASCIIEncoding();
20             this.Invoke(new MethodInvoker(
21                 delegate() { lbHistory.Items.Add(
22                     encoder.GetString(message, 0, bytesRead)); }
23                 ));
24         }
25         else
26         {
27             break; // no data
28         }
29     }
30     tcpLocalClient.Close();
31 }
```

The worker thread initialises a network stream to read data from the PDA (lines 3–6). Next it continuously reads from the stream until no more data is received or the connection is lost (lines 7–29). Data is read in chunks of 1024 bytes and displayed in a listbox control on the user interface. For the worker thread to update the user interface a special “Invoke” method must be used, which allows interaction with user interface controls from threads other than the main thread (Wigley et al., 2007, pp. 412–421). Finally, the connection is closed once no more data is received (line 30).

The above code demonstrates how the PC receives data from the PDA. However, it is also necessary to send data from the PC back to the PDA. The code below shows how to do this.

```
 1  private void SendData()
 2  {
 3      NetworkStream stream = tcpClient.GetStream();
 4      ASCIIEncoding encoder = new ASCIIEncoding();
 5      byte[] message = encoder.GetBytes(txtMessage.Text);
 6      stream.Write(message, 0, message.Length);
 7      stream.Flush();
 8  }
```

Once a remote connection is available another network stream is used to write data back to the client (lines 3–4). The data to be returned can be generated programmatically or read from a textbox (line 5). To send the data it is written to the stream, which is then flushed to make sure all the data is sent (lines 6–7).

The TCP client on the PDA functions in a familiar way to the examples above. After connecting to the server a separate thread is used to send and receive data. This again improves the performance of the application. To connect to the server the following code is used.

```
 1  private TcpClient tcpClient;
 2
 3  private void ConnectToTCPServer()
 4  {
 5      tcpClient = new TcpClient();
 6      IPEndPoint serverEndPoint = new IPEndPoint(IPAddress.Parse("169.254.2.2"), 4400);
 7      tcpClient.Connect(serverEndPoint);
 8      Thread tcpServerThread = new Thread(new ThreadStart(ListenForServer));
 9      tcpServerThread.Start();
10  }
```

The “TcpClient” class used to connect to the server. It uses an IP address and port to determine the endpoint to connect to. In this example the IP address is fixed and has been hard-coded (line 6). A new worker thread is created on which communication with the server takes place (lines 7–9).

The sending and receiving of data works in exactly the same on the PDA as it does on the PC. A network stream is used to read data being received as well as to send data in return.

7.2 Communicating over Bluetooth

This section describes the communications link between the PDA and mobile platform. It shows the client implementation on the PDA and the master using a Bluetooth serial adapter module, the KC111 Wirefree.

7.2.1 KC111 Wirefree Bluetooth Serial Adapter

The PIC18F4620 microcontroller is the host of the KC111 Bluetooth module. It is this module that allows the microcontroller to send packet data to the PDA through a Bluetooth communication link. The KC111 starts of in "command mode" and communications is between the module and microcontroller until the communication link with the remote Bluetooth device (PDA) is established. The microcontroller is in charge of enabling the bonding to take place between the KC111 module and the PDA as well as setting the Baud rate and PIN requirements.

Once the KC111 is enabled to bond with a specific Bluetooth address, the address is stored in the device's bond table. There is no need to enable a bond with a particular device after the procedure was done once. Also the baud rate and security requirements will be set until cleared by the microcontroller. The current implementation has the KC111 as the master node, with the PDA having the client connecting to it. These roles could easily be swapped if desired.

The following sections describe the command sequence of the device specific AT commands (each appended with carriage return and line feed characters). Figure 7.3 shows the commands received by the microcontroller as soon as power is applied to the KC111 module. The microcontroller can enable the KC111 to bond with a number of known nodes by placing these nodes in the device's bonding table. The command flow between the microcontroller and the KC111 module to enable bonding with the PDA is also shown in Figure 7.3.

There is three instances when the KC111 Bluetooth module will either move from the "bypass mode" to the "command mode" or indicate a loss in communications with the remote host to the microcontroller. These cases are shown in Figure 7.4. Here, sending an escape sequence when currently in the "bypass mode" will place the KC111 module back into "command mode". When the Bluetooth connection is broken for whatever reason, the KC111 module indicates this through two command sequences. The one is given if the KC111 is currently in the "command mode" and the other if the KC111 module was in the "bypass mode".

During "command mode" a number of settings on the KC111 module may be configured. Figure 7.5 shows two of the more commonly used command sequences: changing the baud rate and setting the security level.

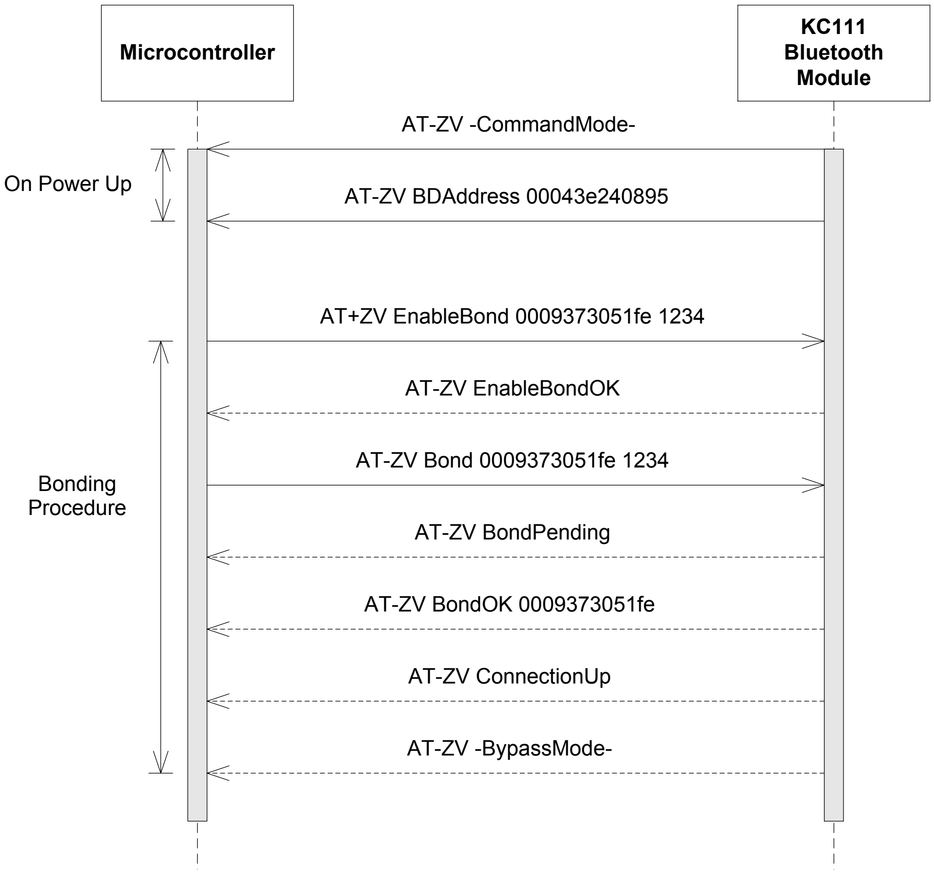


Figure 7.3: KC111 Module Power-up and Bonding Sequence

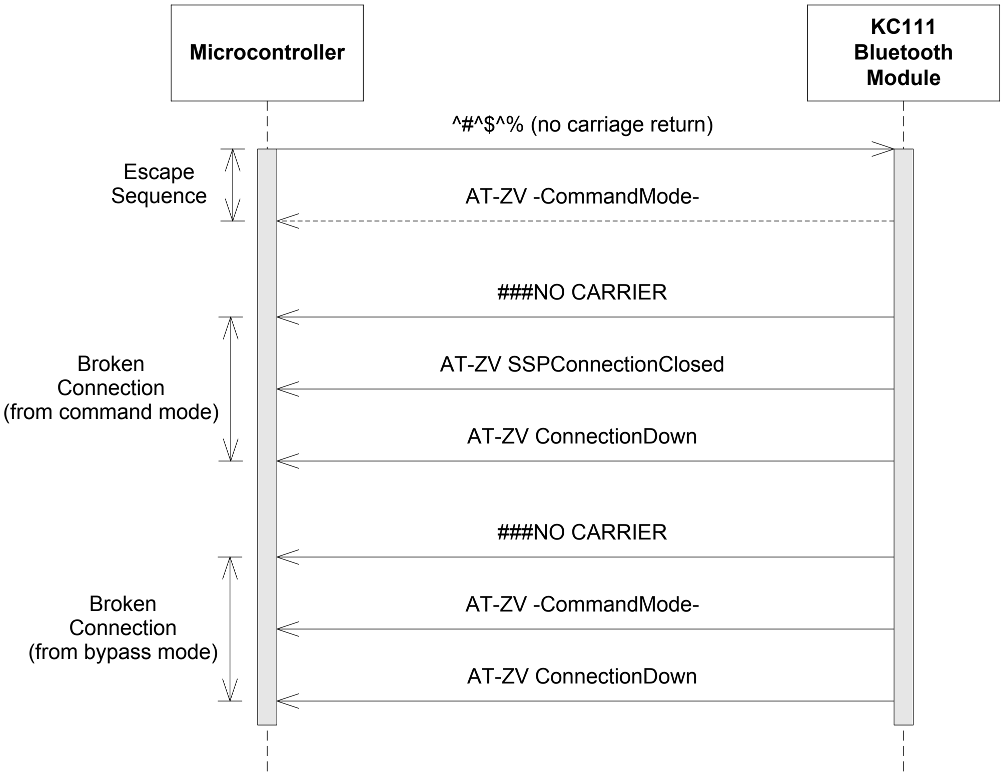


Figure 7.4: KC111 Connection Loss Sequences

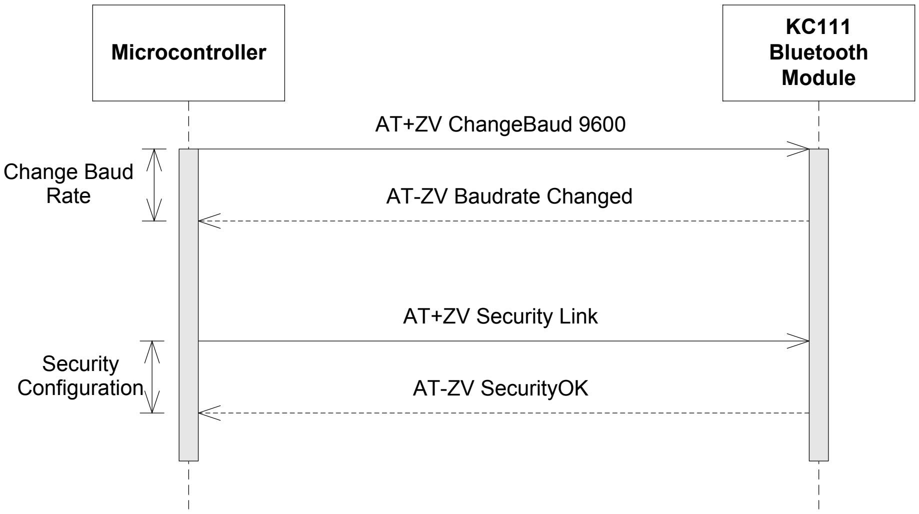


Figure 7.5: KC111 Baud Rate Change and Security Configuration Sequences

7.2.2 PDA Bluetooth Communications

To communicate using Bluetooth is slightly more complicated in Windows Mobile. There is no bundled developer support for Bluetooth and device manufacturers may also choose from several Bluetooth networking stacks. A stack implements a layered architecture to support the connectivity and protocols used to communicate over Bluetooth (Wigley et al., 2007, p. 319). Therefore, to develop this part of the communication architecture, the 32feet.NET (2010) free shared-source library is used.

In the case of Bluetooth the PDA acts as a client which initiates a connection to the mobile robot. The code to accomplish this is shown below.

```
private BluetoothClient bluetoothClient;

private void StartBluetooth()
{
    BluetoothRadio bluetoothRadio = BluetoothRadio.PrimaryRadio;
    if (bluetoothRadio == null)
    {
        MessageBox.Show("No supported Bluetooth radio/stack found!");
    }
    else if (bluetoothRadio.Mode != InTheHand.Net.Bluetooth.RadioMode.Connectable)
    {
        DialogResult result = MessageBox.Show(
            "Make Bluetooth radio connectable?",
            "Bluetooth",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question,
            MessageBoxDefaultButton.Button1);
        if (result == DialogResult.Yes)
        {
            bluetoothRadio.Mode = RadioMode.Connectable;
        }
    }
    bluetoothClient = new BluetoothClient();
}

private void ConnectToBluetoothServer()
{
    SelectBluetoothDeviceDialog dialog = new SelectBluetoothDeviceDialog();
    DialogResult result = dialog.ShowDialog();
    if (result == DialogResult.OK)
    {
        try
        {
            BluetoothDeviceInfo deviceInfo = dialog.SelectedDevice;
            BluetoothAddress address = deviceInfo.DeviceAddress;
            BluetoothEndPoint endpoint = new BluetoothEndPoint(
                address,
                BluetoothService.SerialPort);
            bluetoothClient.Connect(endpoint);
            Thread bluetoothServerThread = new Thread(new ThreadStart(ListenLoop));
41              bluetoothServerThread.Start();
42          }
43          catch
44          {
45              // connection error
46          }
47      }
48  }
```

The “StartBluetooth” function enables the PDA Bluetooth hardware (lines 3-24). A check is done to see whether the PDA support Bluetooth, which is necessary if the same code is ported to another device (line 6). A second check is done to determine whether the Bluetooth is connectable and if not the user is prompted to enable this mode (lines 10-22). If these steps complete without any errors the Bluetooth client is ready to connect to the mobile robot.

To choose the device to connect to a dialog window from the code library is used. This window lists the nearby Bluetooth devices by name and allows the user to choose a device to connect to (lines 28–30). The device’s Bluetooth address is automatically retrieved by the code library and, together with the service type, is used to connect to the mobile robot (lines 32–39). A separate worker thread is created to handle communications between the devices (lines 40–41). Once a connection has been established data is transferred over a network stream in the same way as the TCP example above.

7.3 Custom Communications Protocol

The custom communications protocol is implemented between the PDA and mobile platform’s processing element, the PIC18F4620. The communications mechanism between these two devices is Bluetooth.

The use of Bluetooth as communications mechanism influenced the design of the communications protocol. Due to the KC111 Bluetooth Adapter module used, messages received by the microcontroller are either control messages from the KC111 unit itself or commands from the PDA. These messages are passed using the different operation modes of the KC111.

When in bypass mode, a seamless bi-directional link is established between the PDA and microcontroller. Here any characters received is passed unchanged and directly to the microcontroller. When in the control mode, no messages from the PDA are received and all messages originate from the KC111.

The serial receive components described in Section 6.2.7 cater for the event when the communications link between the PDA and microcontroller is lost and the Bluetooth module automatically (possibly in the middle of a command message from the PDA) reverts back to command mode. This is done by the routine always considering three characters in the receive buffer at a time: the byte currently pointed to, the previous byte fetched and the next byte to be fetched. In this way the "###" sequence is easily picked up as the start of a carrier loss message even when in the middle of an unfinished packet.

7.3.1 Protocol Features

A start character is used to indicate the start of a command or request packet from the PDA. As the KC111 originating messages all start with one of two possible characters. The use of a start character simplifies the discarding of "junk" characters that may be received during connection setup. It also serves to distinguish between KC111 and PDA originating messages.

The protocol supports 15 unique node addresses, where a node can be another robot or base station. The bluetooth module chosen only supports one peer-to-peer connection at any time, but using a high-end model would allow multiple robots to communicate with one another using the address byte to specify who a specific message is intended for. Here broadcast messages would also be applicable and the address byte could be used to indicate this type of message in the future. For the current implementation, the PDA and robot were given arbitrary addresses. The address byte contains both the sender and receiver's address.

Having a single byte that indicates the meaning of the rest of the packet, increases the speed at which a packet can be acted upon. The function type byte is used together with the length byte to indicate the appropriate packet handling. The length byte indicates the length of data to follow. Using a single byte limits the length of data in a single packet to 255 bytes. The data in the packet is indicated through the length byte together with the function byte.

Due to the transport medium being air and susceptible to all kind of electrostatic noise it is important to ensure the integrity of a received packet. A 16-bit redundancy check is used to ensure that the data is valid. Stop bytes are used to allow the reception of variable length packets without first having

Start Byte	Address To/ Address From	Function Type	Length	Data	CRC	Stop Bytes

Figure 7.6: Custom Protocol Packet Structure

to find and check the length byte which is not even present in the packet received from the Bluetooth module. The KC111 Bluetooth module employ variable length packets and ends all incoming command messages with a CR (carriage return) followed by a LF (line feed) character. Having a variable length packet not only accommodates the already variable packets from the bluetooth module, but also allows for packets to be optimally compact. This in turn allows for greater responsiveness of the robot to act on a command received.

7.3.2 Structure of Command and Response Packets

In order to be as flexible as possible, the packet is byte aligned, with 8 bit bytes. This makes implementation easy and efficient on a number of platforms, including the 8bit PIC microcontroller used as the main processing unit on the mobile platform. A packet is a communication element and may be a response type or command type depending on the data contained within the packet. Two types of command packets are defined: request packets and command packets. A request packet will require the microcontroller to respond to the PDA with the requested system data. A command packet will instruct the microcontroller as to one or more actions to be preformed and is coupled with actuator actions.

All packets sent between the PDA and mobile platform are variable length and have a structure as shown in Figure 7.6. Table 7.1 describes the components of the packet structure.

Command Packets

The Function Type byte will take on two formats depending on the type of command packet received. When the packet contains no data (the data length byte is zero), the packet is a request-type command. The Function Type byte will then have the format as shown in Figure 7.7 with each bit indicating a type of sensor data to request.

Table 7.1: Byte Description of Packet Structure

Component	**Length**	**Description**
Start Byte	1 byte	Indicates start of a packet
Address to/from	1 byte	Upper nibble is address to, lower nibble is address from
Function Type	1 byte	Identify type of data sent
Length	1 byte	Number of bytes in the Data field
Data	0 to N bytes	Data being received/sent
CRC	2 bytes	16 bit CRC
Stop Bytes	2 bytes	Indicates end of packet

The "CollectData" routine is used to gather the requested values according to the function type bitmap in Figure 7.7 and populates the data field of the packet to return to the PDA.

The second type of packet requests the change of certain control variables. These variables are specified through the function parameter byte as shown in Figure 7.8.

The "SetData" routine is responsible for setting the requested variables to the value in the data field.

7.3.3 Packet Exchange

The microcontroller sends and receives bytes through serial receive and transmit interrupts together with the use of receive and transmit buffers. These buffers are cyclic in nature.

The serial interrupt receive routine places the incoming bytes into the receive buffer. If the buffer reaches its maximum, it wraps around and starts at position 0 within the buffer.

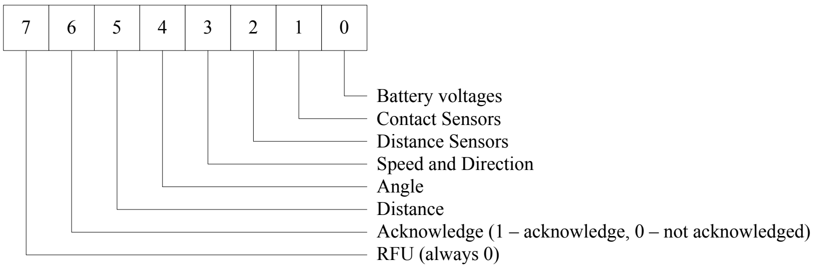


Figure 7.7: Function Type Byte Details (Data Request)

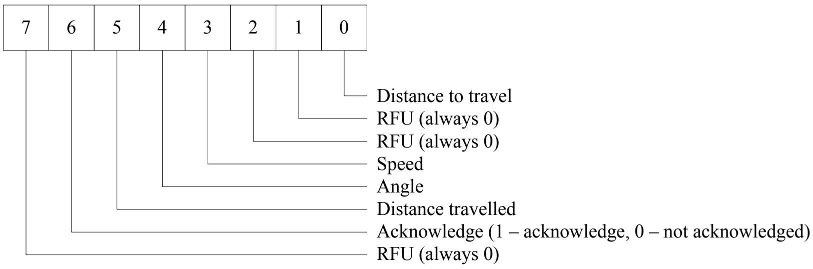


Figure 7.8: Function Type Byte Details (Command)

"BufferedGetc" is a similar implementation to the standard "getc" function with two significant differences. Firstly, the function has the added feature of not trapping the program if a byte is not currently available. This is done using a time out function. The other major difference is that the "BufferedGetc" function fetches a byte from the receive buffer and not directly from the UART buffer as with "getc".

The transmit interrupt together with a transmit buffer and a "Buffered-Putc" implementation is used to transmit characters without influencing the other interrupt sources in the PIC18F4620.

The microcontroller must be able to handle and provide appropriate response messages to each of the command class packets shown in Table 7.2.

Class 1 Commands

Class 1 commands request sensor and/or actuator data according to the bitmap in the function type byte in the packet header as shown in Figure 7.7. For this type of command the data field length should be indicated to be zero.

Table 7.2: Command Class Descriptions

Command Class	**Short Description**
Class 1	Valid, clean packet requesting specific sensor and/or actuator data
Class 2	Valid, clean packet requesting the setting of specific control variables
Class 3	Valid packets (correct start and stop bytes) that do not pass all validity tests
Class 4	Invalid packets
Class 5	Bluetooth module originating packets

The protocol does not currently support commands that both set and return data.

In the response packet, the function type byte indicates that the packet received was acknowledged (by setting the acknowledge bit) and which data is included within the data field of the packet. The function byte should mirror the received command's function byte with the exception of the acknowledge bit that may or may not be the same.

The microcontroller source builds the response packet using the "BuildPakcet" and "CollectData" routines. The "SendPacket" routine sends the response packet to the PDA.

Class 2 Commands

Class 2 commands are related to the actuators on the mobile platform. The current implementation supports setting the speed, angle and distance to travel before stopping. The distance that has been covered can also be updated to correct for odometric errors. This is done by setting global control variables for each of the actuators involved as indicated through the bitmap in the packet type byte. The data field contains each new setting, thus this packet's data length must match the data field's length in bytes.

The response packet will set the acknowledge bit in the function type byte, but sets all other bits in the function type byte to zero. The data length is also zero, as no data is requested using the Class 2 command. This forms a pure acknowledge packet. The response packet is constructed using the "BuildPacket" and sent with the "SendPacket" routine.

Class 3 Commands

Here an error has been detected in the packet body. Three tests are implemented and each has a corresponding error code. These validity tests test the packet length, crc and address. In each case the mobile platform will return a "not acknowledged" to the PDA through clearing the acknowledge bit in the function type byte. All the other bits in the function parameter byte is cleared as well as no sensor or actuator data will be returned in the response packet.

The data length is however set to indicate one byte. The data field is to contain a single length error core. The error code will correspond to the first error picked up in the packet. The validity tests are done consecutively, stopping and returning an error condition on the first test failure. The “ValidDataPacket(byte *packet_ptr, int packet_length)” firsts tests whether the packet has been received by the correct mobile platform. Next, the length byte is validated against the actual data length of the data field. Lastly the CRC is verified. This means that if, for example, a packet is received that has both an incorrect length byte as well as incorrect CRC, only the first error picked up of incorrect length will be indicated in the error code returned. “BuiltPacket” constructs this “error packet” and “SendPacket” sends it to the PDA.

Class 4 Commands

Class 4 commands describes invalid packets. An invalid packet is one that could not be received at all. This could be the case where most, if not all, of the received bytes had to be dropped by the receive routine due to the start byte not being received correctly. Another packet that will also result in an invalid packet is the case when an incomplete packet is received due to a time out condition that occurred. These situations will result in a response packet sent to the PDA in the same format as Class 3 commands, except for the error code returned in the packet data. Invalid packets will return an “invalid packet” error code, while packets that timed out will return a “timed out” error code.

Class 5 Commands

Class 5 commands describe Bluetooth Module-originating command packets. Bluetooth Module-originating packets are only sent to the microcontroller when the communications link between the microcontroller and PDA has not been set up or has been broken for whatever reason. The Bluetooth Module is then in what’s called its ”Command Mode” and all packets sent will only go to the module itself and not be passed to the PDA as in the case of normal communications.

The Bluetooth Module-originating packets have a different structure to command in Class 1 to 4. Here the packets consists of ascii string arrays that end with two stop bytes (a CR followed by a LF). Packets can start with either a ‘A’ or ‘#’ character.

Due to the tedious an time consuming task of copying constant character strings from ROM to RAM in order to compare the two, a hashing function is used to simplify and streamline the process. This method is also faster and easier to implement than a tree search algorithm. The hashing algorithm accepts the Class 5 packet as input and produces a unique hash value from it. This value will be unique for each different string passed to the hashing function. The hash values is used in a "switch()" case statement to preform the needed actions on the different Bluetooth packets from the Bluetooth Module. Only the needed Bluetooth packets are implemented in the "InterpretHashValue" routine. The only difficulty in using the hash function is that all the hash values must be calculated beforehand in order to create the hash table in the source, or in this implementation, the state machine.

Most Bluetooth module originating packets indicates a loss in communication with the PDA (exceptions include "Bypass Mode“ packets). The micro keeps track of the communications link state and will not attempt sending any messages to the PDA before a link has been established and indicated though the "Bypass Mode" Class 5 packet. Any PDA destined packets sent during this mode would only cause the Bluetooth module to return an error message to the microcontroller.

The microcontroller also keeps track of the number of packets that have not been acknowledged by the PDA. If the maximum number of consecutive messages have failed to reach the PDA, communications can be considered to be broken down and the microcontroller can try to recover by sending an escape sequence to the KC111 Bluetooth module.

7.4 Conclusions

This chapter implemented the communications architecture designed in Chapter 4. The PC-PDA communications link was implemented using WLAN communications and the TCP protocol, with the PC implemented as the TCP server and the PDA as TCP client. The PDA also communicates with the mobile platform. This is done through a Bluetooth connection, using the KC111 Bluetooth serial adapter to allow the microcontroller, as the mobile platform's controller, to connect to the PDA through Bluetooth. A custom protocol is described for packet exchange. The framework and protocol defined in this chapter supports the control architecture using distributed hardware. Chapter 8 will test whether the design and communications architecture can be successfully applied to a typical office robot scenario and whether this would have a positive effect on system performance.

Part III

Evaluation

Chapter 8

Experimental Results

Office robots need to be able to preform a number of tasks with varying response-time limitations and processing requirements. An office robot may typically want to learn from its environment, updating a map by particularly noting stationary objects as "discovered" by the robot. To be useful an office robot would need to be able to move in its environment – an office building. A more specific example would be to task the robot with finding and entering a specific office.

This chapter uses this basic office robot task as a starting point for the PDA-incorporated robotic system. The task seems to suite the layered control architecture as the global problem of "finding an office" can be easily and logically broken down into several local and reactive tasks.

The rest of this chapter proceeds as follows. First, the test scenario is described. This is followed by how the task is broken down into subtasks and how these are implemented using the conceptual framework discussed in Chapter 4. The test scenario is implemented using the developed prototype in two configurations. One where the PDA is used solely as message forwarding device and the other according to the developed framework. The data exchange between the PDA and microcontroller, using the custom protocol described in Chapter 7, is shown for one of the local planning tasks implemented on the PDA. The chapter concludes by discussing the results of the two experiments in terms of time variations and power consumption.

8.1 Test Case Description

The scenario chosen has the robot in a passage within an office building. This is a long passageway with several offices leading out from it. The robot is tasked with finding a specific room or office number in this passage, moving to it and entering into the room, ready for further instruction. It is assumed that the robot is facing in the right direction and is already located within the passage. All locatable offices have their doors open and it is assumed that the robot has kept track of its current position within the passage. Figure 8.1 shows the mobile robot in the passage and the route it will need to follow in order to get to the desired location.

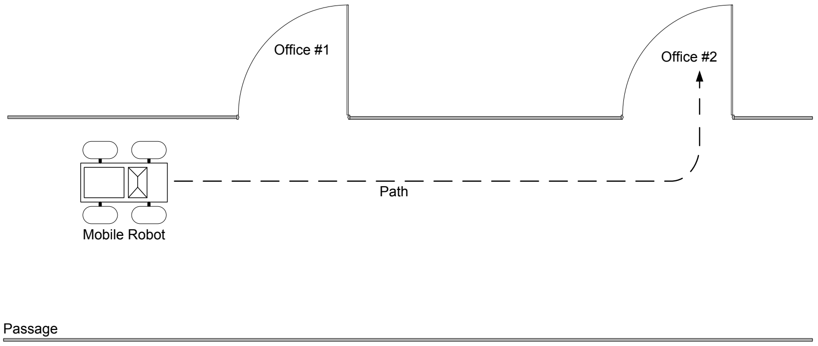


Figure 8.1: Office Test Scenario

8.1.1 Task Breakdown

The high-level task is to find and enter a specific office. In order to preform this task, the robot must be able to:

1. Determine where the destination office is with respect to the robot's current location.

2. Determine how many doorways to pass in the hallway to reach the destination.

3. Locate a doorway by:

   (a) Monitoring the distance sensor data.

   (b) Monitoring the contact sensor data.

   (c) Adjusting the steering motor's angle if needed.
   (d) Stopping the drive motor.

4. Handle "stuck" situations where the robot has become trapped behind some object.

5. If the doorway should be traversed:

   (a) Driving the robot forward as straight as possible.
   (b) Monitoring the distance sensor data.
   (c) Monitoring the contact sensor data.
   (d) Stopping the drive motor.

6. Enter a doorway by:

   (a) Driving a short distance forward.
   (b) Turning at the maximum possible angle into the doorway.
   (c) Stopping after entering a short distance into the office.

Categorising Global, Local and Reactive Tasks

These tasks can now be categorised as either global or deliberative, local planning or reactive tasks. Global tasks associated with finding the office and entering it can include:

1. Determine where the robot is with respect to the goal office.
2. If a door has been found to determine if it's the one to enter.
3. If a door could not be found and the robot is stuck against an object, to update the map of its environment.
4. Deciding how to recover from a stuck position.

The local planning tasks include:

1. Locating a doorway in the passage.
2. Passing a door opening.
3. Entering an office.

Last, reactive tasks will include:

1. Reading sensor information.
2. Stopping the drive motor as soon as contact occurs.
3. Steering the robot.
4. Driving and stopping the robot.

8.1.2 Task Implementation

The subtasks defined and categorised in Section 8.1.1 is implemented using three main functions – “GoToOffice”, “FindDoor” and “PassDoorOpening”.

GoToOffice

Figure 8.2 shows the “GoToOffice” routine’s flow diagram. The routine will call the “FindDoor” routine to follow the passage wall until an opening can be found. If a door has been found, but it is not the correct one, the “PassDoorOpening” routine is used to move past the door opening and find the wall on the other side of the doorway. If the door found is that of the office in question, the “Enter” routine is used to move forward, turn into the office, drive forward and stop just inside the office.

FindDoor

The “FindDoor” routine shown in Figure 8.3, requests data from the IR distance sensors on the mobile platform as well as the contact sensors. The distance value of the IR sensor on the side of the robot facing the wall is checked against a pre-defined minimum wall following distance. Due to many factors influencing the mobile robot it does not drive perfectly straight, even when the steering is set to drive straight. Thus, if the distance to the wall is sensed to be decreasing and crosses a minimum threshold value, the steering angle of the robot is adjusted away from the wall. If the distance to the wall is increasing and crosses a maximum threshold value, the steering angle is adjusted toward the wall. This results in the mobile robot having an oscillating type of action instead of taking a straight path down the passage, as shown in Figure 8.4.

The angle adjustment is done through a command, “SetServo”, to set the angle to the mobile platform. If the distance measured by the side sensor

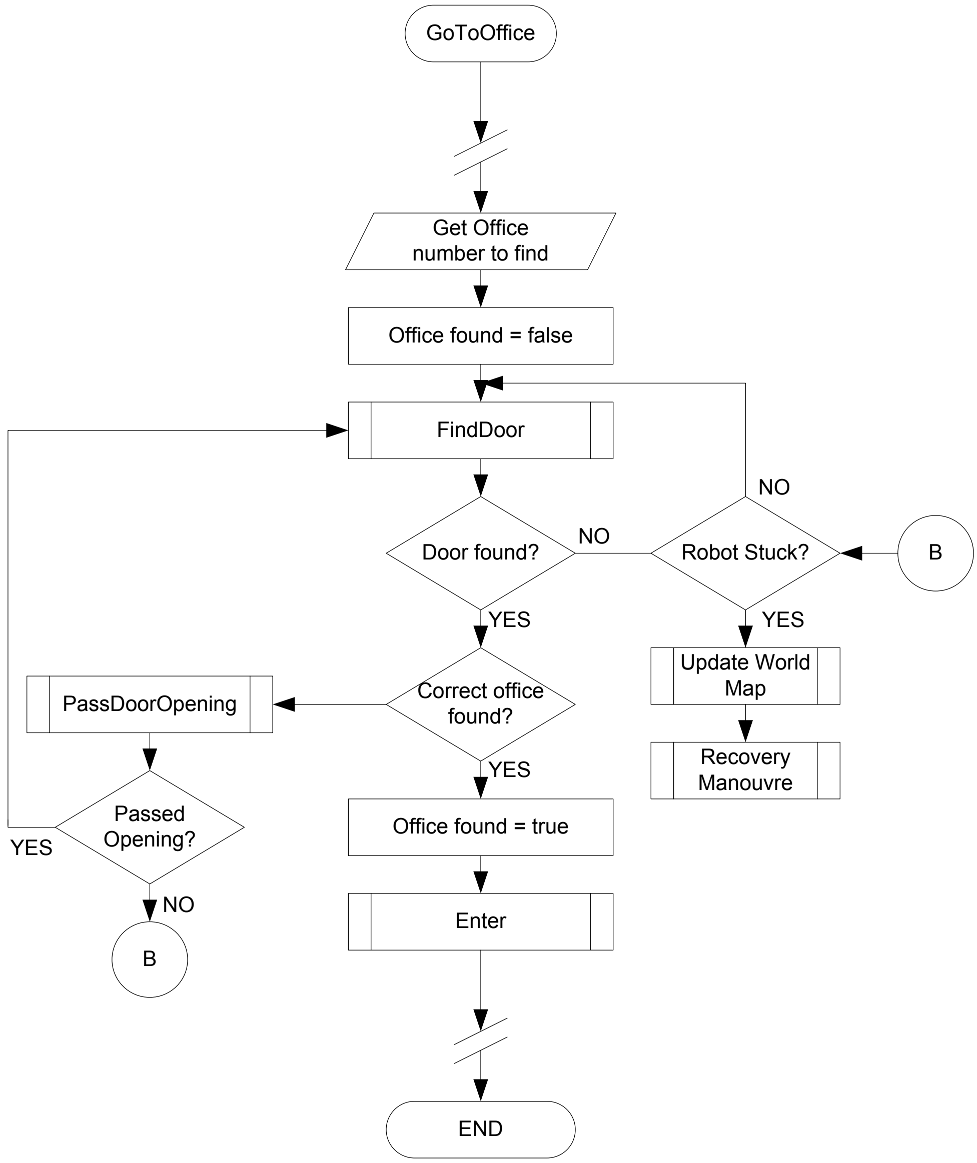


Figure 8.2: GoToOffice() Routine Flow Diagram

gives no reading an opening, i.e. doorway, has been found. Stopping the motor is also a command, “StopDriveMotor”, sent to the mobile platform.

PassDoorOpening

The “PassDoorOpening” routine shown in Figure 8.5 is almost an exact opposite of the “FindDoor” routine. Here the measurements are evaluated in order to find a wall instead of an opening. Once again the distance sensor measurements are used to make this determination.

Enter

The “Enter” routine is shown in Figure 8.6. Here the robot would be just passed the door opening. To ensure that the robot does not hit the doorway as it turns into the office, the robot first moves a short distance forward. The robot now turns towards to opening and enters the office a short distance before coming to a stop.

8.1.3 Task Distribution

Each tasks can be seen as a piece of the control intelligence of the robot system as a whole. This section describes how the tasks outlined for the office robot is implemented in two different ways. The implemented test cases are used to produce the test results in Section 8.2.

For the first test case, the PDA has no intelligent processing. It’s only purpose is to forward a message received through WLAN from the PC, through the Bluetooth link to the mobile platform and back again. The tests done for this case implements both the global and local planning tasks on the PC, with the reactive tasks and sensor/actuator interface on the mobile platform. The mobile platform receives commands formatted according to the custom protocol in Chapter 7. From the flow diagrams in Figures 8.2, 8.3, 8.5 and 8.6 these would include “GetIRandSensorData”, “SetServo”, “StopDriveMotor”, “DriveMotor” and “SetDistanceToTravel” tasks. All other functionality is implemented on the PC.

The second scenario implements the defined distributed framework. Here the global tasks are implemented on the PC, local planning tasks on the PDA and reactive tasks on the mobile platform’s controller. The local tasks translate to the “FindDoor”, “PassDoorOpening” and “Enter” routines. These

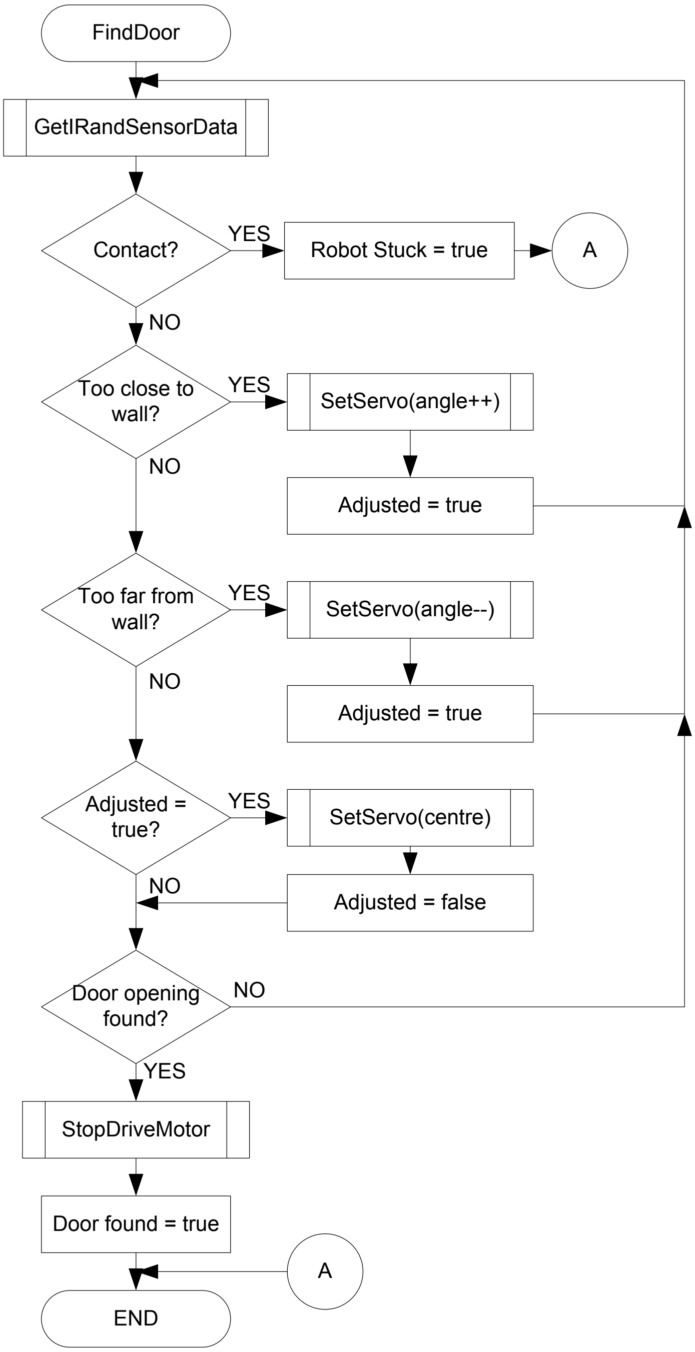


Figure 8.3: FindDoor() Routine Flow Diagram

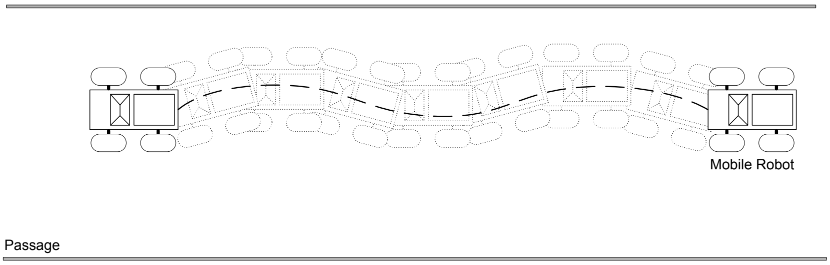


Figure 8.4: Path Followed By Robot Down Passage

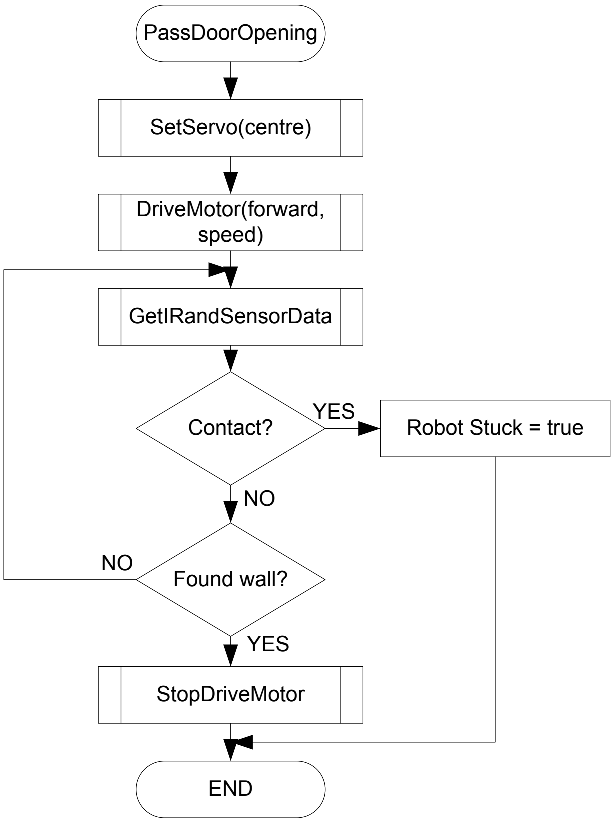


Figure 8.5: PassDoorOpening() Routine Flow Diagram

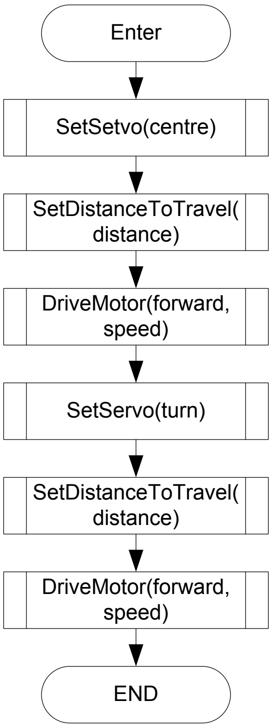

Enter
SetSetvo(centre)
SetDistanceToTravel(distance)
DriveMotor(forward, speed)
SetServo(turn)
SetDistanceToTravel(distance)
DriveMotor(forward, speed)
END

Figure 8.6: Enter() Routine Flow Diagram

are called by the global planning layer which is implemented using the "GoToOffice" routine. The reactive components are the same as for the first implementation and are used by the local planning layer.

With the second test setup, the global planning layer on the PC gives information and receives information from the local planning layer on the PDA. In the same way, information is shared between the local planning layer on the PDA and the mobile platform. The global and local layers do not pass actuator and sensor specific commands and responses to one another – the data is abstracted away. Similarly, the local layer does not receive raw data from the mobile platform but, for example, distances measured in cm. Here too, the data is abstracted. This allows for a flexible implementation, as the PC and global planning layer does not need to have any knowledge of how the mobile platform needs to be controlled in order to reach the goal.

8.2 Results

The following sections describe the results of the test cases.

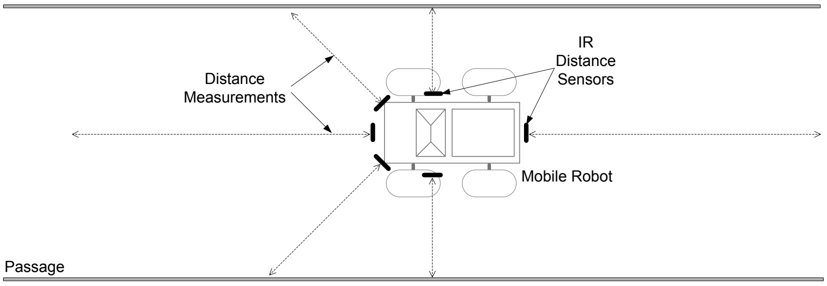


Figure 8.7: Distance Measurements using the IR Distance Sensors

8.2.1 Command-Response Pairs

The find door routine produces a command-response pair sequence similar to that shown in the sequence diagram in Figure 8.8. The distance measurements shown in the responses from the mobile platform's controller includes data from six IR distance sensors placed as shown in Figure 8.7.

The placement of the sensors were chosen specifically for the wall following routine implemented. The side IR sensors are placed off-center and towards the front. This allows using only a single sensor for wall following. If the sensor were placed exactly in the middle of the robot body the robot would be able to turn off course further, before being picked up as a distance change by the sensor. This would cause a path with a bigger oscillation curve than that shown in Figure 8.4. The off-center placement also assists in the robot knowing which section is getting close to an object with respect to the front- or back-side of the robot. When the side of interest's distance sensor reading gives a zero result it means that the sensor is over an opening.

Figure 8.8 corresponds with the "FindDoor" flow diagram in Figure 8.3. Commands sent by the PDA and responses sent back by the mobile platform controller is shown for each action taken. These actions correspond to the "GetIRandSensorData", "SetServo", "StopDriveMotor", "DriveMotor" and "SetDistanceToTravel" tasks shown in the "FindDoor" routine. When the robot is considered next to a wall the side IR sensor, right in this case, indicates a distance of 20cm. Too close is when this same sensor indicates 15cm and too far would be a reading of 25cm or greater. A zero reading indicates a door opening.

The communications between the PC and PDA is shown in Figure 8.9.

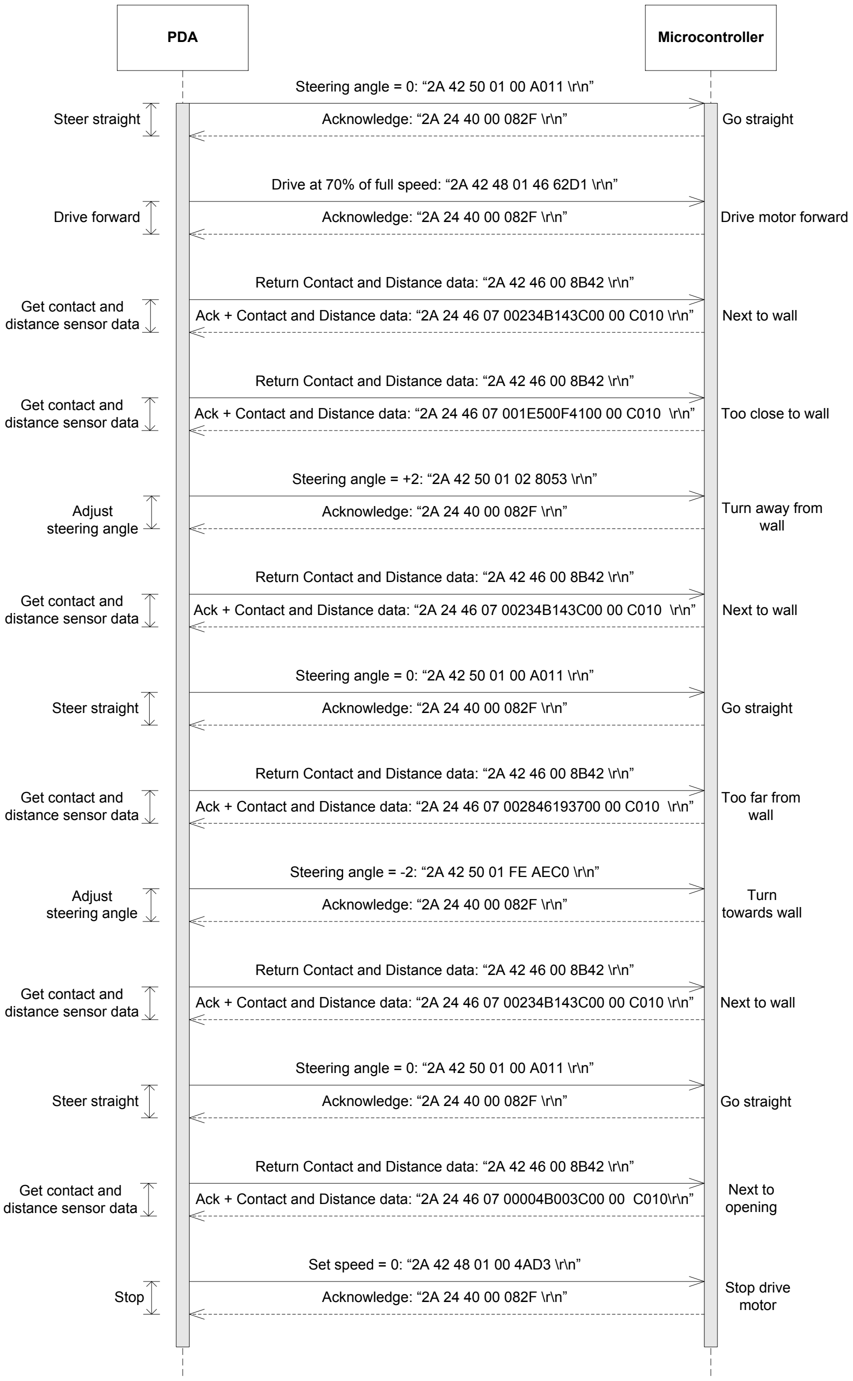

PDA
Microcontroller
Steering angle = 0: “2A 42 50 01 00 A011 \r\n”
Steer straight
Acknowledge: “2A 24 40 00 082F \r\n”
Go straight
Drive at 70% of full speed: “2A 42 48 01 46 62D1 \r\n”
Drive forward
Acknowledge: “2A 24 40 00 082F \r\n”
Drive motor forward
Return Contact and Distance data: “2A 42 46 00 8B42 \r\n”
Get contact and distance sensor data
Ack + Contact and Distance data: “2A 24 46 07 00234B143C00 00 C010 \r\n”
Next to wall
Return Contact and Distance data: “2A 42 46 00 8B42 \r\n”
Get contact and distance sensor data
Ack + Contact and Distance data: “2A 24 46 07 001E500F4100 00 C010 \r\n”
Too close to wall
Steering angle = +2: “2A 42 50 01 02 8053 \r\n”
Adjust steering angle
Acknowledge: “2A 24 40 00 082F \r\n”
Turn away from wall
Return Contact and Distance data: “2A 42 46 00 8B42 \r\n”
Get contact and distance sensor data
Ack + Contact and Distance data: “2A 24 46 07 00234B143C00 00 C010 \r\n”
Next to wall
Steering angle = 0: “2A 42 50 01 00 A011 \r\n”
Steer straight
Acknowledge: “2A 24 40 00 082F \r\n”
Go straight
Return Contact and Distance data: “2A 42 46 00 8B42 \r\n”
Get contact and distance sensor data
Ack + Contact and Distance data: “2A 24 46 07 002846193700 00 C010 \r\n”
Too far from wall
Steering angle = -2: “2A 42 50 01 FE AEC0 \r\n”
Adjust steering angle
Acknowledge: “2A 24 40 00 082F \r\n”
Turn towards wall
Return Contact and Distance data: “2A 42 46 00 8B42 \r\n”
Get contact and distance sensor data
Ack + Contact and Distance data: “2A 24 46 07 00234B143C00 00 C010 \r\n”
Next to wall
Steering angle = 0: “2A 42 50 01 00 A011 \r\n”
Steer straight
Acknowledge: “2A 24 40 00 082F \r\n”
Go straight
Return Contact and Distance data: “2A 42 46 00 8B42 \r\n”
Get contact and distance sensor data
Ack + Contact and Distance data: “2A 24 46 07 00004B003C00 00 C010\r\n”
Next to opening
Set speed = 0: “2A 42 48 01 00 4AD3 \r\n”
Stop
Acknowledge: “2A 24 40 00 082F \r\n”
Stop drive motor

Figure 8.8: Command/Response Sequence between the PDA and Microcontroller

The figure shows the user interface on both devices and the commands being transmitted in each case.

8.2.2 Timing Results

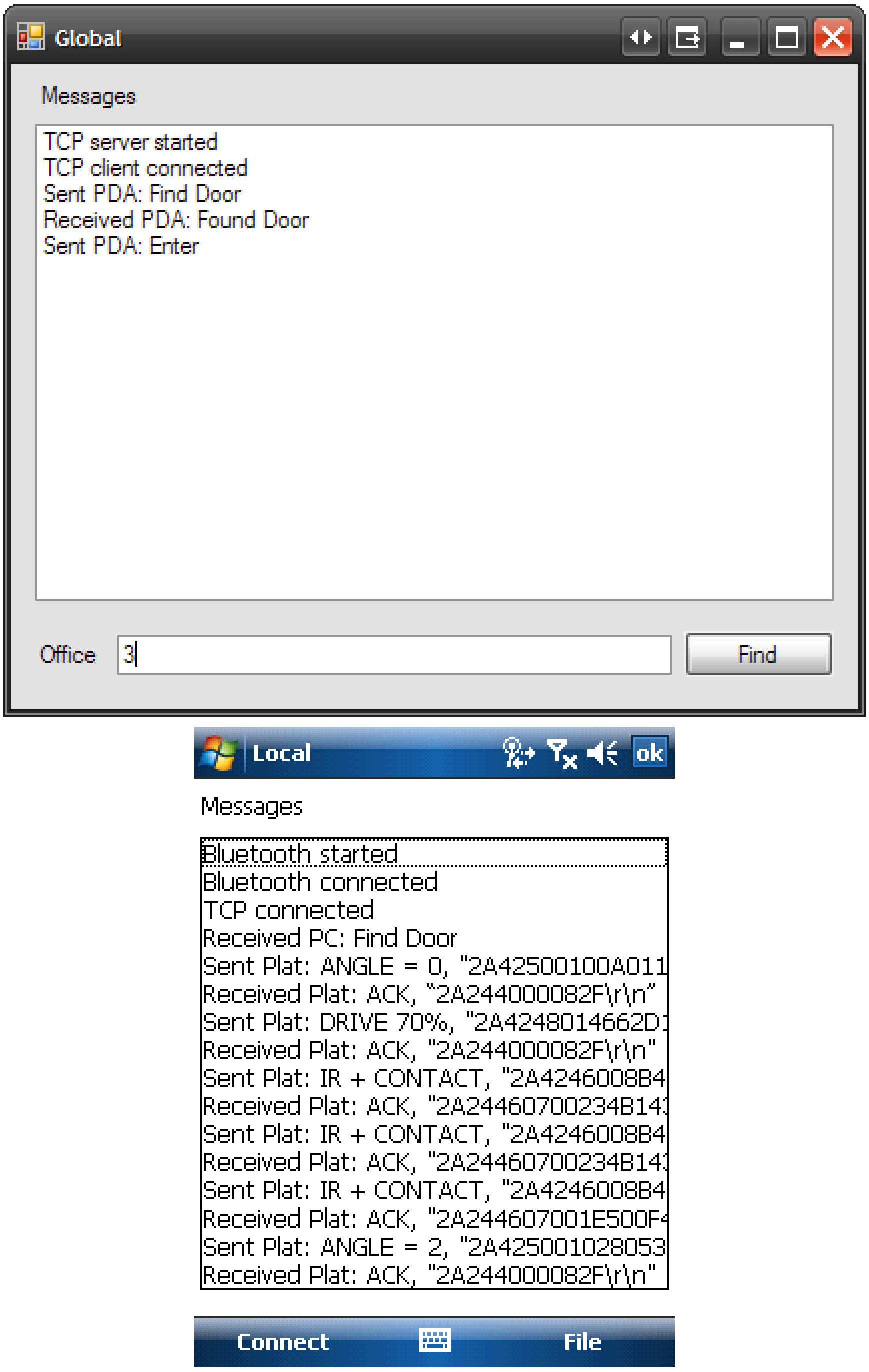
The time taken, calculated as the average of 500 command/response pairs, to send a command from the PC and receive a response back was calculated to be 0.077 ms. When sending the same command from the PDA (communicating over Bluetooth only) a response is received in 0.056 ms. This supports the hypothesis for implementing more time critical tasks on the PDA with respect to the PC.

8.2.3 Monitoring the Battery State

Windows Mobile exposes the current battery state through a system property which can be queried whenever needed. However, the battery level is rounded in blocks of 20% which makes exact measurements impossible (MSDN Library, 2010). Fortunately it is possible to get the exact battery level by invoking a native method of the operating system. The code below shows how this is done.

```
[DllImport("coredll")]
private static extern uint GetSystemPowerStatusEx2(
    SYSTEM_POWER_STATUS_EX2 lpSystemPowerStatus,
    uint dwLen,
    bool fUpdate);

public class SYSTEM_POWER_STATUS_EX2
{
    public byte ACLineStatus;
    public byte BatteryFlag;
    public byte BatteryLifePercent;
    public byte Reserved1;
    public uint BatteryLifeTime;
    public uint BatteryFullLifeTime;
    public byte Reserved2;
    public byte BackupBatteryFlag;
    public byte BackupBatteryLifePercent;
    public byte Reserved3;
    public uint BackupBatteryLifeTime;
    public uint BackupBatteryFullLifeTime;
    public uint BatteryVoltage;
    public uint BatteryCurrent;
    public uint BatteryAverageCurrent;
    public uint BatteryAverageInterval;
    public uint BatterymAHourConsumed;
    public uint BatteryTemperature;
```



```
TCP server started
TCP client connected
Sent PDA: Find Door
Received PDA: Found Door
Sent PDA: Enter
```

Office 3

```
Bluetooth started
Bluetooth connected
TCP connected
Received PC: Find Door
Sent Plat: ANGLE = 0, "2A42500100A011
Received Plat: ACK, "2A244000082F\r\n"
Sent Plat: DRIVE 70%, "2A4248014662D1
Received Plat: ACK, "2A244000082F\r\n"
Sent Plat: IR + CONTACT, "2A4246008B4
Received Plat: ACK, "2A24460700234B14
Sent Plat: IR + CONTACT, "2A4246008B4
Received Plat: ACK, "2A24460700234B14
Sent Plat: IR + CONTACT, "2A4246008B4
Received Plat: ACK, "2A244607001E500F4
Sent Plat: ANGLE = 2, "2A425001028053
Received Plat: ACK, "2A244000082F\r\n"
```

Figure 8.9: PC (top) and PDA (bottom) Applications

```
27     public uint BackupBatteryVoltage;
28     public byte BatteryChemistry;
29 }
30
31 SYSTEM_POWER_STATUS_EX2 status2 = new SYSTEM_POWER_STATUS_EX2();
```

First a function inside a dynamic-link library is invoked, which contains the functionality to retrieve battery status information (lines 1–4). This is done using the "DLLImport" statement which provides a platform invocation subsystem to call into external code (Wigley et al., 2007, p. 495). The specific function that is needed to retrieve battery status information is "GetSystemPowerStatusEx2". This function takes three parameters: a pointer to a buffer that receives power status information, the length of the buffer and whether to get the latest or cached information. It returns the length of the data in the buffer if called successfully (MSDN Library, 2011a).

The "SYSTEM_POWER_STATUS_EX2" structure contains information about the power status of the system (lines 7–29). Inside this structure the "BatteryLifePercent" and "BatteryVoltage" returns the percentage of full battery charge remaining and amount of battery voltage (in millivolts) respectively (MSDN Library, 2011b). These two values are used to measure the battery state. Finally, an instance of the structure is created which can be used from managed code (line 31).

Values are measured in fixed time intervals and logged to a file. The code below shows how this is done.

```
Timer timer = new Timer();

void StartBatteryMonitor()
{
    timer.Interval = 10000;
    timer.Tick += new EventHandler(timer_Tick);
    if (!timer.Enabled)
    {
        timer.Enabled = true;
    }
}

void timer_Tick(object sender, EventArgs e)
{
    if (GetSystemPowerStatusEx2(
        status2,
        (uint)Marshal.SizeOf(status2),
        true) == (uint)Marshal.SizeOf(status2))
    {
        StreamWriter writer = new StreamWriter(
            @"\batteryState.txt", true);
        writer.WriteLine(
23              "{0},{1},{2}",
24              DateTime.Now.ToString(),
25              String.Format("{0}%",
26              status2.BatteryLifePercent),
27              status2.BatteryVoltage.ToString());
28          writer.Close();
29      }
30  }
```

A timer is used to trigger a call to the "GetSystemPowerStatusEx2" function at consistent intervals. A measurement is taken every 10 seconds (line 5). If a measurement is taken successfully (lines 15–18) the relevant information together with a timestamp is written to a log file (lines 19–29). An example of the logged data is given below.

```
1/16/11 12:12:53 PM,46%,3641
1/16/11 12:13:03 PM,46%,3636
```

The logged data shows a comma-separated list of readings. First, the timestamps show that the readings were taken 10 seconds apart. Second, the percentage of battery charge remaining shows no value change in this interval. Third, the battery voltage shows a slight drop in power in this interval.

8.2.4 Power Consumption Results

This section discusses the power consumption results obtained from using the two tests cases within the described office robot scenario.

Idle State

For comparison, the power used by the PDA during idle state is shown in Figure 8.10. Figure 8.10 shows the power drop over a time period of 11 minutes, 52 seconds from 3.767 V to 3.753 V at an average rate of 0.019 mV/s.

Message Forwarder

Using the first test case with the PDA as message forwarder and no intelligent processing implemented, Figure 8.11 shows the resulting power consumption with respect to time. Here, the power level started at 3.863 V and over the experiment time of 10 minutes, 11 seconds the power dropped to 3.780 V. The voltage dropped at an average rate of 0.136 mV/s.

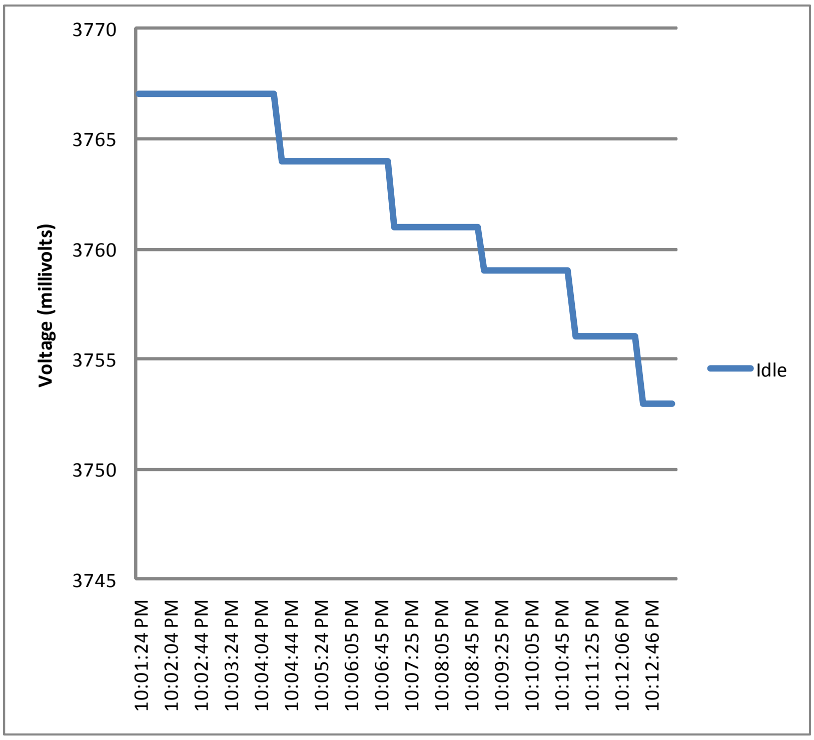


Figure 8.10: Idle Mode Power Drain

Distributed Control

With the second test case that uses the PDA to implement the local planning layer, the results as shown in Figure 8.12 were obtained. From the figure it can be seen that the power level dropped from 3.804 V to 3.780 V, at an average rate of 0.044 mV/s.

Comparing the three results shown in Figures 8.10, 8.11 and 8.12, there is a noticeable difference in power consumption between the PDA being used as a forwarding device and the PDA being used within a distributed control framework. The latter case uses less than a third of the power of forwarding commands and would thus seem preferable in this regard. The discrete steps of measured voltage as seen on the three graphs is due to the resolution of the PDA's power measurement system.

8.3 Conclusions

This chapter described a test scenario for an office robot. The typical task of finding a specific office and entering it was divided into subtasks and dis-

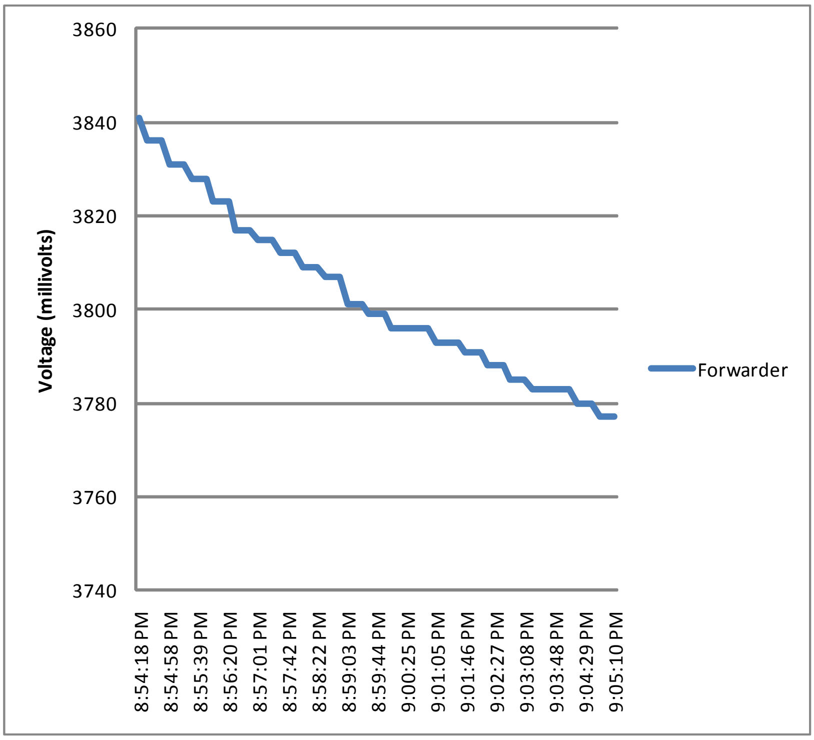


Figure 8.11: PDA as Message Forwarder Power Drain

tributed among the three processing elements (PC, PDA and mobile platform controller) according to the conceptual framework detailed in Chapter 4. Two test cases were implemented using the same tasks and the same amount of processing, in order to form an objective opinion of the success of the proposed framework. One test used the PDA as message forwarder, while the other used the PDA as proposed by the conceptual framework.

Performance was measured against response-time as well as power consumption of the PDA's battery. The results show that the power drain on the PDA battery is less when it is used as an intelligent agent in the robot control. The timing tests also indicated that the response between the PDA and mobile platform is considerably faster than when using the PDA to forward motor specific commands between the PC and mobile platform controller.

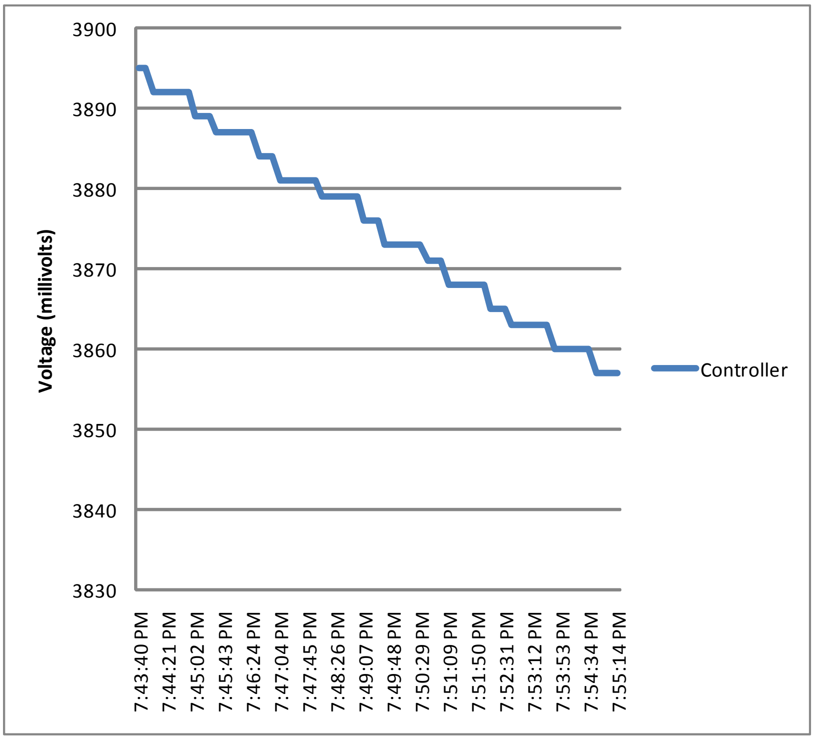


Figure 8.12: PDA as Message Forwarder Power Drain

Part IV

Epilogue

Chapter 9

Conclusion

The PDA's processing capability is underutilized when it is only used to forward control messages from the main (often remote) control station to the onboard platform controller. Often these messages are motor specific commands for the individual motors and servos on the robot base, increasing the latency between sending a command and its execution. In addition the PDA is not the ideal device to fulfil heavy processing demands, such as image processing. This leads to not implementing complete systems, due to the lack of resources, or settling for a less desirable output that can successfully be implemented on the PDA. Incorporating a wirelessly connected remote PC is common to many projects, but most only use the PC to give goal directives or motor commands. Some make use of a remote PC to perform debugging and logging of important parameters, while others implement a user interface on the PC. Few use the PC's processing power, speed and memory, implementing all the control software either on the onboard controller or PDA.

This research did not seek to argue whether using a PDA within a mobile robot system is the optimum choice or not. The fact is that many research projects have done so in the past and will undoubtedly do so in the future. In this study a possible framework that facilitates the inclusion of a PDA was presented.

The framework designed and described in this dissertation attempts to maximise the use of all the PDA's built-in abilities, including user interface strengths and processing capacity. A way in which control tasks can logically be distributed across three processing elements was shown using the typical three layer hybrid control architecture and the theory of data abstraction. This allowed for the most time critical tasks to be done by the onboard controller and the least time critical and most processing and memory intensive tasks to be done by a remotely connected PC.

The following sections describe how the research problem and objectives guided this work to the novel framework that was designed, implemented and tested.

9.1 Revisiting the Problem Statement

This dissertation addressed the issue of how a PDA can be used within a robot design to utilise its built in processing capability to assist with control. To arrive at a suitable answer to this problem, a number of sub-problems were posed concerning robot control using wirelessly distributed hardware. This dissertation answered these questions in the following way.

9.1.1 Distributing Intelligence in a Wireless Mobile Robot System

It was established that control intelligence can be distributed among different processing elements by applying parallel processing. The control software components that benefit from running in parallel on different systems depend greatly on the chosen control architecture. With a chosen control architecture, the various components thereof must be able to share information. With the hardware components of the robot system in this dissertation consisting of a PC, PDA and mobile platform, a wireless communications architecture was used to share the needed information between the physically distributed hardware components.

9.1.2 Considerations using a PDA in a Distributed Control System

PDAs were thoroughly examined in Chapter 2 as well as Chapter 3. It was found that many researchers in different areas of the robotic field have made good use of PDAs within their systems. The small size of the device and the many built-in features that can almost instantaneously be added to a robot system were the most common reasons for choosing the device. PDAs have been used in many different ways without much thought given to how they fit within a formal control architecture. The dissertation grouped and categorised projects according to the hardware, software and communications strategies employed.

9.1.3 Distributed Control vs. System Performance

In order to determine whether distributing control intelligence has a positive influence on system performance a framework was designed and discussed in Chapter 4. The framework implemented a hybrid robot control architecture, but added to this by implementing each control layer on a physically distributed hardware component. The suggested framework was implemented using the communications architecture detailed in Chapter 7, together with a mobile platform developed in Chapters 5 and 6. The effects on system performance was evaluated in Chapter 8.

9.2 Meeting the Desired Objectives?

Section 1.3 presented three objectives for the suggested framework and implementation. The following sections evaluate to what extent the proposed framework meets these requirements.

9.2.1 Considerations Using a PDA in a Robot System

Considering how best to make use of a PDA within a mobile robot system was formed with the knowledge gained from past projects evaluated in Chapter 3 together with the features listed in Chapter 2. The PDA is a small, powerful device that incorporates a rich and familiar user interface. At this point in time, the PDA's processing, memory and battery life cannot match a desktop or laptop PC. It was determined that processing intensive tasks such as image processing is currently best done on a PC for both speed and memory considerations. Distributing the right type of tasks to the PDA is therefore important to ensure the success of the control system as a whole.

9.2.2 Suitable Control to Distribute to a PDA

A hybrid three-layered control architecture was chosen for the framework design. The three layers of the classic model run in parallel, with each higher layer handling tasks with greater acceptable latency periods. Within these layers the middle, or local planning layer, seemed to have the largest scope for implementation on the PDA. This decision was due to:

- The PDA not being directly attached to the sensors or actuators, thus inducing a delay unsuitable for the reactive layer.
- The PDA lacking memory and processing power to do the relatively slower and more deliberative planning tasks of the global planning layer.

9.2.3 System Performance

A mobile platform was designed and developed through Chapters 5 and 6 in order to interact with the world model through the sensors and actuators implemented on the platform, but also to implement the reactive layer of the three-layered control architecture. The PDA, performing local planning tasks, communicated with the mobile platform through a Bluetooth communications link and with the global planning layer on the PC through WLAN. System performance was measured using a single test scenario for the office robot, implemented in two configurations. The first has the PDA, as in many projects, as a relay for control commands from a remote PC to the mobile platform. The second implements the proposed framework with control intelligence distributed among the three hardware components.

The results obtained, both for power consumption and response time, supports the hypothesis in favour of distributing intelligence to a PDA within a PDA-incorporated robot system design. Distributing control intelligence to the PDA is faster and saves power when compared to the popular use of a PDA as message forwarding device.

9.3 Implementation Issues

Implementing the selected hardware components resulted in a number of challenges. Some could be overcome while others could not.

Using a commercially available RC car as the mobile platform had many advantages, as indicated in Chapter 5, but also brought a number of challenges:

- The use of parallel instead of Ackerman steering has a number of negative effects on the accuracy of the steering.

- The RC car has a large turning radius as well as large tires that are not fixed tightly to the body, causing more accuracy problems in steering the robot.

- The RC car came with an unusable servo that had to be replaced. Replacing parts on the plastic, rigid body of the RC car was a difficult task.

- Due to the large turning radius only 38 degrees of the 60 degree operating angle of the servo motor could be utilised. The software needed to compensate for these limits.

- Attaching the servo motor in a necessary off-centre position with respect to the shaft, meant that a new centre position and resulting PWM pulse adjustment had to be calculated. This resulted in the central position not being at 1.5 ms high pulse as is the industry standard, but around 1.48 ms. The software needed to compensate for the zero-shift.

- Low-cost RC cars come with low-cost DC motors, meant for running at full speed. Replacing the drive motor was not an option due to mounting limitations and gear assembly. This resulted in the drive motor struggling to move from a starting position at 50% duty cycle due to inertia within the motor coils.

- Noise induced by the drive motor was reduced by:

  1. Placing capacitors across the winding of the motor.
  2. Twisting the power supply wires together.
  3. Separating the analog and digital supplies by using two battery packs.
  4. Placing an inductor on the digital supply.

- The mobile platform cannot come to an immediate dead stop as there is a stopping time involved. The software takes this into account, especially when switching between driving directions, by first stopping the robot and after a delay obtained through trial and error starting the motor in the opposite direction.

In addition, the GP2D12 IR distance sensors were found to be very noisy. A 100 uF capacitor was placed across the supply, as close to the sensor as possible, to reduce the effect. This was effective in producing a variance over 1000 samples taken to be in the order of 0.0007. The sensors also come with a smaller-than-normal connector which made interfacing difficult. In order to connect with standard .1mil headers some modification was required. As a solution, a three-pin male header was soldered to the back of each Sharp IR sensor's PCB.

The first KC-Wirefree Bluetooth module was faulty and the second came incomplete. Only the third module purchased could do the job.

9.4 Research Contribution

This research contributed to the field of mobile robotics and specifically to wireless mobile robot control. Through a thorough literature study, many PDA-based robot systems were identified and classified in the way in which each project made use of the subsystems as well as incorporated control intelligence. This grouping and classification showed the breadth of projects within different research topic areas that have chosen to use a PDA within their designs. Bringing together a new area within mobile robotics – PDA-incorporated robot control.

A novel framework was discussed in great detail, indicating how intelligent control processes can be distributed among physically distributed hardware components. The framework also included how these may interact using a hybrid communications architecture.

These contributions have been communicated through the publication in Appendix A. The research also identified several interesting areas worthy of further investigation.

9.5 Further Research

The work contained in this dissertation could be extended by future research initiatives in the following ways:

- The designed framework does not include how the tasks should be structured within each layer. Task execution and arbitration across the control layers, and thus processing elements, could be added to the design laid out in this work.

- The implementation and tests performed in this work was on a single control task that can typically be expected of an office robot. Implementation on a more capable mobile platform with the addition of more tasks on all the control layers can be done. This will provide scope for more tests to be performed, making a more definitive case for (or against) the use of a PDA within a mobile robot system.

- The communications protocol was designed with the popular field of swarm robotics in mind, having the capability to support a Bluetooth master with several slave nodes. An interesting topic for future research would be to investigate a case with one deliberative PC and one or more PDAs each with a number of mobile platforms associated. This would extend the framework developed in this work to a pyramid-like layout.

9.6 Epilogue

The author trusts that this work has raised the awareness of the issues within the domain of this discourse. It is also hoped that the possibility of using technologies outside their field of operation, as well as improving existing ways, have been shown. The author would like to conclude this research by expressing the hope that this work will stimulate further work in the subject area.

Part V

Appendices

Appendix A

Academic Paper

An academic paper based on the research contained in this dissertation was accepted at the COMA'10 International Conference on Competitive Manufacturing. Details of the paper:

> Ophoff, M. and Van Niekerk, T. (2010).
> PDA-Bots: How Best to Use a PDA in Mobile Robotics,
> *COMA'10 International Conference on Competitive Manufacturing, Stellenbosch, South Africa, 3–5 February 2010*,
> `http://www.coma.org.za`

PDA-Bots: How Best to Use a PDA in Mobile Robotics

M. Ophoff, T. I. Van Niekerk
Department of Electrical and Mechatronic Engineering,
Nelson Mandela Metropolitan University, South Africa

Abstract

PDAs (personal digital assistants) has recently become a popular component in mobile robots. This compact processing device with its touch screen, variety of built-in features, wireless technologies and affordability means that a PDA can perform various roles within a robotic system. Applications include low-cost prototype development, rapid prototyping, low-cost humanoid robots, robot control, robot vision systems, algorithm development, HRI (human-robot interaction), mobile user interfaces as well as wireless robot communication schemes. Limits on processing power, memory, battery life and screen size impacts the usefulness of a PDA in some applications. No comparison of the advantages and disadvantages of the different strategies and resulting architectures exist. This makes it difficult for designers to decide on the best use of a PDA within their mobile robot. This paper examines and compares the available mobile robot architectures. A thorough literature study identified robot projects using a PDA and examined how the designs incorporate a PDA and what purpose it fulfils within the system it forms part of. The paper categorizes the architectures according to the role of the PDA within the robot system. It concludes that using a distributed control system architecture makes optimal use of the rich feature set gained from including a PDA in a robot's design and simultaneously overcomes the device's inherent shortcomings. This paper describes the use of the distributed control system architecture in a novel way through the choice of wireless connection scheme and strategy for the distribution of intelligence across processing elements in a mobile office robot.

Keywords

Mobile Robotics, Personal Digital Assistant, Robot Architectures, Robot Control, Distributed Intelligence

1 INTRODUCTION

Mobile robotics continue to grow in popularity within the research and hobbyist communities. Mobile robotics is itself a hot research topic but they are also valuable research tools for overlapping fields such as intelligent manufacturing [1], artificial intelligence and human-robot interaction to name but a few. Competitions such as RoboCup [2] have also encouraged further development within the field.

Mobile robots are robots that have the ability to move around in their environment, other than industrial robots which usually attach to a fixed surface. The task of roaming in a dynamically changing environment safely and accurately while carrying out meaningful tasks requires a mobile robot to have at least four basic components in its hardware architecture:

1. A hardware platform, or body, that houses all the other robot components.
2. A drive system that allows the robot to move from point A to point B. It usually consists of a combination of motors and either wheels, tracks or legs.
3. Several actuators and sensors that enable the robot to act on its environment as well as gain information from it.
4. A brain that interprets sensory information, navigates the robot within the working environment, controls actuator actions and monitors robot health. The robot's brain is often one or more PEs (processing elements) and can combine both onboard and remote PEs. Popular choices for robot PEs include desktop PCs, laptop PCs, embedded PCs, microcontrollers, and LEGO RX bricks.

Advances in technology inadvertently provide robot designers with an ever-expanding range of choices to make their robots smaller, more capable and cheaper to produce. One example is how the PDA evolved into what is today a powerful palm-sized processing and UI (user interface) tool. Ever resourceful, robotisists have used this device in unique ways to improve their robot designs. Making them more capable (intelligent), remotely accessible, smaller and lighter, easier to communicate with and more cost-effective.

This paper looks at the different ways in which PDAs have become part of mobile robot architectures. To understand why PDAs are such a popular choice for developers of small, low-cost mobile robots Section 2 gives some of the common features of modern PDAs. Section 3 describes the five categories of PDA-bots - robots that make use of a PDA within its design in some way.

Our research focuses on using a PDA as a parallel processing device that also allows maximum flexibility in PDA-bot design. We discuss a distributed control architecture to accomplish this goal in Section 4. Finally, Section 5 concludes the paper.

2 PERSONAL DIGITAL ASSISTANTS

2.1 What is a PDA?

PDAs, also called handhelds, have been around since the late 1980s and have since developed into devices that provide users with a rich feature set that converges the different technologies of personal information manager, portable PC, mobile phone and wireless connectivity into a single portable unit.

PDAs have several features that are now common to most devices:

- The microprocessor is the brain of the PDA and coordinates all the functions according to programmed instructions. PDAs mostly use smaller, cheaper microprocessors when compared with desktop and laptop PCs.
- Memory (RAM, ROM and Flash) while some devices have tiny MicroDrive hard drives.
- Wireless connectivity through WLAN, Bluetooth, IR and GPRS/3G.
- Several input methods that include a touch sensitive display, buttons and optionally a QWERTY keyboard.
- Power supply.
- Able to synchronize with a standard PC.
- SDIO card slot or slots to add peripherals to the device.

Today's PDAs does much more than managing personal information such as contacts, appointments and to-do lists. They can also connect to the Internet, have a GPS receiver built-in, run multimedia software and have a built-in mobile phone [3]. Other functionality might include a built-in camera, an accelerometer, the ability to read business documents in various formats, software for playing music, browsing photos and, recording and viewing video clips.

2.2 Why use a PDA in a Mobile Robot System?

Mobile robot developers have just as many reasons for choosing to use a PDA within their designs as features that come built into these devices.

The authors in [4] found that using a PDA gives them a small, lightweight robot interaction device that is also portable, robust and affordable. In contrast, single board computers used in many robot systems lack a display and possibilities for user I/O, need a power supply, wireless communication, housing, and cooling.

The touch sensitive screen combined with a stylus allow users to interact using touch [5,6]. It also allows novice users to control a robot through a familiar environment, minimizing the need for training [5,7]. The PDAs UI also equip the robot with debugging and development advantages over embedded devices [8]. All of this allows for rapid development of multimedia applications for robots [9].

The built-in wireless technologies such as Bluetooth and WLAN allow for remote operation "anywhere and any time" [10,11,7,12] and gives "seamless network coverage" to mobile agents and operators [13].

The authors in [9] and [14] found it to be an ideal device for use in cost and size limited platforms. It is also easy to attach to small robots through one of the multiple interface options available [15]. This leads to powerful small robots that are safer and cheaper when compared to larger, heavier robots [8].

The PDA allows for integrating high-level planners and implementation of computationally expensive algorithms [1]. This means that it can give simple robots that usually only consist of one or more microcontroller-based control boards the processing power and vision sensor capacity for use as autonomous robots [15,16].

3 PDA ROBOTS

From the many existing PDA-bots it is possible to identify five general ways in which PDAs are used across the different application areas:

1. main controller;
2. message forwarder;
3. teleoperation device;
4. part of a multimodal interface;
5. part of a distributed control scheme.

Sections 3.1 through to 3.5 define each of the above categories. Table 1 summarises the various PDA-bots in terms of hardware architecture and control software application.

3.1 Main Controller

This architecture (shown in Figure 1) uses the PDA mainly for its processing abilities. The PDA connects through a direct if not physical link to the robot base either through an IR or serial link. The robot base houses all the robot's actuators and sensory equipment and has an interfacing board and controller that can directly connect to these parts. An onboard controller is a necessary part of any PDA-based robot system. A PDA cannot connect directly to the robot's sensors and actuators and therefore some form of interfacing hardware is required. The onboard PE performs basic control algorithms for the actuators, or basic data processing algorithms for the sensors. It also manages the needed communication tasks to communicate with the PDA.

Processing done on the PDA may include interpreting sensory information, calculating motor commands, image processing and managing the communications link with the onboard embedded controller. The tasks performed will directly relate to the level of autonomy implemented, the robot's working environment and its role.

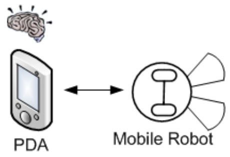


Figure 1 - PDA as the main controlling device.

3.2 Message Forwarder

Many PDA-bots include a PDA into their designs, with the sole purpose of adding a wireless link between a remote processing device and the physical robot. The PDA provides this link with little effort, hardware changes and minimal added cost to the developer.

Here a remote PE connects wirelessly to the PDA. The PDA in turn connects to the robot base and the onboard controller. The PDA relays control messages (usually without any form of abstraction or modification) from the remote PE to the onboard processor of the robot. Likewise, sensory information is sent from the onboard unit to the PDA, which then sends the information to the remote PE.

Control intelligence is implemented either solely on the remote PE (a PC or another PDA) or divided between the remote and local, onboard processor. Figure 2 gives a representative architecture of this type of implementation.

The PDA is generally connected through a direct, physical link to the interfacing circuit and PE on the robot platform. Exceptions are the WiMo [17] and NiVEK JD [18] robots which make use of a Bluetooth link between the PDA and robot base.

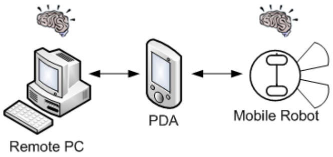


Figure 2 - PDA as a message forwarding device.

3.3 Teleoperation Device

Teleoperation is the remote control of a robot. PDAs have become a popular teleoperation device because of its built-in and user-recognisable UI that includes a touch screen that's usually in colour. A few buttons and QWERTY keyboard also come with many PDA models. The built-in wireless technologies enable the UI to connect wirelessly to the robot platform. Figure 3 shows a general representation of the architecture used by PDA-teleoperated robots.

The PDA connects wirelessly to the robot base. An exception is the robot in [19] where the PDA connects wirelessly to an intermediary PC which connects to the robot base. A UI on the PDA allows the robot operator to send control commands remotely to the robot. Sometimes sensor data is also displayed on the screen. Usually no processing is done on the teleoperation device (PDA) itself. All control intelligence is implemented on the robot platform's onboard controller.

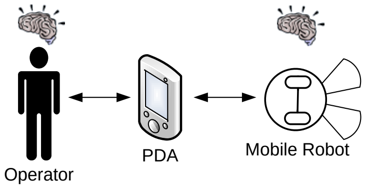


Figure 3 - PDA as a teleoperation device.

3.4 Part of a Multimodal Interface

Some robots use a PDA as part of a multimodal interface. This means the PDA provides the operator with an optional or extra interface to the robot. Here a UI is always present on the PDA. Other user interfaces may include a UI on a remote PC or a joystick. The operator has the choice of which UI or combination of UIs to use. Often the PDA does not connect directly to the robot's onboard controller, but rather to a user interface module which may be running on the robot's onboard controller or on a remote PC. This module first consolidates the user inputs through the multiple possible sources such as touch, gesture and speech before giving the command to the subsequent responsible module. Figure 4 shows a typical representation of the architecture used by robots in this category.

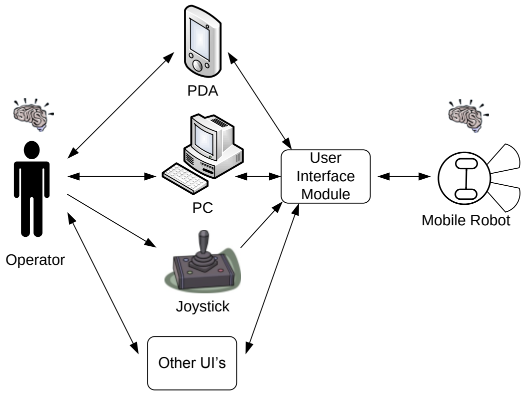


Figure 4 - PDA as part of a multimodal interface.

3.5 Part of a Distributed Control System

A distributed control system, divides the control software between all the available PE within the system. Figure 5 shows three PEs each with intelligent control software implemented. The tasks implemented on each PE depend on the robot application and the capabilities of the various PEs.

This architecture allows multiple tasks to be done in parallel and thus increasing responsiveness of the robot. There is however a limit to the number of PEs that can be used in this manner, as the addition of each extra PE adds to the complexity and time needed to manage the communications and data between the different PEs [20].

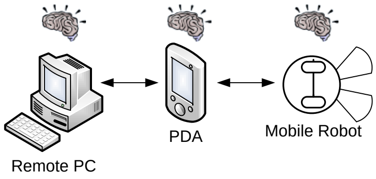


Figure 5 - PDA as part of a distributed control system.

3.6 PDA-bot Projects

Table 1 shows a table summarising the reviewed PDA-bots. It shows their use of and combination of different processing devices. Use of wireless technology and implementation of control software and user interfaces are also shown. Robota is the only robot listed that does not make use of a mobile platform in its design, therefore it has no *Level of Autonomy* and instead has five degrees of freedom.

It is interesting to note how many of the projects are in support of HRI research. In this area some use the PDA to provide a novel remote control (teleoperation) device. Others use the PDA as a means of giving the user feedback from the robot and allowing user response and direction. And some use it as part of a multimodal set of I/O devices, where the PDA is an optional input device. Here the PDA needs to be wirelessly connected to the robot base and was a critical consideration in the development of our architecture described in Section 4.

It is interesting to see how few projects make use of the PDA as part of a distributed control system. The FLIP robot is one of only two examples of projects that use distributed processing in their designs. FLIP makes use of three processing elements in a distributed fashion implementing intelligent control functions on all three (the RCX brick on the LEGO-based platform, the PDA itself as well as on the remote PC). The authors' reason for imposing this distribution was not so much a conscious effort but forced for two reasons. First the LEGO Mindstorms RCX brick is unsuitable to carry out multi-robot systems research, with severe limits on both memory and processing power. Secondly, the infrared port of the RCX brick only allows line of sight communication between robots restraining the possibility of using cooperative planning. Adding the PDA to their LEGO robot increased the memory and processing power of their platform as well as providing WLAN for long-range, out of sight, communications.

The Caddie Paradigm project does not concern multi-robot systems, but rather teleoperation. It may be that they too implemented some intelligent processing on the PDA due to the inherent limitations on the LEGO RCX brick that they also chose to use on their robot platform.

Other factors limiting the usefulness of the PDA in its chosen capacity within projects include the inherent latency between the time sensing the command via a PDA interface and the robot carrying out the command. This means operators need special training to cope with the delays [7].

The PDA screen has a low contrast when in power saving mode making it difficult to read the object on the screen without first activating the screen again by tapping somewhere on the screen [4, 13].

Projects that have decided to use the IR port of the PDA were necessitated to write PDA-specific IrDA handlers [1].

The limited resources (memory, processing power, processing speed) of the PDA when compared with standard PCs forced the authors in [9] to implement only simple vision and speech recognition algorithms and those in [8] to develop optimized algorithms for use on the PDA.

4 DEFINING AN OPTIMUM ARCHITECTURE

4.1 Parallelism through Abstraction

Implementing all intelligence on a single processing unit such as a PDA, limits the functionality of the overall system. This puts restraints on navigation and planning capabilities, due to memory and processing limits, as has also been noted by many of the developers in past projects. Distributing the required processing of a mobile robot across multiple PEs would increase overall throughput, and using multiple processors provides the robot the opportunity to take advantage of parallelism for improved throughput [21]. Few projects have made use of a distributed control architecture.

Parallelism is the process of doing multiple tasks simultaneously. The authors in [20] shows that there is no robot architecture that is perfectly parallel with designers choosing single areas to implement in parallel. They distinguish between eight ways in which parallel processing can be implemented within robotics. One such way is parallelism on the abstraction level. The control architecture of robots is often divided into levels according to the degree of abstraction of processed data and response time. The layers, performing tasks which have different response times, are considered to work simultaneously and semi-independent from each other, and can be implemented on different PEs.

Figure 6 show a typical representation of the hybrid architecture which usually has three layers – one reactive and two deliberative with data presented more abstractly to each following layer [22]. This architecture promotes performing efficient low-level control and connections with high-level reasoning. Many architectures use the hybrid theory and although defining the layered structure varies and the interlayer mechanisms differ, the basic role of each layer is similar. The reactive layer performs hard real-time tasks and interacts directly with the sensors and actuators of the robot. It is also responsible for the physical movement of the robot. The middle layer (often called the sequencer) does soft real-time tactical tasks such as local navigation. The deliberative layer does the most processing intensive and slowest tasks. Tasks done in this layer may include strategic and global planning or navigation. Here the robot can reflect about its interaction with its environment and make its own models and plans based on what it learnt from information received.

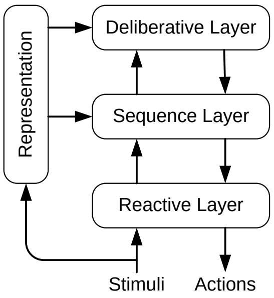


Figure 6 - Traditional hybrid control architecture.

4.2 Hardware Implementation

Considering the different uses robot projects have for the PDA we believe that the hardware layout shown in Figure 7 will be the most flexible and computationally efficient. Here the PDA connects to an onboard controller through a Bluetooth link and to a remote PC through WLAN. Except for the most basic models, both these wireless technologies are available on most modern PDAs. This also overcomes the disadvantages of using IR. The wireless connection with the onboard controller provides the needed flexibility to allow implementing a UI on the PDA. The Bluetooth provides several advantages over WLAN for this connection including 1) less overhead and easier to implement on devices such as microcontrollers that commonly act as the onboard controllers on small mobile robots and 2) uses less power – an important consideration for a battery operated device such as a PDA.

We consider the operating distance of Bluetooth sufficient for remote teleoperation and monitoring of the robot. The WLAN link between the PDA and remote PC provides a longer range of communication as well as faster up and download speed with respect to Bluetooth that is necessary when working with, for example, video data.

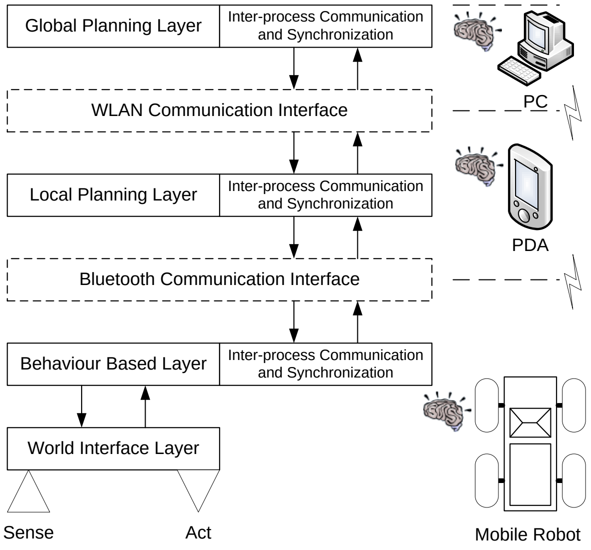

Global Planning Layer
Inter-process Communication and Synchronization
WLAN Communication Interface
PC
Local Planning Layer
Inter-process Communication and Synchronization
PDA
Bluetooth Communication Interface
Behaviour Based Layer
Inter-process Communication and Synchronization
World Interface Layer
Sense
Act
Mobile Robot

Figure 7 - Distributed control and hardware architecture.

4.3 Software Implementation

The control software is implemented according to the theory of abstraction as well as the layered architecture of the popular hybrid control system architecture. Figure 7 shows the three PEs (the mobile robot with its onboard controller, a PDA and a PC) and how the layered architecture in Figure 6 is implemented in a wirelessly distributed fashion.

The onboard controller is the direct interface between the various actuators and sensors on the mobile platform. It is also here that time critical tasks can best be implemented – communicating with higher level PEs necessitating additional time delays before action can be taken. The behaviour based layer and its related functions are implemented on the onboard controller. The local planning and navigation is done by the wirelessly connected PDA. Global planning, which can tolerate the greatest response delays, is done by a remote PC. The remote PC also does all tasks that require fast processing and large amounts of memory.

5 CONCLUSIONS

The PDA's processing capability is underutilised when it is used to only forwards control messages from the main (often remote) control station to the onboard platform controller. Often these messages are motor specific commands for the individual motors and servos on the robot base, increasing the latency between sending a command and its execution. The PDA is also not the ideal device to fulfil heavy processing demands, such as image processing requires. This leads to not implementing complete systems because of the lack of resources, or settling for a less desirable output that can successfully be implemented on the PDA. Incorporating a wirelessly connected, remote PC is common to many projects, but with many only using

		Hardware			Control Intelligence			UI			
	Robot/ Project Name	Remote PC?	PC-PDA interface	PDA-Robot Interface	PC	PDA	Onboard Controller	PDA User Interface	PC Monitoring/ Debugging	Level of Autonomy	Area of Research
Controller Device	PDA Robot [23]	√	WLAN	IR	X	√	X	√	√	Semi-Autonomous	Robot Control Systems
	Robota [9]	X	−	RS232	−	√	X	√	−	N/A	Human-Robot Interaction
	Toni [15]	√	WLAN	RS232	X	√	X	X	√	Autonomous	Humanoid Robots
	NimbRo RS [16]	√	WLAN	IR	X	√	X	X	√	Autonomous	Humanoid Robots
	Abarenbou and DaoDan [14]	X	−	RS232	−	√	X	X	−	Autonomous	Humanoid Robots
	PDA Based Robot Vision System [8]	X	−	RS232	−	√	X	√	−	Autonomous	Image Processing
Message Forwarder	Mini-Whegs Robot [24]	√	WLAN	RS232	X	X	X	X	√	Teleoperated	Wireless Communications Networks
	WiMo [17]	√	WLAN	BT	X	X	X	X	√	Teleoperated	Robot Control Systems
	NiVek J.D [18]	√	WLAN	BT	X	X	X	X	√	Teleoperated	Robot Control Systems
	PEWIR Robot [25]	√	WLAN	RS232	X	X	X	X	√	Teleoperated	Wireless Communications Networks
	Calligrapher Robot [10]	√ (PDA)	BT	RS232	√	X	X	X	X	Teleoperated	Robot Control Systems
	PDA LEGO Robot [11]	√	WLAN	IR	X	X	√	X	√	Autonomous	Robot Navigation
Teleoperation	PdaDriver [7]	X	−	WLAN	−	X	√	√	−	Guarded-Teleoperated	Teleoperation Interfaces
	PDA Touch Screen Teleoperation [4]	X	−	WLAN	−	X	X	√	−	Teleoperated	Human-Robot Interaction
	EMG-based Morse Code [19]	√	BT	None	√	X	√	√	√	Semi-Autonomous	Human-Robot Interaction
	COMMBOTS [12]	√	−	GPRS	X	X	X	√	√	Teleoperated	Human-Robot Interaction
Multimodal Interface	SCOUT [26]	√	RS232	None	√	X	√	√	X	Teleoperated	Human-Robot Interaction
	Laser Scanner Interface [5]	X	−	WLAN	−	X	√	√	X	Semi-Autonomous	Human-Robot Interaction
	Sketch Interface [6]	√	WLAN	None	√	√	X	√	X	Teleoperated	Human-Robot Interaction
	Pocket CERO [13]	√	WLAN	None	√	X	√	√	√	Semi-Autonomous	Human-Robot Interaction
Distributed Control	FLIP [1]	√	WLAN	IR	√	√	√	X	√	Autonomous	Intelligent Manufacturing
	Caddie Paradigm Robot [27]	√	WLAN	IR	√	√	√	X	X	Guarded-Teleoperated	Teleoperation Interfaces

Table 1 - Summary of PDA-bots.

the PC to give goal directives or motor commands. Some make use of a remote PC to perform debugging and logging of important parameters, while others implement a UI on the PC. Few use the PC's processing power, speed and memory, implementing all the control software either on the onboard controller or PDA.

Our architecture attempts to maximise use of all the PDA's built-in abilities, including user interface strengths and processing capacity. A way in which control tasks can logically be distributed across three PEs is shown using the typical three layer hybrid control architecture and the theory of data abstraction. This allows for the most time critical tasks to be done by the onboard controller and the least time critical and most processing and memory intensive tasks to be done by a remotely connected PC.

6 REFERENCES

[1] Jensen, L. K., Kristensen, B. B., & Demazeau, Y. (2005). FLIP: Prototyping multi-robot systems. *Robotics and Autonomous Systems* (53), 230-243.

[2] The RoboCup Federation. (2009). *RoboCup World Championship*. Available from: http://www.robocup.org. (Last cited May 2009)

[3] Carmack, C., & Freudenrich, C. (2008a). *How PDAs Work.* Available from: http://communication.howstuffworks.com/pda.htm. (Last cited July 2008)

[4] Keskinpala, H., Adams, J., & Kawamura, K. (2003). PDA-Based Human Robotic Interface. In *IEEE International Conference on Systems, Man and Cybernetics* (pp. 3931–3936).

[5] Perzanowski, D., Schultz, A. C., Adams, W., Marsh, E., & Bugajska, M. (2001). Building a Multimodal Human-Robot Interface. *IEEE Intelligent Systems*.

[6] Skubic, M., Bailey, C., & Chronis, G. (2003). A Sketch Interface for Mobile Robots. In *IEEE International Conference on Systems, Man and Cybernetics.*

[7] Fong, T., Thorpe, C., & Glass, B. (2003). PdaDriver: A Handheld System for Remote Driving. In *IEEE International Conference on Advanced Robotics 2003.*

[8] Jantz, S., & Doty, K. L. (2002). PDA Based Real-Time Vision for a Small Autonomous Mobile Robot. In *Florida Conference on Recent Advances in Robotics.*

[9] Calinon, S., & Billard, A. (2003). *PDA Interface for Humanoid Robots* (Tech. Rep.). Lausanne, Switzerland: Swiss Institute of Technology.

[10] Kovan Research Lab. (2006). *Calligrapher Robot.* Available from: http://www.kovan.ceng.metu.edu.tr. (Last cited June 2008)

[11] Buschmann, C., Müller, F., & Fischer, S. (2003). *Grid-Based Navigation for Autonomous, Mobile Robots* (Tech. Rep.). Institute of Operating Systems and Networks, Technical University of Braunschweig, Germany.

[12] De Ipina, D., V´azquez, I., De Garibay, J. R., & Sainz, D. (2005). GPRSbased Real-Time Remote Control of MicroBots with M2M Capabilities. In *Wireless Information System.*

[13] Hüttenrauch, H., & Norman, M. (2001). PocketCERO mobile interfaces for service robots. In *Mobile HCI, International Workshop*

on Human Computer Interaction with Mobile Devices.

[14] Baltes, J., McCann, S., & Anderson, J. (2006). *Humanoid robots: Abarenbou and DaoDan, RoboCup 2006-Humanoid League Team Description*. Bremen, Germany.

[15] Behnke, S., Müller, J., & Schreiber, M. (2005). Toni: A Soccer Playing Humanoid Robot. In *The 9th RoboCup International Symposium.*

[16] Behnke, S., Langner, T., Müller, J., Neub, H., & Schreiber, M. (2004). NimbRo RS: A Low-Cost Autonomous Humanoid Robot for Multi-Agent Research. In *Workshop Proceedings: Methods and Technology for Empirical Evaluation of Multi-Agent Systems and Multirobot Teams.* 27th German Conference on Artificial Intelligence, Ulm.

[17] Cross, B. (2007). *WiMo the Windows Mobile Robot.* Available from: http://www.wimobot.com/. (Last cited July 2008)

[18] Wolf, K. (2007). *NiVek J.D.'s Mainden Voyage.* Available from: http://www.efficientcoder.com/CategoryView,category,Robotics.aspx. (Last cited July 2008)

[19] Nilas, P. (2005). A PDA-bsed Human-Robot Interaction using EMG-based Modified Morse Code. In *International Conference on Computational Intelligence for Modelling, Control and Automation.*

[20] Henrich, D., & Honiger, T. (1997). Parallel Processing Approaches In Robotics. *IEEE International Symposium on Industrial Electronics.* Guimarães, Portugal: IEEE.

[21] Hu, H., & Brady, M. (1996). A parallel processing architecture for sensor-based control of intelligent mobile robots. *Robotics and Autonomous Systems* (17) , 235-257.

[22] Satana, P. F. (2005). Survival Kit: A Bottom Layer for Robot Navigation, a Thesis. Lisboa: New University of Lisbon.

[23] Williams, D. H. (2003). *PDA Robotics: Using Your Personal Digital Assistant to Control Your Robot*. McGraw-Hill.

[24] Joshi, A. (2003). *A PDA Enabled Wireless Interface for a Mobile Robot*. Unpublished master's thesis, Case Western Reserve University.

[25] Langeveld, R., Narang, A., & Buenaventura, C. (2005). *PDA Enabled Wireless Interfaced Robot* (Senior Design Project). University of California, Riverside.

[26] Rybski, P. E., Burt, I., Dahlin, T., Gini, M., Hougan, D. F., Krantz, D. G., et al. (2001). System Architecture for Versatile Autonomous and Teleoperated Control of Multiple Miniature Robots. In *IEEE International Conference on Robotics.*

[27] Lemoine, P., Gutiérrez, M., Thalmann, D., & Vexo, F. (2004). The "Caddie Paradigm": a Free-Locamotion Interface for Teleoperation. In *CAPTECH 2004, Workshop on Modelling and Motion Capture Techniques for Virtual Environments.*

7 BIOGRAPHY

Madri Ophoff is an MTech Engineering student at the Nelson Mandela Metropolitan University. Her specific focus and area of research interest is in the wireless control of mobile robots.

Theo van Niekerk obtained his DTech in Eng from the PE Technikon. His research interests include Mobile Industrial Robotics, Automation of Manufacturing Systems, Process Control and Instrumentation. He teaches in the Department of Mechatronics.

Appendix B

PIC18F4620 Microcontroller Source Code

```
/******************************************************************************
 * Code for PIC18F4620-based mobile robot platform.
 * Author: M Ophoff
 * Date: 2010
 *****************************************************************************/
#include <18F4620.h>

#device HIGH_INTS=true
#device adc=8
#fuses HS,NOPROTECT,NOWDT,BROWNOUT,PUT,NOLVP
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

#priority CCP2, TIMER1, TIMER0

#byte PIR1 = 0xF9E
#byte T3CON = 0xFB1
#byte T0CON = 0xFD5
#bit TMR1IF = PIR1.0

#include <crc.c>

    /* Public interface */

 /* Initialisation and main*/
void main(void);
void Init(void);
void InitPins(void);
void InitTimers(void);
void InitAnalog(void);
void InitInts(void);

 /* Serial RX */
void SerialReceiveISR(void);
char BufferedGetc(void);
int ReceivePacket(byte *packet_ptr, int *length_ptr);
int ValidDataPacket(byte *packet_ptr, int packet_length);
void SetData(byte function_param, byte *packet_ptr);

 /* Serial TX */
void SerialTransmitISR(void);
void BufferedPutc(char c);
void CollectData(byte function_param, byte *data_ptr, int *data_length_ptr);
void BuildPacket(byte receiver, byte func_param, int1 ack, int data_length,
    byte *packet_data_ptr, byte *packet_length_ptr, byte *packet_ptr);
void SendPacket(int packet_length, byte *packet_ptr);

 /* Hash */
void InterpretHashCode(int16 hashing_value);
unsigned long DJBHash(char* str, unsigned int len);

 /* Timers */
void TIMER0Isr(void);
void TIMER3Isr(void);
void TIMER1Isr(void);

 /* Drive motor */
void StopDriveMotor(void);
void GoForward(void);
void GoBackward(void);
void BrakeDriveMotor(void);

 /* Encoder */
void EncoderISR(void);
void CCP2Isr(void);

 /* Robot */
void RobotAct(void);
void SetServo(signed int angle);
void DriveMotor(int1 direction, int pwmDuty);

    /* Constants */
 /* Operating modes */
#define AUTONOMOUS 1
#define TELEOPERATED 2
#define OPERATING_MODE AUTONOMOUS //set mode of operation here

 /* serial tx/rx */
#define BUFFER_SIZE 90  //circular buffer to hold 3 consecutive incoming messages
#define RX_PACKET_SIZE 32  //a single message can never be more than 32 bytes
#define TX_DATA_SIZE 20  //no more than 20 bytes in the payload of outgoing message
#define TX_BUFFER_SIZE  64
#define CHAR_DELAY  500  //time in milliseconds allowed to receive a char
#define MAX_UNACKNOWLEDGED_PACKETS  5

 /* packet structure */
#define START_BYTE 0x2a //'*'
#define ROBOT_ADDRESS 0x04 //combining robot and send address gives 0x24 - '$'
#define SEND_ADDRESS  0x02 //remote (pc's) address
#define CR 0x0D //'\r'
#define LF 0x0A //'\n'
#define BT_START_BYTE1 0x41 //'A'
```

```
#define BT_START_BYTE2 0x54 //'T'
#define BT_CARRIER_LOSS 0x23 //'#'
#define NUM_HEADER_BYTES  4 //Start, Addresses, Function Parameters and Data Length bytes
#define NUM_FOOTER_BYTES 4 //16 bit CRC, 2 stop bytes

 /* packet type and validity */
#define INVALID_PACKET 0
#define CONTROL_PACKET 1
#define BT_MODULE_PACKET  2
#define PACKET_TIMED_OUT  5
#define NO_IR_SENSORS  6
#define CLEAN_PACKET 1
#define INCORRECT_LENGTH 2
#define CRC_ERROR 3
#define WRONG_ADDRESS 4

 /* Function Parameters' bit masks */
#define BATTERY_MASK  0x01
#define CONTACT_MASK  0x02
#define IR_SENSOR_MASK  0x04
#define SPEED_MASK 0x08
#define ANGLE_MASK  0x10
#define DISTANCE_TRAVELLED_MASK 0x20
#define ACKNOWLEDGE_MASK 0x40
#define DISTANCE_TO_TRAVEL_MASK 0x01
#define ACKNOWLEDGE_BIT 6

 /* drive motor */
#define FORWARD TRUE
#define BACKWARD FALSE
#define PR2_VALUE  255
#define MOTOR_BW_ENABLE  PIN_C3
#define MOTOR_FW_ENABLE  PIN_C4
#define SLOW_SPEED  153 //60% duty cycle
#define MED_SPEED  179 //70% duty cycle
#define TOP_SPEED  204 //80% duty cycle
#define STOP_TIME 1500 //time in ms robot takes to come to rest (complete stop)

 /* data available */
#define bkbhit (rx_buffer_next_in != rx_buffer_next_out) //data available

 /* Contact Sensors */
#define BACK_LEFT 4 //connections to port B of the four touch sensors
#define BACK_RIGHT 5
#define FRONT_LEFT 6
#define FRONT_RIGHT 7

 /* Servo */
#define SERVO PIN_C0
#define CENTRE 7400
#define MAX_LEFT 8570 //Centre +1070 calibrate here
#define MAX_RIGHT 6330 //Centre -1070 calibrate here
#define MAX_ANGLE_LEFT 19
#define MAX_ANGLE_RIGHT 19

 /* Encoder */
```

```
#define WHEEL_CIRCUMFERENCE 2*(3.14*WHEEL_RADIUS)
#define WHEEL_RADIUS  70
#define ENCODER_RESOLUTION 1
#define DISTANCE_PER_TICK WHEEL_CIRCUMFERENCE/ENCODER_RESOLUTION
#define BytePtr(var, offset) (char *)((char *)&var + offset)

    /* look-up tables */
const int16 servo_table [] = {  0,  57, 114, 171, 228, 285, 342, 399, 456, 513, 570,
627, 684, 741, 798, 855, 912, 969, 1026, 1083, 1140}; //servoTable

const int16 duty_table [] = {   0,  3,  6,  8, 11, 13, 16, 18,
                                21, 23, 26, 29, 31, 34, 36, 39,
                                41, 44, 46, 49, 51, 54, 57, 59,
                                62, 64, 67, 69, 72, 74, 77, 80,
                                82, 85, 87, 90, 92, 95, 97,100,
                                102,105,108,110,113,115,118,120,
                                123,125,128,131,133,136,138,141,
                                143,146,148,151,153,156,159,161,
                                164,166,169,171,174,176,179,182,
                                184,187,189,192,194,197,199,202,
                                204,207,210,212,215,217,220,222,
                                225,227,230,233,235,238,240,243,
                                245,248,250,253,255}; //dutyTable

const byte ad_channels_table [] = {0,1,2,3,4,5,6,7,255}; //adChannelsTable

const int8 ir_range_table [] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 80, 76, 72, 68, 65, 62, 59, 57, 54, 52, 50, 48, 47,
                                45, 44, 42, 41, 40, 38, 37, 36, 35, 34, 33, 33, 32, 31, 30, 30,
                                29, 28, 28, 27, 26, 26, 25, 25, 24, 24, 23, 23, 23, 22, 22, 21,
                                21, 21, 20, 20, 20, 19, 19, 19, 18, 18, 18, 18, 17, 17, 17, 17,
                                16, 16, 16, 16, 16, 15, 15, 15, 15, 15, 14, 14, 14, 14, 14, 14,
                                13, 13, 13, 13, 13, 13, 13, 12, 12, 12, 12, 12, 12, 12, 12, 11,
                                11, 11, 11, 11, 11, 11, 11, 11, 10, 10, 10, 10, 10, 10, 10};
                                //irRangeTable

    /* global variables */
 /* Serial RX */
byte rx_buffer[BUFFER_SIZE]; //circular buffer to hold incoming data bytes
int rx_buffer_next_in = 0; //where next to store byte of data
int rx_buffer_next_out = 0; //where next to fetch byte of data
int1 remote_comms_link = FALSE;  //TRUE if communication is established with PDA
int1 timed_out = FALSE; //ensure system does not get stuck waiting for next byte
int1 acknowledge = TRUE;

 /* Serial TX */
byte tx_buffer[TX_BUFFER_SIZE]; //circular buffer to hold outgoing data bytes
byte tx_buffer_next_in = 0; //where next to store next byte of data in txBuffer
byte tx_buffer_next_out = 0; //where next to fetch a data byte from txBuffer
int tx_unacknowledge_count = 0; //keep track of how many consecutive packets
                                    //were not received correctly

 /* Contact Sensors */
int timer0_count = 0; //debounce time counter
byte contact_information = 0b00000000;
int1 stuck = FALSE;
```

```

 /* Servo */
int16 servo_val;
signed int current_angle = 0;
signed int angle;

 /* Drive motor */
int speed;
int1 direction = FORWARD;
int1 current_direction = FORWARD;
int1 stopped = TRUE;

 /* IR sensors */
int8 a2d_readings[8];

 /* Encoder */
int8 timer1_rollover_count = 0;
int32 isr_CCP_delta;
int1 capture_flag = FALSE;
int16 current_RPM;
int16 distance_travelled = 0;
int16 distance_to_travel = 0;

/*****************************************************************************
 $ Function: InitPins
 $ Synopsis: Initializes the PIC18F4620's port pins as inputs or outputs
 ****************************************************************************/
void InitPins(void)
{
    set_tris_a(0b11111111); /* Port A is all analogue inputs. */
    set_tris_b(0b11111111); /* Port B is all inputs */
    set_tris_c(0b10000000); /* Pins 0-6 are outputs. C7 (RX) is input. */
    set_tris_d(0b00000000); /* Port D is all outputs. */
    set_tris_e(0b00000111); /* Port E is analogue inputs. */
} /* init_pins */

/*****************************************************************************
 $ Function: InitAnalog
 $ Synopsis: Initializes the PIC18F4620's analogue port and sampling clock
 ****************************************************************************/
void InitAnalog(void)
{
    setup_adc_ports (AN0_TO_AN7|VSS_VDD);
    setup_adc(ADC_CLOCK_DIV_32);
}/* init_analogue */

/*****************************************************************************
 $ Function: InitTimers
 $ Synopsis: Initialises TIMER 0 1 and 2, CCP1 and CCP2 modules
 ****************************************************************************/
void InitTimers(void)
{
    set_timer0(63036); //set up timer 0 to interrupt every 1ms
    setup_counters(RTCC_INTERNAL,RTCC_DIV_2);

    setup_ccp1(CCP_PWM);
    setup_timer_2(T2_DIV_BY_1, PR2_VALUE, 1);

    setup_ccp2(CCP_CAPTURE_RE);
    setup_timer_1(T1_INTERNAL|T1_DIV_BY_8); //period of timer 1 = 1.6us per tick
    set_timer1(0);
}/* init_timers */

/*****************************************************************************
 $ Function: InitInts
 $ Synopsis: Initialises the various interrupts used as well as the global
            interrupt enable
 ****************************************************************************/
void InitInts(void)
{
    enable_interrupts(int_rda); //serial receive interrupt
    enable_interrupts(INT_RB); //contact sensor interrupt
    enable_interrupts(INT_TIMER0); //servo and contact sensor time base

    clear_interrupt(INT_TIMER1); //ensure interrupt flag bit is cleared
    clear_interrupt(INT_CCP2); //ensure interrupt flag bit is cleared
    enable_interrupts(INT_CCP2);
    enable_interrupts(INT_TIMER1); //provide time base for speed related calculations

    enable_interrupts(GLOBAL);
}/* init_ints */

/*****************************************************************************
 $ Function: Init
 $ Synopsis: Calls the initialisation functions amd initialises the steering
 motor position
 $ Interface
  return:
  params:
  global: servo_val
 ****************************************************************************/
void Init(void)
{
servo_val = CENTRE;
    InitPins();
    InitAnalog();
    InitTimers();
    InitInts();
} /* init */

/*****************************************************************************
 $ Function: DJBHash
 $ Synopsis: Calculates hash of the input using the DJB hashing algorithm
 $ Interface
    return: hash - the calculated hash value
    params: str - the input to calculate hash on
    len - the length of the input in bytes
    global:
 ****************************************************************************/
unsigned long DJBHash(char* str, unsigned int len)
{
   unsigned long hash = 5381;
   unsigned int i = 0;

   for(i = 0; i < len; str++, i++)
   {
      hash = ((hash << 5) + hash) + (*str);
   }
   return hash;
}/* DJBHash  */

 /*****************************************************************************
 $ Function: InterpretHashCode
 $ Synopsis: Determine the string value from the calculated hash. Use "hash table" to
 look up appropriate action. Will set whether the communications link is up or down.
 $ Interface
    return:
    params: hashing_value - the calculated hash value to compare
    global: remote_comms_link
 ****************************************************************************/
void InterpretHashCode(int16 hashing_value)
{
    switch(hashing_value)
    {
        case 13162: //"###NO CARRIER"
            remote_comms_link = FALSE;
            break;
        case 53632: //"AT-ZV Baudrate Changed"
            break;
        case 32306: //"AT-ZV BDAddress 00043e240895"
            break;
        case 941:   //"AT-ZV BondFail"
            break;
        case 45791: //"AT-ZV -BypassMode-"
            remote_comms_link = TRUE;
            break;
        case 55724: //"AT-ZV -CommandMode-"
            remote_comms_link = FALSE;
            break;
        case 13302: //"AT-ZV ConnectionDown"
            remote_comms_link = FALSE;
            break;
        case 57123: //"AT-ZV ConnectionUp"
            remote_comms_link = TRUE;
            break;
        case 32342: //"AT-ZV ResetPending"
            remote_comms_link = FALSE;
            break;
        case 17611: //"AT-ZV SPPConnectionClosed"
            remote_comms_link = FALSE;
            break;
        default: //"Unknown BT responce"
            break;
    }//end switch()
}/* interpret_hash_code */

 /*****************************************************************************
 $ Function: SerialTransmitISR
 $ Synopsis: Serial transmit interrupt routine. Transmits character from transmit
            buffer.
 ***************************************************************************/
#int_tbe
void SerialTransmitISR(void)
{
   if(tx_buffer_next_in != tx_buffer_next_out)
   {
      putc(tx_buffer[tx_buffer_next_out]);
      tx_buffer_next_out = (tx_buffer_next_out+1) % TX_BUFFER_SIZE;
   }
   else
      disable_interrupts(int_tbe);
}/* serial_transmit_isr */

 /***************************************************************************
 $ Function: BufferedPutc
 $ Synopsis: Places 1 character at a time into the transmit buffer.
 $ Interface
    return:
    params: c - character to place in buffer
    global:
 ***************************************************************************/
void BufferedPutc(char c)
{
   int1 restart;
   int next_in;

   restart = tx_buffer_next_in == tx_buffer_next_out;
   tx_buffer[tx_buffer_next_in] = c;
   next_in=(tx_buffer_next_in+1) % TX_BUFFER_SIZE;

   while(next_in == tx_buffer_next_out);
    tx_buffer_next_in = next_in;

   if(restart)
      enable_interrupts(int_tbe);
}/* BufferedPutc*/


 /***************************************************************************
 $ Function: SetData
 $ Synopsis: Updates the global control variables with the data received in the
 control packet.
 $ Interface
    return:
    params: function_param - bitmap indicating data in payload
    packet_ptr - reference to control packet data
    global:

Structure of function_param byte for setting data:
+---+---+---+---+---+---+---+---+
: 7 : 6 : 5 : 4 : 3 : 2 : 1 : 0 :
+---+---+---+---+---+---+---+---+
where:
0 - DistanceToTravel
```

```
1 - RFU
2 - RFU
3 - Speed
4 - Angle
5 - DistanceTravelled
6 - acknowledge (1-acknowledge previous 0-not acknowledged)
7 - RFU (future use, 0 - possibly for a follow bit to indicate another
        function parameters byte to follow)
and 1 - include, 0 - not included
 ****************************************************************************/
void SetData(byte function_param, byte *packet_ptr)
{
    int i = 0;
    signed int temp;
    int data_start_pos = NUM_HEADER_BYTES; //position where first data byte is located

    i = data_start_pos;

    if((function_param & DISTANCE_TRAVELLED_MASK) != 0)
    {
        distance_travelled = (packet_ptr[i++]) << 8; //removed casting to (int16) here
        distance_travelled += packet_ptr[i++];
    }

    if((function_param & ANGLE_MASK) != 0)
    {
        angle = packet_ptr[i++];
    }

    if((function_param & SPEED_MASK) != 0)
    {
        temp = packet_ptr[i++];
        if (temp<0)
        {
            speed = (int)(-temp);
            direction = BACKWARD;
        }
        else
        {
            speed = temp;
            direction = FORWARD;
        }
    }

    if((function_param & DISTANCE_TO_TRAVEL_MASK) != 0)
    {
        distance_to_travel = (packet_ptr[i++]) << 8; //removed casting to (int16) here
        distance_to_travel += packet_ptr[i++];
    }
}
 /*****************************************************************************
 $ Function: CollectData
 $ Synopsis: Constructs the needed payload in response to a "Request command".
 $ Interface
  return:
  params: function_param - indicates what data should be included in payload
  data_ptr - refence to packet payload to which to add data
  data_length_ptr - reference to update with the resulting payload length
  global:

Structure of function_param byte for getting data:
+---+---+---+---+---+---+---+---+
: 7 : 6 : 5 : 4 : 3 : 2 : 1 : 0 :
+---+---+---+---+---+---+---+---+
where:
0 - battery voltages
1 - contact sensors
2 - IR sensors
3 - Speed and direction
4 - Angle
5 - Distance
6 - acknowledge (1-acknowledge previous 0-not acknowledged)
7 - RFU (future use, 0 - possibly for a follow bit to indicate another function
        parameters byte to follow)
and 1 - include, 0 - not included
*****************************************************************************/
void CollectData(byte function_param, byte *data_ptr, int *data_length_ptr)
{
    int i = 0;
    int ir_chan = 0;
    int16 extern_CCP_delta;
    int16 frequency;
    int16 RPM_plus_Direction;

    if((function_param & DISTANCE_TRAVELLED_MASK)!=0)
    {
        data_ptr[i++] = distance_travelled>>8;
        data_ptr[i++] = distance_travelled;
    }

    if((function_param & ANGLE_MASK)!=0)
    {
        data_ptr[i++] = current_angle;
    }

    if((function_param & SPEED_MASK)!=0)
    {
        if(stopped || stuck)
        {
            current_RPM = 0;
            RPM_plus_direction = 0;
        }
        else
        {
            disable_interrupts(GLOBAL);
      extern_CCP_delta = isr_CCP_delta;
      enable_interrupts(GLOBAL);
      frequency = ((625000L + (int32)(extern_CCP_delta >> 1))/ (float)extern_CCP_delta);
      current_RPM = (int16)(frequency * 60);

      if(direction==FORWARD)
            {
                RPM_plus_direction = current_RPM;
            }
            else
            {
                RPM_plus_direction = -current_RPM;
            }
        }

        data_ptr[i++] = RPM_plus_direction >> 8;
        data_ptr[i++] = RPM_plus_direction;
    }

    if((function_Param & IR_SENSOR_MASK) != 0)
    {
        for (ir_chan = 0; ir_chan < (NO_IR_SENSORS); ir_chan++)
        {
            if(a2d_readings[ir_chan] > 127)
                data_ptr[i++] = 10; //the closest accurate value possible
            else
                data_ptr[i++] = ir_range_table[a2d_readings[ir_chan]];
        }
    }

    if((function_param & CONTACT_MASK)!=0)
        data_ptr[i++] = contact_information;

    if((function_param & BATTERY_MASK)!=0)
    {
        data_ptr[i++] = a2d_readings[NO_IR_SENSORS];
        data_ptr[i++] = a2d_readings[NO_IR_SENSORS + 1];
    }

    *data_length_ptr = i;
}/* CollectData */

 /*****************************************************************************
 $ Function: BuildPacket
 $ Synopsis:  Constructs a packet according to the communications protocol
 $ Interface
  return:
  params: receiver - address of destination node
  func_param - data contained in the payload of packet
  ack - whether previous command was understood
  data_length - length of the payload
  packet_data_ptr - refernce to payload start
  packet_length_ptr - refernce to total lenth of packet
  packet_ptr - refernce to start of packet
  global:
 ****************************************************************************/
void BuildPacket(byte receiver, byte func_param, int1 ack, int data_length,
    byte *packet_data_ptr, int*packet_length_ptr, byte *packet_ptr)
{
    int i = 0;
    int16 CRC;
    int8 send_receive;
    int n = 0;
    byte function_parameters;

    function_parameters = func_param;

    packet_ptr[i++] = START_BYTE;
    //build send/receive address into a single byte
    send_receive = (receiver << 4)| ROBOT_ADDRESS;
    packet_ptr[i++] = send_receive;

    if(ack == TRUE)
        bit_set(function_parameters, ACKNOWLEDGE_BIT);
    else
        bit_clear(function_parameters, ACKNOWLEDGE_BIT);

    packet_ptr[i++] = function_parameters;
    packet_ptr[i++] = data_length;

    for(n = 0; n < data_length; n++)
        packet_ptr[i++] = packet_data_ptr[n];
    //calculate CRC
    CRC = generate_16bit_crc(packet_ptr, (data_length + NUM_HEADER_BYTES), CRC_CCITT);

    packet_ptr[i++] = CRC >> 8;
    packet_ptr[i++] = CRC;

    packet_ptr[i++] = CR; //stop chars
    packet_ptr[i++] = LF;

    *packet_length_ptr = i;
}/* BuildPacket */

 /***************************************************************************
 $ Function: SendPacket
 $ Synopsis: Buffers outgoing data by placing each character in turn into the
 transmit buffer
 $ Interface
    return:
    params: packet_length - length of data that needs to be sent
    packet_ptr - reference to data to be sent
    global:
 **************************************************************************/
void SendPacket(int packet_length, byte *packet_ptr)
{
    int i;

    for(i = 0; i < packet_length; i++) //send packet data
    {
        BufferedPutc(packet_ptr[i]);
    }
}/* SendPacket */

 /***************************************************************************
 $ Function: SerialReceiveISR
 $ Synopsis: Serial interrupt routine. Places incoming characters into a receive
            buffer
 **************************************************************************/
```

```
#int_rda
void SerialReceiveISR(void)
{
    char char_in;

    char_in = getc();
    rx_buffer[rx_buffer_next_in] = char_in; //get a byte and put it in buffer

    if(++rx_buffer_next_in == BUFFER_SIZE)  //increment counter
        rx_buffer_next_in = 0; //wrap buffer index
}/* SerialReceiveISR */

 /*****************************************************************************
 $ Function: BufferedGetc
 $ Synopsis: Fetches a byte from the receive buffer.
 $ Interface
    return: char_out - a single byte from the receive buffer
    params:
    global:
 ****************************************************************************/
char BufferedGetc(void)
{
    char char_out;
    unsigned int16 timeout = 0;

    // wait until data available or timeout occurs
    while(!bkbhit && (++timeout < (CHAR_DELAY * 100)))
        delay_us(10); //max time before time-out occurs  = 10us*100 = 1ms

    if(bkbhit)
    {
        char_out = rx_buffer[rx_buffer_next_out]; // get the byte

        if(++rx_buffer_next_out == BUFFER_SIZE) // increment counter
            rx_buffer_next_out = 0;

        return char_out;
    }
    else
    {
        timed_out = TRUE;
        return 0;
    }
}/* BufferedGetc */

 /*****************************************************************************
 $ Function: ReceivePacket
 $ Synopsis: Retrieves a packet from the receive buffer.
 $ Interface
    return: packet_type - indicates origin of packet, if its valid or if a timeout
            occurred while waiting for rest of packet
    params: packet_ptr - reference to buffer into which to place received data
    length_ptr - reference of calculated packet length
    global:

Structure of Packet Protocol:
```

```
+-------+-------------------------+---------------+--------+------+-----+-----+
: Start : Address To/Address From : Function Type : Length : Data : CRC : Stop:
+-------+-------------------------+---------------+--------+------+-----+-----+
Start:           1 byte Indicates start of packet
Address To/Address From:   1 byte UN - Address To/ LN - Address From
Function Type:       1 byte Used to identify type of data sent
Length:           1 byte Number of bytes in Data field
Data:           0 to N  Data being sent
CRC:           2 bytes 16 Bit CRC
Stop:           2 bytes <CR><LF> Indicates end of packet
 ****************************************************************************/

int ReceivePacket(byte *packet_ptr, int *length_ptr)
{
    int packet_type = 0;
    int i = 0;
    char char_rcved = 0x00;
    char previous_char_rcved = 0x00;
    char next_char_rcved = 0x00;
    int1 complete_packet = FALSE;

    char_rcved = BufferedGetc(); //get byte from buffer

    //test for a valid start char
    while ((char_rcved != START_BYTE) && (char_rcved != BT_START_BYTE1) &&
        (char_rcved != BT_CARRIER_LOSS))
    {
        char_rcved = BufferedGetc();

        if((char_rcved == LF) && (previous_char_rcved == CR))
        {
            packet_type = INVALID_PACKET; //No valid start of packet could be found
            return packet_type;
        }

        previous_char_rcved = char_rcved;
    }//end while(! a valid start char)

    //test if it’s a packet from the BT Module
    if((char_rcved == BT_START_BYTE1) || (char_rcved == BT_CARRIER_LOSS))
    {
        packet_type = BT_MODULE_PACKET; //Bluetooth packet - either AT command or carrier loss
    }

    if((char_rcved==START_BYTE))
    {
        packet_type = CONTROL_PACKET;
    }

    while(!complete_packet) //Search for end of packet or timeOut
    {
        packet_ptr[i++] = char_rcved;

        if((char_rcved == LF) && (previous_char_rcved == CR))
        {
            complete_packet = TRUE;
            *length_ptr = i;
            break;
        }

        next_char_rcved = BufferedGetc(); //look ahead byte initialise

        if(!timed_out)
        {
            if((previous_char_rcved == BT_CARRIER_LOSS) && (char_rcved == BT_CARRIER_LOSS)
                && (next_char_rcved == BT_CARRIER_LOSS))
            {
                //a valid but un-finished packet will be lost due to a comms break
                remote_comms_link = FALSE;
                //drop half-finished packet
                i=0;
                //2 chars should not be dropped - place back in buffer
                packet_ptr[i++] = BT_CARRIER_LOSS;
                packet_ptr[i++] = BT_CARRIER_LOSS;
                packet_type = BT_MODULE_PACKET; //type changes to BT packet type
            }

            previous_char_rcved = char_rcved;
            char_rcved = next_char_rcved;
        }
        else
        {
            packet_type = PACKET_TIMED_OUT;
            break;
        }
    }//end while(!completePacket)

    return packet_type;
}/* ReceivePacket */

 /***************************************************************************
 $ Function: ValidDataPacket
 $ Synopsis: Validates packet received by checking address, crc and payload length.
 $ Interface
    return: error code depending on validation check results
            NOTE: will only return first error condition encountered
    params: packet_ptr - reference to received packet
    packet_length - the received packet's length
    global:
 ***************************************************************************/
int ValidDataPacket(byte *packet_ptr, int packet_length)
{
    int data_length;
    int16 CRC;

    if(((packet_ptr[1] & 0xF0) >> 4) != ROBOT_ADDRESS) //packet meant for this robot?
        return WRONG_ADDRESS;

    data_length = packet_ptr[3];

    //check actual length vs length given in packet
    if(data_length != (packet_length - NUM_HEADER_BYTES - NUM_FOOTER_BYTES))
```

```
        return INCORRECT_LENGTH;

    CRC = (int16)(packet_ptr[data_length + NUM_HEADER_BYTES]) << 8; //calculate CRC
    CRC += packet_ptr[data_length + NUM_HEADER_BYTES + 1];

    //compare with CRC given in packet
    if(CRC != generate_16bit_crc(packet_ptr, packet_length - NUM_FOOTER_BYTES, CRC_CCITT))
        return CRC_ERROR;

    return CLEAN_PACKET; //passed all checks - packet is ok!
}/* valid_data_packet */

 /*****************************************************************************
 $ Function: TIMER0Isr
 $ Synopsis: TIMER 0 interrupt service routine. Provides period for steering motor.
             Times the A2D process. Provides debounce time for touch sensors
 ****************************************************************************/
#int_TIMER0
void TIMER0Isr()
{
    static int8 state = 0; //initialise state for analogue reading
    static int8 ad_channel = 0; //set to first analogue channel
    static int8 ms_count = 0; //keep track of when to update the servo
    static int8 min = 0; //keeps track of the smallest reading (possible low spike)
    static int8 max = 0; //keeps track of the highest reading (possible high spike)
    static int16 sum = 0; //running total of the samples taken (10 samples per channel)
    static int8 samples_taken = 0;
    byte a2d_value = 0;
    byte final_val = 0;

    set_timer0(63036); //create 1ms timebase
    ms_count++; //increment milli-second counter

    if (ms_count == 20) //only update servo every 20ms (20 * 1ms = 20ms)
    {
        setup_timer_3(T3_INTERNAL|T3_DIV_BY_1); //use timer-based pwm for servo control
        set_timer3(65536 - servo_val); //timer3 controls length of HIGH-time of servo pulse
        enable_interrupts(INT_TIMER3);
        output_high(SERVO);

        timer0_count++; //used as debounce time for the contact sensors
        ms_count=0; //re-init ms counter for next servo update
    }

    //used to get a2d values and apply filtering without waiting for a2d capture to finish
    switch(state)
    {
        case 0: //first time period (tick #1): select the a2d channel
            //available channels saved in a lookup table that also provide scanning order
            set_adc_channel(ad_channels_table[ad_channel]);
        state++;
        break;
        case 1: // tick #2: start the a2d conversion process
            read_adc(ADC_START_ONLY);
        state++;
            break;
        case 2: // tick #3: read the a2d channel and initialise the filtering variables
            a2d_value = read_ADC(ADC_READ_ONLY); //init
            sum = a2d_value; //initialise filtering variables with first reading
            min = a2d_value;
            max = a2d_value;
            samples_taken++; //keep track of samples accumulated
            state++;
            read_adc(ADC_START_ONLY); //start next conversion
            break;
        case 3:
            if(samples_taken<10) //filter and averaging over 10samples
            {
                a2d_value = read_ADC(ADC_READ_ONLY); //get next sample
                sum = sum + a2d_value; //keep running total of the samples
                samples_taken++;
                if(a2d_value < min) //is current value a new minimum reading?
                    min = a2d_value; //if it is, update the minimum tracker
                else if(a2d_value > max) //is the value a new maximum reading?
                    max = a2d_value; //if it is, update the maximum tracker

                read_adc(ADC_START_ONLY); //start next sampling but don't wait in isr
            }
            else //all samples for filtering has been taken
                state++; //go to next state on the following tick
    break;
        case 4:
            //remove high and low spikes and average remaining 8 samples
            final_val = (sum - max - min) >> 3;
            a2d_readings[ad_channel] = final_val; //store filtered value in buffer
            sum = 0; //re-initialise all filtering variables
            min = 0;
            max = 0;
            samples_taken = 0; //restart sampling
            state = 0;

            if (ad_channels_table[++ad_channel] == 255) //done all channels?
                ad_channel = 0; //restart the channel scan
            break;
        default:
            state = 0;
            ad_channel = 0;
            break;
    }
}/* TIMER0Isr */

 /****************************************************************************
 $ Function: TIMER1Isr
 $ Synopsis: TIMER 1 interrupt service routine. Keeps track of number of roll
             over events. TIMER 1 is also the timer associated with the CAPTURE
             mode used here to determine the robots speed and distance traveled.
 ***************************************************************************/
#int_TIMER1
void TIMER1Isr()
{
    ++timer1_rollover_count; //increment count when a timer rollover event occurs
    clear_interrupt(INT_TIMER1);
```

```
}/* TIMER1Isr */

/***************************************************************************
 $ Function: TIMER3Isr
 $ Synopsis: TIMER 3 interrupt service routine. Provides the 20ms period for
             the servo pulses together with TIMER 0 interrupt.
 **************************************************************************/
#int_TIMER3 HIGH
void TIMER3Isr()
{
    output_low(SERVO); //set output low
    setup_timer_3(T3_DISABLED);//disable timer 3 interrupt
    disable_interrupts(INT_TIMER3);
    clear_interrupt(INT_TIMER3);
}/* TIMER3Isr */

/***************************************************************************
 $ Function: ContactSensors
 $ Synopsis: Port B interrupt on change interrupt service routine. Used to
 determine the make and unmake of the contact sensors.
 $ Interface
    return:
    params:
    global: contact_information - the state of the touch sensors.
 **************************************************************************/
#int_rb
void ContactSensors(void)
{
    byte current = 0x00;
    static byte last = 0x00; //keep track of the last value of the contact sensors
    static int16 last_timer0_count;

    if(last_timer0_count == timer0_count)
        break; //debounce time has not elapsed
    else
    {
        current = input_b();

        if(current == last)
            break; //no change
        else
        {
            if(bit_test(current,BACK_LEFT))
                bit_set(contact_information,4);
            else
                bit_clear(contact_information,4);

            if(bit_test(current,BACK_RIGHT))
                bit_set(contact_information,5);
            else
                bit_clear(contact_information,5);

            if(bit_test(current,FRONT_LEFT))
                bit_set(contact_information,6);
            else
                bit_clear(contact_information,6);
            if(bit_test(current,FRONT_RIGHT))
                bit_set(contact_information,7);
            else
                bit_clear(contact_information,7);

            //check whether any of the touch sensors are made
            if(contact_information != 0x00)
            {
                //the robot is stuck against an object
                stuck = true;
                StopDriveMotor();
            }
            else
                stuck = false;

            //restart debounce time
            last_timer0_count = timer0_count;
            last=current;
        }//end else
    }//end else
}/* ContactSensors */

/*****************************************************************************
 $ Function: CCP2Isr
 $ Synopsis: Capture interrupt service routine. Occurs when encoder sensor goes
                high. Calculates period used to calculate the distance and speed.
 $ Interface
    return:
    params:
    global: distance_travelled - the total distance covered by robot
 ****************************************************************************/
#INT_CCP2
void CCP2Isr(void)
{
    int32 end_time;
    char timer1_local_copy;
    static int32 start_time = 0;

    end_time = (int32)CCP_2;
    timer1_local_copy = timer1_rollover_count;

    //do required calculations outside of interrupt
    capture_flag = TRUE;

    //check if timer1 interrupt pending and handle it here
    if(TMR1IF)
    {
        if(*BytePtr(end_time, 1) < 2)
        timer1_local_copy++;
        ++timer1_rollover_count;
        TMR1IF = 0; //clear interrupt
    }

    *BytePtr(end_time, 2) = timer1_local_copy; //24-bit value
    //ensure correct subtraction
    isr_CCP_delta = (end_time > start_time)? end_time - start_time : end_time +
        (0x1000000 - start_time);

    //end of one pulse is the start of the next;
    start_time = end_time;

    //keep track of distance moved by the robot
    distance_travelled = distance_travelled + DISTANCE_PER_TICK;

    clear_interrupt(INT_CCP2);
}/*  CCP2Isr */

/****************************************************************************
 $ Function: SetServo
 $ Synopsis: Calculates the high time of the pulse going to the servo motor
 given the angle and using a table look-up function. Also ensures safe operating
 angles only is used.
 $ Interface
    return:
    params: angle - angle to steer robot
    global: current_angle - the angle of the steering motor
    servo_val - value to control HIGH-time of servo control pulse train
 ****************************************************************************/
void SetServo(signed int angle)
{
    int16 temp_Servo_val;
    signed int temp_angle;

    if(angle < 0) //turn left
    {
        temp_angle = -angle;
        temp_servo_val = CENTRE + servo_table[temp_angle];
    }
    else //turn right
    {
        temp_servo_val = CENTRE - servo_table[angle];
    }

    //ensure save operating angle
    if ((temp_servo_val <= MAX_LEFT) && (temp_servo_val >= MAX_RIGHT))
    {
        servo_val = temp_servo_val;
        current_angle = angle; //update global angle variable
    }

    //if requested angle is greater that the robot can physically accommodate
    else if (angle < 0)
    {
        //turn maximum allowable in desired direction
        servo_val = MAX_RIGHT;
        current_angle = MAX_ANGLE_RIGHT;
    }
    else
    {
        //turn maximum allowable in desired direction
        servo_val = MAX_LEFT;
        current_angle = MAX_ANGLE_LEFT;
    }
}/* SetServo */

/****************************************************************************
 $ Function: DriveMotor
 $ Synopsis:  Controls the robot's backward and forward movement through the
 drive motor.
 $ Interface
    return:
    params: direction - whether to go forward or backwards
    pwm_duty - given as percentage of 100% duty cycle
    global: current_direction - whether the robot is busy going forward or backwards
 ****************************************************************************/
void DriveMotor(int1 direction, int pwm_duty)
{
    int duty;

    if (current_direction != direction)
    {
        //stop motor before changing direction
        StopDriveMotor();
        Delay_ms(STOP_TIME);
    }

    if(direction == FORWARD)
        GoForward();

    if(direction == BACKWARD)
        GoBackward();

    current_direction = direction;

    if(pwm_duty == 0)
        StopDriveMotor();
    else
    {
        duty = duty_table[pwm_duty]; //calculate duty
        set_pwm1_duty(duty); //set the PWM duty
    }
}/* DriveMotor */

/****************************************************************************
 $ Function: GoForward
 $ Synopsis: Sets the required enable pin combination on the H-bridge for forward
             movement
 ****************************************************************************/
void GoForward(void)
{
    //Output low on L2 on Compact Motor Driver
    output_low(MOTOR_BW_ENABLE);
    //Output high on L1 on Compact Motor Driver
    output_high(MOTOR_FW_ENABLE);
}/* GoForward */

```

```
/****************************************************************************
 $ Function: GoForward
 $ Synopsis: Sets the required enable pin combination on the H-bridge for
             backward(reverse) movement
 ***************************************************************************/
void GoBackward(void)
{
    //Output low on L1 on Compact Motor Driver
    output_low(MOTOR_FW_ENABLE);
    //Output high on L2 on Compact Motor Driver
    output_high(MOTOR_BW_ENABLE);
}/* GoBackward */

/****************************************************************************
 $ Function: GoForward
 $ Synopsis: Sets the required enable pin combination on the H-bridge to stop
                the drive motor
 ***************************************************************************/
void StopDriveMotor(void)
{
    //Output low on L1 on Compact Motor Driver
    output_low(MOTOR_FW_ENABLE);
    //Output low on L2 on Compact Motor Driver
    output_low(MOTOR_BW_ENABLE);
    set_pwm1_duty(0);
}/* StopDriveMotor */

/****************************************************************************
 $ Function: GoForward
 $ Synopsis: Sets the required enable pin combination on the H-bridge for pulsed
             brake mode
 ***************************************************************************/
void BrakeDriveMotor(void)
{
    //Output high on L1 on Compact Motor Driver
    output_high(MOTOR_FW_ENABLE);
    //Output high on L2 on Compact Motor Driver
    output_high(MOTOR_BW_ENABLE);
    set_pwm1_duty(TOP_SPEED);
}/* BrakeDriveMotor */

/****************************************************************************
 $ Function: RobotAct
 $ Synopsis: Control the robot's movements through the global variables.
 Describe robot reactions here.
 $ Interface
    return:
    params:
    global: stopped - whether robot is stationary or not
    angle - current angle of the steering motor
    direction - current direction of movement
    speed - current speed setting
 ***************************************************************************/
void RobotAct(void)
{
    SetServo(angle);
    DriveMotor(direction, speed);
    stopped = FALSE;
    //define how robo should react here...
}/* RobotAct */

/***************************************************************************
 * Main routine.
 ***************************************************************************/
void main(void)
{

    int16 hash_code;
    byte tx_packet[TX_BUFFER_SIZE],tx_data[TX_DATA_SIZE],rx_packet[RX_PACKET_SIZE];
    byte function_param = 0x00;
    int data_length = 0, length = 0, packet_length = 0,packet_type, packet_validity;

    Init();

    SetServo(CENTRE);

    while (TRUE)
    {
        if(OPERATING_MODE == AUTONOMOUS)
        {
            if((distance_travelled >= distance_to_travel) && !stopped)
            {
                StopDriveMotor();
                stopped = TRUE;
                distance_to_travel = 0;
                //add any additional autonomous behaviours here
            }
        }

        if(bkbhit) //there is data to be processed
        {
            //retrieve a message from the buffer and analyse it
            packet_type = ReceivePacket(rx_packet, &length);
            timed_out = FALSE; //reset packet time-out variable

            switch(packet_type) //determine action according to status of received packet
            {
                case CONTROL_PACKET: //a packet from the PDA for the robot
                //test packet integrity
                packet_validity = ValidDataPacket(rx_packet, length);

                switch(packet_validity) //determine action according to packet integrity
                {
                    case CLEAN_PACKET: //perfectly received packet
                    {
                        acknowledge = TRUE; //packet understood, therefore send acknowledge
                        function_param = rx_packet[2]; //data given or asked for in packet
//determine whether it is a "request for data" packet ie rxPacket[3] = 0
//or "command" packet rxPacket[3] != 0
                        if(rx_packet[3] == 0)
                        {
                            //gather requested data
                            CollectData(function_param, tx_data, &data_length);
                            //build tx packet
                            BuildPacket(SEND_ADDRESS, function_param, acknowledge, data_length,
                                tx_data, &packet_length,tx_packet);
                        }
                        else
                        {
                            //set requested variables with given data
                            SetData(function_param, rx_packet);
                            RobotAct(); //put changes into action
                            function_param = 0x00; //no data to send
                            data_length = 0; //length of an acknowledge packet is 0
                            //build tx packet
                            BuildPacket(SEND_ADDRESS, function_param, acknowledge, data_length,
                                tx_Data, &packet_length, tx_packet);
                        }
                        break;
                    }
                    //a number of error conditions can be reported back to PDA
                    case INCORRECT_LENGTH:
                    case CRC_ERROR:
                    case WRONG_ADDRESS:
                    {
                        //packet was not correctly received, therefore send not acknowledge
                        acknowledge = FALSE;
                        function_param = 0x00; //no data to send
                        tx_data[0] = packet_validity; //first error condition
                        data_length = 1; //error condition sent, so length is 1
                        //build tx packet
                        BuildPacket(SEND_ADDRESS, function_param, acknowledge, data_length,
                            tx_data, &packet_length, tx_packet);
                        break;
                    }
                    default:
                        //unknown error
                        break;
                    }//switch(valid data in packet)

    //if packet received from PDA indicates that it did not receive a packet correctly
                if((rx_packet[2] & 0x40) == 0)
                {
                    //keep track of sequential incorrectly sent packets
                    tx_unacknowledge_count++;
                    //packets not being received correctly
                    if(tx_unacknowledge_count > MAX_UNACKNOWLEDGED_PACKETS)
                    {
                        //reset the BT module
                        //reset_cpu();
                     }
                }
                else
                    //reset counter keeping track of sequential unacknowledged packets
                    tx_unacknowledge_count = 0;

    //only send packets when a communications link has been established with the PDA
                if(remote_comms_link == TRUE)
```

```
                    SendPacket(packet_length, tx_packet); //transmit tx packet

                break;

            case BT_MODULE_PACKET:
                //calculate hash code for the received BT packet
                hash_code = DJBHash(rx_packet, length);
                //determine which BT message was received and act accordingly
                InterpretHashCode(hash_code);
                break;

            case INVALID_PACKET:
            case PACKET_TIMED_OUT:
                acknowledge = FALSE;
                function_param = 0x00;
                tx_data[0] = packet_type;
                data_length = 1;
                //build tx packet
                BuildPacket(SEND_ADDRESS, function_param, acknowledge, data_length,
                    tx_data, &packet_length, tx_packet);

                if(remote_comms_link == TRUE)
                    SendPacket(packet_length, tx_packet); //transmit tx packet
                break;
            }//end switch(packetStatus)
        }//end if(bkbhit)
    }//end while(true)
}/* main */

/////////////////////// Driver to generate CRC //////////////////////////
////                                                                 ////
////  generate_8bit_crc(data, length, pattern)                       ////
////        Generates 8 bit crc from the data using the pattern.     ////
////                                                                 ////
////  generate_16bit_crc(data, length, pattern)                      ////
////        Generates 16 bit crc from the data using the pattern.    ////
////                                                                 ////
////  generate_32bit_crc(data, length, pattern)                      ////
////        Generates 32 bit crc from the data using the pattern.    ////
////                                                                 ////
/////////////////////////////////////////////////////////////////////////
////        (C) Copyright 1996,2003 Custom Computer Services         ////
//// This source code may only be used by licensed users of the CCS  ////
//// C compiler.  This source code may only be distributed to other  ////
//// licensed users of the CCS C compiler.  No other use,            ////
//// reproduction or distribution is permitted without written       ////
//// permission.  Derivative programs created using this software    ////
//// in object code form are not restricted in any way.              ////
/////////////////////////////////////////////////////////////////////////

#define CRC_16    0x8005      //bit pattern (1)1000 0000 0000 0101
#define CRC_CCITT 0x1021      //bit pattern (1)0001 0000 0010 0001
#define CRC_32    0x04C11DB7  //bit pattern (1)0000 0100 1100 0001 0001 1101 1011 0111


int generate_8bit_crc(char* data, int16 length, int pattern)
```

```
{
   int   *current_data;
   int   crc_byte;
   int16 byte_counter;
   int   bit_counter;

   current_data = data;
   crc_byte = *current_data++;

   for(byte_counter=0; byte_counter < (length-1); byte_counter++)
   {
      for(bit_counter=0; bit_counter < 8; bit_counter++)
      {
         if(!bit_test(crc_byte,7))
         {
            crc_byte <<= 1;
            bit_test(*current_data, 7 - bit_counter) ?
               bit_set(crc_byte,0) : bit_clear(crc_byte,0);
            continue;
         }
         crc_byte <<= 1;
         bit_test(*current_data, 7 - bit_counter) ?
            bit_set(crc_byte,0) : bit_clear(crc_byte,0);
         crc_byte ^= pattern;
      }
      current_data++;
   }
   for(bit_counter=0; bit_counter < 8; bit_counter++)
   {
      if(!bit_test(crc_byte,7))
      {
         crc_byte <<= 1;
         continue;
      }
      crc_byte <<= 1;
      crc_byte ^= pattern;
   }
   return crc_byte;
}


int16 generate_16bit_crc(char* data, int16 length, int16 pattern)
{
   int   *current_data;
   int16 crc_Dbyte;
   int16 byte_counter;
   int   bit_counter;

   current_data = data + 2;
   crc_Dbyte =  make16(data[0], data[1]);

   for(byte_counter=0; byte_counter < (length-2); byte_counter++)
   {
      for(bit_counter=0; bit_counter < 8; bit_counter++)
      {
         if(!bit_test(crc_Dbyte,15))
         {
            crc_Dbyte <<= 1;
            bit_test(*current_data, 7 - bit_counter) ?
               bit_set(crc_Dbyte,0) : bit_clear(crc_Dbyte,0);
            continue;
         }
         crc_Dbyte <<= 1;
         bit_test(*current_data, 7 - bit_counter) ?
            bit_set(crc_Dbyte,0) : bit_clear(crc_Dbyte,0);
         crc_Dbyte ^= pattern;
      }
      current_data++;
   }

   for(bit_counter=0; bit_counter < 16; bit_counter++)
   {
      if(!bit_test(crc_Dbyte,15))
      {
         crc_Dbyte <<= 1;
         continue;
      }
      crc_Dbyte <<= 1;
      crc_Dbyte ^= pattern;
   }

   return crc_Dbyte;
}

int32 generate_32bit_crc(char* data, int16 length, int32 pattern)
{
   int   *current_data;
   int32 crc_Dbyte;
   int16 byte_counter;
   int   bit_counter;

   current_data = data + 4;
   crc_Dbyte =  make32(data[0], data[1], data[2], data[3]);

   for(byte_counter=0; byte_counter < (length-4); byte_counter++)
   {
      for(bit_counter=0; bit_counter < 8; bit_counter++)
      {
         if(!bit_test(crc_Dbyte,31))
         {
         crc_Dbyte <<= 1;
         bit_test(*current_data, 7 - bit_counter) ?
            bit_set(crc_Dbyte,0) : bit_clear(crc_Dbyte,0);
            continue;
         }
         crc_Dbyte <<= 1;
         bit_test(*current_data, 7 - bit_counter) ?
            bit_set(crc_Dbyte,0) : bit_clear(crc_Dbyte,0);
         crc_Dbyte ^= pattern;
      }
      current_data++;
   }

   for(bit_counter=0; bit_counter < 32; bit_counter++)
   {
      if(!bit_test(crc_Dbyte,31))
      {
         crc_Dbyte <<= 1;
         continue;
      }
      crc_Dbyte <<= 1;
      crc_Dbyte ^= pattern;
   }

   return crc_Dbyte;
}
```

Appendix C

Sharp IR Sensor Approximation Function

To measure the non electric quantity distance, an IR-based distance measuring sensor is used. It transforms the distance measured into the electrical quantity, voltage. The relationship between the voltage measured and the measured distance is non-linear. This makes the conversion between voltage and distance more complicated. A function is needed that adequately represents the sensor's response to change in distance.

Figure C.1 shows the experimental setup used to obtain the data and the procedure used to determine the approximation function.

The approximation function is obtained and tested using a LabVIEW prototype, a DAQ card as well as a custom LabVIEW program (called vi). The following steps are preformed.

1. Voltage signals relating to the measured distance is returned from the Sharp IR distance sensor to the prototype board on an NI ELVIS workstation.

2. The prototype board is connected to a DAQ card.

3. This provides the sampled signals to the LabVIEW program.

4. The LabVIEW program gathers all the necessary statistical information from the samples taken at distance x including maximum, minimum, mean, standard deviation over the number of samples taken.

5. The samples are stored in CSV format to file, indicating distance vs voltage.

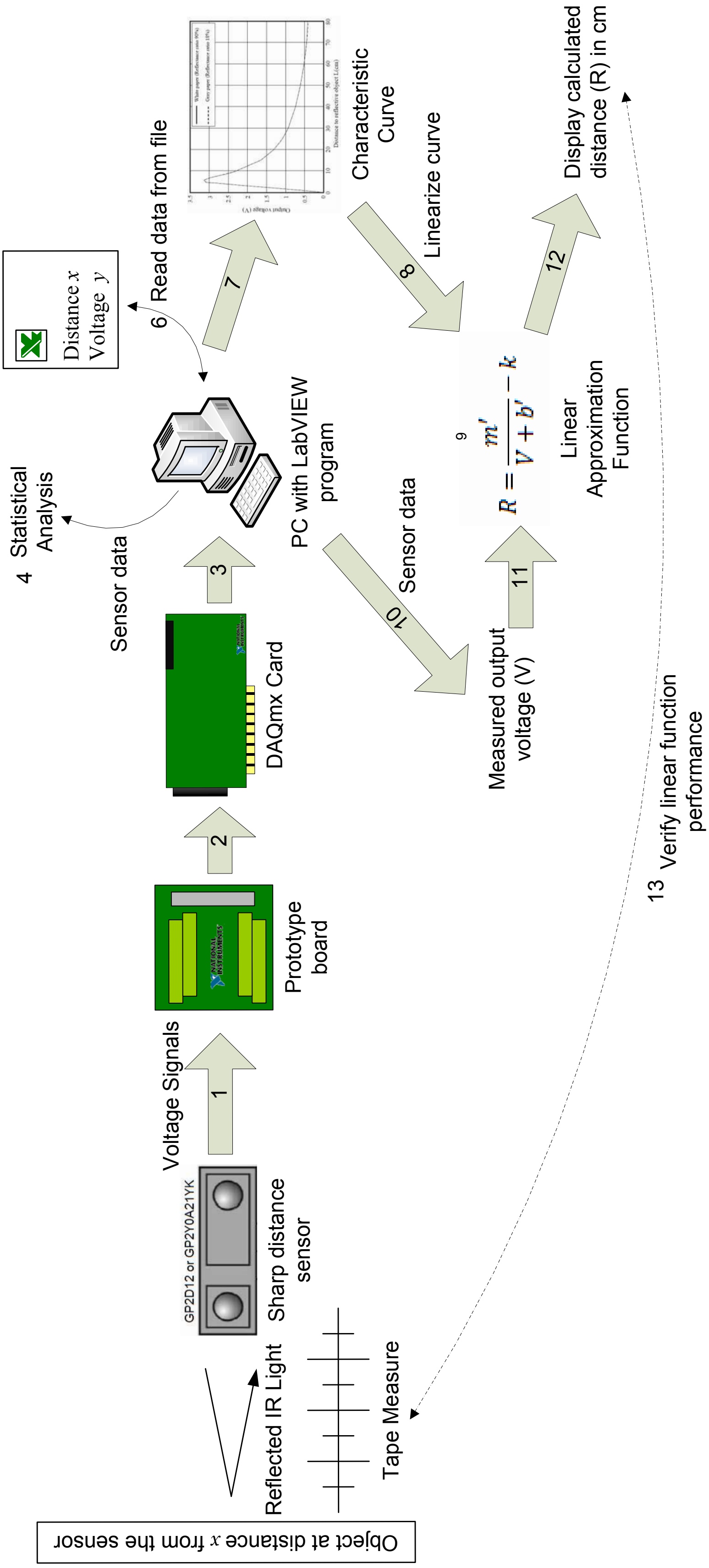


Figure C.1: Experimental Setup and Procedure to Obtain Approximation Function

6. Once all samples has been taken at every distance required, the program reads back the data written to file and plots the characteristic curve of the sensor (distance vs. voltage).

7. The curve is linearized.

8. The straight line approximation is calculated.

9. The approximation function is tested by reading voltage values and validating the measured distance against the calculated distance.

Figure C.2 shows the characteristic curve obtained using two Sharp IR sensors.

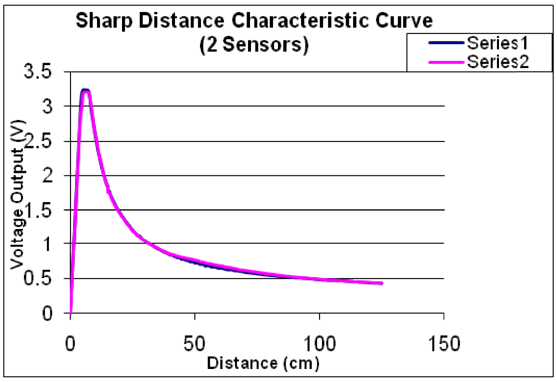


Figure C.2: Characteristic Curve of Two Sharp GP2D12 IR Sensors

From the data used in Figure C.2 the reciprocal of distance characteristic can be plotted for the sensor as shown in Figure C.3.

The next step is to find a straight line approximation that relates the read voltage to the inverse measured distance as shown in Figure C.3 (Acroname, 2006). This involves finding suitable m and b constants for the familiar straight line equation:

$$y = mx + b \tag{C.1}$$

In this case, y is equal to the linearised range. Substituting the linearising function from above for y and substituting V for x yields:

$$\frac{1}{R + k} = m \times V + b \tag{C.2}$$

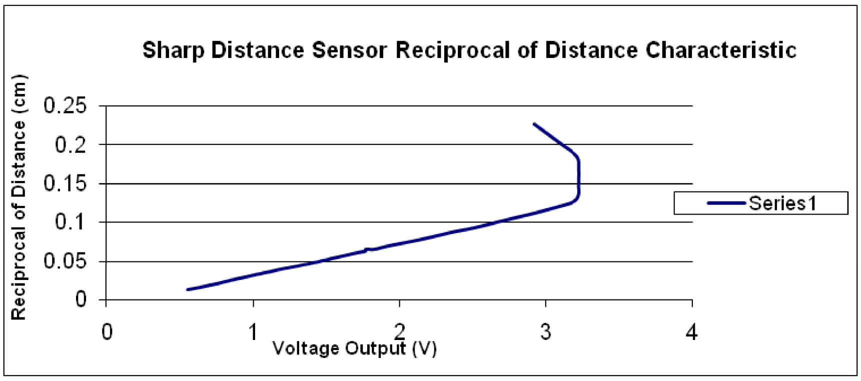


Figure C.3: Reciprocal of Distance Characteristic Curve for a Sharp GP2D12 IR Sensor

Rearranging the equation terms gives range as a function of voltage:

$$R = \frac{1}{m \times V + b} - k \tag{C.3}$$

This is a useful result for languages that support floating point math, but it can be rearranged further to get:

$$R = \frac{m'}{V + b'} - k \tag{C.4}$$

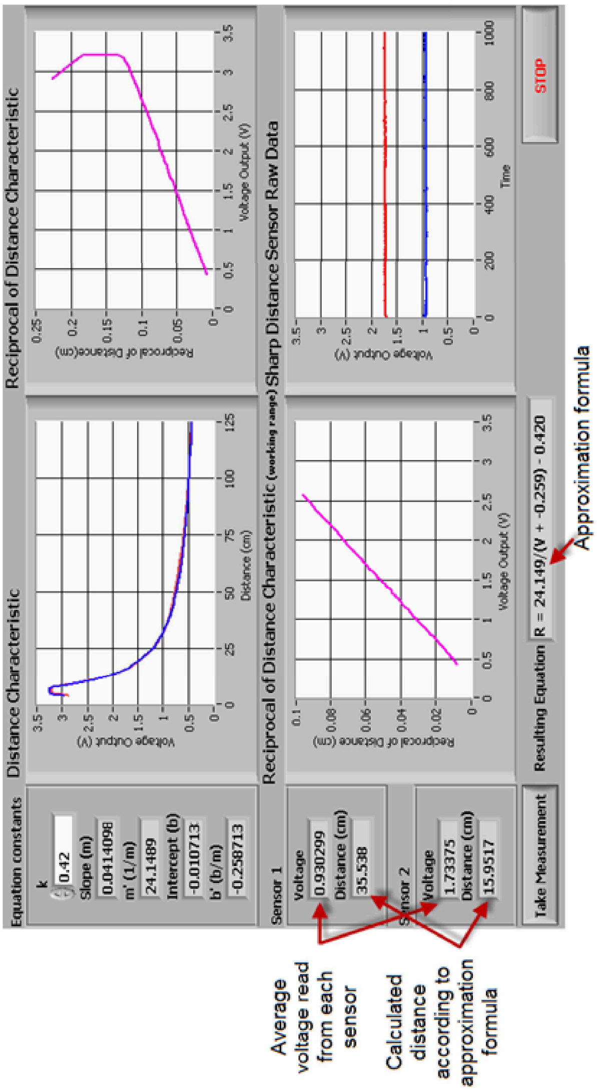
where $m' = \frac{1}{m}$ and $b' = \frac{b}{m}$. This extra step produces an equation that works nicely with integer math though the use of floating point values will increase accuracy.

The value of k was experimentally obtained. A value of 0.42 for this linearising constant gives a fairly straight line. The values for m and b is determined through applying linear regressing to the resulting straight line from C.3. The value for m is determined to be 0.041 and using integer maths results in a value for m' of 24.149 The value for b is determined to be -0.0107 and using integer maths and the equation given in C.4 the value for b' is determined to be -0.259.

Therefore the resulting approximation function is:

$$R = \frac{24.149}{V - 0.259} - 0.420 \tag{C.5}$$

Figure C.4 shows the LabVIEW application with the experimental results obtained.


Equation constants

k 0.42

Slope (m) 0.0414098

m' (1/m) 24.1489

Intercept (b) -0.010713

b' (b/m) -0.258713

Sensor 1

Voltage 0.930299

Distance (cm) 35.538

Sensor 2

Voltage 1.73375

Distance (cm) 15.9517

Resulting Equation R = 24.149/(V + -0.259) - 0.420

Figure C.4: IR Distance Sensor Approximation Results

References

32feet.NET. (2010). *32feet.NET - Personal Area Networking for .NET.* Available from: `http://32feet.codeplex.com/`. (Last cited January 2011)

Acroname. (2006). *Linearising Sharp Ranger Data.* Available from: `http://www.acroname.com/robotics/info/articles/irlinear/irlinear.html`. (Last cited December 2008)

Advanced System Technologies Ltd. (1999). *PDA Buyers Guide.* Available from: `http://www.comparestoreprices.co.uk/pdas-buyers-guide.asp`. (Last cited March 2008)

Apple Inc. (2008a). *Accelerometer: Made to Move.* Available from: `http://www.apple.com/iphone/features/accelerometer.html`. (Last cited September 2008)

Apple Inc. (2008b). *iPhone Applications.* Available from: `http://support.apple.com/kb/HT1872?viewlocale=en_US`. (Last cited September 2008)

Arkin, R., & Balch, T. (1997). AuRA: Principles and practice in review. *Journal of Experimental and Theoretical AI*, 175–189.

ARM. (2008). *THE CORTEX-A8 MICROPROCESSOR* (Tech. Rep.). ARM.

Baltes, J., McCann, S., & Anderson, J. (2006). Humanoid Robots: Abarenbou and DaoDan. *RoboCup - Humanoid League Team Description.*

Behnke, S., Langner, T., Müller, J., Neub, H., & Schreiber, M. (2004, September). NimbRo RS: A Low-Cost Autonomous Humanoid Robot for Multi-Agent Research. In *Workshop Proceedings: Methods and Technology for Empirical Evaluation of Multi-Agent Systems and Multi-robot Teams.* 27th German Conference on Artificial Intelligence, Ulm.

Behnke, S., Müller, J., & Schreiber, M. (2005a). Playing Soccer with RoboSapien. In *The 9th RoboCup International Symposium.* Osaka, Japan.

Behnke, S., Müller, J., & Schreiber, M. (2005b). Toni: A Soccer Playing Humanoid Robot. In *The 9th RoboCup International Symposium.*

Behnke, S., Müller, J., & Schreiber, M. (2005c). Using Handheld Computers to Control Humanoid Robots. In *1st International Conference on Dextrous Autonomous Robots and Humanoids.* Yverdon-les-Bains, Switzerland.

Boerner, G. (2010). *History of Hand-Held Computers... Part 3.* Available from: `http://www.boerner.net/jboerner/?p=10690`. (Last cited 22 January 2011)

Bonasso, R., Firby, R., Gat, E., Kortenkamp, D., Miller, D., & Slack, M. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 237–256.

Brooks, R. (1986, March). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *1*, 14–23.

Bugard, W., Cremers, A., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., et al. (1999, October). Experiences with an interactive museum tour-guide robot. *Experiences with an interactive museum tour-guide robot*, *114*(1), 3–55.

Buschmann, C., Müller, F., & Fischer, S. (2003). *Grid-Based Navigation for Autonomous, Mobile Robots* (Tech. Rep.). Instritute of Operating Systems and Networks, Technical University of Braunschweig, Germany.

Calinon, S., & Billard, A. (2003). *PDA Interface for Humanoid Robots* (Tech. Rep.). Lausanne, Switzerland: Swiss Institute of Technology.

Carmack, C., & Freudenrich, C. (2008a). *How PDAs Work.* Available from: `http://communication.howstuffworks.com/pda.htm`. (Last cited July 2008)

Carmack, C., & Freudenrich, C. (2008b). *The PDA Computer.* Available from: `http://communication.howstuffworks.com/pda4.htm`. (Last cited September 2008)

CNET Networks Inc. (2008a). *Handheld Operating Systems: BalckBerry.* Available from: `http://reviews.cnet.com/4520-11309_7-6624304-4.html?tag=rb_content;rb_mtx`. (Last cited September 2008)

CNET Networks Inc. (2008b). *Handheld Operating Systems: Palm.* Available from: `http://reviews.cnet.com/4520-11309_7-6624304-2.html?tag=rb_content;rb_mtx`. (Last cited September 2008)

CNET Networks Inc. (2008c). *Handheld Operating Systems: Symbian.* Available from: `http://reviews.cnet.com/4520-11309_7-6624304-5.html?tag=rb_content;rb_mtx`. (Last cited September 2008)

CNET Networks Inc. (2008d). *Handheld Operating Systems: Windows Mobile.* Available from: `http://reviews.cnet.com/4520-11309_7-6624304-3.html?tag=rb_content;rb_mtx`. (Last cited September 2008)

Conger, D. (2003). *Processors, Memory, Expansions, and Wireless Explained.* Available from: `http://www.davespda/features/index.htm`. (Last cited July 2008)

Conger, D. (2005). *Personal Digital Assistant.* Available from: `http://www.davespda/features/index.htm`. (Last cited July 2008)

Coste-Maniere, E., & Simmons, R. (2000). Architecture, the Backbone of Robotic Systems. In *Ieee international conference on robotics & automation.* IEEE.

Cross, B. (2007). *WiMo the Windows Mobile Robot.* Available from: `http://www.wimobot.com/`. (Last cited July 2008)

d'Angelo, P., & Corke, P. (2001). A Robot Interface Using WAP Phone. In *Australian Conference on Robotics and Automation* (pp. 141–145).

De Ipiña, D., Vázquez, I., De Garibay, J. R., & Sainz, D. (2005). GPRS-based Real-Time Remote Control of MicroBots with M2M Capabilities. In *Wireless Information Systems .*

El-Rabbany, A. (2002). *Introduction to GPS: the Global Positioning System.* Artech House Inc.

Firmansyah, I., Hermanto, B., & Handoko, L. (2007). *Control and Monitoring System for Modular Wireless Robot* (Tech. Rep.). Indonesian Institute of Sciences.

Fong, T., Cabrol, N., Thorpe, C., & Baur, C. (2001). A Personal User Interface for Collaborative Human-Robot Exploration. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space.* Montreal, Canada.

Fong, T., Conti, F., Grange, S., & Baur, C. (2001). Novel Interfaces for remote driving: gesture, haptic and PDA. *Autonomous Robots*.

Fong, T., Thorpe, C., & Glass, B. (2003). PdaDriver: A Handheld System for Remote Driving. In *IEEE International Conference on Advanced Robotics.*

Franklin, D. (2006). *Windows CE.* Available from: `http://searchwinit.techtarget.com/sDefinition/0,,sid1_gci213372,00.html`. (Last cited September 2008)

Gade, L. (2004). *PDA Keyboard Reviews.* Available from: `http://www.mobiletechreview.com/tips/stowaway_Bluetooth_Keyboard.htm`. (Last cited August 2008)

GADGETS. (2008). *Intel reveals Intel Atom low power processors for mobile devices.* Available from: `http://gadgets.qj.net/Intel-reveals-Intel-Atom-low-power-processors-for-mobile-devices/`. (Last cited March 2008)

Gat, E. (1992). Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Tenth National Conference on Artificial Intelligence* (pp. 809–815).

H2G2. (2004). *History of the Personal Data Assistant (PDA).* Available from: `http://www.bbc.co.uk/dna/h2g2/A2284229`. (Last cited July 2008)

Hall, R. (2007). *New Windows Mobile 6 Devices.* Available from: `http://www.pocketpcmag.com/cms/_archives/jun07`. (Last cited September 2008)

Heinrich, D., & Honiger, T. (1997). Parallel Processing Approaches In Robotics. In *IEEE International Symposium on Industrial Electronics.*

Holland, O. (2011). *Background information (The Grey Walter Online Archive).* Available from: `http://www.ias.uwe.ac.uk/Robots/gwonline/gwonline.html`. (Last cited January 2011)

Howell, A. L., & Sersen, D. W. (2004). Using a Web Service, Mobile Device and Low-Cost Robot to Teach Computer Science and Engineering. In *EIT Conference.*

HPC. (2001). *The History of Windows CE.* Available from: `http://www.hpcfactor.com/support/windowsce/`. (Last cited April 2008)

Hu, H., & Brandy, M. (1996). A parallel processing architecture for sensor-based control of intelligent mobile robots. In *Robotics and Autonomous Systems (17)* (pp. 235–257).

Hüttenrauch, H., & Eklundh, K. S. (2003). *Fetch-and-carry with CERO: Observations from a long-term user study with a service robot* (Tech. Rep.). Interaction and Presentation Laboratory, Royal Institute of Technology, Stockholm, Sweden.

Hüttenrauch, H., & Norman, M. (2001). PocketCERO mobile interfaces for service robots. In *Mobile HCI, International Workshop on Human Computer Interaction with Mobile Devices.*

Jantz, S., & Doty, K. L. (2002). PDA Based Real-Time Vision for a Small Autonomous Mobile Robot. In *Florida Conference on Recent Advances in Robotics.*

Jensen, L. K., Kristensen, B. B., & Demazeau, Y. (2005, September). FLIP: Prototyping multi-robot systems. *Robotics and Autonomous Systems*(53), 230–243.

Joshi, A. (2003). *A PDA Enabled Wireless Interface for a Mobile Robot.* Unpublished master's thesis, Case Western Reserve University.

Keskinpala, H., Adams, J., & Kawamura, K. (2003). PDA-Based Human Robotic Interface. In *IEEE International Conference on Systems, Man and Cybernetics* (pp. 3931–3936).

Konolige, K., Myers, K., Ruspini, E., & Saffiotti, A. (1997). The Saphira Architecture: A Design for Autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, *9*, 215–235.

Kovan Research Lab. (2006). *Calligrapher Robot.* Available from: `http://www.kovan.ceng.metu.edu.tr`. (Last cited February 2007)

Laberge, R., & Vujosevic. (2003). *Building PDA Databases for Wireless and Mobile Development* (C. Long, Ed.). Wiley Publishing, Inc.

Laengle, T., & Lueth, T. (1994). Decentralised control of distributed robots and subsystems. *Artificial Intelligence in Real Time Control*, 281–286.

Langeveld, R., Narang, A., & Buenaventura, C. (2005). *PDA Enabled Wireless Interfaced Robot* (Senior Design Project). University of California, Riverside.

Lemoine, P., Gutiérrez, M., Thalmann, D., & Vexo, F. (2004). The "Caddie Paradigm": a Free-Locomotion Interface for Teleoperation. In *CAPTECH 2004, Workshop on Modeling and Motion Capture Techniques for Virtual Environments.*

Mataric, M. J. (1997). Behaviour-Based Control: Examples from Navigation, Learning, and Group Behaviour. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents*, *9*(2–3), 323–336.

Microchip. (2007). *PIC18F2525/2620/4525/4620 Data Sheet* (Tech. Rep.). Microchip Technology Inc.

Microsoft. (2008). *The Ultra-Mobile PCAnywhere you go.* Available from: `http://www.microsoft.com/windows/products/winfamily/umpc/default.mspx`. (Last cited January 2011)

Miller, J. (2004). *Robotic Systems for Inspection and Surveillance of Civil Structures.* Unpublished master's thesis, Mechanical Engineering, University of Vermont.

Mobile Tech Review. (2008). *What is a PDA?* Available from: `http://www.mobiletechreview.com/genfaq.shtml`. (Last cited July 2008)

MobileTechReview. (2002). *PDA Keyboard Reviews: Snap N Type.* Available from: `http://www.mobiletechreview.com/tips/snapntype.htm`. (Last cited August 2008)

MSDN Library. (2010). *BatteryLevel Enumeration.* Available from: `http://msdn.microsoft.com/en-us/library/microsoft.windowsmobile.status.batterylevel.aspx`. (Last cited January 2011)

MSDN Library. (2011a). *GetSystemPowerStatusEx2.* Available from: `http://msdn.microsoft.com/en-us/library/aa453173.aspx`. (Last cited January 2011)

MSDN Library. (2011b). *SYSTEM_POWER_STATUS_EX2.* Available from: `http://msdn.microsoft.com/en-us/library/ms940385.aspx`. (Last cited January 2011)

Mukhar, K., & Johnson, D. (2003). *The Ultimate Palm Robot.* McGraw-Hill.

Murphy, R. R. (2000). *An introduction to ai robotics.* MIT Press.

Murtazin, E. (2003). *Review Sony Ericsson Bluetooth Car.* Available from: `http://www.mobile-review.com/review/sonyericcson-btcar-en.shtml`. (Last cited July 2008)

NCCW. (2008). *History of the PDA.* Available from: `http://www.nccw.net/`. (Last cited September 2008)

Nicolo, S. (2008). *Intel reveals Intel Atom low power processors for mobile devices.* Available from: `http://gadgets.qj.net/Intel-reveals-Intel-Atom-low-power-processors-for-mobile-devices/pg/49/aid/115137`. (Last cited March 2008)

Nilas, P. (2005). A PDA-based Human-Robot Interaction using EMG-based Modified Morse Code. In *International Conference on Computational Intelligence for Modeling, Control and Automation.*

Nokia. (2008). *Symbian OS.* Available from: `http://europe.nokia.com/A4153292`. (Last cited September 2008)

Oates, T. (2005). Book Review: R. Siegwart and I. Nourbakhsh, Introduction to Autonomous Mobile Robots, MIT Press (2004). *Artificial Intelligence*, *169*(2), 146–149.

Page, M. (2005). *First, what is a subnotebook?* Available from: `www.transmetazone.com`. (Last cited September 2008)

Palm Inc. (2007a). *Historical Timeline.* Available from: `http://www.palm.com/us/company/corporate/timeline.html`. (Last cited March 2008)

Palm Inc. (2007b). *Historical Timeline.* Available from: `http://www.palm.com/us/company/corporate/timeline.html`. (Last cited April 2008)

Patra, A., & Ray, S. (2007). Guiding Robots Using Mobile Phones. In *International Symposium on Automation and Robotics in Construction* (pp. 339–344).

PC MAG.COM Encyclopedia. (2008). *Definition of PDA.* Available from: `http://www.pcmag.com/encyclopedia_term`. (Last cited July 2008)

PDA Phone Blog. (2010). *PDA & Handheld Computers Buying Guide.* Available from: `http://pdaclone.com/pda-handheld-computers-buying-guide.html`. (Last cited January 2011)

Perzanowski, D., Schultz, A. C., Adams, W., Marsh, E., & Bugajska, M. (2001, January). Building a Multimodal Human-Robot Interface. *IEEE Intelligent Systems*.

phoneArena.com. (2008). *Windows Mobile ships fewer smartphones than BlackBerry OS in Q2 2008.* Available from: `http://www.phonearena.com/htmls/Windows-Mobile-ships-fewer-smartphones-than-BlackBerry-OS-in-Q2-2008-article-a_3156.html`. (Last cited September 2008)

Posadas, J., Perez, P., Simo, J., Benet, G., & Blanes, F. (2002). Communications structure for sensory data in mobile robots. *Engineering Applications of Artificial Intelligence*, *15*, 341–350.

Posadas, J., Poza, J., Simo, J., Benet, G., & Blanes, F. (2007, July). Agent-based distributed architecture for mobile robot control. *Engineering Applications of Artificial Intelligence*.

Psion. (2008). *Psion Organiser.* Available from: `http://members.surfeu.at/org2/psion1/`. (Last cited April 2008)

Ranky, G. (2007). Mobile robot sensor and networking design to enable cooperative behaviour. *Industrial Robot*, 21–25.

Rosenblatt, J. (1997). *DAMN: A Distributed Architecture for Mobile Navigation* (Tech. Rep.). Robotics Institute, Carnegie Mellon University.

Rybski, P. E., Burt, I., Dahlin, T., Gini, M., Hougan, D. F., Krantz, D. G., et al. (2001). System Architecture for Versatile Autonomous and Teleoperated Control of Multiple Miniature Robots. In *IEEE International Conference on Robotics.*

Santana, P. F. (2005). *Survival Kit: A Bottom Layer for Robot Navigation.* Unpublished master's thesis, New University of Lisbon.

Schrödinger & co. (1996). *The Filofax: A Brief History of Personal Time Management.* Available from: `http://www.univie.ac.at/Schrodinger/3_96/time24.htm`. (Last cited July 2008)

SearchMobileComputing.com. (2007). *Personal Digital Assistant.* Available from: `http://searchmobilecomputing.techtarget.com/sDefinition`. (Last cited July 2008)

Selinger, S., & Jakl, A. (2007). *ShakerRacer.* Available from: `http://www.symbianresources.com/projects/shakerracer.php`. (Last cited July 2008)

Selvatici, A. H. P., & Costa, A. H. R. (2005). A Hybrid Adaptive Architecture for Mobile Robots Based on Reactive Behaviours. In *Proceedings of the Fifth International Conference on Hybrid Intelligent Systems* (pp. 29–34).

Shah, A. (2008). *Microsoft to limit capabilities of cheap laptops.* Available from: `http://www.pcworld.com/businesscenter/article/145719/microsoft_to_limit_capabilities_of_cheap_laptops.html`. (Last cited January 2011)

Simmons, R. (1994). Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 34–43.

Skubic, M., Bailey, C., & Chronis, G. (2003). A Sketch Interface for Mobile Robots. In *IEEE International Conference on Systems, Man and Cybernetics.*

Skubic, M., Blisard, S., Carle, A., & Matsakis, P. (2002). Hand-Drawn Maps for Robot Navigation. In *AAAI Spring Symposium.*

Smith, T. (2004). *Sharp to ship world's first HDD-based PDA.* Available from: `http://www.theregister.co.uk/2004/10/15/linux_4gb_hdd_pda/`. (Last cited August 2008)

Smith, T. (2008). *Sharp unwraps 'world first' Intel Atom phone.* Available from: `http://www.reghardware.co.uk/2008/04/14/sharp_willcom_atom_first`. (Last cited September 2008)

Sofge, D., Trafton, G., Cassimatis, N., Perzanowski, D., Gugajska, M., Adams, W., et al. (2004). Human-Robot Collaboration and Cognition with an Autonomous Mobile Robot. In *Intelligent Autonomous Systems.*

Solarbotics. (2008). *L298 Compact Motor Driver Kit.* Available from: `http://www.solarbotics.com/products/k_cmd/`. (Last cited July 2009)

Stasse, O., & Kuniyoshi, Y. (2000). PredN: Achieving efficiency and code re-usability in a programming system for complex robotic applications. In *International Conference on Robotics and Automation* (pp. 81–87).

Symbian. (2008). *Symbian Fast Facts Q2 2008.* Available from: `http://www.symbian.com/about/fastfacts.html`. (Last cited September 2008)

T-Mobile. (2008). *Introducing T-Mobile G1 with Google.* Available from: `http://t-mobileg1.com/g1-learn-video-press-conference.aspx`. (Last cited September 2008)

The Robot Shop. (2008). *Spykee Cell.* Available from: `http://www.therobotshop/catalog/spykee-cell-mobile-phone-controller-robot`. (Last cited July 2008)

The Robotics Institute at Carnegie Mellon University. (2001). *PPRK: Palm Pilot Robotic Kit.* Available from: `http://www.cs.cmu.edu/~reshko/PILOT/`. (Last cited July 2008)

Tilley, C. (2001). *The History of Windows CE.* Available from: `http://www.hpcfactor.com/support/windowsce/`. (Last cited March 2008)

Wigley, A., Moth, D., & Foot, P. (2007). *Microsoft Mobile Development Handbook.* Micrcosoft Press.

Wikipedia. (2008). *Android (mobile device platform).* Available from: `http://en.wikipedia.org/wiki/Android_(mobile_device_platform)`. (Last cited September 2008)

Wikipedia. (2008). *Palm OS.* Available from: `http://en.wikipedia.org/wiki/Palm_OS`. (Last cited January 2011)

Wikipedia. (2008a). *Palm-Size PC.* Available from: `http://en.wikipedia.org/wiki/Palm-size_PC`. (Last cited September 2008)

Wikipedia. (2008b). *T1-Mobile G1.* Available from: http://en.wikipedia.org/wiki/T-Mobile_G1. (Last cited September 2008)

Williams, D. H. (2003). *PDA Robotics: Using Your Personal Digital Assistant to Control Your Robot.* McGraw-Hill.

Wolf, K. (2007). *NiVek J.D.'s Mainden Voyage.* Available from: http://www.efficientcoder.com/CategoryView,category,Robotics.aspx. (Last cited July 2008)

Yang, B., Zheng, P., & Ni, L. M. (2007). *Professional Microsoft Smartphone Programming.* Wiley Publishing.

Yasuda, G. (2003). Distributed autonomous control of modular robot systems using parallel programming. *Journal of Materials Processing Technology*, 357–364.

Zeldes, N. (2005). *The first PDA.* Available from: www.nzeldes.com/HOC/Newton.htm. (Last cited September 2008)

Zhang, T. (2002). *flexis FX100 Keyboard Review.* Available from: http://www.mobiletechreview.com/tips/flexis.htm. (Last cited August 2008)

Zhang, T. (2004). *Pocketop Wireless Link IR Keyboard.* Available from: http://www.mobiletechreview.com/tips/pocketop.htm. (Last cited August 2008)