Introduction

Summary

Before starting with actual labs, this will get you acquainted with the Lego Mindstorms NXT 2.0 kit and the RWTH Mindstorms NXT Toolbox for Matlab.

A. Required equipment
B. Matlab
C. RWTH Toolbox
D. Mindstorms NXT 2.0 kit

Required equipment

1) Lego Mindstorms NXT 2.0 kit
2) Matlab 7.7 (R2008b) or higher (32-bit)
3) RWTH NXT Toolbox for Matlab

Matlab

If Matlab is not already installed on your computer, you can take it to CCIT and have them install it for you. **Be sure to ask for a 32-bit version**. The RWTH Toolbox is not compatible with 64-bit Matlab. For Mac users, the most recent version you will be able to use is 2010a. For Windows users: 2010b. More recent versions are 64-bit only.

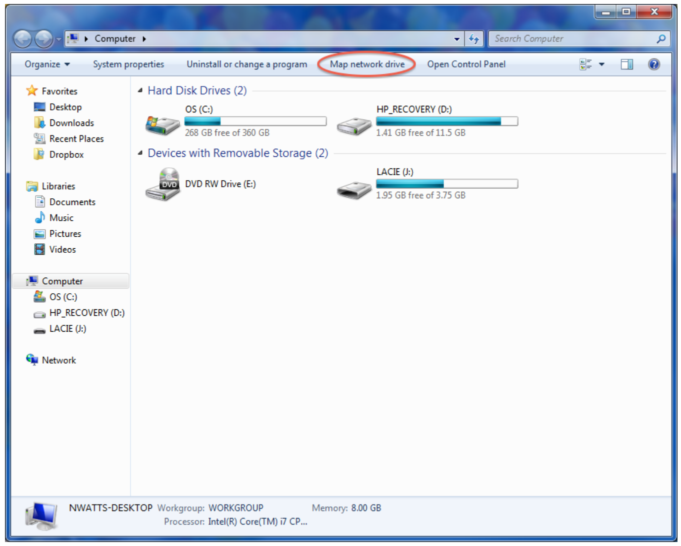
If you want to download and install Matlab yourself, follow the instructions below.

Download & Installation

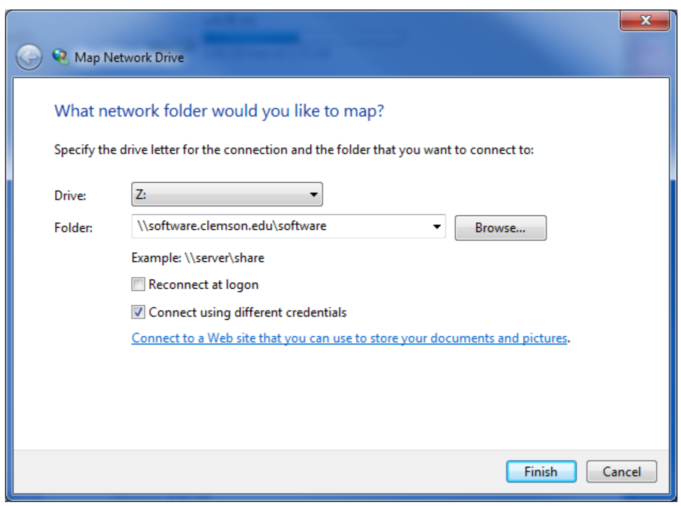
Matlab is available for download on the Clemson University software server. Before you start, make sure your computer is connected to an Ethernet cable, as the files you will need to download are about 2 gigabytes in size.

On Windows:

1) Connect to the software server.
    a. Open “My Computer.”
    b. Click “Map Network Drive.”



    c. At the prompt, select a drive letter and enter “\\software.clemson.edu\software” for the folder. Check the box for “Connect using different credentials.



    d. A window will open asking for a username and password. Enter your Clemson University username and password. A window will open showing the contents of the software server.

2) Go to the Software\Matlab folder.
3) Go to Documentation folder and copy “file installation keys.txt” and “license.dat” to your desktop.

4) Go back up one folder and go to the Win\2010b folder and run "CU_Install_32bit.vbs."
    a. Select "Install without using the Internet."
    b. Accept the license agreement.
    c. When asked for a file installation key, use the key for R2010b found in "file installation keys.txt."
    d. When asked for the installation type, select "Typical."
    e. Use the default installation folder.
    f. When asked for a license file, click the "Browse" button and select the license.dat file you downloaded in step 3.
    g. Click "Install" and wait for the installation to finish. It should take 10-20 minutes.

On Mac OS X:

1) Connect to the software server.
    a. From Finder, click Go->Connect to Server… in the menu bar.
    b. In the server address, enter smb://user@software.clemson.edu/software, where user is your Clemson University username.

    c. In the box that pops up, make sure the "Registered User" button is selected and enter your Clemson University password. Click "Connect."
    d. From the Finder, click Go->Computer or Go->Network in the menu bar.
    e. In Computer, you should see a disk icon labeled "software." Or in Network, you should see a computer icon labeled "software.clemson.edu." Depending on your Finder settings, the "software" disc icon may also show up on your desktop. Click on the icon to access the software server.
2) Go to the Software/Matlab/Mac/2010a folder. All versions newer than 2010a are 64 bit only and thus are incompatible with the RWTH toolbox.
3) Copy "Matlab2010a_for_Mac.dmg" to your desktop and open it.
4) Run "InstallForMacOSX.app."
    a. Choose "Install manually without using the Internet."
    b. Accept the terms of the license agreement.
    c. Check the box for "Intel" and uncheck "Intel 64."
    d. The file installation key is located in another file on the software server. Go back up to the Matlab folder and then open the Documentation

folder. Open “file_installation_keys.txt” and copy the key for 2010a into the dialog box.

e. At the product selection window, check which toolboxes you want to install. The only one you need for this class is “MATLAB.”
f. From the Documentation folder, copy the file “license.dat” to your desktop. Then in the dialog box, click the “Browse” button and select license.dat.
g. Wait for the installation to finish. It will probably take 10-20 minutes.

RWTH Toolbox

The RWTH Mindstorms NXT Toolbox for Matlab allows you to remotely control a Mindstorms NXT robot using Matlab. This lets you to create programs that take advantage of the advanced math functions and visualizations offered by Matlab along with all the available processing power of your computer.

Note that all the actual processing is done on your computer, not on the NXT itself. Messages are passed back and forth over USB to request sensor readings or change motor settings. Thus, your computer must remain connected to the NXT at all times.

Download & Installation

The RWTH Toolbox is available for download at http://www.mindstorms.rwth-aachen.de/trac/wiki/Download. Download the most recent stable release. Extract the downloaded ZIP file. This should create a folder called “RWTHMindstormsNXT.” Move the RWTHMindstormsNXT folder to wherever you want to keep it. Do not move anything around inside that folder.

Set Path

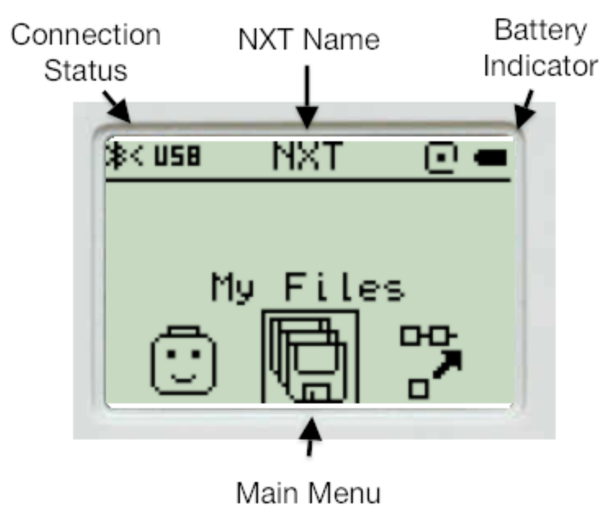
1) In Matlab, go to the “File” menu and click “Set Path.”
2) Click the “Add with Subfolders” button.
3) Select the RWTHMindstormsNXT folder.
4) Click “Save” to save settings for future sessions.
5) At the Matlab prompt, type “ver” and press enter. This will display a list of all the toolboxes you have installed. You should see “RWTH – Mindstorms NXT Toolbox” somewhere in the list.

Mindstorms NXT 2.0 kit

NXT Brick

The NXT brick is the interface between Matlab programs running on your computers and the robot. It can be connected to your computer via USB or Bluetooth and messages are passed back and forth between it and Matlab to request sensor readings and control motors.

The NXT brick has four input ports for sensors, labeled 1-4. It also has three output ports for motors, labeled A-C.



Sensors
The Mindstorms NXT 2.0 kit comes with several different types of sensors.

Touch Sensor

The touch sensor allows the robot to sense when it is touching objects. It reads true or false indicating whether or not it is being pressed.

The touch sensor can be used to detect if the robot has bumped into an obstacle or it can be used to have the robot perform an action on command.

Color Sensor

The color sensor can be used to measure a limited range of colors. It can distinguish between black, white, red, yellow, green, and blue. It can also be used as a light sensor to measure the brightness of ambient light.

The color sensor could be used to sort Lego bricks, make a robot follow a line, or have the robot respond to different colored lights.

Light Sensor

The light sensor is used to measure the brightness of ambient light. It is more precise in doing this than the color sensor.

The light sensor could be used to make a robot follow a line or have the robot find a lighted area.

Ultrasonic Sensor

The ultrasonic sensor can be used to measure the distance to objects. It can be operated continuously or set to take snapshots when told to do so.

The ultrasonic sensor could be used to have the robot avoid objects while moving around or to have the robot react to movement.

Motors

The servo motors give the robot the ability to move and interact with objects. The motors can be programmed to run for a certain amount of time. You can also set them to rotate a certain number of degrees, giving you more precise control.

Also, the motors can be used as rotation sensors to measure how far they have been turned.

Lab 1: Connecting to the NXT

Summary

This lab will explain how to connect to your computer to the Lego Mindstorms NXT 2.0 brick using the RWTH Mindstorms NXT Toolbox for Matlab. It will cover connecting via USB cable and Bluetooth. It will also explain how to work with the “handles” created in the connection process. The steps for connecting are dependent on your operating system.

A. Transfer motor control program
B. USB
C. NXT handles

Required equipment

1) Lego Mindstorms NXT 2.0 kit
2) Matlab 7.7 (R2008b) or higher (32-bit)
3) RWTH NXT Toolbox for Matlab

Part A: Transfer motor control program

In order to control the motors attached to the NXT brick, RWTH Mindstorms Toolbox communicates with a program than runs on the NXT. In order to use the RWTH Toolbox, this program has to first be loaded on to the NXT. Follow the instructions for your operating system.

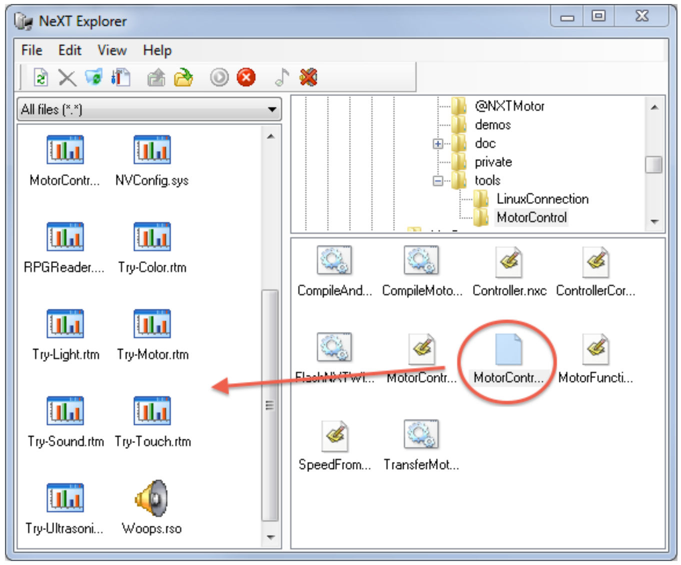
Steps for Windows 7:

Download driver

1. Go to http://mindstorms.lego.com/support/updates#Driver
2. Download the “Fantom” USB driver. (Click the blue “Fantom Driver” button, then click the “Downloads” tab, and finally click the “PC” button) Do not download “Driver 1.02”.
3. Extract the downloaded zip file and run setup.exe.
4. Follow the on screen instructions to complete the installation.

Transfer MotorControl.rxe with NeXT Tools

5. Download NeXTExplorer from http://bricxcc.sourceforge.net/utilities.html.
6. Extract the downloaded zip file.
7. Connect the NXT with the USB cable and turn it on.
8. Run NeXTExplorer.exe. When prompted for a port, select “usb.”
9. If MotorControl22.rxe is not already in the list of files on the NXT (shown in the left panel), navigate to the RWTHMindstormsNXT/tools/MotorControl folder (in the right panel) and drag MotorControl22.rxe from the right panel to the left.



10. MotorControl22.rxe should now appear in the list of files on the NXT
11. Close NeXTExplorer.

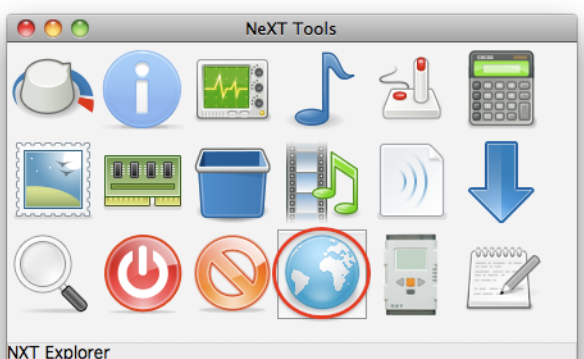
Steps for Mac OS X:

Download Fantom driver

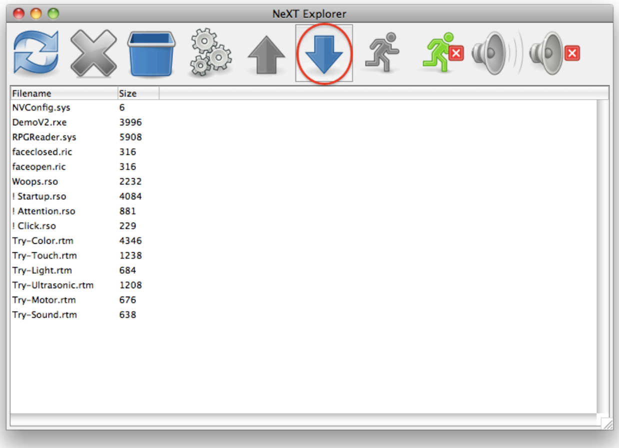
1. Go to http://mindstorms.lego.com/support/updates#Driver
2. Download the “Fantom” USB driver. (Click the blue “Fantom Driver” button, then click the “Downloads” tab, and finally click the “MAC” button) Do not download “Driver 1.02”.
3. Extract the downloaded .zip file and run legodriver.pkg.
4. Follow the on screen instructions to complete the installation.

Transfer MotorControl.rxe with NeXT Tools

5. Download NeXT Tools for Mac from http://bricxcc.sourceforge.net/utilities.html.
6. Extract the downloaded zip file.
7. Connect the NXT with the USB cable and turn it on.
8. Run NeXT Tools.app. When prompted for a port, select “usb.”
9. At the NeXT Tools main menu, select NXT Explorer.



10. If MotorControl22.rxe is not already in the list of files on the NXT, select "Download selected files to the NXT."



11. Select MotorControl22.rxe from the RWTHMindstormsNXT/tools/MotorControl folder.
12. MotorControl22.rxe should now appear in the list of files shown in the NXT Explorer window.
13. Close the NXT Explorer window, click Close Communication in the NeXT Tools main menu, and close NeXT Tools.

Modify files

The RWTH Mindstorms Toolbox code doesn't work out of the box for USB connections on Mac OS X, so you will have to alter some of the files. **Windows users skip this step**.

1. Download "Mac USB Files" from the class website.
2. Open the downloaded ZIP file and copy COM_OpenNXTEx.m, COM_SendPacket.m, and COM_CloseNXT.m to your RWTHMindstormsNXT folder, replacing the files that were already there.

Part B: Connecting via USB

Connect to NXT

1. Connect the NXT brick to your computer with the USB cable.
2. In Matlab, run h = COM_OpenNXT(). It should complete without an error. It will also start the MotorControl.rxe program on the NXT.
3. In Matlab, run NXT_PlayTone(440,100,h). The NXT should play a short tone.
4. After you are done, run COM_CloseNXT(h) in Matlab to close the connection.

Part C: NXT handles

The COM_OpenNXT function returns a "handle," or reference, to the created connection, whether USB or Bluetooth. The functions in the RWTH toolbox need the information contained in this handle to know where to send requests for sensor data or changes to motor settings.

For example, in step 15 of the USB connection instructions and step 17 of the Bluetooth connection instructions, you passed the handle "h" to the NXT_PlayTone function. (If no variable is specified to store the return value of COM_OpenNXT, by default the created handle is stored in the "ans" variable.)

To make things easier though, you can use the COM_SetDefaultNXT function. If no handle is passed to the RWTH toolbox functions, they will attempt to use the default handle, provided it has been set.

Thus, the Matlab program

```
h = COM_OpenNXT()
NXT_PlayTone( 440, 100, h )
```

can instead be written as

```
COM_SetDefaultNXT(COM_OpenNXT())
NXT_PlayTone( 440, 100 )
```

Note that the second call to NXT_PlayTone does not include a handle. It will use the default handle.

Unless you need to communicate with more than one NXT from the program, it is easiest to set the default handle and not worry about passing handles to functions.

Thus, most of your Matlab programs should follow this template:

```
COM_SetDefaultNXT(COM_OpenNXT())
…
… % Code
…
COM_CloseNXT(COM_GetDefaultNXT())
```

Note that for this lab, all the Matlab commands could be individually entered one by one at the prompt. However, the labs to come will require that commands be executed immediately one after the other. Thus, you will need to create a separate .m file and enter your Matlab commands in it. Then, you can run the entire file by clicking the "Run" button in the "Debug" menu of the Matlab editor.

Lab 2: Motors

Summary

This lab is an introduction to the use of the Lego Mindstorms NXT motors. The first part of the lab will introduce the NXTMotor class and demonstrate how to use it to move a robot. The final part of the lab will be to use what you have learned to move a robot along a specific path.

A. Moving the robot
B. Monitoring rotational sensors
C. Direct motor commands
D. Challenge: Create your own program to move the robot in a figure eight

Required equipment

1) Lego Mindstorms NXT 2.0 kit
2) Matlab 7.7 (R2008b) or higher (32-bit)
3) RWTH NXT Toolbox for Matlab

Introduction (NXTMotor)

The NXTMotor class will allow you to easily control the NXT's motors. It allows you to set, among other things, the motor's power, direction, and how far the motor will rotate. The argument passed to the NXTMotor function is the port of the motor to be controlled. The NXT brick has three output ports, labeled A, B, and C. In programs, these ports are referred to with MOTOR_A, MOTOR_B, and MOTOR_C.

To move one motor, run the following commands in Matlab. Keep in mind that you must be connected to the NXT and have the default handle set for this to work. (See lab 1 for connection instructions)

Create an NXTMotor instance for the motor on port B.

```
m = NXTMotor( MOTOR_B )
```

Set the motor's power. This is a percentage of the motor's total power and must be an integer between -100 and 100. Negative numbers run the motor in reverse.

```
m.Power = 50
```

Turning speed regulation on will allow the motor to automatically increase its power to reach a constant turning speed.

```
m.SpeedRegulation = true
```

Turning on smooth start causes the motor to smoothly accelerate from a stop.

```
m.SmoothStart = true
```

The tachometer limit defines how many degrees the motor will rotate before stopping. Setting this to 0 will cause the motor to run indefinitely.

```
m.TachoLimit = 360
```

The action at tachometer limit defines how the motor will act when it reaches its set tachometer limit. The options are ‘Coast,’ ‘Brake,’ and ‘Holdbrake.’ Coast allows the motor to rotate until it is stopped by friction. Brake applies the brake to stop the motor precisely at the tachometer limit. Holdbrake applies the brake and keeps it on to hold the motor at its position.

```
m.ActionAtTachoLimit = ‘Coast’
```

The previous commands set up an instance of the NXTMotor class on your computer. However, to actually have the motor move as specified, you must transfer the instance to the NXT. This is done by running the SendToNXT command.

```
m.SendToNXT()
```

As soon as the SendToNXT command is executing, the motor will start moving. At any point, it can be stopped by running the Stop command, which will cause the motor to stop and coast.

```
m.Stop(‘off’)
```

Also, the stop command can be used to brake the motor. Running Stop(‘brake’) holds the brake after the motor is stopped, which uses a lot of battery, so it’s a good idea to run Stop(‘off’) afterward to release the brake.

```
m.Stop(‘brake’)
m.Stop(‘off’)
```

Keep in mind that Matlab commands following SendToNXT will be executed while the motor is still in motion. This may not be desired. For example, if a robot was intended to move forwards and then backwards. Assume m_forward and m_backward are instances of NXTMotor which have already been set up appropriately.

```
m_forward.SendToNXT()
m_backward.SendToNXT()
```

Would not produce the desired behavior. The motor would start to move forward, but then would immediately start moving backward. To make sure the motor runs forward as long as intended, use the code

```
m_forward.SendToNXT()
m_forward.WaitFor()          % Pauses until the motor finishes with m_forward
m_backward.SendToNXT()
m_backward.WaitFor()
```

It is convenient to think of instances of the NXTMotor class as states of the motor or motors that they control. For example, if you were trying to program a robot that moved in a square pattern, you might create one instance of NXTMotor for the straight-line movement and another instance for the turning. Thus, you would only need to set up NXTMotor instances for two states even though there would be eight changes in motor settings. After the NXTMotor instances were set up, controlling the robot's movement would just be a series of SendToNXT statements.

Thus, by breaking down the motor movements required by the project and preparing some NXTMotor instances in advance, the code required to control a robot's movement becomes fairly simple.

Part A: Moving the robot

A common setup for a moving robot is to have one motor drive the left side wheels or track and another motor drive the right side. If this is the case, in order to move the robot in a straight line, both motors must run in sync. Fortunately, the NXTMotor class makes this easy. Create a new .m file and add the following code.

```
forward = NXTMotor( [ MOTOR_B; MOTOR_C ] )
forward.Power = 50
forward.SendToNXT()
```

Creating an NXTMotor instance with an array of two motors instead of a single one will automatically synchronize the two motors. Note that speed regulation cannot be used with motors that are synchronized this way. Run your .m file and see what happens. Keep in mind that you must be connected to the NXT and have the default handle set for this to work. (See lab 1 for connection instructions)

To move the robot backward, simply reverse the sign of the power.

Try programming the robot to move forward, stop, and move backwards to its original position.

Keep in mind that the MotorControl.rxe program that was transferred to the NXT in lab 1 has to be running for any NXTMotor commands to work. So if your robot isn't doing anything, check to make sure MotorControl.rxe is running.

To turn the robot in place, all you have to do is run one motor in reverse of the other for the same duration. For example, to turn left

```
left_turn_l = NXTMotor( MOTOR_B )        % NXTMotor for left motor
left_turn_l.Power = -50
left_turn_r = NXTMotor( MOTOR_C )        % NXTMotor for right motor
left_turn_r.Power = 50                   % Power is reversed from left motor
left_turn_l.SendToNXT()
left_turn_r.SendToNXT()
```

Try programming to robot to turn 180 degrees in place.

There are three options for stopping a motor. You can use NXTMotor:

```
m = NXTMotor( MOTOR_A );
m.Stop();
```

Also, there is a StopMotor function which does not require an NXTMotor instance.

```
StopMotor( MOTOR_A, ‘off’ );
```

This is equivalent to the first method.

Third, there is the orange button on the NXT. When MotorControl.rxe is running, this acts an “emergency stop.” Pressing it will stop all motors. It will also end the MotorControl.rxe program, so you will have to restart it after pressing the button.

Part B: Monitoring rotational sensors

The NXT’s motors have rotational sensors in them that keep track of how far the motors have turned. They monitor the motor’s rotation whether it’s caused by a command or you turning it by hand. To access the motor’s current position, use this code.

```
s = NXT_GetOutputState( MOTOR_B )
s.TachoCount
```

The NXT_GetOutputState function will return a structure containing several fields with information about motor settings. The TachoCount field contains the information about how far the motor has rotated.

Try turning the motor by hand and rerunning this code.

Part C: Direct motor commands

The NXTMotor class provides precise motor control and reusable motor states. However, to get that precision, it sacrifices speed. For situations where very quick motor response is necessary, direct motor commands may prove more useful.

The syntax for the DirectMotorCommand function is as follows:

```
DirectMotorCommand( port, power, tachometerLimit, speedRegulation,
  syncToMotor, turnRatio, smoothStart )
```

These parameters are the same as they are for the NXTMotor class. The syncToMotor parameter serves the same purpose as creating an NXTMotor instance with two motors. For example:

```
DirectMotorCommand( MOTOR_B, 50, 0, ‘off’, MOTOR_C, 0, ‘off’ )
```

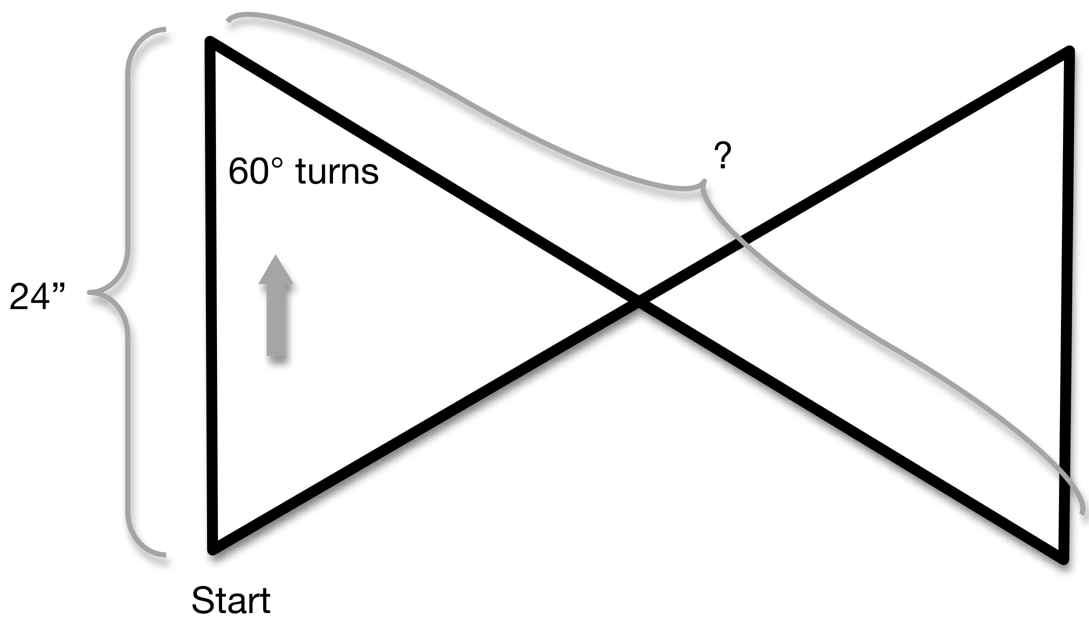
is equivalent to

```
m = NXTMotor( [ MOTOR_B, MOTOR_C ] )
m.SendToNXT()
```

In most cases though, it is recommended that you use the NXTMotor class. It is both easier to use and provides more precise motor control.

Part D: Figure eight

This part of the lab will have you use what you've learned to move the robot in a figure eight pattern as shown below.



1) The robot must return as close as possible to its original position and orientation.
2) Once the program is started, nothing is allowed to touch the robot until it is finished.
3) Each group will have 2 trials to demonstrate their program. Each trial will consist of 3 runs. In each run, the robot will be started from a marked location and its final position and orientation will be compared with its starting position and orientation. The data from the first trial can be used to look for errors in the program, and the program can be changed before the second trial.

Lab 3: Touch Sensor

Summary
This lab will introduce you to using Mindstorms sensors with the RWTH Toolbox, using the touch sensor as an example. The touch sensor is the simplest of the sensors included in the Mindstorms kit. It simply returns a value true or false indicating whether it is being pressed or not. This lab will cover how to use the touch sensor in a program,

A. Understand how to use the touch sensor
B. Create a program to use the touch sensor as a safety switch to keep the robot from running into objects.
C. Challenge: Create your own program to control the robot with touch sensors

Required equipment
1) Lego Mindstorms NXT 2.0 kit
2) Matlab 7.7 (R2008b) or higher (32-bit)
3) RWTH NXT Toolbox for Matlab

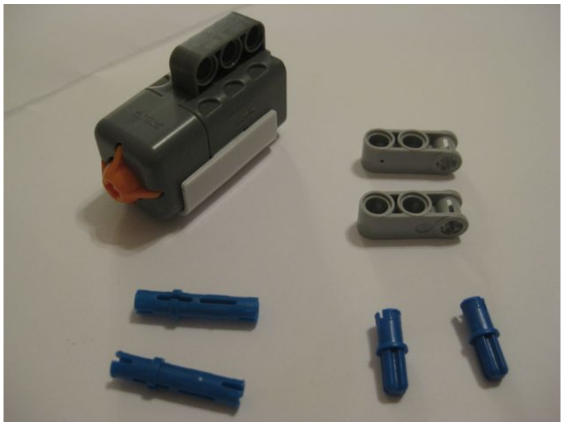
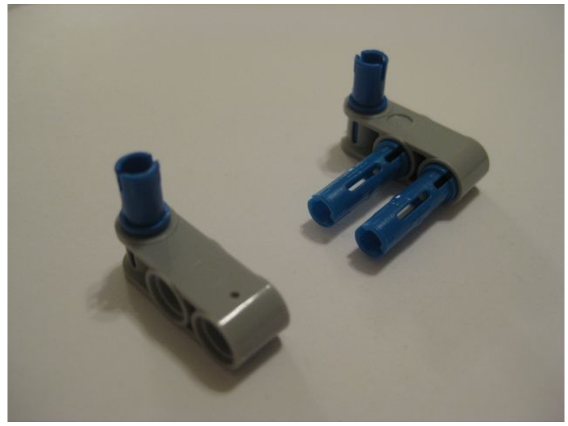
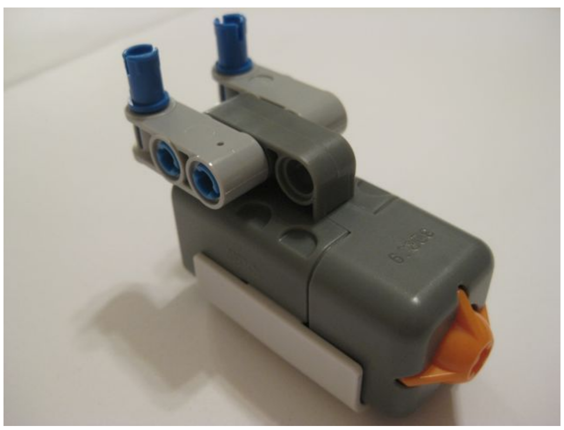
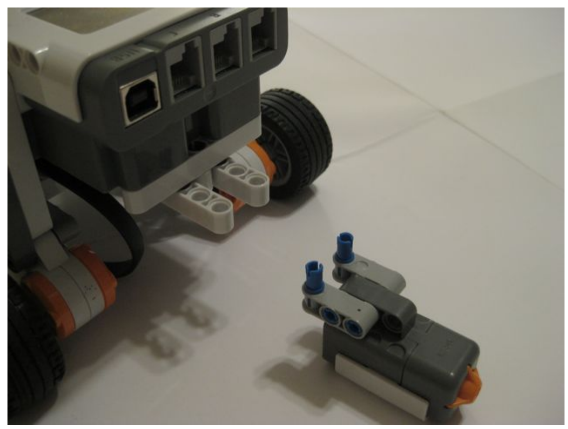
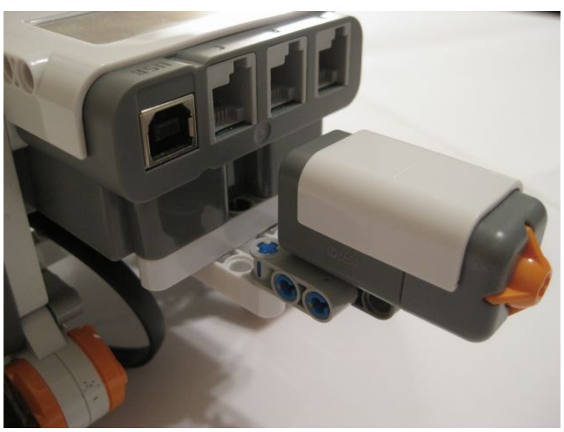
Part A: Using the touch sensor
The touch is very easy to use. It will return a 1 or 0 indicating whether it is being pressed or not. To use the touch sensor, run the following commands:

```
OpenSwitch( SENSOR_1 )      % Tells the NXT to interpret the sensor on port 1
                            % as a touch sensor
GetSwitch( SENSOR_1 )       % Gets a reading from the sensor, either 0 or 1
CloseSensor( SENSOR_1 )     % Closes the sensor after you are done with it
```

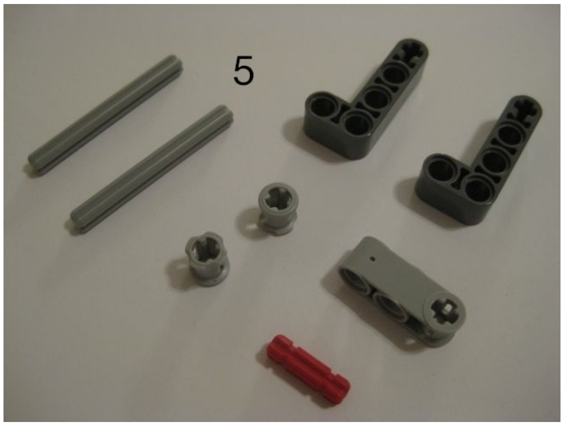
This code assumes that the touch sensor is connected to port 1. The NXT brick has four sensor input ports, labeled 1 through 4. In programs, they are referred to with the constants SENSOR_1, SENSOR_2, SENSOR_3, and SENSOR_4.

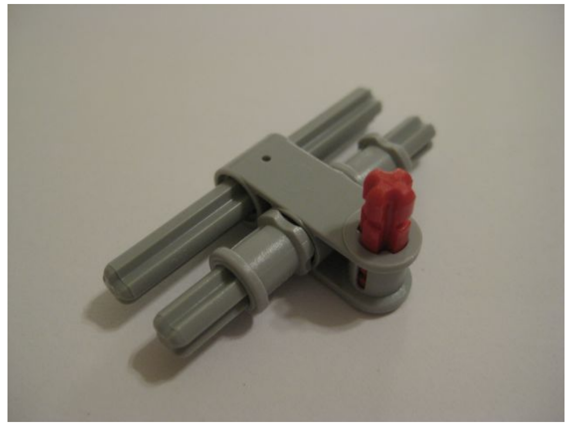
Part B: Safety switch
One practical use of the touch sensor would be to mount it on the front of a moving robot and use it as a safety switch by having the robot stop if the sensor is pressed. This would keep the robot from damaging itself by running into objects.

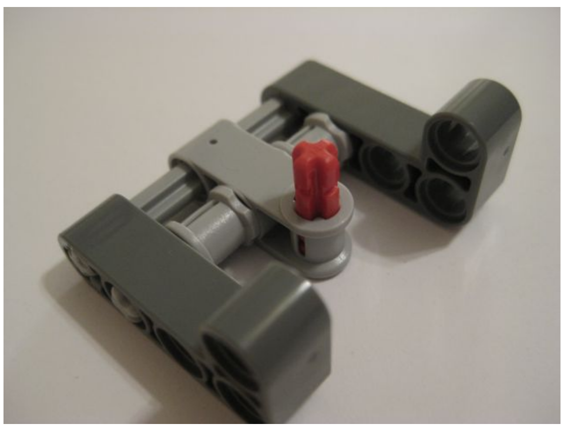
Building Instructions
Mount the touch sensor on the front of the robot. (See the lab manual introduction for instructions on how to build the base robot)

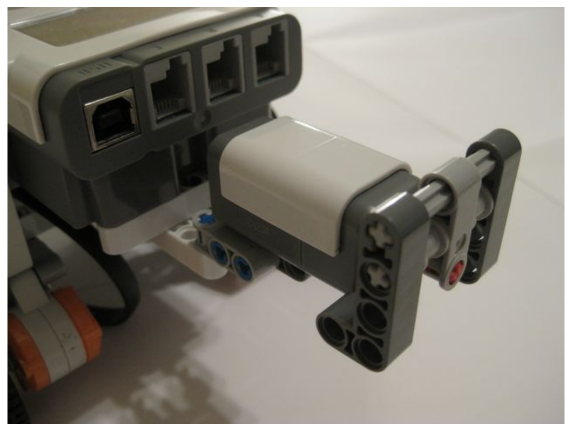






Assemble bumper and attach it to the touch sensor.









Connect touch sensor to input port 1.

Coding Instructions

Open a new script in Matlab and add the following code. First, open the touch sensors.

```
OpenSwitch( SENSOR_1 )
```

Set up the NXTMotor instance to move the robot. It will just move forward until the sensor is pressed.

```
m = NXTMotor( [MOTOR_B; MOTOR_C] );
m.Power = 20;
m.SendToNXT()
```

Continuously read the sensor value. Stop the motors if the sensor is pressed.

```
while true
   if GetSwitch( SENSOR_1 )
      m.Stop('off')
      break
   end
   pause( 0.1 )
end
```

Finally, close the sensor.

```
CloseSensor( SENSOR_1 )
```

Run your .m file. Keep in mind that you must be connected to the NXT and have the default handle set for this to work. (See lab 1 for connection instructions)

Note that this loop repeats infinitely. It tests to see if the condition is true. But the condition is defined as true, so that test will never fail. This is where the break statement comes in. When the touch sensor is pressed, the break statement will cause Matlab to "break out" of the loop.

Another way to exit an infinite loop is by pressing Ctrl+C. Sometimes though, this causes problems with the connection to the NXT, particularly when using Bluetooth. So if your program starts returning errors about sending or receiving packets, try closing and reopening the connection to the NXT. The following commands will do the trick, assuming you are using a USB connection and a default handle.

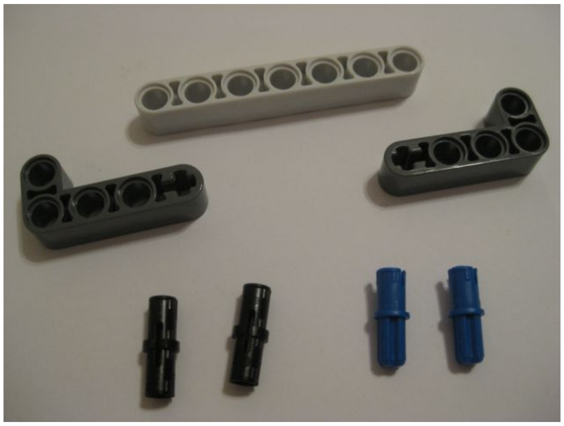
```
COM_CloseNXT( COM_GetDefaultNXT() );
COM_SetDefaultNXT( COM_OpenNXT() );
```

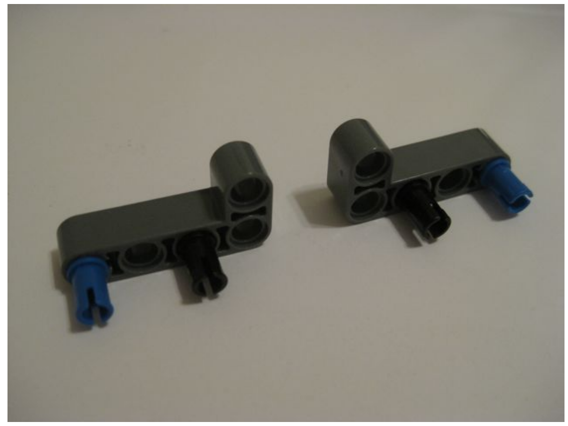
Part C: Control buttons

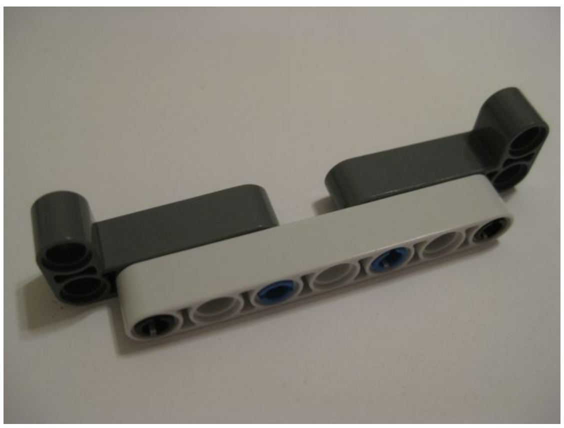
The touch sensors can also be used as buttons to control the robot. Create a program that causes the robot to have the following behavior:

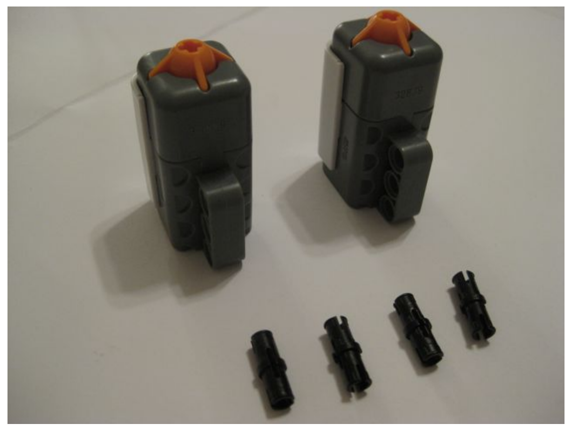
a) If the touch sensor on port 1 is pressed, the robot will turn left.
b) If the touch sensor on port 2 is pressed, the robot will turn right.
c) If both sensors are pressed, the robot will move forward in a straight line.

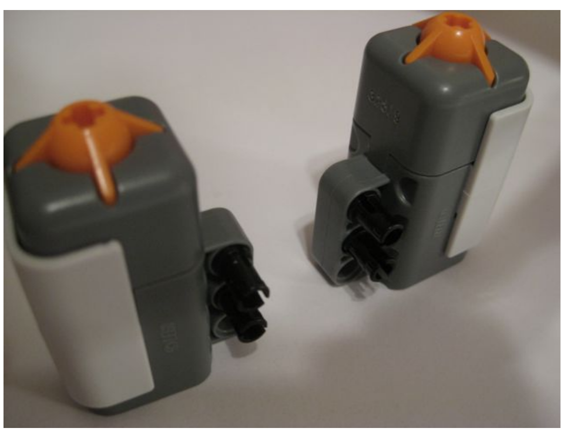
Building Instructions
Assemble controller.

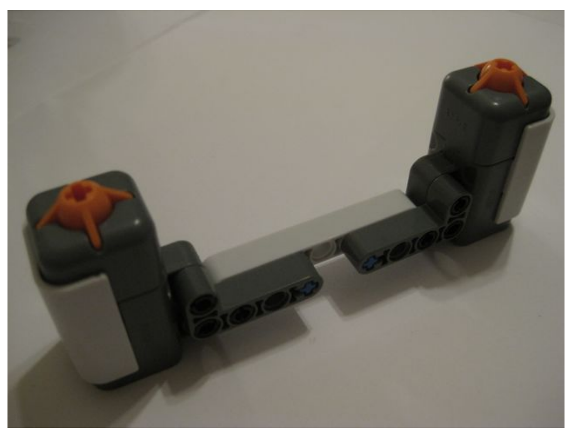












Connect the touch sensors to input ports 1 and 2.

Lab 4: Color & Light Sensors

Summary

In the first lab, the robot was programmed to drive around in a figure eight pattern. Specific motor commands were given to the robot to have it follow that path. However, if the path were to be changed, the entire program would have to be rewritten. Depending on how often the path changes, this can become quite a significant task.

On the other hand, if the robot were programmed to follow a line, the path could be changed as often as desired with no reprogramming required. This lab will demonstrate a simple feedback control system using the Lego Mindstorms NXT 2.0 color sensor.

A. Understand how to use the color and light sensors
B. Create a program to have the robot follow a line using digital sensor feedback
C. Improve the precision of the line follower by using analog sensor feedback
D. Challenge: Improve the performance of the line follower on curves

Required equipment

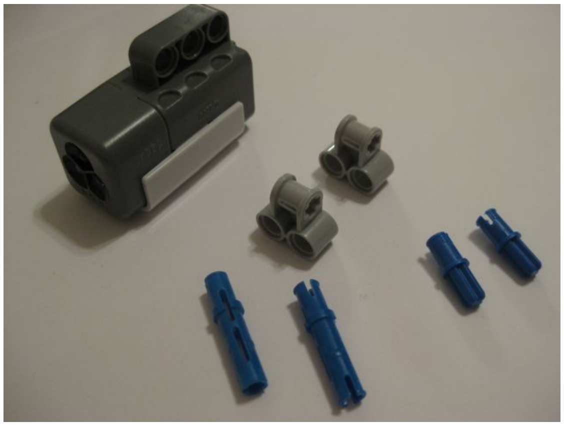
1) Lego Mindstorms NXT 2.0 kit
2) Matlab 7.7 (R2008b) or higher
3) RWTH NXT Toolbox for Matlab (version 4.04 or higher)
4) Test pad (paper included with Mindstorms kit)

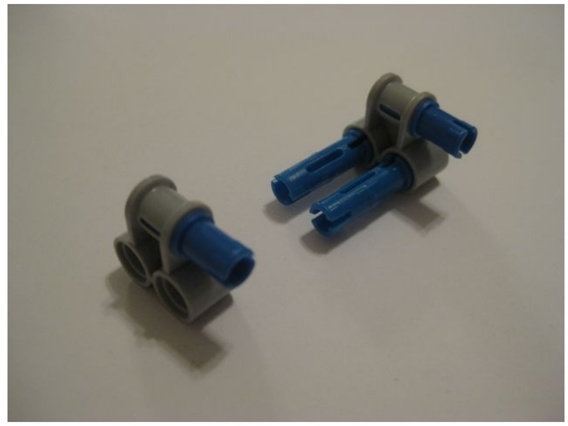
Part A: Using the color & light sensors

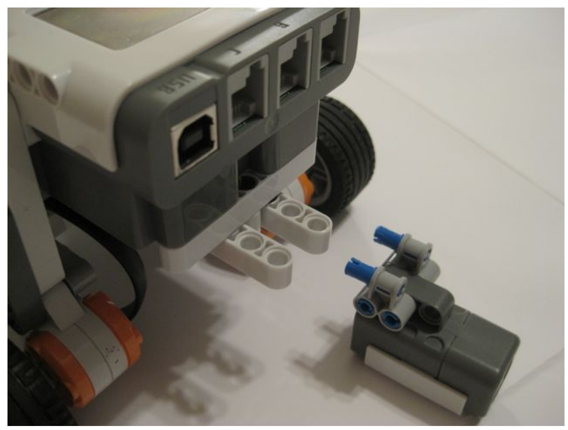
The color sensor can be used in two different ways. One, it can be used to detect color. Used that way, it can distinguish black, white, red, yellow, green, and blue. It can also be used as a light sensor to just sense brightness. In this mode, it returns a numeric value. Higher numbers correspond to brighter surfaces.

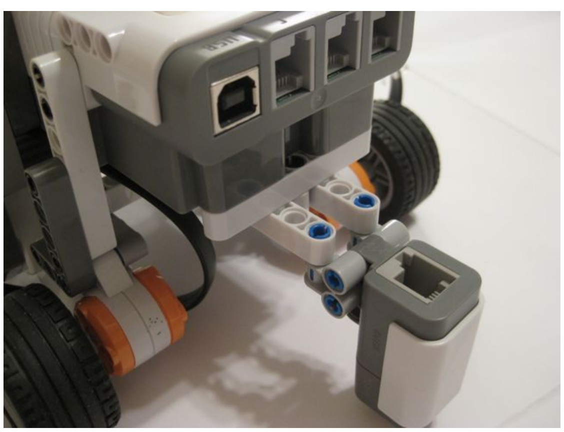
The light sensor operates basically the same as the color sensor when in light mode, but is generally more precise. Just as with the color sensor, it returns a numeric value, and higher readings correspond to brighter light.

Building Instructions
Mount the color sensor on the front of the robot. (See the lab manual introduction for instructions on how to build the base robot)









Connect the color sensor to input port 1.

Coding Instructions

To use the color sensor, use the following code.

```
OpenNXT2Color( SENSOR_1, 'FULL' )          % Puts the sensor in color mode
GetNXT2Color( SENSOR_1 )                   % Returns the color as a string
CloseSensor( SENSOR_1 )                    % After you are finished
```

To use the color sensor as a light sensor,

```
OpenNXT2Color( SENSOR_1, 'RED' )
GetNXT2Color( SENSOR_1 )                   % Returns number 0 – 1023
                                           % Higher numbers = brighter light
CloseSensor( SENSOR_1 )
```

To use the light sensor,

```
OpenLight( SENSOR_1, 'INACTIVE' )          % Changing 'INACTIVE' to 'ACTIVE'
                                              turns on the red LED.
GetLight( SENSOR_1 )                       % Returns number 0 – 1023
CloseSensor( SENSOR_1 )
```

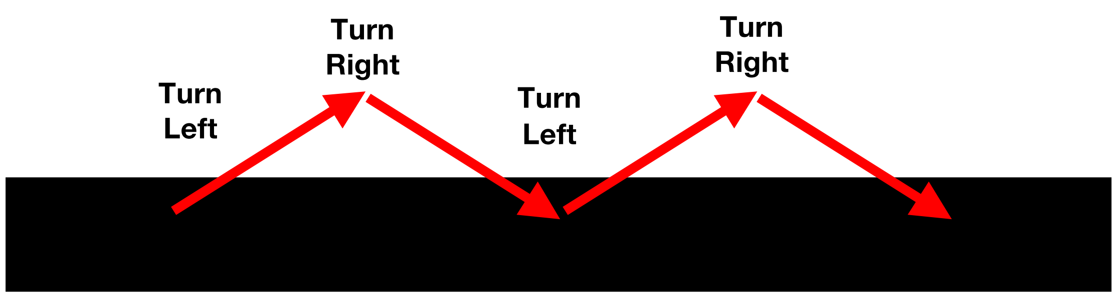
Questions to Consider

a) In color mode, the color sensor has no perception of grayscale. How light does a shade have to be for the sensor to register white? How dark for it to register black? (Use the gradient on the side of the test pad)

b) In light mode, the color sensor's reading is dependent on ambient light. What does the sensor read when scanning a white shade? A black shade?

Part B: Digital sensor feedback

In this part of the lab, we will create a sample control system to have the robot follow the black line on the test pad using the color sensor. A simple way to do this is to have the robot follow the edge of the line as shown below.



Following the edge of the line provides the necessary inputs for the color sensor to function as a control system. On the inside of the line, the color sensor would always read black and the robot could wander about. On the edges though, the white to black and black to white transitions can be used to keep the robot squarely on the line. When a transition from white to black is detected, the robot must have just crossed into the line. When a transition from black to white is detected, the robot must have just exited the line.

This position data can be used to have the robot keep itself on the line by having turning left when a white to black transition is detected and right when a black to white transition is detected. Notice that this will only work if the robot is on the left edge of the line.

Coding Instructions
Start by setting up NXTMotor instances for each wheel.

```
l_wheel = NXTMotor( MOTOR_B );
r_wheel = NXTMotor( MOTOR_C );

% Turn on smooth acceleration
l_wheel.SmoothStart = true;
r_wheel.SmoothStart = true;
```

Activate the color sensor in color mode. Pause to give it time to power up.

```
OpenNXT2Color( SENSOR_1, ‘FULL’ )
pause( 0.1 )
```

Start the right wheel.

```
r_wheel.Power = 20;
r_wheel.SendToNXT()
```

The rest of the program consists of repeating the follow sequence: start turning left, wait for the color sensor to read white, start turning right, wait for the color sensor to read black. With the right wheel running at a constant speed, turning left or right is as simple as slowing down or speeding up the left wheel.

Note that the difference in motor speeds is greater for right turns. This is because while running clockwise around the oval track, the robot will have to make much sharper right turns than left.

```
while true

   l_wheel.Power = 10;          % Slow to turn left
   l_wheel.SendToNXT()
   while ~strcmp( GetNXT2Color( SENSOR_1 ), 'WHITE' )      % Wait until sensor is
      pause( 0.025 )                                        % over white area
   end

   l_wheel.Power = 40;          % Speed up to turn right
   l_wheel.SendToNXT()
   while ~strcmp( GetNXT2Color( SENSOR_1 ), 'BLACK' )      % Wait until sensor is
      pause( 0.025 )                                        % over black line
   end

end
```

Note that the program is in an infinite loop, so you will have to stop the robot by pressing the orange “emergency stop” button.

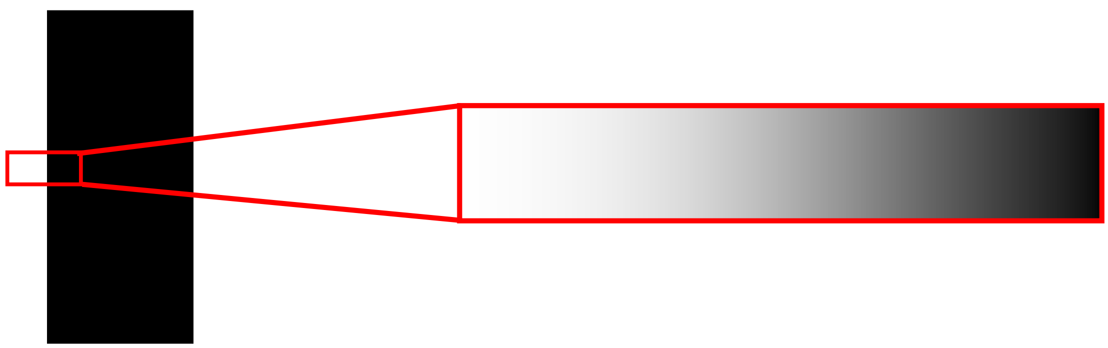
Place the robot on the black line pointed in the clockwise direction and run the program.

Questions to Consider

a) How well did the robot follow the line along the straight edges? Along the curves?
b) Turn the robot so that it is facing counterclockwise around the loop. How well does it follow the inside edge of the loop like this?

Part C: Analog sensor feedback

The precision of the line following program can be improved by using the color sensor as an analog sensor rather than a digital one. In light sensor mode, the color sensor will return the average light intensity over its scanned area. As the robot moves off the edge of the black line, the line will fill less of the scanned area and the value returned by the color sensor will increase. To the color sensor, the edge of the line looks like the following diagram:



If we target a value midway between black and white, the difference between the target value and the actual value can be used to control the speed and direction of the motors. Thus, the farther away the robot gets from the edge of the line, the sharper it will turn to correct itself.

Once again, start by setting NXTMotor instances for each wheel.

```
l_wheel = NXTMotor( MOTOR_B );
r_wheel = NXTMotor( MOTOR_C );
l_wheel.SmoothStart = true;
r_wheel.SmoothStart = true;
```

Open the color sensor in light mode. Pause to give it time to power up.

```
OpenNXT2Color( SENSOR_1, 'RED' )
pause( 0.1 )
```

Start the right wheel.

```
r_wheel.Power = 20;
r_wheel.SendToNXT()
```

The rest of the program consists of constantly adjusting the left wheel's speed to turn left or right based on the difference between the target sensor reading and the actual reading.

```
while true
   target = 450;                  % This value may need to be changed
   diff = target - GetNXT2Color( SENSOR_1 );
   l_wheel.Power = 20 - floor( diff );
   l_wheel.SendToNXT()
   pause( 0.01 )
end
```

The target value is set to halfway between the sensor readings for black and white. Since those depend on ambient lighting, you may have to change the target. Multiplying the difference in the target and actual sensor reading by 3/2 amplifies how sharp the robot turns to correct itself. Finally, the floor function is necessary because the result of the multiplication may have a decimal part and the NXTMotor Power field requires an integer. (Floor truncates a number's decimal part)

Here is a tip for setting the motor power. Power has to be an integer between -100 and 100. But if you set power as the result of an expression, such as "20 – diff" in the example here, that condition may not always be met. A simple way to ensure that power stays within the desired range is to use this code:

```
m.Power = max( -100, min( 100, floor( p ) ) );
```

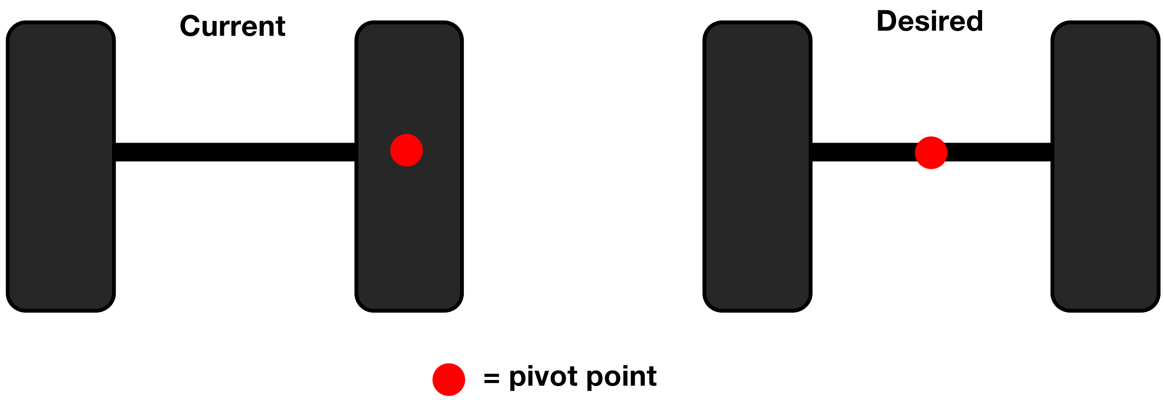
Where p is the expression to set the power. Floor will make sure that the power is an integer. If p is greater than 100, the min(100,p) will return 100, ensuring that the power is never set above 100. Likewise, if p is less than -100, the max(-100,p) will return -100, ensuring that the power is never set below -100.

Questions to Consider

a) Does the robot follow the line more closely now?
b) How does changing the multiplication factor affect the program's performance? Try both lower and higher multiplication factors.
c) Turn the robot so that it is facing counterclockwise around the loop. How well does it follow the inside edge of the loop like this?

Part E: Improved curve following

On the last question in part C, turning the robot so that it goes counterclockwise produces some strange turning behavior. Since the program as written only adjusts the left wheel's speed in order to turn, the robot's pivot point is under the right wheel.



For the pivot point to be in the center of the axle, both motors must take the light sensor data into account. Make the necessary changes to the program from part C for the robot to handle left or right curves equally well.

Lab 5: Ultrasonic Sensor

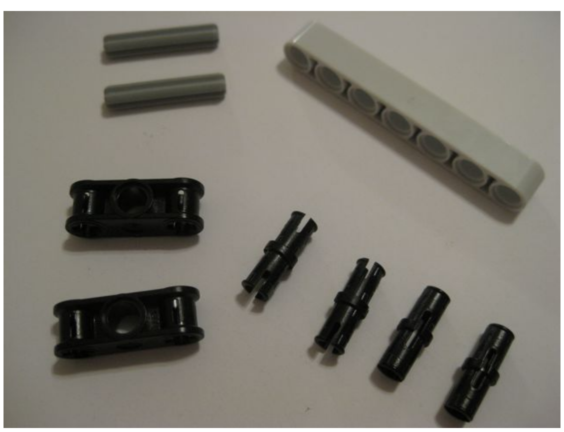
Summary
This lab will introduce you to the Mindstorms ultrasonic sensor. The ultrasonic sensor can be used to determine the distance from the robot to an object. First, the sensor's two modes of operation are explained. Next, an example of a simple feedback control system is shown. Finally, in the challenge section, you will be required to create your own control system to have the robot follow a wall.

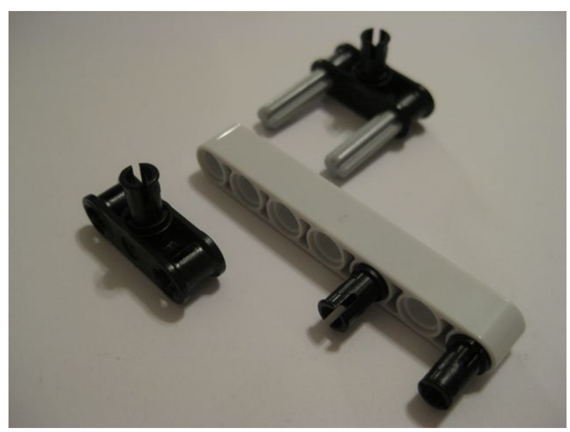
A. Understand how to use the ultrasonic sensor
B. Create a program to have the robot maintain a certain distance from a surface
C. Challenge: Create your own program to make the robot follow a wall

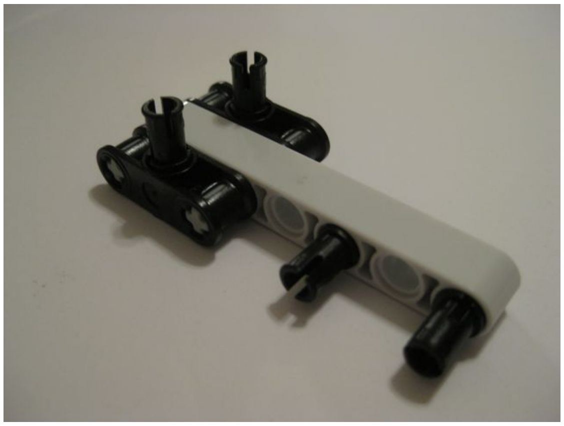
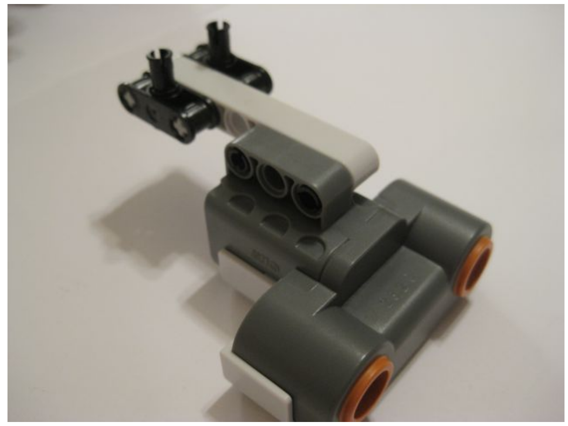
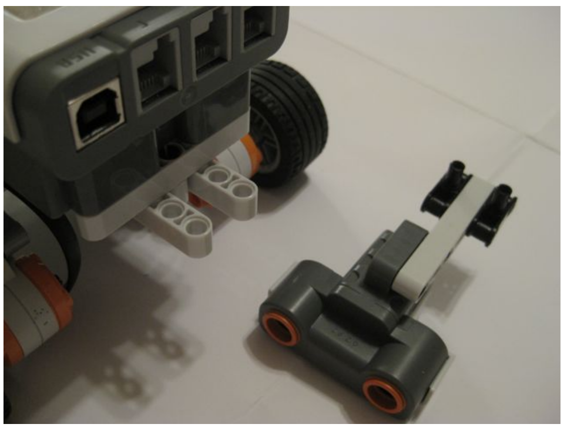
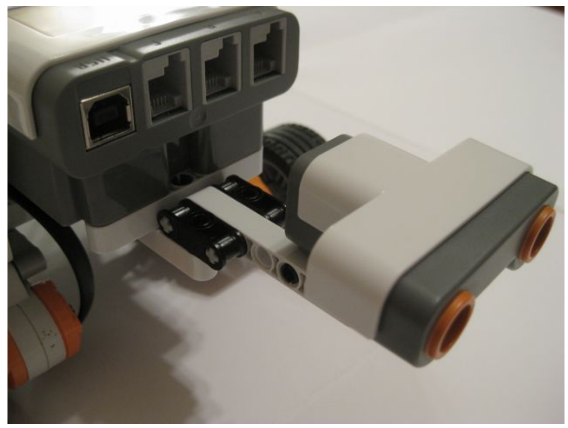
Required equipment
1) Lego Mindstorms NXT 2.0 kit
2) Matlab 7.7 (R2008b) or higher (32-bit)
3) RWTH NXT Toolbox for Matlab

Part A: Using the ultrasonic sensor

Building Instructions
Mount the ultrasonic sensor on the front of the robot. (See the lab manual introduction for instructions on how to build the base robot)





Connect the ultrasonic sensor to input port 1.

Coding Instructions

The easiest way to use the ultrasonic sensor is in continuous mode. In this mode, it will continuously send out pings and measure echoes. To use it this way, run the following commands:

```
OpenUltrasonic( SENSOR_1 )  % Turn on the sensor
pause( 0.1 )                % Allow time for the sensor to measure echoes
GetUltrasonic( SENSOR_1 )   % Get reading
CloseSensor( SENSOR_1 )     % After you are done
```

GetUltrasonic will return a value representing the distance in centimeters between the sensor and the surface in front of it. Keep in mind that due to imperfections in the surface and the resulting scattering of sound waves, this value may not be exactly accurate.

Alternatively, the ultrasonic sensor can be used in snapshot mode. In this mode, it will only ping when told to do so, and then will ping only once. This is useful when there are several ultrasonic sensors operating in the same area where they could interfere with each other if pinging continuously. To use the ultrasonic sensor in snapshot mode, run the following commands:

```
OpenUltrasonic( SENSOR_1, 'SNAPSHOT' )
```

```
USMakeSnapshot( SENSOR_1 )
pause( 0.1 )
echoes = USGetSnapshotResults( SENSOR_1 )
echoes(1)
```

The USMakeSnapshot function sends out a single ping and records up to eight echoes. The USGetSnapshotResults function returns the measured distances for those eight echoes. Usually, the first one is the most accurate.

In either mode, the distance returned will be -1 if there was some error getting a reading, or 255 if the surface was out of range or didn't reflect any sound.

Questions to Consider

a) How might you account for the sensor's imprecision in your programs?

Part B: Maintain distance

In this part of the lab, we will create another feedback control system to have the robot maintain its distance from a surface. The robot will move towards or away from the surface based on the reading from the ultrasonic sensor. As in lab 4, we will use the difference in the target and actual sensor readings to control the robot's movement speed.

Coding Instructions

First, create an NXTMotor instance to control the robot's movement. Since the robot only has to move forwards and backwards, we only need one instance.

```
m = NXTMotor( [MOTOR_B; MOTOR_C] );
m.SmoothStart = true;
```

Open the ultrasonic sensor and give it time to collect readings.

```
OpenUltrasonic( SENSOR_1 )
pause( 0.1 )
```

The rest of the program consists of continuously taking readings and adjusting the motor settings accordingly.

```
target = 15;
while true
d = GetUltrasonic( SENSOR_1 );
   m.Power = floor( (d – target) * 10)
   m.SendToNXT()
pause( 0.1 )
end
```

Try placing a notebook in front of the robot and moving it forwards and backwards.

Questions to Consider

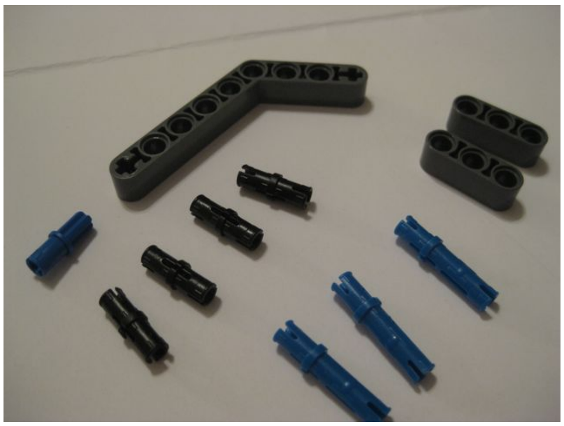
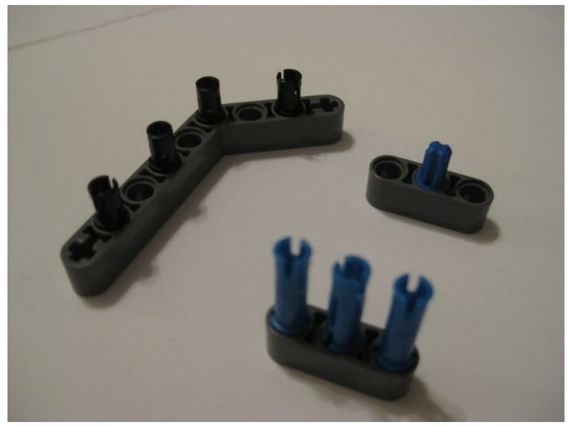
a) Try quickly moving the notebook 40 cm or more away from the robot. What happens? (Be ready to push the orange emergency stop button) What is the problem and how can it be solved?

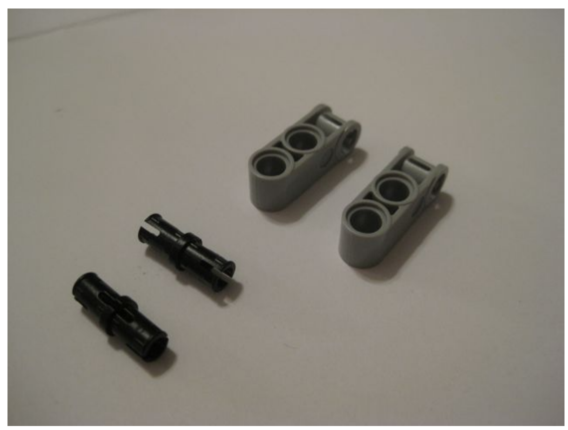
Part C: Wall follower

The ultrasonic sensor can be used as a feedback control system just as the color sensor was in lab 4. In this case, however, the robot can be made to follow a wall instead of a line. Create a program to have the robot follow a wall, staying a given distance away from it.

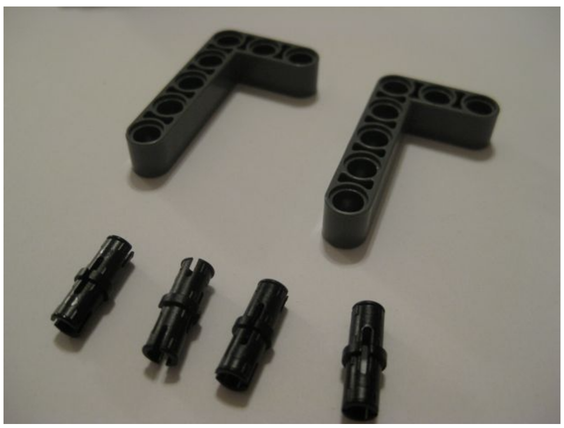
Building Instructions

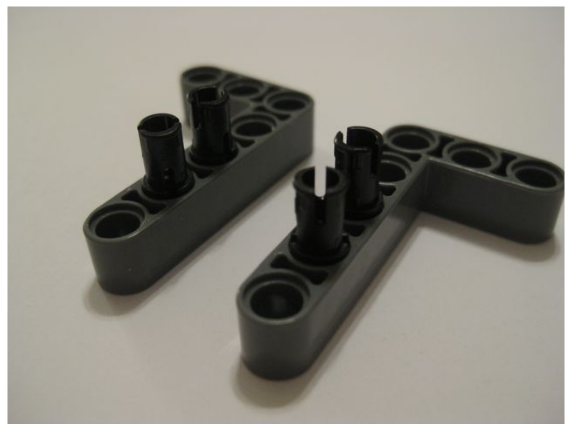
Build ultrasonic sensor mount

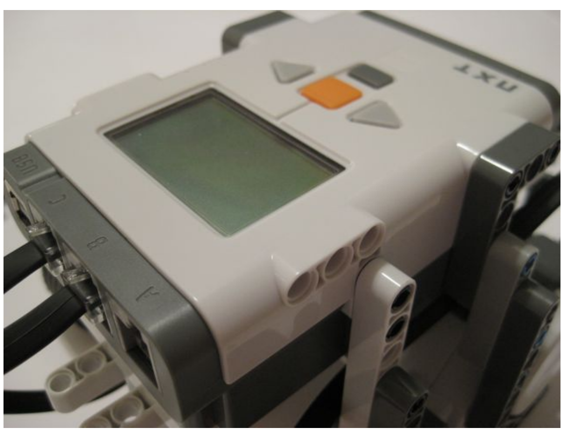





Attach brackets to NXT block







Attach sensor mount to NXT