

A PDA ENABLED WIRELESS INTERFACE FOR A MOBILE ROBOT

Abstract

By

Amishi Joshi

The area of mobile networking and pervasive networking is indeed an area of active research now. The once independent requirements of ‘staying-connected ‘ and ‘information-on-demand’ are now converging to a one-solution requirement.

The relatively inexpensive yet powerful handheld devices and accessories available today provoke the simple question of whether it is possible to provide efficient control over wireless links such as those available for the handhelds. Do the handhelds have enough processing power to support an acceptable service, and what would the power budget be for such an implementation? Are there ways of optimizing traffic over wireless packet networks? These are the questions responsible for proposing this project.

This project describes a wireless application that allows remote control of a robot. The application adopts an object-oriented philosophy in which every robot is a device represented by an object. The application runs in a PC with a web server. The interface is done using common Java GUI's.

The interaction with the robot control system is done through specific hardware and PDA standard interface such as the serial port and by the LAN.

1. INTRODUCTION

The primary aim of this project is to achieve efficient motor control using a personal digital assistant (PDA) that is equipped with wireless networking capabilities. In other words, a PDA should provide the basic functionality of a mobile processor with the added advantage of operating over an “always connected” packet switched IP network.

The goal is to develop a system of PDA-hosted controls required to control the maneuverability of a robot, by using standard TCP/IP protocols so that greater functionality (such as vision feedback) can be later added to the system easily.

1.1 Overview

Some of the issues that will be addressed here are:

- Low-power communication: Given the power constraints on a PDA, a communication algorithm that achieves a low bit rate while keeping power consumption to a minimum was developed.
- Network bandwidth: A wireless communication device that provides a bandwidth sufficient for carrying the signal traffic was efficiently chosen. There are several products that offer wireless connectivity at rates between 9.6 Kbps to more than 1Mbps (802.11b).
- Signaling: Signaling was done in a standard way to ensure compatibility with other systems.

- **Hardware Interfacing:** Hardware interfacing circuitry was designed to provide the necessary interfacing between the motors of the robot and the communication ports of the PDA.

All the development done was open-source, with an aim to achieve robustness and extensibility.

1.2 Basic Idea of the project

The objective was to develop a PDA client and a simple PC host that together can be used to test and demonstrate the functionality of the system. The PDA client also works as an interface to the Ethernet-to-motor control hardware required to control the robot.

1.3 Objective and organization of the project

Chapter 2 starts by describing the general architecture of the system and identifying its main components and their roles. Then the conceptual model of the system is presented and the various classes that allow configuring and controlling the system are described. It also includes the preliminary investigation of the different procedures that were experimented before reaching the final design. Section 2.3 describes the basic infrastructure of the robot- miniwhegs used for the project. Section 2.4 talks about the hardware interfacing and the design of the control board. Section 2.5 explains the communication aspect of the project. Section 2.6 concludes the project with some recommendations to future work.

2. PROPOSED SYSTEM ARCHITECTURE

2.1 System Architecture

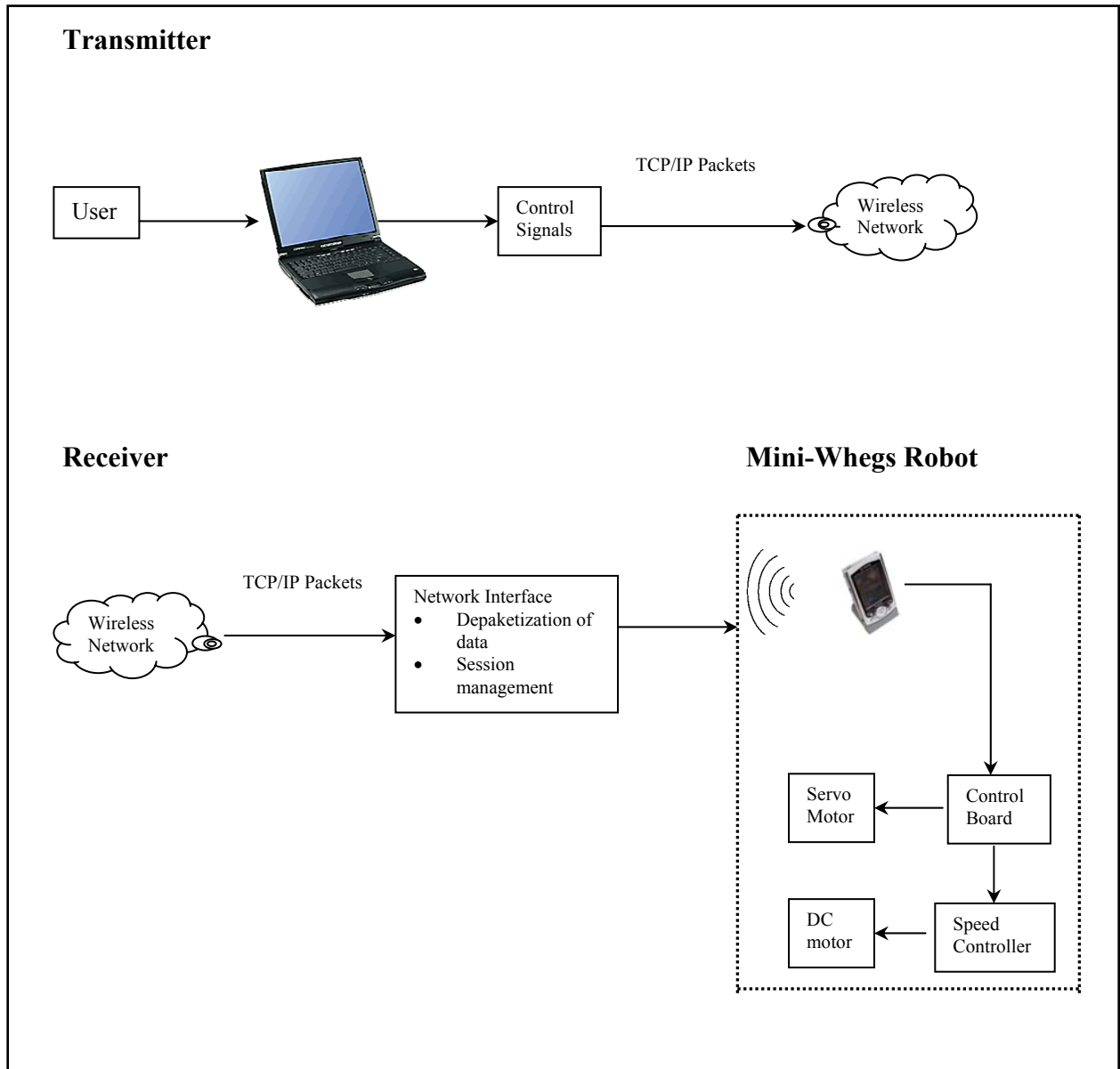


Figure 2.1. Basic System Architecture

2.1.1 User:

The user is able to interact with the system with the help of a simple Graphical User Interface where in the user is able to turn the motors on/off and specify the angle with which the robot is expected to move.

This information is sent in the form of TCP/IP packets over a wireless network. The communication takes place with the help of sockets. A socket is a software endpoint that establishes bi-directional communication between a server program and one or more client programs. The socket associates the server program with a specific hardware port on the machine where it runs so any client program anywhere in the network with a socket associated with that same port can communicate with the server program.

2.1.2 Control Board:

The control board mainly comprises of a peripheral Interface Controller (PIC) 16C73B. This PIC has two serial ports and two pulse-width modulation outputs. It also has 3 ports of digital inputs/outputs. The input to the PIC is in the form of binary signals that are transmitted over the network. The PIC converts these signals into PWM signals that are then sent as an input to the servomotor and the speed controller. A detailed schematic of the control board circuitry is given and explained later in this chapter.

2.2 System Specifications

- Cassiopeia E200: This PDA, the latest Casio Handheld available was used for the project. It is powered by a 206 MHz Intel StrongARM RISC processor with 64 MB RAM and a slot for CompactFlash (Type II) card.
- Wireless Network Card: Linksys WCF11- IEEE 802.11b(WLAN), Type II CompactFlash Standard card with 11 channels. It has a range of 30M at 11 Mbps.
- PocketPC 2002 OS: The Cassio handheld runs on PocketPC 2002. It supports Java programming and TCP/IP socket programming. Several other windows API have been ported to the PocketPC 2002 as well.
- Mini-Whegs: The Mini-Whegs robot prepared by the Bio-Robotics group at Case Western Reserve University was used as the basic platform for communications.
- Servomotor: This motor was used for steering purposes. It is a Cirrus CS-10bb motor with dimensions of 0.09 X 0.61 X 0.37 inches and weighing 0.19 oz.
- Drive Motor + Planetary Transmission: The drive motor was a Maxon RE 13 precious metal brushes motor with a 13mm diameter. It has a 1.2 Watt, 3.6 V Nominal Voltage, 3.82 mNm stall torque, 50mA no load current and 1950 mA starting current.
- Speed Controller: Combined 4 channel receiver and forward only speed controller RX-72 Hybrid was used.
- Control Board: The SV203B manufactured by Pontec was used as the control board for interfacing with the Cassiopeia E200. It comprises of the PIC 16C73B used for serial interface.

2.3 MINI-WHEGS ROBOT

2.3.1 Mini-Whegs Robot

Mini-Whegs robot is a biologically inspired robot abstractly based on the movement of a cockroach. The name, Whegs is derived from wheels plus legs. Traditional biologically inspired robots are usually leg based rather than wheel based, and this particular marriage allows benefits from both the wheeled and legged designs. Among these benefits are a relatively simple leg/drive construction and the ability to climb obstacles nearly as tall as the robot itself. The ability to jump lends further mobility to these relatively small robots and makes them excellent candidates for exploration and autonomous missions. However this feature is not controlled in this project.



Figure 2.2 Mini-Whegs Robot

2.4 HARDWARE INTERFACING

2.4.1 How the Hardware Transfers Bytes

The RS-232C standard is used for the hardware communication. The RS-232C is a standard that describes the physical interface and protocol for relatively low speed serial data communication. RS-232C is the interface used to talk and exchange data between the PDA and the robot which are serially connected to each other.

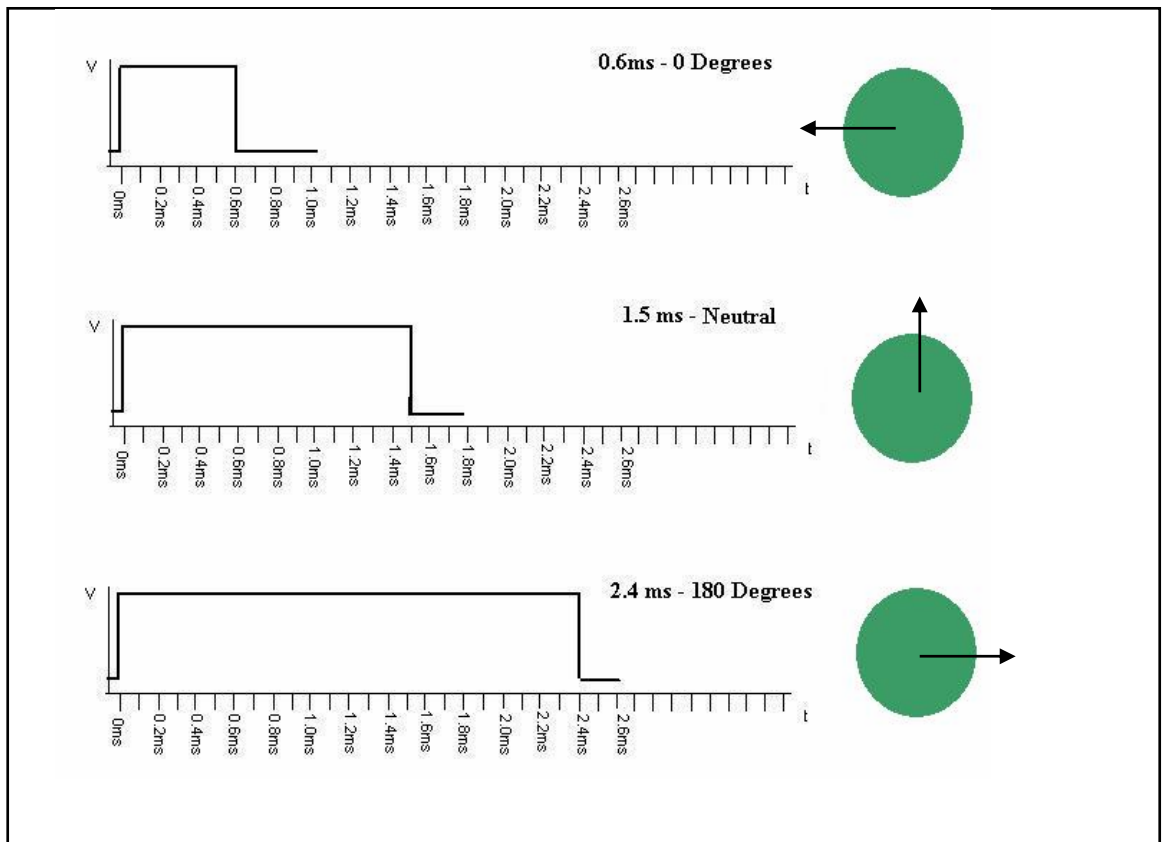


Figure 2.3 Timing diagram of the servo motor.

Communication as defined by the RS-232C standard is an asynchronous serial communication method. The word serial means, that information is sent one bit at a time. Asynchronous means that the information is not sent in predefined time slots. Data transfer can start at any given time and it is the task of the receiver to detect the start and end of a message.

2.4.1.1 Transmitting:

Transmitting is sending bytes out of the serial port away from the PDA. When the PDA wants to send a byte outside the serial port (to the external cable) the processor sends a byte on the bus inside the PDA to the I/O address of the serial port. The serial port takes the byte, and sends it out one bit at a time (a serial bit-stream) on the transmit pin of the serial cable connector.

2.4.1.2 Receiving:

Receiving of bytes by the serial port present on the control board is done in a way similar to transmitting, except that it is in the opposite direction.

2.4.2 Control Board

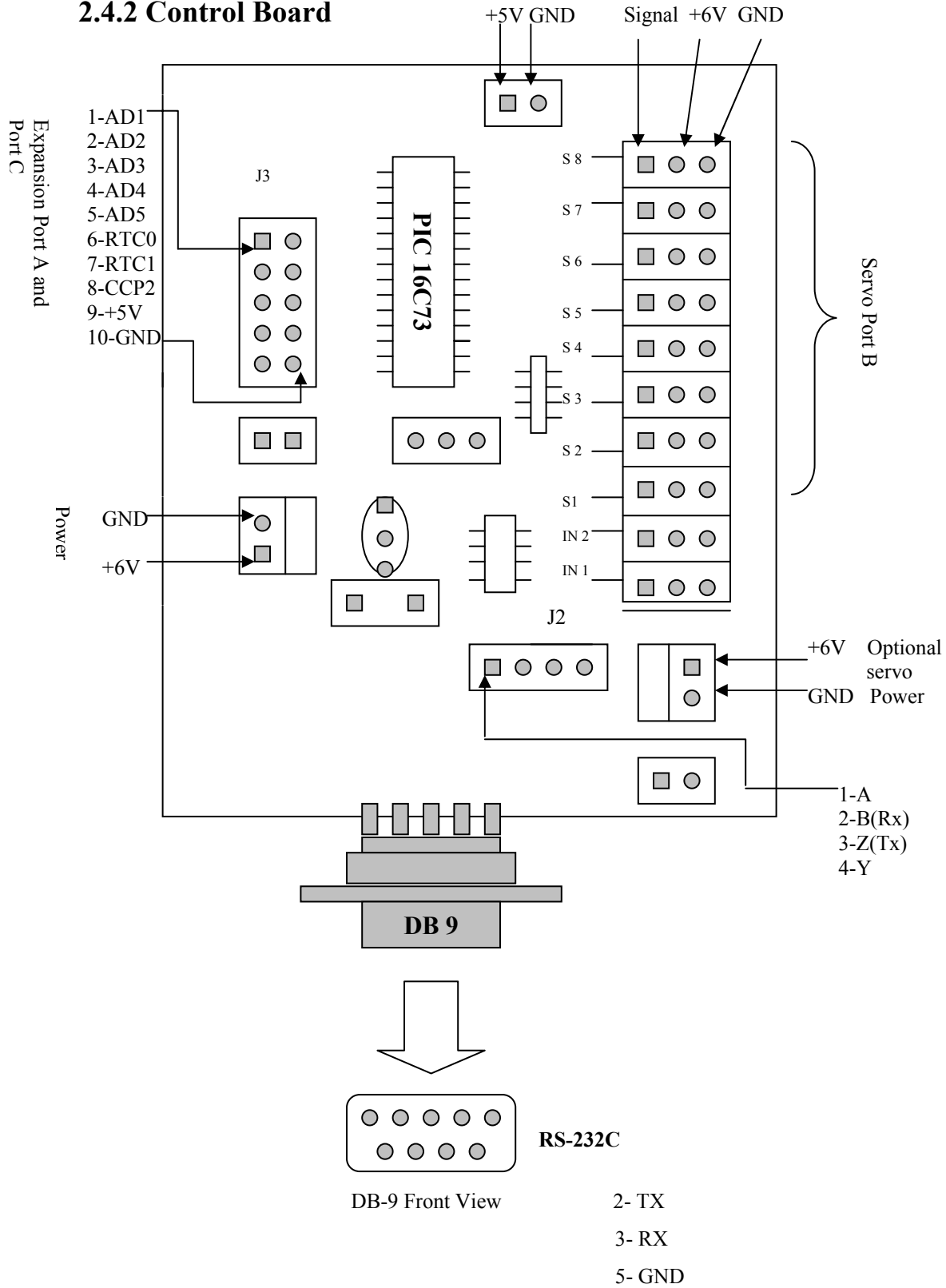


Figure 2.4. SV203 Control Board Pin-out

The SV203 is a Microchip PIC16C73 microcontroller based servo motor controller board. It accepts RS232 serial data signal from the host PDA and outputs PWM (pulse width modulated) signal which can control up to eight RC servo motors (servos used in radio-controlled model airplanes, cars, etc.). Unused servo pins can be reconfigured for digital output to drive on/off devices. In this application only two servo pins are used. One is connected directly to the RC servomotor and the other is connected to the speed controller. The speed controller is connected to the D.C motor. The servo motor is powered through the speed controller. The servo motor is separately powered. However there is a common 6 V battery supplying power to the motors as well as the board.

A 5 channel, 8-bit A/D input is available to read analog voltages between 0 to 5 Volts. Devices such as an analog joystick or potentiometers can be connected to this port and the position can be read by the PDA and sent back to the board to control the servo position.

The communications resources offered by the PIC 16C73 are a synchronous serial port (SSP) that can operate in two modes- Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C)

SPI mode is a synchronous-based protocol that can operate as a full duplex connection. SPI is primarily used to talk to serial EPROMs. The PIC is clocked at 3.5795 MHz and is supported with a MAX489 line driver. Serial data for the PDA is driven by the MAX489 UART, connected to the PIC's synchronous serial port.

The programming is taken care of in a loop after the initialization of all the PIC's internal registers and the external peripherals. This loop is responsible for reading the PIC's A/D converter to get battery and other voltages, switching the power accordingly, then checking the state of the buffers and trying to deal with packet traffic. The voltage checking portion of the code simply activates the PIC's internal A/D converter to record the most current values of battery voltage, battery current, data input voltage, data input current and bus output voltage.

The packet handling functions of the control board accomplishes four things:

- incoming packets from the bus are sent over the motors.
- information from the board is sent over the PDA in the form of packets.
- data for the bus is sent over the bus.
- commands for the motors are executed.

Whenever the main loop of the program notices that it has at least one complete packet buffered it sends it over to the motors if it is available (powered up and not busy) and sets a busy flag. That flag is cleared when the incoming packet handler gets a response from the PDA that the transmission is complete. The incoming packet handler functions by polling the external UART for data and storing it in an internal buffer if there is any. Once that buffer has reached the correct length it is checked to see if it is destined for the board as a command or for the bus. If it is for the latter, it is copied to the buffer and eventually transmitted by the interrupt handler. If the packet received is a command for the board then the second byte is examined for what to do and the command is executed.

The SV203 processes commands sent by a host computer connected to the serial port.

The commands are ASCII character strings that select the board, tell which servo to control, and the position of the servo.

A 6-Volt DC source powers the board, either from 4 alkaline batteries or 5 NiCad cells.

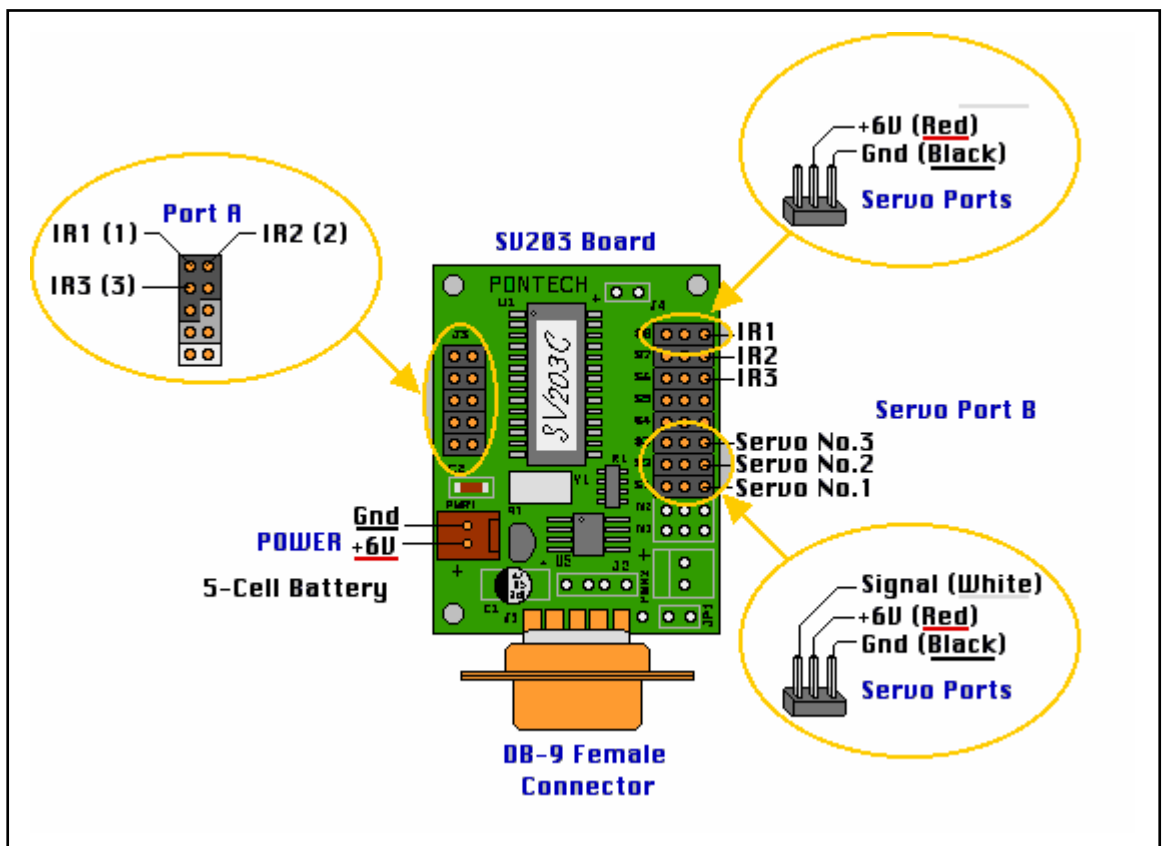


Figure 2.5. SV203 Configuration

2.4.3 Working of the Servomotor

The RC servo operates using feedback to compare the current position to an input pulse width, which typically repeats itself every 14 to 20 ms. If the pulse width lasts for approximately 0.6 ms the servo will rotate to a maximum position. If the pulse width is increased to approximately 2.4 ms, the servo will rotate to the opposite maximum position. A 1.5 ms pulse will set the servo to the neutral (middle) position.

2.4.4 Control Board Commands

The commands used to control the motors connected to the control board are as given below.

SV'n' – Selects the motor to be controlled

As explained above, the control board has eight output pins where motors can be connected. For this particular application only two of the eight have been used. The 'n' denotes the pin number on which the motor is connected and has a range from 1 to 8.

M'n' – Move to absolute position:

This command will move the servomotor to an absolute position. The range of the position is between 1 and 255. On a RC servo, the maximum mechanical movement is

about 180 degrees. The 1 to 255 position ranges gives a precision of a little under one degree. However, the servo motor on board allows a range from 95 to 160 with 128 as the center position. The figure below shows what position the servo will be in the given value of position.

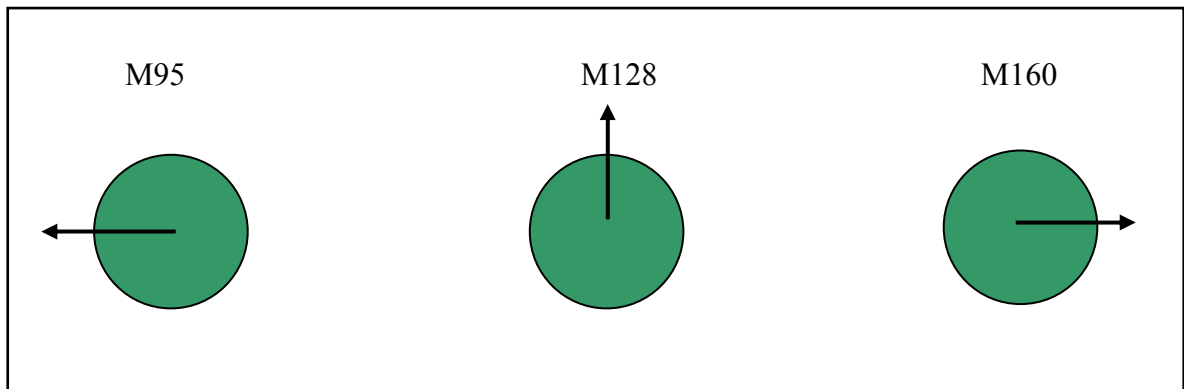


Figure 2.6. Examples of Servo Commands

Sending a zero to the servo can turn off the pulse-width command signal to the servo, which causes the servo to remove power from the motor.

In order to turn the DC motor on, the command M'n' was sent in small increments till the motor was turned on. This happened at M'145'. Next, the same command was sent in increments of 1 to increase the speed of the motor. The robot could successfully move till M'175'. Sending M'0' turns the DC motor off.

2.4.5 Command Description

There is a configuration register at location 60 in the RAM. The value of the register is initialized by the contents of the EEPROM in location 11. The register configures the shift function for the MSF (Most Significant Bit) or LSF (Least Significant Bit), data valid on clock going high or low, and the number of bits to shift in/out.

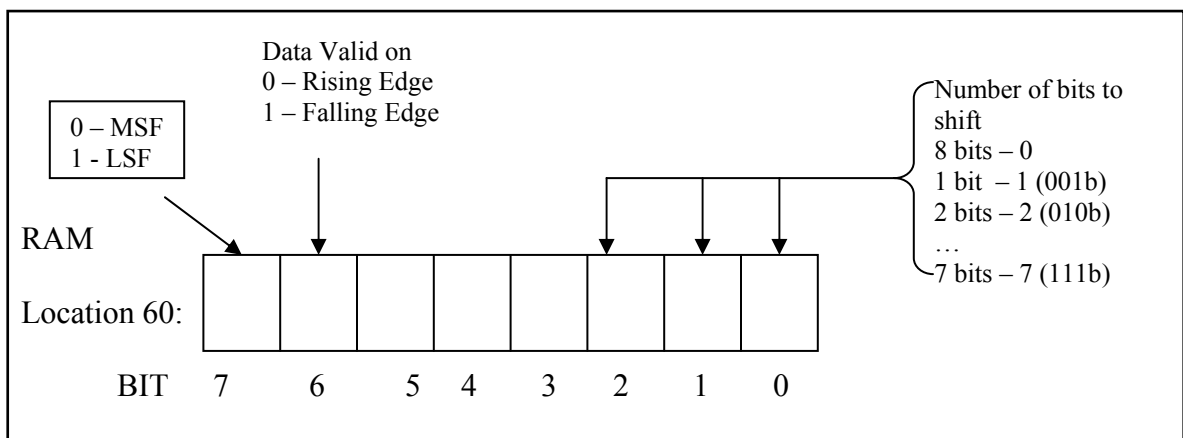


Figure 2.7. Configuration Register

2.4.6 Hardware Interfacing

The hardware is connected in the way shown below. The PDA is connected to the control board through the serial cable. The motors are directly connected to the control board as shown.

The servomotor is connected to the control board in the way shown below.

In order to connect the DC motor to the control board similar connections are made except that instead of a servomotor a speed controller is connected to the board. The DC motor is connected on the other end of the speed controller.

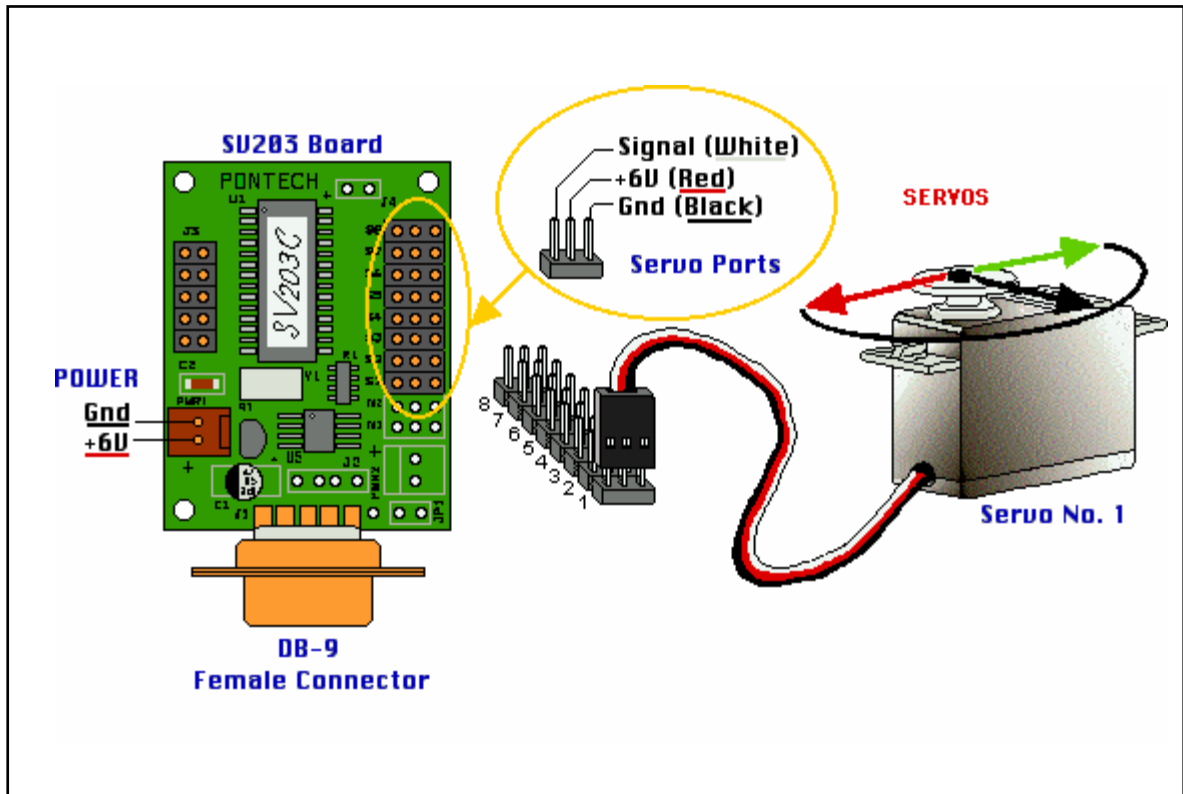


Figure 2.8. Connections between the control board and the servo motor.

To connect the PDA device to the control board a serial sync cable was used. This cable has a DB-9 male connector on one side which hooks directly into the SV203-B board and a 20 pin connector on the other side which hooks on to the PDA device as shown below.

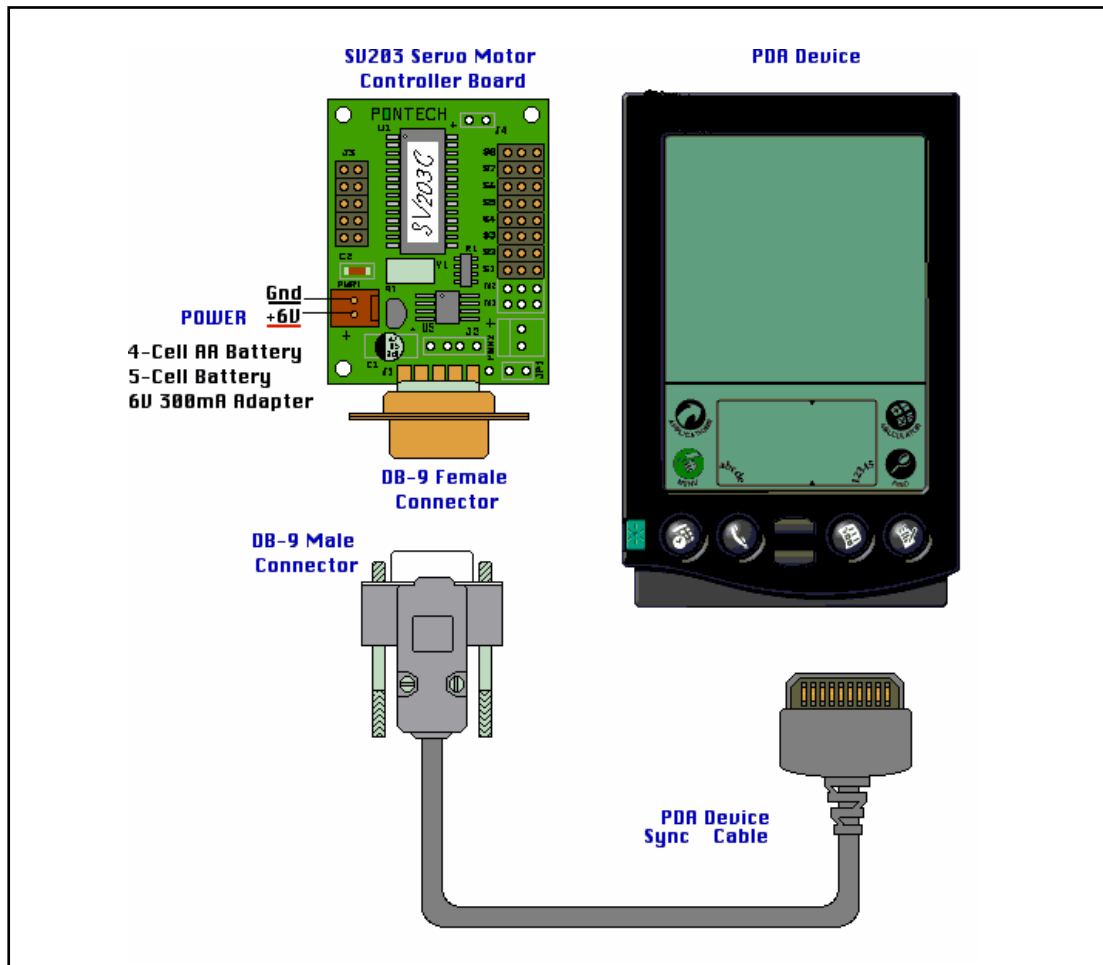


Figure 2.9. Connection between the PDA device and the Control Board.

2.5 COMMUNICATIONS

The topic of communications can be divided into two major sections:

1. Wireless communications which involves talking with the handheld device over the wireless network.
2. Serial communications which involves talking to the serial port of the handheld device.

The wireless handheld device is the client which talks to the server (which in this case is the desktop/workstation) by means of a wireless card. In general, a wireless communication can be classified into two – the Independent Basic Service Set (IBSS) and the Basic Service Set (BSS). The IBSS consists of those wireless networks wherein an access point is provided which would help identify the remote clients. BSS is of the ad hoc nature, wherein no access points are provided.

The wireless communications procedure used between the handheld device running PocketPC2002 as its operating system and a desktop running any Windows operating system is explained below.

A typical wireless setup of the IBSS system is described in the following figure

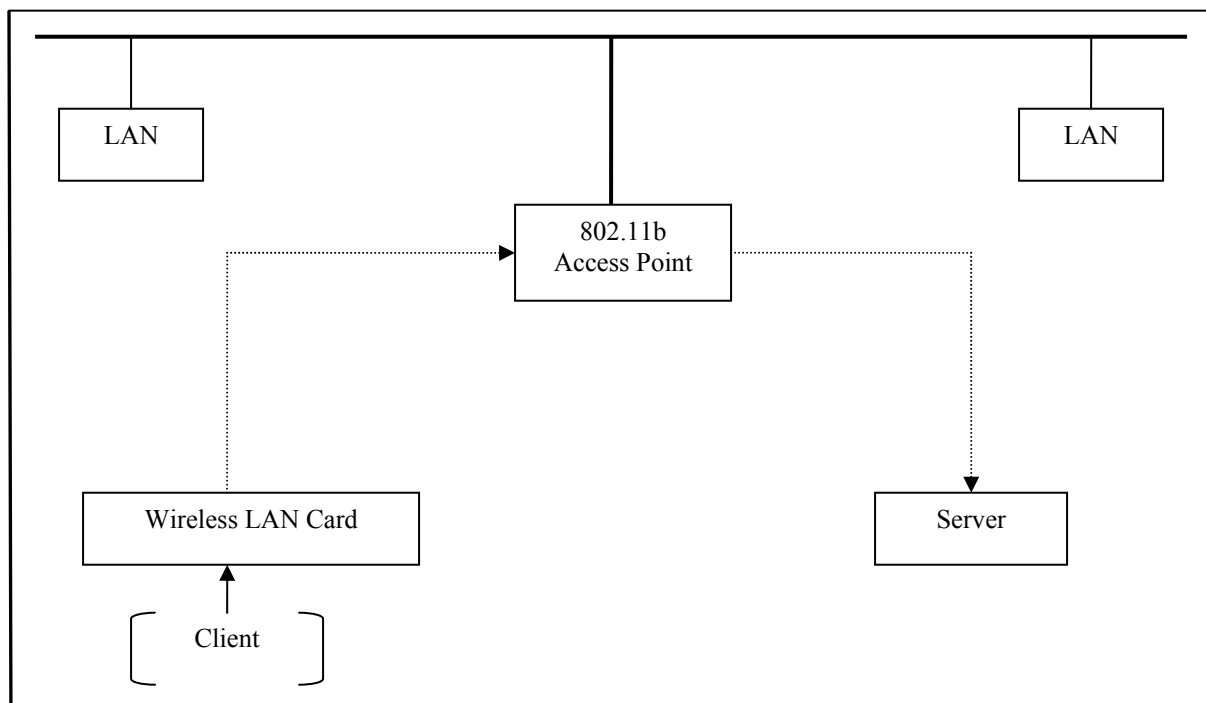


Figure 2.10. IBSS System

For the handheld to become an active member in the wireless network it has the wireless LAN card installed along with its driver. Once the driver is correctly installed, the handheld is ready to operate over the internet. The handheld is assigned an IP address using the wireless LAN card. Once the IP address is assigned the device becomes visible in the network. Communication can take place between the handheld and the desktop in the same network.

The 802.11b serves as the Network Access Point, which is connected to the Ethernet. It consists of the LAN access service driver. When the wireless device needs to have the services of the local LAN, it connects using the Access Points.

In the case of this approach, the handheld connecting as a client, first communicates with the access point to query the IP address of the server. The socket communication is now established and the client and server can exchange data.

2.5.1 Networking Basics

Computers running on the Internet communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), as this diagram illustrates:

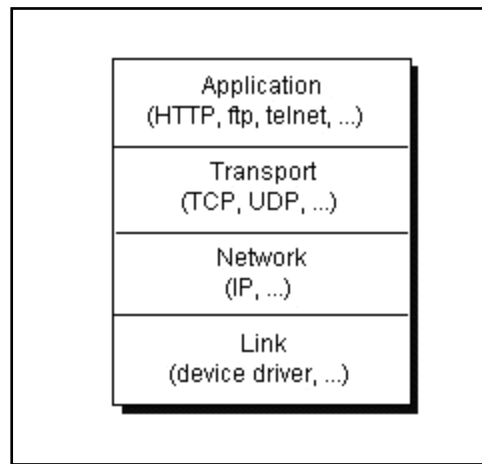


Figure 2.11. Different Network layers.

2.5.1.1 TCP

TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers.

2.5.1.2 UDP

UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.

In this case the socket communication established is of the TCP type.

2.5.2 Programming Sockets

A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

The server (which in this case is the hand held device) has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

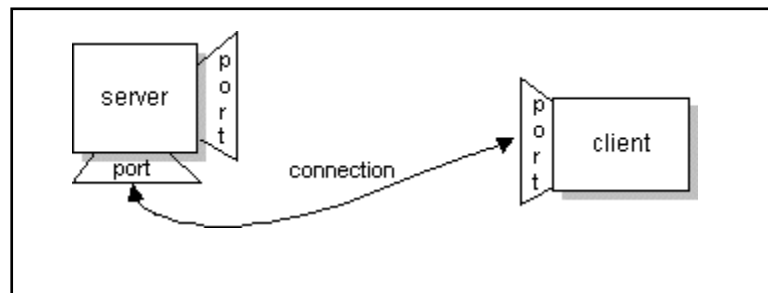


Figure 2.12.b. Connection accepted by server.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number to which the server is connected. To make a connection request, the client (which is the desktop computer in this case) tries to rendezvous with the server on the server's machine and port.

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (and consequently a different port number) so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. Note that the socket on the client side is not bound to the port number used to rendezvous with the server. Rather, the client is assigned a port number local to the machine on which the client is running.

The client and server can now communicate by writing to or reading from their sockets.

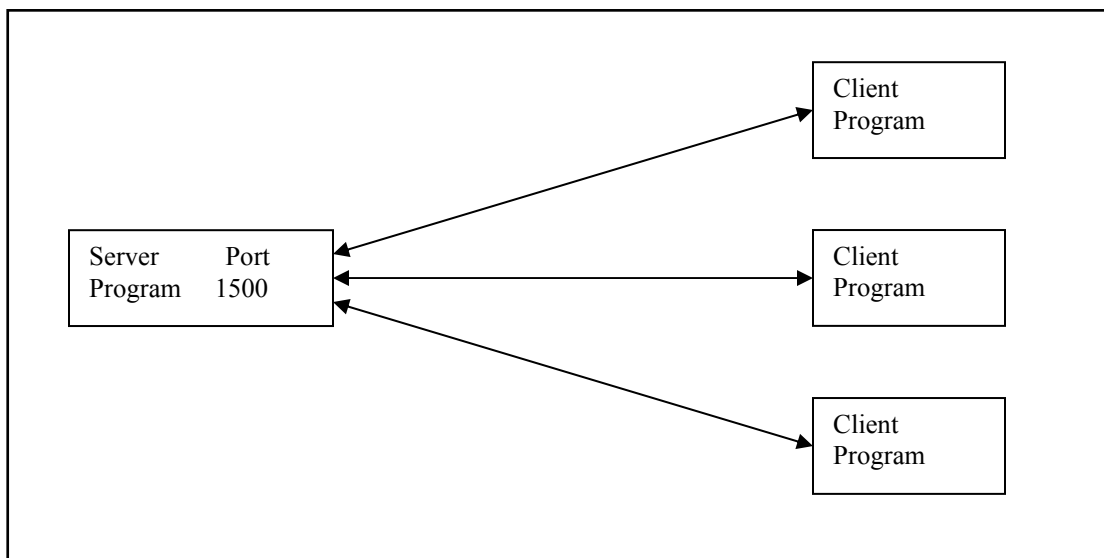


Figure 2.13. Client–Server architecture

2.5.2.1 Porting

Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined. The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network. Ports are identified by a 16-bit number, which use to deliver the data to the right application.

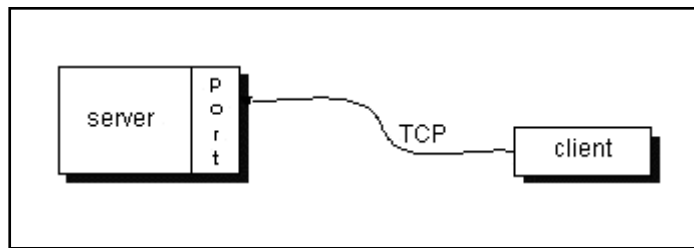


Figure 2.14. A TCP connection between client and server

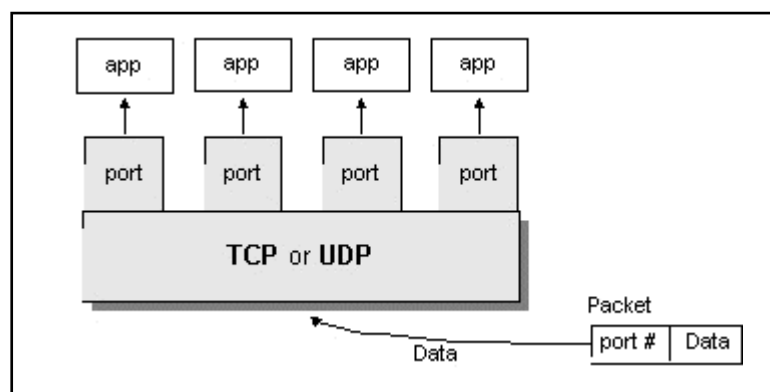


Figure 2.15. Connection between the network and application layer.

In connection-based communication i.e. the TCP, the server application binds a socket to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then rendezvous with the server at the server's port.

2.5.2.2 The Product

The executable is entirely composed of Java classes written to conform to Insigia's Jeode which confines to Sun's PersonalJava standard, an API that includes a majority of the features of the Java 1.2 Standard Development Kit(SDK). The runtime libraries are available for a fee for most desktop platforms, as well as for PocketPC 2002 handhelds. The PersonalJava is a runtime environment and can execute only the class files. The class files must be made only by a Java 1.1.x SDK version.

2.5.2.3 Testing the connection

The first step was to test whether a basic connection link between the desktop computer and the handheld device could be established. This would also test the working of the wireless card for the handheld. To do this a simple socket code was written. Since the desktop computer was behind a firewall and the wireless node was on the other side it was decided to make the desktop computer the server and the handheld device would be the client. Due to the presence of the firewall it was not possible for the handheld to initiate the connection.

In the test program, the computer opens a socket and listens on a particular port for the client. Once the client program is executed on the handheld, it also opens a socket and establishes a connection with the desktop computer since it has its IP address. Now any data sent by the server is accepted by the client. This is verified since all the data sent by the user on the computer is made visible in a console window on the handheld device. The client program was written such that when a user types a special character “.”, it would close the connection.

Screenshots of the client and server are given below.

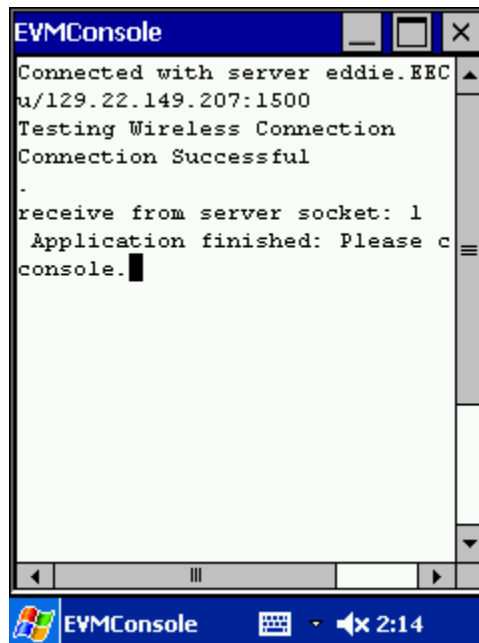
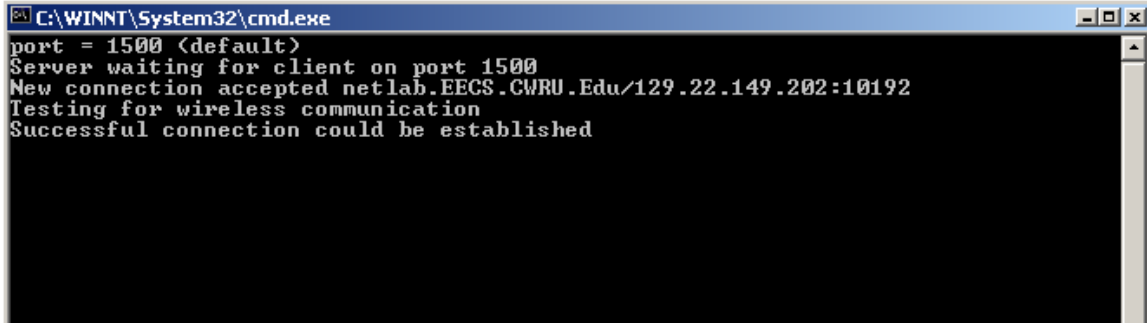


Fig. 2.16. Screen shot of PDA



```

C:\WINNT\System32\cmd.exe
port = 1500 (default)
Server waiting for client on port 1500
New connection accepted netlab.EECS.CWRU.Edu/129.22.149.202:10192
Testing for wireless communication
Successful connection could be established

```

Fig. 2.17. Screen shot of the desktop (client)

2.5.3 Implementing the Server

Since the connection was successful, the positions of the server and client was changed.

The handheld was made the server since it was supposed to open a connection and handle multiple client requests. As this was not possible with the presence of the firewall, all the initial testing was done without the wireless node using only the LAN connections.

The programming for server can be described in two parts:

1. Programming the socket such that it reads the commands sent by the client and
2. Programming the serial port of the server such that the commands sent by the client are passed to the serial port. These are then passed to the control board from where they reach the motors.

2.5.3.1 Programming the Server

The following are the steps involved in creating the server side program.

1. Open a socket by creating a SocketServer object.
2. Create an input stream to read the commands from the client.
3. Create an output stream that sends these commands to the Serial Port of the handheld device.
4. Close the socket when done.

2.5.3.2 Programming the Serial Port

Programming the serial port of the handheld device was one of the most difficult part of this project. Initially, the working of the control board was tested by connecting it to the serial port of the desktop. This task was successfully accomplished with the help of the communications API supplied by Java in its Comm API package. Java provides a slick interface with the serial port in its communication package- the Comm API.

The actions performed at the server side were in the following order:

1. A socket is opened and the machine listens for all clients at a particular port number.
2. Simultaneously the serial port of the device is also opened.
3. The data input stream created by the socket is connected all the way to the serial port. So any data stream that the socket receives is immediately sent out of the serial port. This is done a string at a time.
4. Once the data string is sent out of the serial port the machine is ready to receive another string of data from the client.

5. The steps 3 and 4 are repeated till the connection is closed by the client.
6. Once the connection is closed, the machine stops listening on the port number and simultaneously closes the serial port.

To start a new connection the server program will have to be executed again.

It is important to know that the server can receive commands from any number of clients, provided all the clients have the IP address of the server and the same port number.

2.5.4 Connections

The control board was connected to the desktop computer using a RS232 null-modem cable. This cable was used since on the computer side pin 3 is the transmitter and pin 2 is the receiver. On the control board side, pin 2 is the transmitter and pin 3 is the receiver. Thus the null-modem cable basically connects pin 2 of the control board to pin 2 of the computer and pin 3 of the control board to pin 3 of the computer and pin 5- which is the common ground on both the sides is tied together.

Multiple servo motors are connected to the port B of the control board. The board is powered up by a 6V DC supply. A rechargeable 4 cell AA Alkaline battery pack was used for this purpose.

2.5.5 Settings of the Serial Port

Baud Rate	9600 Bps
Data Bits	8
Parity	No Parity
Stop Bit	1
Flow Control	No Flow Control

The serial port on the computer is configured to the settings as stated in the table above.

A default baud rate of 9600Bps is used since that is the rate at which the control board is programmed to accept bits.

2.5.5.1 Baud rate

Data bits are sent with a predefined frequency, the baud rate. Both the transmitter and receiver must be programmed to use the same bit frequency. After the first bit is received, the receiver calculates at which moments the other data bits will be received. It will check the line voltage levels at those moments. Since the baud rate of the controller was set to 9600Bps, the baud rate in the program was set to the same value.

2.5.5.2 Data Bits

Directly following the start bit, the data bits are sent. A bit value 1 causes the line to go in marking state, the bit value 0 is represented by a space. The least significant bit is always the first bit sent.

2.5.5.3 Parity

For error detecting purposes, it is possible to add an extra bit to the data word automatically. The transmitter calculates the value of the bit depending on the information sent. The receiver performs the same calculation and checks if the actual parity bit value corresponds to the calculated value. This feature was not used in this application and thus the parity was set to none.

2.5.5.4 Stop Bit

The stop bit identifying the end of a data frame can have different lengths. Actually, it is not a real bit but a minimum period of time the line must be idle (marking state) at the end of each word. On PC's this period can have three lengths: the time equal to 1, 1.5 or 2 bits. 1.5 bits is only used with data words of 5 bits length and 2 only for longer words. A stop bit length of 1 bit is possible for all data word sizes.

2.5.5.5 Flow control

The flow control was set to none because using bytes on the communication channel takes up some bandwidth. One other reason is more severe. Handshaking is mostly used to prevent an overrun of the receiver buffer, the buffer in memory used to store the recently received bytes. If an overrun occurs, this affects the way new coming characters on the communication channel are handled. In the worst case where software has been designed badly, these characters are thrown away without checking them. If such a character is XOFF or XON, the flow of communication can be severely damaged. The sender will continuously supply new information if the XOFF is lost, or never send new information if no XON was received.

Even though the hardware flow control is superior compared to software flow control using the XON and XOFF characters the main problem is, that an extra investment is needed. Extra lines are necessary in the communication cable to carry the handshaking information. Thus the flow control was set to none for this application.

A screenshot of the server is given below.

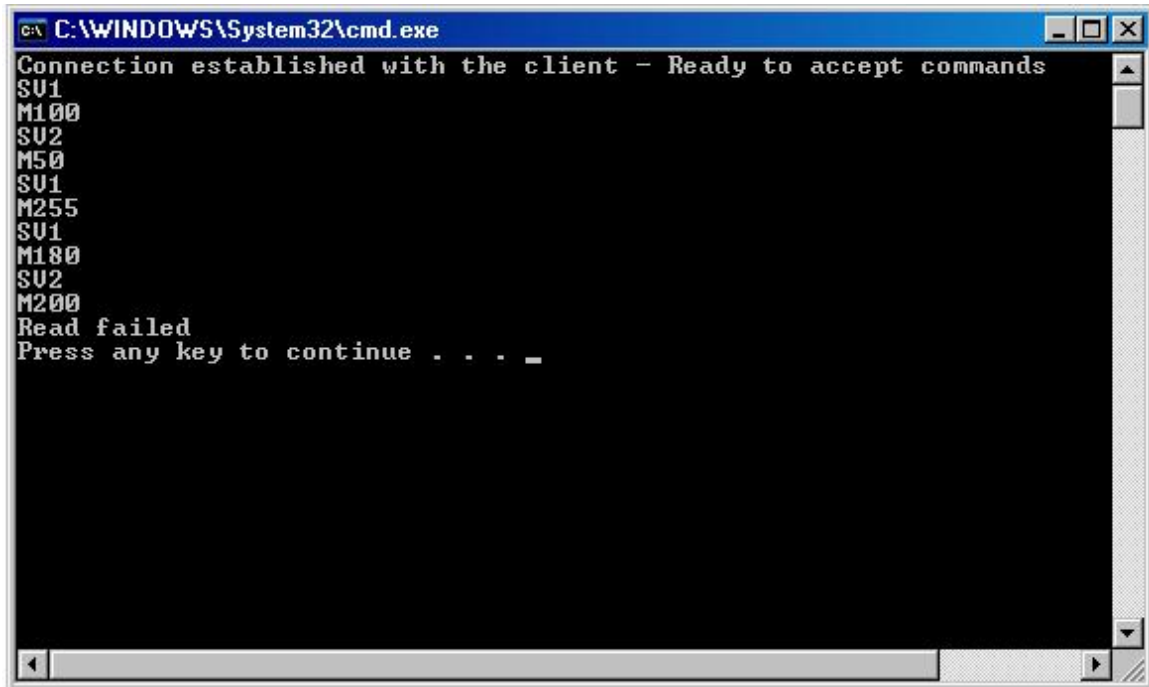


Figure 2.16. Server Side –Command prompt

A screenshot of the GUI for the server is given below

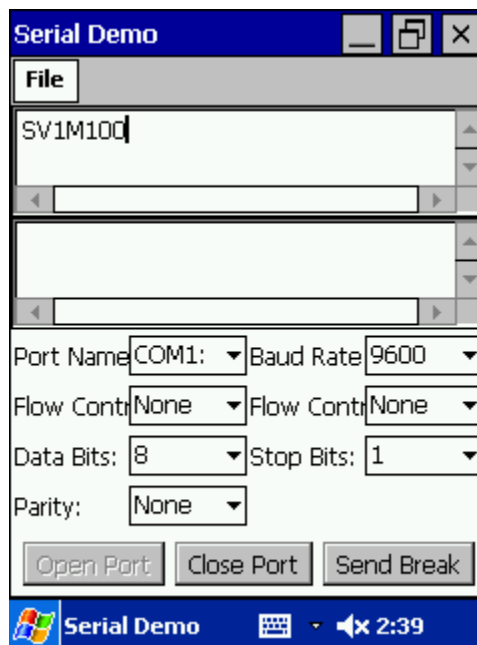


Figure 2.17. Server-Side GUI

The GUI is flexible in terms of the settings of the serial port. The user is able to select any COM port present on the computer. The baud rate can be selected from 300, 2400, 9600, 14400, 28800, 38400, 57600, 152000. Flow control can be of the type Xon/Xoff, RTS/CTS or none. Data bits can be 5, 6, 7 or 8. Stop bits can be 1, 1.5 or 2. Parity can be even, odd or none. These options were made available to the user so that it would be possible to connect other microcontrollers with different settings to the same port.

However there was no need for a GUI at the server end since this would be the appearing on the side of the robot which is not manned. The GUI was kept for testing and debugging the code. The last command entered by the client is visible in the bottom window. Once a new command is sent by the client, the previous command is flushed out of the window and the new command is displayed.

The program of the serial port was first debugged on a computer in the absence of the control board. This was done by connecting the two serial ports of the computer with the help of a null modem cable with full handshaking i.e. pin 2 of one port was connected to pin 3 of the other and vice versa. The diagram below shows the details of the cable used.

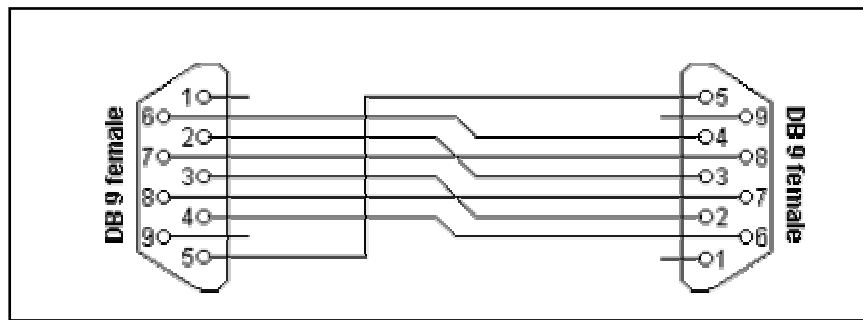


Figure 2.18. Null modem cable with full handshaking.

The program was run in two separate windows since two ports were opened. The top window of the GUI shows the data which is sent from the port. The bottom window of the GUI shows the data which is received by the port.

Once the program was fully tested, the attention was moved to the control board. The cable described in section 2.4.6 was used to connect the serial port of the computer to the control board. The program was verified by means of a visual feedback. The servo motors connected to the port B of the control board gave an accurate response to the commands entered by the user.

The next step was to alter this code such that it opens a socket which is accompanied with a simultaneous action of opening the serial port. After a successful connection was established between two separate computers, the code was again verified through a visual feedback. The servo motors moved by the exact amount as stated by the user.

To do this the “SerialConnections” program in the CommAPI package was altered such that any data sent by the client would be immediately sent to the port output.

However, since the board is programmed to receive commands in a specific format, the servo motors respond only to those commands. Any other string sent is ignored by the microcontroller. Thus the user will have to follow the commands mentioned in section 2.4.4 in order to get the servo motors to respond.

The next step was to implement the server program on the control board. For this, the PDA was connected to the control board by means of a serial cable. The PDA communicates with the client through the wireless network card. The two motors were connected to the control board and the board was powered by an external power supply. All commands mentioned in section 2.4.4 were successfully tested.

2.5.6 Implementing the Client

Programming the client –the desktop computer in order to send commands for the motor over the network involved the following steps:

1. Open a socket by creating a socket object
2. Create an output stream consisting of the commands for the control board.
3. Send the input stream over the network.
4. Close the socket when the application is completed.

The client program has the IP address of the handheld device. This program runs on the desktop computer. . To make a connection request, the client tries to rendezvous with the server on the handheld device and port.

A screenshot of the GUI for the client is shown below.

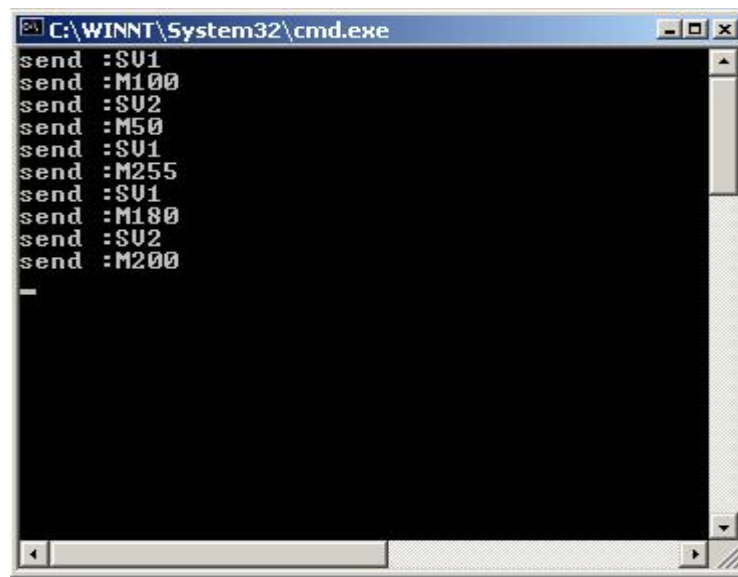


Figure 2.19. Screenshot of the Command Screen on the Client side.

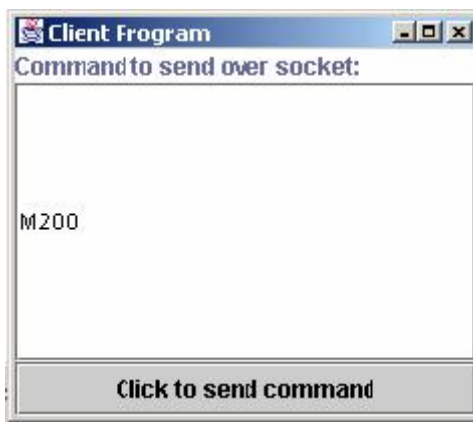


Figure 2.20. Screenshot of the Client Program.

The client program is written in such a way that once the user enters the command as to which servo motor is to be controlled and how much that motor should move, the user will have to click on the button “Click to send command”. Only then will the command be sent over the socket. Once the button is clicked, the data is flushed out and the screen

becomes blank. Then the command is visible on the command screen. The command screen maintains a list of all the commands entered by the user. When all commands are transmitted and the user no longer wishes to send any more commands, the connection can be closed by closing the GUI window on the client side.

After all the pieces were proved to be working correctly, the attention was then moved to modifying the source code for the server in terms of the handheld. Here, problems arose both from the discrepancies in the hardware between the desktop and the handheld, and the scarcity of programming resources caused by the relative infancy of PDAs.

A problem less anticipated but far more difficult to solve was the matter of communicating with the Cassiopeia's serial port. Preceding all attempts to write to the serial port was the challenge of assembling the appropriate combination of adaptors to link the two. The final bundle included three components, a serial port converter which converts the available port on the Cassiopeia to serial and USB port, a serial cable that converts the 20 pin serial port on the converter to a DB8 female connector and a DB9 male to DB9 male null modem cable. The trials did not end with the physical connection, however. Java has a set of routines to put data directly in the serial port. Class libraries to allow this on Windows or Solaris platforms are freely available from Sun, but the only such package available for a PocketPC 2002 handhelds comes commercially from SerialIO or Jeode from Insignia.

Through Java has made great strides towards its “write once, run anywhere” aims, the experience of migrating code to a form suitable for the PDA shows that the theory has not entirely made its way into practice. The leading edge APIs for the various platforms on which Java operates do not match, as seen in the fact that many of the classes supported in the Java2 language found on desktop platforms do not exist in the PersonalJava incarnation. In fairness, part of this is due to the trimming that occurred to reduce the memory footprint of PersonalJava.

2.5 Results and Conclusions

All the components were assembled and the final product was put to test. The robot mini-whegs alone weighs around 150 g. The entire setup which included the cables and the control board weighed around 450 g excluding the robot. It was observed that when the motors on the robot were turned on, the robot had some difficulty in moving around. This was only because of the weight that it was carrying. The robot was mounted on some wooden block such that its legs do not touch the ground and then the client and server programs were tested. The two motors could be efficiently controlled. The servo motor could be moved from extreme right position to center position and to extreme left position. Also, the speed could be controlled from a minimum start over speed of ... to a maximum speed of A picture of the final product is given below which includes the control board circuitry along with the serial cable and the PDA with wireless card mounted on the robot. A cardboard support was added to make the PDA sit in an angle on the robot. This was done mainly because the length of the PDA with the wireless card was more than the robot

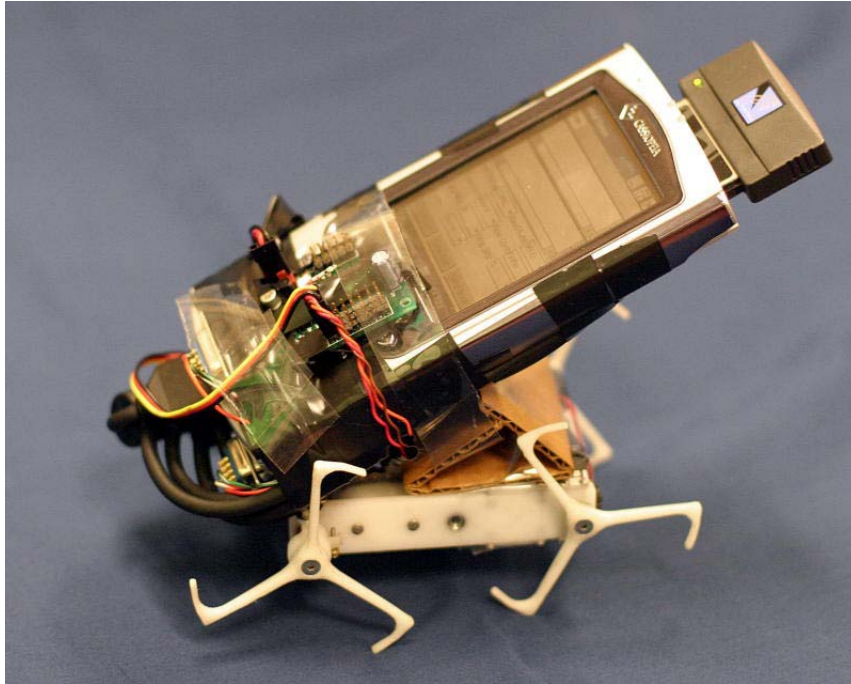


Figure 2.21 Mini-whegs with entire setup.

2.7 Suggested future Work

This report investigated the possibility of controlling miniwhegs wirelessly. This analysis needs to be extended for different kinds of robots. This would require a rigorous characterization of the system to account for flexibility of the algorithm.

The solution to handle wireless connectivity can be improvised. The solution presented in this report results in a comparatively heavy setup. This weight could be reduced so that it enables the robot to move more efficiently. The primary aim of the algorithm presented in this report is to build a system which could control the robot wirelessly. A better solution would be to look into the possibility of connecting the processor of the PDA directly to the control board, thereby eliminating the excess weight of the LCD screen,

the frame of the device and the cable. Also, one could look into the possibility of introducing some means of visual feedback with the help of the extra digital outputs present on the control board.

This project demonstrated the success of wireless control of a robot with the help of a PDA interface. Direct interface of the wireless control with the robot would be another interesting topic for future work.