To the Graduate Council:

I am submitting herewith a thesis written by Arun Srikantaiah entitled "PC-Based Control of the Andros Mark VI Hazardous Duty Mobile Robot". I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

Reid L. Kress

Reid L. Kress, Major Professor

We have read this thesis and recommend its acceptance:

William R. Hamel, Professor

Tyler A. Kress, Professor

Acceptance for the Council:

Vice Provost and Dean of Graduate Studies.

PC-BASED CONTROL OF THE ANDROS MARK VI HAZARDOUS DUTY MOBILE ROBOT

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Arun Srikantaiah
May 2002

Acknowledgements

I would like to express my profound gratitude to my major advisor, Dr. Reid Kress, for his invaluable guidance, precious technical advice and continued support throughout the writing of this thesis. I am extremely thankful to my committee: Dr. William Hamel and Dr. Tyler Kress for their suggestions and guidance that went a long way in making the writing of this thesis a success.

I would also like to thank Dr. Hal Aldridge and Mr. Steve Cliff of Remotec Inc. for their invaluable support in providing the drawings and details about the Andros Mark - VI robot without which this thesis would not have been possible. Special thanks go to Dr. Lynne Parker of the Oak Ridge National Laboratory for providing us with the Andros Mark - VI robot for research and development. I would also like to thank Mr. Dan Bowen of the Department of Computer Science for his valuable assistance with the computer code.

Last but not the least I would like to thank my entire family for their support and encouragement during the writing of this thesis.

Abstract

Teleoperation refers to the remote control of a robotic manipulator or a vehicle. A telerobot is a device that extends a person's sensing or manipulating ability to a remote location that is controlled by a computer and supervised by a human operator. This thesis describes the development of a PC- based control architecture for the Andros Mark VI hazardous duty mobile robot. The Andros is widely used in a wide variety of hazardous tasks by law enforcement agencies throughout the world. These include explosives handling SWAT operations, HazMat response and nuclear surveillance / maintenance. In this context the present control scheme was found to be inadequate with reference to operator friendliness which greatly affects the operation of the robot. PC based control was incorporated to the robot using RS-232 and serial communications. A user friendly interface was built using the Visual Basic programming language. This new interface duplicates all the existing functions found on the current control box thus giving way to a new and improved computer interface. The thesis also briefly examines the suitability of the Sequential Modular Architecture for Robotic Teleoperation (SMART) to the control of the Andros Robot and discusses future applications like web-based control.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and previous work done

A robot manipulator can be described as an open-loop link mechanism consisting of several links connected together by joints **[Yoshikawa, 90]**. Typical joints could be revolute joints or prismatic joints. In this context the control of the robot assumes prime importance since it involves the control of these links and joints. Sciavicco and Siciliano **[2000]** describe control as an issue directly related to the tasks assigned to the manipulator. The fundamental elements of tasks performed by robot manipulators are (1) to move the end effector, with or without a load, along a desired trajectory and (2) to exert a desired force on an object in contact with the end effector. For a robot with feedback heteroceptive sensors control can be either position control or force control or a combination of both called hybrid control. Heteroceptive sensors are sensors that provide the robot with knowledge of the surrounding environment. For a robot which lacks any heteroceptive sensors, like the Andros Mark VI, control takes on a whole new meaning since the only feedback that the operator gets is through sensory aids such as cameras, for video feedback and, microphones for audio feedback.

The Andros is a completely teleoperated hazardous duty mobile robot manufactured by Remotec Inc. **[Remotec, 01]**. The Andros Mark VI features a unique articulated track

chassis which allows it to operate over rough terrain, cross obstacles and ditches, climb stairs and operate in sand, gravel, mud and grass. The Andros is similar to a variety of other teleoperated robots like the Brokk **[2002],** which is used in a variety of heavy duty construction tasks, and the Cybermotion **[2001],** which is used in nuclear facilities all over the US, in that it uses low level teleoperability for control. All these robots have limited sensory capabilities and computing power.

A very innovative development in the past three decades has been the computerized control of robot manipulators. The need to reduce operating times for high risk operations as well as the hazards to personnel led to the development of remotely controlled mobile robots. Meieran and Mosci **[1991a]** outline the difference in the design of such systems from traditional robotics in that the robots in these systems must meet specific requirements and operate in unstructured environments to perform tasks which are very seldom repeated and which are quite often significantly different from each other. This is particularly true for the Andros robot. The Andros is a mobile robot which is used for a spectrum of missions and situations that are conducted in a variety of hazardous environments. These include explosive ordinance disposal, the nuclear industry, fire fighting and toxic material handling to name a few.

Computerized robot manipulator control has for a long time been associated with the development of dedicated programming languages **[Hayward, 86]**. Examples of some dedicated programming languages include the Victor's Assembly Language (VAL) and Machine Control Language (MCL). The goal of these languages has been to provide a

suitable framework for the expression of robot tasks. This has resulted in robot programming languages becoming increasingly more powerful and capable of handling interactions among robots. The language on which robot control is based must cope with the complexities of real-time control of the arm and its end effector, world modeling, sensory feedback and general input- output **[Hayward, 86]**. Thus the flexibility of the robot is necessarily fixed by the programming and control methods. Storr **[1987]** describes various programming methods for industrial robots. These are depicted in Figure 1.1. Programming methods are broadly classified into direct and indirect programming. Direct programming is further divided into manual programming and teaching processes. These are movement oriented. Manual programming is carried out on a keyboard or other methods of data storage. In explicit motion oriented control through programming, the programmer, having consideration of the robot's workspace, kinematic singularities and collision avoidance, describes the traversing path of the robot between different positions. In contrast, implicit programming procedures pre-suppose known environments which have to be previously described. The spatial conditions of the robot (collision range, traversing range etc.) are all specified in a model. The other data requried to perform a task is specified by the programmer.

Another aspect of robot control, which is gaining widespread popularity, is the use of modular architecture for real time control of telerobotic systems. In his paper on the Sequential Modular Architecture for Robotic Teleoperation (SMART) Anderson **[1993]** develops and describes a modular approach for constructing telerobotic systems consisting of input devices, sensors and output devices. This architecture is both open

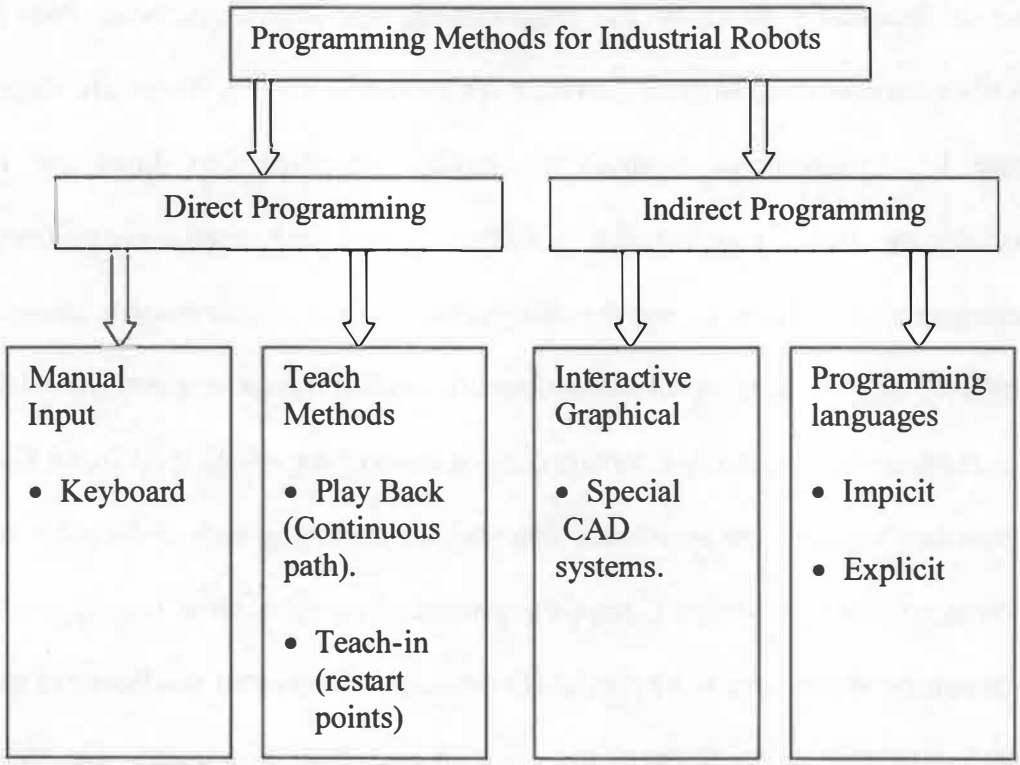


Figure 1.1: Programming Methods for Industrial Robots,
Source: Storr **[1987]**

and modular, where distinct modules are defined for each sensor, input device and actuator. Using SMART it is possible to build flexible telrobotic work cells that incorporate different devices and robots. SMART is distinctly different from dedicated programming languages in that it does not plan tasks or optimize the control architecture for particular sensors. Instead it provides a reconfigurable programming environment for telerobotics with scope for high level task planning **[Anderson, 96a], [Anderson, 96b].**

Even though significant advances have been made in the fabrication of teleoperated mobile robots and their application to hazardous environments, they remain unsophisticated in the realm of 'intelligence' with a human operator still providing intelligence for the system. The Andros Mark VI robot used for the purpose of this research does not have 'intelligent' capabilities such as path planning and sensor integration, however, this can be a major area for future research. In the applications where the Andros is being used, computing power is mainly limited to the control functions of the vehicle with the human in the feedback loop making the decisions. The major focus of this research is to provide the Andros with such a computing power. A schematic of such a system is outlined in Figure 1.2. From Figure 1.2 it is seen that the programming environment separates the control programs into two basic modules, the operator interface and the robot interface, allowing for each part to be developed and optimized separately.

Control of the Andros is achieved through data acquisition boards on board the robot and the controller box, with the RS-232 serial communication protocol serving as the medium

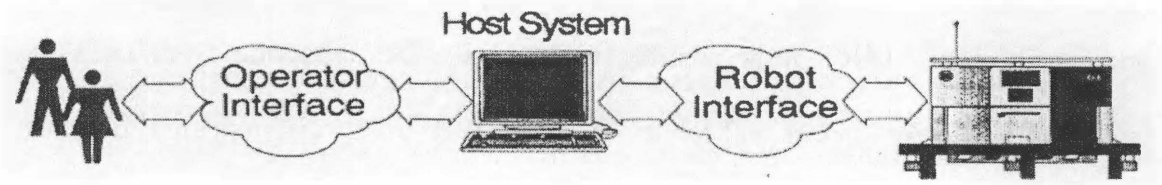


Figure 1.2: Conceptual Organization of PC based Robot Control
Source: Pettus et al. **[1991]**

of data transfer. Control of the Andros thus amounts to controlling specific motors which in turn impart motion to the desired joint. Hence the task of controlling the Andros focuses on establishing an RS-232 serial link between the computer and the robot itself. Once this is achieved a program is written to transfer data using the computer. The software consists of code that is written in the Visual Basic programming language using Microsoft's COM control and can be implemented on any PC with a Windows 95/98/NT/2000 operating system offering the Visual Basic programming environment. Since the robot involves a "human in the loop", a graphical user interface (GUI) is also developed. The user interface will emulate the functionality of the controller, with ease of operation and reduced down times being the main goals.

1.2 Purpose

This thesis aims at demonstrating PC-Based control of a mobile robot. The ultimate goal of this thesis is to build a Graphical User Interface (GUI) that controls the robot in a manner similar to the present controller while exploring the possibilities of adding greater sensory capabilities to the Andros.

1.3 Outline of a Thesis

Chapter 2 briefly describes the Andros Mark VI mobile robot. Since a vast majority of the work done in this thesis involved is based upon serial communication and the RS-232 protocol they are explained in detail in Chapters 3 and 4 respectively. Chapter 5 describes the actual communication link between the PC and the Andros including programming and specific examples of the protocol and the computer code. Chapter 6 describes the development of the graphical user interface (GUI) and results of the research. Chapter 7 concludes the work done for this research and includes a discussion on the Sequential Modular Architecture for Robotic Teleoperation (SMART), Web Based Robot Control and future implications of the research.

Chapter 2

The Andros Mobile Robot

This chapter describes the Andros Mark VI heavy duty mobile robot. The robot used for the purpose of this research is manufactured by Remotec Inc. The Kinematics for the Manipulator arm is outlined. The section ends with a short discussion on the control scheme used in the Andros.

2.1 Main Features

The Andros Mark VI is a heavy duty mobile robot used for a spectrum of situations conducted in a wide variety of hazardous environments. Some of these include explosive ordnance (bomb) disposal, response to emergency incidents and accidents, security and mining. The Andros is from the family of robots manufactured by Remotec Inc. Others in Remotec's family of robots include the Andros Wolverine, the Andros F6A and the Mini Andros II.

The Andros Mark VI is one of the largest and strongest robot in the Remotec family. Its 3.5 mph speed coupled with its unique articulated tracks allows the Andros to rapidly maneuver over rough terrain and obstacles, climb stairs and cross ditches as wide as 24 inches. The vehicle is environmentally sealed to operate in any weather condition and

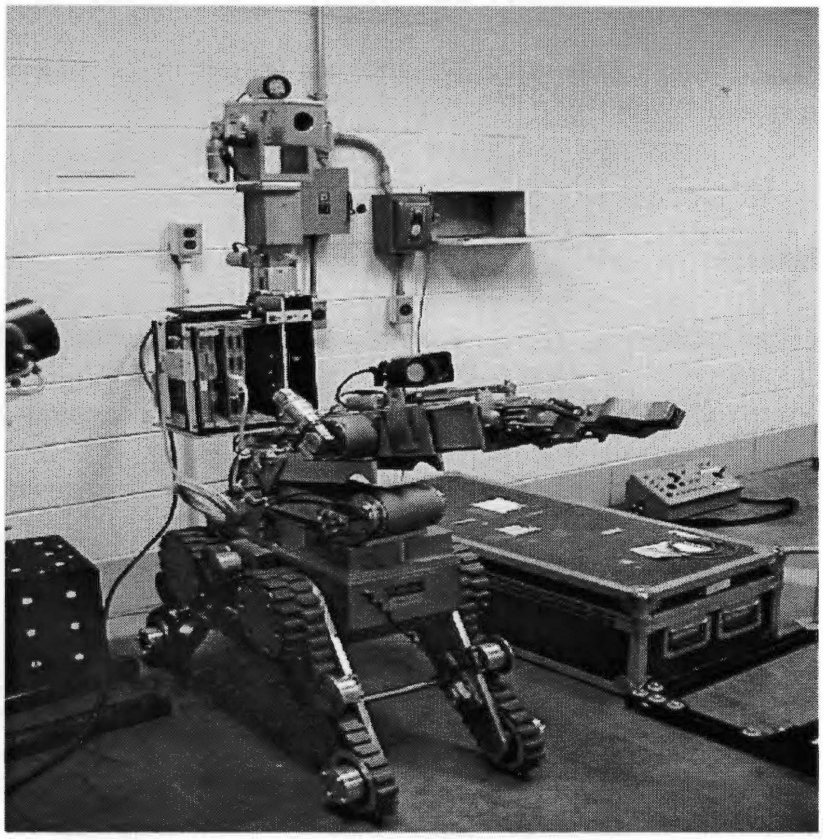


Figure 2.1: The Andros Mark VI

areas of high temperature and humidity. Figure 2.1 depicts the Andros Mark VI robot used for the purpose of this research.

The Andros is a teleoperator controlled mobile robot comprising of six drive wheels of which four are steering wheels mounted on legs which are able to rotate 360 degrees. Like most robots which are used for hazardous operations the Andros uses a tracked locomotion technique. This is very suitable for operations involving climbing steep inclines and stairs. The vehicle can abruptly change or reverse direction of movement sideways to the right or left without requiring a turning radius. This kind of movement

can be achieved because the vehicle's drive tracks can be reconfigured to a different geometry. Table 2.1 lists the principal dimensions and features of the Andros Mark VI.

2.2 Kinematics

The Andros manipulator is a six degree of freedom structure. Figure 2.2 illustrates the link structure for the Andros. The links for the manipulator are numbered 1,2,...n, starting from the base of the manipulator **[Yoshikawa, 90].** The displacement of joint i is denoted q_i and is called the joint variable. The collection of joint variables

$$\mathbf{q} = [q_1, q_2, \ldots\ldots q_n]^T \qquad (2.1)$$

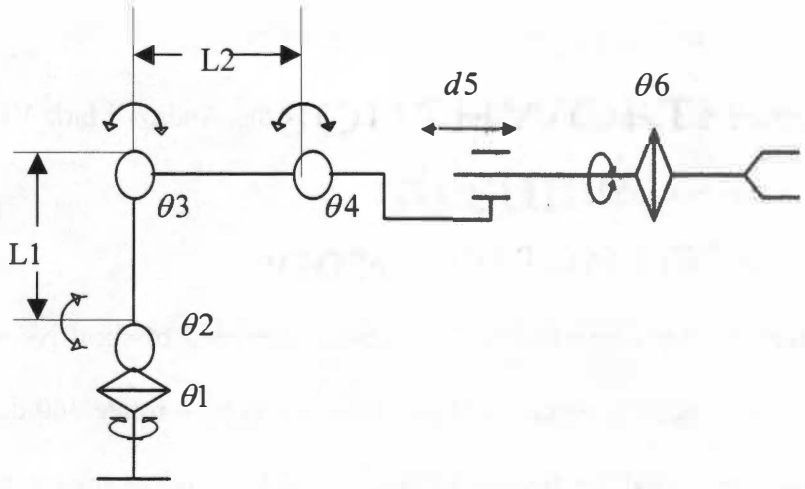


Figure 2.2: Andros Link Structure

Table 2.1: Andros Mark VI features
Source: **Remotec [2001]**

Vehicle Features: • Size: 28.5 in. (72.4 cm) wide X 43 in. (110 cm) high X 45 in. (115 cm) long. • Weight: 600 lbs. (275 kg) • Speed: Variable 0 to 2.0 mph (0 to 3.2 kph.) • Vehicle Power: Two-65 amp-hr., 12V DC batteries.
Locomotion Technique: • 6 wheels; Four with drives and variable configurations; two with fixed positions, tracked locomotion
Manipulator Features: • Shoulder pivot (210 degrees), wrist pitch (90 degrees), wrist roll (continuous), gripper with 12 in. (30.5 cm) full opening. Individual switches for each joint with variable speed control • Capacity: 60 lbs. (27 kg) at full extension and 100 lbs(45 kg) at 18 in (45 cm) reach. • Reach: 64 in (162 cm) from front of the vehicle and 92 in. (233 cm) above the floor. • Shoulder Rotation (+ /- 170 degrees from center). • Wrist pitch with 8 in. (20 cm) grip extend.
Video System: • Arm Camera: Black / White mounted on upper arm with wide angle lens and auto iris. • Surveillance Camera: Mounted on pan (± 170 degrees) and tilt (± 90 degrees), focus and iris.
Control Station: • Size: 18 in. (46 cm) deep X 22 in. (56 cm.) wide X 10 in. (25 cm.) high • Weight: 38 lbs (17 kg) • Removable switch box for walk along control • Optional Radio Control.

is called the joint vector. The position of the end effector can then be denoted by the m-dimensional vector

$$\mathbf{r} = [r_1, r_2, \ldots\ldots r_n]^T \qquad (2.2)$$

According to Yoshikawa **[1990],** for a general case where the end effector can take an arbitrary position and orientation in three dimensional Euclidean space, m = 6. However, when the manipulator moves in a two dimensional plane and the only concern is to determine its end point in the plane m can be set to 2. The relation between r and q, determined by the manipulator mechanism is nonlinear. This relation can be given by

$$\mathbf{r} = f_r(q) \qquad (2.3)$$

where, r and q are as defined earlier. This equation is called the kinematic equation of the manipulator. The problem of obtaining r for a given q is called the direct kinematics problem, and that of obtaining q corresponding to a given r is referred to as the inverse kinematics problem.

For the purpose of this research, a brief description will be given of only the direct kinematics problem.

The first step in solving the kinematics problem is to assign link frames to the manipulator structure. Although a lot of conventions are available to assign link frames, this thesis prescribes to the method outlined by Yoshikawa **[1990].** In general one coordinate frame is fixed to each link. The origin of coordinate frame Σ_i of link i is set at

the end point of the mathematical model of link I on joint axis i.. The Z-axis of Σ_i, denoted by Z_i, is selected in such a way that it aligns with joint axis I in the direction pointing toward the distal end of the manipulator. If the direction of the distal end of the manipulator is not clear then the direction of the Z-axis is arbitrary. The X axis of Σ_i, is selected so that it is on the common normal and points from joint i to joint i+1. The Y axis is selected in such a way that the link frame is a right hand coordinate frame. The link frame assignments for the Andros are shown in Figure 2.3. Once the link frames are set, a table detailing the link parameters is drawn. This table is known as the Denavit-Hartenberg (D-H) table. The table makes it easier to solve for the kinematics of the given manipulator. The D-H table for the Andros is outlined in Table 2.2.

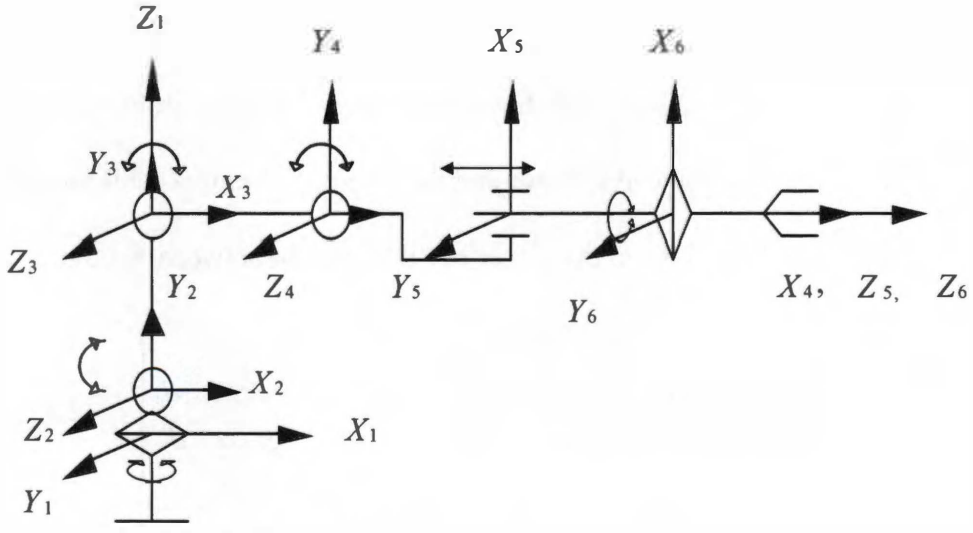


Figure 2.3: Link Frames for the Andros

Table 2.2: D-H table for the Andros

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	+ 90	0	θ_2
3	L1	0	0	θ_3
4	L2	0	0	θ_4
5	0	+90	0	θ_5
6	0	0	d_5	0

Referring to Table 2.2:

a_i = the distance along the X_i axis from Z_i to Z_{i+1},

α_i = the angle measured clockwise about the X_i axis from Z_i to Z_{i+1}.

d_i = the distance measured along the Z_i axis from X_{i-1} to X_i and

θ_i = the angle measured clockwise about the Z_i axis from X_{i-1} to X_i

Once the parameters in the D-H table have been determined, the problem reduces to finding a relation between the end effector and the reference frame Yoshikawa **[1990].** defines the homogenous transform that describes the relation between Σ_i and Σ_{i-1} as

$$T_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \cos\alpha_{i-1}\sin\theta_i & \cos\alpha_{i-1}\cos\theta_i & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ \sin\alpha_{i-1}\sin\theta_i & \sin\alpha_{i-1}\cos\theta_i & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

where, T_i^{i-1}, is the homogenous transformation of the vector from the Σ_i frame to the Σ_{i-1} frame. If the homogenous transform relating Σ_n to Σ_0 is T_n^0, then we can write

$$T_n^0 = T_n^0 T_2^1 \ldots\ldots T_n^{n-1} . \qquad (2.5)$$

When all the values of the link parameters are known T_i^{i-1} is a function of the joint variables and T_n^0 is a function of the joint vector q **[Yoshikawa, 90]**.

2.3 Control Scheme

The Andros is a completely teleoperated mobile robot. The robot is controlled by an operator using a control module, with feedback being either through visual contact or video transmission. Under these conditions the human operator must exercise extreme care while operating the vehicle in hazardous and obstacle ridden environments. The control scheme for the Andros is depicted in Figure 2.4. It has to be noted that with the Andros having limited sensory capabilities this kind of control is virtually 'open-loop'.

Referring to Figure 2.4, T is the motor torque, R is a gear ratio. F is the force and Meq is the equivalent mass. The present scheme for control of the Andros does not allow for any kind of autonomous control. This can be a drawback during the operation of the vehicle, especially in environments where vision is limited (for example smoke filled areas). It is in this context that computer control of the Andros assumes prime importance. This thesis aims to replace the present controller with a PC based controller thus paving the way for new interfaces to be added to achieve complete autonomous control. Even though the type of control proposed in this research is pure functional control it has great prospects for future work. PC based functional control of the Andros via an RS-232 serial link is discussed in the forthcoming chapters.

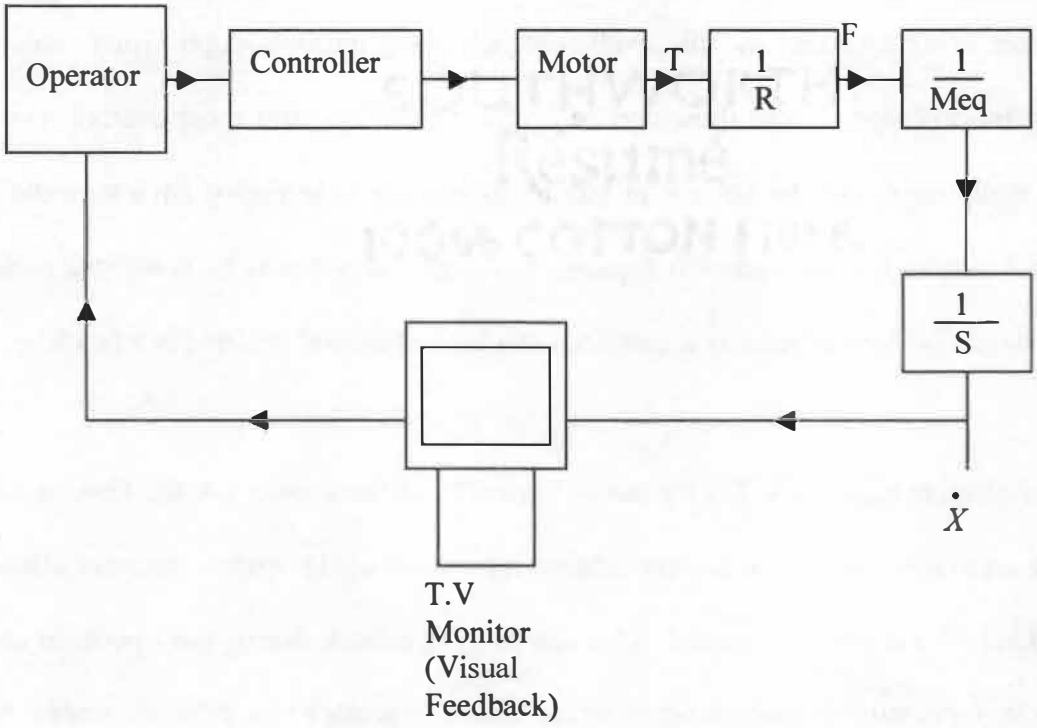


Figure 2.4: The Andros Control Scheme

Chapter 3

Serial Communication

Since a major part of the research involved serial communication between the PC and the robot an effort was made to understand the different aspects of serial communication including data transfer and serial port architecture. This part of the research focuses on a detailed study of the serial port and its usefulness in communicating with the Andros robot.

3.1 Serial Interfaces

Serial communication refers to the transfer of data on a bit-by-bit basis as compared to parallel transfer of data wherein data are transmitted and received in multiple bits. In very simple terms a serial port is an interface that provides a path for information to pass to and from a digital computer. In modern computer jargon the serial port is also known as a 'COM' port or an 'Asynchronous communications adapter'. Information sent from the computer is sent to the serial port by communication software known as the serial port driver.

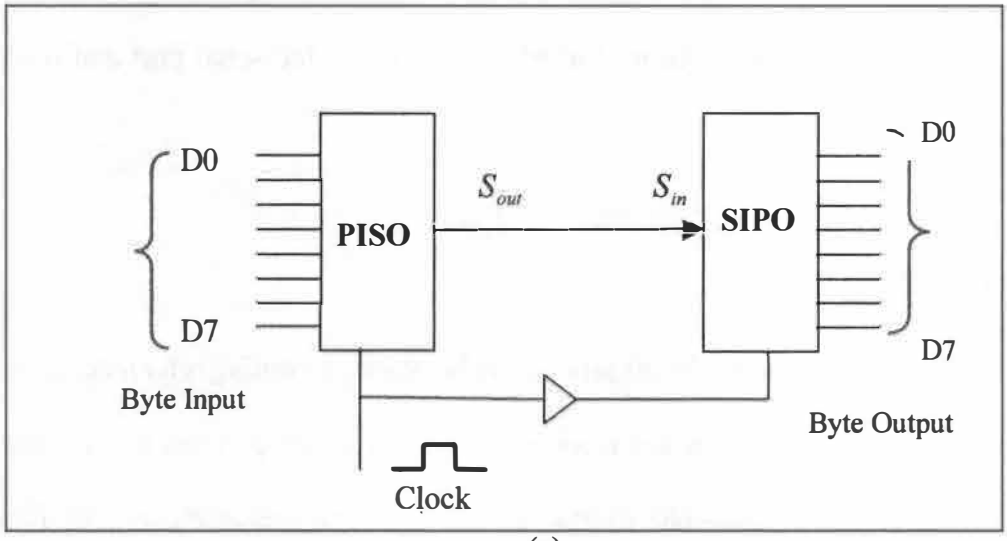
The serial port is much more than just a connector. It converts data from parallel form to serial form. This is done using two digital devices called the shift registers, registers will be discussed in a later section. The two registers are the parallel-in-serial-out (PISO)

shift register and the serial-in-parallel-out shift register (SIPO). PISO shift register serializes a byte of data into a single stream of bits. The SIPO shift register provides a parallel output for a serial input.

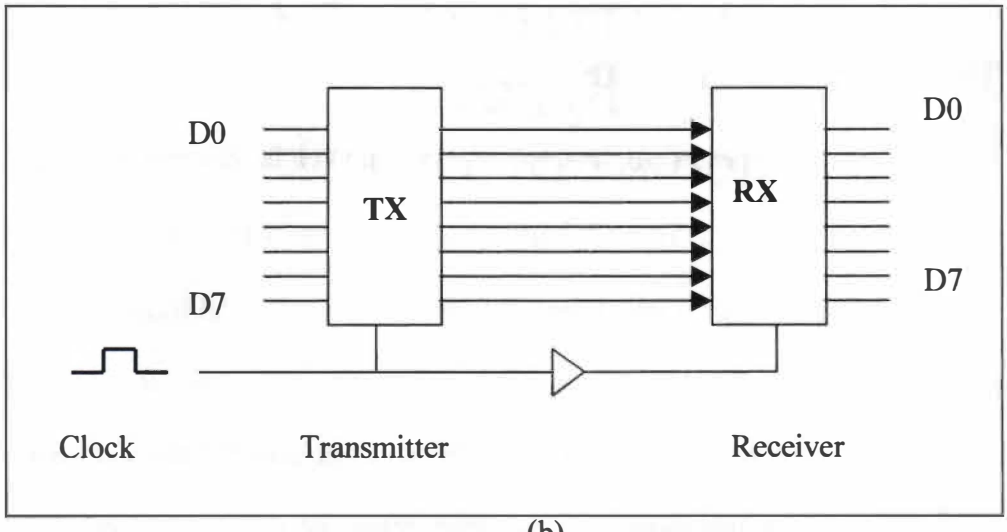
Figure 3.1 shows models for serial and parallel interfaces. Figure 3.1 (a) illustrates the PISO and SIPO shift registers. It is seen that data in the serial interface is sent on a single line. A Parallel interface on the other hand has each bit of data sent on a separate path, this is illustrated in Figure 3.1 (b). In serial communication a pair of wires can be used to simultaneously transmit and receive data between two devices, this makes serial communication a very convenient mode of communication. However, serial communication has its drawback in being slow. Parallel transfer of data can transmit a byte of data in a single clock pulse the same transfer of data via the serial mode would take a much longer time **[Putman, 87]**.

3.2 Transmitting Serial Data

A serial port provides a path for information to pass to and from a digital computer. Information sent from the computer is sent to the serial port by "communication software" or a serial port driver. A driver is a low-level program, which is usually written in assembly language, that is used to control a specific I/O device. The driver's software sends the information to the serial port by writing a character (for example, the letter "A") to the appropriate serial port address. The serial port receives this character, converts it into a signal which can be sent through a serial cable, and sends it to the device connected to the serial port.



(a)



(b)

Figure 3.1: Serial and Parallel Interfaces

Information sent to the computer is received by the serial port from the cable, and prepared for the computer. When the serial port has prepared a character for the computer, the serial port sends a signal to the computer through an interrupt request line telling the computer that a character is ready. When the computer sees the interrupt request, the communication software (or driver) "services" the serial port and reads the character.

3.3 Data Formats

An essential signal required by all serial links is a clock, or timing reference, to control the flow of data. The transmitter and receiver use a clock to decide when to send and read each bit. Two types of serial-data formats are synchronous and asynchronous formats, and each uses clocks in different ways.

3.3.1 The Synchronous Format

In a synchronous transmission, all devices use a common clock generated by one of the devices or an external source. Figure 3.2 illustrates this type of transmission. The clock may have a fixed frequency or it may toggle at irregular intervals. All transmitted bits are synchronized to the clock. In other words, each transmitted bit is valid at a defined time after a clock transition (rising or falling edge). The receiver uses the clock transitions to decide when to read each incoming bit. The exact details of the protocol can vary. For example, a receiver may latch incoming data on the rising or falling clock edge, or on detecting logic high or low level **[Axelson, 00]**.

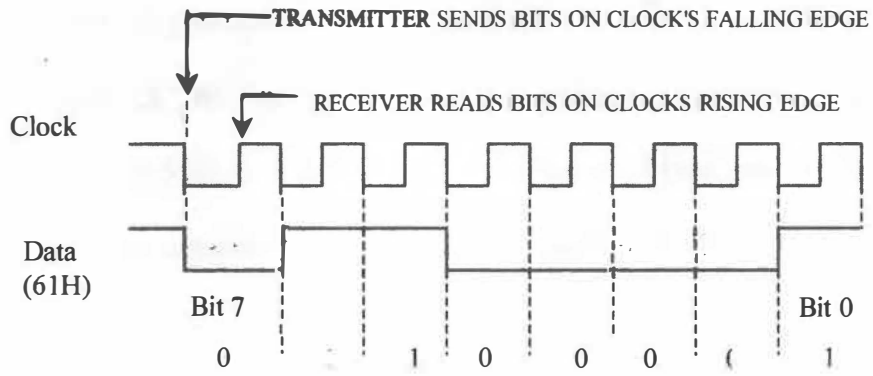


Figure 3.2: Synchronous Transmission

Synchronous formats use a variety of ways to signal the start and end of a transmission, including start and stop bits and dedicated chip-select signals. Synchronous interfaces are useful for short links. For longer links, typically 15 feet or more, synchronous formats are less practical because of the need to transmit the clock signal, which requires an extra line and is subject to noise.

3.3.2 The Asynchronous Format

In the asynchronous format each device uses its own internal clock resulting in bytes that are transferred at arbitrary times. Hence, instead of using time as a way to synchronize the bits, the data format is used. This eliminates the need for a separate system clock. Each transmitted byte includes a start bit to synchronize the clocks and one or more stop bits to signal the end of a transmitted word. In other words, the data transmission is synchronized using the start bit of the word, while one or more stop bits indicate the end of the word. The requirement to send these additional bits causes asynchronous communications to be slightly slower than synchronous transmissions.

However, , asynchronous communication has the advantage that the processor does not have to deal with the additional idle characters. Most serial ports operate asynchronously. The RS-232 ports on PC's use the asynchronous format to communicate with modems and other devices. The transmission of each character on an asynchronous communications line is prefaced by a start bit. This start bit is a space (logic 0) that has the duration of a one bit cell. The order in which data transmission takes place is as follows:

1. The start bit is transmitted with a value of 0.
2. The data bits are transmitted. The first data bit corresponds to the least significant bit (LSB), while the last data bit corresponds to the most significant bit (MSB).
3. The parity bit (if defined) is transmitted.
4. One or two stop bits are transmitted, each with a value of 1.

This format is commonly known as the 8-N-1 format and is illustrated in Figure 3.3.

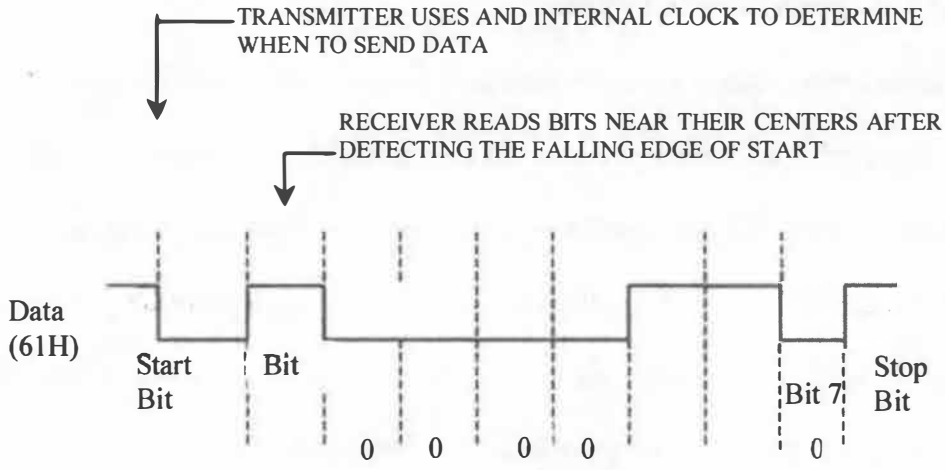


Figure 3.3: Asynchronous Transmission

The N in the 8-N-1 indicates that the transmissions do not use a parity bit. However, a parity bit may be included in some formats for error checking. Parity can be Even, Odd, Mark or Space. The parity bit is an error detection technique in which an extra bit, called a parity bit, is added to each byte in a transmission, holding a 0 or 1 depending on whether the byte has an odd or even number of 1 bits. The sender and receiver agree on odd parity, even parity, or no parity. If they agree on even parity, a parity bit will be added that will make each byte even. If they agree on odd parity, a parity bit will be added that will make each byte odd. If the data is transmitted incorrectly the change in parity will reveal the error. Table 3.1 lists the various parity types.

With Even parity, the parity bit is cleared or set so that the data bit plus the parity bit contain an odd number of 1's. As an example, consider the format 7-E-1, here the transmitter sends 1 start bit, 7 data bits, 1 parity bit and 1 stop bit. The receiver examines the received data and informs the transmitter of an error if a result isn't the desired value. With Mark parity the parity bit is always 1 and with Space parity the parity bit is always 0. Mark and Space parity are seldom used since they offer minimal error detection.

3.4 Data Bits and the Bit Rate

The data bits transferred through a serial port may represent device commands, sensor readings, error messages, and so on. The data can be transferred as either binary data or as text (ASCII) data. Most serial ports use between five and eight data bits. Binary data is

Table 3.1: Parity Types

Parity Type	Description
Even	The data bits plus the parity bit produce an even number of 1's.
Mark	The parity bit is always 1.
Odd	The data bits plus the parity bit produce an odd number of 1's.
Space	The parity bit is always 0.

typically transmitted as eight bits. Text-based data is transmitted as either seven bits or eight bits. If the data is based on the ASCII character set, then a minimum of seven bits is required since there are 2^7 or 128 distinct characters. If an eighth bit is used, it must have a value of 0. If the data is based on the extended ASCII.

A link's bit rate is defined as the number of bits per second transmitted or received per unit of time, usually expressed as bits per second (bps). The number of possible data transitions per second is defined as the Baud rate. All the bits that are required to transmit a value from Start to stop bit for a word and the data bits in a word form a character. Characters can be actual test or just binary values. The bit rate times the number of bits in a word gives the number of characters transmitted per second **[Axelson, 00]**.

3.5 Serial Port Architecture

Serial ports have been an integral part of digital computers right from their inception. On modern digital computers the serial port is also called a 'COM' port. Each 'COM' port on a computer is an asynchronous serial port. In recent years however, newer serial interfaces like the Firewire and the Unified Serial Bus (USB) have been developed. These

use different protocols and different components. Serial ports, however, , along with similar interfaces continue to be popular.

In recent years data transmission using the asynchronous format via the serial port has been made simpler with the development of a hardware component known as the Universal Asynchronous Receiver and Transmitter (UART). In all modern digital computers the operating system(s) and programming languages include support for programming serial links without having to understand every detail of the UART architecture.

3.6 The Universal Asynchronous Receiver and Transmitter (UART)

The introduction of the UART has a very great effect on the computer design interface. The UART is a special type of integrated circuit (IC) that is fully programmable and therefore is suited to any RS-232 interface design. In simple terms the UART helps translate between serial and parallel data. To send a byte the application writes the byte to the transmit buffer of the selected port, and the UART sends the data bit by bit adding the start, stop and parity bits when needed. The computer's UART supports both full and half duplex communications. A useful application of the terms of full and half duplex is to describe which end of a link is responsible for marking characters to a display.

A block diagram for a UART is shown in Figure 3.4. The UART is powered by +5 V and ground. All the inputs and outputs meet standard TTL logic level specifications. TTL stands for Transistor –Transistor Logic and refers to a special type of digital circuit.

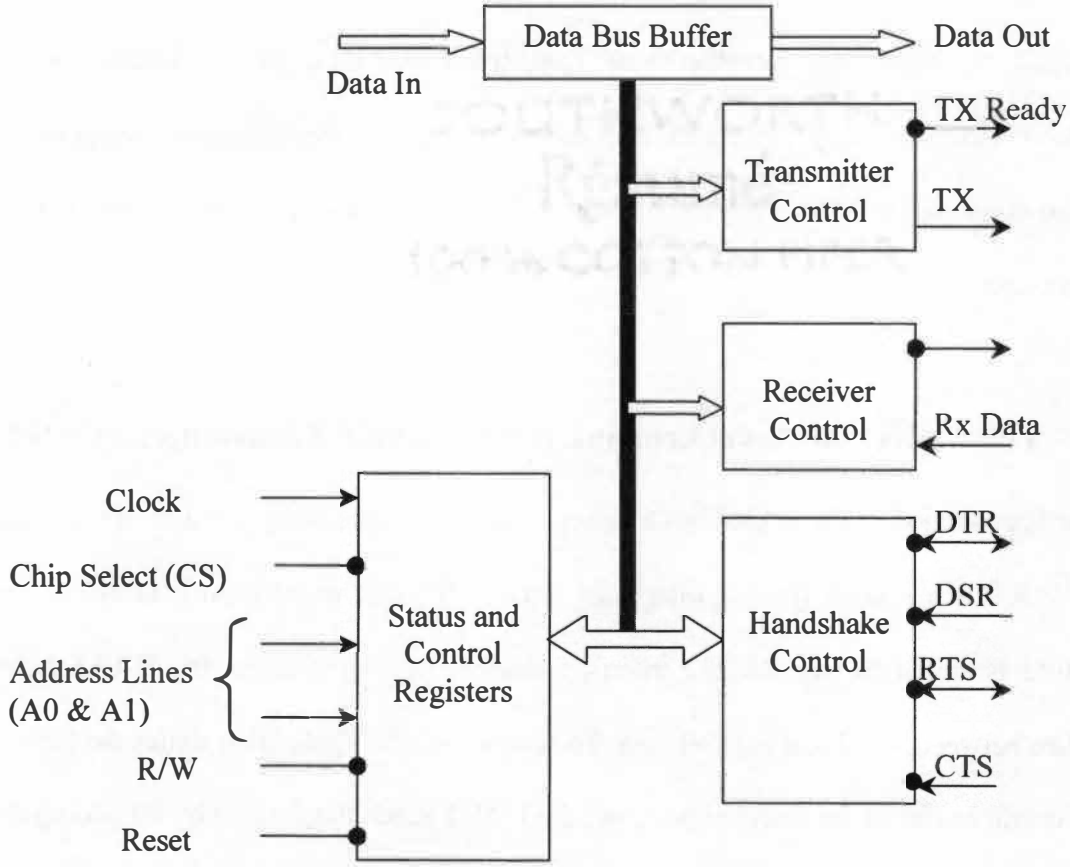


Figure 3.4: Block Diagram of a UART

Referring to Figure 3.4 the small bubble shown indicates that a line is active at a logic 0 level. This means that a bubbled input is enabled or active at logic 0 and disabled at logic 1. A bubbled output indicates a symbolic inverter that compliments the normal state of the output. The command register is used to enable the transmitter and receiver and to control the handshaking outputs of the Data Terminal Ready (DTR) and Ready to Sent (RTS) signals.

The clock pin is an input which is driven by an external clock. The clock input is usually 16 times the clock rate this helps the UART to transmit the output bit stream as well as to sample the received line precisely. Since the microprocessor is capable of communicating with only one device at a time a way has to be chosen for effective communication. This is accomplished by the chip select (CS). The microprocessor selects the UART by pulling its chip select (CS) to an active- low this in-turn is accomplished by enabling the address outputs of the microprocessor by an IC called the address decoder. The transmit data register and the receiver data register are respectively contained in the transmitter control and receiver control blocks. The actual pinout and names on UART's may vary but the basic concepts explained above are applicable to all commercially available UART's. The UART has been extensively discussed by Putman **[1997]** and Axelson **[2000]**.

3.7 Port Address and Interrupt-Request (IRQ's)

Since a serial port is most commonly used for data communication the computer or any other equipment connected to it must know its location, this is made possible by assigning addresses to the serial port. Most serial ports use pre-assigned addresses. The addresses are almost always described in Hexadecimal (Base 16). Whenever a serial port receives a large amount of data (bytes) into its FIFO buffer (First in First Out) it signals the CPU to retrieve them by sending a signal known as an interrupt. This signal is normally sent on a certain wire used by that port.

Thus the computer needs to know where the serial port exists and which wire the serial port must use to request service from the CPU. In other words the computer needs to know the serial ports address and interrupt request (IRQ). Thus every serial port device must store in its non-volatile memory both its I/O address and its Interrupt Request Number (IRQ). I/O addresses are not the same as memory addresses. When an I/O addresses is put onto the computer's address bus, another wire is energized. This tells the main memory to ignore the address and also tells all devices which have I/O addresses (such as the serial port) to listen to the address to see if it matches the device's. If the address matches, then the I/O device reads the data on the data bus. The port addresses and IRQs used for serial ports are different for ISA bus computers and IBM PS/2 computers. The "ISA" standards for COM1 and COM2 were set by IBM when they originally designed the personal computer. The addresses and IRQs for COM3 and COM4 are widely accepted conventions. The conventional 'COM' port addresses and IRQ lines are shown in Table 3.2.

Table 3.2: COM port addresses

Port	Address	IRQ
COM 1	3F8h	4
COM 2	2F8h	3
COM3	3E8h	4 or 11
COM 4	2E8h	3 or 10

3.8 Buffers

A buffer is a means to ensure that the computer does not miss any data while it is busy performing tasks other than communication. A computer may want to transmit at a time when the receiving computer is occupied with another task or may want to receive data while it is busy the buffer would then store data to be sent or received for a time when the link becomes available.

The buffers may be in the software or hardware or both. Most of the modern digital computers have 16-byte hardware buffers built into the UART's. This means that the UART can store up to 16 bytes of data before the software can read them. When the hardware buffers are not large enough the computer may use software buffers, which are programmable in size and maybe as large as the system memory permits. The transfer of data between the hardware and software buffers is accomplished by the ports software driver. In addition to buffers computers use other means to ensure that data is not missed. Some of these include handshaking, polling and interrupts.

3.9 Registers

As seen from section 3.1 shift registers are a type of sequential logic circuit, mainly for storage of digital data. One can also connect shift registers to each other to get more input and output options. Shift Registers are a group of "flip-flops" connected (joined) in a chain so that the output from one "flip-flop" becomes the input of the next "flip-flop".

Most of these shift registers do not possess a characteristic internal sequence of states. However, , all the "flip-flops" are driven by a common clock, and all are set or reset simultaneously. For example the IBM standard 16450 UART contains twelve 8-bit registers. The registers hold the next byte to transmit, the last byte received, the bit rate and other port settings. They also hold status and control information for handshaking and interrupts. The UART requires just eight port addresses even though it has twelve internal registers. The Fife's are internal to the UART and require no port addresses. The line control register (LCR) stores configuration information such as the number of stop, data and parity bits used in each transmission.

The serial port can be considered to be a 'gateway' to the communication link between two devices. This link is known as an RS-232 link and is discussed next.

Chapter 4

The RS-232 Communications Protocol

Computer control of the Andros was achieved using serial communication and the RS-232 communications protocol. This section discusses the RS-232 protocol in detail. Central to the discussion is the type of RS-232 signals, links to and from the computer including connectors, pin outs and cables. Much of this information can be found in standard texts on serial communication. Interested readers can refer to Axelson **[2000]** and Seyer **[1984]**. However, for readers unfamiliar with the RS-232 standard a good understanding of the protocol will enable the reader to better understand future discussions.

4.1 Introduction and Historical Background

The RS-232 (RS stands for recommended standard) has been one of the most widely followed and accepted computer input-output (I/O) interface standards. It can be considered as a workhorse that is used in just about every PC as well as other types of computers from microcontrollers to mainframes and the devices to which these connect to. Despite the advent of newer, faster and more sophisticated devices like the Universal Serial Bus (USB), the RS-232 continues to be popular due to its simple hardware and programming requirements as well as its inexpensive nature. The fact that a lot of existing devices already have such an interface built into them also makes it a preferred

mode of data communication. The standard was developed in the early 1960's by the Electronic Industries Association (EIA) as a standard for connecting business machines with serial interfaces and to facilitate compatibility and interchangeability between computers and peripheral devices that were connected via serial ports. Since data exchange between computers is only possible if there exists a compatible format and a standard. This led to the development of the RS-232 standard. RS-232 was used as the link when it became desirable for the end-users to access computers from remote locations. For short distance RS-232 communication, for example two locations within the same building, the connection could be made by the addition of extra wires; however, for truly long distance communications a new device called the modem was needed in addition to the direct RS-232 link.

Modems were developed because digital waveforms cannot be transmitted over long lengths of wires without significant losses. A means had to be devised to impress a digital signal onto an analog carrier; this process is known as modulation. The reverse process of recovering a digital signal from an analog carrier is called demodulation. The modem is thus a transceiver which means it transmits as well as receives data. RS-232 was initially developed as a means of data transfer between the computer and the modem **[Putman, 87]**. Such a connection is illustrated in Figure 4.1. Referring to figure 4.1 two new terms are introduced viz the DTE (Data Terminal Equipment) and the DCE (Data Circuit-terminating Equipment). The standard defines these to be the terminal end of the link and the modem end of the link respectively. These names reflect the original purpose of the interface. typical DTE is an ordinary terminal with a keyboard and a video display

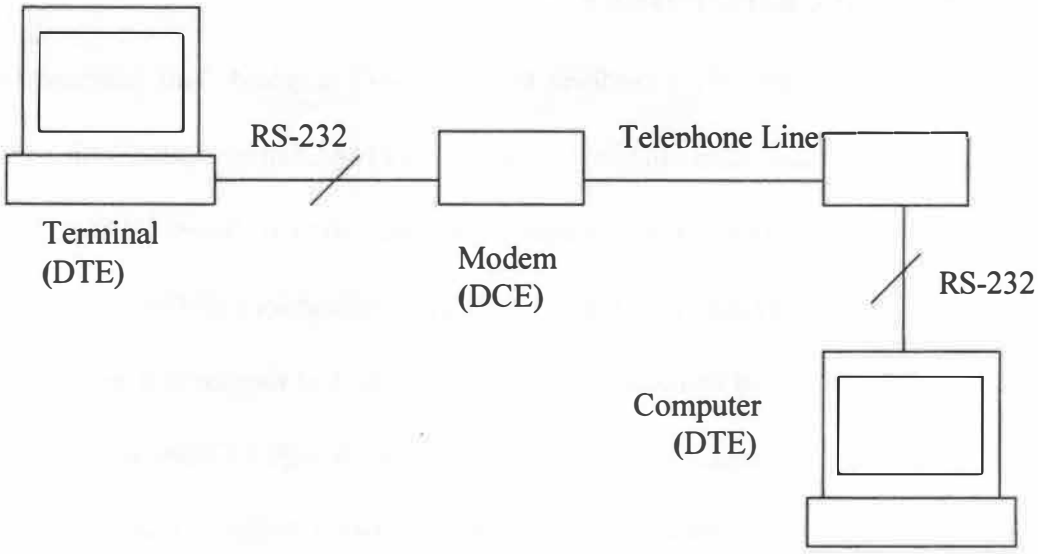


Figure 4.1: Computer / Modem connection using RS-232

unit. For a successful RS-232 link it is imperative that the link must have one each of the DTE and the DCE. For this research the PC was termed as the DTE while the Andros robot was designated as the DCE. It is also interesting to note that the link with the Andros is 'one-way' in that the PC transmits command signals to the robot but does not receive any signal from the robot. This is in contrast with the traditional RS-232 link where the modem, which is the DCE, can both send and receive data.

A detailed discussion of the RS-232 protocol and serial communication has been presented in Putman's "RS-232 Simplified" **[Putman, 87]** and Seyer's "RS-232 Made Easy" **[Seyer, 84]**.

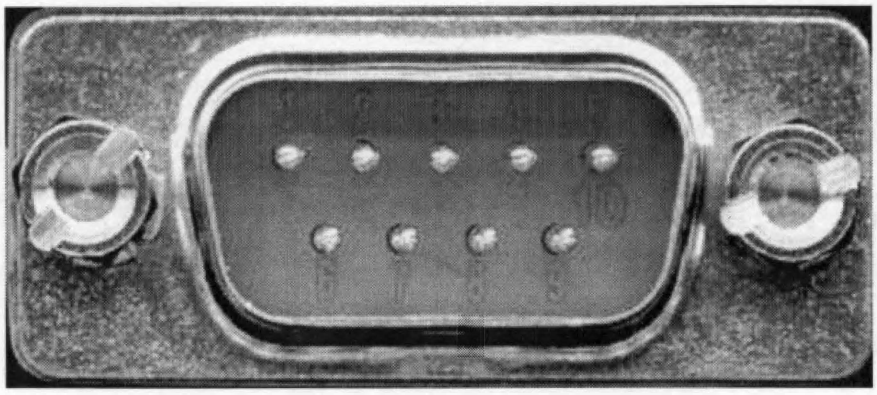
4.2 RS-232 Signal Characteristics

Standard serial ports on most PC's conform to the RS-232 standard. This standard is defined by the Telecommunications Industry Association (TIA), which outlines them in a technical document. An extract of this document can be found in Seyer **[1988].** The standard broadly outlines three aspects of the signals viz the functions and names of the signals in the link, the electrical characteristics of the signals and mechanical aspects of the signals including pin assignments on the interface. Even though 25 lines are defined in the interface, PC's and other devices using the asynchronous format of data transfer rarely support more than nine signals. Some devices, the Andros Robot for example, use only two signal for successful communication. The remaining signals are intended for use with the synchronous format of data transfer. Table 4.1 lists the signals for a standard 9-pin and 25-pin connector. Figures 4.2 (a) and (b) illustrate a standard "D-type" 9-pin RS-232 connector.

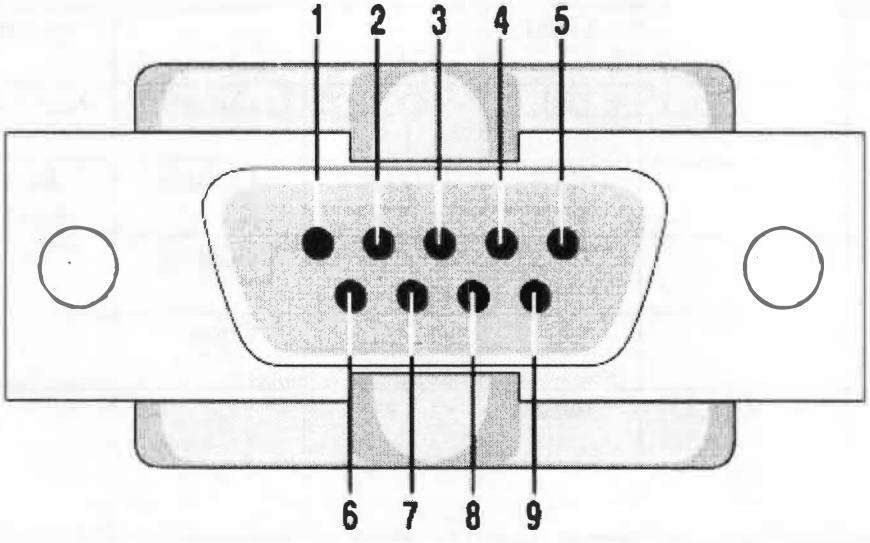
For successful two-way communication to take place the essential signals are, Transmitted Data (TD), which carries data from the DTE to the DCE, Received Data (RD) which carries data from the DCE to the DTE and Signal Ground (GND). In addition to data paths a link also has additional lines to indicate the status of the transmitter and the receiver. This link indicates when a transmitter is able to send data and when a receiver is able to receive data. This process of exchanging information about the status during a transmission is called 'handshaking'. RS-232 had two pairs of handshaking signals viz, the Data Transmit Ready (DTR) and the Data Set Ready (DSR).

Table 4.1: Signal Description for the 9-pin and 25-pin RS-232 Connectors

Pin (9-Pin)	Pin (25-Pin)	Signal	Source	Type	Description
1	8	DCD	DCE	Control	Carrier Detect
2	3	RD	DCE	Data	Received Data
3	2	TD	DTE	Data	Transmitted Data
4	20	DTR	DTE	Control	Data Terminal Ready
5	7	GND	-	-	Signal Ground
6	6	DSR	DCE	Control	Data Set Ready
7	4	RTS	DTE	Control	Request to Send
8	5	CTS	DCE	Control	Clear to Send
9	22	RI	DCE	Control	Ring Indicator
-	1,9-19,21,23-25	Unused	-	-	-



(a)



(b)

Figure 4.2: Pin Out for a 9-pin RS-232 Connector.

Handshaking can be considered to be 'co-operation' between the devices that are communicating or exchanging data.

An important question that arises here is : how is the state of the control signal described ? A common way to do this is to describe a signal with a valid positive voltage as 'On' or 'True' to indicate its active state. For example when DTR is true the data terminal is ready. To bring the signal to 'True' the controlling device raises the volage. A signal with a valid negative voltage may be described as 'Off' or 'False'. This would indicate that it is in the inactive state. Referring to Figure 4.2 illustrates the standard 9-pin D-type connector found on a PC's COM port. A brief description of the signals follows:

Data Carrier Detect (DCD): used by a DCE to indicate to the DTE that it has received a carrier signal from the modem and that real data is being transmitted.

Received Data (RD): This line is used for receiving data.

Transmitted Data (TD): This line is used to transmit data.

Data Terminal Ready (DTR): Used by a DTE to indicate that it is available to begin communication.

Signal Ground (GND): This is the common ground used by all the signals.

Data Set Ready (DSR): This is the corresponding signal to the DTR, used by the DCE to indicate that it is ready to begin transmission.

Ready to Send (RTS): Used by the DTE to indicate that it wants to send data.

Clear to Send (CTS): Used by the DCE to indicate that it is available to send data and used in response to an RTS request for data.

Ring Indicator (RI): This is true when a ringing signal is present on the communication channel.

4.3 RS-232 Voltage Levels

RS-232 logic levels are indicated by positive and negative voltages. As opposed to TTL (transistor-to-transistor logic) where a more positive voltage is a logic 1 and the more negative voltage is a logic 0, RS-232 employs negative-logic levels; a logic 0 is defined as equal to or more than +5 V, and a logic 1 is defined as equal to or more negative than -5 V. Thus, the more negative voltage is logic 1.

Two very popular designations describing RS-232 line levels are inherited from teletype technology. A line at a logic level 1 is said to be a 'mark' to indicate that the device is idle and not sending any data. A device that is not transmitting data holds it's transmit line at a marking level – logic 1. The term used to describe logic 0 is 'Space'. A data line that is transmitting is at a spacing level. The control lines that perform handshaking use the same voltages but with positive logic. A handshaking line that is at a logic level 1 can be termed 'on' or 'active'. A negative voltage indicates that the function is 'off'. The main idea is that when a control line is at a logic 1 it is in a marking or idle state. When a control line is asserted to logic 0 it is in an active or on state.

To allow for voltage attenuations and noise which may ride on a RS-232 link the minimum required voltages at the receiver are less than at the driver. Thus an input more positive than +3 V is logic 0 at the received data end. An input more negative than -3 V is

Table 4.2: RS-232 Voltage levels

Voltage	Logic	Control	Terminology
+3 V to +25 V	0	On	Space
-3 V to -25 V	1	Off	Mark

logic 1. The noise margin or voltage margin is the difference between the output and input voltages. RS-232's large voltage swings result in wide noise margins. The maximum allowed voltage swing is (+/-) 15 V. Most RS-232 devices operate with +12 V and -12 V power supplies. A typical Mark is from -9 V to -12 V and a typical space is from +9 V to +12 V. Voltages out of this range usually indicate interface malfunction.

Table 4.2 lists the standard RS-232 voltage levels and their interpretations

4.4 Cables and Interfacing

In order for DCE and DTE devices to communicate with each other, two wires must be used--one for transmission and the other for reception--and both devices must not use the same wire for the same purpose. Should this happen, nothing would be communicated because both devices would be talking on the same line and listening to a line on which nothing is transmitted. To solve this problem, DTE and DCE devices have complementary pin-outs to allow terminals and modems to be connected directly using a one-to-one cable; this is depicted in Figure 4.3.

In certain situations no DCE device is needed, such as a desktop computer communicating with a laptop. In this case, a null modem cable or modem eliminator cable is used to connect the two DTE devices. This cable effectively changes the wires

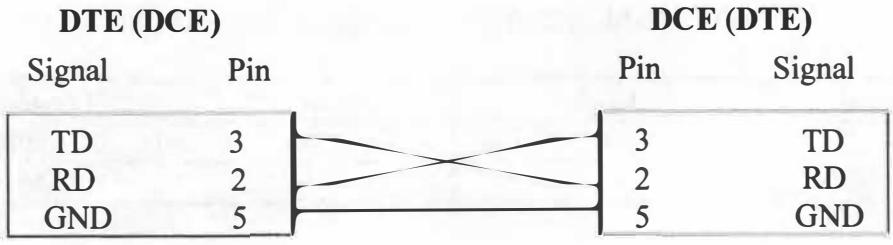


Figure 4.3: Typical Null Modem Connection Between a DTE and DCE

the second device uses for transmission and reception to ensure that both sides can communicate effectively. This is known as a 'null modem' connection. The simplest of the null modem connections is for 3 wire connections. The null modem swaps the RD and TD lines so that each TD connects to the opposite RD. This kind of connection was used between the PC and the Andros.

Another important consideration while making an RS-232 connection is the length of the cable involved. Early versions of the standard recommended limiting cable length to 50 feet; however, for asynchronous data transfer this length can be increased to as much as 500 feet. Generally for data transfer rates of 20,000 bps or less just about any type of cable can be used. An important parameter effecting cables is the capacitance. Capacitance tends to limit the rate of voltage change when an output switches. Capacitance between wires can also result in crosstalk, where the signal on one wire also shows up on adjacent wires. To reduce cross talk twisted pair cables and multiple ground wires can be used. Twisted pairs are effective at canceling low frequency noise caused by magnetic coupling.

Chapter 5

Communication with the Andros Robot

This chapter discusses the actual transfer of data to the Andros. The discussion centers on the protocol used for data transfer. Also discussed are the methods used to decipher the existing protocol from the Andros. Visual Basic programming is discussed with specific examples explaining how the program opens the PC's COM port for communication and how the protocol controls the various functions in the robot.

5.1 Reading Data From the Controller

The Andros communicates with its controller using RS-232. Essentially this means that the controller sends packets of information (bytes) to the Andros, each packet controls specific motors which impart motion to the desired joints. Figure 5.1 shows the front panel of the control box used to control the Andros The Vehicle control section has controls for the robot movement and safety settings. The arm control section has switches to control motors for the torso, shoulder, elbow, wrist roll, wrist pitch and gripper. Camera controls include switches for the zoom, focus and camera change. Each switch, when operated, transmits a voltage signal which is interpreted by the controller on the robot as high or low (1 or 0). The controller then sends out a corresponding signal to the motor to move the joint.

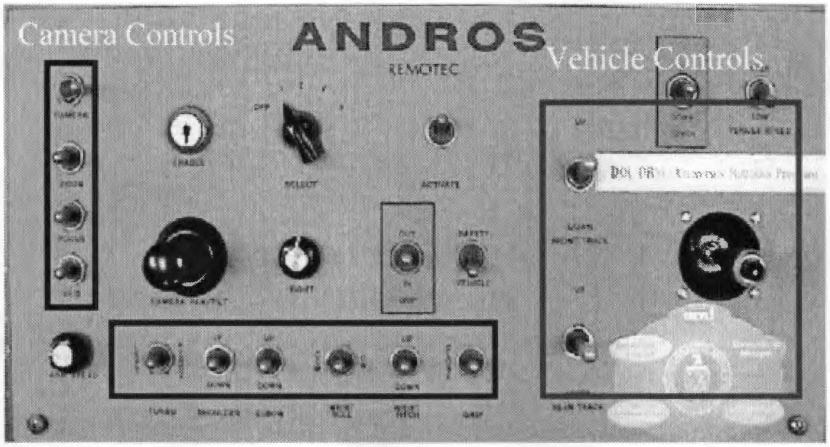


Figure 5.1: Front Panel of the Control Box

It is interesting to note that the Andros Mark VI used for the purpose of this research does not have speed control on its motors. Motor speed is set by a potentiometer which varies the current supplied to the motors, thereby controlling speed.

To communicate with the Andros, one has to have knowledge of the control signals. During the initial stages of the research no information was available regarding the protocol that the Andros used for communication. Hence some 'reverse engineering' was applied in that the control box was connected to a PC's serial port and data were sent to the computer. The computer read the serial port using a data acquisition software package named 'HHD Serial Monitor' **[HHD, 02]**. The task at hand was to then devise a means to effectively replicate the same data. This was done using a custom-designed Visual Basic program, which is explained in a later section. Subsequent to the 'reverse engineering' step, information about the protocol was obtained from Remotec Inc. This was helpful in comparing the two sets of data and verifying the results of the reverse engineering.

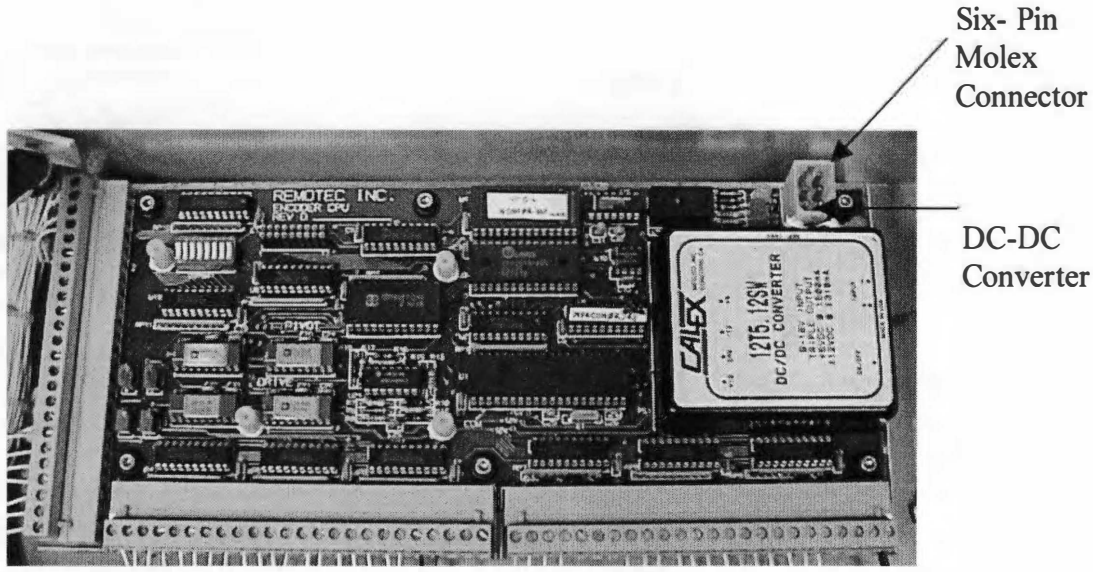


Figure 5.2: Encoder CPU for the Andros

The first challenge in reading data from the control box was making a connection between the controller chip and the PC. Figure 5.2 shows a picture of the encoder CPU for the Andros. Figure 5.3 depicts routing of the power supply and the RS-232 signal to the Andros.

Referring to Figure 5.3, the robot is powered by a 12 V DC power supply. This can come either from an on board battery or from an internal transformer which converts standard 110 V AC (from the wall) into 12 V DC. The 12 V DC is then fed to the encoder CPU board, to power the board. Power comes to the board via a standard 6-pin 'Molex' type connector. This connector serves a dual purpose in that it routes power into the board as well as transmits RS-232 from the board to the robot. To make a connection it is important to know the power supply lines as well as the signal lines and the common ground. This can be easily verified using a standard multimeter. To determine the RS-232 signal an oscilloscope can be used.

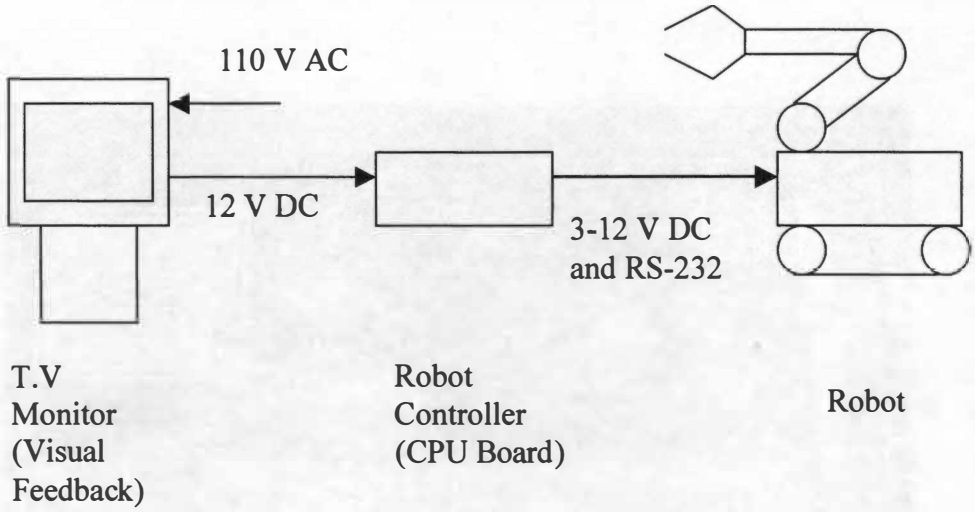


Figure 5.3: Power and Data Flow Schematic for the Andros

Figure 5.4 depicts the pin outs from the 'Molex' type connector used on the CPU board. From the pin-out it is clear that pin 6 transmits data to the robot. This is confirmed using an oscilloscope. The oscilloscope is set at about 5 volts per division on any channel. The oscilloscope ground probe is then connected to the common ground, and the signal probe is then attached to each of the pins on the connector. It is seen that pin 6 emits an RS-232 signal, this is shown in Figure 5.5. This signal has all the characteristics of an RS-232 signal. Similarly it can be verified that pins 1 and 5 are +12 V and +5 V DC respectively. Once the signals are determined the next step is to connect the control box to the PC's COM port (serial port) and then read data from the chip. This can be done using standard data acquisition software which would read data from any instrument or device connected to the computers COM ports. Examples of such software include the HHD- serial monitor **[HHD, 02]** and Hyperterminal **[Hyperterminal, 01]**.

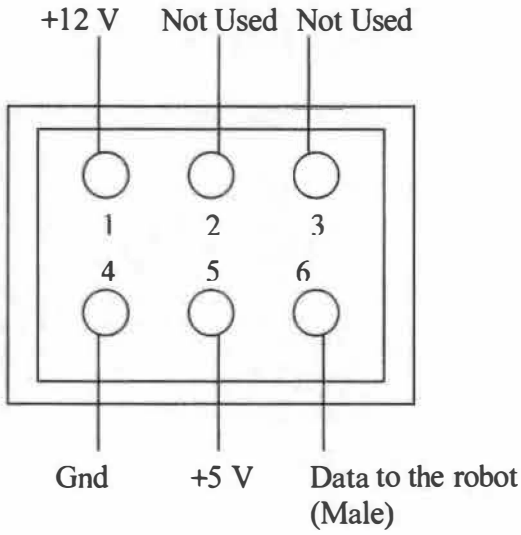


Figure 5.4: Pin Outs of the 'Molex' connector used on the CPU board

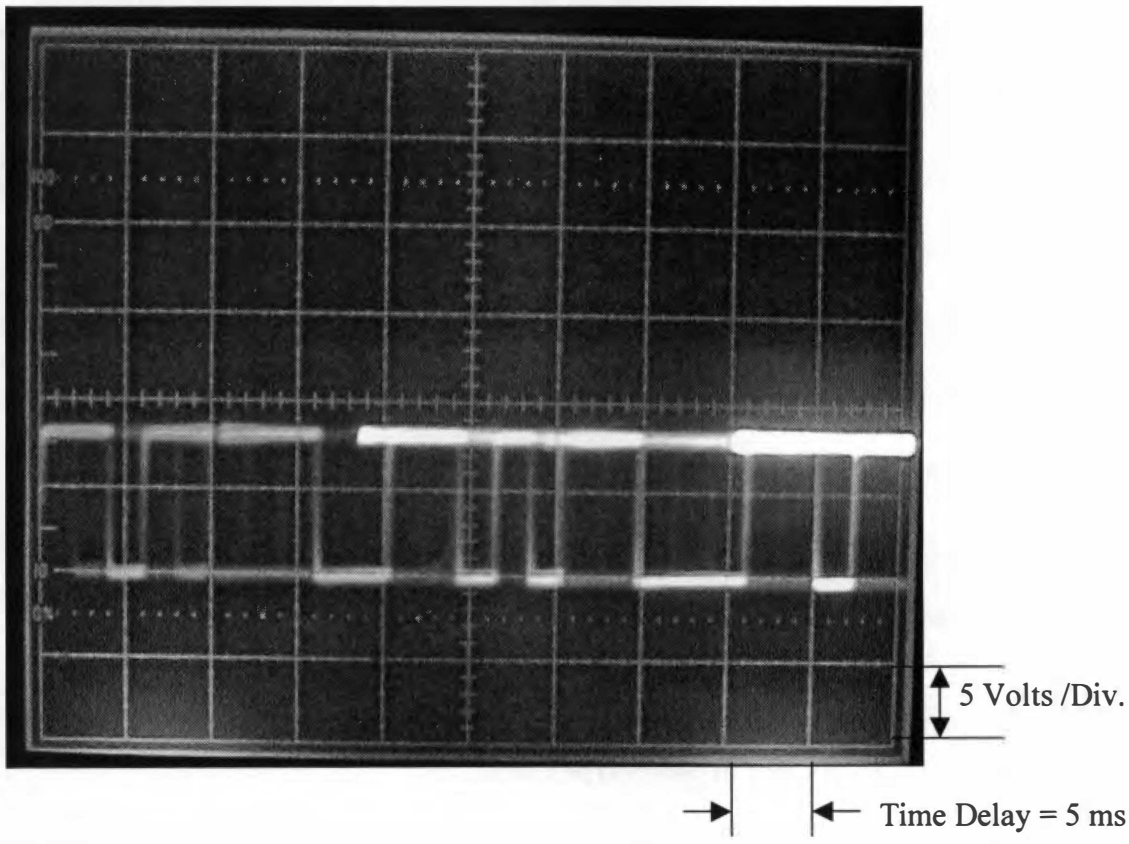


Figure 5.5: RS-232 Signal as seen on an Oscilloscope

The next step is the connection between the encoder CPU and the PC's COM port. Referring to Figure 5.4 and also to Figure 4.2 of the previous chapter pin no 6 of the Molex connector (Figure 5.4) is connected to pin-2 of the PC's com port. This is the receive pin (refer to Figure 4.2). Thus the 'send' on the controllers processor board is connected to the COM port's 'receive' pin. A standard twisted pair serial cable can be used for the connection. Length of the cable is discussed in a later section. This type of cable is easily available in computer hardware stores. The computer is now ready to read data from the board.

With the cable connected the serial port is first opened with an application like "hyperterminal.exe". HyperTerminal **[2001]**, is a program that was originally written to obtain diagnostic information from the modem. Hence it can access the serial port and can be used to 'open' the port for data communication. Once the port is open the serial port data acquisition software is initialized. The power to the controller is now switched on and the controller starts transmitting data the computer. Note that at this stage no commands have been issued. Also since the robot never sends any status signal back to the controller, the computer does not have to send any signal back to the robot. This kind of one way communication makes it easy to replicate the signals due to lesser computational burden involved.

5.2 Analysis of the data obtained

Once the data were read from the robot's CPU the important task of deciphering lay ahead. Example of data obtained are outlined in Appendix 2. It is seen that data to control

the Andros is in the hexadecimal format. It is observed that data is sent in a packet of 25 bytes, this is later verified upon cross examination with the protocol received from Remotec Inc., which is outlined in Appendix 1.

5.3 Connecting the robot to the PC

The essence of successful communication between the robot and the PC is the connection. After a brief insight into the PC's serial port and RS-232 the reader can understand this section better. The robot is connected to either of the PC's COM ports: (COM 1 or COM 2), over an RS-232 link. The PC is designated as the Data Terminal Equipment (DTE) and the robot is designated as the Data Communications Equipment (DCE). From the previous section it was inferred that signal transfer between the Andros and its controller was one way. There was no handshaking between the two systems. This means that the controller sends data packets to the robot without actually receiving any feedback from the robot. Even though the robot does not send a 'status' signal back to the controller, the controller is programmed to time out if no information is sent for a specified time. This time out period is about 100 milli seconds. For the link the serial port's transmit pin (pin 3) was connected to the receive from the 'Molex' connector's female pin. Figure 5.6 shows a schematic of the connection.

An often overlooked but extremely important factor effecting data transfer is the length of the cable. The cable must be chosen carefully taking into considerations the distance for which data are to be transferred. As mentioned earlier capacitance effects the

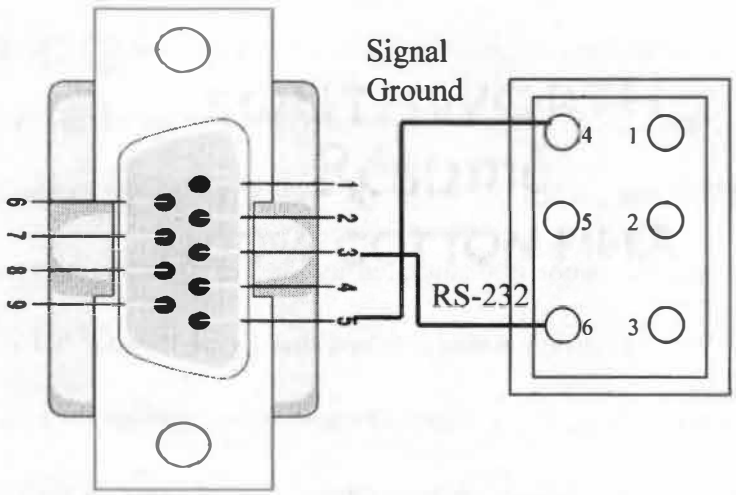


Figure 5.6: The Connection Between the PC and the Andros Robot

cable in several ways and care must be taken to choose a cable of correct capacitance. Axelson **[2000]**, defines an important relation to calculate the length of a shielded cable.

$$CL = (2500 - IC) / CC * 3 \qquad (4.1)$$

Where, CL is the cable length in feet, IC is the input capacitance of the receiver in Pico Farad (pF) and CC is cable capacitance in pF/ft. The cable chosen for this research was a standard twisted pair ethernet cable commonly referred to as Category 5 cable. This has a cable capacitance of 17 pF and a characteristic impedence of 100 ohms. Assuming that the receivers input is 100 pF and using equation 4.1 the length of the cable is calculated as follows:

$$CL = (2500 - 100) / (17*3)$$

$$CL = 47 \text{ ft.}$$

A cable of length 47 ft was used for data transmission.

5.4 Understanding the Protocol

A protocol is a set of rules that defines how the computer (or the robot controller) transmits data to the robot. It includes the rate at which the data are transmitted (too fast or too slow transmission will affect the functionality), the length of the data packet and the interval between two packets. Appendix 1 lists the protocol for the Andros Mark VI robot. For the Andros, when the robots firmware receives the proper communication status, called a green light, a positive communication link is established. When this is achieved any function on the robot can be activated. In order for the robot to achieve a

'green-light' situation the robot must be constantly receiving correct data from the controller (or computer). Determining what this data is, is what outlines the 'reverse engineering' discussed in section 5.1. These data are determined using the protocol information and comparing it with the data generated by the controller's CPU board. If the robot does not receive what it wants it times out very quickly (on the order of 100 milli seconds) and disables all its functions for safety reasons. For the Andros data are transmitted serially with 8 data bits and 1 stop bit with no parity. The data is transmitted at a rate of 1200 Baud with the pause between any two packets being 224 ms. However, this pause does not have any effect on the transmission and data can also be successfully transmitted without any pause. A 25 byte data packet is necessary to control the full range of the robot's functions.

Referring to Appendix 1 and Appendix 2 it is seen that the first two bytes are 'Start of Packet' bytes. These indicate to the robot that any information coming after these is data. The vehicle computer will start to store all data after receiving these two bytes. The last byte is the 'Checksum'. The check sum is a method to check errors and to verify that all the data are received correctly. The data sent after the 'Start of Packet' byte will be used only if the check sum is correct. The checksum is simply the addition of all the data bytes with only the least significant byte being sent. If the checksum is incorrect then the robot controller does not perform any function.

The protocol can be analyzed as follows:

For example consider the first data byte (Byte 1) this is in-fact the third data byte after the two 'Start of Packet' bytes. From Appendix 2, for Case 1 (no robot commands), this byte is Hex 30. In binary this translates to the number, 00110000, thus forming a whole byte.

The first four bits (0011) make up the Most Significant Nibble and the last four bits (0000) make up the Least Significant Nibble. The reader must note that in the protocol document the first four bits are referred to as 'LSB' and the last for bits are referred to as the 'MSB', even though they are nibbles (four bits constitute a nibble). The value of the Most Significant Nibble (MSN) is determined from the rules of the protocol, which indicate that the value of the MSN can be either 3 or 4 depending upon whether the value of the Least Significant Nibble (LSN) is lesser or greater than respectively.

Figure 5.7 depicts one data byte out of the 25. This is the 'neutral' byte for the movement of the vehicle tracks. From Figure 5.7 it is seen that the whole data byte comprises of two nibbles combining these two the value for the data byte is obtained as H30. It is seen that it is the LSB which controls the movement of various functions. In the 'neutral' byte the value of the LSB is 0000 (binary or hex). This is interpreted as a low by the computer on the robot and no function is performed. To have the robot perform 'Front Track Up', (byte 1 protocol document, Appendix 1), the value of the least significant nibble in this byte is changed from 0000 (Hex0) to 0001 (Hex1). This is read by the robot computer as a high and a signal is sent to the corresponding motor to perform the

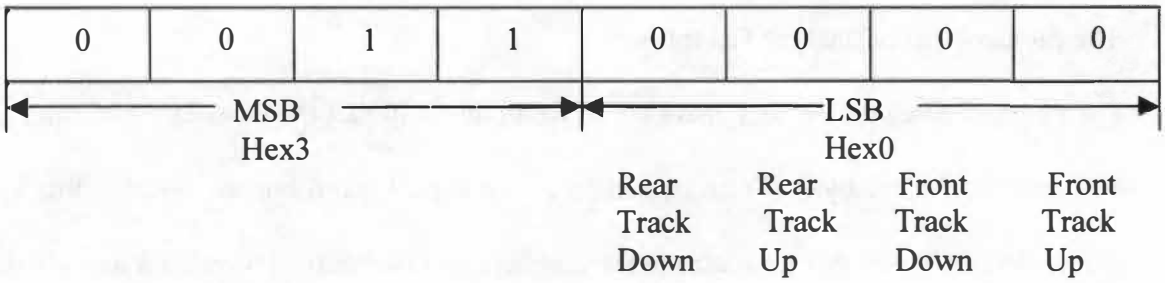


Figure 5.7: Triggering Bits for the Robot

indicated function. The robot continues to perform this function (front track up) until such time as this bit is reset to 0.

This analysis is done for all the 25 data bytes and a 'neutral' data set is constructed. This neutral set is the data that the robot needs to keep receiving continuously to achieve a 'green light' status signal as described previously. A schematic for such a data transmission is shown in Figure A1, Appendix 1. From Appendix 1 it is seen that this information packet is sent for 28 ms with the interval between any two packets being 224 ms. The idea here is to keep sending the status signal bytes and the check sum to the robot and then depending on which function is desired the corresponding bit is set to a high or a low. These functions are carried out by the computer program explained in the following section.

5.5 Programming

The computer programming described in this section primarily outlines the programming language used for accessing the serial port and transmitting data. This

program performs the same functions that the robot's control box would perform, by transmitting the same kind of data in a similar format. This section includes several code examples. These are intended to provide a full level of understanding to a novice programmer. This section is intended as an aid to future students using this experimental platform for control of the Andros and other similar mobile robots. Interested readers can also refer to Drexel **[2001]** and Cornell **[1998]**

Many programming languages for PC's support serial communications by including functions for configuring, reading and writing to serial ports. Serial port controls are also available from other sources like the Microsoft Windows API (Applications Programmers interface). The programming described in this section uses the Microsoft Windows platform. The code is written in the Visual Basic programming language. Even though most of the current Object Oriented Programming Languages (OOPL) support serial communication, Visual Basic was chosen due to its simplicity and the ability to build user friendly Graphical User Interfaces (GUI's) under the Windows platform.

5.5.1 Introduction to Visual Basic and Microsoft Comm. control

Graphical User Interfaces (GUI's) have revolutionized the microcomputer industry. They demonstrate that a picture is worth a thousand words. Instead of the cryptic C:> prompt that's DOS users have long seen, a desktop filled with icons is presented **[Cornell, 98].** This kind of visual programming means that you 'paint' the look of your program and the tie the overall processing with the procedural code. Visual basic and its follow up applications pioneered and refined such type of visual programming. A

detailed tutorial on the Visual Basic 6.0 Programming Language can be found in Cornell's " Visual Basic 6.0 From the Ground Up" **[Cornell, 98]**. Interested readers can also access a variety of tutorials available on this subject on the World Wide Web like MacMillan **[2002]**. These will help the reader to better understand the process of building a graphical user interface (GUI).

The programming environment for this research covered two broad areas which were the following:

1. Transmitting serial data over the serial port to the Andros.
2. Building a user friendly interface.

The first step in the process is to access the serial port using the program (Visual Basic in this case). To do this Microsoft has incorporated the 'MSComm ' control in its Visual Basic Applications. MSComm is Visual Basic's custom control for serial communication. It provides serial communication for the application by allowing transmission and reception of data through a serial port. The Communications control provides an interface to a standard set of communications commands. It allows you to establish a connection to a serial port, connect to another communication device (a robot, for instance), issue commands, exchange data, and monitor and respond to various events and errors that may be encountered during a serial connection. Like other controls MSComm has associated properties. The properties relate to configuring ports, transferring data, use of handshaking signals and identifying the control.

5.5.2 Initializing the application

The first step in configuring a serial port is to open the Visual Basic programming environment. This is done from the 'Start – Programs' menu on Microsoft Windows and opening a new project by clicking on the 'Standard EXE' icon. Visual Basic also has other environments like the 'Activex EXE'. These are mostly used in building internet applications and are not discussed here. A new project dialog box is shown in Figure 5.8.

Double clicking on the 'Satandard EXE' icon takes the user to the Integrated Development Environment (IDE). This is where the entire event driven programming is done. Visual Basic is extremely versatile due to its event driven nature. The idea of event driven programming is that the application is coded so as to respond to certain events. These events can be either user driven events or system events. A user event is triggered upon a command from the user; for example a mouse click, or by pressing a key on the key board. A system generated event is one that takes place within the application, when it is first loaded into memory, but before it appears on the screen. Figure 5.9 describes the Visual Basic Integrated Development Environment

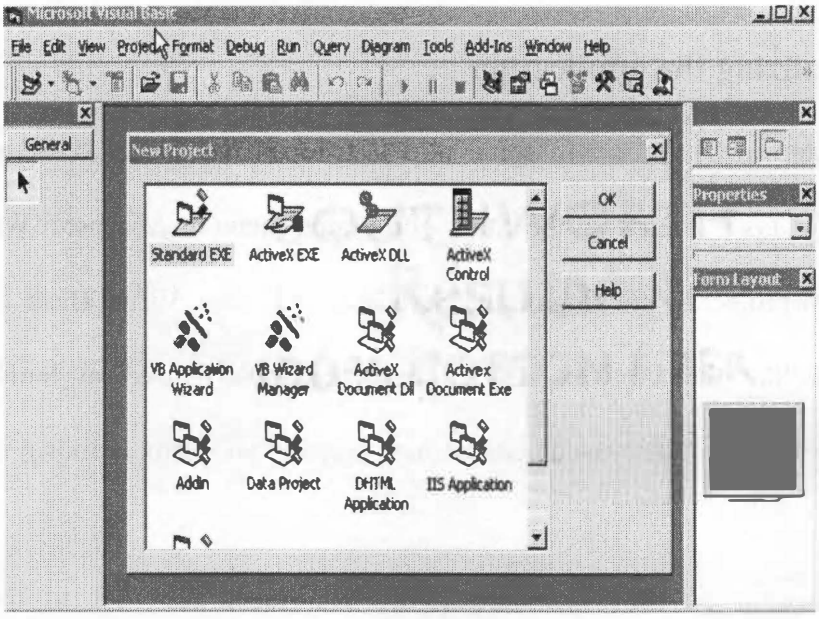


Figure 5.8: New Project Dialog Box

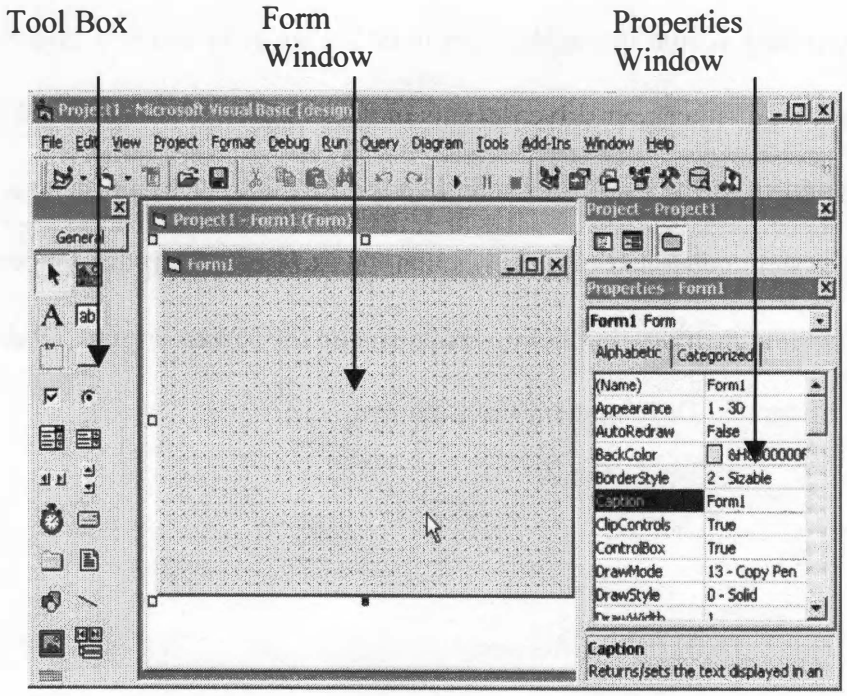


Figure 5.9: Visual Basic Integrated Development Environment (IDE)

Referring to Figure 5.9, the window with the grid of dots is called the 'Form' window. This is the window where the command buttons will be added to customize the applications **[Cornell 1998].** To Add MSComm control to the application click on project and then select 'components', then under controls tab 'Microsoft Comm. Control 6.0' is selected to add comm control to the application. This is illustrated in Figure 5.10.

The next step is to set the data transmission properties namely the baud rate, parity its, data bits and parity bits. From the protocol information outlined in Appendix 1 it is seen that the baud rate for transmitting data to the Andros is 1200 baud with no parity, 8 data bits and 1 stop bit. This format has been discussed earlier under section 5.3. These settings are entered under the 'settings' in the properties dialog, as shown in Figure 5.11.

Once the transmission rate and number of bits are set the portion of code that actually opens the serial code can be entered. This is done in the code design area of visual basic. The code that opens the serial port for communication and initializes the port settings is shown in Example Code 5.1

Example Code 5.1

```
(1)Private Sub Form_Load()
 '  Use COM2
(2) MSComm1.CommPort = 2
'Define COM port Settings : 1200 baud, no parity, 8
data  bits, 1 stop bit
(3)MSComm1.Settings = "1200,N,8,1"
' Open the port
(4)MSComm1.PortOpen = True
(5)End Sub
```

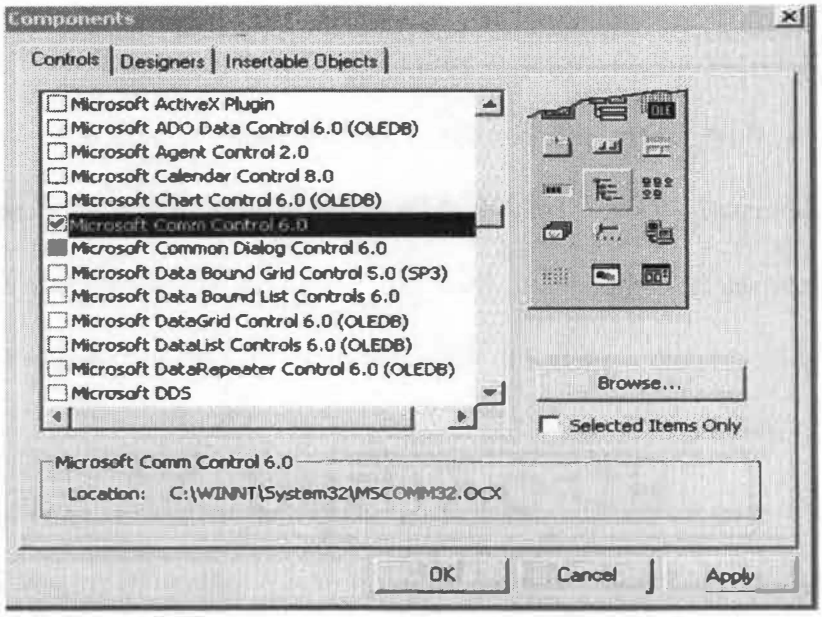


Figure 5.10: Selecting MSComm Control

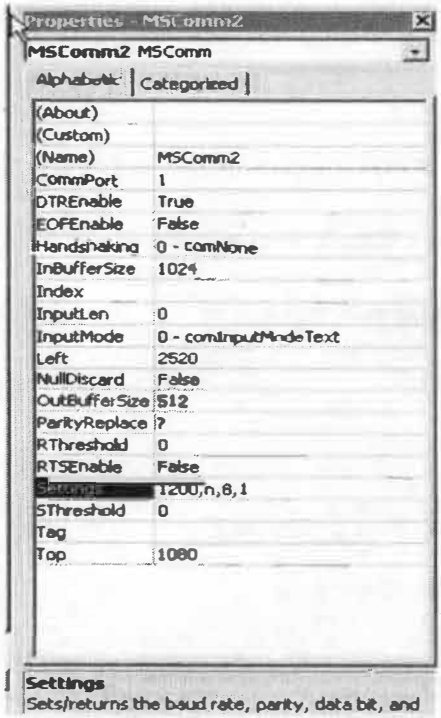


Figure 5.11: COM Port Settings

Referring to the listing, line 2 sets the Com port in the computer to '2'. Line 3 sets the data transfer format and line 4 opens the port for communication. Once the serial port is open for communication data can be sent or received. A detailed listing of the code can be found in Appendix 3. Interested readers can also refer to Axelson **[2000]** for a complete understanding of serial communication under the Visual Basic programming environment.

5.5.3 Sending Data:

From section 5.3 and the protocol document listed in Appendix 1, it is clear that the robot has to receive data in a particular format (25 bytes of data in Hexadecimal, with the pause between each packet being 20 ms). To send data in this format the code shown in Example Code 5.2 was used.

Example Code 5.2

```
(1)' Initiates timer control. This runs the events
under the '_Timer' event
(2) Private Sub StartTransButton_Click()
(3) Neutralize
(4) ' Time is defined in milliseconds.
(5) Timer1.Interval = 1
(6) End Sub
(7) ' Stops Data Transfer.
(8) Private Sub StopTransButton_Click()
(9) Timer1.Interval = 0
(10) End Sub
(11) ' Runs the code when the 'Start Transfer Button is
     clicked. 'Sends Neutral Data packets.
(12) Private Sub Timer1_Timer()
(13) ' Type Definition statements
(14) Dim k As Long
(15) Dim jj As Integer
(16) Dim p As Long
(17) For jj = 3 To 24
(18) k = k + bytData(jj)
```

```
Next
(19)' Set byte no 25 to the checksum
(20) bytData(25) = LOBYTE(k)
(21) buffer = bytData()
' Define the MSComm function to writed data to the
buffer
(22) MSComm1.Output = buffer
(23) End Sub
```

Referring to Example Code 5.2, Line 1 is a comment line. Line 5 initiates the timer control. The timer function is used in this case because it is best suited for applications where the code has to be executed many times in regular intervals, communication with the Andros being one such case. In the actual form this code is linked to a button which starts transmitting data when it is activated; this is an example of a user event. To stop transfer of data, the timer interval is set to 0. This again is linked to a button which would set the interval to 0 and stop any data transfer. The interval property defines how often the code in the '_Timer' event will be executed. This is measured in milliseconds. Hence if you want the code to be executed every second, then the timer will have to be set to 1000. Another very important piece of information in the protocol is the Checksum. Without receiving the proper check sum the robot will not respond to any request from the controller. The checksum is simply an addition of all the bytes of data, with only the Least Significant Byte (LSB) being sent. Line 20 calculates the checksum using a function called 'Lobyte'. The 'Lobyte' function returns the lower 8-bit byte from a low word of a 32-bit long integer. This is similar to the 'LOBYTE' in the 'C' programming language. Finally all the data is stored in the buffer and this is then written to the Com port, line 22 details this function.

The data stored in the buffer is the 'Neutral' Data set described earlier. The neutral set itself is constructed and stored separately in a module. A module is usually written for applications where the it is desirable to reuse the code. The module This is outlined in the following example code 5.3.

Example Code 5.3

```
Function Neutralize()
'Setting up an Array for the Robot Data
For ii = 1 To 25
bytData(ii) = &H30
Next
'Changing the bytes according to the protocol
bytData(1) = &HF1
bytData(2) = &HF1
bytData(4) = &H36
bytData(7) = &H31
bytData(8) = &H38
bytData(9) = &H32
bytData(13) = &H39
bytData(15) = &H38
bytData(16) = &H31
bytData(17) = &H37
bytData(18) = &H41
bytData(19) = &H32
bytData(20) = &H35
bytData(21) = &H34
bytData(22) = &H33
'bytData(25) = &H69
End Function
```

The 'Neutral' set was constructed by first creating an array of 25 elements and setting them to an initial value. This was set to be Hex 30, represented in Visual Basic Syntax as &H30. The symbol '&H' indicates that data is in the Hexadecimal format. The number 30 was chosen at random, however, a number of bytes have 30 as their initial value making this selection practical. Once the array was constructed, the bytes were changed according to the protocol information which was obtained. For example, from Appendix

2, it is seen that Byte No 1 has a value of 'F1', hence its value was changed to '&HF1' in the array to obtain the first byte of the 'Neutral' set. Similarly changes were made to all the bytes depending upon their values in the protocol. This set is the set that achieves the 'green light' condition in the robot. Once this is successfully completed different robot functions can be implemented by changing corresponding bytes.

5.6 Summary

Understanding the data transfer between the robot and the computer is very critical to successful data communication. The user should now have a clear picture of the steps involved in using the PC to control the Andros. In the Visual Basic programming environment the development of the Graphical User Interface (GUI) takes place in parallel with the programming code. This is described in the next chapter.

Chapter 6

Discussion and Results

This chapter discusses the result of this research. The ultimate result of this research is the development of a user-friendly operator interface that controls the robot in a manner similar to the present controller. The building of the Graphical User Interface is a major step in this direction. The chapter describes the process of building a Graphical User Interface using Visual Basic 6.0. Mouse and Keyboard control are also explained. The results of this research are more subjective in nature than objective.

6.1 Importance of the Graphical User Interface (GUI)

The User Interface for the program code was built using the Visual Basic 6.0 Programming Language. As explained in the previous chapter Visual Basic has revolutionized the development of Windows applications by imparting user controls instead of long and complex code. The code in Visual Basic is 'Event Driven', meaning that it 'hides' behind the application. Simply put, Visual Basic programming for Windows applications is a very efficient and fun way to develop user friendly interfaces. Interested readers can refer to Cornell **[1998]** for a complete tutorial on the Visual Basic 6.0 programming language.

The main reason for developing a GUI for this application was to impart operator friendliness to the control of the robot. The user no longer has to deal with 'switches on a box', instead the same controls are imparted using a PC interface. Having the PC communicate with the robot also had the additional advantage of flexibility in terms of achieving web based and wireless control as well as real time monitoring of the system.

6.2 Building the User Interface

The main objective of the interface is to essentially perform the same functions that the controller box does. With this in mind, any interface built had to be both simple and functional. Using Visual Basic, a number of designs for the interface are possible, limited only by the time available and the programmer's creativity. The interface described here was the first one developed. It is still undergoing significant changes and constant development.

The first step in building an interface is to initialize the application. This has been outlined in section 5.4.2 of the previous chapter. Once this is done, a form window is opened in a new project. The form is the area where user controls like buttons are added. This is shown in Figure 6.1. The figure shows the form created to add the 'Start Transfer' control to the interface. This is added in the form of a button. This button when clicked performs the same function as the 'On' button on the controller box would do, viz transfer the 'Neutral' data set to the robot. Once the button is added its properties (color, caption etc.) can be changed in the properties window as shown in Figure 6.1

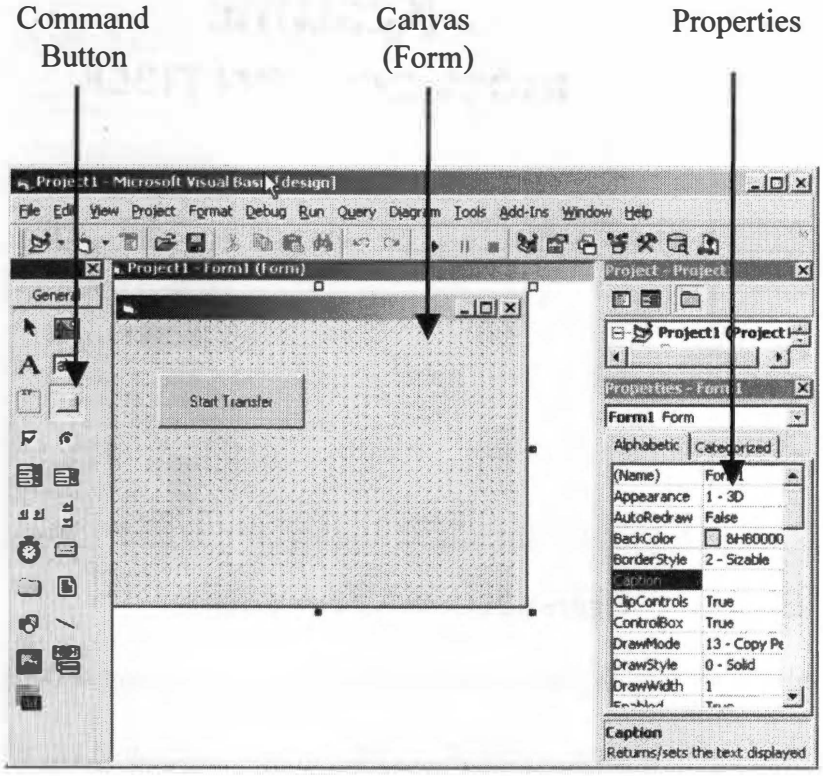


Figure 6.1: Adding a Command Button

The next step is adding controls to the form making the form respond to user actions; This is the essence of Visual Basic Programming. Visual Basic objects can recognize many different events. For example, if a user clicks on an area on the screen, a specific action may need to be performed (such as a moving a robot arm). This is where the programming part comes in. The point is that though Visual Basic objects can recognize many different events. The objects will basically sit there unless a piece of code is written to tell them what to do when the even happens. This means that for every event that the user wants Visual Basic to respond to, an 'event procedure, code has to be written.

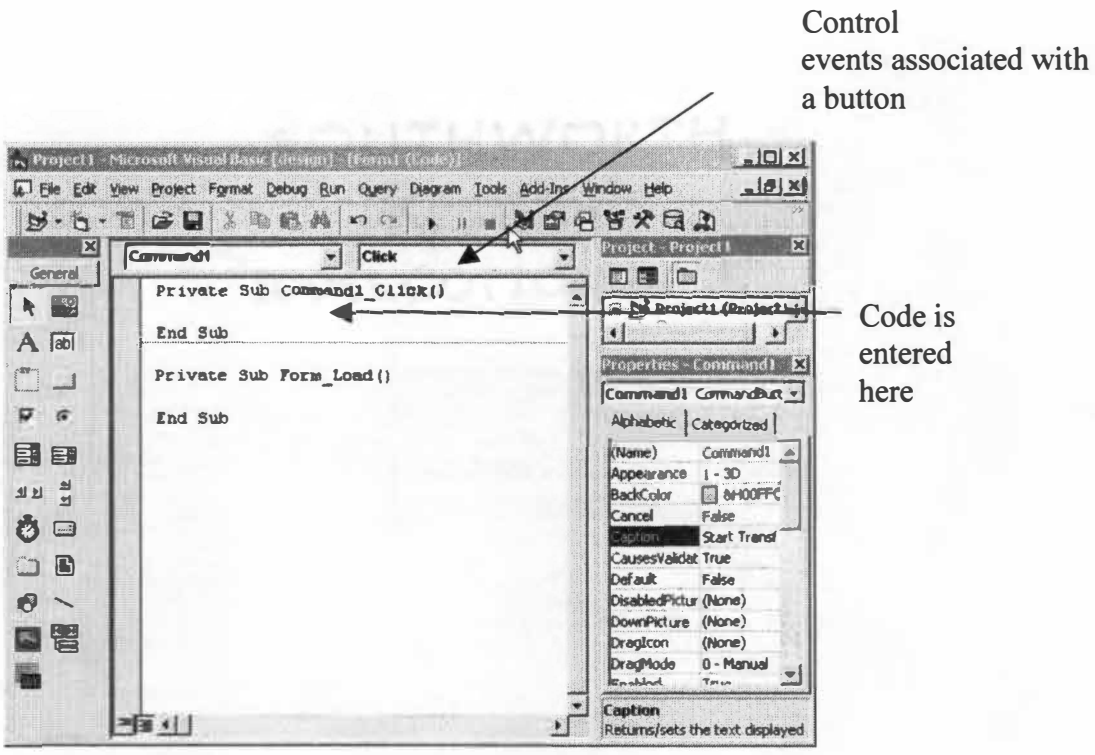


Figure 6.2: Code Window Setting

Event procedures are nothing more than the lines of programming code that tell Visual Basic how to respond to a given event **[Cornell, 98].** To write an event procedure code for the 'Start Transfer' button depicted in Figure 6.1, the button is double clicked. This takes the user to the 'code window'. This window is where the actual code is entered which tells visual basic how to respond to an event. This is shown in Figure 6.2

At the top of the form are two drop down list boxes. These contain a list of all events that a form can recognize. The left hand button displays the controls for the command button and the right hand button controls the events associated with a button. These events are user defined. Visual Basic has 31 events to chose from **[Cornell, 98]**. The

button actually takes the user to the 'event procedure template'. The following is a general form for the event procedure template to add a command button

```
Private Sub StartTransButton_Click()

End Sub
```

The idea here is to set up the button to transmit data. To do this the following piece of code is added between the first two lines.

```
Private Sub StartTransButton_Click()
Neutralize
'Time is defined in milliseconds.
Timer1.Interval = 1
End Sub
```

This line of code has been explained previously in section 5.4.3. If the program is now run, the button is displayed. The button is now active. Upon clicking the button the computer starts transmitting data to the robot. This is similar to the 'On' switch on the robot controller. The computer continues to send the 'Neutral' set as long as the program is running.

Once the procedure to add control to a command button is clear, the same can be done for the motion of the various joints and robot drive. For example, for the 'gripper open / close' event, two buttons are created and named 'Gripper open' and 'Gripper close', respectively. To add controls to these buttons code is added in the code window. This code changes the value of byte 12 [Refer to protocol document, Appendix 1], to Hex 31 (for gripper open) and Hex 32 (for gripper close) respectively. When the program is run, the user first clicks on the 'Start Transfer' button to start sending neutral data. To open the

gripper the user clicks on the 'Gripper Open' button and to close the Gripper the user clicks on the 'Gripper Close' button. This type of 'click' event has a disadvantage when used in applications similar to control of the Andros, in that the event is triggered off once the button is clicked and remains on. Translated into the context of this research it means that once the 'Gripper Open' button is clicked the gripper keeps opening, this is highly undesirable. Hence the importance of Mouse and Key board controls, discussed next.

6.2.1 Adding Mouse and Keyboard control

As discussed in the previous section, imparting functionality to the interface was one of the main objectives of this research. Even though the 'click' event procedure is a good way for the user to understand how to add code to the application, for this type of application it is inadequate. To improve upon this the 'Mouse Down' and 'Mouse Up' events can be used. As the name suggests, the 'Mouse Down' event is triggered when the mouse is held in the 'down' position or in other words the left mouse click button is kept pressed. The event then remains 'on' as long as the mouse is in this position. The event stops when the 'Mouse up' is triggered, that is when the left mouse button is not being clicked anymore.

Adding code to this procedure is similar to adding code to any event procedure. The button to which this event needs to be added is clicked and the code is added by selecting the 'Mouse Down' event from the event selection drop down menu, this is shown in Figure 6.3.

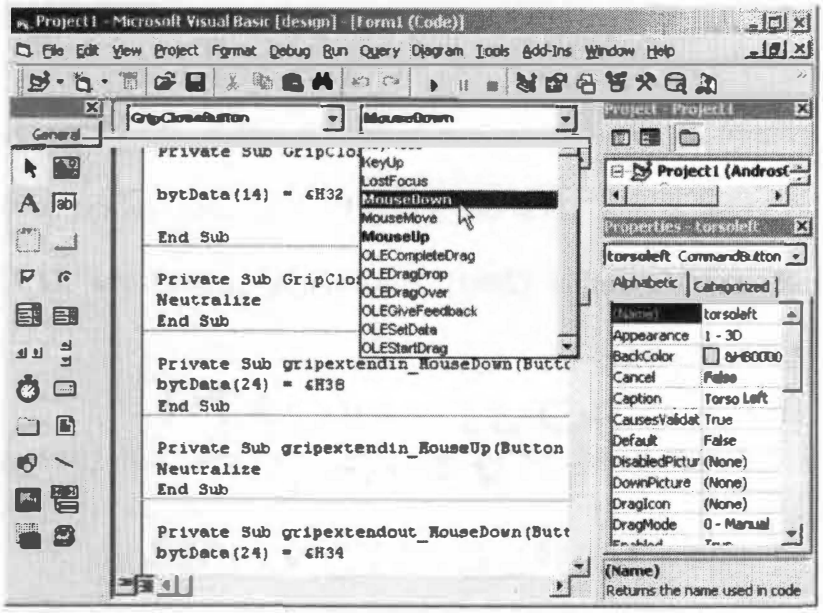


Figure 6.3: Adding Mouse Control

The example shown in Figure 6.3 is for adding mouse control to the 'Gripper Close' function. However, it should be noted that every 'Mouse Down' event has a corresponding 'Mouse Up' event. Failing to add a 'Mouse Up' event in this case will result in the gripper opening continuously, which is undesirable. Ideally the gripper should open in steps upon a single mouse click, and this can be achieved by adding a 'Mouse Up' event.

The functionality of the interface can be further improved by adding Key board control to the application. This would enable two events to be controlled at the same time. For example if the operator wishes to control two events (wrist pitch and gripper open), this would not be possible with just a mouse click, (the operator would have to move the mouse constantly between buttons). With key board control however, , fixed keys can be

used to control events, for example: the 'up arrow' key can control vehicle movement in the forward direction and the down arrow can control reverse movement.

Key board control on Visual Basic can be applied either through the 'Key Up', 'Key Down' and 'Key Press' events. These are simple to understand and explained thoroughly in Cornell **[1998]** and Macmillian **[2002].** For example consider the 'Key Down' event. This has the following syntax:

```
Private Sub Command1_KeyDown(KeyCode As Integer, Shift
As Integer)
End Sub
```

Referring to the syntax, the 'Command1' specifies the name of the command button which controls the event. 'Key Down' is the name of the event and the 'Key Code' refers to the name of the key with which the event needs to be controlled. The 'Key Code' parameter tells the computer what key was pressed, and the event linked to this key is then performed. The key codes for the various possible characters follow the ASCII/ANSI codes only for the A-Z and 0-9 on the key board. For the remaining codes whether for the arrow keys or the function keys, or the numeric keypad, the needed constants are given in the library of constants supplied with Visual Basic. Interested readers can refer to the Visual Basic Library available with the software or to Cornell **[I998]**. Most of the constants are fairly straightforward: they take the form of 'vbkey*x*'.

For example, suppose the user wants to control the robots gripper open / close using the 'Page Up' key on the computers key board, the following piece of code can be used:

```
Private Sub GripOpenButton_KeyDown(KeyCode As Integer,
Shift As Integer)
If KeyCode = vbKeyPageUp Then
bytData(14) = &H31
Else
Neutralize
End If
End Sub
```

Line 3 of the code assigns the key code the value of 'vbPageUp', this indicates to Visual Basic that when the program is run the code occurring after that has to be performed. In this case that code changes the value of byte 12 (gripper open / close, refer to Appendix 1), from Hex 30 to Hex 31. This would then open the gripper. However, It should be noted that this triggers the 'gripper open' event and the gripper will continue to open until it is told to stop. To do this the 'Key Up' event is added. This stops the event associated with the key down event, meaning that when the key is not pressed anymore the event associated with it will cease to occur.

These type of key board controls can be added to the entire range of the robot's functions to make the control of the robot easier. The advantage being that more than 2 events can be controlled at the same time.

6.3 Result

The initial GUI that resulted from this research is shown in Figure 6.4. This GUI incorporates all the basic functions of the control box. At the time of writing this thesis

the GUI was still under development. As stated before the range of possible designs is limited only by the creativity of the programmer and the time available.

The entire research resulted in using the PC to control the robot instead of the existing control box. The control of the robot before and after the research are depicted in Figure 6.5. With Figure 6.5 (a) depicting the type of control that existed before the research and and Figure 6.5 (b) depicting the present control scheme. From Figure 6.4 it is clear that the existing controller was successfully replaced with a PC. This leads to new possibilities in terms of control, such as, modular control, sensor integration, path planning, web based and wireless control. some of these are discussed in the next chapter.

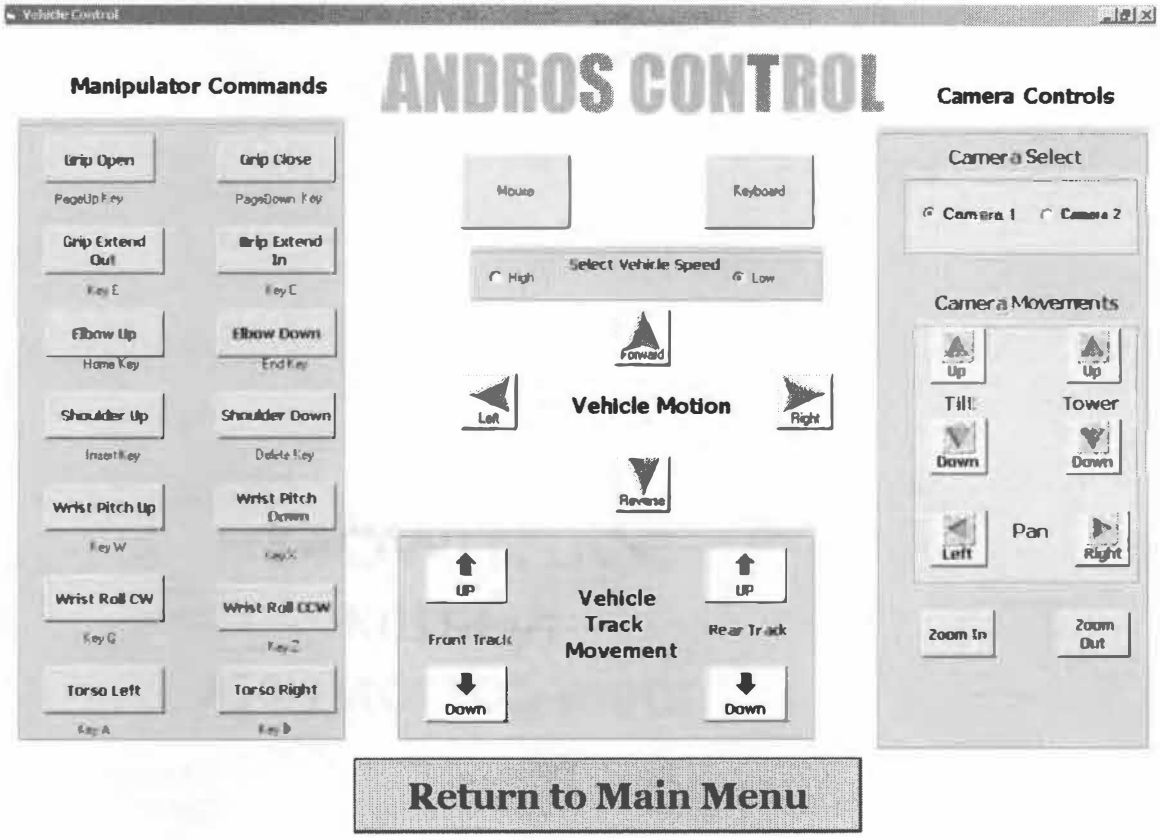
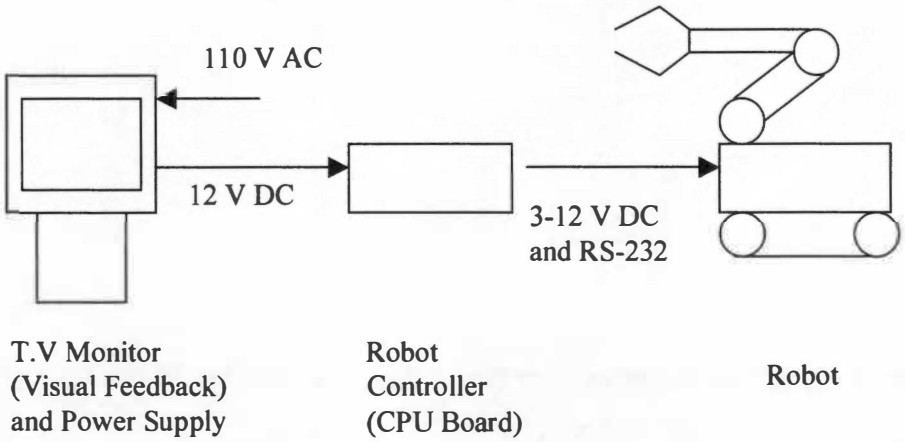
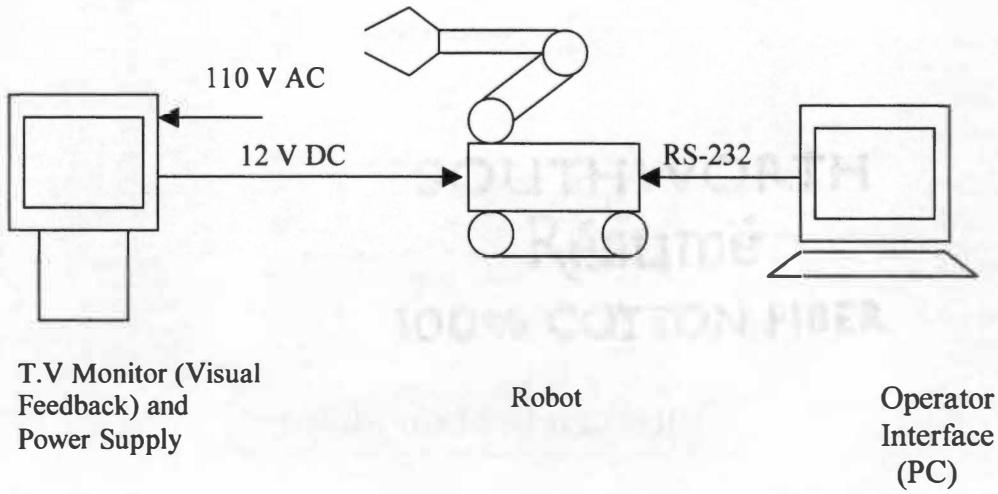


Figure 6.4: The Graphical User Interface (GUI)



(a)



(b)

Figure 6.5: Andros Control: Before and After

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Computer-based control of robots has always been a challenging task. This research comprehensively overcame this challenge by achieving PC-based control of the Andros hazardous duty mobile robot. The interface built, in its present state, is able to achieve the same degree of functionality that the controller previously had. The Andros itself has limited sensory capability and was designed to be teleoperated, meaning that a human was always in the loop. This kind of low-level teleoperation is typical among mobile robots like the Andros. However, with increasing levels of automation among telerobots autonomous control assumes prime importance.

The first step towards achieving autonomous control for a mobile robot is to have some means of computer-based control for the various functions in the robot. Most mobile robots incorporate this in the form of embedded controllers. However, these have limited computing power and functionality. PC-based control has the advantage of faster computing speeds as well as flexibility in terms of adding additional components like modular control, web based control and wireless control. This thesis demonstrated that the PC can be effectively used to incorporate functional computing power to a mobile robot.

Specifically for the Andros this type of control was achieved using serial communication and the RS-232 protocol. The use of Visual Basic programming language for such an application was successfully demonstrated. In robotic applications involving unsophisticated teleoperated vehicles, with a human operator providing intelligence to the system, controlling the functions of the robot is the primary function of the computing power that is available.

7.2 Future Work

Future work in the area of PC based control can include modular control and web based control. These are discussed here in brief.

7.2.1 Modular Control

Anderson **[1993]** describes the development of the Sequential Modular Architecture for Robotic Teleoperation (SMART) for constructing robot and teleoperator systems from modular components. SMART is a real-time icon-based control architecture for telerobotics. The architecture allows rapid synthesis of real time devices. Real-time module exchange and parameter tuning enable the end user to customize the system. The architecture is both open and modular. Distinct modules are defined for each sensor, actuator, input device or kinematic element. The real time component of SMART requires a multi-threaded operating system which supports for POSIX threads (p-threads). The hardware consists of commercially available CPU's and processor boards

running on a VME bus with a UNIX- based operating system (VXworks). The software consists of libraries of 'C' routines. SMART is based on the two-port network theory.

Anderson **[1996a]** represents systems by networks consisting solely of passive (non energy generating) elements. Each module in the SMART system has a network equivalent, either a one port of two port network. When joined together the modules form a complete circuit. The two port networks are a common tool for describing bilateral teleoperation. Figure 7.1 depicts some modules using SMART. The modules shown are for the world space controller for the PUMA robot. Similar modules can be built for the Andros. Examining the suitability of SMART, with regard to implementing it onto the Andros can be a first step in actually using SMART to control the Andros. Outlining the system requirements can be a valuable second step. The system configuration for SMART is outlined in Figure 7.2.

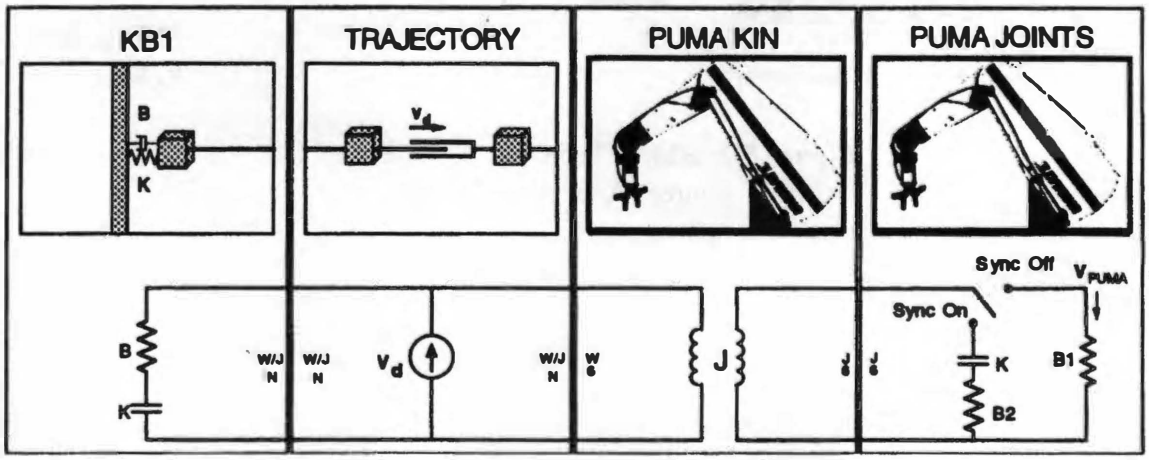


Figure 7.1: SMART Modules
Source: Anderson **[1993]**

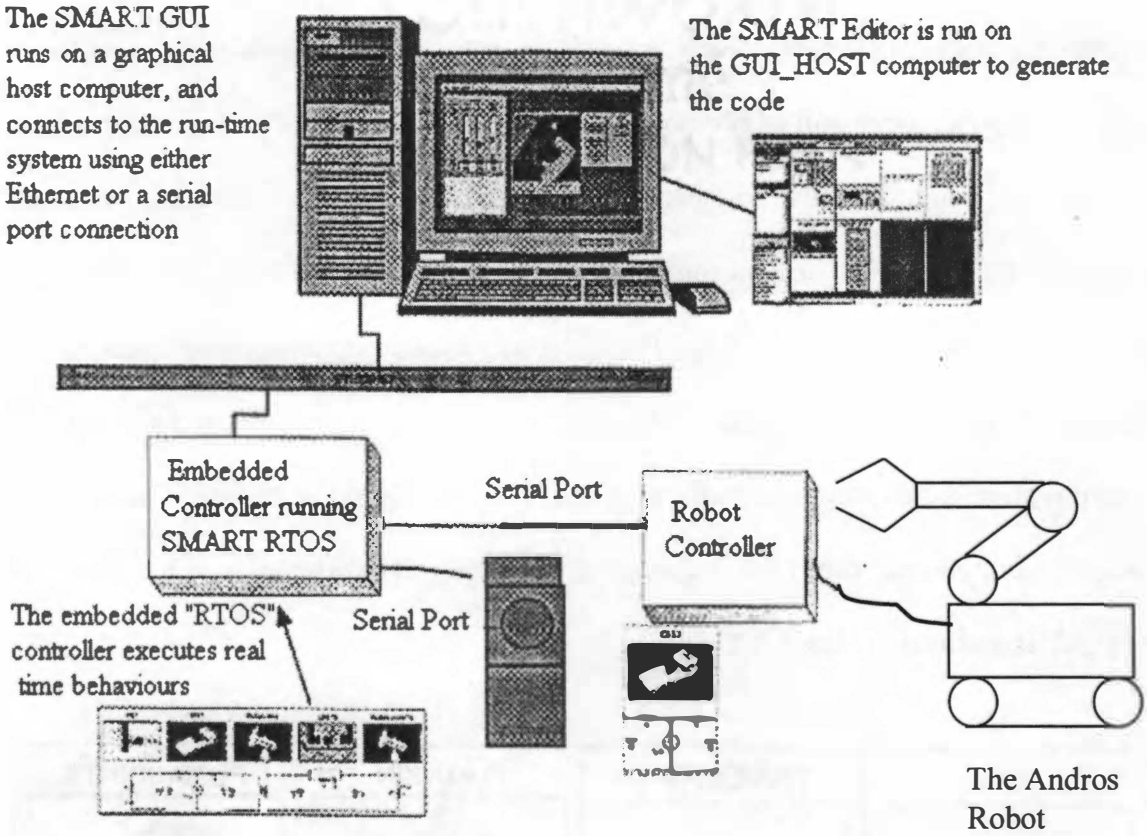


Figure 7.2: SMART: System Configuration
Source: Anderson **[2001]**

Once the suitability of SMART to the Andros is determined, distinct modules will have to be built by studying the existing modules for the PUMA and KRAFT robot manipulators.

7.2.2 Web Based Control

The advent of the internet and new developments in networking have ushered in a new area of telerobotic control namely Web Based control. Over the years a variety of web based control applications have been developed are used in fields varying from space telerobotics, [**Otmane, 00**] to pure interactive amusement and instruction **[Remotebot, 01]**. A web-controlled robot has also been discussed by Malinowski **[2002]**.

To implement web control on the Andros a Network Server needs to be incorporated. The program to control the robot such as the one described in this thesis can then be uploaded on to the server for access via the World Wide Web. Such a scheme is outlined in Figure 7.3. Taylor **[1995]** has outlined such a system for the Web Controlled telerobot developed at the University of Western Australia. For the Andros web control could prove to be a useful tool in training the operator. This would mean that more operators can be trained and familiarized with the robot's control environment without actually having to use the robots and with distance no longer being an issue. This can also lead to the development of greater sensory capabilities for the Andros. Even though there is no substitute for real-time hands on training, web control would, significantly ease the operators burden when actually operating the robot. With increasing importance on

reduced down times for performing remotely controlled tasks, this could result in a significant savings, both in terms of cost and manpower.

This thesis has successfully demonstrated control of a mobile robot using the RS-232 communications protocol outlining innovative ideas for future research.

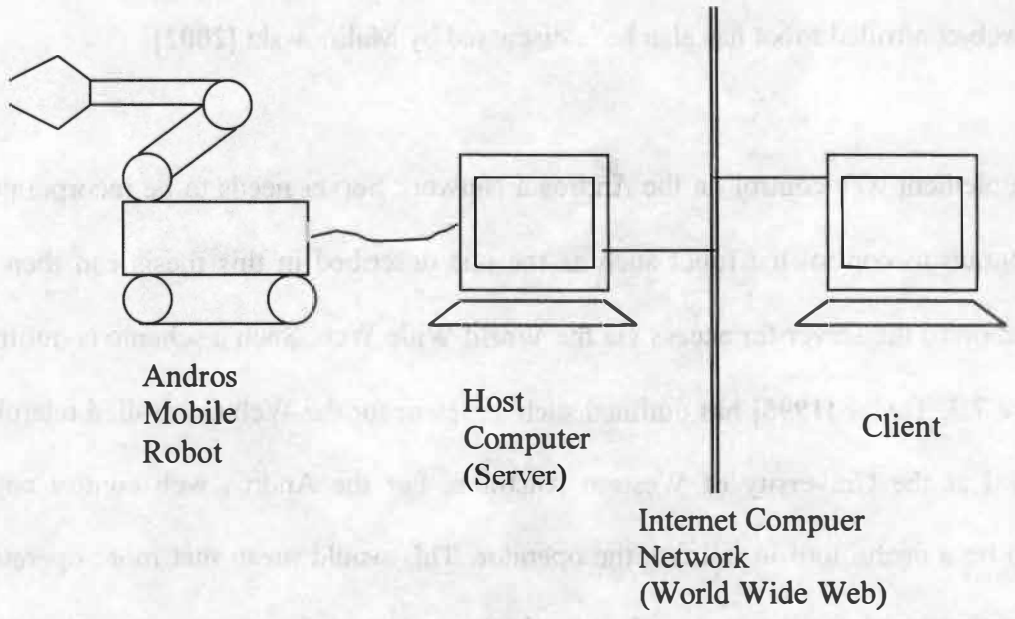


Figure 7.3: Web Based Control Scheme for the Andros

References

[Anderson, 93] Robert J. Anderson, "SMART: A Modular Architecture for Robotics and Teleoperation", *IEEE International Conference on Robotics and Automation*, vol. 2, pp 416-421, 1993.

[Anderson, 94] Robert J. Anderson and Brady Davies, "Using Virtual Objects to Aid Underground Storage Tank Teleoperation", *IEEE International Conference on Robotics and Automation*, pp 1421-1426, 1994.

[Anderson, 96a] Robert J. Anderson, "Building a Modular Robot Control System using Passivity and Scattering Theory", *IEEE International Conference on Robotics and Automation*, pp 698-705, 1996.

[Anderson, 96b] Robert J. Anderson, "Autonomous, Teleoperated, and Shared Control of Robot Systems", *IEEE International Conference on Robotics and Automation*, pp 2025-2032, 1996.

[Anderson, 01] Robert J. Anderson, "SMART 0.8: Developers Guide", R.J. Anderson, 2001.

[ARC, 01] ARC Electronics, "RS-232 Data Interface – A Tutorial", http://www.arcelect.com/rs232.htm [Online], December 2001.

[Axelson, 00] Jan Axelson, *Serial Port Complete: Prgogramming and Circuits for RS-232 and RS-485 Links and Networks.* Madison, WI: Lake View Research, 2000.

[Brokk, 02] Brokk, "Brokk demolition and bricking solutions", http://www.brokkinc.com/index.html [Online], February 2002.

[Cornell, 98] Gary Cornell, *Visual Basic 6 From the Ground Up.* Berkeley, CA: Osborne / McGraw Hill, 1998.

[Cybermotion, 01] Cybermotion, "Nuclear and Hazardous Materials Inspection", http://www.cybermotion.com/html/hazmat.html [Online], February 2002.

[Drexel, 01] Drexel University, "Serial Port Interfacing Using Visual Basic', http://www.pages.drexel.edu/~bns23/tutorial.html, December 2001

[Hayward, 86] Vincent Hayward and Richard P. Paul, "Robot Manipulator Control under Unix RCCL: A Robot Control 'C' Library", in *The International Journal of Robotics Research,* vol. 5, no. 4, pp 94-112, Winter 1986.

[HHD, 02], HHD Software: Serial Monitor, http://www.hhdsoftware.com/index.html [Online], January 2002.

[Hyperterminal, 01], Hilgraeve, "Hyperterminal", http://www.hilgraeve.com/htpe/, [Online], December 2001.

[Macmillan, 02] Macmillan Computer Publishing, "Using Visual Basic 6", http://davidtools905.hypermart.net/data/vb6/6bf8a475.htm [Online], January 2002.

[Malinowski, 02] Aleksander Malinowski, "Bradley Web Controlled Robots", http://cegt201.bradley.edu/~olekmali/projects/telebot/ [Online], January 2002.

[Meieran, 91a] Harvey B. Meieran and Giuseppe Mosci, "The Ansaldo S.M.T.-1 Mobile Robot", in *Proceedings of the Fourth ANS Topical Meeting on Robotics and Remote Systems,* ed Mohammad Jamishidi and Patrick J. Eicker, U.S Government Printng Office, 1991, pp 375-382.

[Meieran, 91b] Harvey B. Meieran, "Mobile Robots Responding to Toxic Chemical Accidents", in *Proceedings of the Fourth ANS Topical Meeting on Robotics and Remote Systems,* ed Mohammad Jamishidi and Patrick J. Eicker, U.S Government Printng Office, 1991, pp 383-391.

[Otmane, 2000] Samir Otmane, "Augmented Reality Interface for Telerobotic Applications via Internet: NASA Space Telerobotics Program", http://lsc.cemif.univ-evry.fr:8080/Projets/ARITI/index.html [Online], 2000.

[Paul, 89] Richard P. Paul, *Robot Manipulators: Mathematics, Programming and Control.* Cambridge, MA: The MIT Press, 1989.

[Pettus, 91] R.O Pettus, J. S. Byrd, M. C. Thaker and S. K. Aswathnarayan, "Software Environment for Mobile Robots", in *Proceedings of the Fourth ANS Topical Meeting on Robotics and Remote Systems,* ed Mohammad Jamishidi and Patrick J. Eicker, U.S Government Printng Office, 1991, pp 81-89.

[Putman, 87] Byron W. Putman, *RS-232 Simplified: Everything you need to know about connecting, interfacing and troubleshooting peripheral devices.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

[Remotebot, 01] Remotebot, http://www.remotebot.net/ [Online], December 2001.

[Remotec, 01] Remotec, "Unmanned Vehicle Systems", http://www.remotec-andros.com/ [Online], October 2001.

[Sciavicco, 00] *L. Sciavicco and B. Siciliano, Modelling and Control of Robot Manipulators*. London: Springer-Verlag, 2000

[Seyer, 84] Martin D. Seyer, *RS-232 Made Easy: Connecting computers, printers, terminals and modems.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

[Seyer, 88] Martin D. Seyer, *Complete Guide to RS-232 and Parallel Connections: A Step by Step Approach to Connecting Computers, Printers, Terminals and Modems.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1988.

[Shaham, 02] Ran Shaham, Rachel Shemesh, Alon Gendel, Gal Shaham, "RS-232 for the course: Protocols and Computer Networks by Dr. D. Koren, Tel Aviv University", http://www.rad.com/networks/1995/rs232/rs232.htm [Online], December 2001.

[Storr, 87] A. Storr, "Programming Methods for Industrial Robots", in *Proceedings of the IFIP WG 5.3/IFAC Working Conference on Off-line Programming of Industrial robots*, ed. Alfred Storr and John F. McWaters, North-Holland, 1987, pp. 1-4.

[Taylor, 95] Ken Taylor and James Trevelyan, "Australias Telerobot on the web", in *26th International Symposium on Industrial robots*, http://telerobot.mech.uwa.edu.au/robot/ sing1.htm, [Online], 1995

[Yoshikawa, 90] T. Yoshikawa, *Foundations of Robotics.* Cambridge,MA: The MIT Press, 1990

Appendices

Appendix 1

Protocol Document

1.A Serial Data Packet Description for the Andros MK-VI

The following is a description of the protocol document as obtained from Remotec Inc. **[Remotec, 01]**. Table 1-A outlines the entire protocol, followed by a description of some important functions.

Byte 1:

LSB	Front Track Up
	Front Track Down
	Rear Track Up
	Rear Track Down
MSB	Hex 3 or 4 (see note 1.)

Byte 2:

LSB	Tool Enable 2
	Safety/Vehicle
	Drive Brakes
	Vehicle High Speed
MSB	Hex 3 or 4 (see note 1.)

Byte 3:

LSB	Focus Near
	Focus Far
	Iris Open
	Iris Close
MSB	Hex 3 or 4 (see note 1.)

Byte 4:

LSB	Pan Left
	Pan Right
	Tilt Up
	Tilt Down
MSB	Hex 3 or 4 (see note 1.)

Byte 5:

LSB	Lights Low
	Lights Medium
	Cable Reel Neutral
	Cable Reel Take-Up
MSB	Hex 3 Or 4 (See Note 1.)

Byte 6:

LSB	Zoom In
	Zoom Out
	Camera Select #2
	Lights Off
MSB	Hex 3 or 4 (see note 1.)

Byte 7:

LSB	1
	0
	1
	Laser on
MSB	Hex 3 or 4 (see note 1.)

Byte 8:

LSB	Weapon 1	Robot Not currently configured for a Weapon
	Weapon 2	
	Weapon vehicle	
	Weapon shotgun	
MSB	Hex 3 or 4 (see note 1.)	

Byte 9:

LSB	Wrist pitch up
	Wrist pitch down
	Wrist roll ccw
	Wrist roll cw
MSB	Hex 3 or 4 (see note 1.)

Byte 10:

LSB	Shoulder up
	Shoulder down
	Torso left
	Torso right
MSB	Hex 3 or 4 (see note 1.)

Byte 11:

LSB	0
	1
	1
	0
MSB	Hex 3 or 4 (see note 1.)

Byte 12:

LSB Grip open

Grip close

Elbow up

Elbow down

MSB Hex 3 or 4 (see note 1.)

Byte 13:

LSB This nibble is the first and most significant nibble of the vehicle pivot command.

MSB Hex 3 or 4 (see note 1.)

Byte 14:

LSB This nibble is the second and least significant nibble of the vehicle pivot command.

MSB Hex 3 or 4 (see note 1.)

Byte 15:

LSB This nibble is the first and most significant nibble of the vehicle drive command.

MSB Hex 3 or 4 (see note 1.)

Byte 16:

LSB This nibble is the second and least significant nibble of the vehicle drive command.

MSB Hex 3 or 4 (see note 1.)

Byte 17:

LSB This nibble is the first and most significant nibble of the command which controls the speed of all other motor factions, arm,front track,rear track,etc.

MSB Hex 3 or 4 (see note 1.)

Byte 18:

LSB This nibble is the second and least significant nibble of the arm speed command.

MSB Hex 3 or 4 (see note 1.)

Byte 19:

LSB This nibble is the first and most significant nibble of the Variable light intensity circuit.

MSB Hex 3 or 4 (see note 1.)

Byte 20:

LSB This nibble is the second and least significant nibble of variable light intensity circuit.

MSB Hex 3 or 4 (see note 1.)

Byte 21:

LSB Graphics on/off "down"

Graphics on/off "up"

Spare

Spare

MSB Hex 3 or 4 (see note 1.)

Byte 22:

LSB Pan/tilt extend up

Pan/tilt extend down

Grip extend out

Grip extend in

MSB Hex 3 or 4 (see note 1.)

Note 1.

This nibble is either hex 3 or 4 depending upon the value of the least significant nibble. If the value of the least significant nibble is greater than 9 this nibble is 4, otherwise, this nibble is 3.

1.B Make-Up of Data Packet

- If any function is activated in both directions at the same time, the function will not move in either direction.

- The amplifier enables should only be active when movement for the particular motion is commanded.

- All functions will be active upon sending of high (1) bits and disabled upon low (0) bits sent through the data packets.

- The vehicle is moved by changing the values of bytes 13 through 16. The combined value of nibbles found in byte 13 and 14 make up the pivot command. The combined value of the nibbles found in byte 15 and 16 make up the drive command. (see Table 1-A). Before any movement of the vehicle, the drive brakes bit must be set, and the vehicle/safety bit must set, as shown in table 1-A. These commands will result in full speed motion of the selected direction. The value 7F is the center, or no movement command of either the drive or the pivot.

Table 1-A: Vehicle Movement Functions

Vehicle Movement	LSB Nibble: 13, 14	LSB Nibble: 15, 16
Forward	0 0	X X
Reverse	F F	X X
Right	X X	F F
Left	X X	0 0

This allows the vehicle to drive and turn at any selected speed and any turning radius. If a 7F is sent to the drive and a command is sent to the pivot, the vehicle will turn about its center axis. If a 7F is sent to the pivot and a value less than 7F is sent to the drive, the vehicle will move forward.

- The "start of packet" bytes (SOP), "F1", are indicators of the beginning of the data packet. The vehicle computer will start to store all data after receiving these two bytes and will actually use the entire data packet if the checksum byte is correct.

- The checksum is simply the addition of all the data bytes (do not add the SOP bytes) with only the least sigificant byte being sent. Any carry over is ignored.

- Figure 1-A (a) Shows the sending procedure of the data packet bytes.

- Figure 1-A (b) Shows the timing of an entire 25 byte data packet at 1200 baud, and the timing interval between data packets.

- All data is transmitted serially with (8) data bits, no pariaty, and (1) stop bit over RS-232.

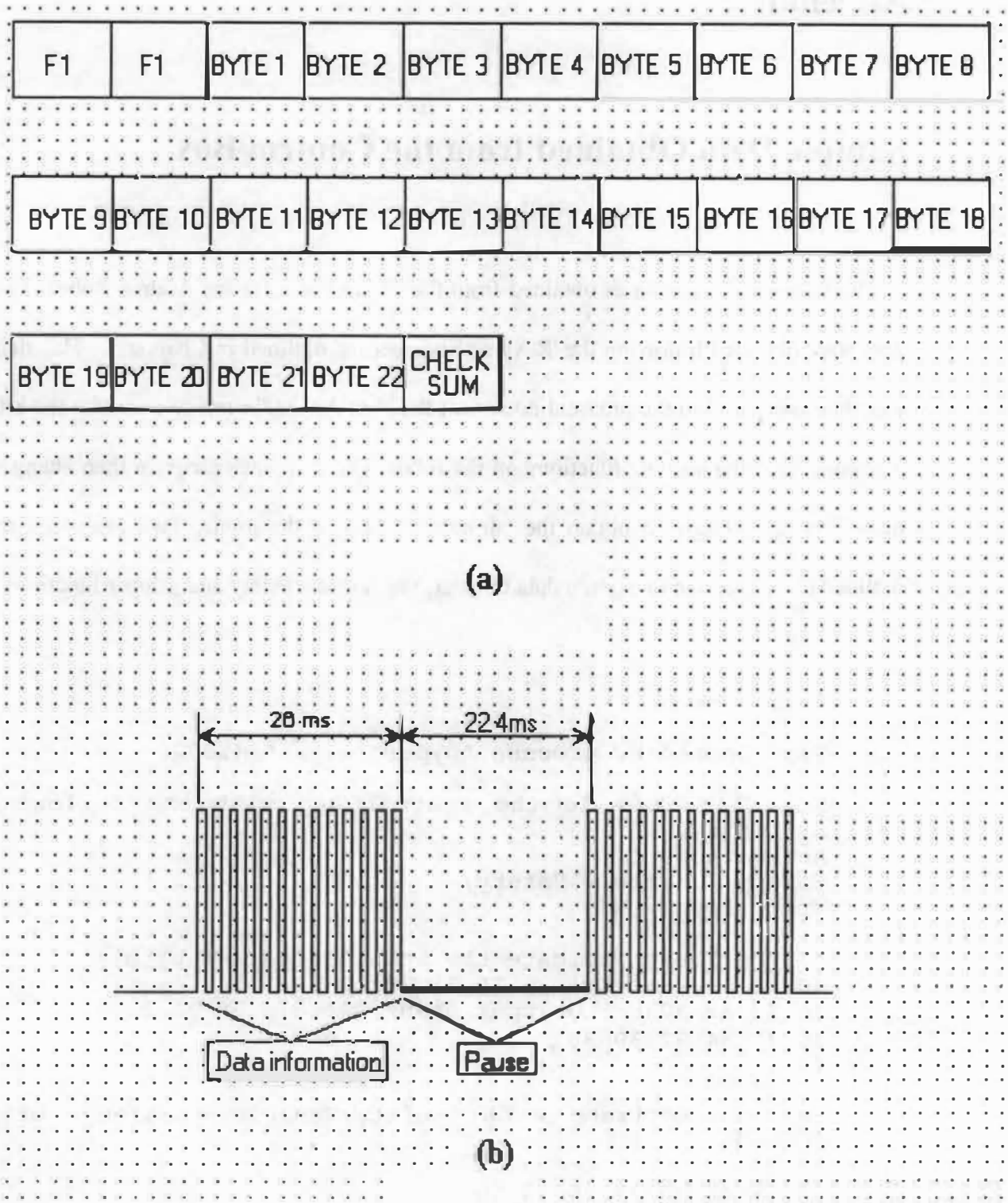


Figure 1-A: Data Transmission for the Andros Mark VI
Source: Remotec Inc.

Appendix 2

Sample Data Obtained from the Control Box

The following lists data as obtained from the Control box for the Andros Robot. This data were obtained following the 'Reverse Engineering' outlined in Chapter 5. This data was then compared to the protocol document listed in Appendix one to ascertain the bits that controlled the various functions on the robot. The computer program then changed these bits accordingly to obtain the desired motion of the joints. Two examples are outlined in this section to explain data transfer, viz Vehicle safety and gripper functions

```
Port closed

Port opened by process "hypertrm.exe" (1372)

Set (1): This is the neutral set with the following
settings
Activate = Off;
Safety/Vehicle = Safety;
Grip  Open = No.

F1 F1 (These indicate the start of packet bits)
30 34 (<--Safety) 30 30 31 38 32 30
30 30 39 30(<-- Gripper open/ close)  38 31 37 42
32 35 34 32 30 30
67

Set (2) Activate = ON; Safety/Vehicle = Safety; Grip
Open=No
F1 F1
30 34 30 30 31 38 32 30
30 30 39 30 38 31 37 41
32 35 34 33 30 30
67
```

```
Set (3) Activate = ON Safety/Vehicle = Vehicle; Grip
Open=No

F1 F1
30 36 30 30 31 38 32 30
30 30 39 30 38 31 37 41
32 35 34 33 30 30 69


Set (4) Activate = ON Safety/Vehicle = Vehicle; Grip
Open=YES

F1 F1
30 36 30 30 31 38 32 30
30 30 39 31 (<-- Gripper open changed to 31) 38 31 37
41
32 35 34 33 30 30
6A
```

Referring to Set 1 above, this is the 'Neutral' Set that the controller needs to send to the robot continuously. Upon receiving this set the robot achieves a 'Green Light' condition. No function is performed at this stage. To give an example of how the robot interprets data, refer to Set 3 it is observed that the data here was obtained by changing the 'Vehicle / Safety' setting to 'Safety', this changed the corresponding safety bit from Hex 34 to Hex 36 (indicated in bold), indicating to the robot that the safety switch was on. The robot then disables all functions as required. Similarly byte 14 controls gripper functions (open and close). In the Neutral set its value is Hex30. To open the gripper its value is changed to Hex31, this is indicated in bold in Set 4 These changes are performed in the computer program and bytes are sent to control the various functions on the robot.

Appendix 3

Listing of Computer Code

3.A Androscontrol.frm

```
'Program to Control the Andros Mark VI Hazardous Duty Mobile
'Robot.
'BY: Arun Srikantaiah, Copyright©2002.
'Latest Modification: 03/05/2002
'Load a Form
  Private Sub Form_Load()
' Use COM2: Select the COM port to use for communication
  MSComm1.CommPort = 2
'Define COM port Settings : 1200 baud, no parity, 8 data
'bits, 1 stop bit
  MSComm1.Settings = "1200,N,8,1"
' Open the port
  MSComm1.PortOpen = True
  End Sub
'Initiates timer control. This runs the events under the
'_Timer' event
  Private Sub StartTransButton_Click()
  Neutralize
'Time is defined in milliseconds.
  Timer1.Interval = 1
  End Sub
'Runs the code when the 'Start Transfer Button is clicked.
'Sends Neutral Data packets.
```

```
 Private Sub Timer1_Timer()

'Type Definition statements
 Dim k As Long
 Dim jj As Integer
 Dim p As Long
 For jj = 3 To 24
 k = k + bytData(jj)
 Next

'Set byte no 25 to the checksum

 bytData(25) = LOBYTE(k)

'Write data to the buffer

 buffer = bytData()

'Define the MSComm function to write data to the buffer

 MSComm1.Output = buffer

'Stops Data Transfer by reducing the timer interval to Zero.

 Private Sub StopTransButton_Click()
 Timer1.Interval = 0
 End Sub

'Code for various functions performed by the buttons.

'Gripper Open Function using the Mouse
'Mouse-Up event
 Private Sub GripOpenButton_MouseDown(Button As Integer,
 Shift As Integer, X As Single, Y As Single)

'Changes the value of Byte 14, according to the protocol, to
'Hex 31

 bytData(14) = &H31
 End Sub

'Mouse-Down event

 Private Sub GripOpenButton_MouseUp(Button As Integer,
 Shift As Integer, X As Single, Y As Single)
 Neutralize
 End Sub


'Gripper open using keyboard control
'Using the key-down event
```

```
  Private Sub GripOpenButton_KeyDown(KeyCode As Integer,
  Shift As Integer)


'Define the key used to control the function

  If KeyCode = vbKeyPageUp Then
  bytData(14) = &H31
  Else
  Neutralize
  End If
  End Sub

'key-up event

  Private Sub GripOpenButton_KeyUp(KeyCode As Integer, Shift
  As Integer)
  Neutralize
  End Sub

'Elbow Up

  Private Sub elbowupbutton_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(14) = &H34
  End Sub

  Private Sub elbowupbutton_MouseUp(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Elbow Down

  Private Sub elbowdownbutton_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(14) = &H38
  End Sub

  Private Sub elbowdownbutton_MouseUp(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Shoulder Up

  Private Sub shoulderup_MouseDown(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  bytData(12) = &H31
  End Sub
```

```
Private Sub shoulderup_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Shoulder Down

Private Sub shoulderdown_MouseDown(Button As Integer,
Shift As Integer, X As Single, Y As Single)
bytData(12) = &H32
End Sub

Private Sub shoulderdown_MouseUp(Button As Integer, Shift
As Integer, X As Single, Y As Single)
Neutralize
End Sub

'Torso Left

Private Sub torsoleft_MouseDown(Button As Integer, Shift
As Integer, X As Single, Y As Single)
bytData(12) = &H34
End Sub

Private Sub torsoleft_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Torso Right

Private Sub torsoright_MouseDown(Button As Integer, Shift
As Integer, X As Single, Y As Single)
bytData(12) = &H38
End Sub

Private Sub torsoright_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Front Track Up

Private Sub frontrackupbutton_MouseDown(Button As Integer,
Shift As Integer, X As Single, Y As Single)
bytData(3) = &H32
End Sub

Private Sub frontrackupbutton_MouseUp(Button As Integer,
Shift As Integer, X As Single, Y As Single)
Neutralize
End Sub
```

```
'Front Track Down

  Private Sub Frontrackdown_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(3) = &H31
  End Sub

  Private Sub Frontrackdown_MouseUp(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Rear Track Up

  Private Sub reartrackup_MouseDown(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  bytData(3) = &H38
  End Sub

  Private Sub reartrackup_MouseUp(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Rear Track Down

  Private Sub reartrackdown_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(3) = &H34
  End Sub

  Private Sub reartrackdown_MouseUp(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Wrist Pitch Up

  Private Sub wristpitchupbuttuon_MouseDown(Button As
  Integer, Shift As Integer, X As Single, Y As Single)
  bytData(11) = &H31
  End Sub

  Private Sub wristpitchupbuttuon_MouseUp(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  Neutralize
  End Sub
```

```
'Wrist Pitch Down

  Private Sub wristpitchdown_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(11) = &H32
  End Sub

  Private Sub wristpitchdown_MouseUp(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Wrist Roll CCW

  Private Sub wristrollccw_MouseDown(Index As Integer,
  Button As Integer, Shift As Integer, X As Single, Y As
  Single)
  bytData(11) = &H34
  End Sub

  Private Sub wristrollccw_MouseUp(Index As Integer, Button
  As Integer, Shift As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Wrist Roll CW

  Private Sub wristrollcw_MouseDown(Index As Integer, Button
  As Integer, Shift As Integer, X As Single, Y As Single)
  bytData(11) = &H38
  End Sub

  Private Sub wristrollcw_MouseUp(Index As Integer, Button
  As Integer, Shift As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Grip Extend Out

  Private Sub gripextendout_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(24) = &H34
  End Sub

  Private Sub gripextendout_MouseUp(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  Neutralize
  End Sub
```

```
'Grip Extend In

  Private Sub gripextendin_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(24) = &H38
  End Sub

  Private Sub gripextendin_MouseUp(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

'Camera Controls

  'Camera Tilt Up

  Private Sub cameratiltup_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(6) = &H34
  End Sub

  Private Sub cameratiltup_MouseUp(Button As Integer, Shift
  As Integer, X As Single, Y As Single)
  Neutralize
  End Sub

  'Camera Tilt Down

  Private Sub cameratitldown_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(6) = &H38
  End Sub

  Private Sub cameratitldown_MouseUp(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  Neutralize
  End Sub


  'Camera Pan Right

  Private Sub camerapanright_MouseDown(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  bytData(6) = &H32
  End Sub

  Private Sub camerapanright_MouseUp(Button As Integer,
  Shift As Integer, X As Single, Y As Single)
  Neutralize
  End Sub
```

```
'Camera Pan Left

Private Sub camerapanleft_MouseDown(Button As Integer,
Shift As Integer, X As Single, Y As Single)
bytData(6) = &H31
End Sub

Private Sub camerapanleft_MouseUp(Button As Integer, Shift
As Integer, X As Single, Y As Single)
Neutralize
End Sub

'Pan / Tilt Extend Up

Private Sub pantiltextendup_MouseDown(Index As Integer,
Button As Integer, Shift As Integer, X As Single, Y As
Single)
bytData(24) = &H32
End Sub


Private Sub pantiltextendup_MouseUp(Index As Integer,
Button As Integer, Shift As Integer, X As Single, Y As
Single)
Neutralize
End Sub

'Pan / Tilt Extend Down

Private Sub pantiltextenddown_MouseDown(Index As Integer,
Button As Integer, Shift As Integer, X As Single, Y As
Single)
bytData(24) = &H31
End Sub

Private Sub pantiltextenddown_MouseUp(Index As Integer,
Button As Integer, Shift As Integer, X As Single, Y As
Single)
Neutralize
End Sub


'Camera Select 1

Private Sub cameraselect1_Click()
Neutralize
End Sub
```

```
'Camera Select 2

Private Sub cameraselect2_Click()
bytData(8) = &H34
End Sub

'Camera Zoom In

Private Sub zoomin_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
bytData(8) = &H32
End Sub

Private Sub zoomin_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Camera Zoom Out

Private Sub zoomout_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
bytData(8) = &H31
End Sub

Private Sub zoomout_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Robot Movement

'Forward Using the Mouse

Private Sub forward_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
bytData(17) = &H4F
End Sub

Private Sub forward_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Forward Using the key board
'Initialize KeyDown event for key board control

Private Sub forwad_KeyDown(KeyCode As Integer, Shift As
Integer)
```

```
'Define the key used for control. Up Arrow for forward
'motion

If KeyCode = vbKeyNumpad8 Then

'Change Byte Value according to the protocol (Appendix 1)

bytData(17) = &H4F
End If
End Sub

'Key Up event for forward motion

Private Sub forward_KeyUp(KeyCode As Integer, Shift As
Integer)
Neutralize
End Sub

'Reverse Using the Mouse

Private Sub reverse_MouseDown(Button As Integer, Shift As
Integer, X As Single, Y As Single)
bytData(17) = &H30
End Sub

Private Sub reverse_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Reverse Using the key board


'Initialize KeyDown event for key board control

Private Sub reverse_KeyDown(KeyCode As Integer, Shift As
Integer)

'Define the key used for control: Down Arrow for reverse
'motion, i.e. the number key '2' on the keypad.

If KeyCode = vbKeyNumpad2 Then

'Change Byte Value according to the protocol (Appendix 1)

bytData(17) = &H30
End If
End Sub
```

```
'Key Up event for reverse motion

Private Sub reverse_KeyUp(KeyCode As Integer, Shift As
Integer)
Neutralize
End Sub

'Pivot Left Using the Mouse

Private Sub pivotleft_MouseDown(Button As Integer, Shift
As Integer, X As Single, Y As Single)
bytData(15) = &H4F
End Sub

Private Sub pivotleft_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub


'Pivot Left Using the Key board

'Initialize KeyDown event for key board control

Private Sub pivotleft_KeyDown(KeyCode As Integer, Shift As
Integer)

'Define the key used for control: Left Arrow for left
'pivot motion, i.e. the number key '4' on the keypad.

If KeyCode = vbKeyNumpad4 Then

'Change Byte Value according to the protocol (Appendix 1)

bytData(15) = &H4F
End If
End Sub


'Key Up event for Pivot Left motion

Private Sub pivotleft_KeyUp(KeyCode As Integer, Shift As
Integer)
Neutralize
End Sub
```

```
'Pivot Right Using the Mouse

Private Sub pivotright_MouseDown(Button As Integer, Shift
As Integer, X As Single, Y As Single)
bytData(15) = &H30
End Sub

Private Sub pivotright_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Neutralize
End Sub

'Pivot Right Using the Key Board


'Initialize KeyDown event for key board control

Private Sub pivotright_KeyDown(KeyCode As Integer, Shift
As Integer)

'Define the key used for control: Right Arrow for Right
'pivot motion, i.e. the number key '6' on the keypad.

If KeyCode = vbKeyNumpad6 Then

'Change Byte Value according to the protocol (Appendix 1)

bytData(15) = &H30
End If
End Sub


'Key Up event for Pivot Right motion

Private Sub pivotright_KeyUp(KeyCode As Integer, Shift As
Integer)
Neutralize
End Sub
```

3.B Androscontmodule.bas

```
'Type Definition

Global buffer As Variant
Global bytData(25) As Byte
Global ii As Integer
```

```
  Global total As Byte


'Defining a function called Neutralize to hold the Neutral
'Data set

  Function Neutralize()

'Setting up an Array for the Robot Data
  For ii = 1 To 25
  bytData(ii) = &H30
  Next

'Changing the bytes according to the protocol

  bytData(1) = &HF1
  bytData(2) = &HF1
  bytData(4) = &H36
  bytData(7) = &H31
  bytData(8) = &H38
  bytData(9) = &H32
  bytData(13) = &H39
  bytData(15) = &H38
  bytData(16) = &H31
  bytData(17) = &H37
  bytData(18) = &H41
  bytData(19) = &H32
  bytData(20) = &H35
  bytData(21) = &H34
  bytData(22) = &H33
  End Function

'Calculate the checksum

  Function LOBYTE(ByVal w As Integer) As Byte
  LOBYTE = w And &HFF
  End Function
```

Vita

Arun Srikantaiah was born in Hubli, India on June 15, 1975. He obtained his Bachelors of Engineering Degree in Mechanical Engineering from Malnad College of Engineering, The University of Mysore, India, in October 1996, Graduating with First Class honors. Thereafter he worked for A.P Moller Singapore Pte. Ltd., as a Marine Engineer Class 4, from June 1997 until June 2000. He came to the University of Tennessee, Knoxville in August 2000 and was awarded a Graduate Teaching Assistantship. He received a Masters Degree in Mechanical Engineering in May 2002.