Operator Controlled Obstacle Avoiding Telecontrolled Robotic Platform

By

VENKATESWARA REDDY

A Thesis submitted to

the Faculty of Graduate Studies in partial fulfilment of

the requirements for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION

Operator Controlled Obstacle Avoiding Telecontrolled Robotic Platform

BY

VENKATESWARA REDDY

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of

Manitoba in partial fulfillment of the requirement of the degree

MASTER OF SCIENCE

Venkateswara Reddy © 2007

Abstract

The work addressed here presents a number of challenges to Internet based telerobotics. One of the main challenges associated with operator assisted telerobotics is that of network delay. Non-deterministic delay can cause a number of problems, not the least being difficulty in remotely controlling the robot in a safe manner in near real time. As such, one of the goals of this project was the design of an obstacle avoidance subsystem within a telerobotic platform. Tertiary goals where to design the platform in manners that are cost effective, reliable, extendable using modern design methodologies and that would result in a platform that is close to an industrial realization. In this thesis, to solve the problems due to network delay a complete platform was developed that is ideally suited for implementing algorithms of local intelligence or reasoning. Specifically, the platform was designed to have some degree of local intelligence (heuristics) to mitigate problems associated with delay by avoiding obstacles in its path of motion using a number of sonar sensors and a rule based inference system. In addition, real time video feedback was provisioned for the platform along with a subsystem for GPS data collection for robot mapping in the future. The platform has minimal hardware and software overhead and combines hardware and software modules in an effective manner as possible which to some extent reduces the amount of time required for processing requests. Several tests were conducted to test the platform which worked as expected in a controlled environment.

Acknowledgements

I would like to thank Dr Robert McLeod who inspired me to take up this research topic and provided me all the technical help required throughout my masters program. I would like to thank him for offering me a few thesis related special courses. I would like to thank CMC and Internet Innovation Center for providing all the research equipment. I would also like to thank Mr. Dennis Stanley who has provided a number of the base mechanical components for this project as well as for others in our lab.

I would like to thank the people in the ECE Tech shop Gordon Tool, Ken Biegun, Mount-First Ng, Guy Jonatschick and Allan McKay for providing us with all the required apparatus and software.

I would like to thank Monir Islam Khan and David Sanders for helping me out in resolving issues in the process of development

I am very thankful to my parents Ravindranadha Reddy and Krishna Kumari, my sisters Anitha Reddy and Swathy Reddy , my brother in laws Chandrashekar Reddy and Harshavardhan Reddy and my friend Ranjana Hegde for their moral support and for encouraging me in pursuing my dreams

Table Of Content

Abstract....................i
Acknowledgements....................ii
Table of Content....................iii
List of Tables....................vi
List of Figures....................vii
Acronyms....................viii

CHAPTER 1: INTRODUCTION

1.1 Motivation....................2
1.2 Objective....................3
1.3 Problem Statement....................6
1.4 Organization of Thesis....................6

CHAPTER 2: BACKGROUND

2.1 Internet Based Robotic Operation....................8
2.2 History of Telerobotics....................9
2.2.1. On the Network Side....................11
2.2.2. At the Robot Side....................11
2.2.3. At the Operator Side....................12
2.3 Packet Loss....................12
2.4 Latency....................13
2.5 Degree of Autonomy....................14
2.6 Local Intelligence at Robot Side....................15
2.6.1. Event Based Telerobotics....................15
2.6.2. Fuzzy Controlled Robots....................16
2.6.3. Behavior Based Telerobotics....................18
2.6.4. Behavior Based Fuzzy Control Telerobotics....................19
2.7 Suitable Hardware....................20
2.7.1. Single Board Computer (SBC)....................21
2.7.2. Field Programming Gate Array (FPGA) boards....................21
2.8 Embedded Systems and Real Time Operating System....................22
2.9. Embedded Processors....................24
2.9.1 Nios II Processor....................25
2.10 Micro C/OS II (μC/OS-II) RTOS....................26
2.11 Appropriate Programming language for developing Graphical User Interface....................28
2.12 Summary of Chapter 2:....................28

CHAPTER 3: HARDWARE IMPLEMENTATION

3.1 Pulse Width Modulation....................30

3.2 Generation of PWM Using Altera DE2 Board 32
3.3 Motor Controller 34
3.4 Robot Direction Control 36
3.5 Configuring Ethernet on DE2 Board 39
3.6 Ethernet to Wireless Using Soekris Board 40
3.7 CMOS Camera 42
3.8 Summary of Chapter 3 43

CHAPTER 5: SOFTWARE IMPLEMENTATION

4.1 Graphical User Interface 42
4.2 Joystick Interface 46
4.3 Client Sockets 48
4.4 Server Sockets 50
4.5 Command Generation 51
4.5.1 CASE1: To Generate command for Left and Right Turn 51
4.5.2 CASE2: Neutral 53
4.5.3 CASE3: To Generate Commands for Forward and Reverse Motion 54
4.5.4 CASE N 54
4.5.4.1: 360 Degrees Rotation 54
4.5.4.2: Stop Command 55
4.6 Command Processing at Server Side 55
4.6.1 Case 'F' 58
4.7 Summary of Chapter 4 58

CHAPTER 5: ADVANCE IMPLEMENTATION

5.1 Video 59
5.1.1. Frame Grabber 61
5.1.2. Video Streaming Server 61
5.1.3. Video Capture and Display at the Controller End 62
5.2. Sonar Sensor 64
5.2.1 Distance from Obstacle 65
5.3. Global Positioning System (GPS) 66
5.4 Implementation of Obstacle Avoidance 67
5.4.1. Case D1: Obstacle Detected by Right Sensor on the Front of the Robot 69
5.4.2. Case D2: Obstacle Detected by Left Sensor on the Front of the Robot 70
5.4.3. Case D3: Obstacle Detected by the rear Sonar Sensor 72
5.4.4 Case D1 and D2: Obstacle Detected by Both the Front Sensors 72
5.5 Summer of Chapter 5 73

CHAPTER 6: RESULTS AND DISCUSSIONS

6.1 GUI 74

6.2 Basic Functionalities of Telerobotics Platform....................................75
6.3 Video Feedback...75
6.3.1 Video Feedback Using CMOS Camera.......................................76
6.4 Sonar sensor...76
6.5 Obstacle Avoidance Algorithm..77
6.6 Cost Efficient Design..77
6.7 Summary of Chapter 6...77

CHAPTER 7: CONCLUSIONS and FUTURE WORK

7.1 Conclusions..79
7.2 Recommended Future Work...80

Appendix A Task Logic...82
Appendix B Screenshots and Pictures...83
References..85

List of Tables

Table 1: Basic Functionality results for Telerobotics Platform.....................75

List of Figures

Figure 1: Telerobotics Platform Architecture 5
Figure 2: Basic Internet Based Telerobotics 8
Figure 3: The Event Based Control through Internet 15
Figure 4: Simple Fuzzy Logic Control System 17
Figure 5: Finite state Acceptor (FSA) 18
Figure 6: Behavior Based Fuzz Control Robot 20
Figure 7: RTOS Abstraction Layer between Application Software and Embedded Hardware [WWW3] 22
Figure 8: Task States 23
Figure 9: Nios II Soft Core Processor 25
Figure 10: Micro C/OS II Programming Architecture 27
Figure 11: Pulse Train 31
Figure 12: PWM Signals of Varying Duty Cycle 31
Figure 13: System Configuration of the DE2 board to Produce PWM Signals 33
Figure 14: Two Basic States of H-Bridge 34
Figure 15: Motor Driver 35
Figure 16: Differential Steering System 37
Figure 17: Wheels at Different Velocities 37
Figure 18: Communication Interface 40
Figure 19: Ethernet to Wireless 42
Figure 20: Graphical User Interface 45
Figure 21: Joystick Interface Flow Chart 47
Figure 22: Joystick Mapping 48
Figure 23: Command Generation Flow Chart 52
Figure 24: Neutral Region for Joystick Mapping 53
Figure 25: Command Stream in a Buffer 56
Figure 26: Command Regeneration at Server Side 57
Figure 27: Live Video Streaming 60
Figure 28: RTP/UDP Header 62
Figure 29: Sonar Sensor Timing Diagram 64
Figure 30: Implementation of GPS Receiver 67
Figure 31: Implementation of Obstacle Avoidance Control Logic in the Nios II 68
Figure 32: Obstacle Avoidance Algorithm 71

Acronyms

ADC	Analog to Digital Converter
API	Application Programming Interface
CGI	Common Gateway Interface
CMOS	Complementary Metal 0xide Semiconductor
COG	Center of Gravity
CPU	Central Processing Unit
CURV	Cable Controlled Underwater Research Vehicle
DHCP	Dynamic Host Configuration Protocol
FPGA	Field Programmable Gate Array
FSA	Finite State Acceptor
FTP	File Transfer Protocol
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
HTML	Hypertext Markup Language
I/O	Input/Output
ID	Identification
IDE	Integrated Development Environment
IP	Internet Protocol
ISA	Instruction Set Architecture
LAN	Local Area Network
LCD	Liquid Crystal Display
LWIP	Light Weight Internet Protocol
MAC	Media Access Control
MJPEG	Motion Joint Photographic Expert Group
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
NASA	National Aeronautics and Space Administration
OS	Operating System
PC	Personal Computer
PCI	Peripheral Component Interconnect
PHY	Physical Layer
PWM	Pulse Width Modulation
RAM	Random Access Memory
RFC	Request for Comments
ROM	Read Only Memory
RTCP	Real-Time Transmission Control Protocol

RTOS	Real Time Operating System
RTP	Real -Time Transport Protocol
SBC	Single Board Computer
SDRAM	Synchronous Dynamic Random Access Memory
SOPC	System on Programmable Chip
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TELNET	Telecommunication Network
UDP	User Datagram Protocol
USB	Universal Serial Bus
VGA	Video Graphic Array
VLC	Video LAN Client
Wi-Fi	Wireless Fidelity
WWW	World Wide Web

CHAPTER 1

INTRODUCTION

The Internet has revolutionized the way in which we receive information and interact with the world. Internet tools such as WWW, FTP, TELNET, email, etc. have provided the most convenient manner to transfer information to and from remote places. The bi-directional structure of Internet also provides a means to perform remote action and control bases research. One such action oriented field is called Internet based telerobotics. Telerobotics is an emerging field and one of the most actively researched.

Internet based telerobotics can be defined as a field of robotics where robots are controlled from a distance using the Internet as the medium of communication. The Internet is well poised to be the major medium of communication for teleoperation. However, Internet specific problems such as latency, uncertain data loss, and security of data transmission over a given network may lead to unpredictable operation and control.

For telecontrolled robotics, local intelligence at the robot side and optimal use of hardware and software components in the overall system can be used in mitigating uncertainty to achieve stability in the system. The proper use of hardware and software components can also reduce the overall cost of the system and concurrently reduce processing time. Using hardware which can be reconfigured or reprogrammed such as FPGAs and reusable software has an added advantage for future expansion. The addition of few sensors to the telerobot that can sense the environment will be an added benefit for stable and safe operation. Real time video can also provide the operator with feedback to perform a remote task with precision.

1.1 Motivation

In this study, a telerobotics platform appropriate for a real world application is developed. One intent is to reduce the problems due to latency and investigate design trade-offs to reduce the cost and improve the performance of a telerobotics platform. In general, the following factors are often taken into consideration for the development of any telerobotics platform,

- Uncertainty in the network such as latency and packet loss.
- Cost of the overall system.

Hence, in order to address the above mentioned issues, a semi-autonomous robotic platform was developed. This platform not only addresses the problem of latency and design trade-offs but was also developed and implemented within the budget provided.

The semi-autonomous robotic platform has its own intelligence (albeit limited) and can perform a given task even in the case of network uncertainty. Using a limited amount of hardware and processor power a telerobotics platform has been developed which can be extended to many applications and is also a cost efficient platform (within the provided budget). Alternative platforms could be developed with a considerably larger budget but at some point even these platforms will have to deal with some form of constraints. Meeting design specifications is always a challenging task.

1.2 Objective

The objective of this research was to develop a semi-autonomous robotic platform tolerant of the delay in the network, packet losses and within budget. In order to achieve this, a good understanding on how to use FPGAs, optimize hardware design for its very best performance, understanding of motor controllers and algorithms for automation was necessary. An overview of some existing Internet based telerobotics platforms was undertaken to better understand issues of implementation and optimization ideas for hardware and software co-design. As a consequence the approach used in this thesis was:

1. Selection of an appropriate embedded processor along with a Field Programming Gate Array (FPGA) board which can handle the required processes in real time.
2. Selection of an appropriate Real Time Operating System (RTOS).
3. Custom build the required processor and components for the FPGA board.
4. Acquire data from the sensors in order to make decisions locally to avoid obstacles in the robot path. This idea is to implement local intelligence in case of network failure or when the robot is used by a less experienced operator to keep the robot in a safe operating mode.
5. Design a motor controller circuit using pulse width modulation (PWM) for speed control.
6. Design a TCP/IP wireless communication link that can receive commands and send feedback to the operator.

7. Customize a Linux Kernel for Video Streaming and acquiring GPS data.

8. Develop graphical user interface for the operator to know the status of the robot and control the motion of the robot.

9. Develop an interface for a joystick that acts as a control device for the operator and a command generator for the robot.

10. Develop a video capture interface to display the video feedback provided by the robot.

Several researchers have placed a laptop on the robot, while other researchers have used a desktop computer allowing for rapidly developing their robotic platform. The approach in this research is to develop the entire control system incorporating local intelligence and wireless video feedback on an embedded platform. This approach consumes negligible power helping to limit battery power consumption and is closer to an industrial realization. As the communication protocols are IP based, the approach also provides the feasibility to operate anywhere provided there is some form of Internet connectivity.

Much of the research to date have used browser based interfaces for the control interface and video feedback, but in this thesis a Graphical User Interface was developed which is more flexible and can provided added security features. The downside however is the amount of time required in developing the user interface. The overall architecture of Internet based telerobotics platform architecture is shown in Figure 1. At the top level the whole system can be divided into a client and server architecture. The robot acts as a server and the controller act as client. In this thesis robot motors are controlled using the

PWM technique. The movement of the joystick generates commands to the robot, which might be to move forward, reverse, or in left or right direction.

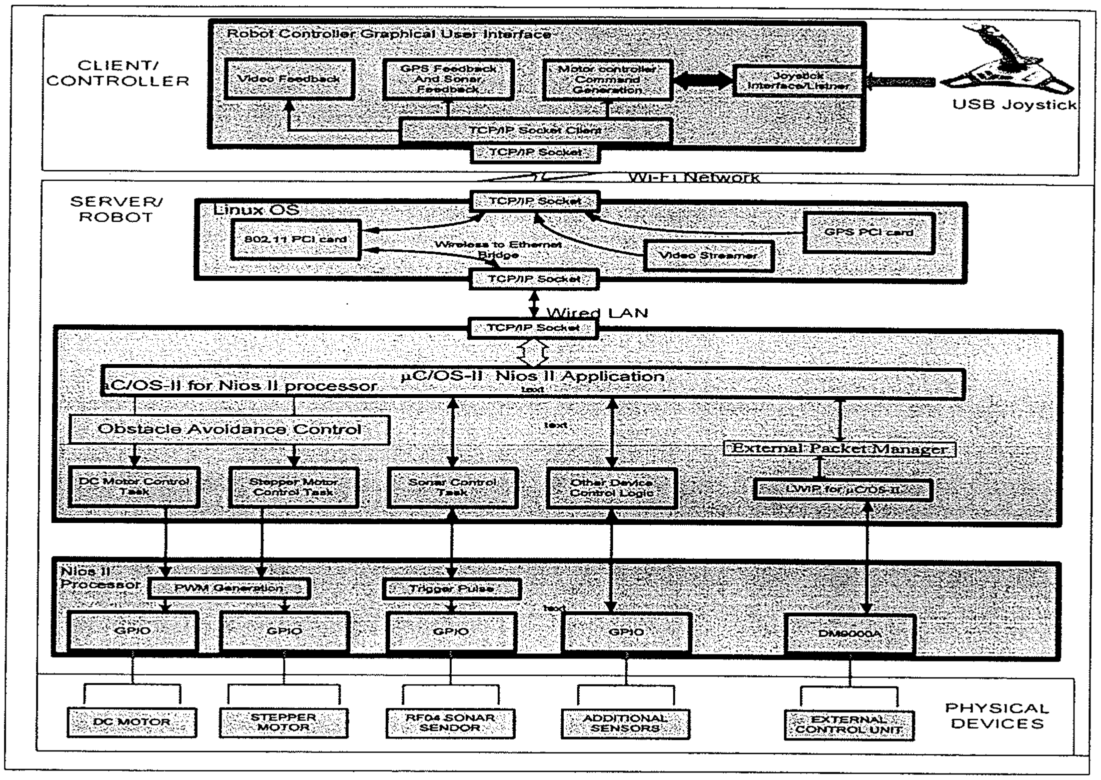


Figure 1: Telerobotics Platform Architecture

In addition to the basic features, some additional features such as rotate on the spot, turn abrupt left and turn abrupt right commands are also added. All these commands are generated by the joystick, converted in terms of duty cycles and streamed over the wireless network using TCP/IP socket streams. At the robot side a TCP/IP socket listen to the stream of commands sent by the controller. Every command is given a unique ID by the command generator on the controller side so that the robot server can recognize the commands and process the commands to perform the required action accordingly. At the same time a video capture device captures video and streams it over RTP/UDP, and at the controller end, the video is displayed.

Obstacle avoidance is implemented using sonar sensors. The robot is equipped with some degree of intelligence to avoid obstacles in its path of motion and decide which way to move on its own when an obstacle is detected and not attended to by the remote operator.

1.3 Problem Statement

One of the problems for Internet based telerobotics is the latency in the network and cost to build the overall system. In order to make Internet based telerobotics more practical, we should have a reliable and affordable Internet based telerobotics platform which can be extended to any application.

Hence, from an Internet based telerobotics perspective, we can state our problem as "How can one overcome the problem of latency and yet develop a reliable and affordable telerobotics platform"

1.4 Organization of Thesis

The remainder of the thesis is organized as follows:

Chapter 2 discusses the background study/ the literature, evolution and various methods in developing Internet based telerobotics.

Chapter 3 provides a detailed architecture on how the entire hardware for the telerobotics platform is developed.

Chapter 4 provides a detailed architecture on how the software for the user interface is developed, communication is established, coordination between the hardware-software and the user controls such as joystick are developed.

Chapter 5 provides the details in developing video streaming server, video capture and display, use of sonar sensors, GPS and implementation of obstacle avoidance control logic.

Chapter 6 provides the results on how practical and cost efficient the system is.

Chapter 7 gives a summary of my thesis and possible future work.

CHAPTER 2

BACKGROUND

Chapter Overview

This Chapter outlines a brief history of Internet based telerobotics, problems in telerobotics, existing models and methods used to overcome those problems and finally introduces some of the hardware concepts used in the work.

2.1 Internet Based Robotic Operation

Ever since the invention of telephone in 1870's there have been significant developments in the field of communication. The Internet – a very complex and revolutionary invention of 1965 has changed our world [WWIE]. The Internet can be defined as a global communication network consisting of millions of inter-connected networks.

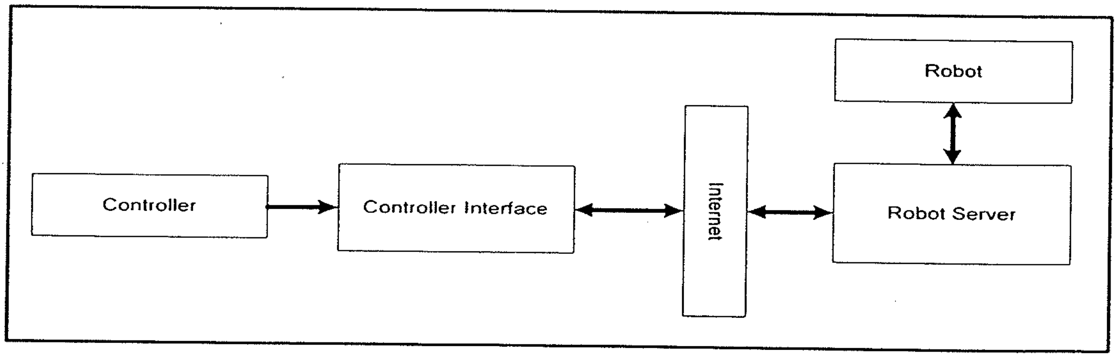


Figure 2: Basic Internet Based Telerobotics

This widely available means of communication is no longer just for data transmission, it can also be used to control robots at remote locations. The control of robots over Internet is termed Internet based Telerobotics.

2.2 History of Telerobotics

Telerobotics is an area of robotics where the robot is controlled from a distance by the controller using a means of communication channel. The means of communication may be wired or wireless. Prior to 1945 there were crude teleoperators for earth moving, construction and related tasks. The first Master-Slave Manipulator, was publicly demonstrated by its inventor, Ray Goertz [TBS92] , at the Argonne National Laboratory of the U.S. Atomic Energy Commission in 1951 [WWRG]. The master slave manipulator was basically an electrical and hydraulic servomechanism. In 1954 a closed circuit television was introduced so that operation could be from an arbitrary distance away [TUNO].

In 1961, an experimental manned submarine intended for deep submergence, the Bathyscaphe Trieste, was equipped with a telemanipulator based on and controlled unilaterally by a keyboard [TUNO]. The mechanisms were immersed in an oil bath, including the electric motors, with the oil staying at the same pressure as the water, independent of the depth reached, in a water-proof casing. In 1966 a Cable Controlled Underwater Research Vehicle (CURV), retrieved a nuclear weapon that had fallen into the sea off the Spanish coast at Palomares, this project directed the attention of the whole world to the existence and usefulness of such devices [TUNO]. By 1965 experiments in academic research laboratories had already revealed the problems of telemanipulation and vehicle control as a consequence of time delay, and the early lunar teleoperated

Surveyor demonstrated the problems vividly in an actual space mission [TUNO].
In 1967, Surveyor III landed on the surface of the Moon. It was equipped with manipulator arms, which took samples of lunar soil and measured the force required to carry out this operation [TUNO]. This was the first example of teleoperation in outer space. The exploratory mission of the Soviet Lunakod followed. This vehicle was telecontrolled directly from earth with only seconds of delay in the transfer of information. The main disadvantage of teleoperation in outer space is the delay in the two-way transmission (which depends on how far the telerobot is from the controller) of commands and information, which the operator could not overcome. The Draper Laboratory at MIT took up this work, and developed the idea of computer-aided teleoperation. At the same time, teams from the Marshall Space Flight Center at Huntsville, the Johnson Space Flight Center at Houston and from Stanford concentrated on the transmission delay effect and computer control. The Viking spacecraft, which landed on Mars in 1976, was programmed to carry out strictly automated operations. This manipulator arm which was more efficient than the Surveyor in taking samples, placing them in an analysis chamber and moving objects to detect changes of color of the soil under rocks [TUNO].

The first successful implementation of Teleoperation via the Internet was developed by Goldberg in 1994 at the University of South California. This Mercury Project, as it was called included the operation of a simple robotic manipulator with CGI (common gateway interface) program interface and video feedback. It was the first laboratory where users using the World Wide Web could order the robot to perform tasks in order to uncover buried artifacts in a sand filled terrarium. The Mercury Project was online for 7 months from September 1994 to March 1995 and received over 2.5 million hits [TUNO]. This breakthrough helped turned Internet based Teleoperation into the huge and ever growing field of research it is now.

An Italian group led by Professor Rovetta has reported several experiments investigating the possible applications of telerobotics, and claims to have carried out the first telerobotic surgery in 1995 [BLP03], a prostate biopsy. Another of the telesurgery projects completed involved the team at the Brady Urological Institute, who designed and developed a robot capable of performing a remote percutaneous renal needle puncture [BLP03].

The Basic issues that are important when dealing with Internet based robotics are listed below

2.2.1. On the Network Side

- Packet Loss
- Latency

2.2.2. At the Robot Side

- Degree of Autonomy
- Suitable Hardware
- Suitable RTOS

2.2.3. At the Operator Side

- Appropriate Programming language for developing the Graphical User Interface
- Appropriate Hardware for controlling the robot

2.3 Packet Loss

One or more packets of data failing to reach their destination due to oversaturated network links, faulty network hardware design, signal degradation over the network medium, rejection due to a corrupted packet, routing routines and maligned system driver devices across a computer network can be termed packet loss.

There are a few network transport protocols such as Transmission Control Protocol (TCP) (RFC 761) which provide reliability in delivering packets. In the case of TCP, whenever a receiver detects a packet loss, the receiver effectively request the transmitter to retransmit the lost packets or the sender automatically sends the packets which have not been acknowledged after a certain period of time lapses. TCP uses a sliding window protocol [RFC1323] for acknowledgements of received packets, this causes a drop in throughput of the connection while improving reliability. In real time control there is every need that packets are delivered on time so that tasks are executed with synchronization. Knowing the End to End delays and loss behavior in a network are very important factors [VER99] in developing real time applications. In general a few packets lost can be neglected for real time applications. If we can trade-off the reliability of packet delivery and include some kind of local intelligence at the robot side then we can either use the User Datagram Protocol (UDP) (RFC 768), Real Time Transport Protocol (RTP) (RFC 1889) or light weight User Datagram Protocol (UDP Lite) (RFC 3832). UDP

Lite is the newest standardized IP transport protocol for error-prone network environments. UDP, RTP and UDP Lite do not guarantee packet deliver like their counterpart TCP. UDP avoids overhead checking to see whether every packet actually arrived or not, which makes UDP faster and more efficient. RTP can be used for video transport, RTP can be used either with TCP or UDP as the transport layer. RTP when used along with Real Time Transport Control Protocol (RTCP) (RFC 3550) can also provide some degree of guarantee of packet delivery.

2.4 Latency

Latency can be defined as the amount of time taken by the data packet to travel from the source to destination in a computer network, or it can also be called the delay in the data packet delivery. End To End delay is the accumulation of transmission, processing and queuing delays in routers, propagation delays in the link and end to end processing delays. Irregular time delay is inevitable and an unpredictable phenomena which is caused by network congestion that needs to be taken into consideration.

In the case of real time operation, robot control over the Internet with uncertain time delay can result in instability and asynchronous operation which will affect the dynamic performance of the system. There is a real need that we either reduce the amount of time delay [MS05] [RGG97] or provide a local intelligence for stable operation. There is very little scope in reducing the time delay in a given network unless dedicated lines are used. Applying local intelligence at the robot side is the most effective way to overcome the problem of asynchronous action and instability caused by time delay [MS05].

2.5 Degree of Autonomy

Robots can be classified in accordance to their degree of autonomy as follows [JJG03]

- **Non Autonomous Robots** : Robots which have to be controlled by the operator and do not process any kind of intelligence locally to perform a task without the supervision of the controller are classified as Non Autonomous Robots
- **Semi Autonomous Robots**: These robots have some degree of artificial intelligence and can perform some task or portion of a task even without the direct supervision of the controller.
- **Autonomous Robots:** These robots have incorporate artificial intelligence algorithms for tasks such as path planning, collision avoidance etc. and require little or no operator control during operation.

To some extent we can overcome the problem faced in the real time control of robots over the Internet by deploying either Autonomous Robots or Semi Autonomous Robots. This can be achieved by introducing artificial intelligence methods such as Fuzzy Logic, Genetic Algorithms or Neural Networks for some of the decision making. Semi Autonomous or Autonomous robotics that are developed using real time operating systems (RTOS) such as Micro C/OS II, μClinux, etc., also contributes to improved performance.

2.6 Local Intelligence at Robot Side

2.6.1. Event Based Telerobotics

To overcome the problems cause by time delay, an event based control can be used which is a non time action reference [JJ03]. This can be achieved by implementing some local intelligence like a fuzzy controller at the robot side. Figure 3 [JJ03] shows the event based control of a robot or a manipulator over the Internet.

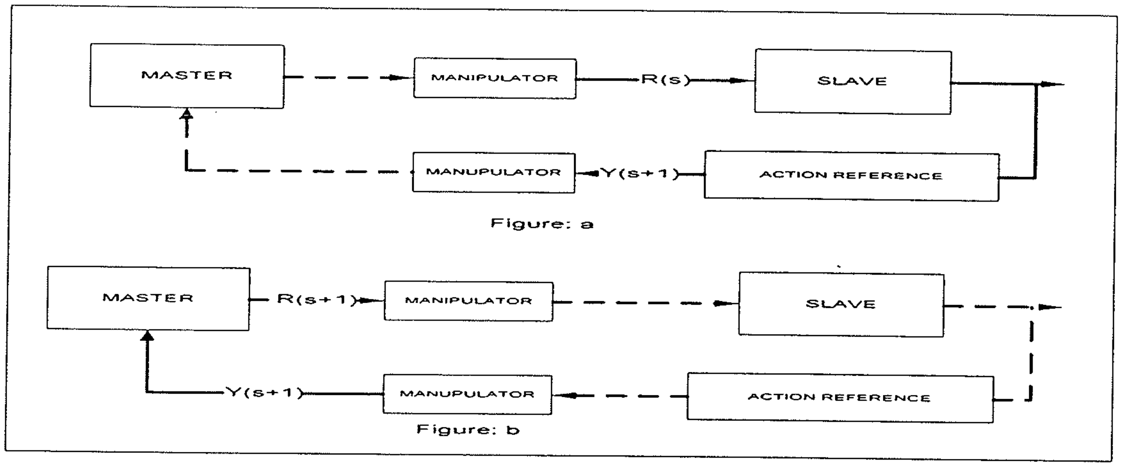


Figure 3: The Event Based Control through Internet (derived from [JJ03])

The basic idea of event based planning and control theory is to introduce a new motion reference variable different from time, but directly relate the sensory measurement of the system. Instead of time, the planned/desired system output is parameterized by the new motion reference variable called an Event [NTA96] [NT99]. From Figure 3 we can see that the new commands R(s+1) are not transmitted until the feedback from the previous command Y(s+1) is received from the previous event of the slave. On the other hand the slave holds the robot by either giving a command for zero velocity or the previous position data into the local controller and no feedback is provided to the master until the next command has arrived. In this system every single action is considered as an event independent of time. Using this feedback system, even though there is a delay in the data packets that carry the commands to the robot, due to the local intelligence and event based approach action synchronization and stability can be achieved. Even though stability and action synchronization can be achieved using the event based approach, there is always a disadvantage that a continuous stream of commands cannot be executed. The controller has to wait until he/she receives a feedback from the robot to send their next command.

2.6.2. Fuzzy Controlled Robots

Mobile robots have to react to what they sense in the environment. Using sensor and video feedback along with some artificial intelligence such as fuzzy logic an autonomous robot can be developed. Fuzzy logic is a method of solving control problems that provides a solution for implementation in systems ranging from simple, small, embedded micro-controllers, to large, networked, multi-channel PC or workstation based data acquisition and control systems. Fuzzy controllers are flexible and easy to implement in hardware, software or a combination of both. Microcontrollers such as MC68HC12MCU come with some fuzzy logic instructions [EM94]. Fuzzy logic provides a simple way to arrive at a definite conclusion based upon vague, ambiguous, imprecise, noisy or missing input information. The fuzzy logic approach to a control problem mimics how a person would make a decision, although typically much faster [SHM03][JN91]. Figure 4 shows a simple fuzzy logic control system. The most import task in fuzzy logic control is to determine what should be controlled and how it should be controlled. For simplicity consider an example to control a robot in an unknown environment which has a simple collision detection sensor. The data collected by the collision detection sensors are given as input to the fuzzy logic controller and the commands sent by the controller to the remote robot act as other input signals to the fuzzy logic controller. When a collision detection sensor detects an obstacle in its path of motion, it sends that signals to the fuzzy logic controller and the fuzzy logic controller outputs a stop signal to the robot even though the input command from the main controller is to move forward. If no feedback is given by the collision detection sensor then the fuzzy logic control outputs the commands sent by the controller.

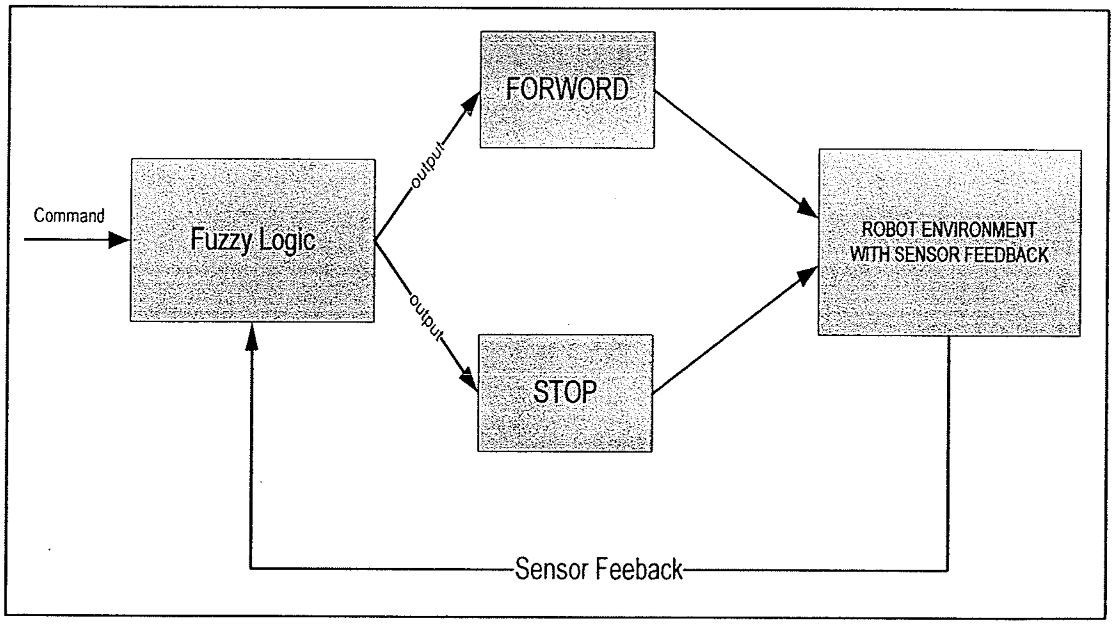


Figure 4: Simple Fuzzy Logic Control System

Applying such intelligence provides safe and reliable operation of the telecontrolled robot. The problems such as packet loss and latency can become obsolete or at least are mitigated if we can implement an appropriate fuzzy logic controller.

2.6.3. Behavior Based Telerobotics

Behavior based robotics is a methodology of developing Artificial Intelligence based on modular intelligence. In behavior based system the intelligence is controlled by a set of independent semi-autonomous modules. Figure 5 shows an implementation of behavior control system (Finite State Acceptor (FSA)) [RCA98]

FSA provides a ready mechanism to express relationships between various behavioral sets and are widely used within robotics to express control systems. FSA provide us with a higher level of abstraction by which we can express the relationship between sets of behaviors. In Figure 5 each behavior is represented as a state, which encodes the robot's goal of moving around an open terrain to locate its target locations. The targets can be anything such as locating a classroom in a hallway. Consider Figure 5, the robot has three major behavioral states, wander, move to target, and return to start.

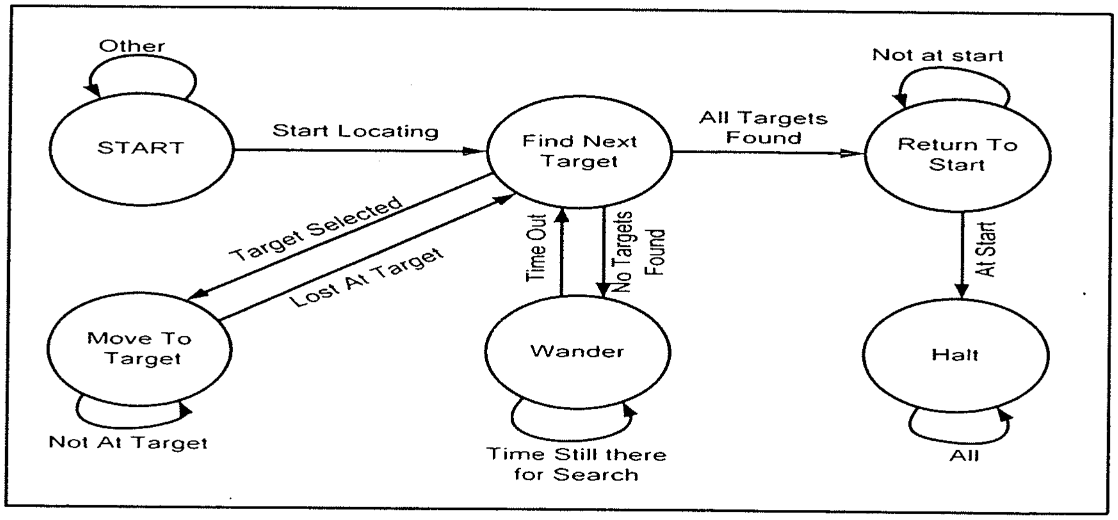


Figure 5: Finite state Acceptor (FSA) (derived from [RCA98])

Move to target consist of a subset action for selecting a target, orienting the robot so it points towards the target, moving to the target, and tracking the target visually during motion until it is reached. Any number of subsets can be created for each set or behavior. FSA shows the sequencing between behaviors as the robot carries out its mission.

2.6.4. Behavior Based Fuzzy Control Telerobotics

Higher level behavior based robots require some intelligent control techniques such as model based fuzzy control [RDH97] or genetic algorithms. Fuzzy control can be applied to mobile robots which have complex control architectures [AS97]. Behavior based telerobotics can be modeled as shown in Figure 6 [SSDN00] used for helpmate robot [WWVU] which has sonar sensors. In Figure 6, the robot has three behaviors; 1. Task Oriented Behavior; 2 Obstacle Avoidance Behavior; and 3 Emergency Behavior. Each behavior represents a concern in the mobile robot control and relates it to sonar sensor data, robot status data, and goal information to control the robot. A task oriented behavior typically has more subsets, such as follow goal and follow wall.

To implement fuzzy control for any task or behavior, there are three basic steps

1. Fuzzification
2. Develop an Inference Engine and
3. Defuzzification.

In Fuzzification the real valued points are often mapped to fuzzy sets by treating them as Gaussian membership function, triangular membership function etc. The Gaussian membership function is given by [SK00]

$$\left.\begin{array}{l} \mu_{A'}(x) = \exp[-((x_1 - x_1^*)^2 / \sigma_1^2))]*\ldots \\ \ldots\exp[-((x_n - x_n^*)^2 / \sigma_n^2))] \end{array}\right\} \quad [1]$$

From these membership functions we arrive at a logical decision in an Inference Engine using If-Then rules. The fuzzy rule base converts input information into output membership functions. Finally Defuzzification is applied to these inferences and the output of the Defuzzification is given as an input to the robot for processing. There are many methods that can be used to convert the inference engine output such as the Center of Gravity method as given in equation [2] [SK00].

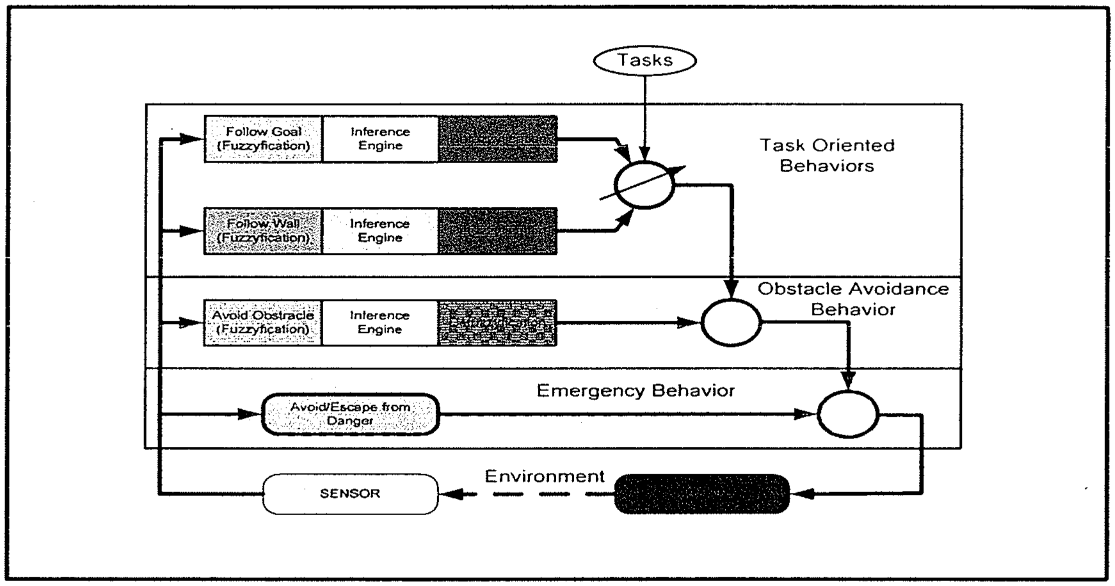


Figure 6: Behavior Based Fuzz Control Robot [SK00]

$$COG, y = \frac{\sum_{i=1}^{n} \mu_A(y_i) y_i}{\sum_{i=1}^{n} \mu_A(y_i)} \qquad [2]$$

2.7 Suitable Hardware

To develop a real time control for robotics, choosing proper hardware or choosing an appropriate board which has minimal processing latency is required. Depending on the requirements we can opt for a Single Board Computer (SBC), Field Programming Gate

Array Board (FPGA) or a combination of both. Choosing an appropriate processor and required peripherals such as USB ports, Ethernet ports, PCI slots etc. is also very important. Communicating and sensing of the robot environment using devices such as video, sonar sensors, GPS, Infra Red, etc. are also important design considerations.

2.7.1. Single Board Computers (SBC)

SBCs are complete computers built on a single circuit board. The design is centered on a single or dual microprocessor with RAM, I/O and all other features needed to be a functional computer on the one board. Single computers boards are basically Hard Core Processors, once designed they cannot be changed.

2.7.2. Field Programming Gate Array (FPGA) boards:

FPGAs are semiconductor devices containing programmable logic devices called "Logic Blocks" and programmable interconnects. Logic blocks can be programmed to perform the function of basic gates or more complex combinational functions. Most logic blocks also include memory elements which are either Flip-Flops or complete blocks of memories. The major advantage of FPGAs is, they can be re-programmed any number of times. All modern FPGAs also support a soft core processor architecture.

The logical blocks and interconnection of traditional FPGAs combined with an embedded soft core multiprocessor and related peripherals form a complete "System on Programmable Chip "(SOPC). Boards such as Altera DE2, Xilinx Vertex-II PRO etc. are all SOPC's. Numerous Intellectual Properties (IP) cores are now available which can be added on to the system using software such as SOPC builder for Altera boards. This helps the developer to choose the required hardware for a specific application and greatly reduces the design time.

2.8 Embedded Systems and Real Time Operating System

An Embedded System is described as a special purpose computer which performs one or many dedicated functions [WWW1]. A Real time Operating System (RTOS) is a multi tasking operating system best suited for real time application such as real time robot control. Embedded systems and a RTOS work hand in hand as shown in Figure 7. Choosing a proper RTOS and an appropriate embedded board is an important consideration when it comes to real time applications.

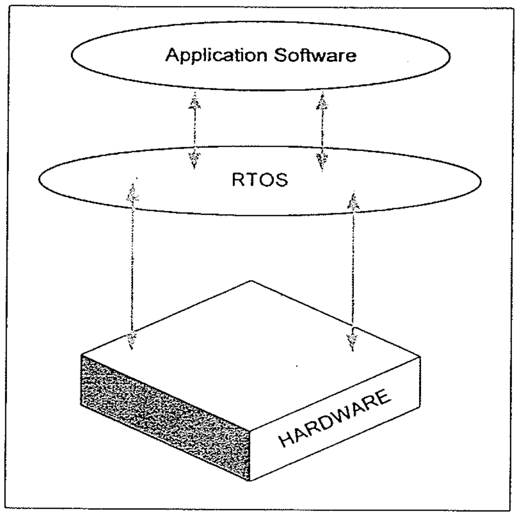


Figure 7: RTOS Abstraction Layer between Application Software and Embedded Hardware [WWW1]

In a RTOS multiple tasks are run by switching tasks or threads [WWW1]. A task is nothing but a simple program. An RTOS can be designed either as event driven design, time sharing design or a combination of both.

- Event Driven Design: Switching between tasks is done only when a higher priority task needs to be served, (denoted a preemptive priority).
- Time Sharing Design: Switching between tasks takes place on a clock interrupt or on an event, task switching can also follow a round robin schedule [WWW2].

In a typical design, at any point in time a task can be in any one of the following three states [ROJ04] Running state, Ready state or Waiting state.

- **Running State:** In this state, the given task is being executed by the microprocessor. Only the task that is in the running state is processed unless the processer is a multiprocessor.

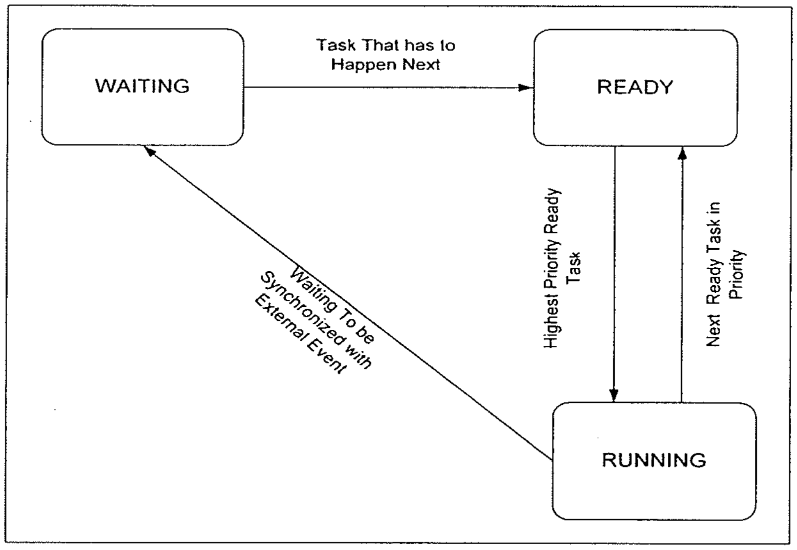

WAITING
Task That has to Happen Next
READY
Highest Priority Ready Task
Next Ready Task in Priority
Waiting To be Synchronized with External Event
RUNNING

Figure 8: Task States (from [ROJ04])

- **Ready State:** When some other task is in the running state or is being processed by the microprocessor the task that is ready to execute is in a ready state, and will execute when the microprocessor becomes available. There could be any number of tasks in this state.

- **Waiting State:** In this state the task has nothing to do even when the microprocessor is ready to process this task. A task enters this state because it is waiting for some external event to be synchronized with this task.

Part of an RTOS is a scheduler [DAES] which keeps track of the state of each task and decides which task should go into the running state. There are a wide variety of RTOSs available to choose from in the present day market.

2.9. Embedded Processors

Embedded processors can be divided into two distinct categories

1. **Microprocessors:** Integration of numerous useful functions into a single IC package, with the ability to execute a stored set of instructions to carry out user defined tasks are called microprocessors. Microprocessors also have the ability to access external memory chips to both read and write data.

2. **Microcontrollers:** A microcontroller is a device which integrates a number of components of microprocessor systems onto a single microchip. A typical microcontroller has a Central Processing Unit (CPU), memory (both RAM and ROM), parallel digital I/O, a trimmer module, an analog to digital converter (ADC) and a serial I/O port.

There are many different CPU architectures used in embedded system such as x86, PIC, ARM, Power PC etc. Embedded systems communicate with the outside world via peripherals such as RS-232, USB, and Ethernet.

2.9.1 Nios II Processor

A Nios II processor system is equivalent to a microcontroller or a “Computer on chip” that includes a processor and a combination of peripherals and memory on a single chip.

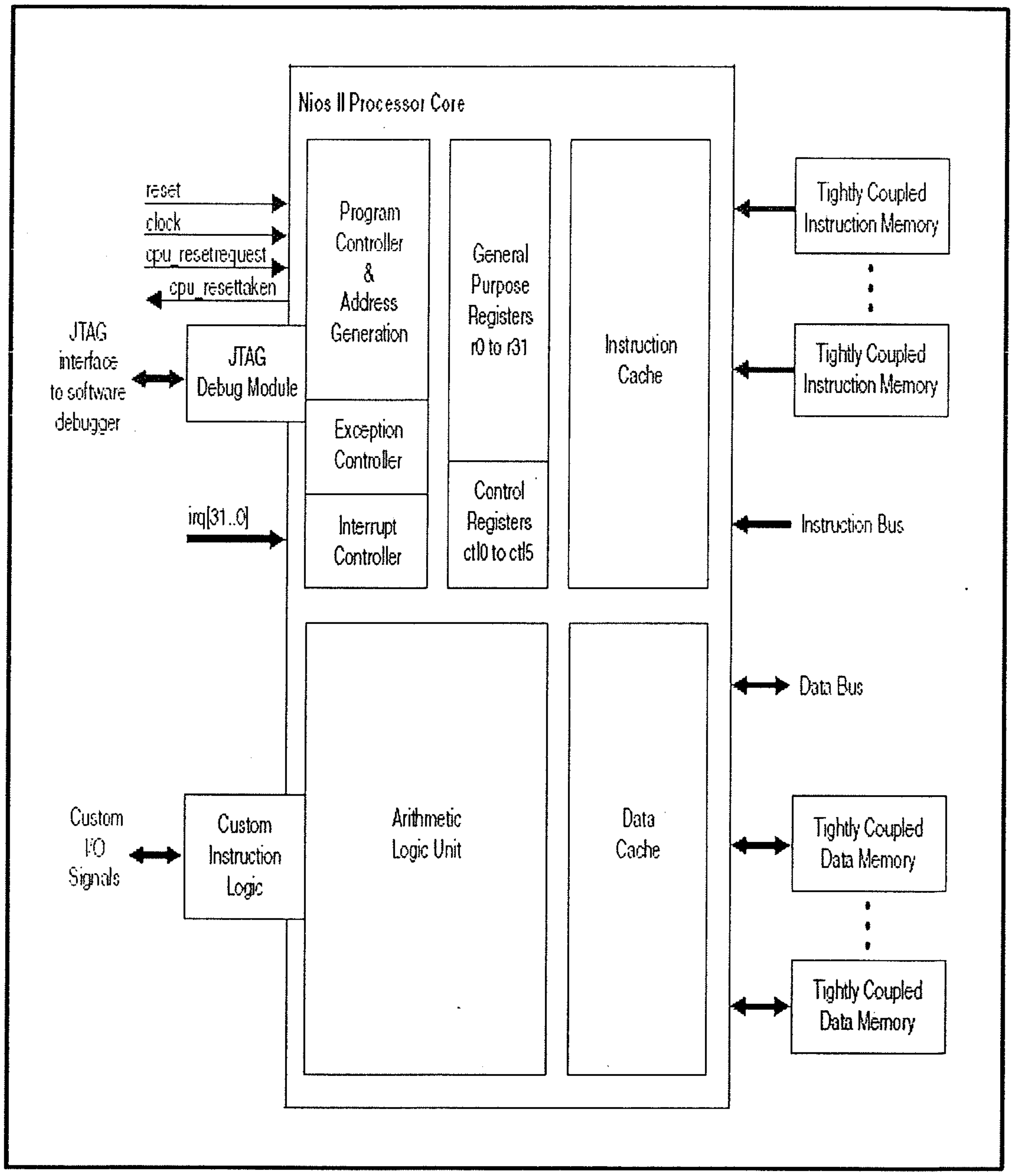

Nios II Processor Core
reset
clock
cpu_resetrequest
cpu_resettaken
JTAG interface to software debugger
JTAG Debug Module
Program Controller & Address Generation
General Purpose Registers r0 to r31
Instruction Cache
Tightly Coupled Instruction Memory
Tightly Coupled Instruction Memory
Exception Controller
Interrupt Controller
irq[31..0]
Control Registers ctl0 to ctl5
Instruction Bus
Data Bus
Custom I/O Signals
Custom Instruction Logic
Arithmetic Logic Unit
Data Cache
Tightly Coupled Data Memory
Tightly Coupled Data Memory

Figure 9: Nios II Soft Core Processor (from [ALTE0])

Like any other microcontroller, all Nios II processor systems use a consistent instruction set and programming model. The environment on which application software can be developed for Nios II processor is called as a Nios II integrated development environment (Nios II IDE) which is based on GNU C/C++ compiler and eclipse IDE. A Nios II processor is a soft core processor which can be configured for a particular application for its best performance. Peripherals can be developed and added to the Nios II processor which cannot be done easily with microcontrollers. Because of this flexible and soft-core processor we can easily use the Nios II processor system with the exact peripheral set required for the target application.

The Nios II architecture describes an instruction set architecture (ISA). The ISA in turn necessitates a set of functional units that implement the instructions. The processor core does not include peripherals or the connection logic to the outside world. It includes only the circuits required to implement the Nios II architecture. The Nios II processor cores with all the functional units are shown in Figure 9 [ALTE0]. The functional units of the Nios II architecture form the foundation for the Nios II instruction set. Functional units of the Nios II processor can be implemented in hardware emulated in software or omitted entirely.

2.10 Micro C/OS II (μC/OS-II) RTOS

μC/OS-II is a highly portable, ROMable, scalable, preemptive real time, multitasking kernel for microprocessors and microcontrollers [ALTE1]. μC/OS-II runs on a large number of processors architectures and can be ported to a wide range of processors. One of the key advantages of μC/OS-II is that execution time does not depend on the number of tasks running in an application which results in providing a consistent and deterministic performance.

Altera implementation for μC/OS-II [ALTE1] is shown in Figure 10. The Hardware Abstraction Layer (HAL) is a lightweight runtime environment that provides a simple device driver interface for programs to communicate with the underlying hardware. The HAL [ALTE2] application program interface (API) is integrated into an ANSI C standard library which allows users to access devices and files using standard C library functions.

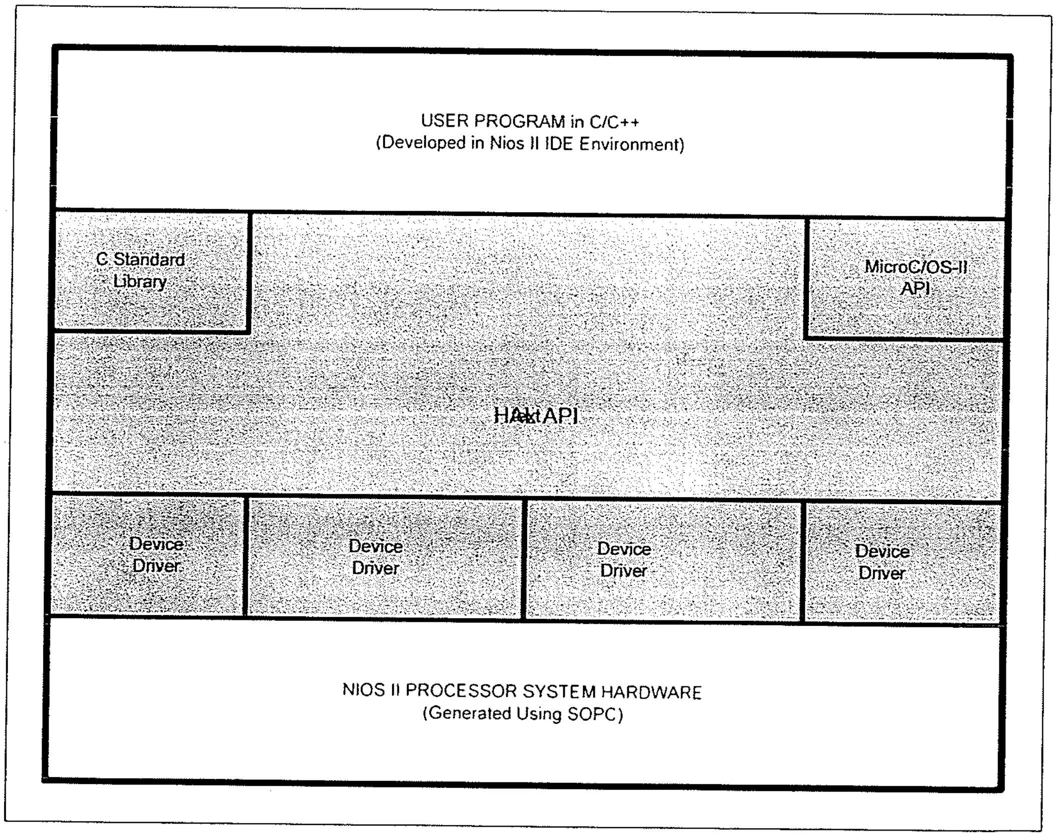


Figure 10: Micro C/OS II Programming Architecture [ALTE2]

HAL serves as a device driver package for Nios II processor systems, providing a consistent interface to the peripherals in the system. μC/OS-II for Nios II processor is essentially a superset of the HAL. The HAL environment is extended by the inclusion of the μC/OS-II scheduler and the associated μC/OS-II API. The complete HAL API is available from within the μC/OS-II.

2.11 Appropriate Programming Language for Developing a Graphical User Interface

Web browsers which are traditionally used for controlling robots over Internet are very vulnerable to hackers. Alternatively programming languages such as Visual Basic, C or C++ [MSDN] can be used for developing a custom GUI. However these programming languages are tedious and have very limited user classes needed to develop a high level GUI. C# from Microsoft provides a wide range of built in classes and provides rich options of looks for developing the front end application of the GUI. Visual C++ which is fast and reliable can be used to develop the applications within the GUI for communication and encryption purposes.

2.12 Summary of Chapter 2

Chapter 2 described a brief history and evolution of telerobotics. An overview of the problems associated with latency and methods that can be used to overcome latency were discussed. Different approaches to achieve autonomous and semi autonomous modes of operation were briefly described. This chapter also explained the importance of choosing appropriate hardware. Basic working principles of a RTOS and the architecture of the Nios II processor and Micro C/OSII RTOS programming architecture were also described. Finally a discussion of why C# was chosen as the programming language for GUI development was presented.

CHAPTER3

HARDWARE IMPLEMENTATION.

Overview

This Chapter explains in detail the working principle of PWM, generation of PWM signals using the DE2 board, working principles of motor controllers, differential steering and our wireless configuration.

Parts of this chapter were done in collaboration with Monir Khan. Specifically we both developed our own versions of the module required. The final version that was used was the one that had greater functionality. Some aspects were designed in collaboration while others were solely the work of the author. Detailed contributions are footnoted in each section.

3.1 Pulse Width Modulation

Pulse width modulation is basically the digital encoding of an analog signal level. The voltage or current source is supplied to the load by means of a repeated switching between **on** and **off** of the supply. The *on* time is when a supply voltage is applied to the load and the **off** time is when the supply voltage is cut off from the load. The proportion of time during which a component or a device is operated or is in the **on** state is defined as a **duty cycle**. Considering the pulse train in Figure 11 [WWDC] with T as the period and τ as the duration of the pulse which is non zero we can mathematically represent the duty cycle D as in equation [3]

$$D = \frac{\tau}{T} \qquad [3]$$

Figure 12 [WWVDC] shows three different PWM signals with 10%, 50% and 90% duty cycle. PWM output at 10% duty cycle means that the signal is **on** for 10% of the period and **off** for 90% of the period

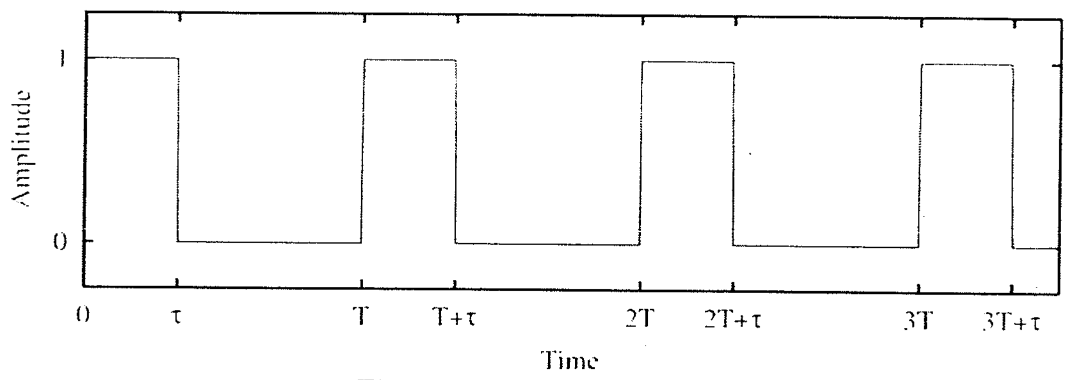


Figure 11: Pulse Train

This implies that the PWM output encodes the analog signal at 10% of full strength. For example, if the supply voltage is 9 and the duty cycle is 50%, on average a 4.5 analog signal results. Various duty cycles PWM signals are illustrated in Figure 12.

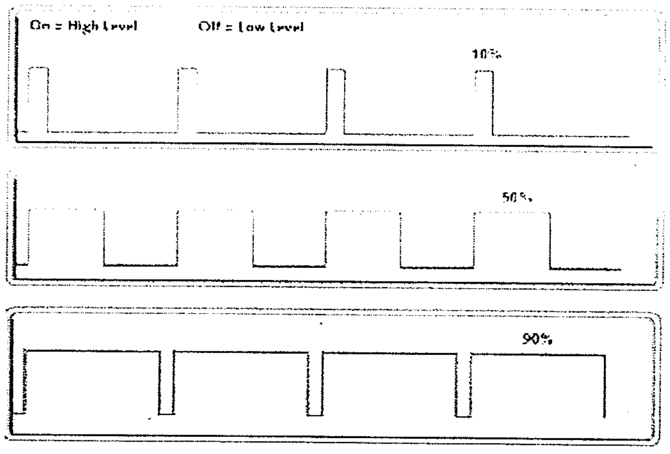


Figure 12: PWM Signals of Varying Duty Cycle (from [WWDC])

The average value of the pulse train in Figure 11 is given by:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt$$

$$= \frac{1}{T}\left(\int_0^{\tau} y_{\max} dt + \int_{\tau}^{T} y_{\min} dt\right.$$

$$= \frac{\tau y_{\max} + T(1-\tau) y_{\min}}{T}$$

We know that $\tau = DT$, so we have

$$\bar{y} = D.y_{\max} + (1-D)y_{\min}$$

At $y_{\min} = 0$ $\quad \bar{y} = D.y_{\max}$ [4]

From equation 4 it is evident that the average value ($\bar{y}$) is directly dependent on the duty cycle D.

3.2 Generation of PWM Using Altera DE2 Board1

A DE2 board with Nios II processor running on Micro/OSII was used to generate PWM signal for controlling the motion and direction of the robot's electric motors. The HDL logic used to produce PWM is given in Appendix A. The system configuration of the DE2 board is shown in Figure 13.

Using the task logic (Appendix A) code and register mapping a custom PWM_Z component is created. The component serves the following purpose.

1 With the specific task of the PWM motor controller I was responsible for developing motor controller and direction control using PWM and M. Khan was responsible for developing PWM using the DE2

- It defines the interface to the component hardware, such as the names and the type of I/O signals.
- PWM_Z declares the PWM_Z component and specifies the logic that has to be used to produce the require PWM signal.
- It describes a graphical user interface for configuring an instance of the component in SOPC builder.
- It provides script and other information that the SOPC builder needs to generate the hardware description language (HDL) files for PWM_Z and integrate the PWM_Z into the system module.
- It contains PWM_Z related information, such as register memory map and Avalon memory mapped interface.

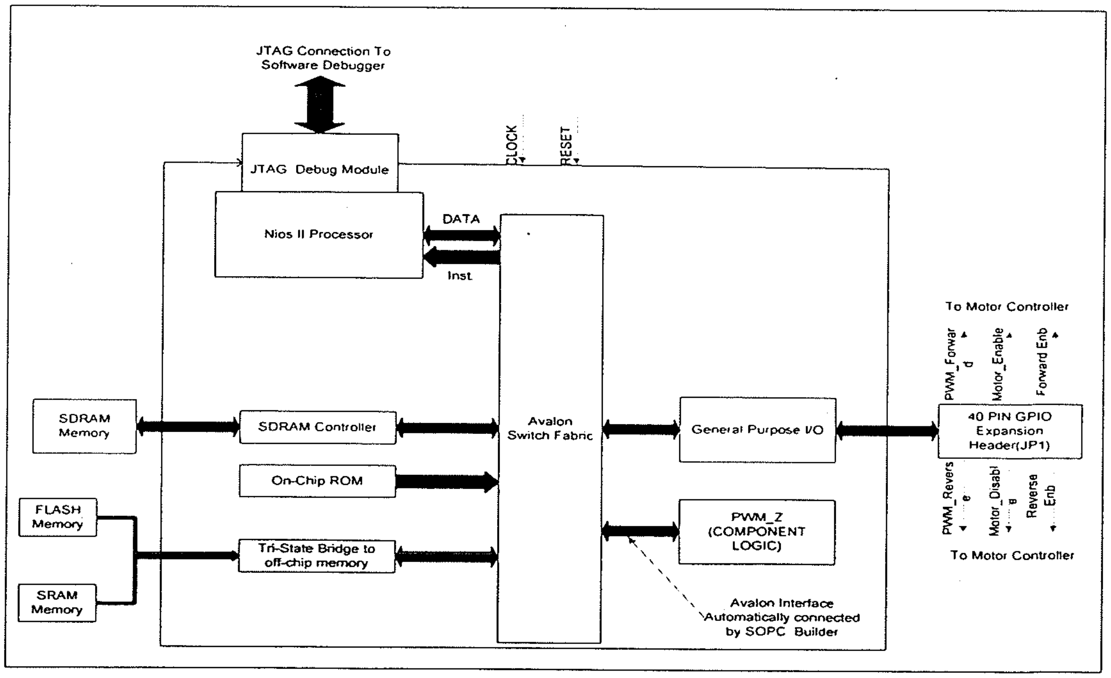


Figure 13: System Configuration of the DE2 board to Produce PWM Signals

PWM_Z connects to the system interconnect fabric using the Avalon memory mapped interface. A single PWM_Z component can provide more than one Avalon port, in other words a single PWM_Z component can provide ports for PWM_Forward to control the forward motion, PWM_Reverse to control the reverse motion of the robot and so on.

3.3 Motor Controller

PWM Signals generated by the DE2 are then given to the motor controller. An Open Source Motor controller (OSMC) [WWRP] from Robot Power [WWRP2] was used. The motor controller is based on the H-Bridge principle. For the robot to move forward

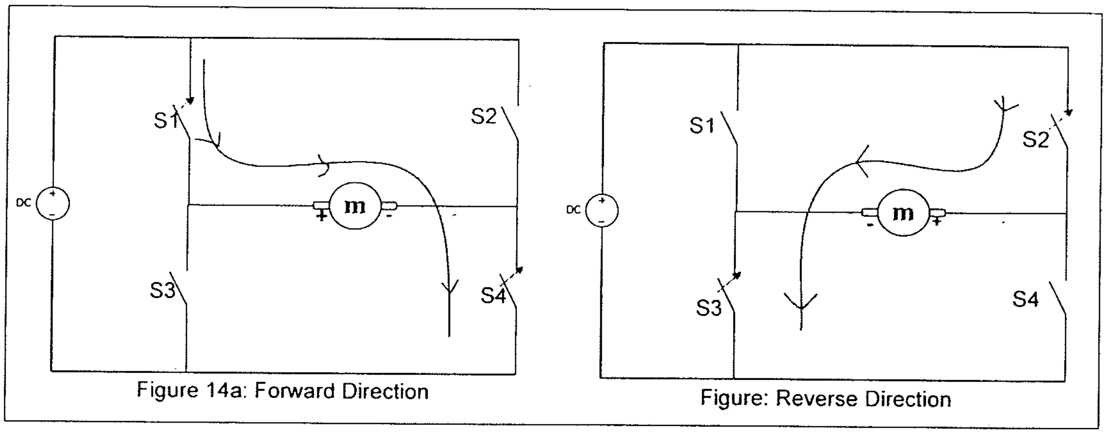


Figure 14: Two Basic States of H-Bridge

Switches S1 and S4 are closed as shown in Figure 14a, then there is a positive voltage across the motor terminal which rotates the shaft in forward direction. Conversely when switches S2 and S3 are closed as shown in Figure 14b, the voltage across the motor is reversed and results in the reverse motion of the shaft. In reality the switches are replaced with the solid state switches such as MOSFETs.

The ON states and OFF states of the MOSFET are triggered by giving a PWM pulse at the gate terminal. The detailed schematic of the motor controller along with the protection circuits for shoot through is shown in Figure 15.

When voltage is applied between the gate and source terminal an "inversion channel "is created which creates a conduit through which current can pass. By varying the voltage between the gate and the source, the current flow between the drain and the source can be controlled. The average voltage across the motor is controlled by rapid switching action of the MOSFET. This switching action is controlled by the PWM pulse, depending upon the required speed; the average voltage across the motor is varied. Say for example a motor runs at full speed in forward direction at a rated voltage of 24V, then we should apply a PWM with a duty cycle of 100 % at Q1 and Q4 MOSFET''s. Similar if we want the motor to run at half the rate speed we apply a PWM with a duty cycle of 50%.

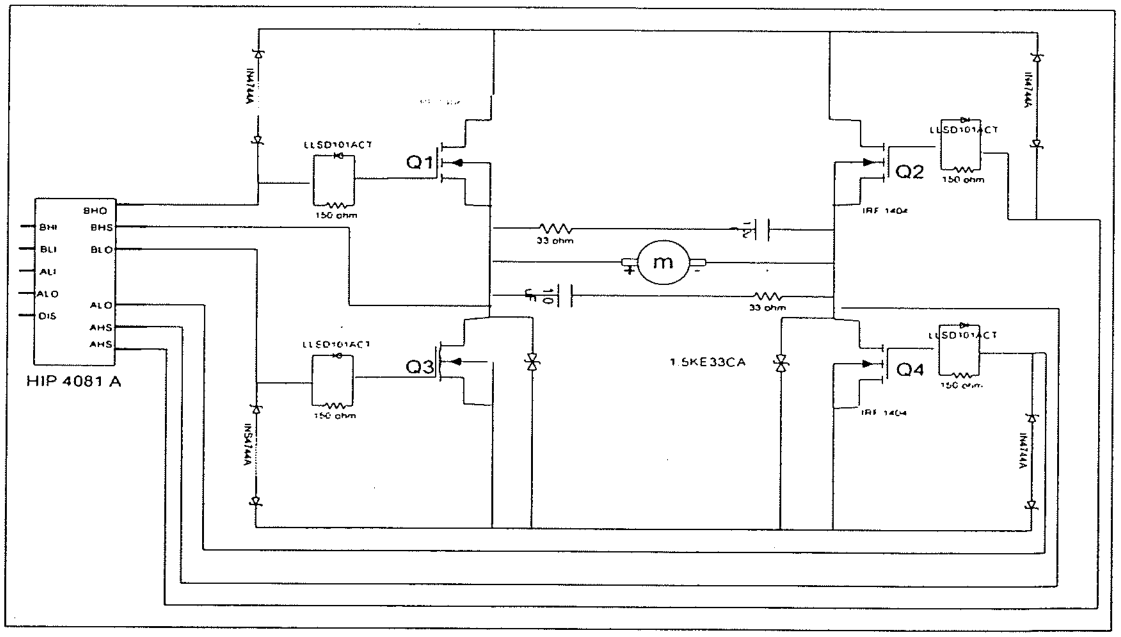


Figure 15: Motor Driver

In practice for a motor to run forward the signals Motor Enable [Figure 13] and Forward Enb [Figure 13] or Q4 [Figure 15] are kept at high and a PWM signal is given at PWM_Forword [Figure 13] or Q1 [Figure 15], while the rest of the signals are kept low . Depending on how much speed is required the duty cycle is varied. For the robot here, there are 2 independent motors for the left and the right wheel, each motor is controlled independently. For moving forward or reverse the PWM duty cycle remains the same for both motors whereas the PWM duty cycle is different for both the motors if the robot has to move either to the right or to the left.

3.4 Robot Direction Control

Instead of having a separate motor just for steering the robot a principle called a differential steering system is applied to the robot to achieve the required direction of motion. Two wheels mounted on individual axis are independently powered and controlled thus providing both drive and steering capability.

Steering control of the robot was purely based on differential velocities, even though complex algorithms can be implemented to achieve differential steering such as in [WWDS]. When we want the robot to turn left as shown in Figure 16A, the speed of the outer wheel (right wheel) should have higher speed compared to inner wheel (left wheel) Due to the difference in the speeds of each wheel the robot tends to move towards the lower wheel speed, i.e. the wheel on the right covers more distance compared to the wheel at the left in a given time, which results in a turning of the robot to the left. From Figure 17 the x and y co-ordinates of the robot center will change depending on the speed of the motion along the direction vector.

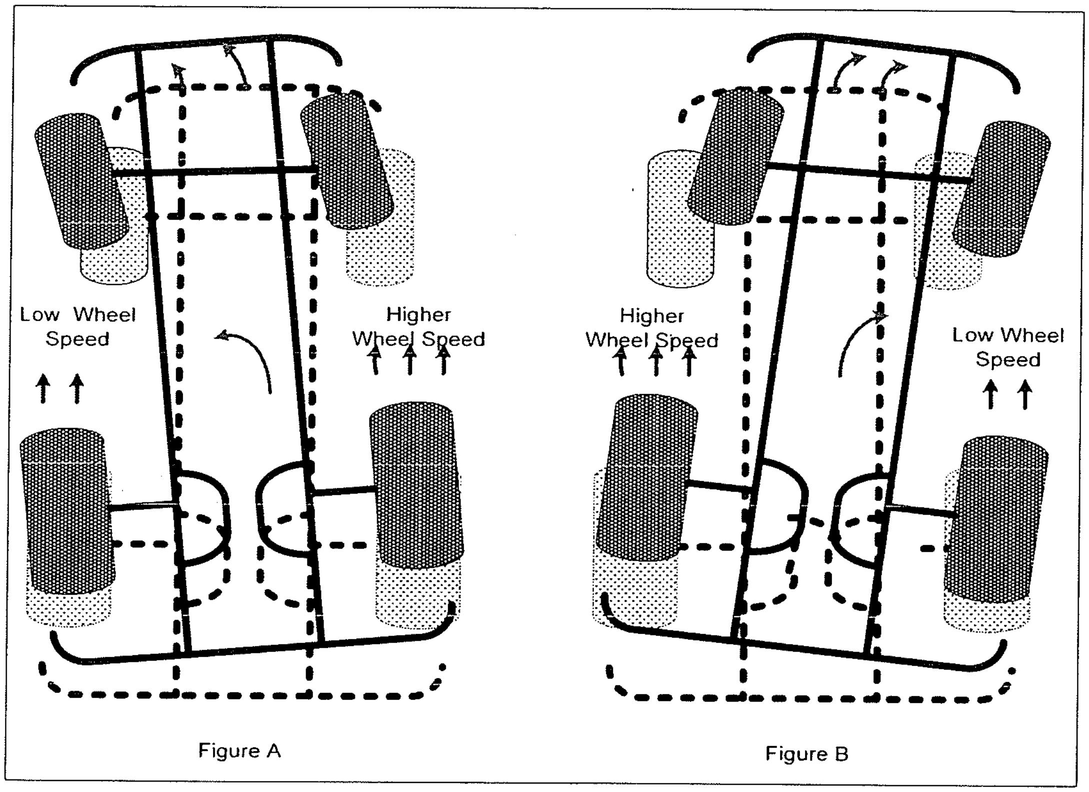


Figure 16: Differential Steering System

The direction giving the forward motion of the robot will simply be in terms of $\sin\theta$ and $\cos\theta$. Where θ is the angle of turn in radians.

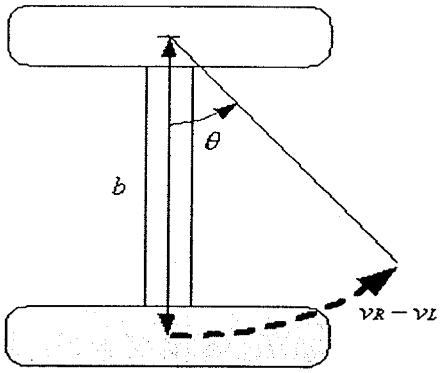


Figure 17: Wheels at Different Velocities

Taking m(t) and θ(t) as the time dependent function for our robot speed and direction , we have

$$\begin{aligned} dx/dt &= m(t)\cos(\theta(t)) \\ dy/dt &= m(t)\sin(\theta(t)) \end{aligned} \qquad [5]$$

We can define angle as the length of the arch divided by the radius of a circular arch. The length of the arc from Figure 17 is the relative velocity of the right wheel which gives the length of the arc per unit time and the length from the wheel to the center point gives us the radius, combining the above two facts we have

$$d\theta / dt = (VR - VL)/b \qquad [6]$$

Note: This approximation equation uses the center point of the left wheel as a reference point. All motion in this frame of reference is treated relative to the left-wheel point. Because the right wheel is mounted perpendicular to the axle, its motion *within the frame of reference* follows a circular arc with a radius corresponding to the length of the axle (from hub center to hub center).

Integrating the above equations and taking the initial orientation of the robot as $\theta(0) = \theta o$ we find a function for calculating the robot's orientation as a function of wheel velocity and time:

$$\theta(t) = (VR - VL)/t + \theta o \qquad [7]$$

Velocity is simply the average of that for the two wheels, or $(VR + VL)/2$ we combine this fact with what we know (Equation 5) about orientation as a function of time, and get the following differential equations:

$$\left.\begin{aligned} dx/dt &= [(VR+VL)/2]\cos(\theta(t)) \\ dy/dt &= [(VR+VL)/2]\sin(\theta(t)) \end{aligned}\right\} \qquad [8]$$

Integrating and applying the initial position of the robot, $x(0) = xo$ and $y(0) = yo$, we have

$$\left.\begin{aligned} x(t) &= xo + \frac{b(VR+VL)}{2(VR-VL)}[\sin((VR-VL)t/b+\theta o)-\sin(\theta o)] \\ y(t) &= xo + \frac{b(VR+VL)}{2(VR-VL)}[\cos((VR-VL)t/b+\theta o)-\cos(\theta o) \end{aligned}\right\} \quad [9]$$

With different VL, VR and θ the values for y(t) and x(t) were calculated. These calculated values were used as reference values to calculate the velocities VR and VL for a given turning angle. However considerable error is introduced by the caster wheel hitting bumps and deflecting the robot. Wheel slippage also contributes to the uncertainty in controlling the robot with differential steering.

3.5 Configuring Ethernet on DE2 Board2

To control the robot, that is, to send commands to the robot and receive feedback from the robot environment we need a means of communication. The most widely available communication technology is associated with the Internet. As such, the DE2 board which acts as a main control unit has to be connected to the external world through Ethernet and eventually to a wireless access point. To configure Ethernet on the DE2 we configured the Devicom DM9000A fast Ethernet controller chip The DM9000A includes a general processor interface, 16Kbytes SRAM, a media access control(MAC) unit and a 10/100 PHY transceiver.

2 I was responsible for configuring Ethernet on DE2 board while M.Khan was responsible for building custom Linux kernel and wireless configuration.

The DE2 architecture uses DMA technology to increase CPU usage and time it is connected via the Avalon Bus allowing improved data channel and SDRAM efficiency. The communication interface is shown in Figure 18. An Ethernet component developed by Terasic [WWTS] was used which provided the required driver for DM9000A. This driver was also used to develop the required user application such as creating a TCP/IP stack, web server, etc., for the NiosII processor using C/C++ language in the NiosII Integrated Development Environment.

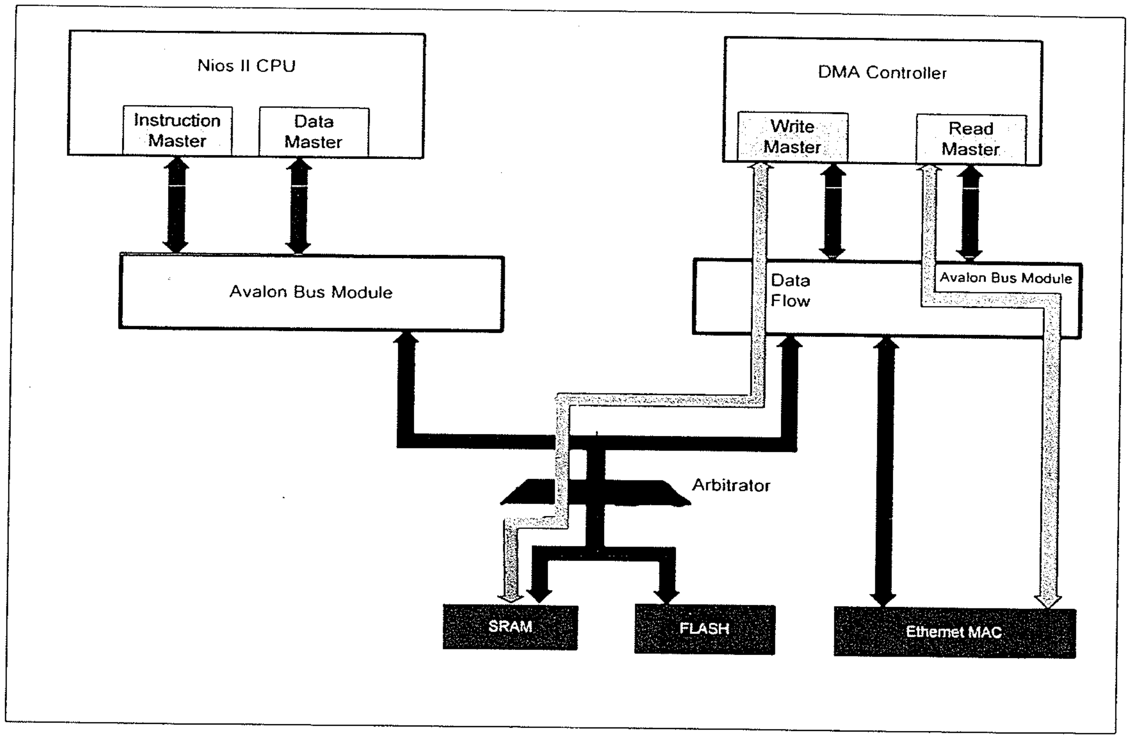


Figure 18: Communication Interface

3.6 Ethernet to Wireless Using Soekris Board

To make the robot more mobile and practical, it is necessary that we make the robot a wireless mobile robot. During development we could communicate to the robot by means of Internet Protocols a wired LAN. Subsequently it became necessary to broadcast these signals to a base station through a wireless medium, such as a wireless local area network (Wi-Fi) [WWWF]. The task of creating a wireless network look pretty simple, but they are not as simple as they appear. We had to interface the robot to 802.11b, which provided the Wi-Fi network to the DE2 board or the robot server which has Ethernet configured in it. Due to the limitation of SDRAM, flash memory on the DE2 board and limited support for USB we could not use wireless adapters or a wireless bridge. As an alternative we used a Soekris net5501 embedded processor which provided the solution for making the robot wireless.

The Net5501 comes with 1 PCI slot which is used for the wireless PCI card. To configure the wireless PCI slot for 802.11b, a custom Linux kernel had to compiled and a proper driver had to be installed. The Ethernet cable from the DE2 board was connected to the Ethernet hub (one port of the net5501). A bridge was created between the Ethernet hub and the 802.11 b PCI card, so that the packets from the DE2 pass through the bridge and then to the 802.11 b PCI card and are then broadcast. The base station receives these signals and passes them on to the client (or controller) which is connected through a LAN. When a client wants to send a command signal to the robot, the signals (packets) are first transmitted to the base station which is connected via the wired LAN and the base station broadcasts the signals which are in turn received by the 802.11b PCI wireless card and passed on to the Nios II processor for processing the commands and performing the required task.

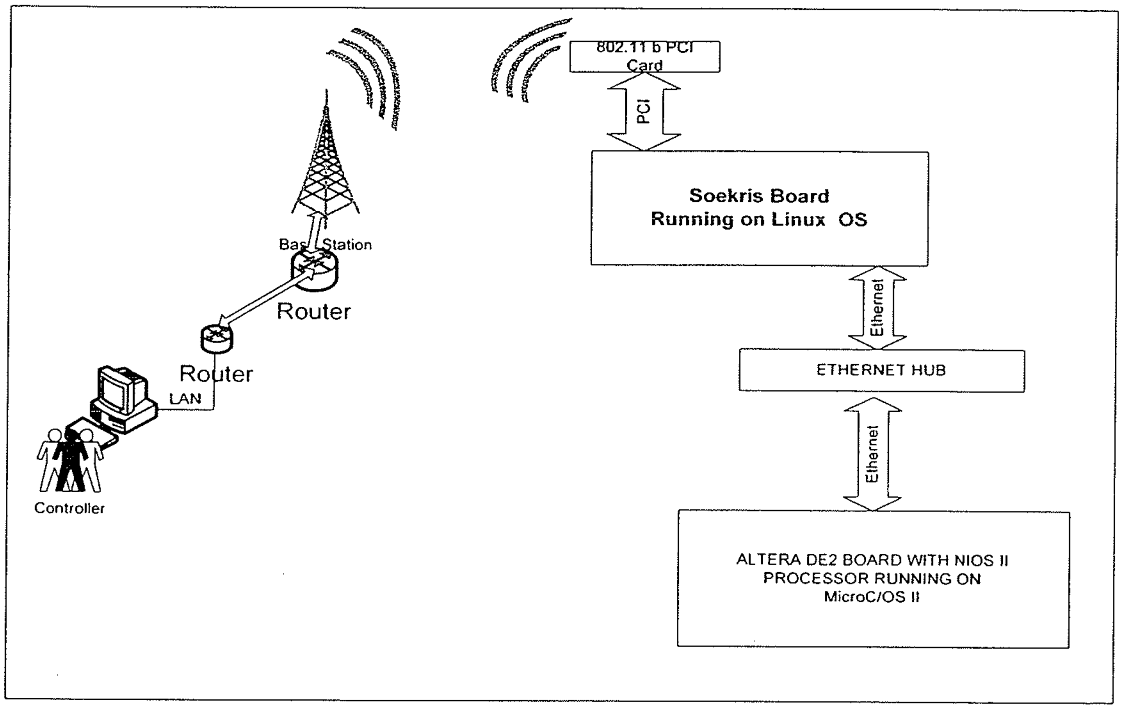

802.11 b PCI Card
PCI
Soekris Board Running on Linux OS
Ethernet
ETHERNET HUB
Ethernet
ALTERA DE2 BOARD WITH NIOS II PROCESSOR RUNNING ON MicroC/OS II
Base Station
Router
Router
LAN
Controller

Figure 19: Ethernet to Wireless

3.7 CMOS Camera

A TRDB_DC2 1.3Mega pixel digital camera module was used for capturing high resolution video. This module can use two CMOS image sensor (MT9M011) which are capable of 1280x1024 resolution. This camera module outputs a raw RGB which can be processed and displayed either on a VGA monitor or can be transmitted over Wi-Fi network with image compression. The module is connected to the 40 pin GPIO header of the DE2 board. The images are captured using a HDL code written in Verilog. The code captures the images from the CMOS sensors through the first 20 GPIO pins and stores them in the SDRAM of the DE2 board for further processing. The images are stored in raw RGB format, which are further processed and displayed on the VGA monitor. Difficulties were encountered when trying to stream the video from the CMOS camera over the Wi-Fi network.

3.8 Summary of Chapter 3

In this chapter the design concepts for PWM generation using the DE2 board and direction control which together form the core of any robot at the hardware level has been presented. This chapter also explains the process and steps necessary to configure a Wi-Fi, so that the robot can communicate with the external world remotely. Finally the configuration and implementation of a CMOS camera using a DE2 board is described.

CHAPTER 4

SOFTWARE IMPLEMENTATION

Chapter Overview

This Chapter explains the functionalities of the GUI developed and techniques used in transformation of joystick stick movement by the controller into robot commands

Parts of this chapter were done in collaboration with Monir Khan. Specific responsibilities are footnoted in the appropriate sections.

4.1 Graphical User Interface3

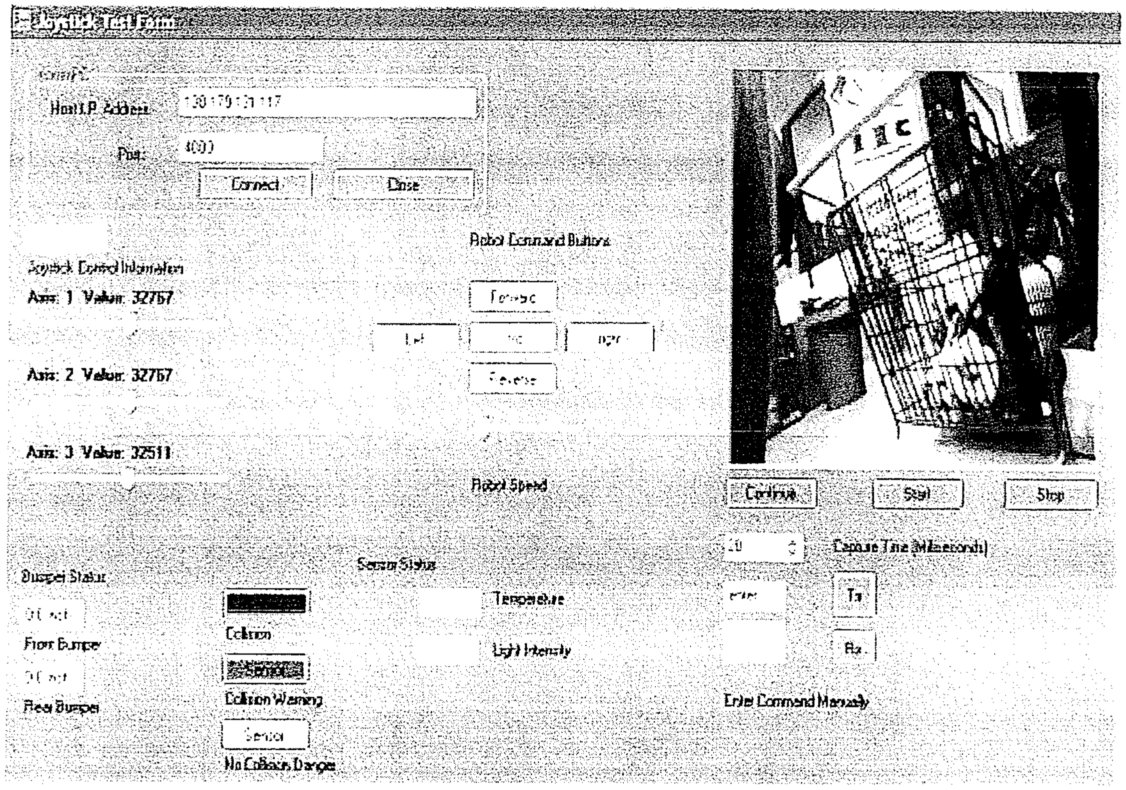
For any type of operator assisted robot, the operator environment is very important. To make things simple and user friendly, a Graphical User Interface (GUI) was developed in the .Net environment. The languages used for the development are C# and VC++. The GUI developed shown in Figure 20 can communicate with the DE2 board on the robot to send commands or receive the feedback from the robot/server (recall that DE2 board acts as a server).

The GUI provides the following functionalities:

- Connects the client/controller to the server/robot using socket connections.
- Provides near real time visual data feedback in MJPEG format to the controller for reliable control operation.

3 With the specific task of the GUI development, I was responsible for the video interface, M. Khan was responsible for joystick interface, and we both collaborated on layout of GUI and socket programming.

- Provides GPS data that helps the operator know the exact location of the robot.



Host I.P. Address: 130.179.131.117

Port: 4000

Axis: 1 Value: 32767

Axis: 2 Value: 32767

Axis: 3 Value: 32511

Figure 20: Graphical User Interface

- Provides feedback such as speed and the distance from the obstacle if any are in its path of motion.
- Provides an interface to a joystick and converts the joystick commands which are in terms of the x and y axis into motor duty cycles and forwards these command packets to a socket which forwards them to a socket at the server side to perform the required action.

4.2 Joystick Interface

Extreme 3 joysticks from Logistic was used in the development of the robot controller. To make the code more robust and reusable a joystick interface was developed. Figure 21 shows the flow chart for joystick interface. The joystick interface basically is a class, the main purpose of joystick interface is to poll the external USB devices connected to the computer and set the device to active if it finds a suitable joystick that can be used to control the robot. The following function polls for a joystick and throws an error if it doesn't find a joystick.

```
private void Poll()
     {
        try
        {
            joystickDevice.Poll();
                        state = joystickDevice.CurrentJoystickState;
        }
        catch (Exception err)
        {
          Debug.WriteLine("Poll()");
          Debug.WriteLine(err.Message);
          Debug.WriteLine(err.StackTrace);
        }}
```

Once the joystick is detected, the joystick interface looks for the joystick properties such as the number of axis it can operate and the number of buttons attached to it. For the robot control, we required a minimum of 4 axes and 5 buttons if not, it throws an error. Most joysticks basically have 2 or 3 axes i.e. with X and Y as its co-ordinates and Z in some cases. The Z axis is not taken into account here. To make things simple and easier for development, the joystick interface developed uses axes A, B, C and D. Axis A represents +Y and Axis B represents –Y, similarly Axis C represents +X and Axis D represents –X. If a joystick is moved forward that is in +X direction, the interface will

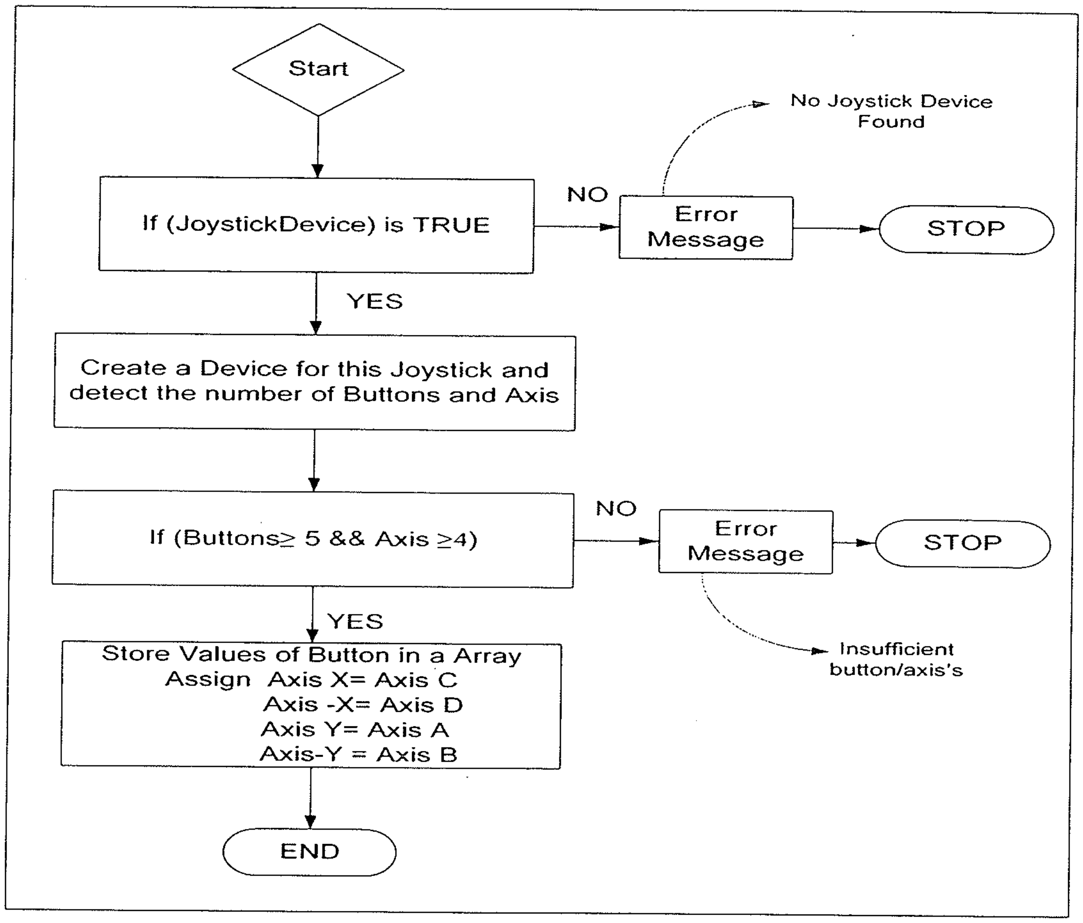

Start
If (JoystickDevice) is TRUE
NO
Error Message
STOP
No Joystick Device Found
YES
Create a Device for this Joystick and detect the number of Buttons and Axis
If (Buttons≥ 5 && Axis ≥4)
NO
Error Message
STOP
Insufficient button/axis's
YES
Store Values of Button in a Array
Assign Axis X= Axis C
Axis -X= Axis D
Axis Y= Axis A
Axis-Y = Axis B
END

Figure 21: Joystick Interface Flow Chart

send a command to the DE2 or Nios II processor to produce a PWM with exactly the same duty cycle to both left and right motors so that the motors will have the same speed and direction, in this case the robot moves forward. In the process a need arose to convert the value read from joystick into an integer value between 0-100 which represents the duty cycle.

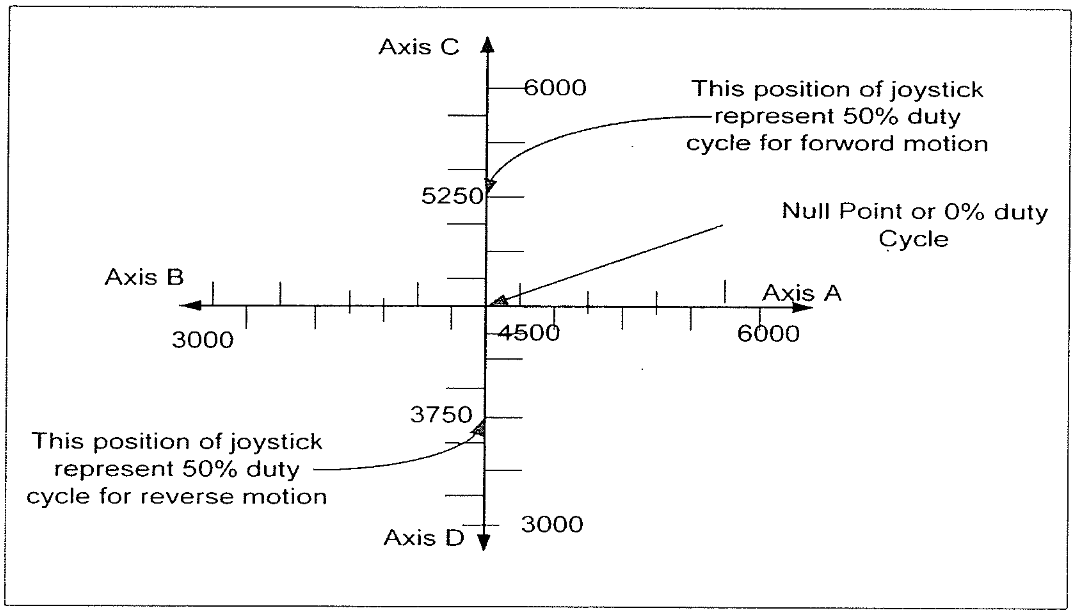

Axis C
6000
This position of joystick represent 50% duty cycle for forword motion
5250
Null Point or 0% duty Cycle
Axis B
Axis A
3000
4500
6000
3750
This position of joystick represent 50% duty cycle for reverse motion
Axis D
3000

Figure 22: Joystick Mapping

As an example, if the maximum and minimum values between Axes C and D are 6000 and 3000 respectively as shown in Figure 22, where the null position or 0% duty cycle is 4500, then every 15 divisions read from joystick represent 1% duty cycle. To be more precise for the motor to move at half the rated speed in forward direction a duty cycle of 50% is required , for which the joystick has to move 50*15 divisions i.e. is 750 divisions from the null position (4500) which is at 5250 divisions.

<u>**Note:**</u> The Values and Axes used in Figure 22 are used for explanation

4.3 Client Sockets

A socket is basically an end of a bi-directional communication link in an IP networking protocol. Using sockets at both Client (Controller) and Server (Robot) a communication channel is established for data flow across the network. The DE2 board (Robot Controller) is configured to obtain its IP address (every computer on the Internet has a 32 bit address, often referred to as its IP address) dynamically when it is connected to a network which has a DHCP server. In the case where we do not have a DHCP server we can configure the DE2 board to have a static IP address. In our case the IP address for the DE2 board is dynamical configured and displayed on the LCD panel of the DE2 board. Knowing the IP address of the DE2 board a socket connection between the client and the server is established using the IP address of the host machine, (in addition a port is required (a port is a 16 bit unsigned integer)). Lower port numbers are reserved for standard services; hence any port above 2000 can be used. Two fields in Figure 20, host IP address and port number should be typed into the GUI. Clicking on the button **Connect** the following function is executed

```
private void cmdConnect_Click(object sender, System.EventArgs e)
    {
      try
      {
        //create a new client socket ...
        m_socWorker = new Socket(AddressFamily.Internetwork, SocketType.Stream,
ProtocolType.IP);
        String szIPSelected = txtIPAddress.Text;
        String szPort = txtPort.Text;
        int alPort = System.Convert.ToInt16(szPort, 10);

        System.Net.IPAddress remoteIPAddress =
System.Net.IPAddress.Parse(szIPSelected);
        System.Net.IPEndPoint remoteEndPoint = new
System.Net.IPEndPoint(remoteIPAddress, alPort);
        m_socWorker.Connect(remoteEndPoint);
        cmdConnect.Enabled = false;
        cmdClose.Enabled = true;
      }
      catch (System.Net.Sockets.SocketException se)
      {
        MessageBox.Show(se.Message);
        cmdConnect.Enabled = true;
        cmdClose.Enabled = false;
      }
    }
```

A new socket connection is created with the IP address provided in the txtIPAddress and the port number specified in txtPort. When the host (robot) accepts the connection, the client can keep sending commands using a socket stream function. At the host side (robot), a server socket is created.

4.4 Server Sockets

The host runs a µC/OS II as its RTOS(Real Time Operating System) that uses the LWIP (Light Weight Internet Protocol) and NicheStack TCP/IP.

The following lines of code illustrate how a socket connection is created on port 4000

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);

memset (&serv_addr,0,sizeof(serv_addr));

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(4000);
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) <  0)
{
 printf(" [send task ]ERROR on binding");
}

listen(sockfd,5);
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd,
   (struct sockaddr *)&cli_addr,
    (socklen_t*)&clilen);
if (newsockfd < 0)
{
 printf (" [send task ] ERROR on accept");
}
```

The server socket is now ready to accept the incoming message from the client. It is necessary that the commands generated by the client are streamed in a proper format such that each command can be recognized and processed accordingly by the server.

4.5 Command Generation

Commands generated are streamed through the communication link established between the client and server using sockets. The whole system is divided into two parts; one is the controller/client and the other server/robot. When a command is given by the controller using a joystick, the interface code check for several conditions and interprets what commands should be sent. A flow chart in Figure 23 represents on how every command is interpreted

4.5.1 CASE1: To Generate Commands for Left and Right Turn

When the controller enters CASE1, the axis is checked. If the axis is A and button0 is pressed, it moves on to check the value of the Axis. If the value resulted in the motion of the joystick which is less than 32111(Figure 24), then the duty cycle for left motor (DutytCycle1) and right motor (DutyCycle2) are calculated as shown in Figure 23. The calculated duty cycle value is between 0 and 1, if the resultant value is 0.25 that implies a 25% duty cycle. The calculated duty cycles along with a letter "L" to indicate that the user wants the robot to move left is packed into a single text value and sent as a command. The commands produced in this manner are streamed continuously over the Wi-Fi using sockets. If the value resulted in moving the joystick is greater than 32911 (Figure 24) the controller moves to an "else" loop and the duty cycles for right motor and left motor are calculated. The calculated duty cycles along with the letter "R", to indicate that the controller wants the robot to move right is packed into a single text value and sent as a command.

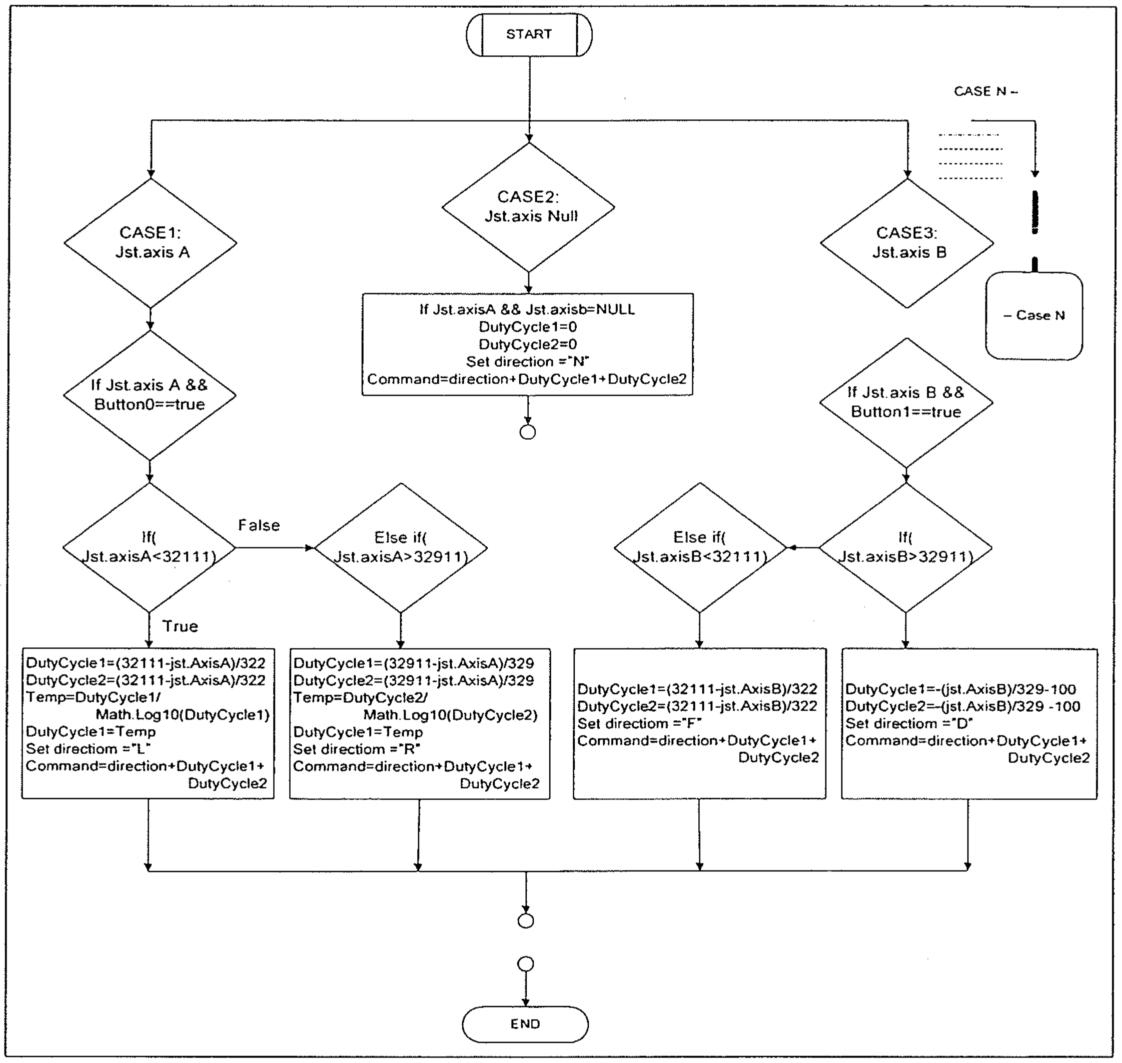


Figure 23: Command Generation Flow Chart

The command structure to turn right is as follows:

```
cl =1, cr = 0.2;                \\Constant
sl = (int)(cl * dutycycle1);        \\Casting
sr = (int)(cr * dutycycle2);
direction = "R"         \\Casting
command = "+";
command += direction + "+" + sl + "+" + sr + "+";
```

Similarly to turn Left:

```
cl =1, cr = 0.2;                \\Constant
sl = (int)(cl * dutycycle1);        \\Casting
```

```
sr = (int)(cr * dutycycle2);          \\Casting
direction = "L"
command = "+";
command += direction + "+" + sl + "+" + sr + "+";
```

Note: The Symbol "+" at the begning indicates the start of a command

4.5.2 CASE2: Neutral

When the joystick is in the null position, that is values between 32111 and 32911 on both

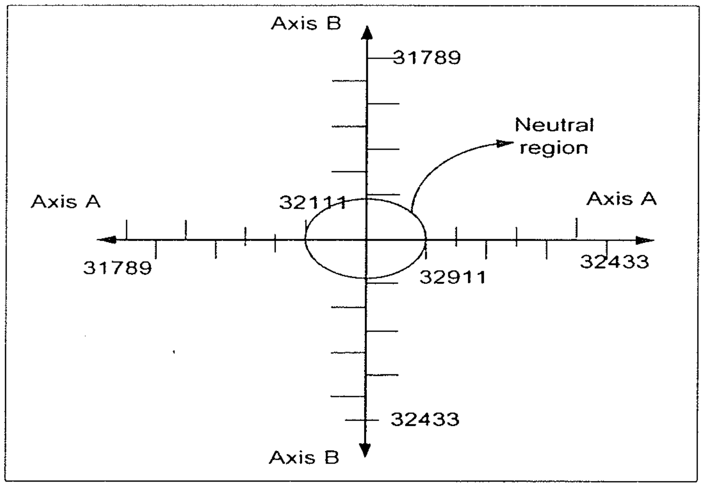


Figure 24: Neutral Region for Joystick Mapping

axis's (axis A and axis B region marked with a circle in Figure 24), no task is performed.

In other words the robot is in the standstill position.

Command for Neutral:

```
sl = 0;
sr = 0;
direction = 'N';
command = "+";
command += direction + "+" + sl + "+" + sr + "+";
```

4.5.3 CASE3: To Generate Commands for Forward and Reverse Motion

When the controller enters CASE3, the axis is checked, if the axis is B and button1 is pressed, it moves on to check the value of the axis. If the value resulting from the motion of the joystick is less than 32111 (Figure 24), then the duty cycle for left motor (DutytCycle1) and right motor (Duty Cycle2) are calculated as shown in Figure 23. Here the duty cycles for both the motors are the same. The calculated duty cycles along with a letter "F" to indicate that the user wants the robot to move forward is packed into a single text value and sent as a command, the commands produced here are streamed continuously over the Wi-Fi using the sockets.

If the value resulting in moving the joystick is greater than 32911 (Figure24) the controller moves to an "else" loop and the duty cycles for right motor and left motor are calculated. Again the duty cycles for both the motors are equal. The calculated values are assigned a "-" sign to indicate that the controller requires the robot to move in reverse direction. The calculated duty cycles along with the letter "D" to indicate that the controller wants the robot to move in reverse, is packed into a single text value and sent as a command.

4.5.4 CASE N

A few more cases are implemented which are not shown in the Figure 23, these cases are labeled as CASE N

4.5.4.1: Rotation

To generate a command for the robot to rotate by itself we need to have two equal duty cycles but opposite in sign. A fixed duty cycle of 25% is assigned to both the motors and this task is achieved when button 2 of the joystick is pressed. This command is represented by the letter "O" followed with dutyclyle1 and dutycycle2.

4.5.4.2: Stop Command

A stop command has a duty cycle1 =duty cycle2=0. This command is associated with a letter "S". This command is generated when button 3 of the joystick is pressed.

Similarly the code developed is very flexible for any further modification and can be expanded to have any number of cases such as U-turn from the right, U-turn from the left etc. All commands generated are transmitted to the server via the sockets, the server interprets these commands and processes the commands to perform the required task.

4.6 Command Processing at Server Side

The request/commands sent by the controller consist of three fields, one indicating the direction of motion, the second the duty cycle of the left motor and finally the duty cycle of the right motor. A buffer is created to read the raw data in the string format. Commands in the buffer are differentiated by using the symbol "+" at the start of the command. A stream of commands in a buffer is shown in Figure 25. The stream of commands are then read from the buffer and the values are processed accordingly as shown in Figure 26. First the code checks to find the starting point of the command which is identified by the symbol "+", if it finds the starting of the command, it moves to the next stage and looks for the direction and stores the value of the

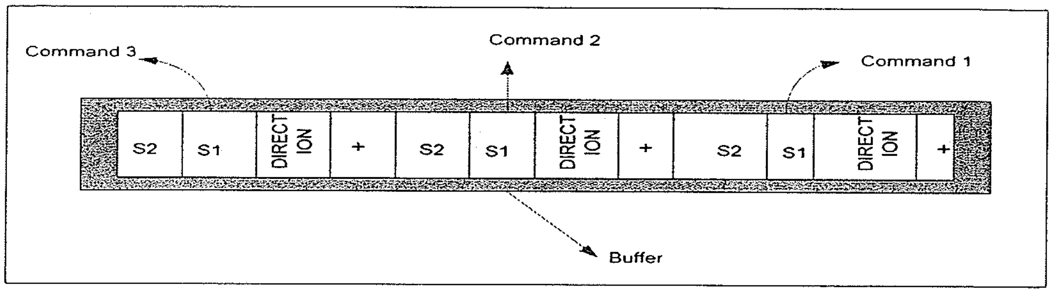
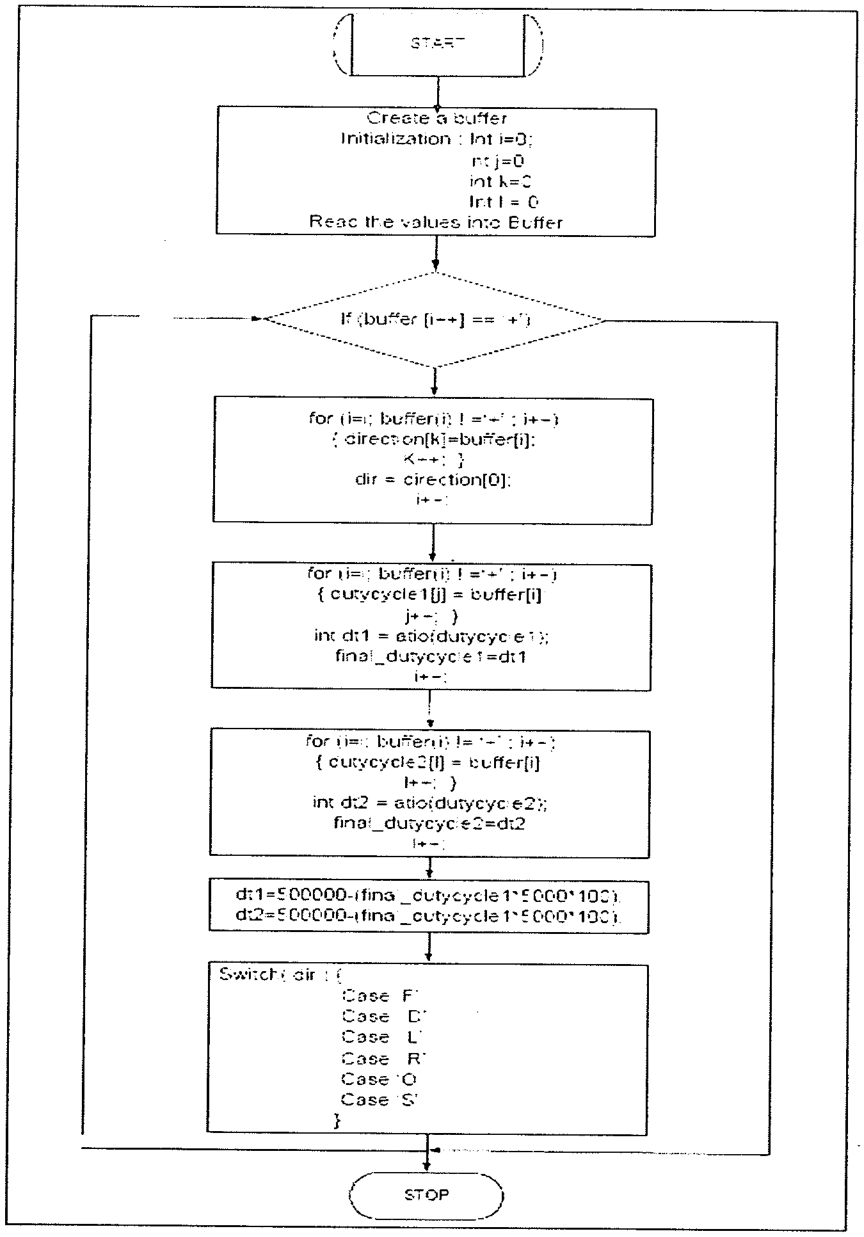


Figure 25: Command Stream in a Buffer

direction in a variable called "dir", next it checks if there is a start of new command, if not then the value of dutycycle1 is read and stored as dt1. The obtained value is then converted from string to an integer value and stored in a variable "dt1", similarly the next value read is duty cycle2 , the value read is then stored into variable "dt2". dt1and dt2 are converted with respect to the clock_divide value (Appendix A). These values are further processed according to "dir" value read using the switch and case statements.

As an example consider case "F" (Figure 26) (section 4.6.1) (to move forward). We have two registers/components Z_PWM_0 for left motor and Z_PWM_1 for right motor. The register that controls the direction of motion, forward (FORWARD_BASE) is set to 1 and reverse (REVERSE_BASE) to 0, then a check is made if the requested dutycycle1 and dutycycle2 are greater than 0, if the condition is satisfied it further checks if the dutycycle1 requested by the user is less than the clock_divide value or the counter value. If the duty cycle value is greater than the clock_divide value an error shows up. The new duty cycle is returned to the task logic (Appendix), this value is compared with the counter value and a new Z-PWM_0 is produced. Similarly Z_PWM_1 is also produced for right motor. In this case the robot is to move forward so the Z_PWM_0 and

Z_PWM_1 are equal. Similar cases have been implemented [Appendix] for left turn, right turn, reverse, U-Turn from the right, U-Turn from the left and for arbitrary degree rotation.



```
Create a buffer
Initialization : Int i=0;
                 nt j=0
                 int k=0
                 Int l = 0
Reac the values into Buffer

If (buffer [i++] == '+')

for (i=i; buffer(i) ! ='+' ; i++)
{ cirection[k]=buffer[i]:
K++; }
dir = cirection[0]:
i++;

for (i=i; buffer(i) ! ='+' ; i++)
{ cutycycle1[j] = buffer[i]
j++; }
int dt1 = atio(dutycycle1);
final_dutycycle1=dt1
i++;

for (i=i; buffer(i) != '+' ; i++)
{ cutycycle2[l] = buffer[i]
l++; }
int dt2 = atio(dutycycle2);
final_dutycycle2=dt2
l++;

dt1=500000-(final_dutycycle1*5000*100);
dt2=500000-(final_cutycycle1*5000*100);

Switch( dir ) {
        Case 'F'
        Case 'D'
        Case 'L'
        Case 'R'
        Case 'O'
        Case 'S'
      }
```

Figure 26: Command Regeneration at Server Side

4.6.1 Case 'F'

```
 IOWR (FORWARD_BASE,0, 0x01);
 IOWR (REVERSE_BASE,0, 0x00);
  if ( dt1>0 && dt2>0)
    {
     duty_cycle1 = dt1;
     duty_cycle2 = dt2;
  if (duty_cycle1 <
IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER(Z_PWM_0_BASE))
    {
    return_code = altera_avalon_pwm_change_duty_cycle(Z_PWM_0_BASE,
                                          duty_cycle1);
    check_return_code(Z_PWM_0_BASE, return_code);
    IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER(Z_PWM_0_BASE) );
    }
  if (duty_cycle2<
IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER(Z_PWM_1_BASE))
   {
   return_code = altera_avalon_pwm_change_duty_cycle(Z_PWM_1_BASE,
                                         duty_cycle1);
   check_return_code(Z_PWM_1_BASE, return_code);

   IORD_ALTERA_AVALON_PWM_CLOCK_DIVIDER(Z_PWM_1_BASE) );

   }}
```

Chapter 4 Summary

In this chapter the major functionalities of the GUI were outlined. This chapter described how the joystick interface was developed to map the movement of joystick into commands. A explanation of client-server programming, command generation, encoding and decoding of commands was also presented.

CHAPTER 5

ADVANCED IMPLEMENTATION

Chapter Overview

This Chapter outlines the development of video feedback, GPS implementation and an obstacle avoidance algorithm using sonar sensors. Parts of this chapter were done in collaboration with Monir Khan. Specific task responsibilities are noted in the footnotes.

5.1 Video

Video for robotics is an essential component to achieve a reliable telecontrolled robot. For this project the video is captured using a Logitech quickcam (ID: 046d: 092c) webcam located at the robot side. Only the front looking view of the robot is captured. Even though using a USB web camera might not be fast enough for critical real time applications, it is sufficient for the near real time application being used here. The webcam is connected to Soekris Net5501 a single computer board running a Linux Operating System which has a USB port. The kernel is custom configured for minimum memory usage and maximum processor throughput. [WWKC] briefly explains how a custom Linux kernel is built complied and installed. Net5501 can communicate with the external world using a Wi-Fi network

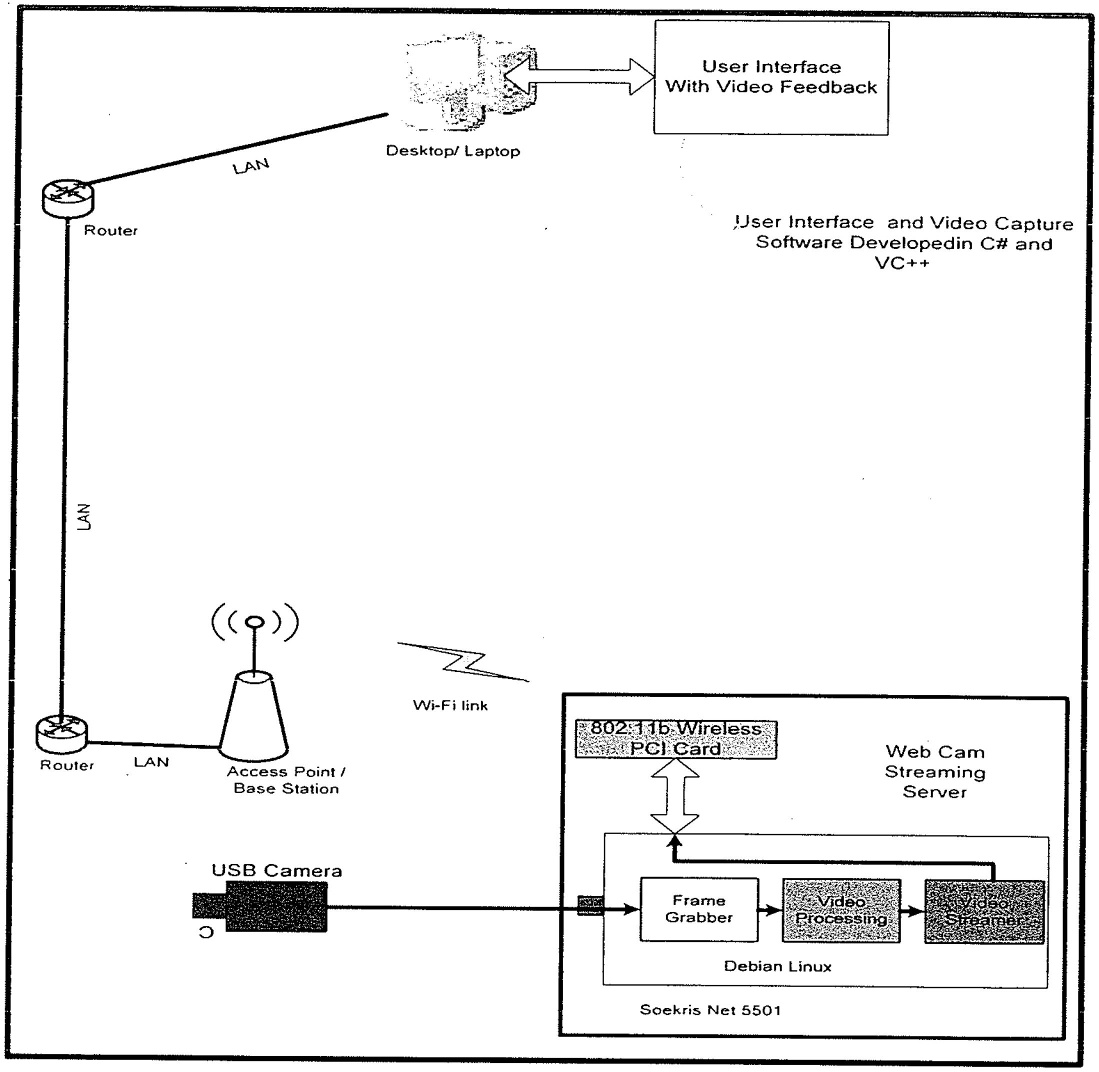

User Interface With Video Feedback
Desktop/ Laptop
LAN
Router
User Interface and Video Capture Software Developedin C# and VC++
LAN
Wi-Fi link
802.11b Wireless PCI Card
Web Cam Streaming Server
Router
LAN
Access Point / Base Station
USB Camera
Frame Grabber
Video Processing
Video Streamer
Debian Linux
Soekris Net 5501

Figure 27: Live Video Streaming

The main components in deploying a video feedback with some image processing are as follows:

- Webcam Frame Grabber
- Video Processing and Video Streaming Server
- Video Capture and Display at the Controller End.

5.1.1. Frame Grabber

A device that captures and stores a complete video frame is defined as a frame grabber. If the input from the video device is analog then a frame grabber converts the analog video signal into a digital video signal. The Logitech webcam used is capable of capturing images up to 640 x 480 pixels. A GSPCA [GSPC] driver is used to capture images from the webcam. The driver has been compiled and added to the kernel. [WWVC] explains how to compile the driver code and load the module into the Linux Kernel. This driver works as a frame grabber capable of capturing 17frame/sec. Due to the limitation in the USB webcam used, the frame rate is kept between 15- 17 frames/sec. Higher frame rates can be archived using the CMOS or high speed IP cameras. When the driver has been installed, a video device video0 is created (in /dev/video0). Video0 acts as a buffer which sends out the frames that are captured.

5.1.2. Video Streaming Server

Soekris Net 5501 is a video server which serves as a bridge between the webcam and the Internet. The frame grabber provides a continuous stream of images which have to be processed and converted into a video stream. The frames captured by the frame grabber are played at a certain speed (minimum 11frames/sec) in order to view these pictures as video. This video stream is processed and broadcast over the network in real-time. VLS (Video LAN Stream) is used to process the frames captured by video0 device and convert it to a moving picture format i.e. MJPEG format. MJPEG streams can be broadcast over Internet either in uni-cast or multicast mode. MPEG stream is set to unicast mode when the intent is to send the video feedback to only one end user (controller). If there are N end users then the video stream is set to a multicast stream. Unicast streams provide some degree of security features because only one end user can have video feedback (without some degree of effort). In the case of a video stream, a loss of few packets does not hinder the overall performance. What is needed is for the packets to be delivered on time hence RTP (Real Time Transport Protocol) protocol is used for communication. RTP provides end-to-end delivery service for real-time applications. RTP packet structure is shown in Figure 28.

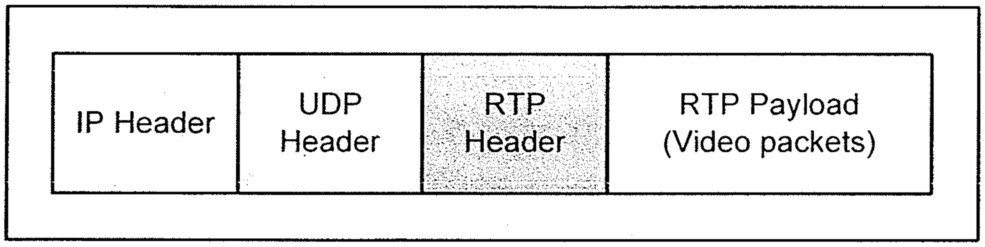


Figure 28: RTP/UDP Header

In this case RTP uses UDP as a transport protocol and UDP packets are encapsulated within the IP packets for transfer over an IP network. RTP headers contain the information related to the payload such as the source size, encoding type etc. The captured MJPEG is streamed over the networking using UDP sockets.

5.1.3. Video Capture and Display at the Controller End

A UDP socket at the controller end is listening to the video server and waiting to receive video streams. Even though the captured video stream can be displayed in a web browser [MJ01] which can be achieved by writing a simple HTML code, a Graphical User Interface (GUI) provides more flexibility and reliability if modified for future developments. VLC player's dlls (Dynamic Link libraries) are used to access all the required functions of VLC media player and develop a video capture interface. The interface developed here uses UDP sockets through which the RTP packets are received with time stamps. The packets are reframed by the video capture interface and displayed as MJPEG streams. At this time the video display within the GUI is not operational and we have had to resort to a web browser for the display. The following is the glimpse of how a C# code should be written to output video on the controller screen.

```
public Control VideoOutput
    {
      get { return m_wndOutput; }
      set
      {
        // clear old window
        if (m_wndOutput != null)
        {
          m_wndOutput.Resize -= new EventHandler(wndOutput_Resize);
          m_wndOutput = null;
          if (m_iVlcHandle != -1)
            SetVariable("drawable", 0);
        }

        // set new
        m_wndOutput = value;
        if (m_wndOutput != null)
        {
          if (m_iVlcHandle != -1)
            SetVariable("drawable", m_wndOutput.Handle.ToInt32());
          m_wndOutput.Resize += new EventHandler(wndOutput_Resize);
          wndOutput_Resize(null, null);
        }
      }
    }
```

5.2. Sonar Sensor4

SRF04 sonar sensors from Devantech are used to implement obstacle avoidance and intelligent navigation using obstacle avoidance control logic. A SRF04 emits a short burst of sounds and listens for the echo to detect if any objects are present.

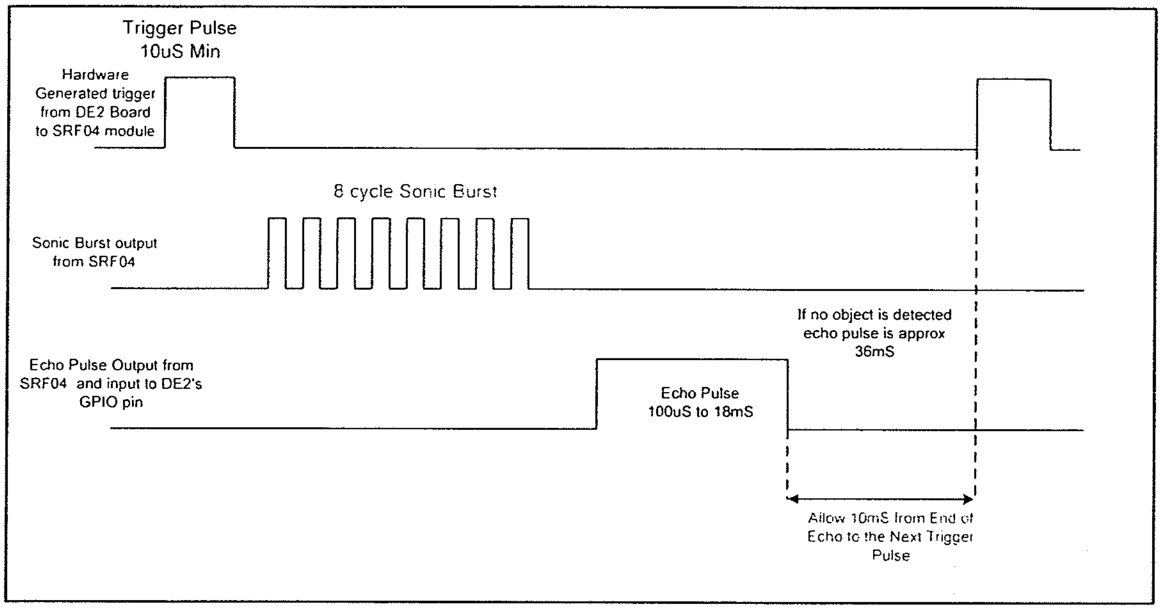

Trigger Pulse
10uS Min
Hardware Generated trigger from DE2 Board to SRF04 module
8 cycle Sonic Burst
Sonic Burst output from SRF04
If no object is detected echo pulse is approx 36mS
Echo Pulse Output from SRF04 and input to DE2's GPIO pin
Echo Pulse 100uS to 18mS
Allow 10mS from End of Echo to the Next Trigger Pulse

Figure 29: Sonar Sensor Timing Diagram

A trigger input to the sonar is generated using HDL implemented on DE2 Development board. One of the 40 GPIO pins from the Altera DE2 board is used to output the generated pulse. The trigger pulse generated is required to be at least be 10 microseconds in duration. At every trigger pulse the sonar sensor emits the sonic burst which consists of 8 cycle sonic burst shown in Figure 29 [WWSNR]. This pulse travels at a speed of sound 1.125feet per millisecond, hits a object and bounces back if it is reflected by an obstacle in its path or otherwise just diminishes. The distance to the object is measured using the

4 With the specific task of the Sonar, M. Khan and I worked in developing the required hardware and software components collaboratively.

time between the transmission of the sonic burst and the echo pulse. The echo pulse outputs a high going pulse that corresponds to the time required for the echo to return. The echo pulse is given as input to the DE2 through the GPIO pin. Using the time of echo pulse, the distance to the obstacle is calculated. Also the trigger pulse must wait at least 10 milliseconds from the time the echo pulse is returned or at least 36 millisecond if no echo is returned before being re-triggered.

5.2.1 Distance from the Obstacle

The echo pulse is read and stored into one of the DE2 board registers, the register acts as a buffer; there is a continuous inflow of data whenever it approaches obstacles. The value is read in units of time (in microseconds), this value is converted into distance of the obstacle from the present position of the robot either in inches as follows:

X- Output of the sonar sensor in microseconds.

D – Distance of the robot from Obstacle.

Also, a pulse width of 74 microseconds = 1inch from the obstacle.

Therefore we have

$$D = \frac{X}{74} \text{inches} \qquad \text{---Equation 10}$$

In real time, not every individual value read is passed on for further processing, instead a average of 5 values is passed on to the obstacle avoidance controller for a smooth and stable operation.

Note: There is a possibility of +/- 5% error from the calculated value (74) in Equation 10.

In the implementation here, 2 SRF04 sonar sensors were used at the front of the robot and 1 in at the rear of the robot.

5.3. Global Positioning System (GPS)5

GPS is a Global navigation satellite system. The GPS receiver uses at least 3 satellites of the 24 medium earth orbit satellites that transmit precise microwave signals to determine location, direction, speed and time of where the GPS receiver is placed. A mini-PCI GPS card MP-954GPD is used here. This PCI card is plugged into the Soekris board which runs Linux. When customizing the kernel for the GPS application, a driver shipped along with the card is configured into the kernel. Figure 30 shows detailed schematic on how the PCI GPS receiver is used. The Soekris board which connects the remote robot to the Internet is connected to the GPS receiver. The GPS receiver calculates all the required data using the information provided by 3 or more satellites. The GPS receiver is attached to an antenna which is tuned to the frequency transmitted by the satellite, the receiver processes the signal using a highly stable clock produced by a crystal oscillator. The processed signals are sent to the driver installed for MP954 PCI GPS, which provide all the information to the user. MP954GPS is capable of using up to 12 satellites. This information is streamed over the wireless network using sockets

5 With the specific task of the GPS, M. Khan and I collaborated on the implementation.

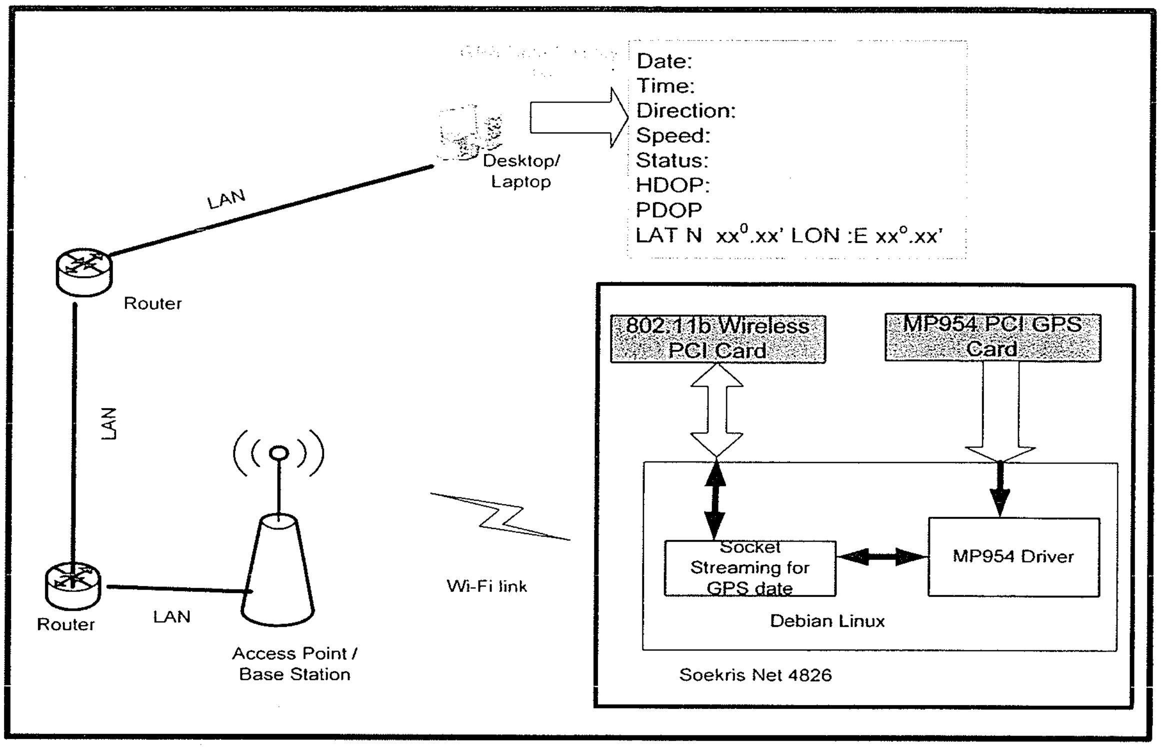


Figure 30: Implementation of GPS Receiver

At the controller side, a TCP/IP socket receives the data and differentiates between all the different forms of data (such as date, time etc) and displays it on the screen. The GPS receiver MP954GPS provides the data, time, direction, speed, and horizontal dilution of precision (HDOP), positional dilution of precision (PDOP), latitude, longitude and also satellite status. The HDOP and GDOP describe the geometric strength of satellite configuration on GPS accuracy. Using this data we can locate the position of the remote robot to within +/- 3 feet.

5.4 Implementation of Obstacle Avoidance

Navigation using sonar sensors is achieved by implementing obstacle avoidance control logic locally on the remote robot. Three SRF04 sonar sensors are used in the implementation of semi autonomous robot navigation. Figure 31 shows the implementation of obstacle avoidance control logic in NiosII processor.

When an obstacle is detected in the path of the robot, an echo pulse is generated and the distance is calculated as shown in equation 10. The value of distance calculated, which is in inches, is sent to the obstacle avoidance controller for any further action on how it should generate a PWM pulse for the motion of the robot. If no obstacle is detected the robot controller listens to commands sent by the remote operator or controller. In case an obstacle is detected the algorithm shown in the flow chart (Figure 32) is implemented.

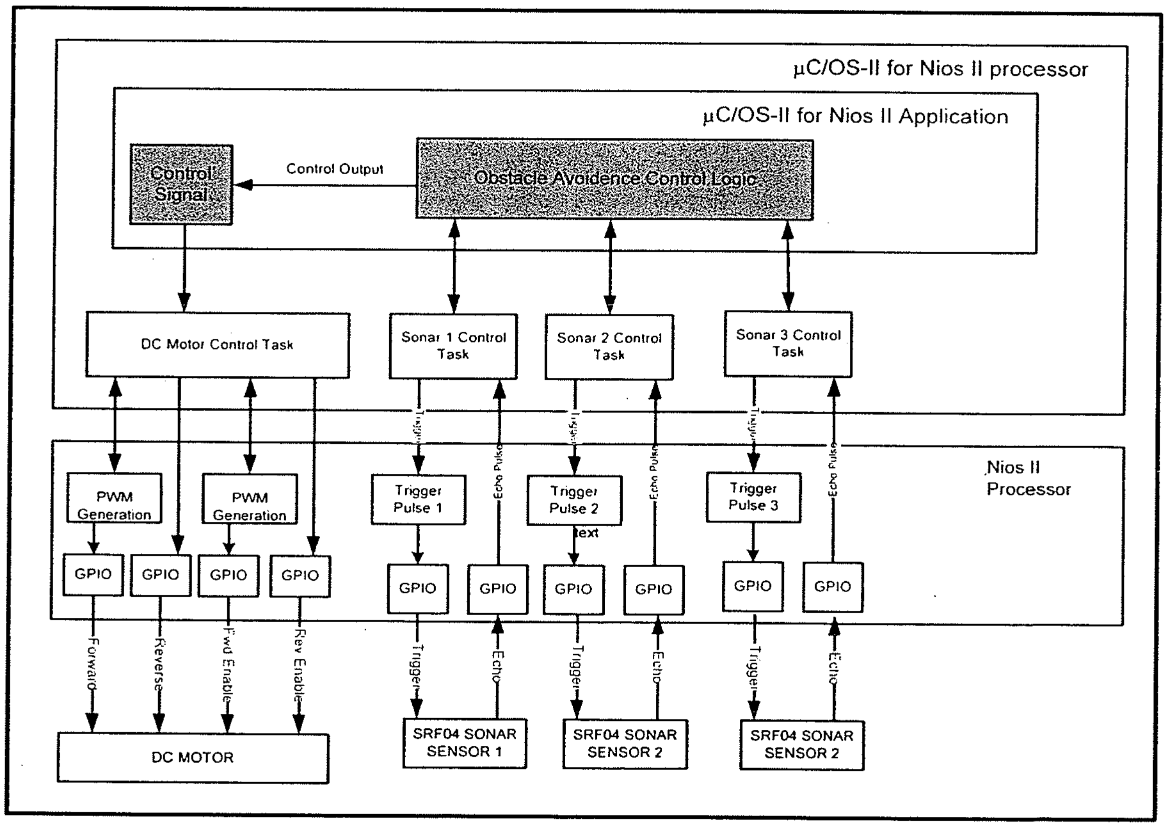


Figure 31: Implementation of Obstacle Avoidance Control Logic in the Nios II

Implementation of the obstacle avoidance algorithm is written in C programming using the Nios II IDE programming environment. The code uses case statements as shown in the flow chart. Different modules or cases are used for different sensors. Every sensor data is labeled so as to identify which senor has detected an object in its path of motion. The control logic enters case D1 whenever a sensor D1 located on the front towards the right side of the robot detected an object in its path of motion. The control logic enters case D2 whenever a sensor D2 located on the front towards the left side of the robot detected object in its path of motion. The control logic enters Case D3 whenever a sensor D3 located in the center of robot rear side detected object in its path of motion.

When sensors D1 and sensor D2 detect an object in their path of motion then case D1 "&&" D2 send the control signal to the robot.

5.4.1. Case D1: Obstacle Detected by Right Sensor on the Front of the Robot

When an obstacle is detected in the front on the right side of the robot, the robot should react in such a way that it would avoid the obstacle and move toward the left. Before processing the data from the sonar sensors to send out a command to the motor controller, it first checks if the motion is forward, if false it moves on to the next case statement. To make the robot move towards the left, it is necessary that the duty cycle (directly proportional to speed) of the left wheel should be less than the duty cycle of the right wheel. If the distance of the obstacle from the robot is less than or equal to 10 inches (Critical Distance) then the robot halts irrespective of the values given by the other sensors. If the distance to the obstacle for sensor D1 from the robot is less than or equal to 40 inches then the duty cycle set by the obstacle avoidance algorithm for the left motor is 25% of the actually duty cycle given by the controller and consequently the duty cycle of the right motor is 50% of the actually duty cycle given by the controller. A check is made to determine if the distance to the obstacle is still less than the critical distance, if it approaches the critical distance then the robot halts and overrides the user commands completely. Similar actions are performed in the case of obstacles detected at distances less than or equal to 80 inches by the robot's local intelligence. When an obstacle is detected between 40 and 80 inches away from the robot, the modification to the speeds of right and left wheel of the motors are higher. When the distance of the obstacle is greater than 100 inches then, the robot just obeys the commands sent by the controller.

5.4.2. Case D2: Obstacle Detected by Left Sensor on the Front of the Robot

When obstacle is detected in the front on the left side of the robot, the robot should react in such a way that it would avoid the obstacle and move towards the right. Before processing the data from the sonar sensors to send out commands to the motors, it first checks if the motion is forward, if false it moves on to the next case statement. To make the robot move towards the right, it is necessary that the duty cycle of the right wheel should be less than the duty cycle of the left wheel. If the distance of the obstacle from the robot is less than or equal to 10 inches (Critical Distance) then the robot halts irrespective of the values given by the other sensors If the distance of the obstacle for sensor D2 from the robot is less than or equal to 40 inches then the duty cycle set by the obstacle avoidance algorithm for the right motor is 25% of the actually duty cycle given by the controller and consequently the duty cycle of the left motor is 50% of actually duty cycle given by the controller.

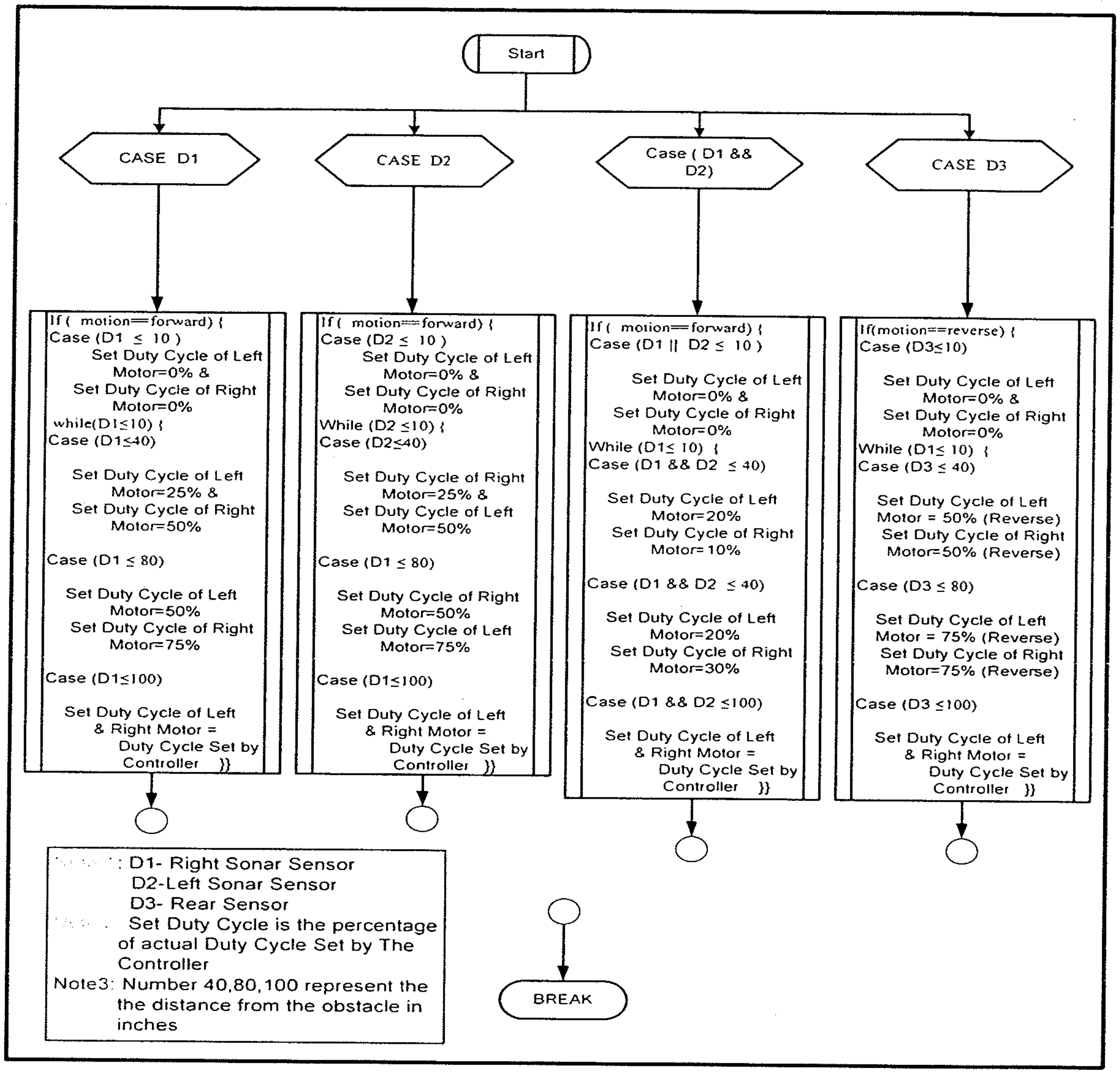

Start
CASE D1
CASE D2
Case (D1 && D2)
CASE D3
If(motion==forward) { Case (D1 ≤ 10) Set Duty Cycle of Left Motor=0% & Set Duty Cycle of Right Motor=0% while(D1≤10) { Case (D1≤40) Set Duty Cycle of Left Motor=25% & Set Duty Cycle of Right Motor=50% Case (D1 ≤ 80) Set Duty Cycle of Left Motor=50% Set Duty Cycle of Right Motor=75% Case (D1≤100) Set Duty Cycle of Left & Right Motor = Duty Cycle Set by Controller }}
If(motion==forward) { Case (D2 ≤ 10) Set Duty Cycle of Left Motor=0% & Set Duty Cycle of Right Motor=0% While (D2 ≤10) { Case (D2≤40) Set Duty Cycle of Right Motor=25% & Set Duty Cycle of Left Motor=50% Case (D1 ≤ 80) Set Duty Cycle of Right Motor=50% Set Duty Cycle of Left Motor=75% Case (D1≤100) Set Duty Cycle of Left & Right Motor = Duty Cycle Set by Controller }}
If(motion==forward) { Case (D1 || D2 ≤ 10) Set Duty Cycle of Left Motor=0% & Set Duty Cycle of Right Motor=0% While (D1≤ 10) { Case (D1 && D2 ≤ 40) Set Duty Cycle of Left Motor=20% Set Duty Cycle of Right Motor= 10% Case (D1 && D2 ≤ 40) Set Duty Cycle of Left Motor=20% Set Duty Cycle of Right Motor=30% Case (D1 && D2 ≤100) Set Duty Cycle of Left & Right Motor = Duty Cycle Set by Controller }}
If(motion==reverse) { Case (D3≤10) Set Duty Cycle of Left Motor=0% & Set Duty Cycle of Right Motor=0% While (D1≤ 10) { Case (D3 ≤ 40) Set Duty Cycle of Left Motor = 50% (Reverse) Set Duty Cycle of Right Motor=50% (Reverse) Case (D3 ≤ 80) Set Duty Cycle of Left Motor = 75% (Reverse) Set Duty Cycle of Right Motor=75% (Reverse) Case (D3 ≤100) Set Duty Cycle of Left & Right Motor = Duty Cycle Set by Controller }}
: D1- Right Sonar Sensor
D2-Left Sonar Sensor
D3- Rear Sensor
Set Duty Cycle is the percentage of actual Duty Cycle Set by The Controller
Note3: Number 40,80,100 represent the the distance from the obstacle in inches
BREAK

Figure 32: Obstacle Avoidance Algorithm

The algorithm checks again if the distance to the obstacle is still less than the critical distance, if it approaches critical distance then the robot halts and overrides the user commands. Similar actions are performed such as in the case of obstacles less than or equal to 80 inches from the robot except that the modification to the speeds of right and left wheel of the motor are higher. When the distance of the obstacle is greater than 100 inches then, the robot just obeys the commands sent by the operator or controller.

5.4.3. Case D3: Obstacle Detected by the Rear Sonar Sensor

When an obstacle is detected by SRF04 sensor on the rear end of the robot, the distance from the obstacle along with a tag D3 is sent to the obstacle avoidance algorithm. First it checks if the direction of motion is reverse and only then starts the processing on how to generate the commands for the motor controller. When the distance between the robot and obstacle is less than 10 inches, the controller overrides the user commands and sends a control signal to generate 0% duty cycle on both the wheels. When the distance between the robot and the obstacle is less than 40 inches then, a control signal to produce a duty cycle which is 50% of the actually duty cycle requested by the controller. The algorithm also checks if the distance of the robot to the obstacle is in critical distance. If it approaches the critical distance then the robot halts. Similarly when the distance of the robot from the obstacle is less than 80 inches then the duty cycle is set to 75% of the actually duty cycle requested by the user. Finally when there is no obstacle found in its path of motion the robot just follows the control sent by the user.

5.4.4 Case D1 and D2: Obstacle Detected by Both the Front Sensors

When an obstacle is detected by both the sensors in the front, the obstacle control logic has to decide which way it has to move either to its right, left or stop. By default if the obstacle is detected by both the sensors the robot moves towards its right. When both the sensors locate an obstacle within its critical distance then the robot stops and overwrites all the commands of the operator or controller. When the obstacle detected is in the range of 10 inches to 40 inches, a signal is sent to the motor controller to move towards its right. Duty cycles of 10% and 20% are set for right and left wheels respectively. The command to move towards right is executed 5 times. If both the sensors still finds an obstacle, a command to move left with a duty cycle of 20% on the right and 10% on the left is executed 10 times. If there still exists an object detected by both the sensors then a stop command is given to the robot (not shown in Figure5). Similar actions are performed when an obstacle is detected in a range of 40-80 inches by both the sensors, except that they have a dutycycle of 20% for the right wheel and 30% for the left wheel. When moving towards right they have a dutycycle of 20% for the right wheel and 30% for the left wheel.

Summary of Chapter 5

Chapter 5 described the overall development of video feedback, and outlined the working principles of GPS and sonar. Also explained is a detailed obstacle avoidance algorithm for the mobile robotic platform developed by the author. The obstacle avoidance control subsystem was motivated by related research in fuzzy control. The algorithm developed here is considerably more heuristic driven but operates along similar lines of reasoning.

CHAPTER 6

RESULTS AND DISCUSSIONS

Chapter Overview

This Chapter discusses results in controlling the telerobotics platform over a Wi-Fi network.

Parts of this chapter were done in collaboration with Monir Khan. Specifically we collaborated on the basic test that were performed and the protocol for their evaluation. Although there are a number of more formal methods of evaluation we were more interested in basic function and ease of operation.

6.1 GUI:

The interface developed is shown in Figure 20. This interface runs without any major issues and it appears to be reliable as far as we were able to validate from the experiments performed. Moving the joystick caused the sliders Axis 1, Axis2 and Axis3 to represent the joystick motion. Using the host IP address and port number, this interface was able to communicate with the remote telerobotic platform (or server) and sends commands through a designated socket. The interface was able to display the sonar sensors data provided by the telerobotics platform. Initially platform independent GUI was developed using WxWidgets, but due to the many limitations and bugs in WxWidgets the final version had to be developed in the .Net environment. The GUI can be easily extended to incorporate additional sensors.

6.2 Basic Functionalities of Telerobotic Platform:

The basic functionalities of telerobotics platform include the ability to move the robot forward, reverse, stop, turn left, turn right, rotate, turn right by 90 degrees and turn left by 90 degrees. All these functionalities were tested in the lab and the entire prototype platform was functional within a local wireless network. The operator used a joystick to guide the robot. The following table provides the detailed list of operations performed:

Direction Of Motion	Result	Notes
Forward	Successful	-
Reverse	Successful	-
Turn Left	Successful	Turning angle not precise
Turn Left	Successful	Turning angle not precise
360^0 rotation	Successful	-
90^0 Left Turn	Successful	-
90^0 Right Turn	Successful	-
Stop	Successful	-

Table 1: Basic Functionality results for Telerobotics Platform

6.3 Video Feedback

The video feedback from the robot is displayed in a web browser. The final frame rate was approximately 7-9 frames/sec. The web browser can be either Mozilla or Internet Explorer, both in theory refresh with a 0 second delay between the frames. The delay was well below 1 second under test conditions (No simulator or tools were used to perform delay analysis, it was by observation only). Robot control using the joystick was possible using video feedback. Under test condition the command delay was negligible. Every frame was associated with a time stamp on it which helped us to roughly determine the delay in the network, the time a frame is captured using the USB camera and the time the frame is displayed at the controller end is almost identical (less than 1 second).

6.3.1 Video Feedback Using CMOS Camera

The initial idea for video feedback was to use a CMOS camera along with the DE2 board which would have enabled us to obtain a high resolution picture. The images could be captured and processed using the FPGA, but could not be streamed over network due to memory limitations. The high resolution video could be displayed on a VGA monitor, successfully demonstrating the capture and image processing on the robot. The difficulty in video streaming was due to the limitations of the Micro C/ OSII and the frame buffer of the CMOS camera which used the same memory component i.e. SDRAM of DE2 board.

6.4 Sonar Sensor

Sonar sensors successfully detect objects and sent out echo pulses to the DE2 board. The time period of the echo pulse represents the distances of the robot from obstacle. Even though the distance to an obstacle obtained by sonar is not as accurate as the laser sensor detectors, the sonar sensors are sufficient for the application here.

6.5 Obstacle Avoidance Algorithm

The obstacle avoidance algorithm was designed successfully. The algorithm could read the distance values and enter into the appropriate case statements. When the sensor detected an obstacle in the front or in the rear the algorithm worked partially with some degree of accuracy. An obvious modification at this point would have been to attempt to introduce a learning algorithm to tune the heuristic based control of the initial implementation.

6.6 Cost Efficient Design

The robot platform in this thesis is designed with the minimum number of available resources. The components, boards, sensors, motor controllers etc. inclusive of the robot body cost less than $ 1300 CAN. A completely functional system can be expanded to cover a wider range of more critical life saving applications. These could include extending the platform for land mine detection, search and rescue, etc. without substantively larger costs. The main point here is that with much of the technology that was recently cost prohibitive telecontrol of robots is now within feasible limits.

6.7 Summary of Chapter 6

In this chapter the test results while operating telerobotic platform where outlined. The tests conducted were undertaken over a local wireless network. Factors associated with delay were considered negligible although further tests are required to more completely exercise the prototype. The wireless LAN was a private network albeit co-located with a number of other 802.11 networks. In a more typical deployment one would like to use the existing 802.11 infrastructure and services in which case a greater degree of unpredictable behavior can be expected. Some test results for obstacle avoidance were also outlined and finally we concluded that robot development of this type is cost effective.

CHAPTER 7

CONCLUSIONS and FUTURE WORK

7.1 Conclusions

For this thesis, an operated assisted telerobotic platform was developed. This platform used Wi-Fi as the medium of communication. The platform can perform all the basic operations of motion that were designed for and we successfully demonstrated that the methodology lends itself to the inclusion of additional hardware devices and sensors. The controller uses a joystick as control device to remotely maneuver the robot. Video feedback was provided to the operator to help him/her perform a given task with some degree of precision. An obstacle avoidance algorithm is one illustrative example of how feasible this design is for any future extensions.

The platform design is flexible for future modification as the platform was developed using μC/ OSII as its RTOS and uses an FPGA for the hardware components. The soft-core nature of FPGAs can be used to reconfigure or redesigned the hardware according to user requirements. These features make the robotic platform ideally suited for prototyping and closer to their industrial realizations. Even though the basic design and algorithms are complete, some parts of the design could not be realized. Results however illustrated that the design of telerobotic platform is a success with some minor exceptions. The development of the motor controllers and remote control of robot using a joystick are considered the more major contributions in this thesis along with the implementation of the video feedback. Mitigating the problems associated with delay will remain active research areas. The two most effective means, namely improved video communication and provisioning the robot with a degree of local control were addressed in this work.

7.2 Recommended Future Work

The primary objectives in this line of study will be to develop a more versatile telerobotic platform that should be reliable and as universal as possible. To achieve these objectives the following issues should be taken into consideration:

1. This thesis focuses only on a Windows OS GUI; in the future we would like to expand it to be a platform independent GUI
2. This thesis currently has a limitation of using web browser for video feedback; in the future we would like to combine the video feedback and the control system into one GUI
3. We did not consider the factors such as wheel slip, terrain uncertainty, etc. In the future we would like to consider some of these uncertainties to develop a more complete telerobotic platform.
4. In the future use we would like to use the GPS data and incorporate the data into either Google maps or Yahoo maps to exactly locate the robot location when left all by itself and have the ability to query its position from any browser.

5. Personally, I would like to co-ordinate with the people working on land mine detection and design a cost efficient wireless controlled land mine detector.

Appendix A: Task Logic

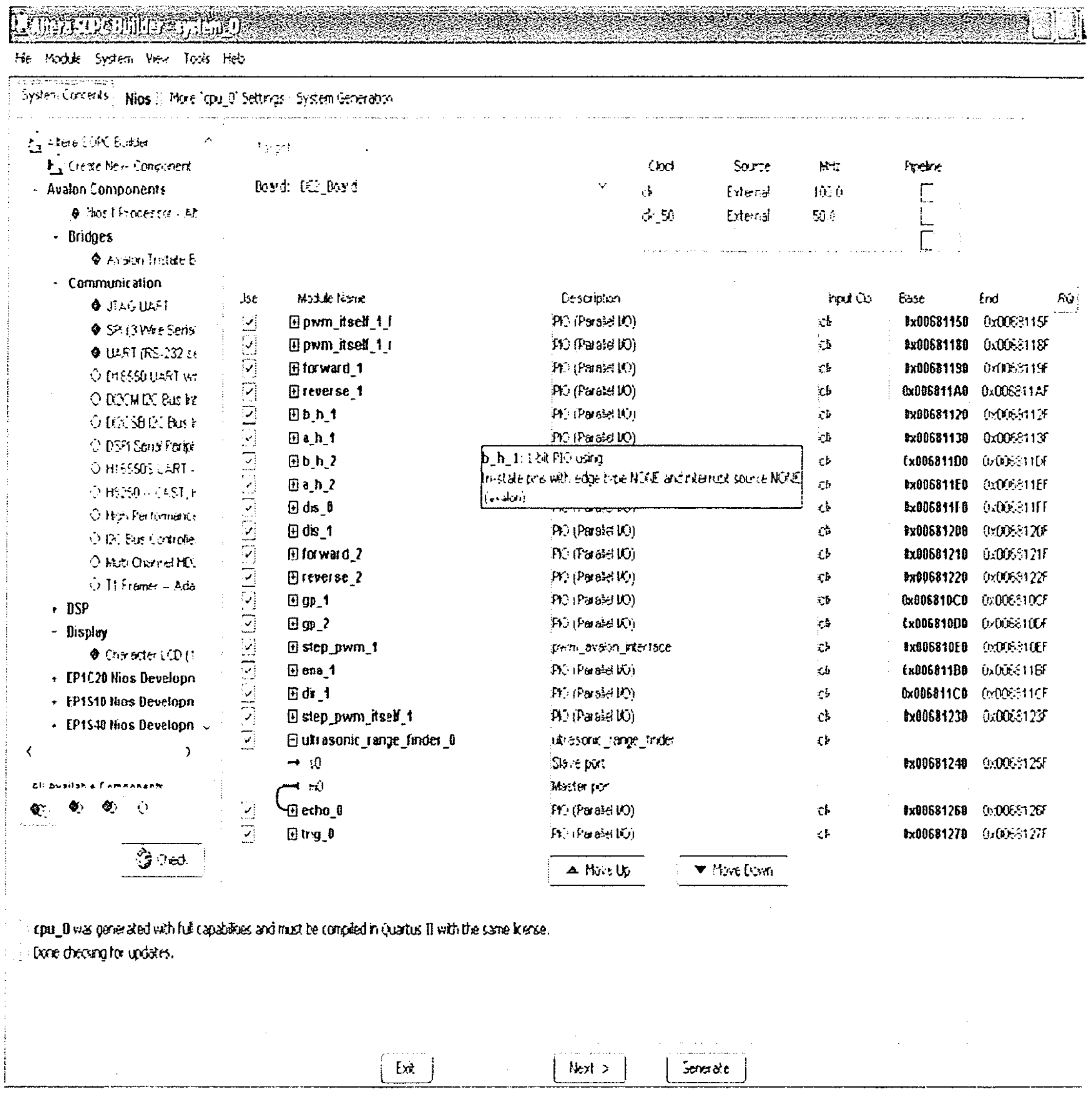
This appendix includes samples of the task logic for the PWM component.

Task Logic Code

```
always @(posedge clk or negedge resetn)       //PWM Counter Process
begin
        if (~resetn)begin
                counter <= 0;
        end
        else if(pwm_enable)begin
                if (counter >= clock_divide)begin
                        counter <= 0;
                end
                else begin
                        counter <= counter + 1;
                end
        end
        else begin
                counter <= counter;
        end
end
always @(posedge clk or negedge resetn)      //PWM Comparitor
begin
        if (~resetn)begin
                pwm_out <= 0;
        end
        else if(pwm_enable)begin
                if (counter >= duty_cycle)begin
                        pwm_out <= 1'b1;
                end
                else begin
                        if (counter == 0)
                                pwm_out <= 0;
                        else
                                pwm_out <= pwm_out;
                        end
                end
        else begin
                pwm_out <= 1'b0;
        end
```

Appendix B: Screenshots & Pictures

This appendix includes a screen shot of the SOPC builder and a picture of the platform itself.



Board: DE2_Board

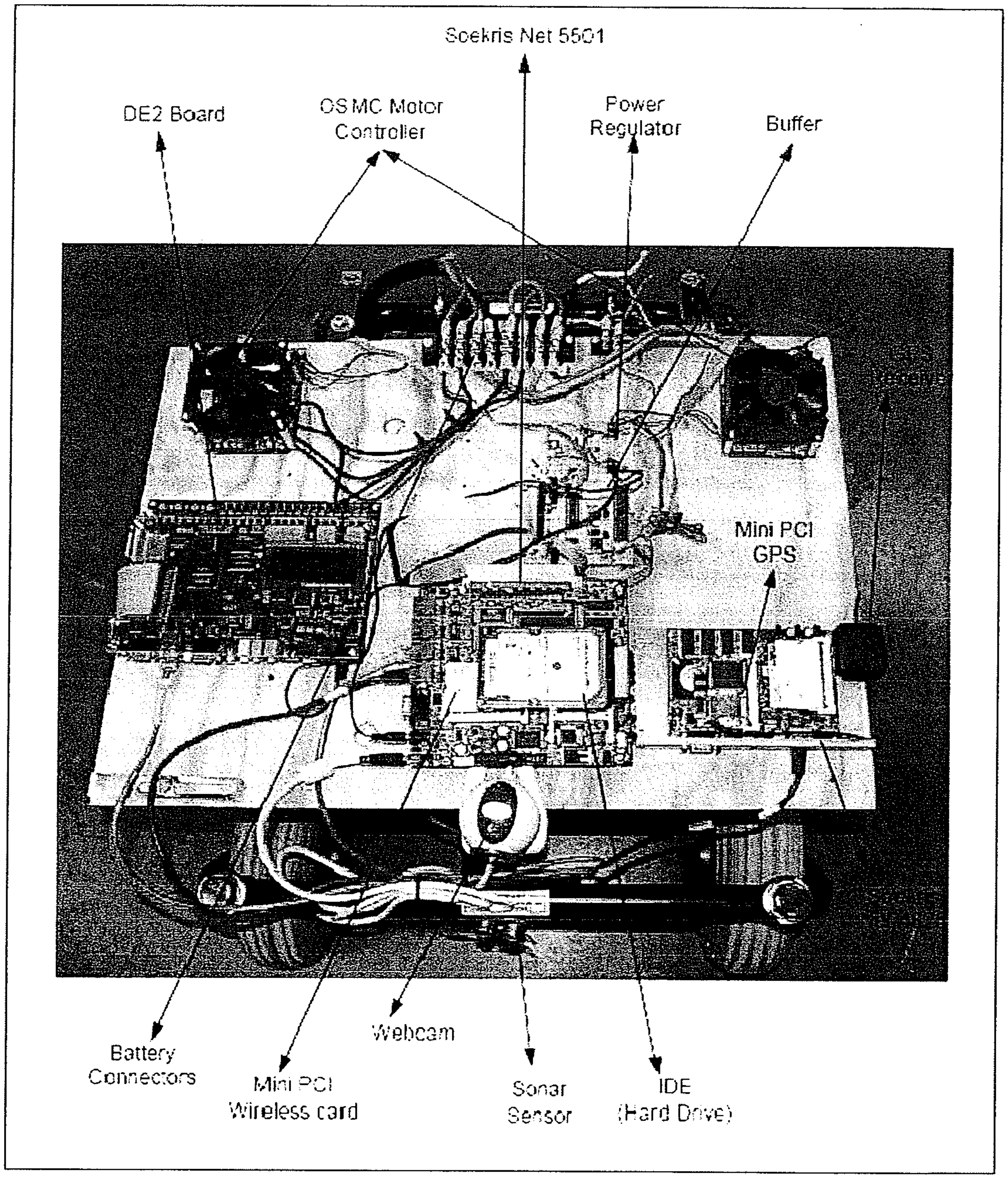
Clock	Source	MHz
clk	External	100.0
clk_50	External	50.0

Use	Module Name	Description	Input Clock	Base	End
✓	pwm_itself_1_f	PIO (Parallel I/O)	clk	0x00681150	0x0068115F
✓	pwm_itself_1_r	PIO (Parallel I/O)	clk	0x00681180	0x0068118F
✓	forward_1	PIO (Parallel I/O)	clk	0x00681190	0x0068119F
✓	reverse_1	PIO (Parallel I/O)	clk	0x006811A0	0x006811AF
✓	b_h_1	PIO (Parallel I/O)	clk	0x00681120	0x0068112F
✓	a_h_1	PIO (Parallel I/O)	clk	0x00681130	0x0068113F
✓	b_h_2		clk	0x006811D0	0x006811DF
✓	a_h_2		clk	0x006811E0	0x006811EF
✓	dis_0		clk	0x006811F0	0x006811FF
✓	dis_1	PIO (Parallel I/O)	clk	0x00681200	0x0068120F
✓	forward_2	PIO (Parallel I/O)	clk	0x00681210	0x0068121F
✓	reverse_2	PIO (Parallel I/O)	clk	0x00681220	0x0068122F
✓	gp_1	PIO (Parallel I/O)	clk	0x006810C0	0x006810CF
✓	gp_2	PIO (Parallel I/O)	clk	0x006810D0	0x006810DF
✓	step_pwm_1	pwm_avalon_interface	clk	0x006810E0	0x006810EF
✓	ena_1	PIO (Parallel I/O)	clk	0x006811B0	0x006811BF
✓	dir_1	PIO (Parallel I/O)	clk	0x006811C0	0x006811CF
✓	step_pwm_itself_1	PIO (Parallel I/O)	clk	0x00681230	0x0068123F
✓	ultrasonic_range_finder_0	ultrasonic_range_finder	clk		
	s0	Slave port		0x00681240	0x0068125F
	m0	Master port			
✓	echo_0	PIO (Parallel I/O)	clk	0x00681260	0x0068126F
✓	trig_0	PIO (Parallel I/O)	clk	0x00681270	0x0068127F

b_h_1: 1-bit PIO using tri-state pins with edge type NONE and interrupt source NONE (avalon)

cpu_0 was generated with full capabilities and must be compiled in Quartus II with the same license.
Done checking for updates.

Picture 1: Screenshot of Custom Components for Nios II processer



Picture2: Prototype of "Operator Controlled Telecontrolled Platform with Obstacle avoidance"

References:

[ALTE0] Nios II processor handbook can be found in the following website http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[ALTE1] Micro C/ OS-II real time operating system for Altera can be found in the following website: http://www.altera.com/literature/hb/nios2/n2sw_nii52008.pdf

[ALTE2] Hardware abstraction layers reference can be found in the following website: www.**altera**.com/literature/hb/nios2/n2sw_nii52003.pdf

[AS97] A. Saffotti, " The Uses of Fuzzy Logic for Autonomous Robot Navigation" Soft Computing Vol. 1, no. 4 pp. 180-197, 1997, Available on-line at http://aass.oru.se/Agora/FLAR/

[BLP03] B.J. Challacombe, L.R. Kavoussi, P. Dasgupta -"Trans-oceanic Telerobotic Surgery "BJU International Volume 92 Issue 7 Page 678- 680, November 2003

[EM94] Eddie Tunstel and Mo Jamshidi –"Embedded Fuzzy Logic Based Wall Following Behavior for Mobile Robot Navigation" NAFIPS/IFIS/NASA'94. Proceedings of the First International Joint Conference of the North American Fuzzy Information Processing Society Biannual Conference, San Antonio, TX USA December 1994

[GSPC] http://mxhaard.free.fr/download.html

[JJ03] Jahng-Hyon Park and Joonyoung Park –" Real Time Bilateral Control for Internet based Telerobotic System " Proceeding og the 2003 IEEE/RSJ , Intl Conference on Intelligent Robots and Systems, Las Vegas, Nevada-October 2003

[JJG03] A thesis by Jitendra Jaising Gaikwad " at The university of Alabama at Birimingham [Online] Available:

http://www.ece.eng.uab.edu/DCallaha/research/RoboticProtocolThesis1.pdf

[JN91] John Yen and Nathan Pfluge –" Path planning and execution using fussy logic" in Proceedings of the AIAA Conference on Guidance, Navigation, and control, Volume 3, pages 1691-1698, New Orlands, LA, August 1991

[MS05] K.Murugan and S. Shanmugavel " Traffic Dependent and Energy-Based Time Delay Routing Algorithms for Improving Energy Efficiency in Mobile Ad Hoc Networks " EURASIP Journals on Wireless Communications and Networking 2005:5, 625-634

[MSDN] http://msdn2.microsoft.com/en-us/default.aspx

[NT99] Ning Xi and T.J Tarn –" Action Synchronization and Control of Internet Based Telerobotic Systems" Proceeding of the 1999 IEEE International Conference on Robotics and Automation, Detroit, Michigan May 1999

[NTA96] Ning Xim Tzyh-Jong Tarn and Antal K.Bejczy –"Intelligent Planning and Control for Multirobot Coordination: An Event-Based Approach" IEEE Transaction on Robotics and Automation Vol12, No3, June 1996

[RCA98] Ronald C. Arkin and Michael Arbib (contributor) –"Behavior-Based Robotics" MIT press, ISB 0262011654 , 1998

[RDH97] Rainer Palm, Dimiter Driankov and Hans Hellendoorn "Model Based

Fuzzy Control: Fuzzy Gain Schedulers and Sliding Mode Fuzzy "
Springer publication, ISB 3540614710

[RGG97] Robert G. Gallager "A Minimum Delay Routing Algorithm Using Distributed Computation "IEEE Transactions and Communications, January 1997

[ROJ04] R. Le Moigne, O. Pasquier, J-P. Calvez " A Generic RTOS Model for Real-Time System Simulation with SystemC

[SHM03] Simon X. Yang, Hao Li and Max Meng –"Fuzzy Control of a Behavior-Based Mobile Robot" The IEEE International Conference on Fuzzy Systems -2003.

[SK00] Siripun Thongchai and Kazuhiko Kawamura –"Application of Fuzzy Control to a Sonar-Based Obstacle Avoidance Mobile Robot "Proceeding of the 2000 IEEE International Conference on Control Applications Anchorage, Alaska, USA, September 25-27, 2000.

[SSDN00] S. Thongchai, S. Suksakulchai, D. M. Wilkes, and N. Sarkar –" Sonar Behaviour –Based Fuzzy Control for a Mobile Robot" in Proceeding of the IEEE International Conference on Systems, Man and Cybernetics, Nashville, Tennessee, October 8-11, 2000.

[TBS92] Thomas B. Sheridan –"Telerobotics, Automation, and Human Supervisory control" MA, MIT press, Published year 1992

[TUNO] 2601050 Robotics and Teloperation Lecture Notes, Tamper University of Technology [Online], Available:

http://www.iha.tut.fi/education/IHA-3506/book/Teleoperation_Notes_2004.pdf

[VER99] Vern Paxson " End to End Internet Packet Dynamics" IEEE/ACM Transaction on Networking Volume 7 Issue 3 June 1999, pages 277-292

[WWDC] Wikipedia Pulse Width Modulation, Duty cycle [online] Available: http://en.wikipedia.org/wiki/Duty_cycle

[WWDS] Principle of Differential steering [Online] Available: http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html

[WWIE] Hobbes Time Line [Online] Available: http://www.zakon.org/robert/Internet/timeline/

[WWKC] Kernel configuration [Online] Available: http://www.ultradesic.com/index.php?section=21

[WWRG] Wikipedia Raymond Goertz [Online] Available: http://en.wikipedia.org/wiki/Raymond_Goertz

[WWRP] OSMC Motor controller project [Online] available: http://www.robotpower.com/downloads/OSMC_project_documentation_V4_21.pdf

[WWRP2] OSMC Motor controller project [Online] available: http://www.robotpower.com

[WWSNR] The SRF 04 Sonar Sensor [online] Available: http://www.acroname.com/robotics/parts/R93-SRF04.html

[WWTS] DE2 board samples and manuals [Online] Available:

http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=39&No=30

[WWVC] Webcam kernel compilation instructions [Online] Available: http://gentoo-wiki.com/HOWTO_logitech_quickcam_on_2.6.x_kernel

[WWVDC] Introduction to Pulse Width Modulation [Online] Available: http://www.netrino.com/Publications/Glossary/PWM.php

[WWVU] Helpmate telecontrolled robot [Online] Available: http://eecs.vanderbilt.edu/CIS/IRL/helpmate.shtml

[WWWF] Wikipedia Introduction to Wi-Fi [Online] Available: http://en.wikipedia.org/wiki/WiFi

[WWW1] Basics concepts of RTOS [Online] Available: http://linuxdevices.com/articles/AT4627965573.html

[WWW2] Real time Operating Systems Wikipedia [Online] Available: http://en.wikipedia.org/wiki/Real-time_operating_system