DESIGN AND IMPLEMENTATION OF NETWORK CONTROLLED EYE

By

SARAH MAHMOOD

AQSA MALIK

Submitted to the Faculty of Electrical Engineering, Military College of Signals
National University of Sciences and Technology, Rawalpindi in partial fulfillment for
the requirements of a BE in Telecommunication Engineering

JUNE 2012

It is hereby certified that the project is duly completed by a group of EE department under the supervision of Maj Asim Rasheed Chaudhry.

____

(Signature)

Name: Maj Asim Rasheed Chaudhry

Date:

ABSTRACT

DESIGN AND IMPLEMENTATION OF NETWORK CONTROLLED EYE

This document gives the design and implementation of a small-sized Unmanned Ground Vehicle various modules of which are explained in detail. Each module was successfully implemented and tested.

The system aims to provide a practical solution to the problems encountered by large-sized UGVs in real time monitoring. The Network Controlled Eye is a UGV that can be used for various purposes to monitor areas where human presence is not feasible. Thus, it has been kept to a size that is small and easily controllable from the remote server. Furthermore, the system has been designed on Wi-Fi which makes it a wirelessly controlled robot.

A camera on the UGV captures real time video and sends the stream back to the remote server. Here, an application controls the movement of the NCE according to the video information that is displayed there. These control commands are sent via a TCP link on the basis of which the ARM11 board mounted on the UGV outputs values onto its serial port. The latter is further interfaced with a motor control circuit comprising a microcontroller which has been programmed to control two geared DC motors accordingly. The complete design is dissected into various modules that are discussed individually.

DECLARATION

No portion of work presented in this thesis has been submitted in support of another award or qualification either in this institution or anywhere else.

Dedicated to

Our parents without whose support and help, we couldn’t have accomplished this gigantic task.

ACKNOWLEGDEMENTS

All praises for Allah Almighty who gave us the strength to accomplish this mighty task despite numerous difficulties and hardships on our way.

We could not have been successful in our endeavours without the support, help and encouragement of our project supervisor Maj. Asim Rasheed Chaudhry.

A name that must be mentioned here with all the gratitude possible is that of Sir Intisar from the Department of Electrical Engineering for his unfaltering help and guidance throughout the year.

We are also very thankful to Sir Yasir from Mohammad Ali Jinnah University for his guidance in Java programming

Finally, this work could not have been completed without the sincerest help and support of our family and friends.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure No. **Page No.**

KEY TO ABBREVIATIONS

GUI: Graphical User Interface

HTTP: Hypertext Transfer Protocol

IP: Internet Protocol

JMF: Java Media Framework

MODEM: **Mo**dulator- **Dem**odulator

NCE: Network Controlled Eye

QoS: Quality of Service

RTP: Real Time Protocol

RTCP: Real Time Control Protocol

RTSP: Real Time Streaming Protocol

TCP: Transmission Control Protocol

TTL: Transistor-transistor logic

UART: Universal Asynchronous Receiver Transmitter

UAV: Unmanned Aerial Vehicle

UDP: User Datagram Protocol

UGV: Unmanned Ground Vehicle

USART: Universal Synchronous Asynchronous Receiver Transmitter

CHAPTER 1

INTRODUCTION

1.1. Overview

In the present age man has explored most parts of the world. Geographical, topographical and environmental information is just a click away. However, there still are certain areas where human existence is very difficult. These places cannot therefore be humanly monitored all the time. For many years now various groups and organizations have been offering different solutions. Several governments monitor such places using satellites. Unmanned aerial vehicles (UAVs) have also been used. But these tools provide limited view and limited real time information of the area only. Under these circumstances there is need for a tool that not only provides monitoring capabilities but also real time information of such areas. The Network Controlled Eye aims to develop a tool that will assist the monitoring of places where human existence is not possible or feasible.

Unmanned ground vehicles (UGVs) are robotic platforms that are capable of operating outdoors, replacing the need for humans to be present there. With counterparts in aerial and naval warfare, unmanned robotics is being deployed for both civilian as well as military use.

1.2. Objectives

In order to facilitate organizations belonging to various genres of work such as defence, wildlife, environmental and topographical data collection, the NCE aims to develop a teleoperated unmanned ground vehicle (UGV) using a toy car. Due to its small size, the UGV that is able to monitor and provide real time information of sites where mobility due to weather or other constraints is limited. This information can then be manipulated according to the needs of the organization deploying the UGV.

Defence and wildlife organizations can use the UGV in its present form for spying purposes and wildlife surveys respectively. On the other hand, environmental and topographical organizations would need to add additional circuitry and sensors for their respective data collection. Mine detection can also be done using the NCE-UGV with additional robotics hardware.

1.2.1. Features

The NCE is equipped with camera(s) and is connected to the control server through Wi-Fi. It is able to monitor sites where mobility due to weather or any other constraints is limited. Cameras mounted on the UGV collect real time information of the surroundings. The video data is then sent on the WIFI link connecting the client (UGV) and the server. Similarly, the control information is sent using the WIFI link from the server to the client on the basis of which the car moves. These features are explained below.

The NCE-UGV has the capability of being controlled wirelessly from the remote base station. The user at the base station uses the keyboard to maneuver the UGV according to his or her requirement.

Cameras mounted on the UGV collect real time information of the surroundings. This is important so that the user is able to see the whereabouts of the UGV and decide his or her requirements and therefore the movement of the UGV accordingly. The UGV has the capability to transmit the video stream back to base station through Wi-Fi.

This is enabled by a Wi-Fi module interfaced with the main controller board of the UGV.

A Wi-Fi ad-hoc link between the server base station and the UGV client is used to send control commands to the UGV. These are then interpreted and output onto the motor control interface on the client. This interface is a circuit that drives the DC motors according to the control information being received.

1.3. Platforms Used

The platforms used to communicate with the hardware and for designing the server application are expanded upon in this section. Here, the client side consists of an ARM 11 board with Linux installed on it as well as a programmed microcontroller. The server side consists of a JAVA application complete with a GUI, developed in Netbeans 7.0.1., running on the Windows OS.

1.3.1. ISIS Proteus

It is a powerful design environment which is able to define numerous aspects of the drawing appearance. It can support rapid entry of complex designs for simulation and PCB layout, or the creation of attractive schematics for publication. Proteus PCB design has the functionality combination of ISIS schematic capture and ARES PCB layout programs to provide a set of tools that is suitable for professional use.

This software has been used for designing the steering circuit of the UGV and for generating its PCB layout.

1.3.2. KEIL uVision

The µVision IDE from Keil combines project management, make facilities, source code editing, program debugging, and complete simulation in one powerful environment. The µVision editor and debugger are integrated in a single application that provides a seamless embedded project development environment. Keil offers an evaluation package that will allow the assembly and debugging of files 2K or less. This package is freely available at their web site. Keil development tools for the 8051 Microcontroller Architecture support every level of software developer.

uVision4 has been used for programming the 8051 microcontroller in C language and generating its hex file which was then checked in the simulation made in Proteus.

1.3.3. LINUX

Linux is a computer operating system assembled under the model of free and open source software development and distribution. The defining component of Linux is the Linux kernel.

Debian Linux has been installed on the ARM11 board. It includes the GNU OS tools and Linux kernel which is being used to run the C codes implementing socket programming in order to establish wireless link with the server. Additionally, the codes write on to the serial port for controlling the steering circuit of the UGV connected to the board via the serial to USB cable.

1.3.4. JAVA

JAVA was chosen as the programming language because of various features that give it an edge over others and make it more suitable.

It is cross-platform compatible and thus runs on almost every platform. Additionally, it has better error detecting and tracing capabilities and provides ease of GUI development.

It has its own native threads and performs excellently. There are numerous 3rd party libraries that can be included to work according to one’s requirements.

1.3.5. Netbeans

Netbeans is a Java IDE (Integrated Development Environment) that supports development in various languages including Java. It can run on various platforms including Linux, Windows, Solaris and Mac OS X.

It is based on the modular structure with a number of modules that can be used. With its drag-and-drop functionality it is ideal for the development of GUI based applications.

1.4. Organization of the Document

Chapter 1 expands upon the concept of UGVs and their importance in today’s scenario. It goes on to explain the objectives of the NCE, its scope in different scenarios, its advantages and the features that make it possible.

Chapter 2 deals with the literature review carried out at the beginning of our project while Chapter 3 illustrates and explains the project design in detail.

Chapter 4 deals with the explanation of the system modules. It describes the working of the project as well as the hardware and software requirements. Furthermore, it justifies the need for the selected hardware components and explains the choice of the software environment.

Chapter 5 elaborates upon the steps involved in the testing of the project while Chapter 6 explains future enhancements and the report is concluded. At the end of the document a list of any reference material used while preparing this document is provided along with the Java and C codes.

CHAPTER 2
LITERATURE REVIEW

This chapter expands upon the literature review carried out at the beginning of the project. It deals with the basic UGV study as well as the hardware required to implement it.

2.1. UGV Modes

There are two general modes of operation of Unmanned Ground Vehicles: Manual and Autonomous.

In the manual mode, the UGV is controlled by an operator from a remote location via a communication link. Sensors provide feedback to the human operator. This can be explained better with the help of an example. In a toy remote control car the user drives the UGV to the required area by sending control commands from the base station. Such UGVs are vastly in use today. Predominantly these come in handy when humans need to be replaced for example in hazardous situations or other non-feasible situations.

In the autonomous mode, the UGV operates autonomously on the surface of the ground. Such a vehicle has the ability to gain information about the environment by using sensory devices such as a video camera, work for extended durations without human intervention since it is autonomous, travel from one point to another without any navigation commands from a human user and detect objects of interest such as people and vehicles.

2.2. Wi-Fi

Wi-Fi allows devices to exchange data wirelessly (using radio waves) over a network.

The Wi-Fi Alliance defines Wi-Fi as any "wireless local area network (WLAN) products that are based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards".

Wi-Fi networks have limited range. A typical wireless access point using 802.11b or 802.11g with a stock antenna might have a range of 32 m (120 ft) indoors and 95 m (300 ft) outdoors.

Another wireless mode is the Wi-Fi Ad-hoc mode, a network formed without a router or access point in between. Hence the name, point-to-point network! Using wireless communication is advantageous in that it enables distant objects to be controlled remotely. Such systems can be designed as secret systems having low cost and more flexibility.

2.3. Hardware

This section deals with the literature review regarding the hardware requirements carried out at the beginning of the project. It includes all study carried out with regard to the equipment used.

2.3.1. Power Control Circuitry and DC Motor

For the motion of vehicle in the desired direction it is necessary that the direction of rotation of motors should be controlled according to the commands. Since the direction of motion of vehicle is controlled by DC motors, the power circuit should properly adjust the polarity of DC voltage supplied to motors for motion in desired direction. Also when stopped, this circuitry should not give power to motors. A DC motor is an electric motor that runs on direct current (DC) electricity.

It generates torque directly from DC power supplied to the motor by using internal commutation, stationary permanent magnets, and rotating electrical magnets.

2.3.2. Serial Communication

Data communication generally takes place in two ways: parallel and serial. The parallel mode data transfer is fast and uses more number of lines. This mode is good for short range data transfer.

Computers transfer information (data) one or more bits at a time. Serial refers to the transfer of data one bit at a time. In this type of communication each word (i.e. byte or character) of data is sent or received one bit at a time. Each bit is either on or off. The speed with which the serial data is sent is expressed as bits-per-second ("bps") or baud rate. It represents the number of ones and zeroes that can be sent in one second.

Generally serial devices or ports are either labeled as Data Communications Equipment ("DCE") or Data Terminal Equipment ("DTE"). These two differ in that every signal pair, for example transmit and receive, is swapped. A serial null-MODEM cable or adapter is used to connect two DTE or two DCE interfaces together, that swaps the signal pairs.

2.3.2.1. RS-232

RS-232 is an electrical interface for serial communications. RS-232 comes in 3 flavors (A, B, and C), each defining a different voltage range for the *on* and *off* levels. The most commonly used variety is RS-232C, which defines a mark (on) bit as a voltage between -

3V and -12V and a space (off) bit as a voltage between +3V and +12V. This is the general form of communication that is being used in the project.

2.3.2.2. Asynchronous Communication

For the computer (or any other device) to understand the serial data coming to it, there should be a way to know where one character ends and the next begins.

In asynchronous mode the serial data line stays in the ON (1) state until a character is transmitted. A *start* bit precedes each character and is followed immediately by each bit in the character; there can be an optional parity bit, and one or more *stop* bits. The start bit is always a space (0) and tells the computer that new serial data is available. Data can be sent or received at any time. Hence, the name!

2.3.2.3. PC to Motor Circuit Interface

The simplest connection between a PC and microcontroller requires a minimum of three pins, RxD (receiver, pin2), TxD (transmitter, pin3) and ground (pin5) of the serial port of computer as shown in Figure 2.1.

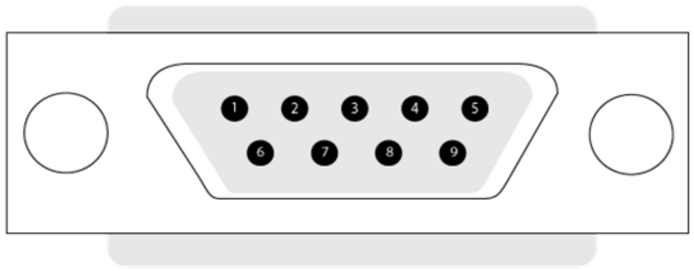

Figure 2.1 Serial Port Pins

The microcontroller has an inbuilt UART port used for carrying out serial communication, which is done in asynchronous mode i.e. that is simultaneous transmission and reception. A serial port is a physical interface that enables bit by bit data transfer between a PC and any external device.

AT89C51 microcontroller uses different software instructions in order to communicate with the serial devices. These are needed so that its baud rate is the same as that of the communicating PC. Timer1 sets the baud rate of serial data transfer for the microcontroller used in mode2. This mode is an 8-bit auto reload mode. Table 2.1 gives the values for TH1 for setting different baud rates compatible with the PCs.

Table 2.1 Hex values for different baud rates

Baud Rate (bps)	**TH1 (Hex value)**
9600	FD
4800	FA
2400	F4
1200	E8

2.3.2.4. Serial Ports Under Linux

Serial port and USB ports are considered as files in linux, the open() function is used to access it. Device files, when accessed as a root user are usually not accessible.

Writing the write() system call sends data on to the port – just the same as when using Hyperterminal in Windows OS. The write function returns the number of bytes sent or -1 if an error occurs.

In the same way it can read from a serial port but only one way information is required to travel while the close system call is used to close the port.

The number of data bits, baud rate, parity, hardware flow control and stop bits are set using the c_cflag. There are constants that enable the setting of the number of data bits to 8, baud rate to 9600, parity to none, flow control none and stop bits 1 since a single character is being sent at a time which is an 8-bit value.

2.4. Client-Server Architecture

The client-server architecture is based around an asymmetric/symmetric (depending on the protocol used) mode of communication between a client and a server. Some protocols, like the FTP for example, identify one as the master and the other as slave while for others such as Telnet, either side can play as the master or slave. The application running at the server end listens for connection requests from the client at a well-known port. Once the connection is established, it services the client as required.

In addition to such connection-oriented services, there are others such as the datagram sockets that do not require any connection establishment prior to communication.

2.5. Video Streaming

Video Streaming is a method that involves a server sending video to one or multiple clients such that the medium can be viewed while still being transmitted. It surpasses the need for complete download of the file before the end-user can start accessing it.

Live streaming, delivering live over the Internet, involves a camera for the media, an encoder to digitize the content, a media publisher, and a content delivery network to distribute and deliver the content. [1] The user is able to watch the video almost as soon as it arrives and does not have to wait for the whole file to download first.

On the contrary, the downloading/progressive downloading method works using the HTTP protocol, where the video file is downloaded before it can be viewed. This is not true streaming; only a passable imitation.

A video is basically a series of still images called ‘frames’, whose constant transmission gives the impression of a continuous video. More the frames per second, better the video quality.

2.5.1. Video Codec

A Video Codec is a device or software that performs **co**mpression (usually lossy) and **dec**ompression of the video file in order to reduce redundancy in the data. This usually involves both spatial image compression as well as motion compensation in time.

Video codecs represent analog video signals in digital signal form. The first step in compression is to store the analog image in a YCbCr color space. Before the basic encoding starts, some downsampling is also done both spatially as well as temporally.

2.5.2. Video Resolution

Video Resolution refers to the dimensions of the video in terms of the pixels displayed in each direction. For example, a resolution of 1024 x 768 means that the displayed video is 1024 pixels wide and 768 pixels long.

Several formats are available and are given in Table 2.2.

Table 2.2 List of Common Video Formats

Format Name	**Resolution**
QCIF	176 x 144
SCIF	256 x 192
SIF	352 x 240
CIF	352 x 288

2.5.3. Protocols – RTSP and RTP

RTSP – Real Time Streaming Protocol - is an application layer protocol used to set up and control multimedia sessions between endpoints with another protocol (usually RTP) responsible for the actual delivery of the content. It, like HTTP, uses TCP as the transport layer protocol for maintaining an end-to-end connection. It defines the control sequences required to control multimedia playback.

The RTP – Real Time Protocol - defines the packet format for delivering the streaming multimedia content over IP networks. It is used in conjunction with RTCP (Real Time Control Protocol) which is responsible for monitoring the QoS as well as for synchronizing multiple streams. Even port numbers are used for the transmission and reception of RTP streams with RTCP using the next higher odd port number.

The RTP packet format is given in Figure 2.2.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.1 RTP Packet Format

2.5.4. Java Media Framework – JMF

The JMF is a Java API that deals with real-time multimedia content. It enables audio, video and other time-based media to be added to applications and applets built on Java technology. [2]

The JMF architecture consists of three stages: Input which can be a capture device, a file or a network stream, Processing which consists of codecs and effects and Output which can be a video renderer, a file or a network stream.

It comes with a GUI based application (shown in Figure 2.3) that can be easily used for video streaming.

Figure 2.2 JM Studio: a GUI based application available from JMF

2.5.5. VLC Player

The VLC Player is a software capable of performing numerous functionalities including video streaming. Figure 2.4 gives an overview of the VideoLAN streaming solution. It can be used as a client as well as a media server. It supports numerous protocols, codecs, transcodings and data rates and can work on multiple platforms including Linux, Windows, Mac OS X, Solaris etc.

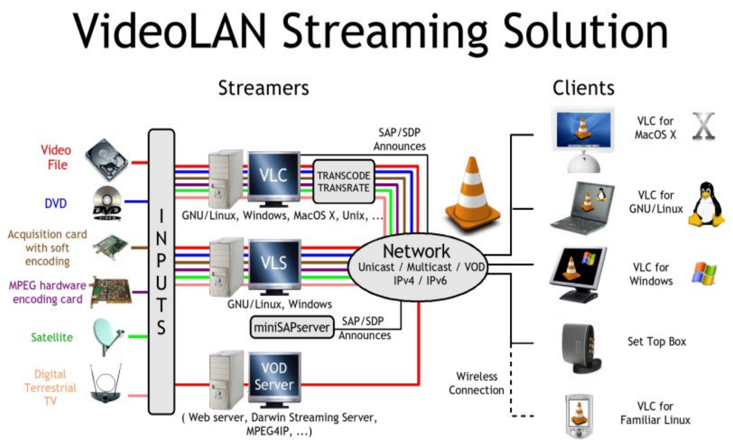

Figure 2.3 VideoLAN Streaming Solution

2.6. Sockets

A socket is a virtual ‘door’ between an end-process and the transport layer protocol - TCP or UDP. It carries the IP address of the end-system and a PORT number thereby uniquely identifying the process.

In TCP, the server must first be running and listening at a well-known port while the client tries to connect with it (requests connection) using its IP address and the port number.

The server then creates a new socket for communication with the client and keeps the other socket reserved for listening purposes. In UDP, no connection is made between the client and the server. Hence, the sender needs to explicitly attach the IP address and the port number with each packet sent.

CHAPTER 3

PROJECT DESIGN

The NCE-UGV is based on the Client-Server architecture where the UGV is the client controlled by a server application wirelessly. Figure 3.1 shows the block diagram for the system.

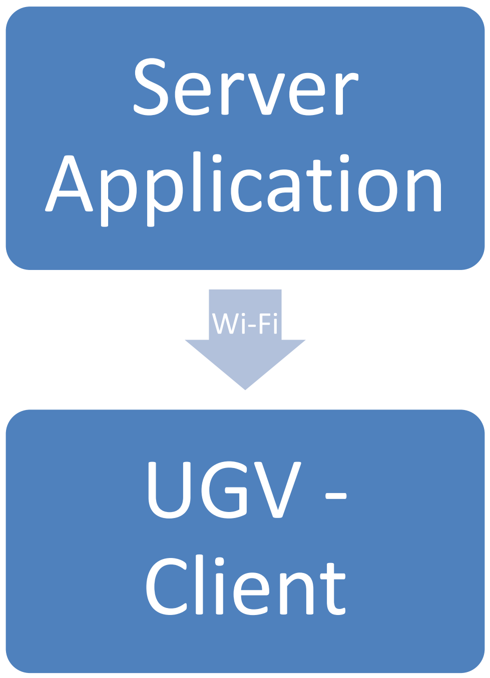

Figure 3.1 Client-Server Architecture

The UGV client and the server application are communicating via a TCP socket. A video stream from an IP camera mounted on top of the UGV is being set back in real time to the server end via HTTP.

3.1. UGV Overview

The NCE-UGV is an ordinary vehicle having three wheels: one front and two rear.

The rear wheels are powered by two independent, same speed DC geared motors. The direction of rotation of each motor determines which path the vehicle will follow. A self made platform has been used for NCE which operates in the manual mode.

The UGV is operated in manual mode i.e. it is controlled remotely from the server computer. For this purpose, a Wi-Fi link is used for wireless control whose range is 100m. The user controls the UGV by using the keyboard at the server computer. A camera is mounted on the car that enables the user to see the surroundings of the UGV and maneuver it according to his own requirement.

3.2. Hardware

Figure 3.2 shows the hardware which consists of two independent modules in the client part: the IP Camera and the ARM11 board. The former is connected to the server computer via a Wi-Fi link while the latter is connected via an ad-hoc link.

The client part consists of a robot (a car) mounted with a video sensor (a camera) and a motor control circuit. The latter is interfaced with a controller board that receives information from the server and controls the robot accordingly (movement control). Additionally the video stream is sent back to the server and displayed there.

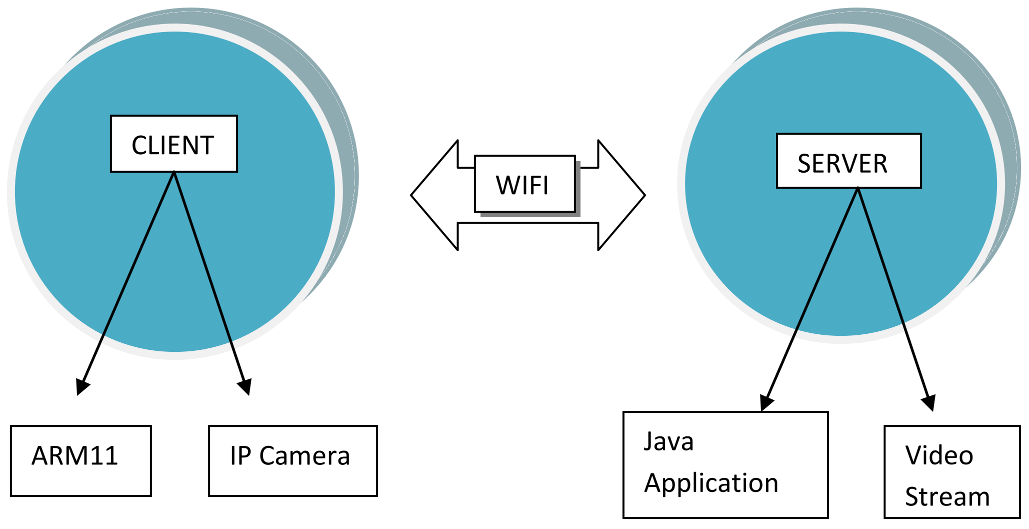

Figure 3.2 Project Design

The following sections expand upon the equipment used at the client end.

3.2.1. Geared DC Motors

2 x 12V DC geared motors control the forward and backward movement of the car, as well as the right and left movement. Each has a heat sink mounted on it to absorb heat. The motors are high torque low speed, and are capable of driving 4-6kg load. Figure 3.3 shows a simple DC motor.

Figure 3.3 DC motor

3.2.2. ARM11 board

OK6410 is an Embedded Computer (Single Board Computer) based on the 667MHz Samsung S3C6410 (ARM11) microcontroller. OK6410 embodies abundant of built-in resources and powerful video processing capacity, which make the OK6410 reliable for the development of higher-end products. The Debian Linux operating system has been installed on this board.

Figure 3.4 shows the ports and interfaces of the OK6410-A board.

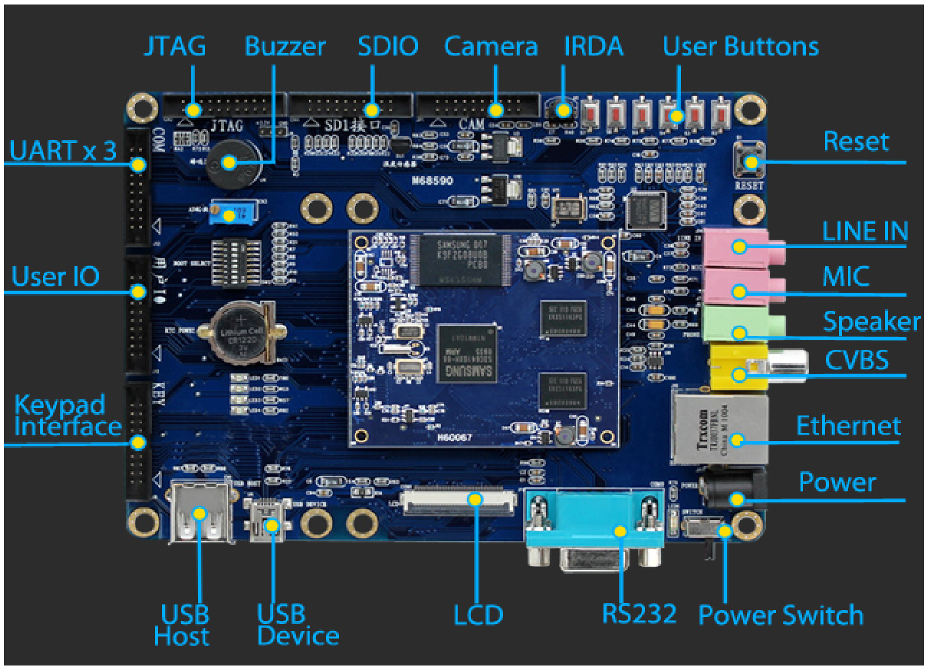

Figure 3.4 OK6410-A Board

3.2.3. Wi-Fi Card

The WM-G-MR-09 (Marvell8686) SDIO Wi-Fi module has been used with the ARM11 board mounted on the UGV. This provides the functionality necessary for enabling Wi-Fi on the client side.

It has support for Linux, WindowsCE6 and Android operating systems. It provides Wi-Fi connectivity on WEP, WPA, WPA 2 and TKIP encryption schemes. Figure 3.5 shows an image of the Wi-Fi module that is inserted in the SDIO interface shown in Figure 3.4.

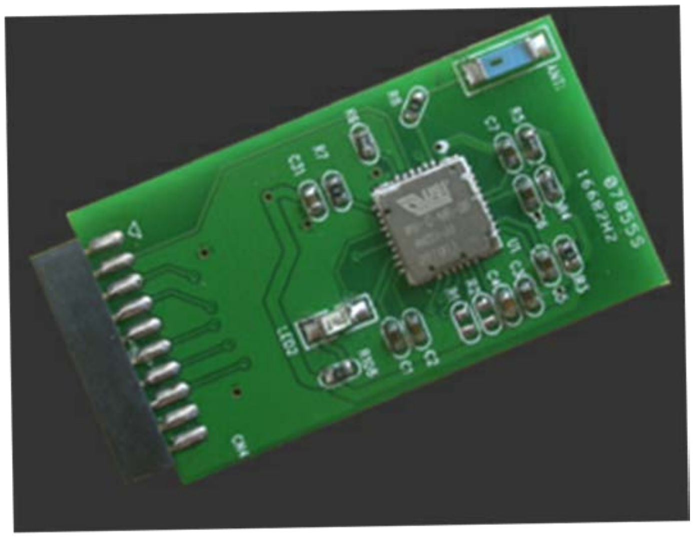

Figure 3.5 WM-G-MR-09 (Marvell8686) SDIO Wi-Fi module

3.2.4. AT89C51 Microcontroller

The microcontroller incorporated in the project is from the 8051 family of the microcontrollers namely “AT89C51”. In this project the 40 pin package is used. It can be used with external clock up to 24MHz. It has 4KB program memory. Its EEPROM is 4KB and SRAM is 128 bytes. It has 2x16-bit timers and has four ports in total. Every pin of those five ports can be used for either input or for output. Figure 3.6 shows the AT89C51 pin configuration.

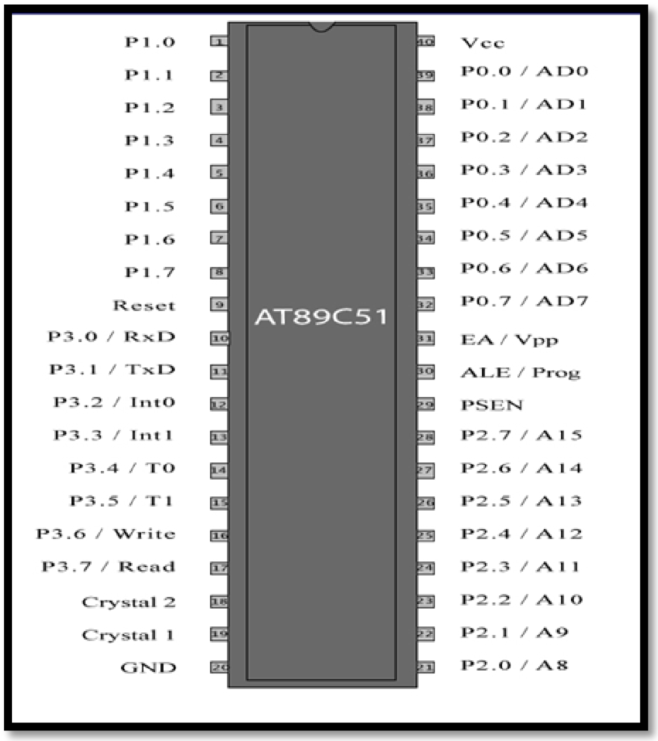

Figure 3.6 AT89C51 Pin Configuration

The purpose of this microcontroller is to control the motors via the H-bridge IC and to connect with the ARM11 board at serial interface via the max232 IC.

3.2.5. Max232 IC

The purpose of this IC is to convert the RS-232 port signals to TTL level. It is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals. [3]

The RS232 voltage output levels are driven by a single +5 V supply. These levels are usually ±7.5 V. Hence, it is suitable for devices that have a power supply range of 0 to +5 V.

Figure 3.7 shows a MAX232 IC.

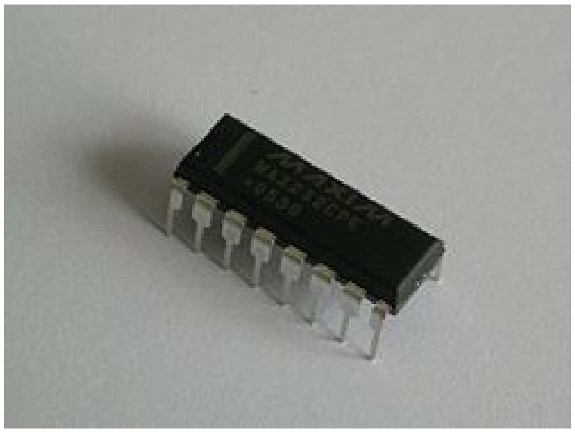

Figure 3.7 MAX232 IC

The voltage level conversions that take place are given in Table 3.1.

Table 2.1 MAX232 Voltage Conversions

RS232 Voltage (V)	**TTL Voltage (V)**
+3 to +15	0
-3 to -15	5
-3 to -15	5
+3 to +15	0

3.2.6. L293D

It is a simple quadruple half-H driver (only 600 mA) designed to accept standard TTL logic levels, widely used to control small dc and even stepper motors.

A single chip enables the user to connect and control two DC motors as given in Table 3.2.

Table 3.2 L293D Pin Outputs

Enable	Input1	Input 2	Result
H	H	L	Motor turns right
H	L	H	Motor turns left
H	L or H	L or H	Fast stop
L	L or H	L or H	Slow stop

The L293D has an auto over-temperature protection. This enables the IC to stop working automatically if the temperature gets too high and can be controlled very easily with any microcontroller.

3.2.7. Wireless IP Camera ID520

The camera used is a basic lightweight wireless IP camera ID520 (with additional Pan/Tilt/Zoom capabilities). It gives 270° left-right and 90° up-down movement.

The camera's embedded operating system supports multiple protocols like the TCP/IP, UDP, SMTP, FTP, UPNP and HTTP. The hardware operates on DC 5V, 2A.

The camera can be easily accessed using a simple Web browser which brings up the camera's own GUI that can be used to view, control and manage it. Additionally, the video segments can be recorded on the computer.

The camera also has DDNS support which enables it to be accessed despite frequent changes in IP. Setting a username and password helps protect the privacy of the camera.

Figure 3.8 shows the ID520 Wireless IP Camera.

Figure 3.8 Wireless IP Camera ID520

3.2.8. Power Supply

An onboard power supply has been designed for the UGV using Tesla MA7805 and a 12 V dry cell battery. Figure 3.9 shows a 12 V dry cell battery.

Figure 3.9 Dry Cell Battery

The Tesla MA7805 is a voltage regulator that takes its input from the 12V supply battery and provides 5V, 2A output for the ARM11 board as well as for the on board camera (since both operate on 5V). Figure 3.10 shows two Tesla MA7805 voltage regulators.

Figure 3.10 Tesla MA7805

CHAPTER 4

SYSTEM MODULES

This chapter expands upon and explains the numerous system modules in detail. Each module is dissected to contain every step implemented in the course of the project.

4.1. Client

In order to control the UGV from the server, a Wi-Fi link is used. The receiver of this Wi-Fi link is interfaced with the ARM11 board which is then connected to a motor steering circuit via a serial port. As the keys are pressed on the server PC, commands are issued which are then sent to the UGV via a TCP socket. On the basis of these commands, the UGV sends output to the 8051 microcontroller for forward, backward, left and right movement.

Figure 4.1 shows the block diagram representing the manual control of the UGV at the client end.

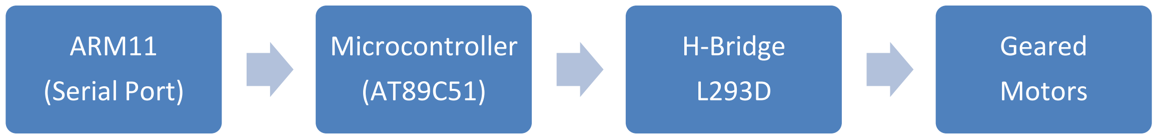

Figure 4.1 Block diagram representing the client part

With each key pressed the motors are on for 1 second and then stop. The steering mechanism of the UGV is given in Table 4.1.

Table 4.1 Reactions to Key Presses

Keys	**Motion**	**Motor Rotation**
W	Forward	left anticlockwise, right clockwise
A	Left	Right clockwise, left stops
S	Reverse	Left clockwise, right anticlockwise
D	Right	Left anticlockwise, right stops

A basic serial interfacing circuit is used to interface the controller with PC through serial port. Figure 4.2 shows the schematic.

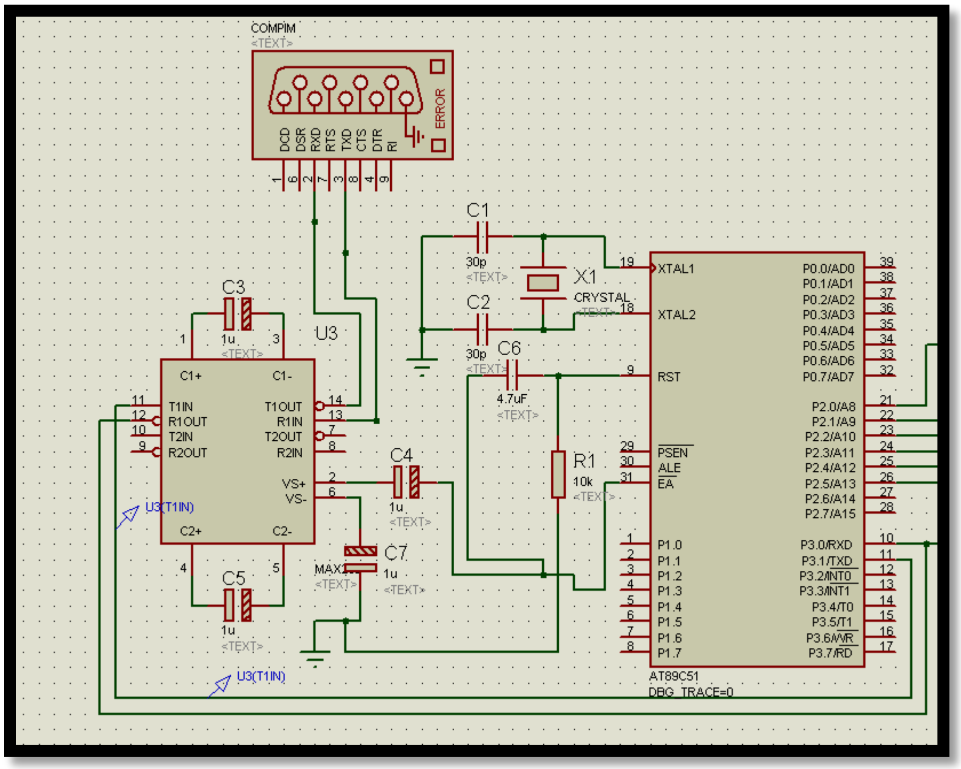

Figure 4.2 Interfacing Circuit made in Proteus

This circuit uses MAX232 IC that converts signal from RS232 level to TTL and TTL signal is used as input to controller using UART module of 89C51.

RS232 levels have to be converted down to lower levels in order to communicate with different microcontrollers. This is done using the MAX232 IC.

Serial RS-232 communication works in the voltage range of -15V to +15V. TTL, on the other hand, works in the voltage range of 0V and +5V. Table 4.2 shows these conversions.

Table 4.2 RS232 to TTL Conversions

RS-232	**TTL**	**Logic**
-15V … -3V	+2V … +5V	High
+3V … +15V	0V … +0.8V	Low

Thus the RS-232 voltage levels are far too high TTL electronics, and the negative RS-232 voltage for high can’t be handled at all by computer logic. For receiving serial data from an RS-232 interface this voltage has to be reduced to an appropriate level. Also the low and high voltage levels have to be inverted.

Figure 4.3 shows a simple MAX232 circuit used to convert RS232 signals to TTL level. It uses five capacitors.

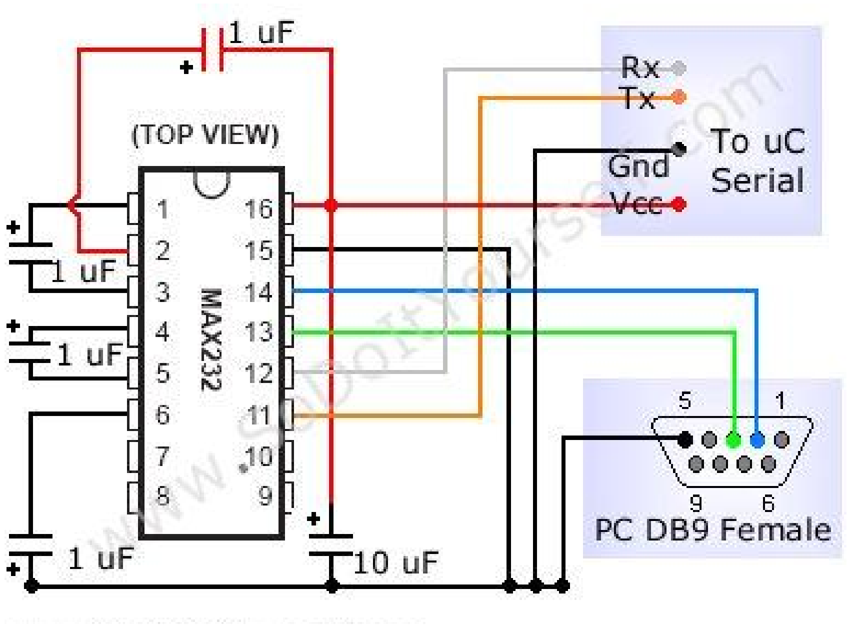

Figure 4.3 MAX232 Circuit

ARM11 sends the signal is sent onto its serial port in ASCII form. From MAX232, TTL signal is given as input to the controller using its UART module. The pin no. 11 and 10 of AT89C51 is Tx and Rx pin respectively. Here UART module of controller is used in order to communicate with computer through serial port.

4.1.1. AT89C51 Serial Interface

A serial communication interface can either be a USART or UART, depending on whether it supports both synchronous and asynchronous modes. A USART supports both asynchronous and synchronous modes, whereas a UART supports only asynchronous mode. USART stands for "universal synchronous asynchronous receiver transmitter."

Each USART module uses the SCON (Serial Control Register), SBUF (Serial Interface Buffer Register) and SMOD (Double Baud rate) registers to control its operation.

In the UART Asynchronous Mode, the USART uses the data format of one start bit, eight or nine data bits, and one stop bit for data transmission and reception. The least significant bit is transmitted and received first. The transmitter and receiver use the same data format and baud rate. The baud rate of the microcontroller is set with the help of timer for serial communication. It activates the timer and sets the mode 2 with the help of TMOD register which signifies 8-bit auto reload and SCON register is used to set the serial channel mode to 8-bit variable baud rate.

Whenever user software has data to be transmitted, it writes into the SBUF register which takes 8 bits at a time. This results in the RI flag being set to 1 which stops further transmission. After the register’s value is shifted to a port, the RI flag is reset.

When controller receives the signal in ASCII format from MAX232 it sends the specific bit pattern at its PORT1 according to the character received. At PORT1 8 bit pattern is sent. The program burnt in controller has cases on what character is received and depending upon these cases specific bit pattern is sent to respective ports of controller. The bit pattern is adjusted such that it drives the H-bridge of the motor driving IC.

4.1.2. AT89C51 Timers

The AT89C51 has two general purpose 16bit timers. They are identified as timer0 and timer1. They can be configured in variety of modes according to the requirements. Both these timers can work independently of one another.

The timer has been used in mode 2 for setting the baud rate. Mode 2 configures timer 0 as an 8-bit timer (TL0 register) that automatically reloads from the TH0 register TL0 overflow sets TF0 flag in the TCON register and reloads TL0 with the contents of TH0, which is preset by software. The benefit of using this mode is that in other modes the timer has to be reset which takes execution time of instructions while in this mode the microcontroller hardware does it automatically.

For turning the motors for 1sec, a delay of 1sec has been generated using the timer in mode 1, in which timer register is run with all 16 bits. Mode 1 configures a 16-bit timer with the TH0 and TL0 registers connected in cascade. The selected input increments the TL0 register.

4.1.3. L293D H-Bridge

The DC motors are controlled with the help of L293D IC which is interfaced directly with the microcontroller. It gets the TTL level signals from microcontroller port 2 and gives the output to the motors accordingly. Figure 4.4 gives the schematic of the circuit.

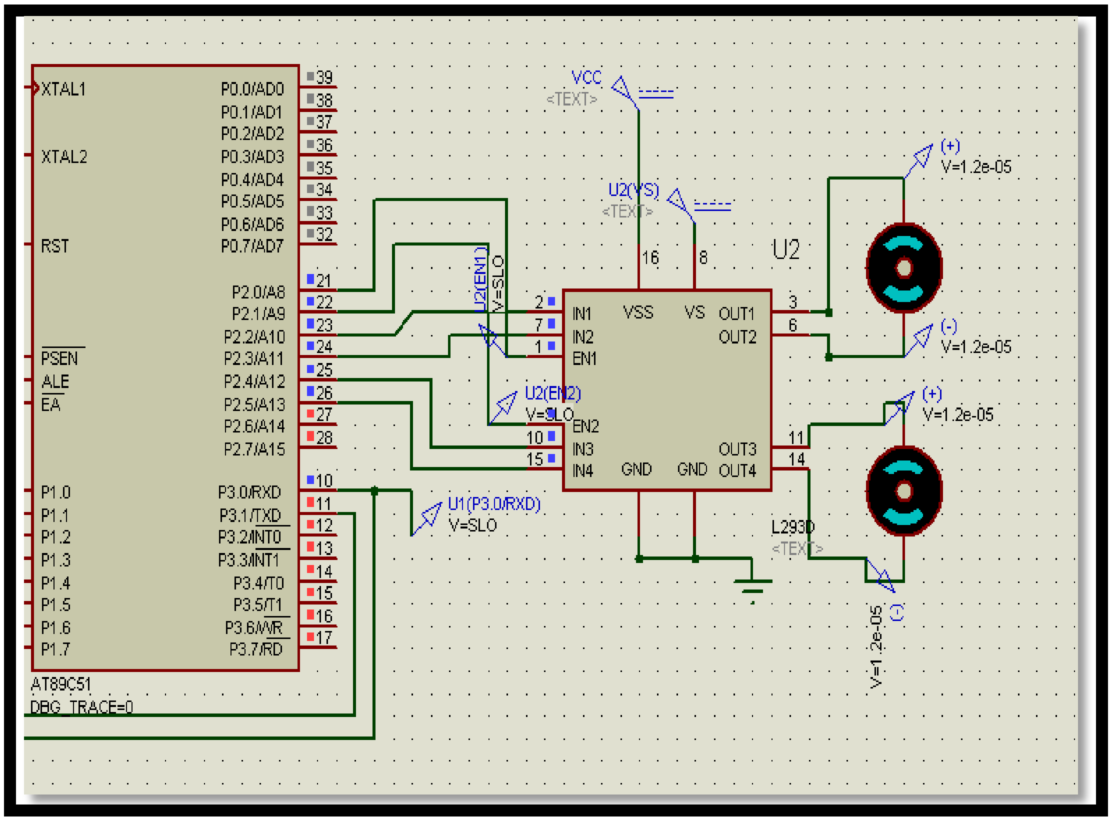

Figure 4.4 Schematic showing 89C51 pin outputs controlling the motors

This is how UGV is controlled manually from base station computer from wireless radio link. User at base station maneuvers the UGV by pressing keys of keyboard. The reactions to different key presses are given in Table 4.3.

Table 4.3 UGV movement in response to different key presses

Keys	**Description**
A	Turn Left
D	Turn Right
S	Move Backward
W	Move Forward

4.1.4. Programming in ARM11 Board

Using the OK6410-A (ARM 11) board, several steps are performed before making it fully functional. These are Installing Linux (Debian) on the board, configuring Internet availability on it, downloading and installing the GCC Compiler and finally configuring the wireless module for adhoc communication

Since the ARM11 board has been mounted on the UGV, the Linux environment has been used to run the C codes. The socket programming code is used to develop the connection with the server PC using the Wi-Fi which is in continuous listening mode. Interfaced with this code is the code for writing on to the USB port. Whenever a character is received from the server it automatically takes the 8 bit value (i.e. a single character at a time) and writes it on to the USB port which gives it to the microcontroller. The server goes back to the listening mode.

4.1.5. Control Circuitry of Geared Motor

Electromagnetism dominates the operation of an electric motor. A current carrying conductor generates a magnetic field when placed in an external magnetic field. This causes a force to act on it. This force is proportional to the current in the conductor and the strength of the magnetic field. The DC motor generates rotational motion by exploiting the magnetic interaction between the current-carrying conductor and the external magnetic field.

A geared motor consists of an AC or DC motor with an integral gearbox or gear head. This gear head steps the delivered speed up or down.

300RPM 12V DC geared motors have been used in the project. Each one is capable of giving a torque of 30Kgcm. For controlling the direction of UGV signal is to be generated by controller and it keeps the motor on for 1sec.

Figure 4.5 shows the control flow of the DC motor. The microcontroller output is programmed to control the H-bridge which then drives the DC motors.

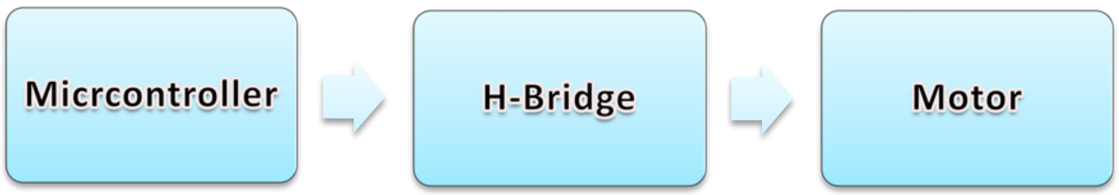

Figure 4.5 Block diagram showing the control of DC motors

4.1.6. H-Bridge

An H bridge is an electronic circuit that enables a voltage to be applied across a load in either direction. Such circuits are often used in robotics to allow DC motors to run forwards and backwards.

H-bridge is so named because it has four switching elements forming the vertical bars of the ‘H’ with the motor forming the cross bar as shown in Figure 4.6.

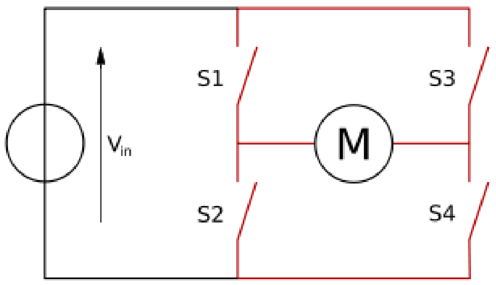

Figure 4.6 H-bridge Driver

The four switches are called, high side left, high side right, low side right, and low side left (in clockwise order).

The switches are turned on in pairs, either high left and lower right, or lower left and high right. Both switches on the same "side" of the bridge are never turned on together for this creates a short circuit between the positive and negative terminals of the battery. This is referred to as shoot through. A sufficiently powerful bridge will absorb that load and the batteries will drain quickly. Usually however the switches melt.

To power the motor, two diagonally opposed switches are turned on. Figure 4.7 shows the current direction when high side left and low side right switches are turned on.

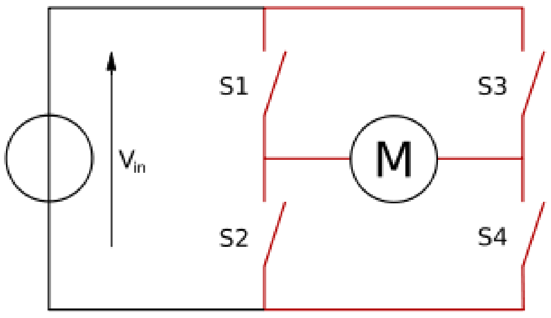

Figure 4.7 Current flow direction for positive movement of motor

This causes the current to flow and the motor begins to turn in a "positive" direction. If the high side right and low side left switches are turned on, the current flows in the opposite direction and the motor rotates in the “negative” direction.

4.1.7. Camera Selection

The project required a minimal video camera able to stream video over a network/on the same network. For this there were several options available such as an IP Camera, a Webcam or a CMOS Camera.

Selecting the CISCO-2500 IP Camera for video streaming in the NGN Lab, a video stream was obtained on LAN using the VLC Player. The protocol used was RTSP.

This particular camera however was too heavy and expensive and had functionalities that were not required. Thus, a basic lightweight Wireless IP Camera ID520 (with additional Pan/Tilt/Zoom capabilities) which was sufficient for the project was chosen. It had 270° left-right and 90° up-down movement and used the http protocol. Furthermore, it operated on DC 5V, 2A.

This camera is available both as a wired and a wireless device. It first has to be configured for operation on the wireless network of choice by using it as a wired device and connecting it via LAN cable to the PC. Its default IP is 192.168.1.111 which is then used to access it at port 81 using HTTP.

4.2. Server

The server end consists of a Java application developed in Netbeans 7.0.1., Windows OS.

It has three modules: the GUI, TCP socket and motor control code. These have been programmed so as to take the commands from the user in the GUI, which are then processed using the motor control code and are sent via the TCP socket to the client.

4.2.1. GUI – Graphical User Interface

This is the part of the application that the user interacts with. It has a text box that takes in the user commands (ASWD ‘Key Pressed’ Events). Along with it is a ‘Clear Button’ that once pressed clears the text field.

4.2.1.1. Look and Feel

The ‘UIManager.setLookAndFeel’ sets the Look (refers to the appearance of the GUI components) and Feel (refers to the way these components behave) of the GUI. The L&F (Look and Feel) has been set to the ‘MetalLookAndFeel’ which is the Java L&F that is the same on all platforms (also referred to as the CrossPlatformLookAndFeel). It is the default L&F.

Other options for the L&F are the SystemLookAndFeel that is native to the system the application is run on, Synth which allows the application developers to create their own L&F using an XML file and Multiplexing which allows for a number of L&Fs to be implemented at the same time.

The ‘UIManager.put’ is the part of the code that has been set to turn off metal's use of bold fonts by setting the inputs to the method ‘put’ of class UIManager to "swing.boldMetal" and Boolean.FALSE.

4.2.1.2. Key Listener

When a user types at the keyboard, certain kinds of ‘key events’ are fired by the component having the keyboard focus. The typing of a Unicode character fires the Key Typed Event while the pressing and releasing of a key fire the Key Pressed and Key Released Events respectively. In the code, the focus has been set to return to the typing area after the CLEAR button is pressed.

4.2.1.3. Action Listener

An action listener is an event handler that listens for an action that the user performs. In the code, it is the click of the CLEAR button. Here the operation performed is the clearing of the text field and returning the keyboard focus back to it using the typingArea.setText('') and typingArea.requestFocusInWindow() commands.

4.2.2. TCP Socket

The server side TCP socket listens for client connection requests at a particular port (known to the client). Once the client attempts connection with the server, it accepts it and opens the Data Output Stream which transfers the commands as the keys are pressed.

4.2.3. Motor Control Module

This part of the code deals with the actions performed as reactions to the key typed events. Depending on the key, it sets the value of a string variable ‘s1’ equal to the alphabet pressed (possible only for A-S-W-D keys) and sends it via the Data Output Stream to the client.

CHAPTER 5

TESTING

This chapter deals with the testing of the project and the results that were obtained at different stages. It has been divided into two parts: the video stream section and the TCP link section. Each section deals with the steps that were performed in order to successfully test the NCE-UGV.

5.1. Obtaining the Video Stream

The initial testing was done using two laptops as the client and server with the integrated camera of the client serving as the video sensor. VLC media player 2.0.0 Twoflower was used to obtain and display the video stream wirelessly.

The VLC Player was used both from the Command Prompt as well as its own GUI. At the server end the sout command was used to open the stream after the setting the directory to that containing the VLC Player.

At the client end 'vlc rtsp://ip-address-of-the-server:8080/test.sdp' was used to obtain the video stream from the streaming server.

First, the server was made to send a wireless stream on RTP/MPEG Transport Stream. A URL was set and a port defined. On the other PC, the VLC player was made to act as a client receiving a video stream from an IP address. This was done on LAN and WLAN as well as on the Wi-Fi Ad-hoc mode.

The problem with using the VLC Player however was a latency of almost 3s. This was improved to around 1.5s by changing the caching values for the network to 100 ms. However, even this delay was not acceptable for the NCE-UGV.

Later on, an IP camera was used to obtain the video stream. For this it was first connected via an Ethernet cable to a PC. On the PC, the IPv4 settings of the Local Area Connection were then set as given below.

IP address: 192.168.1.112

Default Gateway: 192.168.1.1

Subnet Mask: 255.255.255.0

Preferred DNS Server: 192.168.1.1

In the web browser search box, http://192.168.1.111:81 was typed. This brought up a dialog box prompting for username and password. Entering admin in both these fields brought up a window with video displayed. From here the wireless settings were configured according to the router/connection used and saved.

The camera was then disconnected from the LAN and allowed to wirelessly connect to the router (on which it was configured). The static IP address is then typed in the browser window of the server (which is connected to the same router as the camera).
This made a new window pop up displaying the video stream in the inbuilt-GUI of the IP camera. This in addition to the stream, contained several functions to control and configure the camera. One that was extensively used was the Pan/Tilt capability that provided a wide scope of view. Additionally, the resolution of the camera could also be set to the required pixel quality.

5.2. Transmission of Data Using TCP Link

The transmission of control information via Wi-Fi link was tested between two laptops.

For this purpose socket programming was used whose libraries were already installed in the Linux environment and the GCC Compiler was used for compilation of the codes.

On the Windows Server, the Java Runtime Environment was first installed. This enabled the compilation of the Java codes without any JDK (Java Development Kit).

The Wi-Fi link was established via an ad-hoc connection between the two computers and later on WLAN. A constant establishment of the connection was ensured through regular ACKS.

Once the ARM11 board had LINUX installed and Wi-Fi working, the client side C codes were run there along with the server application. The latter sent control commands to the board which output characters accordingly onto its serial port that controlled the motors. These then rotated in accordance with the direction defined.

To get the Wi-Fi working the 'ifconfig wlan0 down' command was first typed followed by 'iwconfig wlan0 channel 1 essid AQSA mode ad-hoc'. The SSID was named AQSA whereas the mode was Ad-hoc.

The 'ifconfig wlan0 192.168.1.10 netmask 255.255.255.0 up' was then used to give the client a static IP of 192.168.1.10, whereas the server was given 192.168.1.15. Each was the default gateway of the other with subnet mask as 255.255.255.0. No DNS servers were specified in the Ad-hoc mode.

This chapter explained the various stages of testing the NCE step by step. The video stream was obtained via various ways finally choosing the use of an IP camera for reasons already mentioned. The data transmitted via TCP Link however had only one approach i.e. that of configuring the ARM11 board for Wi-Fi connectivity and making it work as a client. Additionally, it involved Linux installation, serial port configuration and compilation of C codes.

CHAPTER 6
FUTURE ENHANCEMENTS AND CONCLUSION

This chapter suggests several enhancements that can be done in the project to make it more task-specific as well as to overcome the present limitations.

6.1. Future Enhancements

The present shape of the project can be enhanced in various ways to make it practically more usable for different purposes. For this several recommendations are given which can be explored further to make the NCE-UGV more task-specific.

6.1.1. Using a Solar Panel

The UGV is presently being operated using dry batteries on-board. The power supply can further be improved by using a solar panel mounted on top of the UGV that would solve the problem of limited supply. Prior to this, the power consumption of the device would have to be calculated and the size of the panel selected accordingly.

The solar panel would basically be used to supply power to the rechargeable dry batteries. The former comes in various sizes providing different wattages suitable for a variety of purposes.

In addition to the solar panel, a charge controller would have to be used to control the voltage produced in case the sun is too bright. This is important so that the batteries are not damaged due to excessive voltage. It is inserted between the solar panel and the batteries and prevents overcharging.

6.1.2. Adding a GPS

A worthwhile extension would be to use a GPS to send location information along with the video stream. (ARM11 board comes with its own GPS module that uses the serial port to interface with it. Figure 6.1 shows the GPS module.)

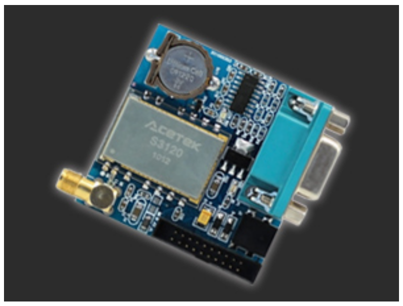

Figure 6.1 ARM11 GPS Module

The module has a very high tracking sensitivity (-159 dBm) and supports the NMEA 0183 protocol for data transmission.

This would help locate the UGV in addition to simple surveillance and control and would especially be helpful for longer distances. Simultaneously, the GUI could be extended to contain a map displaying the position of the UGV at its centre. This would involve further back-end and front-end programming.

6.1.3. Extending the application

The application can also be extended to include the video stream making it more compact. This can be done in various ways. It can be programmed to contain the browser window displaying the IP stream. On the contrary, the stream can first be obtained in a video player, for example the VLC Player, which can then be embedded in the application. Java bindings for the VLC Player are already available that enable the player to be easily embedded in any Java application. Other players such as the JMF and LIVE555 can also be used.

The GUI can be made more attractive by using buttons displaying arrow keys to control the car remotely. This would involve enabling the keyboard focus on a JPanel containing those buttons and connecting them programmatically to the arrow keys on the keyboard.

6.1.4. Inter-networking

Presently, the UGV is being controlled wirelessly on WLAN. This however, can be extended to two different networks connected via the internet. This involves the concept of Port Forwarding on the router as shown in Figure 6.2.

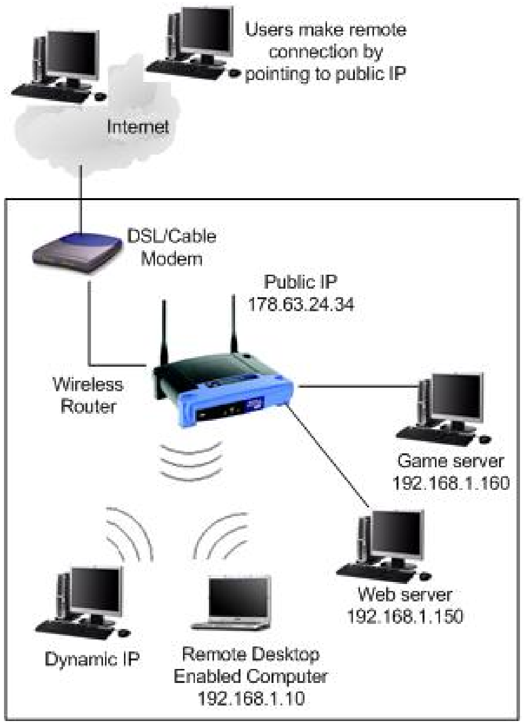
Users make remote connection by pointing to public IP
Internet
DSL/Cable Modem
Public IP 178.63.24.34
Wireless Router
Game server 192.168.1.160
Web server 192.168.1.150
Dynamic IP
Remote Desktop Enabled Computer 192.168.1.10

Figure 6.2 Port Forwarding

Port Forwarding is a method by which the port number specified in the WAN address is mapped onto the device's LAN address. The need for it arises because the network device being accessed is behind a router which could be serving multiple devices. The approachable address on the other hand is only the WAN address which is for the router alone. Hence, arises the need for a port number specific to the device (behind the router) which is actually being accessed!

6.1.5. Making it Task-Specific

For purposes like topographical and environmental data collection as well as mine detection, this project presents the basic structure. The UGV can be mounted with additional sensors (and interfacing circuits) that would make it more task-specific to the particular organization deploying it.

Defence and wildlife organizations can use the UGV in its present form for spying purposes and wildlife surveys respectively. On the other hand, environmental and topographical organizations would need to add additional circuitry and sensors for their respective data collection. Mine detection can also be done using the NCE-UGV with additional robotics hardware.

6.2. Conclusion

This project implements a feasible and economical approach for surveillance and control to achieve a number of ends for both civil and military purposes. Real time video surveillance is proposed that enables remote control of the UGV. The system represents a basic tool (with a lot of potential for task-specific enhancements) for various organizations belonging to the areas of defence, survey, topography, environment, wildlife etc. It promises a practical solution for use at locations where human presence is not feasible.

Appendix A-1

Main Java Code for Server Application

```
package server;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.*;
import javax.swing.*;

import java.net.ServerSocket;
import java.io.DataOutputStream;
import java.net.Socket;

public class KeyEventDemo extends JFrame
        implements KeyListener,
        ActionListener
{

    JTextField typingArea;
    public static String s1;
    public static String keyString;
    public static DataOutputStream outToClient;


    public static void main(String[] args) {
        /* Use an appropriate Look and Feel */
        try {

            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
        } catch (UnsupportedLookAndFeelException ex) {
            ex.printStackTrace();
        } catch (IllegalAccessException ex) {
            ex.printStackTrace();
        } catch (InstantiationException ex) {
            ex.printStackTrace();
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        }
        /* Turn off metal's use of bold fonts */
        UIManager.put("swing.boldMetal", Boolean.FALSE);
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });


         try
                              {
                   ServerSocket serverSocket = new ServerSocket(12000);
                   Socket client = serverSocket.accept();
                   System.out.println("Connection made at port 12000");


                   while (true)
                   {

                   outToClient = new DataOutputStream(client.getOutputStream());

                   }


                    }

catch(Exception exc)
                       {
                                  exc.printStackTrace();
                       }

      }

    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {
        //Create and set up the window.
        KeyEventDemo frame = new KeyEventDemo("KeyEvent");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Set up the content pane.
        frame.addComponentsToPane();
        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    private void addComponentsToPane() {

        JButton button = new JButton("Clear");
        button.addActionListener(this);

        typingArea = new JTextField(20);
        typingArea.addKeyListener(this);

        getContentPane().add(typingArea, BorderLayout.PAGE_START);
        getContentPane().add(button, BorderLayout.PAGE_END);
    }

    public KeyEventDemo(String name) {
        super(name);
    }


    /** Handle the key typed event from the text field. */
    public void keyTyped(KeyEvent e) {
        displayInfo(e, "KEY TYPED: ");
    }


    /** Handle the key pressed event from the text field. */
    public void keyPressed(KeyEvent e) {
        displayInfo(e, "KEY PRESSED: ");
    }

    /** Handle the key released event from the text field. */
    public void keyReleased(KeyEvent e) {
        displayInfo(e, "KEY RELEASED: ");
    }

    /** Handle the button click. */
    public void actionPerformed(ActionEvent e) {
        //Clear the text components.
        typingArea.setText("");

        //Return the focus to the typing area.
        typingArea.requestFocusInWindow();
    }


    private void displayInfo(KeyEvent e, String keyStatus){


        int code;
        code = e.getKeyCode();


                    if (code == KeyEvent.VK_A)
                    {s1= "a";}
                    if (code == KeyEvent.VK_W)
                    {s1= "w";}
                    if (code == KeyEvent.VK_S)
                    {s1= "s";}
                    if (code == KeyEvent.VK_D)
                    {s1= "d";}

                    System.out.print(s1);

                            try
                            {

                                outToClient.writeUTF(s1);
                                 System.out.println("Sending stream");

                            }

                            catch (Exception exc)
                            {

                                exc.printStackTrace();

                            }



    }
}
```

Appendix A-2

C Code (in AT89C51) for Driving the Motors

```
#include <reg51.h>


sbit EN1=P2^0;

sbit EN2=P2^1;

sbit IN1=P2^2;

sbit IN2=P2^3;

sbit IN3=P2^4;

sbit IN4=P2^5;



void ini()     // Initialize Timer 1 for serial communication

{

TMOD=0x20;  //Timer1, mode 2, baud rate 9600 bps

TH1=0XFD;

SCON=0x50;

TR1=1;

}



void serial_int (void) //Function to receive serial data

{

unsigned char chr;

while(RI==0);
```

```
chr = SBUF;
P1=chr;
RI=0;
}


void forward()
{
EN1=1; EN2=1;
IN1=0;IN2=1;IN3=1;IN4=0;
delay_1s();
}


void left()
{
EN2=0; EN1=1;
IN1=0;IN2=1;
IN3=0; IN4=0;
delay_1s();
}


void right()
{
EN1=0; EN2=1;
IN3=1;IN4=0;
```

```
IN1=0; IN2=0;
delay_1s();
}


void reverse()
{
EN1=1; EN2=1;
IN1=1;IN2=0;IN3=0;IN4=1;
delay_1s();
}


void stop()
{
EN1=0; EN2=0;
IN1=0;IN2=0;IN3=0;IN4=0;
}


delay_1s()        // timer of 1 sec
{
int d;
for(d=0;d<=20;d++)
{
TMOD=0x01;
TL0=0xFD;
```

```
TH0=0x04B;
TR0=1;          // start timer.
while(TF0==0);          // run until TF turns to 1
TR0=0;          // stop timer
TF0=0;          // reset the flag

}
}

void pwm() //50ms
{
TMOD=0x02;
TH1=0x4C;
TL1=0x00;
TR1=1;
while(TF1==0);
TR1=0;
TF1=0;
}


void main()
{
while(1)
{

        ini();
```

```
            serial_int();

if (P1=='w')
{
pwm();
forward();
stop();
 }

else if (P1=='a')
{
pwm();
left();
stop();
 }

else if (P1=='d')
{
pwm();
right();
stop();
 }

else if (P1=='s')
{
pwm();
reverse();
stop();
 }

else
stop();

}
}
```

Appendix A-3

C Code for Client End Socket Programming and Writing on Serial Port

```
#include <fcntl.h>
#include <termios.h>
 #include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;


    char buffer[8];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
while(1)
{
    bzero(buffer,8);
    n = read(sockfd,buffer,8);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);

int fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY );
if(fd < 0)
 {
    perror("Could not open device");
 }
printf("Device opened\n");


struct termios options;
tcgetattr(fd, &options);
cfmakeraw(&options);
cfsetispeed(&options, B9600);
cfsetospeed(&options, B9600);
options.c_cflag &= ~CSIZE; /* Mask the character size bits */
options.c_cflag |= CS8;    /* Select 8 data bits */
options.c_cflag &= ~PARENB;
options.c_cflag &= ~CSTOPB;
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;
options.c_iflag &= ~(IXON | IXOFF | IXANY);  //software

tcsetattr(fd, TCSANOW, &options);
```

```
char txpacket[] = {'A', 'B', 'C', 'D', 'E', 'F'};
ssize_t written = write(fd, buffer, sizeof(buffer));
printf("Written %d bytes\n", written);


}
close(sockfd);
return 0;
}
```

BIBLIOGRAPHY

[1] “Streaming media,” *http://en.wikipedia.org/wiki/Streaming_media*

[2] “Java SE Desktop Technologies,” *http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html*

[3] “MAX232,” *en.wikipedia.org/wiki/MAX232*

[4] “ARM9 Board,” *http://www.arm9board.net/sel/prddetail.aspx?id=348&pid=200*

[5] Michael R. Sweet, “Serial Programming Guide for POSIX Operating Systems,” *http://www.easysw.com/~mike/serial/serial.html*

[6] “C51 Development Tools,” *www.keil.com/c51*

[7] “AT89C51 Microcontroller,” *http://www.engineersgarage.com/electronic-components/at89c51-microcontroller-datasheet*