STATE UNIVERSITY OF NEW YORK AT BUFFALO

Department of Mechanical and Aerospace Engineering

Mapping and Remote Operation of the Lego Mindstorms NXT Platform

by

Colin Lea

Funded by

Gustav and Greta Zimmer Research Scholar Award

NSF Research Experience for Undergraduates

Advisor: Dr. Venkat Krovi

Fall 2008

Table of Contents Pages

Mapping and Remote Operation of the Lego Mindstorms NXT Platform

Due to length of Matlab code, it is available separately.

Abstract.

This experience report documents my application of the Lego Mindstorms NXT robotic platform and Nintendo Wii controller within Matlab to create a mapping and remotely operated system. Using sensor inputs from the NXT I have created a system to identify obstacles in front of the robot and store them to memory. With this information, the NXT is capable of avoiding previously detected obstacles. Mapping is a method of storing information about a location. The robot can be used to survey an area and provide feedback to the computer. Emphasis has been put on user input. It is possible to control using (1) a Nintendo Wii controller, (2) a graphical user interface, and (3) via the Internet.

Chapter 1: Introduction

Creating a robot from scratch takes a lot of time, money, intelligence, and craftsmanship. To develop, manufacture, and code a whole platform by your self is beyond the scope of a one-semester project. In order to focus on a single aspect of robotics one must isolate their area of interest. This could involve creating a robotic vehicle, creating a system of electronics, or in my case programming a mostly prebuilt robotic platform.

1.1 Background

There are many robotic platforms on the market. Many of these are sold as kits that allow users to create a robot of their own design. The Lego Mindstorms NXT and Vex Robotics Design System are such platforms. Included in these kits are wheels, motors, sensors, connectors, and other pieces. These types of kits enable all sorts of robots to be made. Other robotics platforms can be less modular, only allowing for a few configurations. For example, the Parallax Scribbler's body cannot be changed. It uses a Basic Stamp 2 plus sensors to allow users to program movement of the robot. Different robotic platforms are meant for different levels of control and functionality.

Figure 1 Lego Mindstorms NXT

Figure 2 Vex System

Figure 3 Parallax Scribbler

My project stemmed from interest in the FIRST Tech Challenge (FTC). This is a six-week competition during which teams of 10 high school students plus mentors work to compete in a robotic competition. Teams use a Lego Mindstorms NXT, motors, and other parts to create a semi-autonomous robot able to carry out a task. My goal was to get more familiar with the system and hopefully act as a student mentor for a local school.

1.2 FIRST Robotics

The goal of FIRST is to get kids interested in science, technology, engineering, and math (STEM) through robotics competitions. Different challenges exist for different ages. The Junior FIRST Lego League (Jr.FLL) is for ages 6-9 and involves basic research on a topic and modeling using Lego bricks. The FIRST Lego League (FLL) is for ages 9-14 and involves designing and building an autonomous robot using the Lego Mindstorms NXT platform. The students compete in a "friendly" competition where learning is stressed over winning. The FIRST Tech Challenge (FTC) is for high school aged students and is like a more advanced version of the FLL. It uses the Lego Mindstorms, motors, and other mechanical pieces to compete in a competition. The FIRST Robotics Challenge (FRC) is also for high school students and uses a National Instruments Compact Rio platform, motors, and other parts to create an autonomous robot. This is much more costly than the FTC. The students are expected to do the work but adult and student mentors should be available to guide and teach.

A few problems surfaced that thwarted my initial efforts to get involved with the FTC. First, there are no local teams currently competing - the closest school is more than an hour away. Second, the kickoff was September 13th which didn't allow enough time to try to organize a team at a local school.

Because of my initial research into FIRST I was already familiar with the capabilities of the Lego Mindstorms NXT platform. Combined with the large user base and plethora of online documentation the NXT was a suitable platform to do my own research.

Chapter 2: Equipment and Software

The NXT has an open programming interface which makes it relatively easy for developers to interact. Lego offers a software development kit (SDK), hardware development kit (HDK) and Bluetooth development kit (BDK) on the official website. This means that with the necessary skill anyone can build their own language that is capable of working with the NXT platform. Developers have taken full advantage of the SDK and HDK allowing much greater capabilities and more advanced robots.

2.1 Software

One of the advantages of the NXT is its large support in the software community. Independent developers and university students have created many programming languages and libraries to interface with the NXT. The software included is inadequate for anything more than basic manipulation and data acquisition. Several languages have been created that take advantage of C and Java libraries. While there are dozens of working languages, only a handful of them continue to have much support. Some more popular ones include RobotC, LeJos, Matlab and the Labview Toolkit

Likewise, because of Lego's open hardware development kit, companies are able to create additional sensors and motors to integrate with the Mindstorms NXT. HiTechnic, Vernier, and other companies sell additional adaptors for proximity detection, orientation and color sensing. Vernier offers many sensors for general data acquisition such as a flow rate sensor, barometer, and thermocouple. It is even possible to create your own sensors and attach them using a connector available through Lego.

2.2 Remote Control

While the Mindstorms NXT is fully capable of working autonomously, it is not always ideal. There are many solutions available to provide such control. However, using Matlab limits the number of usable controllers. Any unsupported controller would require a new library to be written. There are some popular devices that have lots of support and documentation online. For convenience, I chose something that is already supported.

One popular device is the Nintendo Wii controller (Wiimote). It features an ADXL330 3-axis accelerometer, a PixArt optical sensor for infrared detection, a direction pad, and 7 pushbuttons.

It is also capable of connecting with a “Nunchuck” controller attachment. This

Figure 4 Nintendo Wiimote

Figure 5 Nintendo Nunchuck

provides an additional 3-axis accelerometer, an analog stick for directional control, and two pushbuttons. The Wiimote has gained a lot of attention over the last couple of years. While nothing about the controller is completely revolutionary, its significance is in bringing motion control into the eye of the public. Because of its compact, capable design and inexpensive price, the Wiimote is being used in many robotics labs across the world. For example, research on 3D tracking for desktop virtual displays was recently done by Graduate students at the Human-Computer Interaction lab at Carnegie Mellon[1].

The unique abilities along with available Matlab drivers and documentation made the Wiimote a practical choice.

2.3 Goals

After tinkering with the NXT and Wiimote for a week or so I devised a general goal for what I wanted to accomplish. I sought to take advantage of what both products had to offer. I had experimented with the accelerometer data from the Wiimote but wanted to implement something on a more broad level. I wanted something that could take information from the Wiimote, manipulate it, take information from sensors on the NXT, and combine the data in Matlab to perform a function on the robot. I also wanted to be able to take information from the accelerometer and other control buttons on the Wiimote as well as take input from a

[1] http://www.cs.cmu.edu/~johnny/projects/wii/

graphical user interface (GUI) on a computer allowing for possible future development using networking to control the robot from a second computer. With all of the data I wanted to show the user what was happening with the system on the GUI.

I decided to create a system for mapping and remote control. The goal was to create a system that depicted where the robot was in relation to obstacles. The map is used to store information about an area such as the location of obstacles. Obstacles should be able to be inputted either by the user or detected by the robot. I wanted three modes:

- <u>WiiNav</u>: Controlled using the data from the accelerometer on the Wiimote
- <u>Fine Control</u>: Control using the directional pad on the Wiimote or using controls on the GUI. It allows for the robot to turn left, turn right, go forward a set distance, and go backward a set distance.
- <u>Point Find</u>: The user inputs a point and the robot has to move to that location

At the end of creating this system, I decided to set up remote control over the Internet. I had little experience with web development so I learned how to set up a server, script with PHP, and control this input through Matlab.

I started with basic knowledge of Matlab from an intro programming for non-computer science engineers course from freshman year (EAS230). Accomplishing my goals not only required solving the problems of controlling the NXT but also learning Matlab in greater detail.

2.4 Setting up the Lego Mindstorms NXT

One advantages of the NXT is its ability to communicate with a computer over multiple protocols. It is possible to connect using USB as well as via Bluetooth. With NXT-G it is possible to download programs directly to the NXT unit using either method. Being able to use Bluetooth eliminates problems with wires and makes testing more convenient. Unfortunately, current Matlab libraries only transfer data over Bluetooth. I had several problems getting the NXT set up on my Macbook Pro. Detailed documentation of how I got it to work is located in Appendix A. While it worked sporadically with the Bluetooth card in my laptop, my best results were with using a Broadcam adapter.

2.5 Programming the NXT in Matlab

While Mathworks, a company renowned for its excellent help system, created the Matlab NXT library, there is not a lot of documentation about how to control the robot. The documentation that does exist does not provide enough detailed information. It would be helpful to have previous experience handling COM ports. The best place to get started is on a blog entry written by a Mathworks employee[2]. Basic controls are listed below.

[2] http://blogs.mathworks.com/loren/2008/06/30/lego-mindstorms-nxt-in-teaching/

Function	Description
`NXT = legoNXT('COM18')`	Initiates NXT handle (COM# determined by the Bluetooth)
`Left_Motor = NXT.portB`	Initiates variable to output port. Output port is "A" "B" or "C."
`Start(Left_Motor, p)`	Starts motor with power "p" where "p" is a number 0-100
`Stop(Left_Motor)`	Stops motor
`Ultrasonic = NXT.port1`	Initiates variable to input port. Input port is "1" "2" or "3."
`set(NXT.Port4, 'type', 'distance');`	Initialize ultrasonic sensor
`Ultrasonicdata = getdata(nxt.Port1)`	Get input data

2.6 Programming of a Wiimote in Matlab

Two prominent Matlab libraries for the Wiimote are fWIIne and WiiLAB. fWIIne is an open source project that hasn't been updated recently. WiiLAB was developed during a summer undergraduate research program sponsored by the National Science Foundation at Notre Dame University. WiiLAB provides ample control through functions that enable accelerometer data acquisition, button readings and infrared detection. It is also capable of gathering data from the "Nunchuck" add-on.

M-files located in the Wiilab_Matlab folder are well documented and provide enough information to start using the Wiimote with little additional research. Basic controls are shown below.

Function	**Description**
`initializeWiimote()`	Initializes Wiimotes.
`isWiimoteConnected()`	Returns true if a Wiimote is connected.
`[x y z] = getWiimoteAccel()`	Gets accelerometer data. It is a range from -10 to 10. Values are returned to variables “x” “y” and “z.”
`isButtonPressed('A')`	Checks if button on Wiimote is pressed (Available buttons: A, B, ONE, TWO, UP, DOWN, LEFT, RIGHT, Minus, Plus, Home)

Chapter 3: Mechanical Design

The Lego kit includes a manual with instructions on how to build a robot capable of moving around and swinging a golf club-like Lego piece. I put this together rather quickly so that I could start setting it up in Matlab. At the beginning I was interested in the autonomous aspect of the project more than the mechanics of the physical robot. The robot had a myriad of sensors but did not have great mobility. It featured two motors to move the robot, one motor to swing the golf club-like piece, a bump sensor on the back to stop it if it ran into a wall, an audio sensor in front to detect noises, a light sensor to detect brightness levels, and an ultrasonic sensor pointed forward to detect obstacles in the robot's way. I constructed this robot without any of my own additions so I could begin programming.

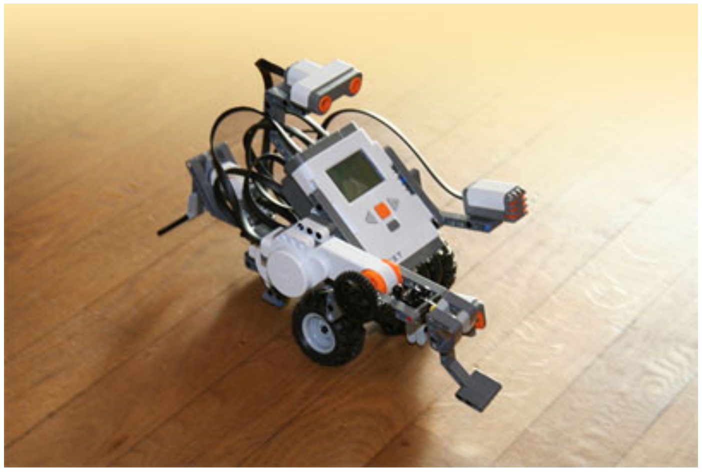


Figure 5 Initial Design (http://www.symbian-freak.com)

Problems arose before I even started developing my mapping system. While testing, I noticed that the robot did not turn well. It was inconsistent and wobbled some of the time. The drive train worked using two motors controlling the front wheels and a third uncontrolled wheel in the back. Unlike an omni-directional wheel, the back wheel must be aligned with the direction of the turn in order to rotate effectively. If the wheel was perpendicular to the turn, the movement would not be smooth and the whole robot would wobble. Instead of a rolling motion from the wheel, it increases friction, which creates drag on the vehicle. This is a big problem because the wheel is perpendicular every time that the robot moves forward or backward. Even if the wheel is at the correct angle to turn, there are problems when the robot then has to move forward.

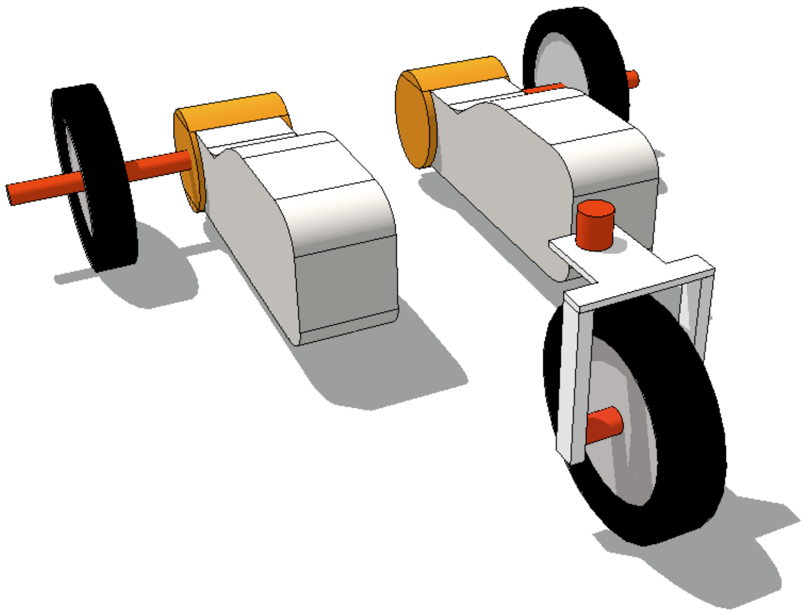


Figure 6 Three-wheeled robot

The problem was fixed by changing the whole design of the robot. I opted for a skid-steer drive train and only one additional sensor. Skid-steer works by isolating the left and right sets of wheels. The front and back of each side are connected so both wheels move simultaneously. This moves the center of rotation inward, allowing the robot to turn on a point. This solution is not perfect but allows for good movement using only two motors.

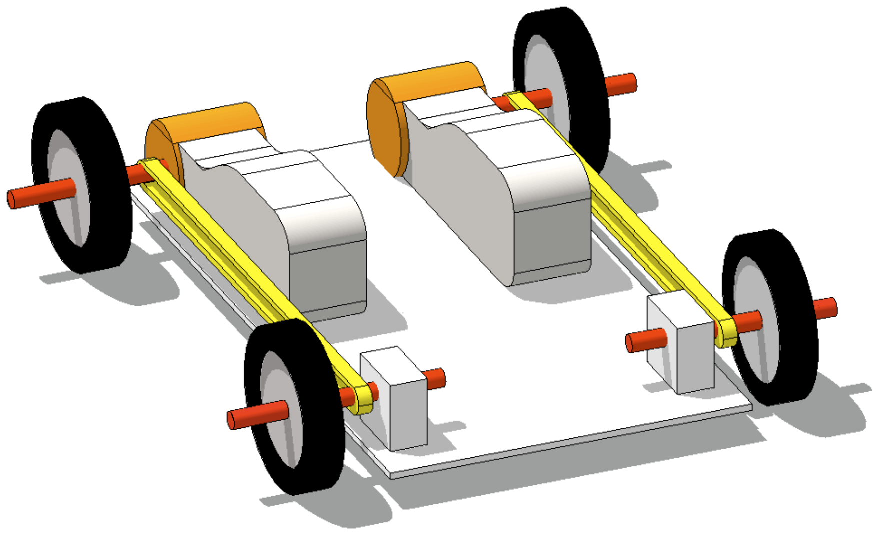


Figure 7 Skid Steer robot

Normally axles are connected with chains held on by gears. The Mindstorms kit does not come with anything like this so I had to come up with my own solution. I attached rubber bands to hubs fixed to each axle and got desirable results. It does not work perfectly but is satisfactory for my needs. Rubber bands offer more slack than chains because the only thing holding them in place is friction. On my robot, the motors are attached to the front axles. By experimenting, I found that the front and back wheels moved at the same rate as when it was driven by the motors. If you manually turn the back wheels, the front wheels move at a slower rate. This is because the motors add a greater resistance force than the frictional force in the rubber bands.

In larger vehicles, a skid-steer drive train can cause excessive wobbling. Because my robot has a low weight to size ratio I have not noticed significant problems.

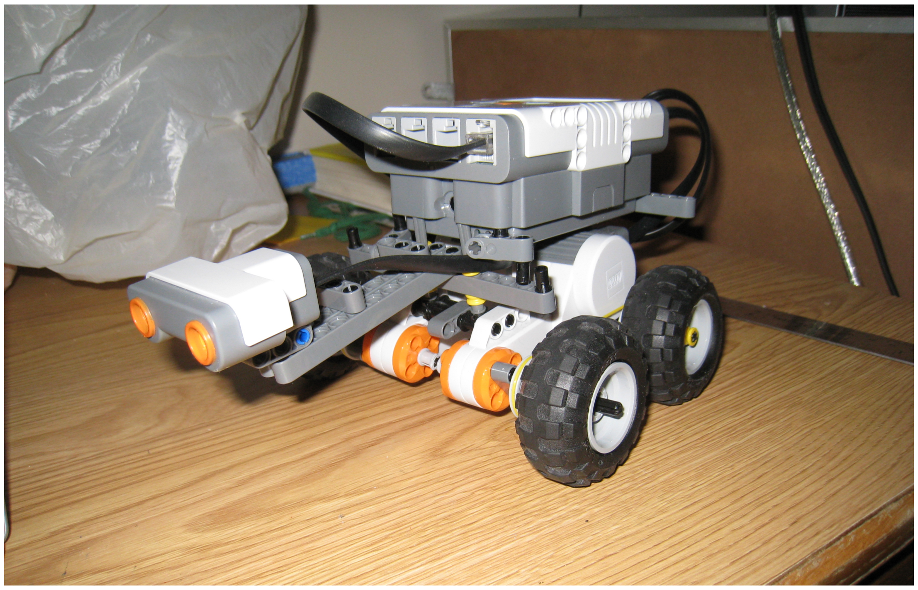


Figure 8 Final Design

Chapter 4: Software Design

The final program was the product of several iterations of code. Each revision used a new approach and brought new insight into how to use Matlab more effectively. I will briefly explain the different iterations and then talk about individual components of the final program. A diagram of how the parts of the system work is below.

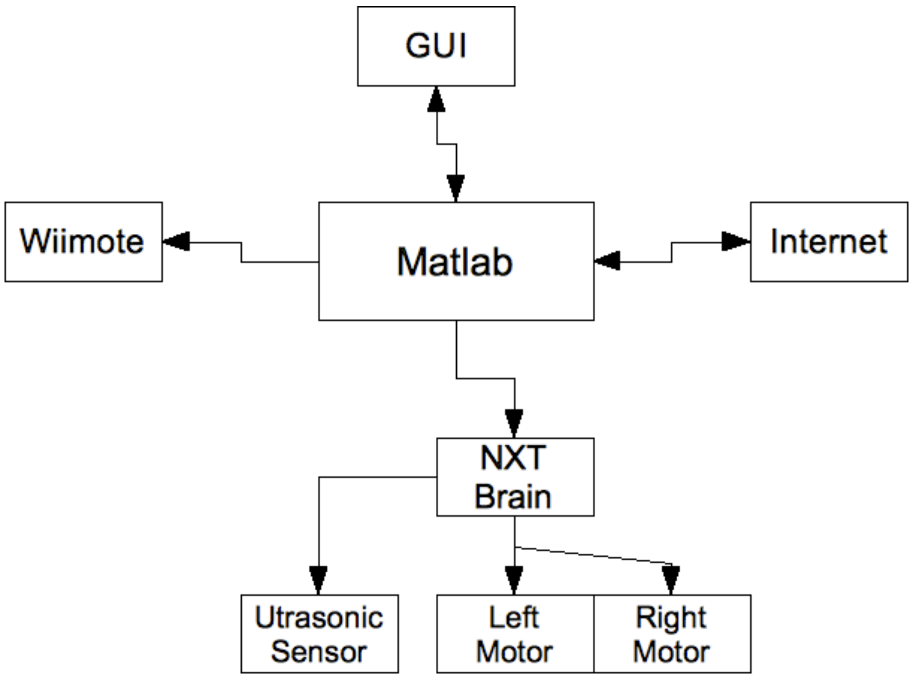


Figure 9 System Overview

4.1 Version history

The program started from the code I wrote while playing around with the NXT and Wiimote. Everything was crudely implemented. I had a system to control the motors on the NXT based on the accelerometer data and direction pad (d-pad) from the Wiimote. Accelerometer data from the Wiimote and the position of the robot were displayed on separate figures. All of the code was kept in one Matlab file, which ran continuously with the use of a while-loop.

Version 2 –Function based

I realized how inefficient and unreadable the code was in some areas. I removed much of the code from the main Matlab file and created routines that were stored in new m-files. I had a main while-loop that called separate functions for different operation modes. I also added comments to the files for better readability.

Version 3 – GUIDE based

At this point, I was concerned about overall usability. The main structure of the program had been created but it wasn't intuitive and relied on the command line for relaying information to the user. I spent a lot of time learning about how to create GUIs in Matlab so I could create one unified interface. Matlab's Graphical User Interface Design Environment (GUIDE) helps simplify the creation of GUIs. It included several types of buttons, sliders, and other controls that allow the user to interface with the program.

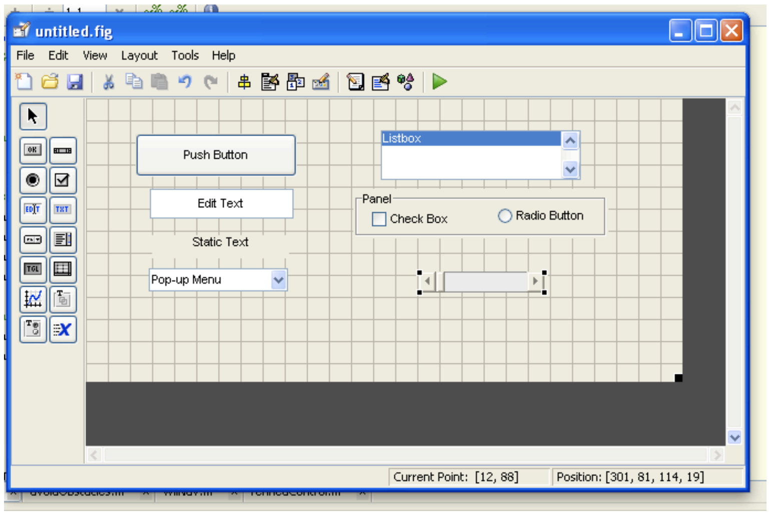


Figure 10 GUI Design Environment

The problem with this was that it required a complete overhaul of my code. It works by running "callback" functions whenever a button is pressed. There is a function for initializing variables and commands and then the program runs through the rest of the code repeatedly until it is stopped. Instead of running through a while- loop, checking whether certain modes have been selected, the program is interrupted by button presses. My biggest problem was allowing for control through the Wiimote and onscreen at the same time. I couldn't poll the Wiimote using a while-loop because it would essentially stop the program from processing other information. This required the implementation of a timer. A timer is set to run a certain function at a set interval. For example, my timer checks for Wiimote button presses every 0.5 seconds.

Version 4 – Internet Control

The last revision deals with remotely accessing the robot over the Internet. I set up a server in Windows that can be accessed from any Internet connection. This is discussed in detail in chapter 6.

4.2 Feature Implementation

4.2.1 Handle

The handle is the basis of any graphical user interface created with GUIDE. It acts as a structure for all information in the interface. In my program, everything is stored in the handle including the map, robot information, and button data. The stored variables are in the table below.

Variable	Description
clearMap	**Button:** Clears current map
Map	**Plot** of map
DistanceText	**Text:** shows value of distance slider
DistanceSlider	**Slider:** Controls distance robot travels
Turn_Right	**Function** for turning right
Turn_Left	**Function**
text10	**Text**: “X position”
ExitMode	**Function**
Heading	**Text**
Ypos	**Text**
Xpos	**Text**
WiimoteControl	**Function**
ExitProgram	**Function**
RIGHT	**Button**:
LEFT	**Button**:
DOWN	**Button**:
UP	**Button**:
text5	**Text:** “Robot Position”
text11	**Text:** “Y position”
text3	**Text:** “Distance”
Ultrasonic	**Plot**
WiiY	**Plot**
text2	**Text:** “Y Axis”
text1	**Text:** “X Axis”

WiiX	**Plot**
internet_mode	**Function**
PointFind_indicator	**Indicator**
Internet_indicator	**Indicator**
WiiNav_indicator	**Indicator**
Point_Find_Mode	**Function**
NXT_Connected_indicator	**Indicator**
Wii_Connect_indicator	**Indicator**
text4	**Text:** "Wiimote Connection"
text6	**Text:** "NXT Connection"
output	Unused part of handle
MODE	**Variable:** 1 for WiiNav, 2 for Fine Control, 3 for Point Find
EXITPROGRAM	**Variable** 1 on exit, 0 while running
NXT_Connect	**Variable:** true if NXT is connected
Lmotor	**Variable**: Stores the port the left motor is attached to
Rmotor	**Variable**: Stores the port the left motor is attached to
wii: [1x1 Wii]	Wii **structure**; includes accelerometer data
map_length	**Variable:** previously defined length of map
map_height	**Variable:** previously defined height of map
robotX	**Variable:** Current X-coordinate
robotY	**Variable:** Current Y-coordinate
robotH:	**Variable:** Current heading
object_Up	**Avoidance:** "1" if object above the robot
object_Down	**Avoidance:** "1" if object below the robot
object_Left	**Avoidance:** "1" if object to left of robot
object_Right	**Avoidance:** "1" if object to right of robot
object	**Avoidance:** "1" if object within surrounding area of robot
Clk	**Variable:** stores previous time; used to find change in time
T	**Timer:** Gets data from wiimote and runs wii functions
internet_timer	**Timer:** Gets information from
timeInit	**Variable:** Gets cputime at initialization

The process of using the handle during a callback is unique. At the beginning of the callback it must be reassigned to a new variable. To get the information you must use the function `handles = guidata(hObject)`. Variables can then be accessed in the format *`handles.[variable]`*. For example, the X position of the robot would be *`handles.robotX`*. At the end of the callback the handle must be saved using the function `guidata(hObject, handles)`.

4.2.2 Initialize

At the beginning of the program the NXT, Wiimote, timer, map, and variables must be initialized. These are all assigned to the handle to be stored for later use.

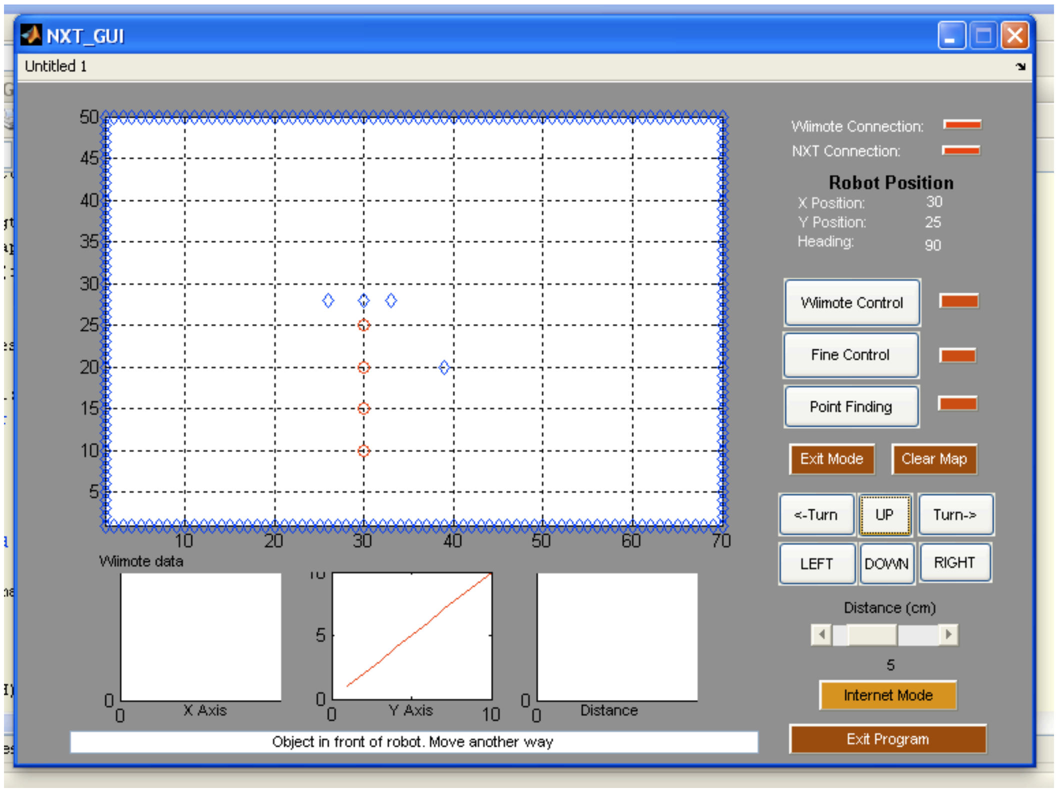
For testing purposes there needed to be a way to use the program without the NXT or Wiimote connected. Without some method of detection, there is an error and the program stops. If the NXT is detected, motors and the ultrasonic sensor are initialized. If it is not the motors must be set to dummy values.

```
try
    NXT = legoNXT('COM16');    %try to initialize the NXT
    handles.Lmotor = NXT.portB; %assign ports to left and right motors
    handles.Rmotor = NXT.portC;
    handles.NXT_Connect = 1; %used to check for NXT in routines

    set(NXT.portB, 'StopType', 'brake'); %motors stop immediately
                              %instead of coming to a rolling stop
    set(NXT.portC, 'StopType', 'brake');
    set(NXT.Port4, 'type', 'distance'); %initialize ultrasonic sensor

    start(wii.Lmotor, 0);  %initializes motors to a speed of 0
    start(wii.Rmotor, 0);

catch E1          % if the NXT is not connected set dummy variables
    disp('NXT not connected')
    set(handles.NXT_Connected_indicator, 'BackgroundColor', 'r')

    handles.NXT_Connect = 0;
    handles.Lmotor = 0;
    handles.Rmotor = 0;
end
```

The command `initializeWiimote()` takes care of errors and does not cause any problems. Other variables are assigned in the following format.

```
handles.wii = Wii(handles.Lmotor, handles.Rmotor, 2, -4);
handles.map = Coordinates(70, 50, 30, 10); %(Size X, Size Y, X robot
                                            position, Y robot position)
```

4.2.3 Map



Object in front of robot. Move another way

Figure 11 GUI overview

The Map is arguably the most important part of the GUI. It is used to show where the robot is, where obstacles are, and where the boundaries are. All of the information for the map is contained in a double-indexed array with a predefined width and height. Zeroes and Ones are used to identify where obstacles are located. The map is printed to the screen using the `plot()` command. The whole map is only printed at the start of the program and when the “Clear Map” button is called. Whenever an object is found or the robot moves the map’s array is modified but only the new point is plotted. This saves processing time because the map doesn’t have to reload every time a new event occurs. Blue diamonds signify obstacles and borders and red circles signify the position of the robot. Previous robot states remain shown to remind the user where the robot has already been.

While testing, one annoyance I found was resetting the program whenever the map became cluttered or I moved the robot. In order to clear the map I had to exit the program and restart, a process that takes at least 10 seconds. This can be a hindrance to anybody playing with the robot in multiple environments or testing it with different obstacles or in different scenarios. This is why I included the function "Clear Map." This button zeroes the map's array, adds a new border, and redraws the map.

Obstacle avoidance is one of the main reasons for creating the mapping system. When I didn't have the NXT connected, I needed a way to test if the robot could successfully evade objects. I also thought it would be useful to be able to add known objects to help eliminate errors in the object detection system. I wanted a system that was quick and intuitive to add objects on the fly. The method I implemented allows you to click anywhere on the map and it will immediately add an object in that location. When you click, the `figure1_WindowButtonDownFcn()` callback is initiated. It gets information from the figure window to find the position of the mouse on the map. This value is turned into an integer to be stored in the map's array. The point is then plotted in the figure.

```
pos = get(H.Map, 'CurrentPoint');
x = uint8(pos(1)); %uint8 used to get integer value
y = uint8(pos(1,2));
H.map.map(x, y) = 1;
plot(H.Map, x,y, 'bd')
```

One feature I would like to see added is an expandable map. Currently when the robot goes outside of the boundary, it creates an error. This is the reason I added boundary markers. A new numbering system would have to be used because an array does not accept negative numbers.

4.2.4 Data graphs:

The graphs provide visual output of data from the Wiimote. It is useful for testing and monitoring inputs. It displays the last 10 seconds of data. Originally, I plotted the X and Y values from the Wiimote and the Ultrasonic data from the NXT. The Ultrasonic data did not appear to be useful so I replaced it with the Z values from the Wiimote. The Z data is not currently used but might be in the future.

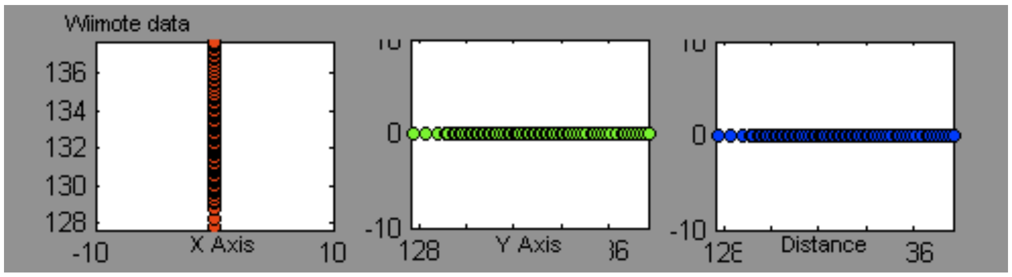


Figure 12 GUI graphs

4.2.5 Timer:

The timer is used to check the Wiimote for information every 0.5 second. If the “A,” “B,” or “One” buttons are triggered it switches the mode. I had trouble setting the timer up. The function I used was `handles.T = timer('TimerFcn', @(h, ev) timer_callback(hObject, eventdata),'execution', 'fixedRate', 'Period', .5).` It must then be started using `start(handles.T).` My problem had to do with the input arguments. The timer checks for the current mode and runs Wiinav and Fine Control when necessary. When the program quits the timer must be stopped and deleted using `stop(handle.T)` and `delete(handle.T)`.

4.2.6 Other GUI Elements:

Several indicators are in place to increase usability. When a mode is selected, the corresponding light turns from red to green. This design aspect is simply used to remind the user of the current mode. The robot’s position is also used to give an exact location so the user doesn’t have to estimate a value based on the map.

The “Exit Program” callback stops running processes and deletes the figure. It changes the *handle.MODE* to “0” which stops other modes from running. It stops and deletes timers that otherwise would continue in the background.

4.3 Modes:

4.3.1 Wiimote control

Controlling the NXT through the Wiimote was one of my first goals. Using Wiilab, it was easy to acquire sensor data from the Wii. I wanted to be able to control the robot using both accelerometer data (allowing the user to tilt the controller to move) and move by pressing directions on the d-pad. Because there are multiple options there needs to be a way to access each mode. Every 0.5 seconds Matlab polls the Wiimote for button data.

If the "A" button is pressed it enables "Wii Control Mode." When this is enabled, Matlab repeatedly polls the Wiimote for accelerometer data. Figuring out how to best use the accelerometer data took lots of testing. Many ways of manipulating the NXT motors with this data makes movement awkward or unstable. For example, if you use the X-axis acceleration for turning control and Y-axis for forward/backward controls it is not easy to go straight. It is hard to keep your hand steady enough that you don't drift to the left or right. To counter this I implemented thresholds to the turning speeds. If the X value is below a threshold it is disregarded and if it is above the threshold it adds the value to one of the motors. The threshold is 6 on a scale of 0-19.6 (2*Gravity).

When going straight, the power of both motors is set to `vel_R = vel_L = Wii.Y_value*y_multiplier`. The y-multiplier is 4 and was set empirically. The x-multiplier is 2. The robot's location on the map is changed with the function `H.robotX = (H.robotX + v*cosd(H.robotH)*dt)` where `v` is the average velocity of the left and right motors, `cosd` returns the cosine of the function in degrees, and `dt` is the time since the last function.

While turning left the power in each motor is given by `vel_L = (Wii.Y*y_multiplier) + (Wii.X*x_multiplier - 6))` and `vel_R = Wii.Y*y_multiplier`. Turning right is the reverse.

While turning, the change in angle is calculated by `dh = atand((vel_R-vel_L)/width)` this is added to the current angle by the function `H.robotH = mod((H.robotH + (dh*dt)), 360)`. The use of "mod" reduces the angle to a value of 0-359. This angle is used to find the new X and Y coordinates of the robot.

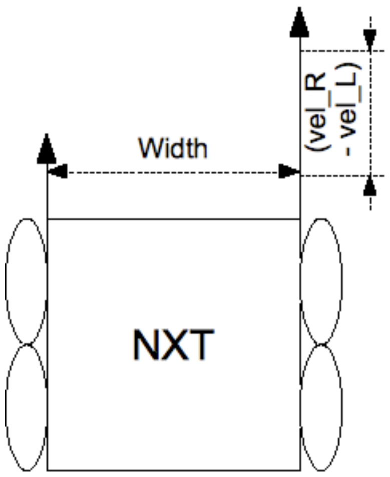


Figure 13 Geometry for Change in Heading

One problem with "Wii Control Mode" pertains to the mapping system. Because the NXT can move at any angle and at any speed, there are sometimes problems where the actual location of the robot drifts from the mapped location.

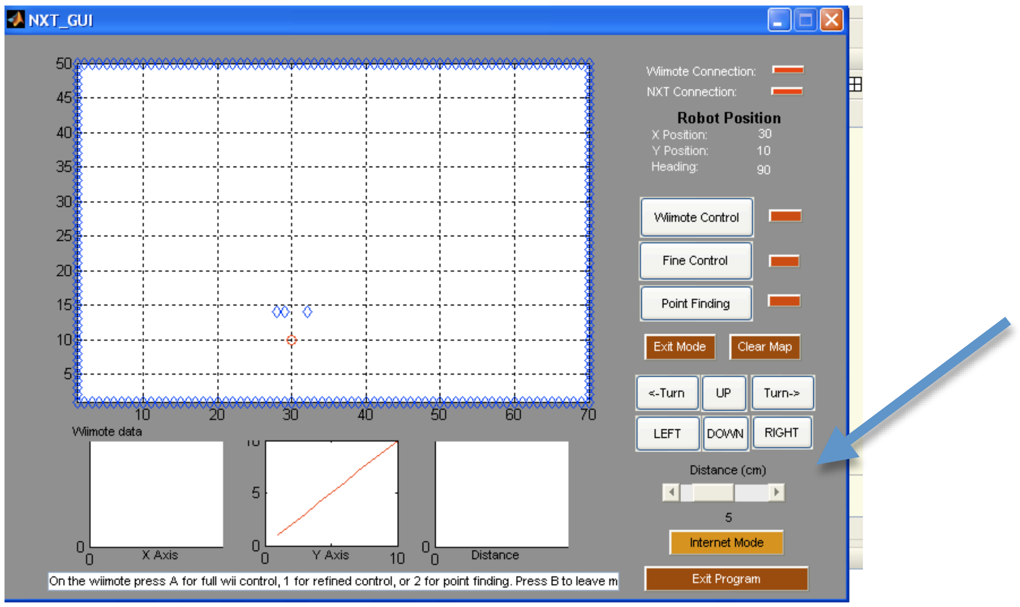
This is in part due to wheel slippage and errors vary with the floor's surface. Adding a digital compass could help eliminate part of this problem.

If the "One" button is pressed it enables "Fine Control." In this mode, users can move the robot forward or backward or turn it left or right using the d-pad on the Wiimote. As with "Wii Control Mode," Matlab repeatedly polls the Wiimote for button presses from all directions on the d-pad. When pushing "Up" or "Down" the robot moves a distance of 5 centimeters. "Left" or "Right" turns the robot 90 degrees counterclockwise or clockwise. This uses the same routines talked about later in the "Fine Control" section.

Regardless of which mode it's in, Matlab also polls the "B" button. The "B" button is used to exit a mode. While in "Wii Control Mode" exiting immediately stops the NXT. While in "Fine Control" the previous action is finished before stopping.

4.3.2 Fine Control:

This mode accepts input from the GUI and Wiimote. It's use it to accurately move the robot a defined distance. By default, the robot moves 5 cm at a time. This can be changed with a slider on the GUI but is unchangeable via the Wiimote.



On the wiimote press A for full wii control, 1 for refined control, or 2 for point finding. Press B to leave m

Figure 14 Distance Slider

On the Wiimote, the d-pad controls are relative. The “Up” button calls the *forward( )* function, “Down” is *backward( )*, “Left” is *turn_Left*, and “Right” is *turn_Right( )*. For example if the robot is turned left, pushing “Up” will move it to the left on the map. On the GUI, controls are absolute. Each button runs its own callback function. The orientation is checked and if it is not facing the direction chosen then it turns left before moving forward.

```
% --- Executes on button press in UP.
function UP_Callback(hObject, eventdata, handles)

H = guidata(hObject); %handle
while (H.robotH ~= 90) %if the heading isn’t 90 degree turn left
    H = turn_Left(H);
end

    H = forward(H); %move forward
    plot(H.Map, H.robotX, H.robotY, 'or') %plot the new location

guidata(hObject, H); %save the handle
```

Because it is non-holonomic (meaning it can only move in the direction that it is pointed), anytime the robot moves a distance it must use the forward or backward function. *forward( )* and *backward( )* get the value from the Distance slider to determine how long to turn the motors on for. On my testing surface the time function was determined experimentally to be `time = .094*distance`. The multiplier varies depending on the floor's texture.

4.3.3 Point Finding:

In "Point Finding " mode, the user inputs a pair of X and Y coordinates and then the robot moves to that location. It uses a basic algorithm to find the new point.

The general outline of the algorithm is as follows.

```
while (H.robotY ~= H.yInput) || (H.robotX ~= H.xInput)
        % ^ while the robot isn't at this location, do the following

        %if point is higher than current robot position:
        if (H.yInput > H.robotY) && H.object_Up == 0
             while (H.robotH ~= 90) %while not facing up, turn
             H = turn_Left(H);
            end

            H = forward(H); %move forward

        %if point is lower than current robot position:
        elseif (H.yInput < H.robotY)&& H.object_Down == 0
             while (H.robotH ~= 270) %while not facing down, turn
                  H = turn_Left(H);
            end

            H = forward(H);

        %if point is more left than current robot position:
        elseif (H.xInput < H.robotX)&& H.object_Left == 0
            while (H.robotH ~= 180)%while not facing left, turn
                  H = turn_Left(H);
            end
             H = forward(H);
        %if point is more right than current robot position:
        elseif (H.xInput > H.robotX)&& H.object_Right == 0
             while (H.robotH ~= 0)%while not facing right, turn
                    H = turn_Left(H);
             end
H = forward(H);
         end
 end
```

One known problem is the possibility that the robot can get cornered into obstacles. The problem occurs when there are objects on the two faces closest to the endpoint. Further investigation into a better algorithm would provide better results and fewer problems. In the image below, the program has crashed because there is no way for the robot to go.

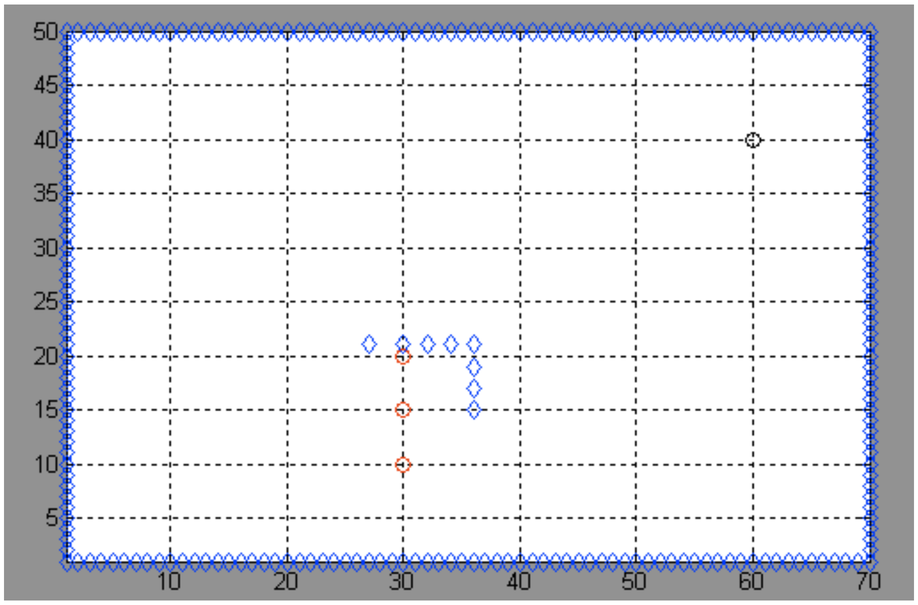
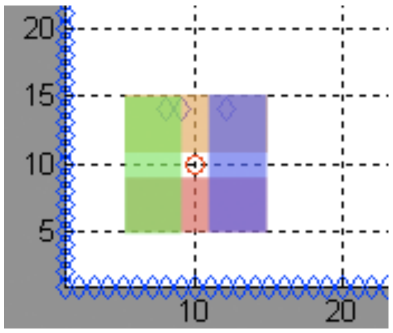


Figure 15 Point Find Error

4.4 Object Avoidance:

Avoidance is implemented with a function that is run every time the robot



moves. Its purpose is to prevent the robot from running into obstacles. It checks a square area equal to twice the travel distance plus one in length and width. By default this is an 11x11 centimeter area; the robot moves five spaces at a time and 5x2 + 1 = 11. The function checks for any non-zero number in the map's array. Because only objects in the way of the robots path need to be avoided there needs to be a way to differentiate which objects matter. To do this I added different zones within the avoidance function. The image below shows the four zones. In the code, the zones are *`handle.object_Left, handle.object_Up, handle.object_Right, and handle.object_Down`*. In this example `object_Left, object_Up`, and *`object_Right`* equal 1. Whenever there is an obstacle anywhere within the zone the value is 1, otherwise its value is 0.

The avoidance function is called before every movement. The call is located in the **forward**, **backward**, **turn_left**, and **turn_right** m-files. My control system is based on the robot's current heading which created problems. Initially if there was an obstacle above the robot and you were moving right, the system would think the object was in your way. The system was treating the forward and backward commands as absolute directions. To fix this I used a switch in both of the routines.

```
switch handle.robotH                % robotH is the robot's heading in degrees
        case {0, 360}
            obj = handle.object_Right;
        case 90
            obj = handle.object_Up;
        case 180
            obj = handle.object_Left;
        case 270
            obj = handle.object_Right;
        otherwise
            obj = handle.object;
 end
```

The variable `obj` is assigned to the direction of the robot. This way the command is only checking for objects in the way of it's path.

One thing I would like to see done is better integration with the point finding function. Currently the avoidance function is called every time the robot moves. If there are obstacles in the way of the robot's path, it must avoid them; this increases the time it takes to travel to the endpoint. If avoidance was calculated before any movement then it could choose to take a more efficient path. In the example below, the red dotted path shows the current path the robot takes to move to point (60,40). The green line depicts a better path using only 90-degree turns. The yellow line is the overall most efficient path.

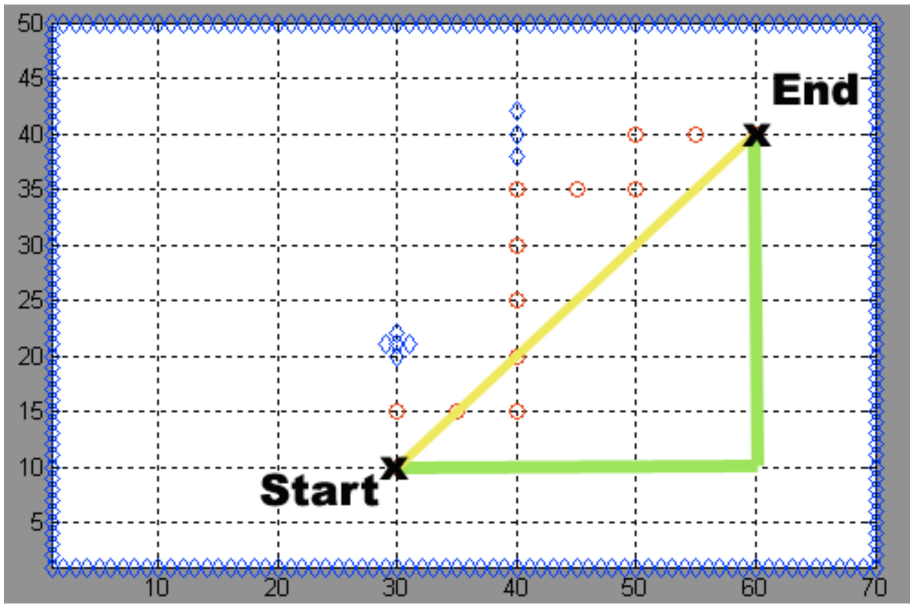
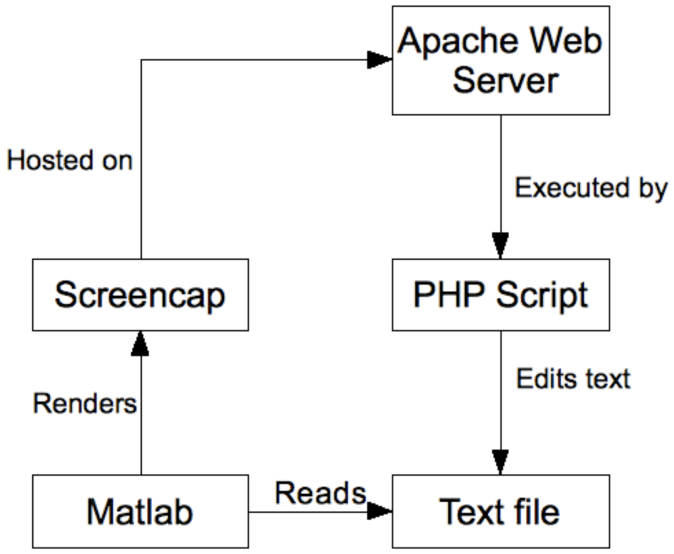


Figure 16 Optimal Path

Chapter 5: Internet control

After completing the structure of the Matlab program, I pondered the idea of controlling the robot over the Internet. I had little experience with web development so the end design comes from a somewhat basic approach. During the process, I learned how to create a web server hosted on my laptop and how to use HTML and PHP to create scripts for accessing and writing data to files and controlling user input.



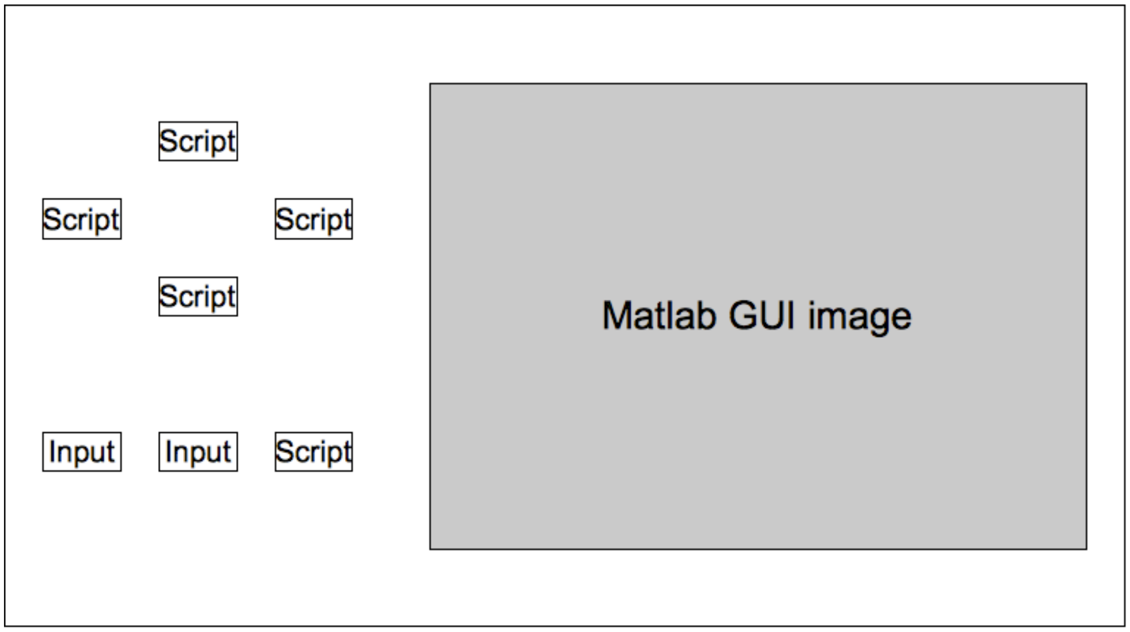
5.1 Server

Apache HTTP Server is an open source, commercial grade implementation of an HTTP web server (http://httpd.apache.org/ABOUT_APACHE.html). It is free and easy to set up for Windows and Linux. It runs in the background and allows anyone to connect to files hosted on my computer over the Internet.

PHP is a scripting language for web development. These scripts can be executed by button presses or other inputs from a webpage. With PHP it is possible to write to files on the host computer. In order to enable these scripts, PHP must be installed on the server. I followed this guide for setting up Apache and PHP (http://thecodecentral.com/2007/03/24/setting-up-your-own-web-server-with-apache-http-server-php-and-mysql-on-a-windows-machine#change_dir).

5.2 Webpage

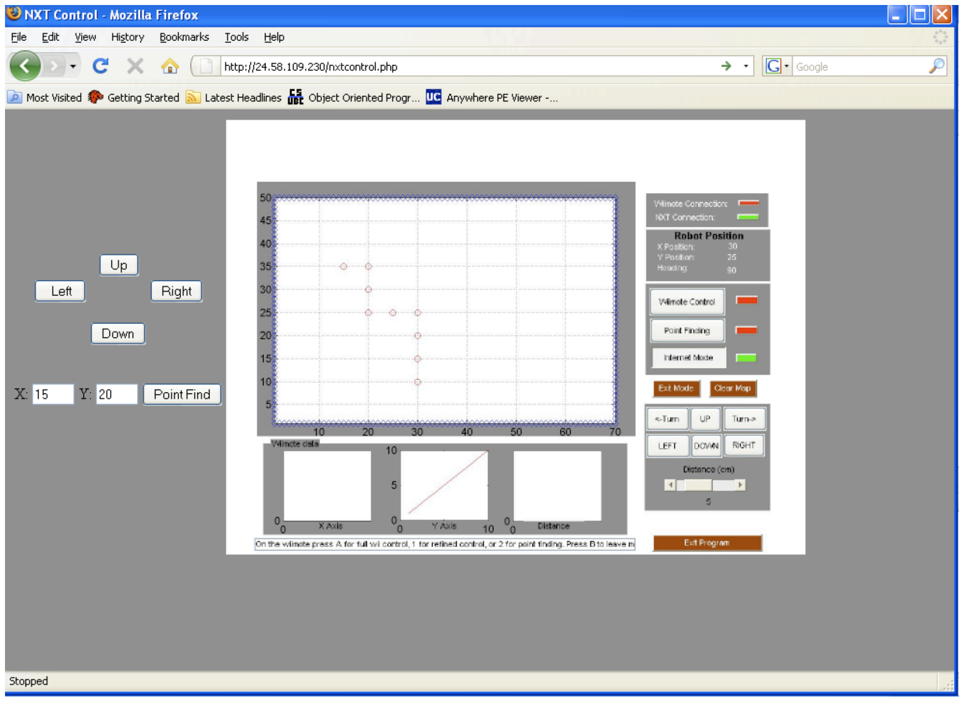
I wanted the robot to move with input over the Internet. I added two functions: Fine Control and Point Find. Using HTML I added buttons for "Up," "Down," "Left," "Right," as well as "X" and "Y" inputs for Point Find. To the right of the controls is a printout of the GUI from Matlab.



Each of the buttons opens *update.php* and posts the direction or coordinates selected. *update.php* reads the action and re-writes text file *update.txt* in the server's directory. An example of one of the buttons is below.

```
<form action='update.php?action=Up' method = "post">
<input type="submit" value="Up">
</form>
```

The "Internet Mode" button on the Matlab GUI starts a timer that checks this file every second. The format of the file is *[U][D][L][R] [X][Y] [Comment] [Timestamp]*. If the "Down" button is pushed the text might read *0100 00 00 (UP, DOWN, LEFT, RIGHT | X,Y COORD)1231101682*. Matlab would then run the Down button function. The function `saveas(H.Map, 'C:\...\htdocs\map.jpg')` prints a JPEG image of the GUI to a file on the server.



On the wiimote press A for full wii control, 1 for refined control, or 2 for point finding. Press B to leave m

Refreshing the image for the user became a problem. Due to the way websites cache images, the GUI didn’t update after every movement. A combination of solutions provided on web forums fixed the issue. I set Javascript to run a function that reloads the image after 2 seconds. I had to trick the page into thinking it was loading a different image by appending “?r=1” to the URL of the image when it is first loaded and “?r=2” when it is reloaded 2 seconds later. The letter “r” is arbitrary.

```
<img src='map.jpg?r=1' width=600 height=450>


<script language="Javascript">

function reloadPage()

       {
       document.images[0].src="map_old.jpg";
       document.images[0].src="map.jpg?r=2";
       }

       setTimeout('reloadPage()', 2000);

</script>
```

Chapter 6: Final Remarks/Future Work

Over the course of this project I have become much more proficient at Matlab and have a better understanding of how to mix hardware and software for remote operation of a robot. I think it has given me some of the basic skills necessary for more advanced research. In the future I would like to expand on this work, possibly implementing the following ideas:

- Attach a Wiimote to the robot. This allows for dead reckoning, which would give more accurate location information.
- Implement a better algorithm into Point Find mode so the robot doesn't get stuck behind obstacles.
- Further refine the way the robot moves while controlling it with a Wiimote.
- Use other NXT sensors to detect other actions such as sound.
- Obtain a digital compass (available through HiTechnic) for more accurate control.
- Use an additional Wiimote to create a 3D mapping system.
- Use additional robots to simulate a robot collective.

Creating this system has taken a lot of work but has been a fun experience and has solidified my interests in robotics. The combination of mechanical and computer engineering has long intrigued me, but until now, I have had little experience mixing the two. I look forward to doing more work in this area.

Appendix A: Bluetooth Setup

The Lego Mindstorms NXT provides a way to interface with many computer languages over Bluetooth. The NXT in theory should simply connect in Windows and OS X with their native Bluetooth setup wizards. Due to a slew of possible hardware configurations and bad on-screen communication on the NXT it does not work exactly as intended. As evidenced by dozens of web posts, this problem is a widespread issue without clear indication from Lego of what will work with different hardware setups.

3.1.1 Hardware

Lego lists a series of Bluetooth adaptors[3] that work with the NXT. All but one of the tested adaptors is said to work. A much larger set of adaptors can be found online[4] through various forums and websites. A large number of adaptors don't work so it is useful to check individual adaptors before buying.

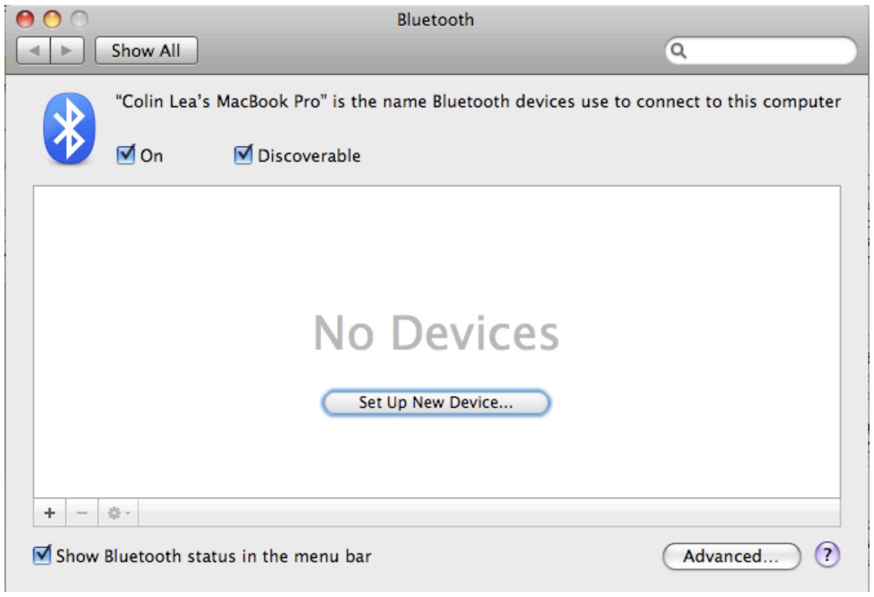
My test system is a Macbook Pro (MBP), which is compatible with the NXT. I am doing testing in OS X natively as well as in Windows XP through VMware Fusion v2.0 and through Bootcamp. For users with OS X 10.5 (Leopard) there is an update necessary to upgrade firmware[5]. I have found that the MBP's internal Bluetooth works some of the time, but in the end had the most success with an adaptor made by Broadcom.

[3] Bluetooth adaptors: http://mindstorms.lego.com/overview/Bluetooth.aspx
[4] http://www.nabble.com/NXT-compatible-bluetooth-adapters-td5161928.html#a5161928
[5] http://mindstorms.lego.com/support/updates/

3.1.2 Connecting – OS X

Connecting through OS X is relatively simple compared to Windows. Before trying to connect, you must close any previous connections with the NXT that you might have with that computer. After closing any connections, you should reset the NXT before trying to connect again.

To connect you must first open the Bluetooth devices window, which can be found in system preferences. Click 'Set Up New Device…'

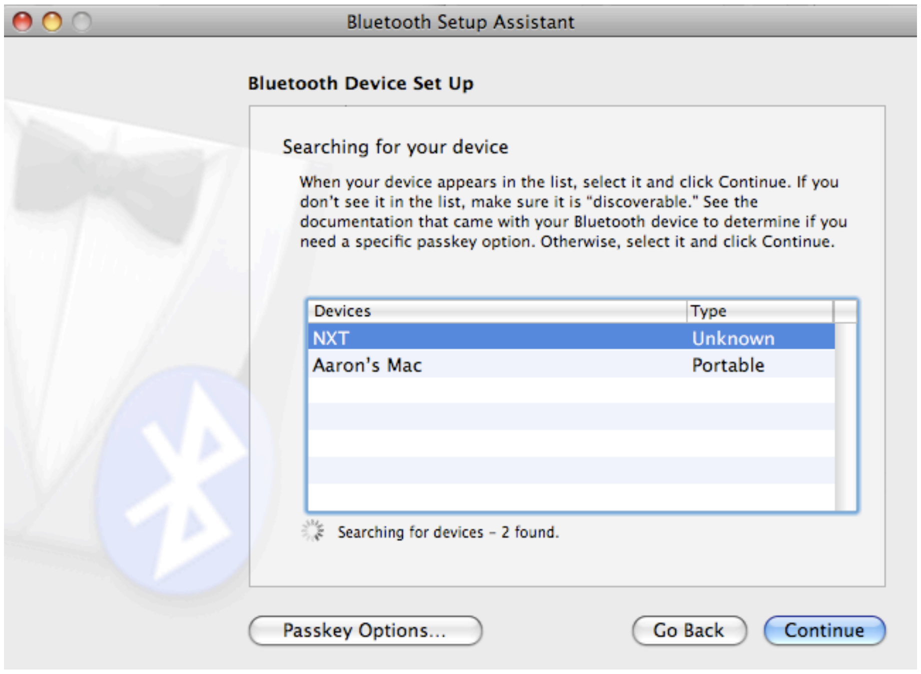


"Colin Lea's MacBook Pro" is the name Bluetooth devices use to connect to this computer

☑ On ☑ Discoverable

☑ Show Bluetooth status in the menu bar

Click ‘Continue’ and then select ‘Any Device’ when asked what device type. On the next screen select ‘NXT’ and click continue.



Bluetooth Device Set Up

Searching for your device

When your device appears in the list, select it and click Continue. If you don't see it in the list, make sure it is "discoverable." See the documentation that came with your Bluetooth device to determine if you need a specific passkey option. Otherwise, select it and click Continue.

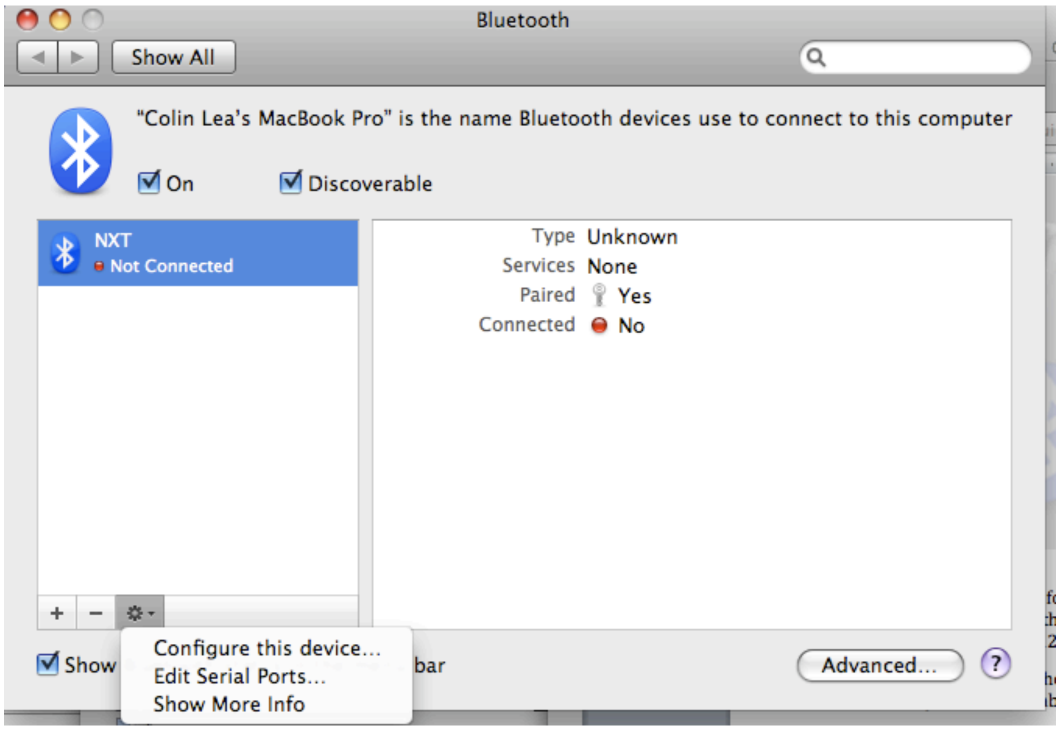
Devices	Type
NXT	Unknown
Aaron's Mac	Portable

Searching for devices – 2 found.

On the NXT it will ask for a passkey. It should default to ‘1234’. Click the orange button for OK. It will then ask for the passkey on the computer. If you chose anything other than ‘1234’ you will have to repeat the passkey on the NXT.

Click continue on ‘Gathering more information’. The next screen will say it cannot find any information about the device. This is correct.

You should now be *paired* with the NXT. This is not the same thing as being connected. At this point you cannot communicate between the platforms. Connecting requires one more step. In the Bluetooth window on your computer go to ‘Edit serial ports…’



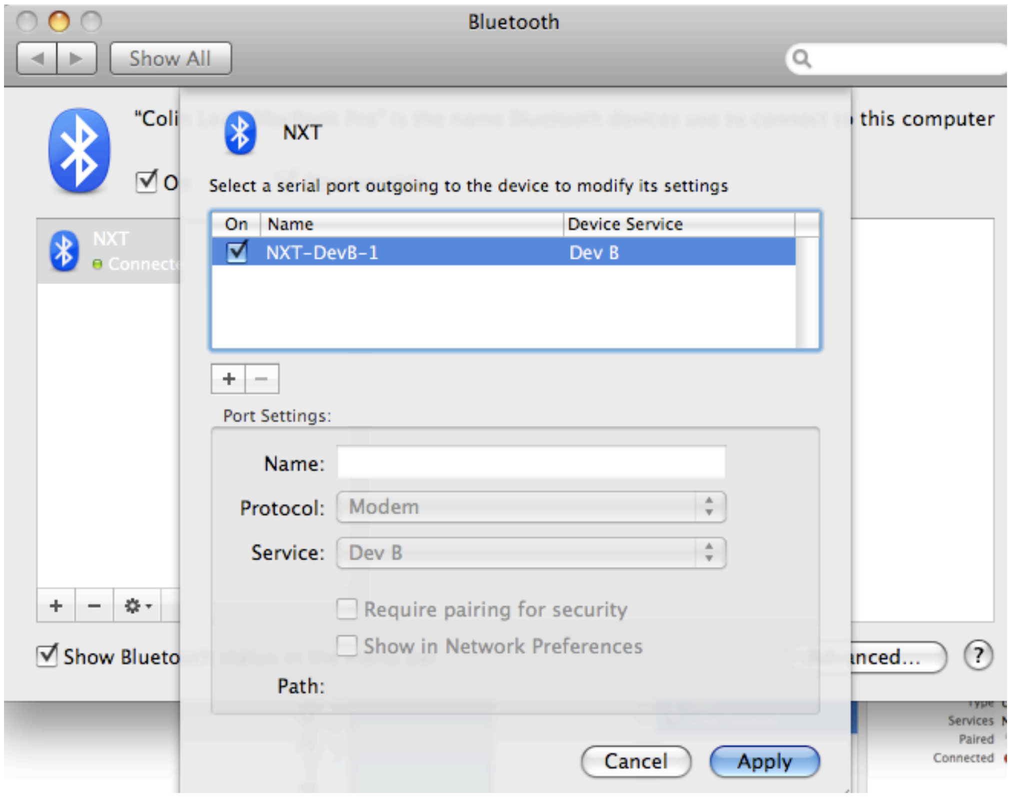
"Colin Lea's MacBook Pro" is the name Bluetooth devices use to connect to this computer

NXT
Not Connected

Type	Unknown
Services	None
Paired	Yes
Connected	No

Click the '+' to add a serial out. By default the device service should be 'DevB'.

If it is not, change the service type.



NXT

Select a serial port outgoing to the device to modify its settings

On	Name	Device Service
✓	NXT-DevB-1	Dev B

Port Settings:

Name:

Protocol: Modem

Service: Dev B

Require pairing for security

Show in Network Preferences

Path:

To check information about the NXT there is a program called NXTBrowser[6]. This gives information such as battery life and firmware version as well as it allows you to run built-in commands on the robot.

If you get a ‘Line is Busy’ error on the robot it means you did not set it up correctly in OS X. It may still work intermittently but occasionally you may face connection errors.

3.1.3 Connecting – Windows XP

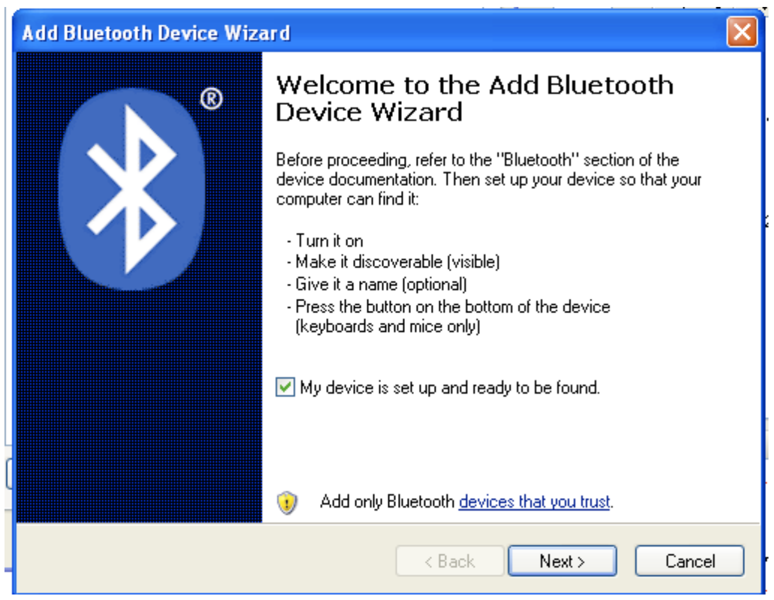
Connecting through Windows XP gives more problems if you don’t do it correctly. While the connection may work temporarily even if you don’t do it right, it is more prone to flaking out. To connect you must use either Window’s native Bluetooth drivers or if you company supplied drivers if you have a Widcomm or Broadcom Bluetooth adapter.

Under my test setup (running Windows XP through VMware or Bootcamp) connecting can be tricky and inconsistent. There is little documentation online about using the NXT with VMware and after spending many hours, I still have problems. There are several different methods that when combined will usually work.

[6] http://web.mac.com/carstenm/Lego/NXT/NXT.html

Start by closing any connections in Bluetooth Devices on the NXT, resetting the NXT and clearing any previous NXT Bluetooth devices from the Bluetooth Control Panel in Windows (which can be found in the system tray or in the control panel).

On Windows, in 'Bluetooth Devices' click 'add' under the 'Devices' panel. Check 'My device is set up and ready to be found' and click 'Next.'


Add Bluetooth Device Wizard

Welcome to the Add Bluetooth Device Wizard

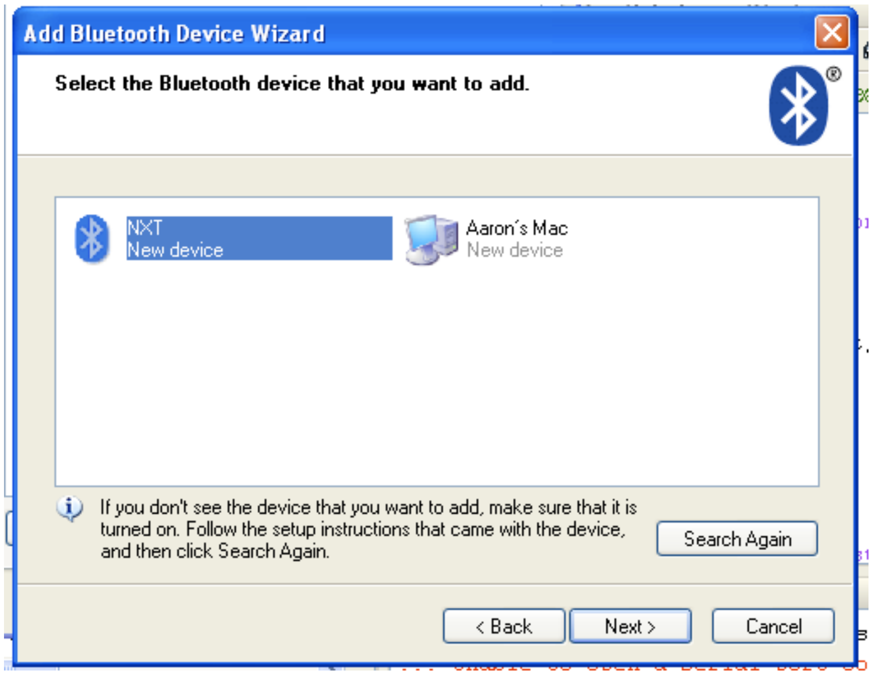
Before proceeding, refer to the "Bluetooth" section of the device documentation. Then set up your device so that your computer can find it:

- Turn it on
- Make it discoverable (visible)
- Give it a name (optional)
- Press the button on the bottom of the device (keyboards and mice only)

[✓] My device is set up and ready to be found.

Add only Bluetooth devices that you trust.

The NXT should show up as a new device. Select it and click 'Next.' If it is not listed make sure the NXT is turned on and click 'search again.'



Add Bluetooth Device Wizard

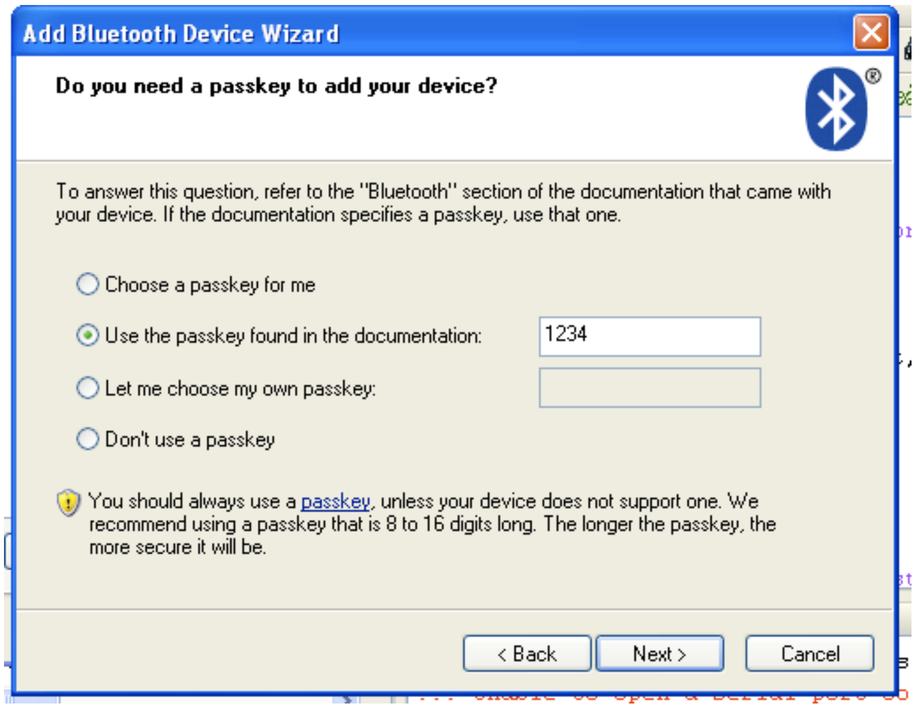
Select the Bluetooth device that you want to add.

NXT
New device

Aaron's Mac
New device

If you don't see the device that you want to add, make sure that it is turned on. Follow the setup instructions that came with the device, and then click Search Again.

Next, select 'Use the passkey found in the documentation' or 'Let me choose my own passkey.' Either method should work. Enter '1234' as the passkey.



Add Bluetooth Device Wizard

Do you need a passkey to add your device?

To answer this question, refer to the "Bluetooth" section of the documentation that came with your device. If the documentation specifies a passkey, use that one.

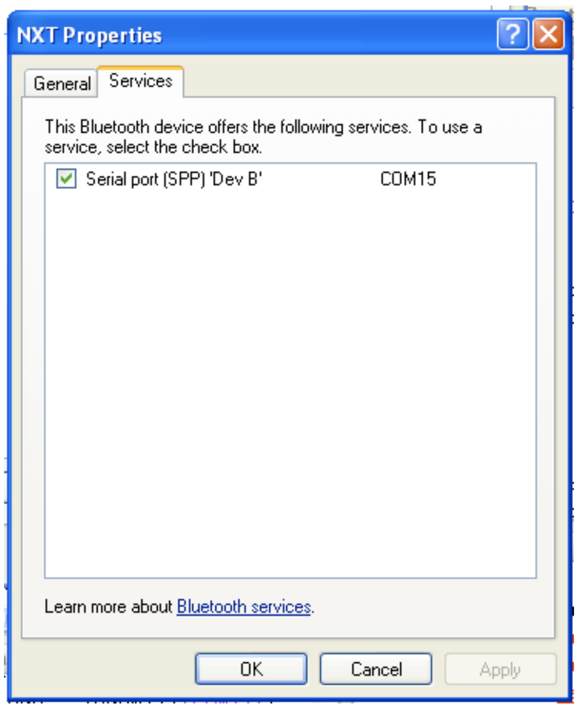
( ) Choose a passkey for me
[✓ Use the passkey found in the documentation:] 1234
( ) Let me choose my own passkey:
( ) Don't use a passkey

You should always use a passkey, unless your device does not support one. We recommend using a passkey that is 8 to 16 digits long. The longer the passkey, the more secure it will be.

If prompted on the NXT, the default the passkey here is also ‘1234’. Click the orange button (enter) on the NXT. Click next on the computer and you should now be paired. This does NOT mean you are connected. Your computer and NXT are now connected but they cannot communicate.

Getting the devices to connect is more complex and can create problems. One or a combination of the following should allow you connect.

1) In Bluetooth Devices select the NXT and click properties. In the ‘Services’ panel all of the services should be connected. By default, this may have between zero and two options, all of which may be either checked or not. All of them should be checked.

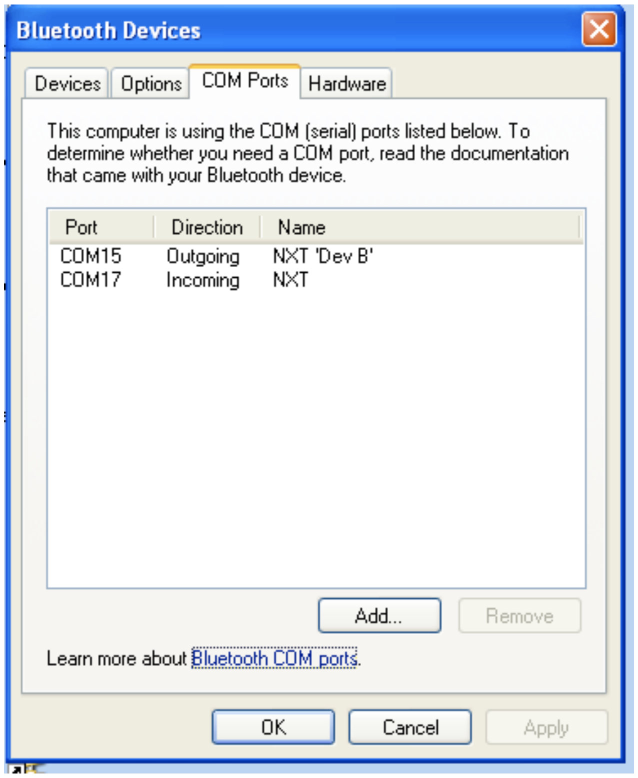


This Bluetooth device offers the following services. To use a service, select the check box.

✓ Serial port (SPP) 'Dev B' COM15

Learn more about Bluetooth services.

2) In Bluetooth Devices select the 'COM Ports' panel. There should be two connections - One that is outgoing and one that is incoming.

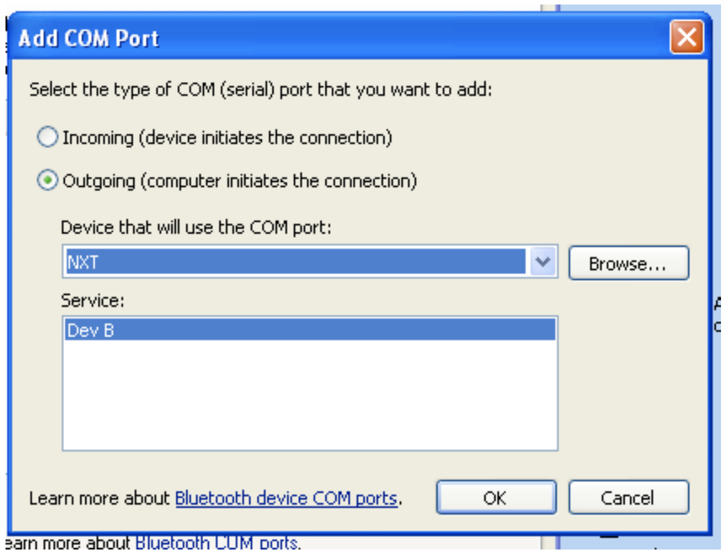


This computer is using the COM (serial) ports listed below. To determine whether you need a COM port, read the documentation that came with your Bluetooth device.

Port	Direction	Name
COM15	Outgoing	NXT 'Dev B'
COM17	Incoming	NXT

Learn more about Bluetooth COM ports.

If the outgoing is missing then go to 'Add' at the bottom of the window. Select 'Outgoing' and change the device to 'NXT' in the drop down menu. Choose Dev B and click 'OK'


Add COM Port

Select the type of COM (serial) port that you want to add:

Incoming (device initiates the connection)

[✓ Outgoing (computer initiates the connection)]

Device that will use the COM port: NXT

Service: Dev B

Learn more about Bluetooth device COM ports.

3) Go to Bluetooth on the NXT and select 'search.' You should see your computer once it's done searching. Select it and chose any number. It should say "Connecting." Once it is done it will either say "Failed," "Line is busy," or it will return to computer selection.

If it returns to the computer selection screen it means that the NXT is connected. You can check the connection by going back to Bluetooth on the NXT and selecting 'Connections.'

If it says "Failed" reset the NXT, delete the connection on the computer, and try again.

If it says "Line is busy" then it can mean a couple of things. It may actually work with the computer. If this is the case you should test using available software. If it doesn't work then you need to try something else. A combination of the other solutions may work. In this case, the NXT usually detects the computer but is missing a valid connection.

To use the connection with other programs go to the COM Ports panel in Bluetooth Devices to check which port is ‘outgoing.’ This port is needed to send commands to the NXT.

3.1.4 Bluetooth remarks

If you still cannot get the NXT to work, search online. There are dozens of web posts dedicated to connecting the NXT over Bluetooth. Many go unsolved but reading through them may provide new problem solving techniques. Make sure you are using the right drivers and have a compatible device. I have found certain methods will only work some of the time. According to online posts, if you have the NXT-G software it is somewhat simpler to set up. This software only comes with the consumer NXT set. It is available for about $50 for the users of the Educational kit. Bluetooth on the NXT is far from perfect and aside from using it with Lego’s own NXT-G does not have the best support from Lego.

It is worth noting that once you get a consistent connection you may never have problems again. I kept the same connection for over a month with no problems.

Appendix B: Internet Operation Code

nxtControl.php

```
<html>

 <meta http-equiv="Pragma" content="no-cache">
 <meta http-equiv="Cache-Control" content="no-cache">
 <meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT">

<title>NXT Control</title>
<body>
<body bgcolor = "Gray">
<table>

<td>
<center>
<table>
<form action='update.php?action=Up' method = "post">
<input type="submit" value="Up">
</form>

<td>
<form action='update.php?action=Left'method = "post">
<input type="submit" value="   Left  ">
</form>
 </td>
 <td> </td>
<td>
<form action='update.php?action=Right'method = "post">
<input type="submit" value="Right">
</form>
</td>

<tr>
<td>
<td>
<form action='update.php?action=Down' method = "post">
<input type="submit" value="Down">
</form>
</tr></td>
</table>
<br>

<form action='update.php?action=coord' method="post">
X:
<input type="text" name="X" value="15" size=3>
Y:
<input type="text" name="Y" value="20" size=3>
<input type="submit" value = "Point Find">
</form>
</center>
</td><td>
```

```
<img src='map.jpg?r=1' width=600 height=450>

<script language="Javascript">
function reloadPage()
      {
      document.images[0].src="map_old.jpg";
      document.images[0].src="map.jpg?r=2";
      }

setTimeout('reloadPage()', 2000);
</script>



</td></table>
</center>
</body>
</html>
```

update.php

Used: *update.php?action=Up, update.php?action=Down, update.php?action=Left, update.php?action=Right, and update.php?action=coord&X=x&Y=y*

```
<html>
<title>NXT Control</title>
<body>

<?php

$file = fopen("C:\\update.txt", "w+");
$value = $_GET['action'];
//$value = 'coord';
$time = time();
$writethis = 'Error in php update.txt';

if ($value == "Up") //find which button was pushed and create the new
output
      {
      $directions = '1000';
      $coords = ' 00 00 ';
      $comments = '(UP, DOWN, LEFT, RIGHT | X,Y COORD)';
      $writethis = $directions . $coords . $comments . $time;
      }
elseif ($value == "Down")
      {
      $directions = '0001';
      $coords = ' 00 00 ';
      $comments = '(UP, DOWN, LEFT, RIGHT | X,Y COORD)';
      $writethis = $directions . $coords . $comments . $time;
      }
elseif ($value == "Left")
      {
      $directions = '0100';
      $coords = ' 00 00 ';
      $comments = '(UP, DOWN, LEFT, RIGHT | X,Y COORD)';
      $writethis = $directions . $coords . $comments . $time;
      }
elseif ($value == "Right")
      {
      $directions = '0010';
      $coords = ' 00 00 ';
      $comments = '(UP, DOWN, LEFT, RIGHT | X,Y COORD)';
      $writethis = $directions . $coords . $comments . $time;
      }
elseif ($value == "coord")
      {
      $X = $_POST['X']; //get X and Y inputs
      $Y = $_POST['Y'];
      if (strlen($X) == 1) $X = ("0" . $X); //make sure the number has
2 digits
      if (strlen($X) > 2)
            {
            $length = strlen($X);
            $X = $X($length-2) . $X($length-1);
            }

      if (strlen($Y) == 1) $Y = ("0" . $Y);
      if (strlen($Y) > 2)
            {
            $length = strlen($Y);
            $Y = $Y($length-2) . $Y($length-1); // /keep digits
            }

      $directions = '0000 ';
      $coords = $X . ' ' . $Y . ' ';
  $comments = '(UP, DOWN, LEFT, RIGHT | X,Y COORD)';
      $writethis = $directions . $coords . $comments . $time;
  }
else
      {
      echo "no action data";
      }


fwrite($file, $writethis);
fclose($file);


header("location: nxtControl.php");

?>
</body>
</html>
```