

Project Number: CS-MXC-0304

# **Neural Impulse Robotic Control**

A Major Qualifying Project Report  
Submitted to the Faculty  
of the

**WORCESTER POLYTECHNIC INSTITUTE**

in partial fulfillment of the requirements for the  
Degree of Bachelor of Science  
in **Computer Science**

by

**Edwin Lui**

Date: 01/12/10

Advisors

**Prof. Michael J. Ciaraldi, Major Advisor**

**Prof. Richard D Beach, Co-Advisor**

## **Abstract**

Neural impulse robotic control refers to controlling a robot with one's mind. In this day and age, everything we do is facilitated by the aids of computer and robots, vacuum cleaners, electronic curtains, etc. What if there is a robot that could be controlled by one's mind? By combining NIA (Neural Impulse Actuator), VEX robot, and with several programs, this may well be possible.

## Table of Contents

Abstract .....	i
Table of Contents .....	ii
Table of Figures .....	iii
1. Introduction .....	1
2. Background .....	2
2.1 NIA .....	2
2.2 VEX .....	5
2.3 Laptops .....	6
3. Design .....	7
3.1 Client and Server .....	8
3.2 VEX C program .....	10
3.3 Robot .....	10
4. Results .....	11
5. Future work .....	12
Appendix A – Bibliography .....	13
Appendix B – Source Code .....	14
Client code .....	14
Server code .....	18
Vex robot code .....	22

# Table of Figures

- Figure 1 NIA headband and black box ..... 3
- Figure 2 Calibration Screen ..... 4
- Figure 3 Choosing Profiles..... 4
- Figure 4 Pong Game ..... 5
- Figure 5 Creating a Profile..... 5
- Figure 6 Brainfingers ..... 5
- Figure 7 Setting Events for each input ..... 5
- Figure 8 Data Flow Diagram 1 ..... 7
- Figure 9 Data Flow Diagram 2 ..... 7
- Figure 10 Client Program ..... 8
- Figure 11 Robot side view ..... 10
- Figure 12 Robot front view ..... 10
- Figure 13 Robot bottom view ..... 10

## 1. Introduction

In this day and age, most of the things we do are facilitated by all types of gadgets. For example vacuum cleaners instead of brooms, computers instead of type-writers, and so on. Not only does this facilitate and make our lives more efficient, the advances in technology also allow us to be lazier. For example, robots that were created to fetch a can of beer from the refrigerator or robots that will automatically vacuum the whole house, etc. However, these robots are either preprogrammed to automatically carry out a task, or have to be controlled remotely, usually by a joystick. What if one day you could control your robot without lifting a finger? With all the technology available nowadays, this may well be possible.

In this MQP (Major Qualifying Project), I utilized both the NIA (Neural Impulse Actuator) and the VEX robot kit to build a robot that can theoretically be controlled by the mind. Because this is a Computer Science MQP, I focused mainly on the software side of this project. In order to have NIA and the VEX robot to act as a team, software modules at both ends are required, which were the main focuses of my project.

In the next chapter, a more detailed description on what I am trying to achieve for the project will be presented, and I will elaborate further on the technical background behind this project. In the Design chapter, the designs for the robot, both software and hardware will be laid out in detail. Finally, I will discuss the final product; its strengths and shortcomings, and future considerations for improvement will also be discussed.

## 2. Background

There are four main components to this project. The first component is the NIA (Neural Impulse Actuator), the second one is the VEX robot kit, and lastly two laptops, one at the NIA end and the other at the VEX end.

### 2.1 NIA

NIA is a Brain-computer interface device developed by OCZ Technology. A Brain-computer interface device is similar to a pointing device, e.g. a mouse, where both are input devices for the computer, where one receives input from the action of clicking and moving, and the other captures input from the brain. In terms of signals captured from the brain, it actually includes a mixture of signals from the muscles, skin, and nerve activity, which will be explained in further detail later on.

In the beginning stages of the project, before deciding on using NIA for this project, I have researched on a few brain-computer devices that are available in the market. They are NIA, Emotiv's Epoc Headset, and Neurosky's Mindset. All three of these brain-computer devices are designed for the gaming population that uses standard Windows PC computers, and no extra hardware or special computers are required. A simple "plug and play" will get the device going. The NIA has three electrodes to receive signals from the brain, Emotiv's Epoc Headset has 14 electrodes, and the Neurosky's Mindset has only one electrode. With more electrodes, it allows the device to capture more varieties of brainwaves and signals with higher accuracy. Emotiv's Epoc Headset seems to be the most powerful one out of all three devices, but at the same time it is also the most expensive one, the market price of it is around \$299 US dollars. Neurosky's Mindset is the least powerful one, and the market price of it is \$199. Out of all three devices, NIA has the lowest price, with a market price of \$149, and its capability is sufficient for this project. Therefore after some considerations, I decided to use NIA for this project, since it is the most affordable choice and is well enough to fulfill the objective of this project.

Brain-computer devices	Price (US Dollar)	Electrodes
NIA	\$149	3

<b>Emotiv's Epoc Headset</b>	\$299	14
<b>Neurosky's Mindset</b>	\$199	1

Table 1 Brain-computer devices comparison

When the NIA is first opened from the box, it consists of three components. The first thing is the NIA headband that captures the brain signal. The second is a small black device that connects the NIA to the computer via USB. The last is the CD Rom that contains all the required software that comes along with NIA.

The headband is made of carbon nanofibers with three electrode sensors attached to it. The sensors pick up three basic types of signal from the brain and forehead; the neuronal discharges in the brain, the electro-oculogram components, and the electro-myograms. The neuronal discharges are the alpha, beta and gamma brain waves. The electro-oculogram components are the positional differential between the front of retina and the retinal pigmented epithelium which changes relative to the eye orientation. The electro-myograms are the neuro-muscular signals along with the electrical discharges resulting from the depolarization of the muscle cells. In other words, the headband takes in signals from the brain, the muscle on the forehead, and orientation of the eyes.

The black box connects the headband to the computer via USB. It processes all the signals captured from the headband and translate it into a desired keystroke on the keyboard through the NIA software.



Figure 1 NIA headband and black box

The NIA software displays all the signals captured and processed from the headband and the black box. Every time the user starts up the software, the software requires the user to calibrate the headband for better reception of signals (Figure 2). When calibrated successfully, the user can either choose a profile that is pre-made by NIA, or create their own profile (Figure 3). A profile is a set list of preset key commands corresponding to each type of signal. The software displays the signals as “Brainfingers” (Figure 6) when creating a profile, so the user can clearly see what signal is currently being captured. The “Brainfingers” are named Glance, Alpha1, Alpha2, Alpha3, Beta1, Beta2, Beta3, and Muscle. These are the 8 basic inputs for NIA where each input can be divided into 4 different key commands at most. Which means a total of 32 key commands can be set at most. A keystroke is sent for each command when the ‘Brainfingers’ reach a certain level. Each command can be set to different events, ‘hold’, ‘single’, ‘dwell’, or any other combination of these three (Figure 7). When the ‘hold’ event is set, it displays the key command when the corresponding signal is detected as if a key is hold and pressed down on the keyboard. When the ‘single’ even is set, it displays the key command as a single hit of the key on the keyboard. When the ‘dwell’ event is set, it displays the key command as if the same key is pressed multiple times. There is also a Pong game (Figure 4) for users to practice using the NIA, since it requires lots of training for a user to use the NIA very effortlessly.

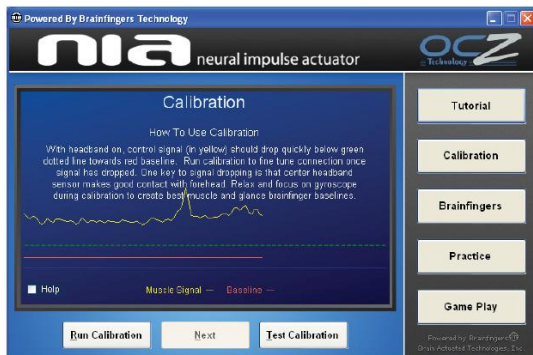


Figure 2 Calibration Screen

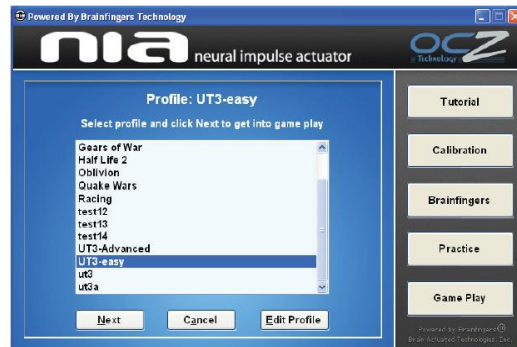


Figure 3 Choosing Profiles



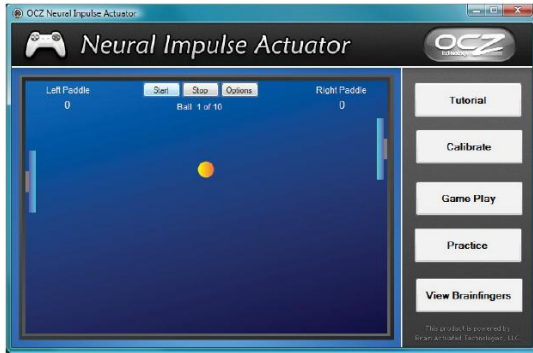


Figure 4 Pong Game

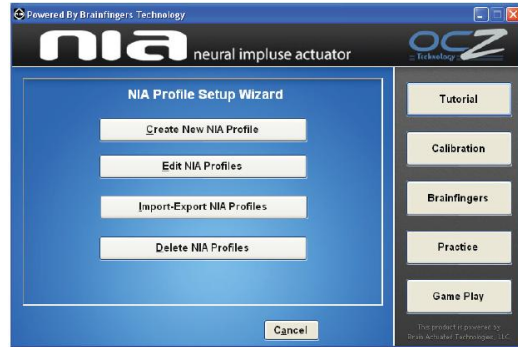


Figure 5 Creating a Profile

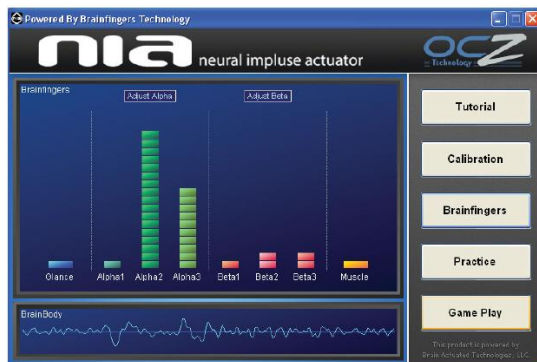


Figure 6 Brainfingers

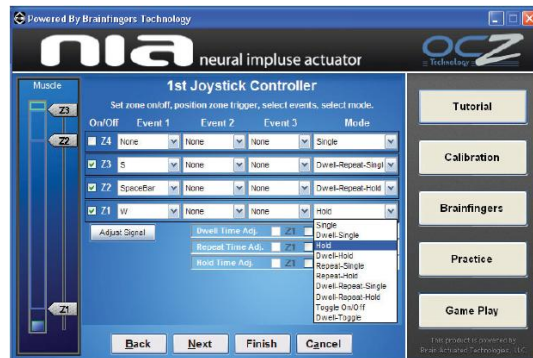


Figure 7 Setting Events for each input

The NIA was mainly designed as a gaming device. It is not a device that is designed to replace the mouse, but to use it in conjunction. It is said that the reaction time could be cut by 50%. Apart from its gaming uses, there has also been other research conducted with NIA. One example of this is someone using NIA in conjunction with the Wii Balance Board to choose, zoom, and view pictures on a self written picture viewing application<sup>1</sup>. Another person has done research using NIA to control the IRobot Roomba by observing the behavior of the robot according to different threshold levels of emotions<sup>2</sup>.

## 2.2 VEX

VEX Robotics Design System is an award-winning platform that intends to introduce students to the world of robotics. As a student in WPI, I am fortunate to have access to most of the VEX parts. Usually a robot that is built with VEX will contain a VEX microcontroller, which functions as the 'Brain' of the robot. The microcontroller's job is to communicate with all other parts of

<sup>1</sup> Reference 2 - NIA with Wii Balance Board - [www.youtube.com/watch?v=ap\\_77mcyqfc](http://www.youtube.com/watch?v=ap_77mcyqfc)

<sup>2</sup> Reference 3 - Research with NIA using bio-electrical signals to influence the social behaviors of domesticated robots (Saulnier, Sharlin & Greenberg, 2009)

the robot, for example the motors and bumper switches. The VEX microcontroller is compatible with C programming language. Therefore usually, a C program that controls all the procedures and signals for each part (e.g. motor) has to be written, and downloaded onto the VEX microcontroller from a computer by using a Serial-to-USB cable. In addition to sending signals to each part of the robot, the VEX microcontroller is also able to receive data through the serial port, which allows the robot to receive commands and act accordingly. Typically, the VEX robot is controlled by a remote control, but because the objective of this project is to have the brain to control the robot, that is where the laptops come in.

### **2.3 Laptops**

As I mentioned earlier, two laptops, one at NIA end and one at VEX end is required. The laptop at VEX end acts as a signal receiver, and contains the Server codes. The Server code is responsible for communicating with the laptop at the NIA end via the wireless network. This enables commands sent from the NIA to be redirected to the VEX microcontroller. Since all laptops nowadays have several USB ports built into them, by using a Serial-to-USB cable, it allows communications going back and forth between the laptop and the VEX microcontroller.

The laptop at NIA end is the signal initiator and contains the Client code. The Client code is responsible for locating the server, in this case the laptop at the VEX end, and sending a key command to it. As previously mentioned, the NIA comes with software which is also installed here.

### 3. Design

With all four components described from above combined, hypothetically a mind-controlled robot could be generated. A person should be able to send a key command from the laptop at the NIA end with NIA worn on the head, as described in Figure 8. The remote laptop that is on the VEX end will receive the key command and forward it to the VEX microcontroller. When the VEX microcontroller receives the key command, it then processes it with the C program that was downloaded from before and performs the corresponding action instructed by the program, as described in Figure 9.

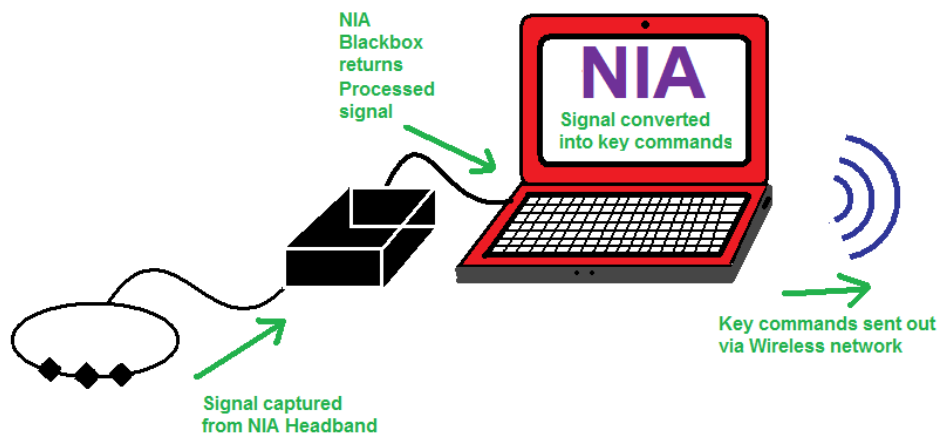


Figure 8 Data Flow Diagram 1

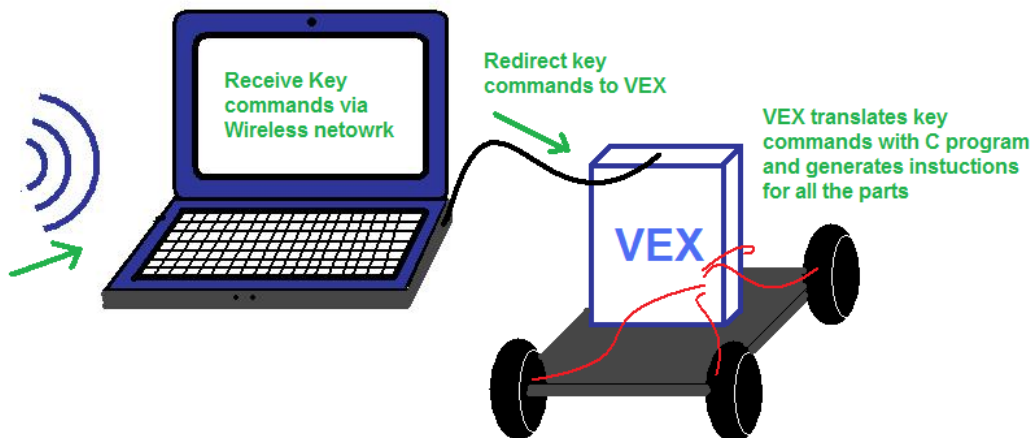


Figure 9 Data Flow Diagram 2

With NIA already preprogrammed by manufacturer, the software left for me to program was a Client on the laptop at the NIA end, a Server on the laptop at the VEX end, and a program on the VEX microcontroller. Last but not least is a simple robot that is operated by motors and can move in different directions.

I will briefly explain how each program works in this section, for further details, please find all software codes in Appendix B.

### 3.1 Client and Server

As previously mentioned, the Client is responsible for locating the Server, where in this case is the laptop at the VEX end, and sends a key command to it via the Wireless network from the laptop at the NIA end. The Server is responsible for receiving the key commands, and redirects it to the VEX microcontroller.

I have programmed both the Client and Server with Java, since Java is relatively simpler in Socket programming, which is programming software that involves network connections.

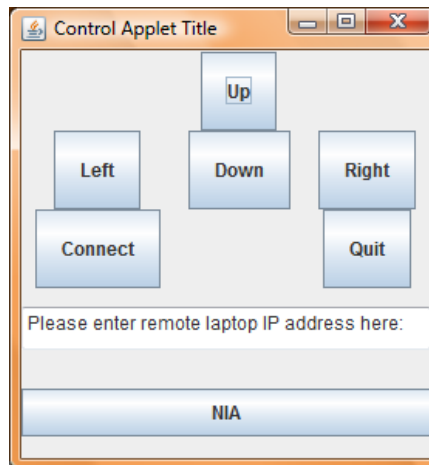


Figure 10 Client Program

The Client is represented with a simple graphic user interface (Figure 10). There are four buttons that are labeled 'Up', 'Right', 'Down', 'Left', a field that takes in the IP address of the remote computer, and three more buttons that are labeled 'Connect', 'Quit', and 'NIA'. The four directional buttons are created for testing purpose, so that when NIA is not plugged in, the robot can still be controlled with a mouse by clicking the buttons. The 'NIA' button acts as a key

listener, when it is clicked, it then starts accepting input from the keyboard. The 'Connect' button establishes the connection with the given IP address, and the 'Quit' button quits the program. Inside the Client code there is a list of classes, and each of the classes is responsible for different jobs. The function of each class is briefly explained in Table 2.

Class	Function of class
<b>Connect(String)</b>	To establish a connection with a given IP address
<b>sendKey(char)</b>	Sends a key command to the connected computer
<b>initLayout()</b>	Initializes the layout of applet
<b>init()</b>	Calls initLayout() and sets background color for applet
<b>start()</b>	Automatically runs when applet starts
<b>stop()</b>	Automatically runs when applet stops
<b>destroy()</b>	Automatically runs when applet stops
<b>clickLeft()</b>	Sets the key command for the 'Left' direction
<b>clickRight()</b>	Sets the key command for the 'Right' direction
<b>clickUp()</b>	Sets the key command for the 'Up' direction
<b>clickDown()</b>	Sets the key command for the 'Down' direction
<b>actionPerformed(ActionEvent)</b>	Detects which button inside applet has been clicked
<b>keyReleased(KeyEvent)</b>	Set instruction after a key is released from the keyboard
<b>keyPressed(KeyEvent)</b>	Set instruction after a key is pressed on the keyboard
<b>keyTyped(KeyEvent)</b>	Set instruction after a key is typed on the keyboard
<b>ControlApplet()</b>	Sets window size for applet and calls init()

Table 2 Client Classes

There is not any graphical interface for the Server, it is a simple executable JAR file. When it is executed, it just invokes the main() method from the Server class. Right in the beginning of execution, the Server will start listening at a specified port of the socket. When a request is detected (which in this case is a request from the Client), it will accept the request and hence establish the connection between the Client and the Server. Once the connection is established, any incoming key commands that were sent from the Client will get redirected to the C program running on the VEX. The redirection of key commands is done by sending the key commands to a specified USB port with the RXTX package (see Appendix B), in which this USB port is where the VEX microcontroller is plugged into with the Serial-to-USB cable.

### 3.2 VEX C program

The C program on the VEX microcontroller is responsible for controlling all the motors and switches connected to the microcontroller. When the VEX microcontroller is switched on, the C program that was downloaded onto the microcontroller from before will execute automatically. When the program starts, it goes into an infinite loop of reading the input from the serial port of the microcontroller and check whether any of those input matches with the preset values. If the input from the serial port matches one of the preset values, the preset function for that corresponding value will execute (Table 3). For example, I have programmed the character 'w' to correspond with the function moveForward(), so if the character 'w' is read from the serial port, the moveForward() function will be called, and hence resulting the robot to move forward.

Character	Function
'w'	moveForward()
'a'	moveLeft()
's'	moveBackward()
'd'	moveRight()
't'	spin()

Table 3 Functions with corresponding key commands

### 3.3 Robot

I have built a fairly simple robot that is run by two motors with four wheels. As can be seen from last section, this robot can simply move forward and backward, turn left and right, and spin in its own position. There is also a camera on the robot, acting as an eye for the robot, and it sends the real time images captured from the camera to the NIA end laptop via radio waves.



Figure 11 Robot side view

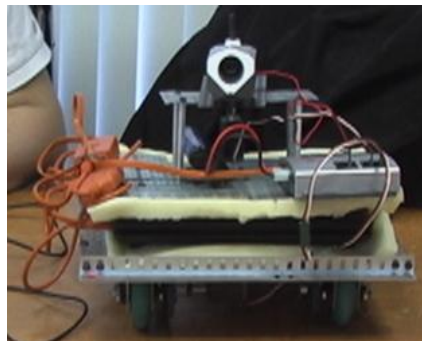


Figure 12 Robot front view

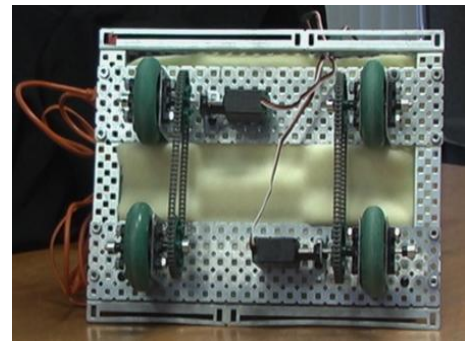


Figure 13 Robot bottom view

## 4. Results

After some trials and errors on all software and hardware, and with some final tweaking, this project has finally come to an end with partially working product.

I managed to have a robot that could be controlled by the mind to a certain extent. I say it is only to a certain extent because the robot can actually be controlled with the NIA headband, but currently it is only using the muscle sensors, therefore not the mind.

When all the equipments are set up accordingly (please refer to Appendix C), this is when both laptops are switched on and have both the Client and Server running and connected successfully, NIA is plugged in and calibrated, the VEX microcontroller is plugged in and switched on, and all the motors are connected to the right port on the VEX microcontroller. A person who has the NIA headband worn on the head should be able to use his or her forehead muscle to move the remote robot. The person should be able to direct the robot to spin in its own position when his or her forehead muscle is half tensed. And when the forehead muscle is fully tensed, the robot will travel towards the forward direction. The person who is controlling the robot should be able to see what the camera on the robot is seeing through the laptop in front of him, and steer it correspondingly.

In the final presentation for my project to both my advisors, it took me few runs to finally have the robot working smoothly for once. I was able to control the robot with my forehead muscle, and was able to direct it to go out of the room that I was in, and continue steering it even when it is outside of the room by looking at the real time image captured from the camera on the robot. One major problem that I encountered while setting up the robot was the trouble of getting wireless connection signals. Although wireless internet is already popularly used everywhere, it is still not to the point that everything works perfectly fine yet. Even when one of my laptops is connected to the internet already, somehow my other one will not be able to connect.

## 5. Future work

Based on what I accomplished with NIA, I can see a very large potential with neural impulse robotic control. Although as I mentioned in my results, I have only built a robot that could be controlled with one's forehead muscle. This is acceptable, as this is mainly due to the difficulties with NIA. NIA requires so much practice, in which I did not have time for. Based on many articles and researches, NIA requires an average of two months time to use it fluently, and much more time in order to master it and use its full potential. If one can master NIA and with its full potentiality exposed, a total of 32 key commands can be set. So far I have only used two key commands with the forehead muscle sensors. With 32 key commands means the robot can have 32 different functionalities. Therefore if I had the time and spent more time practicing using NIA, or at least able to control the glancing sensors, or even better the alpha and beta waves, the robot could have much more functionality to it. For example, the glancing sensors can be used to steer the robot, or control the direction of the camera. Also, if the brain wave sensors are controllable, a person can even have the robot move to a certain position or pick up a particular object just by thinking of a specific thing.

One possible future technology with neural impulse control can be a remote control for home furniture. For example, one can have a neural impulse device worn on the head, and just by thinking of switching off the lights, or closing the curtains, these can all be done without lifting a finger.



## Appendix A – Bibliography

"Alpha and beta brainwaves - OCZ Forum." *OCZ Forum Home*. Web. 13 Oct. 2009.

<<http://www.ocztechnologyforum.com/forum/showthread.php?t=40958>>.

"Assistive technology." *Wikipedia, the free encyclopedia*. Web. 12 Oct. 2009.

<[http://en.wikipedia.org/wiki/Assistive\\_technology](http://en.wikipedia.org/wiki/Assistive_technology)>.

*Brainfingers*. Web. 12 Oct. 2009. <<http://www.brainfingers.com/cyberlink.htm>>.

"Java sample code - Socket Applet - Source code examples." *Happy Codings - Java Programming Sample Code Source Code Search Engine, Code Examples*. Web. 11 Oct. 2009.

<[http://www.java.happycodings.com/Java\\_Applets/code13.html](http://www.java.happycodings.com/Java_Applets/code13.html)>.

"OCZ Neural Impulse Actuator The Future is nia : Profiles and Conclusions." *LostCircuits*. Web. 23 Oct. 2009.

<[http://www.lostcircuits.com/mambo//index.php?option=com\\_content&task=view&id=32&Itemid=1&limit=1&limitstart=5](http://www.lostcircuits.com/mambo//index.php?option=com_content&task=view&id=32&Itemid=1&limit=1&limitstart=5)>.

"OCZ NIA Brain-Computer Interface." *HotHardware.com - Technology Tested and Burned In*. Web. 12 Oct. 2009. <<http://hothardware.com/Articles/OCZ-NIA-BrainComputer-Interface/?page=4>>.

"OCZ NIA review - Neural Impulse Actuator." *Guru of 3D: PC Hardware Reviews*. Web. 22 Oct. 2009.

<<http://www.guru3d.com/article/ocz-nia--neural-impulse-actuator-review/5>>.

*Phidgets Inc. - Unique and Easy to Use USB Interfaces*. Web. 22 Oct. 2009.

<<http://www.phidgets.com/index.php>>.

"A Windows socket/Winsock2 TCP/IP network programming tutorials using C language with illustrations with working client-server C program examples and C code samples." *A programming tutorials on C language, C language, MFC Windows GUI, Winsock, STL, Win32 and GNU C Linux socket hands-on step-by-step with program examples, source code samples and screen snapshots*. Web. 25 Oct. 2009.

<<http://www.tenouk.com/cnwinsock2tutorials.html>>.

"Winsock Networking Tutorial." *MadWizard.org*. Web. 8 Oct. 2009.

<<http://www.madwizard.org/programming/tutorials/netcpp/>>.

## Appendix B – Source Code

### Client code

```
import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.*;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ControlApplet extends JFrame implements KeyListener, ActionListener {

    //Declare Variables
    JButton Left, Right, Up, Down, Quit, Connect, Nia;
    Label message;
    TextField ipfield;
    char direction;
    Socket robotSocket = null;
    PrintWriter out = null;
    BufferedReader in = null;
    TextField Output;

    //Establish a connection with the passed in ip address
    public boolean Connect(String ipaddress) throws IOException {
        boolean connected = false;
        try {
            robotSocket = new Socket(ipaddress, 4444);
            out = new PrintWriter(robotSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(robotSocket
                .getInputStream()));
            connected = true;
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host");
            //System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection");
            //System.exit(1);
        }
        System.out.println("connect ok!");
        return connected;
    }

    //Sends the passed in character
    public void sendKey(char c) throws IOException {
        out.println(c);
    }

    //Initialize layout
    public void initLayout() {
        setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;

        Left = new JButton("Left");
        c.weightx = 1;
        c.fill = GridBagConstraints.VERTICAL;
        c.gridx = 0;
        c.gridy = 2;
    }
}
```

```

add(Left, c);

Right = new JButton("Right");
c.weightx = 1;
c.gridx = 2;
c.gridy = 2;
add(Right, c);

Up = new JButton("Up");
c.weightx = 1;
c.weighty = 0.1;
c.gridx = 1;
c.gridy = 0;
add(Up, c);

Down = new JButton("Down");
c.weightx = 1.0;
c.gridx = 1;
c.gridy = 2;
add(Down, c);

Quit = new JButton("Quit");
c.weightx = 1.0;
c.gridx = 2;
c.gridy = 4;
add(Quit, c);

Connect = new JButton("Connect");
c.weightx = 1.0;
c.gridx = 0;
c.gridy = 4;
add(Connect, c);

ipfield = new TextField("Please enter remote laptop IP address here:");
c.weightx = 0.0;
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 3;
c.ipady = 5; //make this component tall
c.weightx = 0.0;
c.gridx = 0;
c.gridy = 6;
add(ipfield, c);

Nia = new JButton("NIA");
c.weightx = 1.0;
c.gridx = 0;
c.gridy = 8;
add(Nia, c);

Left.addActionListener(this);
Right.addActionListener(this);
Up.addActionListener(this);
Down.addActionListener(this);
Connect.addActionListener(this);
Quit.addActionListener(this);
}

public void init() {
    System.out.println("Applet begins...");
    initLayout();
    setBackground(Color.BLUE);
    Nia.addKeyListener(this);
}

```

```

}

public void start() {
    System.out.println("starting...");
}

public void stop() {
    try {
        out.close();
        in.close();
        robotSocket.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    System.out.println("stopping...");
}

public void destroy() {
    System.out.println("preparing to unload...");
}

public void clickLeft() {
    direction = 'a';
    System.out.println(direction);
}

public void clickRight() {
    direction = 'd';
    System.out.println(direction);
}

public void clickUp() {
    direction = 'w';
    System.out.println(direction);
}

public void clickDown() {
    direction = 's';
    System.out.println(direction);
}

//Check which button is clicked
public void actionPerformed(ActionEvent event) {
    if (event.getSource() == Left) {
        clickLeft();
    } else if (event.getSource() == Right) {
        clickRight();
    } else if (event.getSource() == Up) {
        clickUp();
    } else if (event.getSource() == Down) {
        clickDown();
    } else if (event.getSource() == Connect) {
        try {
            if (Connect(ipfield.getText()))
                Connect.disable();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    } else if (event.getSource() == Quit)
        System.exit(0);
}

```

```

        try {
            sendKey(direction);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void keyReleased(KeyEvent key) {
    }

    //Check when key is pressed
    public void keyPressed(KeyEvent key) {
        char c = key.getKeyChar();
        if (c == 'a')
            clickLeft();
        else if (c == 'd')
            clickRight();
        else if (c == 'w')
            clickUp();
        else if (c == 's')
            clickDown();
        else if (c == 'q')
            System.exit(0);
        try {
            sendKey(c);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void keyTyped(KeyEvent key) {
    }

    public ControlApplet() {
        super("Control Applet Title");
        setSize(280, 300);
        init();

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                setVisible(false);
                dispose();
                System.exit(0);
            }
        });
    }

    public static void main(String[] args) {
        ControlApplet app = new ControlApplet();
        app.setVisible(true);
    }
}

```

## Server code

### Server.java

```
import java.net.*;
import java.io.*;

public class Server {
    public static void main(String[] args) throws IOException {
        System.out.println("Listening...");
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(4444);
        } catch (IOException e) {
            System.err.println("Could not listen on port: 4444: " + e);
            System.exit(1);
        }
        Socket clientSocket = null;
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept failed.");
            System.exit(1);
        }
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
            new InputStreamReader(
                clientSocket.getInputStream()));
        String inputLine;
        vexController vc = new vexController();
        while ((inputLine = in.readLine()) != null) {
            vc.print(inputLine);
        }
        out.close();
    }
}
```

```
        in.close();  
        clientSocket.close();  
        serverSocket.close();  
    }  
}
```

## vexController.java

```
import gnu.io.*;
import java.io.*;

public class vexController {
    SerialPort port = null;
    OutputStream os= null;
    PrintStream ps = null;
    InputStream is= null;
    public vexController() {
        try {
            port = new RXTXPort("COM3");
            port.setSerialPortParams(115200, SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
            port.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
        } catch (PortInUseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (UnsupportedCommOperationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try {
            os = port.getOutputStream();
            ps = new PrintStream(os);
            is = port.getInputStream();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public void print(String s) {
```



```
        ps.print(s);
    }
    public void println(String s) {
        System.out.println("About to send to vex: " + s);
        ps.println(s);
    }
}
```

## Vex robot code

Robot\_main.c

```
#include "BuiltIns.h"
#include "robot_main.h"

void main(void) {
    unsigned int byte;
    Wait(2000);
    while (1) {
        byte = ReadSerialPortOne();
        switch (byte)
        {
            case 'd':
                turnRight();
                Wait(50);
                Stop();
                break;

            case 'a':
                turnLeft();
                Wait(50);
                Stop();
                break;

            case 'w':
                moveForward();
                Wait(50);
                Stop();
                break;

            case 's':
                moveBackward();
                Wait(50);
                Stop();
                break;

            case 't':
                spin();
                Wait(50);
                Stop();
                break;

            default:
                break;
        }
        PrintToScreen("byte: %d\r\n", byte);
    }
}
```

## moverFunc.c

```
#include "BuiltIns.h"
#include "robot_main.h"

//Make Robot moves forward
void moveForward()
{
    SetPWM(LEFT1,0);
    SetPWM(RIGHT1,255);
}

//Make Robot moves backward
void moveBackward()
{
    SetPWM(LEFT1,255);
    SetPWM(RIGHT1,0);
}

//Make Robot turns left
void turnLeft()
{
    SetPWM(LEFT1,127);
    SetPWM(RIGHT1,255);
}

//Make Robot turns right
void turnRight()
{
    SetPWM(LEFT1,0);
    SetPWM(RIGHT1,127);
}

//Make Robot spin
void spin()
{
    SetPWM(LEFT1,255);
    SetPWM(RIGHT1,255);
}

//Make Robot stops
void Stop()
{
    SetPWM(LEFT1,127);
    SetPWM(RIGHT1,127);
}

//Rotate Camera
void camFront()
{
    SetPWM(CAM, 0);
    Wait(2000);
}
```

## Robot\_main.h

```
#ifndef ROBOT_MAIN_H_
#define ROBOT_MAIN_H_

//Define Motors Ports
#define LEFT1 2
#define RIGHT1 1
#define CAM 6

//Define Functions Header
void moveForward();
void moveBackward();
void turnLeft();
void turnRight();
void spin();
void Stop();
void camFront();

#endif /*ROBOT_MAIN_H_*/
```