

ECE478 Project Report

Bohr Robot

12/12/2010

Arada, JC (aradaj@pdx.edu)

Le, Dang Zung(ddle@pdx.edu)

Chhokar, John (jchhokar1980@gmail.com)

Thueson, Mark (m_thueson@hotmail.com)

Contents

Problem Statement	3
The Hardware	3
Wiring diagram.....	5
Speech Recognition	6
The Software.....	6
Speech API	7
Developing Bluetooth application in robot C	9
Neils Bohr Arm Wheel Bluetooth Interface Spec	10
Implement Bluetooth in Visual Basic	12
Blue Tooth Protocol File Calls in VB	14
Blue Tooth Protocol Mailbox in VB.....	15
Visual Show Automation	16
You Tube Example.....	17
Bohr GUI	17
SVN	18
Future Plans ECE479	19

Problem Statement

To bring up the Bohr Robot from last year's working condition and add speech recognition and vision. We are missing a few key components from last year's robot configuration: (laptop, software, 12volt battery for motors). Our goal this term is to not only bring the robot up, but to implement speech recognition and vision. In addition, we wanted to add Bluetooth to the robot. Below is the original configuration of the robot from last year's group.

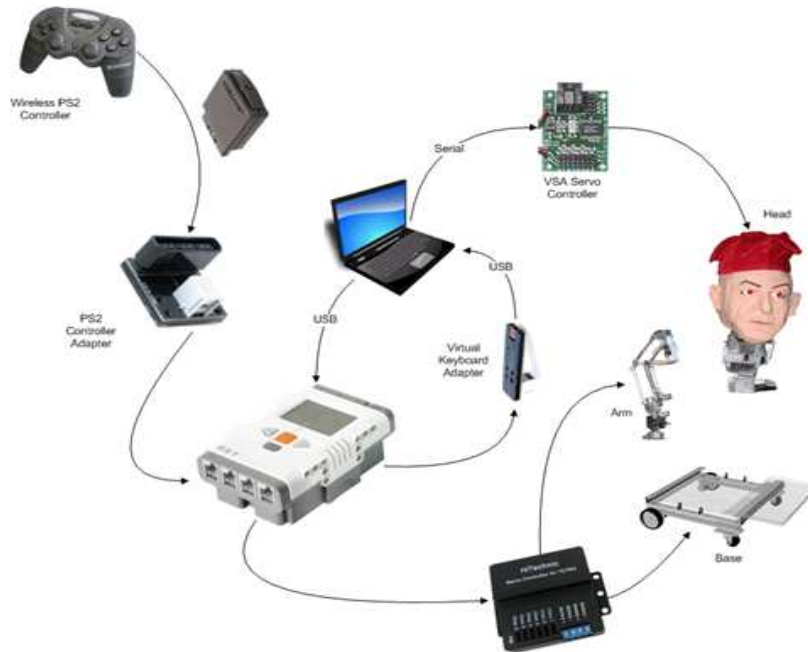


Figure 1: Original Robot Configuration

...will be modified as one of the following:

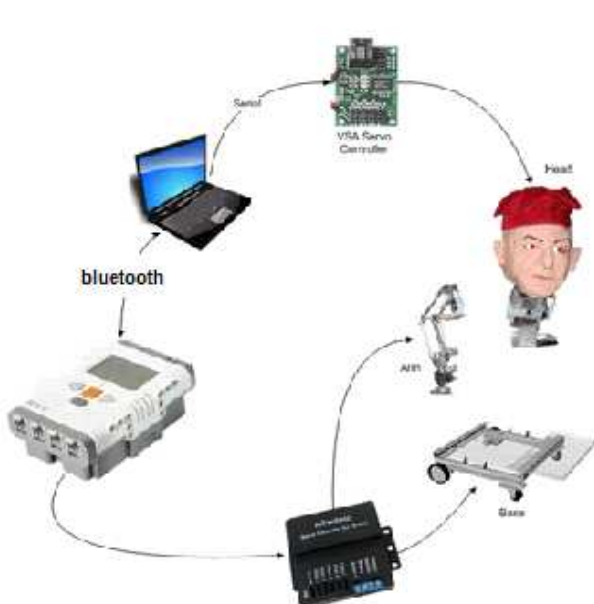


Figure 2: Proposed Re-Design I

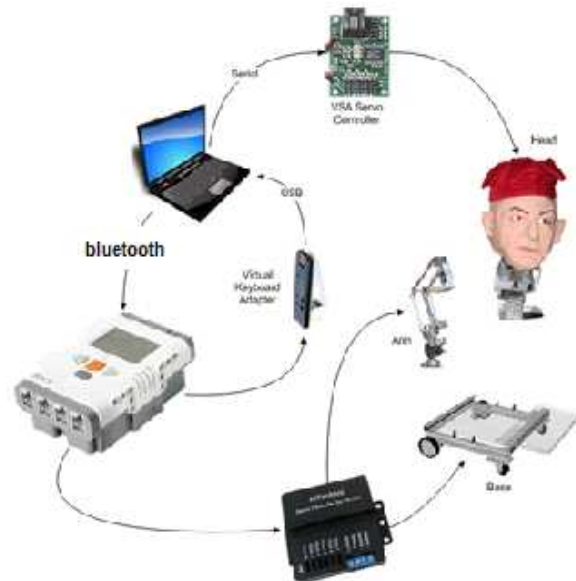


Figure 3: Proposed Re-Design II

To implement Bluetooth communication we purchased the following Bluetooth adapter for my computer. This Bluetooth adapter is made by IOGEAR and has a maximum wireless range of 30feet.



Figure 4: Bluetooth Adapter

To improve speech recognition I bought a pair of high quality noise cancelling headphones from Costco for \$89.00. I realized that this was a crucial part for the project and this headset gives me the ability to communicate through USB wirelessly with 3 feet of range.



Figure 5: High Quality Headset

We need a 12volt battery to power the motors on the base. The battery used by last year's group is missing. The 12 volt battery shown below was purchased for \$30.99.



Figure 6: 12 Volt DC Battery

Lastly, we purchased another NXT Brick for testing purposes. We did not want to erase the functions on the NXT written from the previous group. In order to continue to add to the software files on the NXT we would have to download new firmware and we might erase their code. The NXT brick is \$135.00.



Figure 7: NXT BRICK

Wiring diagram

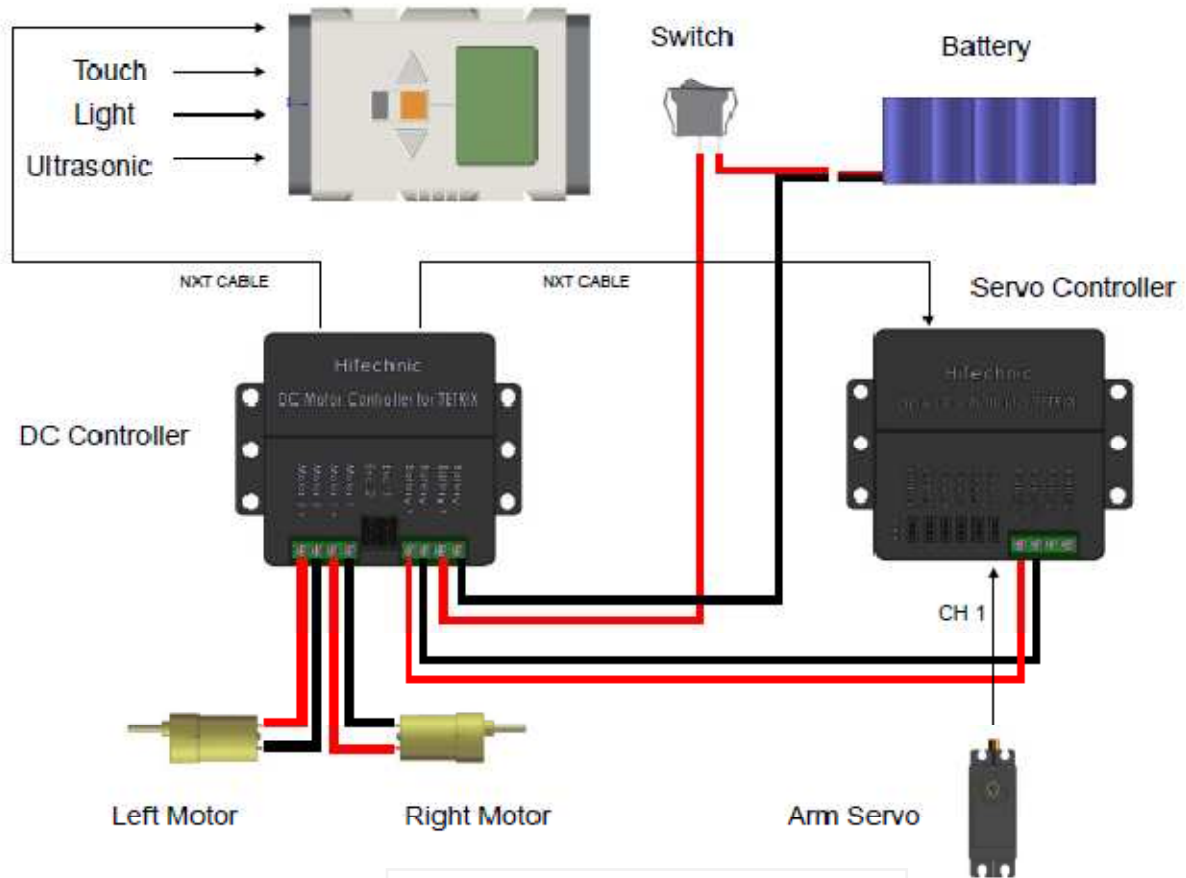


Figure 8: Wiring Diagram

Speech Recognition

Speech recognition is a complicated task because there are large sources of variability that are associated with voice commands. One type of variability is found in a person's voice. Acoustic variability's in ones voice can change the way the computer interprets phonemes. Phonemes are the smallest units of sound and represent 500,000 words in the English language. When the user speaks into the microphone, the user's voice is converted into digital bytes of information which are comprised of phonemes. The phonemes are then compared to a large library, and when a match occurs the word is then given back to the user. The delivery of speech into the microphone is very critical for this reason.

Another variability associated with voice commands is the user's environment. External noise associated to an environment can introduce all sorts of problems for the user. For example, a speech program can be fully functional in a conference room where there is no noise, but can be fully not functional when demonstrated in front of a room of noisy kids. For this reason, it's very important to have a high quality noise cancelling microphone.

Next, it's important that a speech recognition program is tested in all sorts of environments. This is another crucial step for the reasons stated in the previous paragraph. You can record the accuracy of each command in each environment and predict how successful your program will work when it is demonstrated.

Lastly, I will take two approaches to speak commands to the Bohr Robot. The first approach is to communicate to the Bohr Robot over a serial wired cable and give the robot commands over a wireless noise cancelling headset. Secondly, I would like to add Bluetooth to this robot and control the robot by first giving it a command into the headset and then transmitting that command over Bluetooth.

The Software

I am using Microsoft Visual Basic 2008 express edition. In order to include speech recognition into your project you have to include references to the Microsoft Speech Object Library (COM) and System.Speech (.Net). Since my computer is running Windows XP I had to download Microsoft's Software Development Kit 5.1(SDK 5.1) from Microsoft's website. Links to download the required software are provided below. All the software I downloaded was free.

Microsoft Software Development Kit 5.1

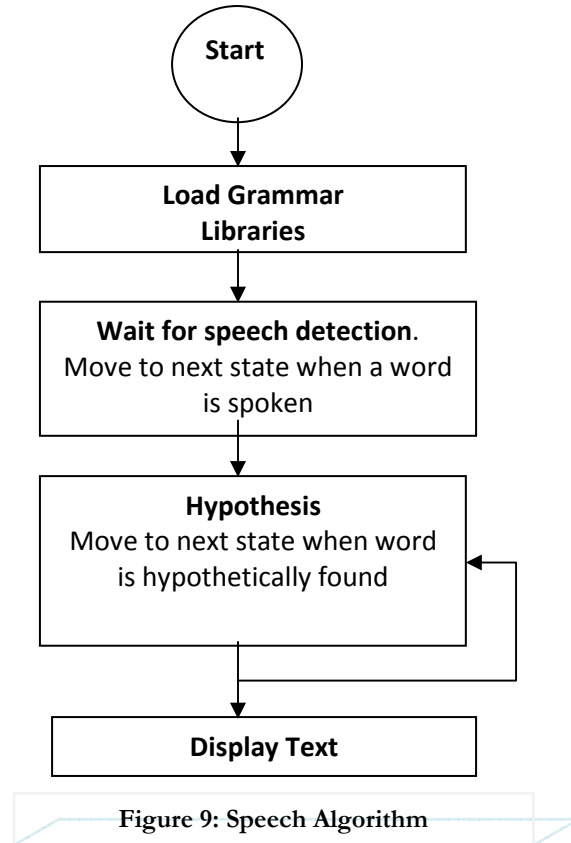
<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=5e86ec97-40a7-453f-b0ee-6583171b4530&displaylang=en>

Microsoft Visual Basic 2008 Express Edition

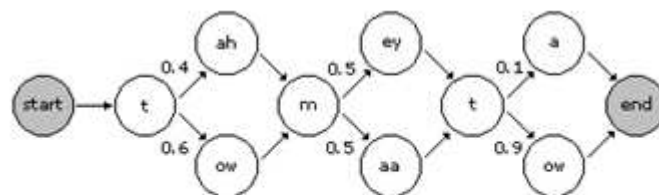
<http://www.microsoft.com/express/Downloads/>

Speech API

After two weeks of researching speech recognition I implemented a speech recognition algorithm in software. Using Microsoft's Speech Application Programming Interface (SAPI) a basic algorithm was developed.



When the program begins, the program waits for speech detection. When a word is spoken into the microphone, the word is broken into phonemes and then goes through Statistical Modeling to try and find the word spoken. An example of modeling used in software is called the Markov Model. An example of how a computer would interpret the word tomato is shown below. The word tomato is broken up into several phoneme's (T, ow, m, aa, t, ow). If you follow the phoneme through the model below you will see that tomato can be pronounced two different ways but at the last branch of the model there is a 90% chance that the word is tomato. That is the word that is then selected. This type of modeling enables a much quicker type of speech recognition when compared to a brute forced method.



An Introduction to Markov Models by James Matthews

Figure 10: Markov Models

There is one significant drawback to this model. Since this method try's to match the word spoken into the microphone to several words which may sound the same in the English language the results are not optimal and often times return the wrong word. For this reason I must look at creating my own language library instead of using Microsoft's library.

One way to improve the speech recognition on a system is to take the speech recognition training on your laptop. The following training is available in your control panel in the speech icon. The more training you take the more accurate your speech recognition becomes.

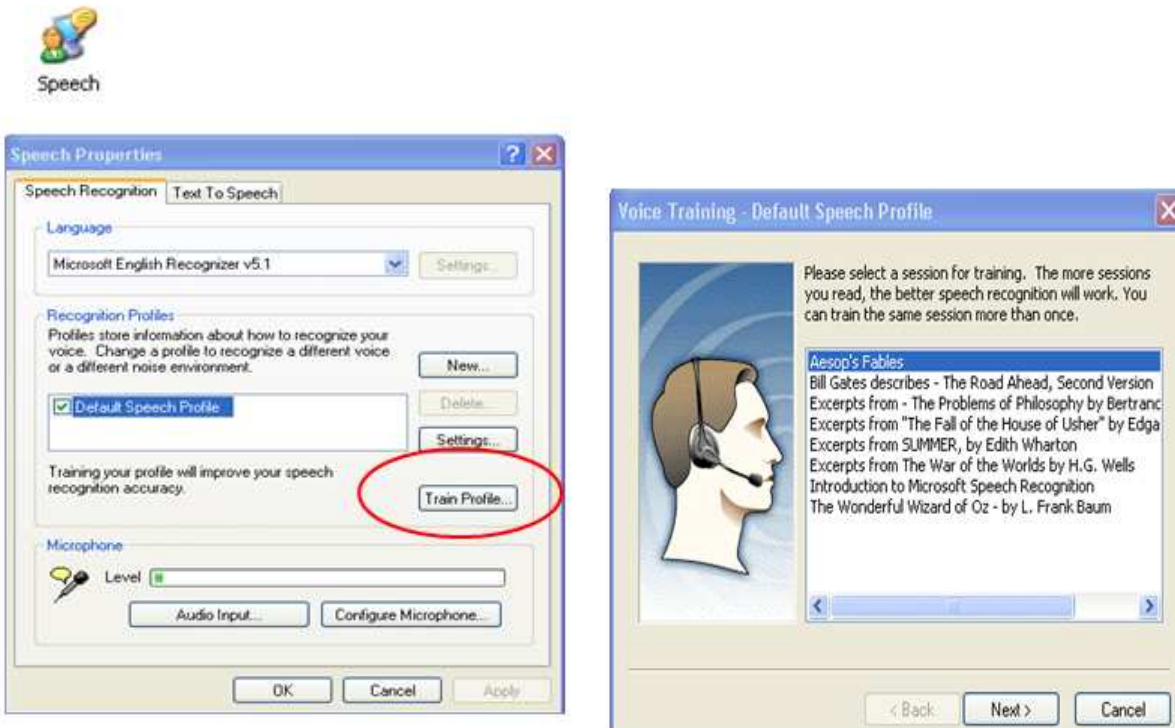


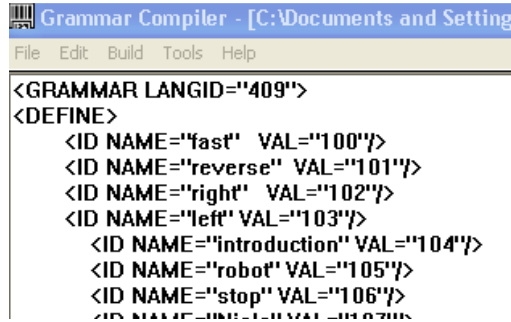
Figure 11: Speech Training

Speech Demo to Class

One possible approach to speech recognition was presented to the class. This demonstrated the background to speech recognition and how you can use voice commands to call functions. The demonstration also showed how difficult speech recognition is. To improve speech recognition we ordered a headset and implemented a custom grammar library using XML. We need to shoot for 95% accuracy of speech recognition.

To improve the speech recognition interface a custom grammar library was created instead of using Microsoft's grammar library. My initial thinking is that this would improve the program significantly because it would limit the amount of words the engine had to find. I had to learn XML syntax to create my own grammar library. After reviewing examples on XML I was able to successfully implement my own grammar library. Microsoft's SDK5.1 comes with a grammar library compiler test environment. Here you can write your grammar libraries, compile and run them. If your voice

commands are recognized here then you can conclude that the voice commands will be recognized in your program. An example of the grammar compiler is shown below.



```
<GRAMMAR LANGID="409">
<DEFINE>
  <ID NAME="fast" VAL="100"/>
  <ID NAME="reverse" VAL="101"/>
  <ID NAME="right" VAL="102"/>
  <ID NAME="left" VAL="103"/>
  <ID NAME="introduction" VAL="104"/>
  <ID NAME="robot" VAL="105"/>
  <ID NAME="stop" VAL="106"/>
  <ID NAME="..." VAL="...">
```

Figure 12: XML Grammar Library

Developing Bluetooth application in robot C

Send/Receive Messages from NXT brick – this part taken from RobotC manual, for more information see:

<http://www.robotc.net/support/nxt/MindstormsWebHelp/index.htm#page=Bluetooth/sendingmessages/Sending%20Messages%20via%20Bluetooth.htm>

The NXT firmware automatically receives messages and adds them to a queue of incoming messages. The application program takes the messages from this queue and processes them one at a time. The variables `message` and `messageParm` contain the contents of the current message being processed. The function `ClearMessage` discards the current message and sets up to process the next message.

Example can be found in 'BtBasicMsg.c'

Other simple methods:

`cCmdMessageGetSize` get the size of the first message in a mailbox containing a queue of received messages. `cCmdMessageRead` removes the first message from a mailbox queue and copies it to a user buffer. Implementation requires checking for messages periodically and read any incoming messages using `cCmdMessageRead`. Then interpretate this message and translate it into instructions.

Example can be found in 'NXT BT Messaging No Error Checking.c'

For Pc-side code, see implementation below in Visual Basic for details.

MESSAGEREAD

Byte 0: 0x00 or 0x80

Byte 1: 0x13

Byte 2: Remote Inbox number (0 – 9)

Byte 3: Local Inbox number (0 – 9)

Byte 4: Remove? (Boolean; TRUE (non-zero) value clears message from Remote Inbox)

Return package:

Byte 0: 0x02

Byte 1: 0x13

Byte 2: Status Byte

Byte 3: Local Inbox number (0 – 9)

Byte 4: Message size

Byte 5 - 63: Message data (padded)

MESSAGEWRITE

Byte 0: 0x00 or 0x80

Byte 1: 0x09

Byte 2: Inbox number (0 – 9)

Byte 3: Message size

Byte 4 - N: Message data, where N = Message size + 3

Message data is treated as a string; it must include null termination to be accepted. Accordingly, message size must include the null termination byte. Message size must be capped at 59 for all message packets to be legal on USB!

Return package:

Byte 0: 0x02

Byte 1: 0x09

Byte 2: Status Byte

Neils Bohr Arm Wheel Bluetooth Interface Spec

Both Arm and Wheels are completely controlled by the NXT block. Interfacing with the NXT requires 2 Bytes (Four HEX digits) sent to the NXT

- The first byte selects the motor
- The second byte determines the values passed to the motor

The first Byte:

Decimal	Hex	Function
1	01	Shoulder
2	02	Arm
3	03	Elbow
4	04	Wrist_ud
5	05	Wrist_lr
6	06	Hand
7	07	Left Wheel

8	08	Right Wheel
9	09	Forward
10	0A	Reverse
11	0B	Left
12	0C	Right
13	0D	RESERVED
14	0E	RESERVED
15	0F	RESERVED
16-255	10-FF	RESERVED

The second byte is a Signed binary with values given in percentages

- MSB (bit 7) determines forward or reverse
 - 0 is forward
 - 1 goes in reverse direction
- Valid values are 0 to 100 on bits 6 to 0

Servo expectations:

Shoulder – Positive is right, Negative is left
 Wrist_ud – Positive is up, Negative is down
 wrist_lr – Positive is right, Negative is left
 Hand – Positive is open, Negative is close

Servo transition expectations:

Shoulder – 2 seconds
 Wrist_ud – 1 seconds
 wrist_lr – 1 seconds

Example code:

```
// check size of message in mail box
nSizeOfMessage = cCmdMessageGetSize(0);
// read bluetooth mailbox and put into receive-buffer
nBTCmdRdErrorStatus = cCmdMessageRead(nRcvBuffer, nSizeOfMessage, 0); // inmail 0
ClearMessage();
int temp;
// decode Bluetooth commands and execute
switch(nRcvBuffer[0]) // decode the first byte
{
    case 0x09:
        nxtDisplayTextLine(3,"forward...");
        temp = nRcvBuffer[1]; // decode the second byte
        forward(temp); // forward at power level 25
```

```
// send response to PC if need
cCmdMessageWriteToBluetooth(nXmitBuffer, kMaxSizeOfMessage, 1); // outmail 1
break;
}
```

Direct command

The NXT supports direct commands to directly instruct the brick over Bluetooth such as start and stop a program loaded on NXT brick. This provides a simple way to control the brick from device other than NXT (PC/handheld...). See the implementation below in Visual Basic for details.

Also see Bluetooth development kit 'Direct command' for more information:

<http://mindstorms.lego.com/en-us/support/files/default.aspx>

Note: RobotC does not support sending direct commands (so that one NXT brick can control other NXT).

Implement Bluetooth in Visual Basic

We thought we could communicate to the Robot using our Bluetooth adaptor. The thinking was that we could give the robot simple commands such as forward, reverse, right and left using the headset and that command will be sent via Bluetooth. The previous group had implemented wireless communication using a PSP controller. We wanted to use Bluetooth because it gives you a chance to learn the Bluetooth protocol for communicating between the NXT controller and your laptop. In addition, we have plans to control the laptop remotely and implement vision.

Bluetooth connection was successfully established between my laptop and the NXT brick. To learn the protocol for Bluetooth and for calling sound files and software files, you have to reference Lego Mindstorms NXT direct commands found in Appendix 2. The link for the document is shown below.

<http://code.google.com/p/smart-robot/downloads/detail?name=Appendix%20-LEGO%20MINDSTORMS%20NXT%20Direct%20commands.pdf>

We were able call sound files and software files on the NXT brick. We implemented the following functions and drove the robot around the engineering lab using Bluetooth: forward, reverse left, right and stop. Next steps are to use speech recognition to drive the robot. The Bluetooth protocol the robot uses is shown below.

In order to call software files, or sound files, it's important to really understand the protocol architecture. To call a software file, you have to follow the protocol shown below under the Start Program. I will cover an example for calling a left function on the NXT. The left program would be already programmed into the NXT to turn the robot left. To call the left program in the NXT you have to call left.rxe.

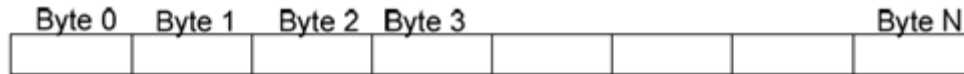


Figure 13: Byte Protocol

STARTPROGRAM

Byte 0: 0x00 or 0x80

Byte 1: 0x00

Byte 2 - 21: File name. Format: ASCIIZ-string with maximum size [15.3 chars] + Null terminator

Return package:

Byte 0: 0x02

Byte 1: 0x00

Byte 2: Status Byte

STOPPROGRAM

Byte 0: 0x00 or 0x80

Byte 1: 0x01

Return package:

Byte 0: 0x02

Byte 1: 0x01

Byte 2: Status Byte

<http://code.google.com/p/smart-robot/downloads/detail?name=Appendix%20-%20LEGO%20MINDSTORMS%20NXT%20Direct%20commands.pdf>

Blue Tooth Protocol File Calls in VB

Byte 0: How many bytes total in your message. Don't count bytes 0 and 1.

Byte 1: Always send 0x00 for this byte.

Byte 2: You have two options here. If you send 0x00 then the NXT will send you a return message. If you send 0x80 then no return message is sent. For most of the programs that I wrote I left this byte programmed with 0x00.

Byte 3: This is the command Byte. Sending the byte 0x00 tells the NXT that you are going to call a function.

Byte 4 - 21: In these bytes you send the ASCII equivalents of your file name. For example, byte 4 would be coded with the ASCII equivalent of the first letter in your software file name. In this example we are calling the software file right so ASCII ("r") is sent.

Byte 5: Here you would send the ASCII value for "e".

Byte 6: Send ASCII value of "f"

Byte 7: Send ASCII value of "t"

Byte 8: Send ASCII value of "."

Byte 9: Send ASCII value of "r"

Byte 10: Send ASCII value of "x"

Byte 11: Send ASCII value of "e"

Byte 12: Send null terminator character "0" to end string. This is important!

An example of calling the left program from the NXT is shown below. You can call all of your software files on the NXT by modifying the program code below to match your own software files in visual basic.

```
!*****  
!* This functions calles the left software file on the NXT  
!*****  
    Dim byteOut(13) As Byte  
        byteOut(0) = &HB '11 bytes in output message  
        byteOut(1) = &H0 'should be 0 for NXT  
        byteOut(2) = &H80 '&H0 = reply expected &H80 = no reply expected  
        byteOut(3) = &H0 'Command  
        byteOut(4) = Asc("l") 'l character  
        byteOut(5) = Asc("e") 'e character  
        byteOut(6) = Asc("f") 'f character  
        byteOut(7) = Asc("t") 't character  
        byteOut(8) = Asc(".") '. character  
        byteOut(9) = Asc("r") 'r character  
        byteOut(10) = Asc("x") 'x character  
        byteOut(11) = Asc("e") 'e character  
        byteOut(12) = &H0  
        SerialPort1.Write(byteOut, 0, 13)
```

Blue Tooth Protocol Mailbox in VB

In the end we implemented a far better method to control the robot using Bluetooth. The technique discussed earlier was still beneficial because it allowed us to run our final program on the NXT. However, for each robot function such as (forward, reverse, left, right, arm up, etc...) we sent values to the NXT mailbox. These values were then manipulated by the program running on the NXT and that function was performed. The commands below are all implemented and fully functional.

- Wrist Up
- Wrist Down
- Elbow Up
- Elbow Down
- Hand Open
- Hand Close
- Arm Left
- Arm Right
- Forward
- Reverse
- Right
- Left
- Stop
- Motor Power Up
- Motor Power Down
- Shake Hand Routine

Below is an example of the hand open function.

```
'*****  
'* Hand Open Function. This function opens the hand when it is called.  
'*****  
    Dim byteOut(9) As Byte  
    Try  
        byteOut(0) = &H7  '7 bytes in output message  
        byteOut(1) = &H0  'should be 0 for NXT  
        byteOut(2) = &H80 '&H0 = reply expected &H80 = no reply expected  
        byteOut(3) = &H9  'Command  
        byteOut(4) = &H0  'inbox 0  
        byteOut(5) = &H3  'message length 2 + 1 null  
        byteOut(6) = &H6  'protocal first byte  
        byteOut(7) = &H8A 'How far to move right  
        byteOut(8) = &H0  'null terminate  
        SerialPort1.Write(byteOut, 0, 9)  
    Catch ex As Exception  
        MsgBox(ex.ToString)  
    End Try
```

Visual Show Automation

Visual Show Automation (VSA), is a software GUI created by Brookshire Software. It allows one to control servos and automation using drag and drop techniques. The software was already purchased by the previous group and we were able to obtain a soft copy of the software from the president of the robotics society. The image below shows an example of the VSA screen. At the bottom of the screen a wav file is shown. This file can be imported by selecting tools -> load audio file from the menu. Before an animation is created, you have to define the servo definitions.

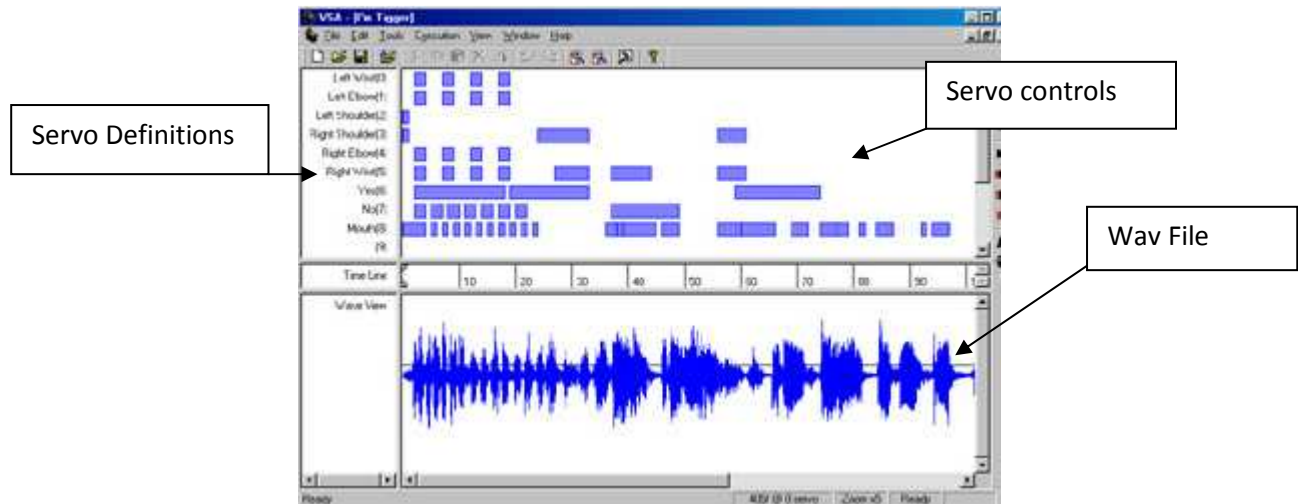


Figure 14: Visual Show Automation

The servo definitions are setup in the tools -> settings -> device settings. This is the first step when defining servo name as it allows you to setup minimum and maximum values. Below are the current settings of the servos in the Bohr robot. After the servos are defined, you can drag bars onto the screen. The length of the bars represent the time an animation will run. If you double click a bar you can edit the servo beginning and end positions. For example, to rotate the head from left to right, drag a bar onto the rotate head line. Double click the bar and edit the begin and end positions. The starting position for the robot head is 156 (robot straight), to move the head to the right edit the end value to be 45. Then when the animation plays and the timeline runs over the bar that you just created, the robot head will move from the straight position and end in the right position.

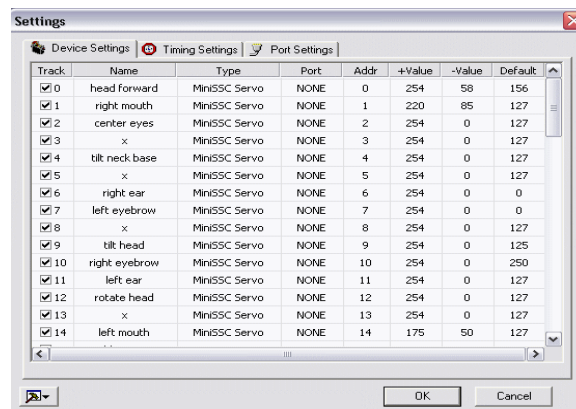


Figure 15: Visual Show Device Settings

You Tube Example

To demonstrate VSA, we choreographed a small skit and posted the video on You Tube. You can view the video by typing in Bohr robot in the search file or view it by clicking on the link below.

<http://www.youtube.com/watch?v=fRFuN9EYGwg>



Figure 16: You Tube Bohr Robot

Bohr GUI

The Bohr robot was controlled using a GUI created in Visual Basic. The GUI allows the user to implement all of the functions discussed in this paper which include the speech recognition system, Bluetooth connection for file and mailbox calls on the NXT and all robot functions.

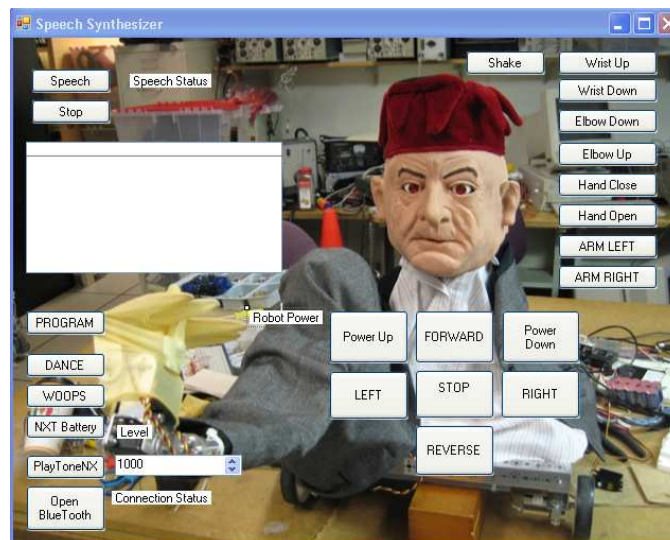


Figure 17: Bohr Robot GUI

SVN

All code for the Neils Bohr Robot can be found on the Open Source Repository of google.com

<http://ece478robot.googlecode.com/svn/trunk/>

Future Plans ECE479

The previous group controlled the robot using a wireless PS2 controller. We've been working on voice and vision control as well as smoothing out some of the basic motions. Like many other groups, in the future we would hope to make the robot more autonomous in nature with the ability to free roam. In order to do so we need to implement two key components.

First, more sensors would need to be added, giving the robot the ability to perform object avoidance. This would most likely be achieved by the use of sonar. This would also facilitate a need to stop using RobotC and switch to something like NXT++ which we have already begun. Depending on the number of sensors NXT++ makes it easier to integrate them all together.

Secondly an algorithm for mapping would need to be added, preventing the robot from wandering around aimlessly. It would also start a foundation for any control algorithms, such as A* or genetic algorithms, that can be implemented in order to reach a goal. In the end it would make the robot much more interesting and fun to interact with.