Ch NXT Package User's Guide

Version 2.0.0

Integration Engineering Laboratory
University of California, Davis
Augest 31, 2012

Contents

1 Introduction

αCh Mindstorms NXT Control Package brings the inherent functionality of the Ch programming language to the intelligence and versatility found in the LEGO Mindstorms NXT robotic design system.

The Ch Mindstorms NXT Control Package consists of a set of API functions enabling programmers to write programs in C or C++ that can access and control the many features of the LEGO Mindstorms NXT controller. The API converts the complex messaging tasks required to communicate with the NXT into easy to use functions; allowing the user to focus their efforts on their robotic application, rather than the details of communication. The API of the Ch Mindstorms NXT Control Package was designed to support and augment all of the functionality found in the LEGO Mindstorms NXT controller. The Ch package further enhances the capabilities of the NXT controller by adding data collection and plotting capabilities. Additionally an NXT control program, written in C source code can be directly run from any platform in Ch without tedious compile/link/execute/debug cycles.

The communication between the user, the computer, the NXT controller, the sensors, and the motors can be described in Figure 1. Once NXT is connected to the computer and a NXT program has started, the program instructions are sent from the computer to the NXT. The NXT controller will process these instructions perform appropriate tasks by sending commands to the motors or receiving data from the sensors. The NXT can collect sensor data and motor encoder counts, and the data can be sent back to the computer for further manipulation, display, or stored in the computer for the user.

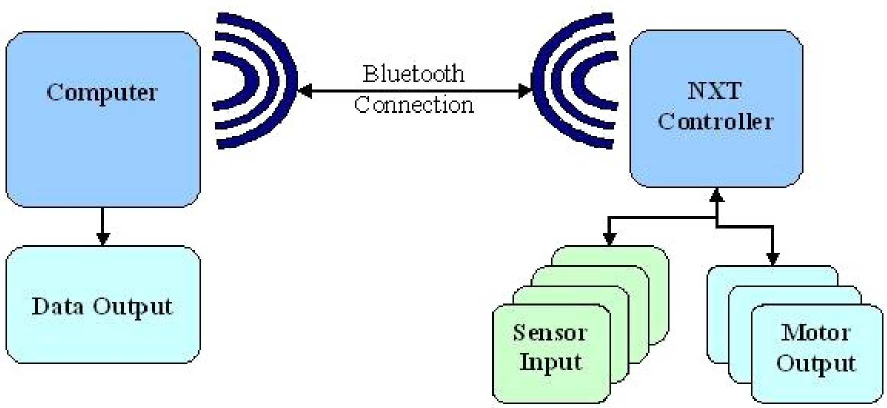

Figure 1: Communication Diagram of NXT

With Ch Mindstorms NXT Control Package, you can quickly develop an NXT robotic application and log your results. The ease of design and added functionality makes the Ch NXT Control Package a good candidate for any NXT programming application.

In this guide, we will go over the basics of a Ch Mindstorms NXT program. We will also discuss about how to control a NXT vehicle's motion. Lastly we will describe how to control non-vehicle NXT robots. After reading this guide, you will be ready to write your own Ch NXT program to control your NXT robot.

2 Configuring Lego Mindstorms NXT for Remote Control

Before using the Ch NXT package to control NXTs, we need to configure the NXTs' bluetooth addresses. Firstly, the NXT modules need to be paired with the PC, which tells the computer the device is able to connect to. After successfully paired with the computer, we need to add the bluetooth addresses of NXTs to the configuration files, which allows the `ChNXT API connect()` to access those devices. The detailed information about paring part will be described in another documentation, which is entitled "chnxt_bluetooth_setup.pdf", and in the next part will introduce you how to setup the bluetooth addresses of NXTs to the Ch NXT configuration file step by step.

2.1 Getting Bluetooth Addresses of NXTs

First, we need to get the Bluetooth address of a NXT. As Figure 2 shown, right click the Bluetooth icon in the right buttom corner.

Figure 2: Right click the Bluetooth icon.

Then, choose the option "Show Bluetooth Devices", in Figure 3, to find the NXT which has already paired with the computer.

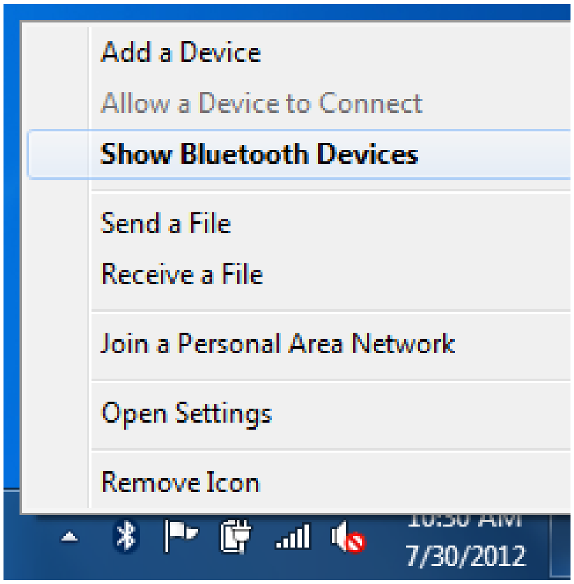

Figure 3: Choose "Show Bluetooth Devices" option.

Then, a dialog with all paired Bluetooth devices in this dialog will appear on the screen. Find the NXT icon and right click on the icon. Then choose "Properties" option as shown in Figure 4.

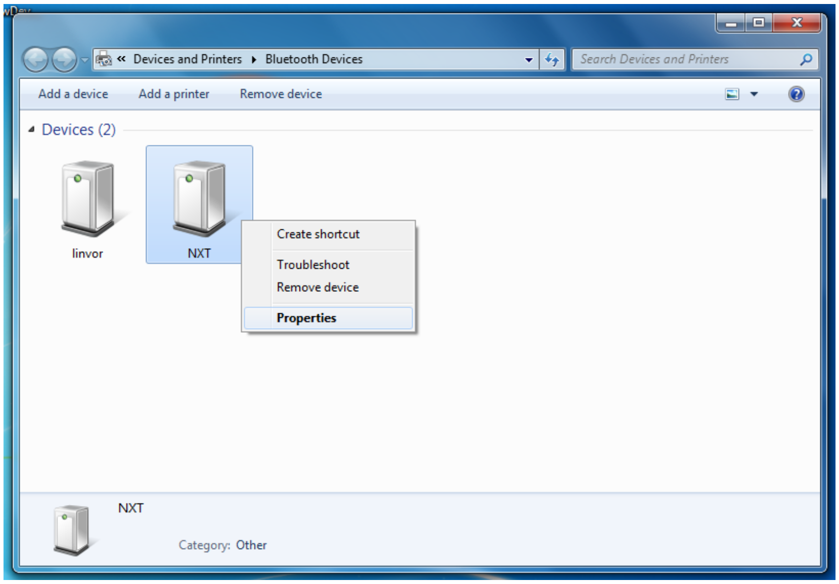

Figure 4: Right click the NXT icon then choose "Properties" option.

In the dialog of NXT properties as shown in Figure 5, click "Bluetooth" tab on the top and the bluetooth address will show as "Unique identifier" with the format as "xx:xx:xx:xx:xx:xx".

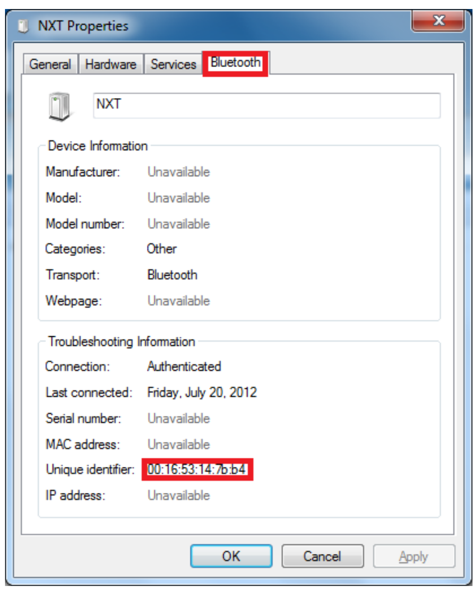

Figure 5: The properties dialog.

Now that we have already learned how to get the Bluetooth addresses of our NXTs, we can begin next section, which will introduce you how to add the Bluetooth address we have gotten into the ChNXT package.

2.2 Adding Bluetooth Addresses of NXTs in ChNXT Controller

The configuration is performed through the `ChNXT Controller` program. Start the provided ChNXT Control Program by clicking on the icon labeled "ChNXTController" on your desktop, as shown in Figure 6.

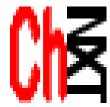

Figure 6: The icon for the ChNXT Controller.

Then the main dialog of ChNXT Controller will appear on your screen as shown in Figure 7 below.

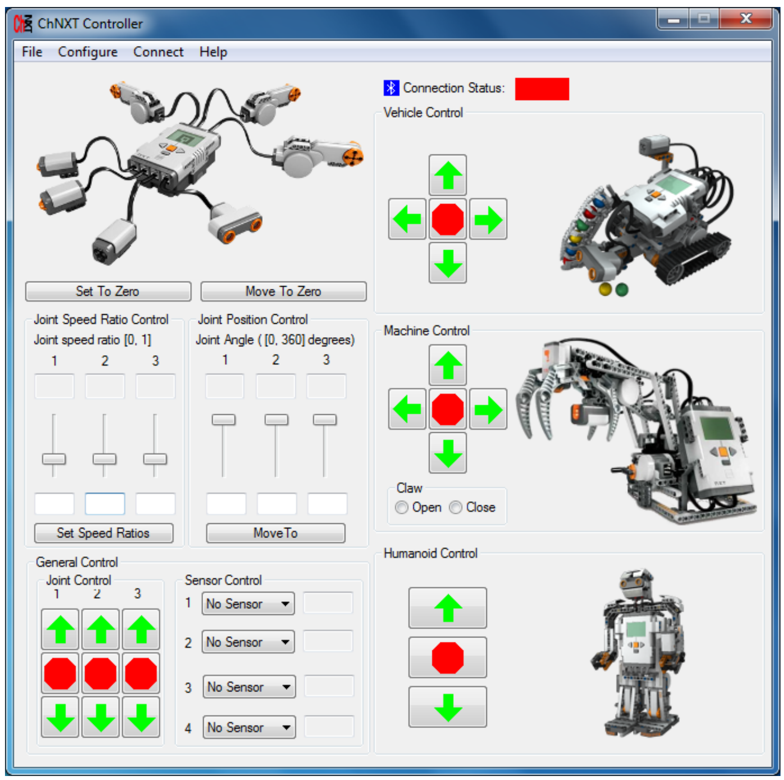

Figure 7: The main dialog of the ChNXT Controller.

Then click the menu item "Configure → Configure Robot Bluetooth", as shown in Figure 8.

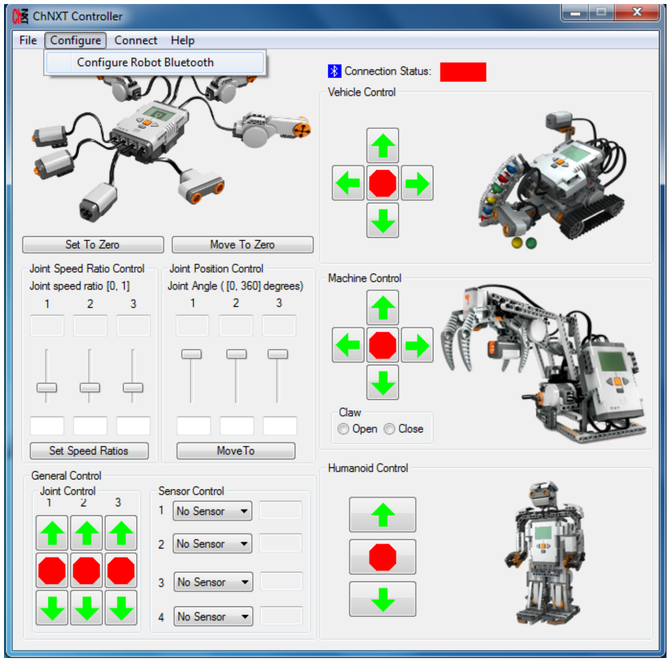

Figure 8: Configuring robot bluetooth connection.

Now, the configuration dialog will appear on your screen, which is titled as "Configure Robot Bluetooth", as shown in Figure 9.

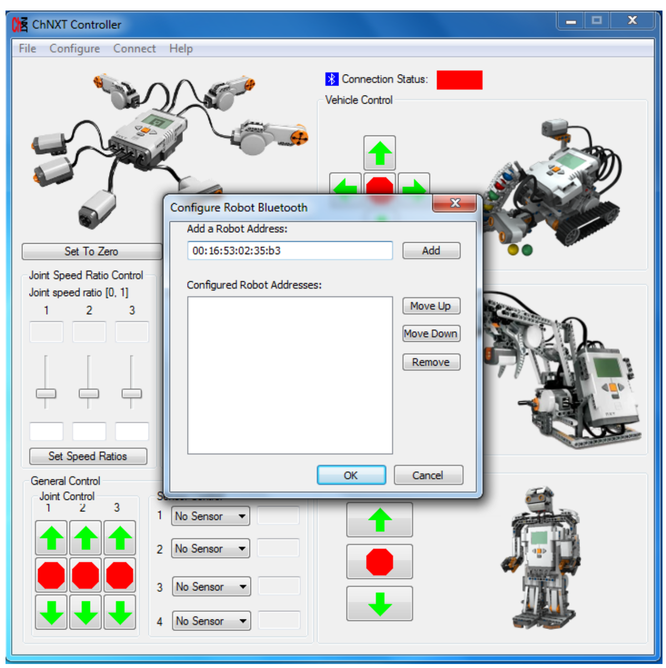

Figure 9: The bluetooth configuration dialog.

In this configuration dialog, we can add robot bluetooth addresses to the list of currently know robot bluetooth addresses. To add an address, first type in the address in the text box on the top of the dialog, as shown in Figure 10.

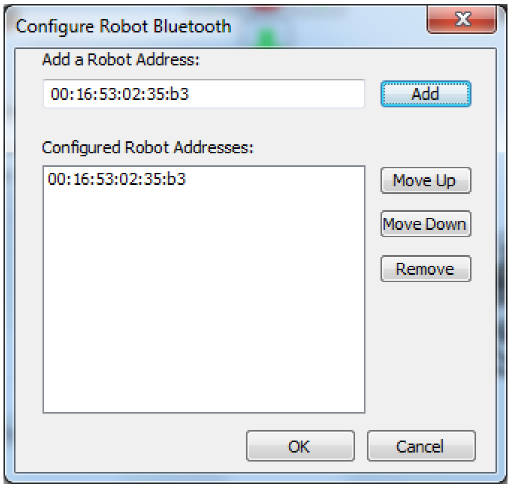

Figure 10: Adding the robot bluetooth address in the dialog window.

Then, click the "Add" button. The newly added address will appear in the list of known addresses and click "Ok" button to exit the configuration dialog. With this method, you can add more bluetooth addresses. Also, the "Move Up" button, "Move Down" button, and the "Remove" button can be used to manage your bluetooth address list.

2.3 Connecting and Disconnecting to NXTs from the ChNXT Controller

Once bluetooth addresses are added to the ChNXT Controller, you are now able to connect to a NXT device by clicking on the "Connect → Connect to Robot" menu item. Then the first NXT on the bluetooth address list will be connected and please make sure the NXT is turned on, otherwise the connection will fail. After connected, the Bluetooth connection status on the main dialog will turn green as Figure 11 shows. Besides, the joints angles and joints speed ratios will show on the main dialog too.

Figure 11: The Bluetooth Connection Status bar.

To disconnect from a robot, you can click "Connect → Disconnect from Robot" menu item, the computer will disconnect from the remote device and then the Bluetooth will turn red. Those two menu items are as shown in Figure 12.

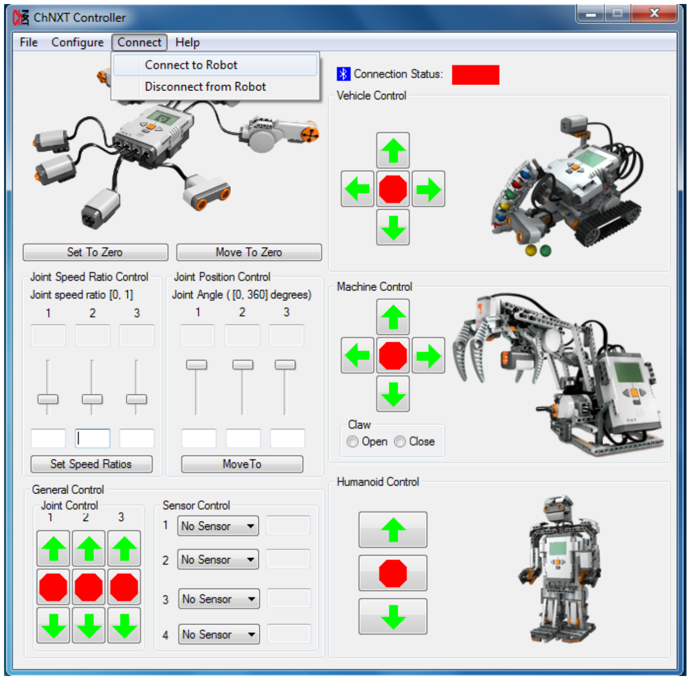

Figure 12: Connecting to and disconnecting from an NXT.

Please note that in order to run a Ch program that controls NXTs, the NXTs should not currently be connected to any other application, including the ChNXT Controller, other Ch Programs, and other programs on other devices.

Furthermore, the Bluetooth devices have a maximum limit of connected devices. The maximum limit is 7 devices connected simultaneously.

3 Control the NXT via the GUI controller

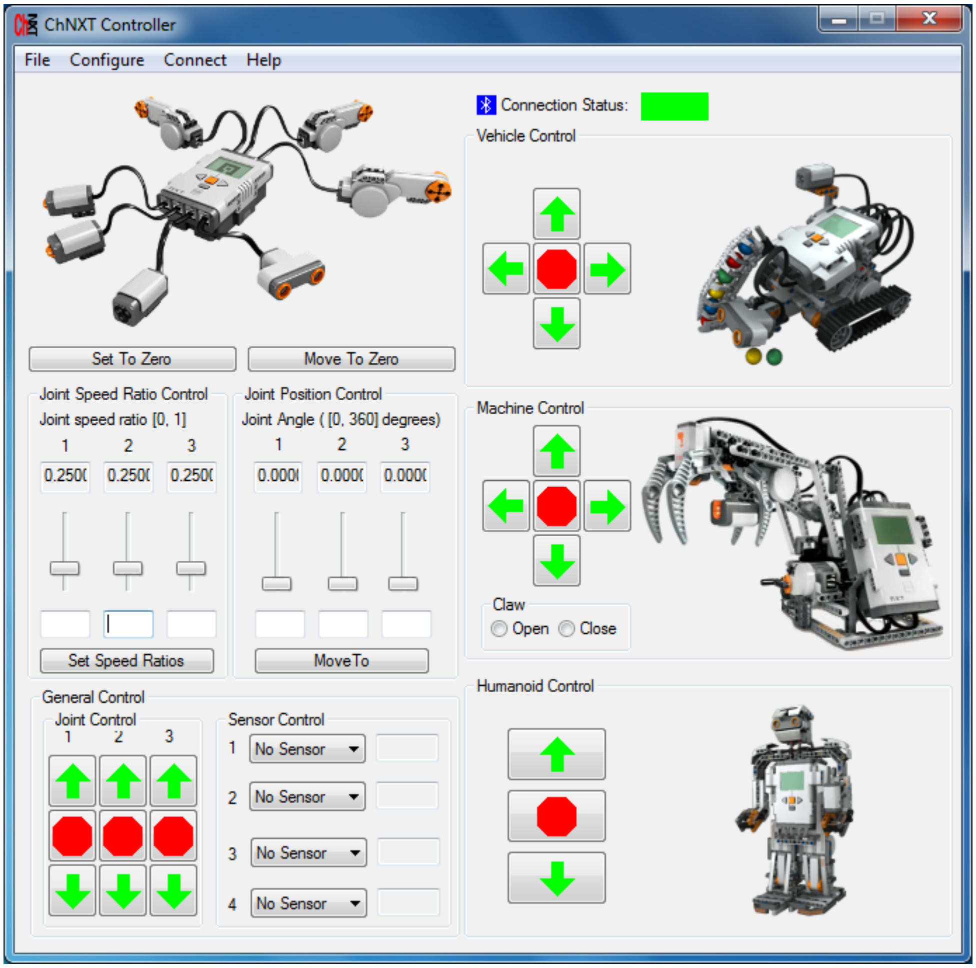

Figure 13: The main dialog of the ChNXT Controller.

In this section, I will introduce how to use Graphic User's Interface (GUI) to control NXT bricks. Compared with the C/C++ code, it is a easier way to control an NXT brick. Even though the GUI controller offers a direct and convinient way for users, it has some limitations. In the following contents, we will introduce all the features of the controller.

3.1 Introduction

The GUI controller consists of six parts. The first part is the main diagram on the left top of the controller. The second part is including the "Reset To Zer"o button and the "Move To Zero" button, and also the speed and joint set side bars. The next four parts contains general control, vehicle control, machine control and humanoid control.

3.2 The main diagram and the two "Zero" buttons

First, as we can see in the main diagram, each NXT brick can have a maximum four sensors and three motors. Under the main diagram is the "Reset To Zero" button and the "Move To Zero" button. Once the "Reset To Zero" button is clicked, the motor counter in the NXT will be reset to zero, which means that the NXT will remove the previous record of position and set the current position as the zero position. The "Move To Zero" will make the motor rotate back to the zero point wherever you set before.

3.3 The Slide Bars

The slide bars are used to set the speed ratios and positions of the joints. The sliders for "Joint Speed Ratio Control", are three the slide bars for each motor and two text boxes. One is on the top of the slide bar and the other is in buttom of each slide bar. The top text box shows the current speed ratio of the coresponding joint and the slide bar. The text box in the bottom can be used to set a speed ratio. Clicking the "Set Speed Ratios" button below the speed ratio of the joint will set it to the one you typed in. Also, you can type three ratios and set them concurrently. The speed ratio is from 0 to 1. Similarly, in the "Joint Position Control" section, it shows the current joint position in top three text boxes that users can type the desired positions in the three text boxes in the bottom and click the "MoveTo" button to move the joints to the desired position.

3.4 General Control

In the general control block, there are three sets of buttons to control each of the three joints. There are also four text boxes used to setup sensors and to view the values of those sensors. The whole block is used for general control of a NXT. The NXT can be controlled in any configuration, such as a vehicle or a humanoid.

For the button sets, it is easy to find that the button with the "up" icon indicates the positive direction. Once it is clicked the joint will move continuously in the direction. Similarly, the "down" button means the negative direction and it will make the joint keep moving in the negative direction. The red "stop" button will stop the joints immediately. The numbers on the top of the "up" buttons indicates the joints.

For the "Sensor Control" section, each drop down list represent one sensor port on the NXT. In the drop-down menu, there is a list of sensor types that you can set and the text box right beside the menu will show the value of the coresponding sensor. Please choose the correct sensor type for the physical configuration.

3.5 Specified Configurations Control

The next three blocks control the NXT in three specified configurations, which are vehicle, machine and humanoid configuration.

In the vehicle part, there are five buttons, which are "up", "down", "right", "left" and "stop" buttons. These buttons can make the NXT as a vehicle move forward and backward, rotate it right and left and also stop it. The default joints of two wheels of the vehicle are Port B and Port C on the NXT. To control the third joint, which is jointA on Port A of the NXT, you can use the buttons in the "General Control" block.

For the machine configuration, we assume that the machine has two joints to mov e an arm. One moves the arm back and forth and the other rotates it around the center. The third joint is used for controlling the claw on the arm. As we can see, the "up" and "down" buttons are used to contrl the arm move back and forth. The "righ" and "left" buttons rotate the arm. There is a subblock called "Claw" controlling it to open or close. Also you can use the buttons in the "General Control" but please note that the joint for the claw has a limited motion. You might break it by moving it continuously.

The last section is for the humanoid control. There are only three buttons in this part. The "up" button will make the robot walk forward. The "down" will move the robot backward, and the "stop" button will stop the robot.

Finally, all three sections share the "Sensor Control" section in the "General Control" block. You can setup sensors for vehicles, machines and humanoids as introduced in Section 3.4 on Page 9.

4 Getting Started with Ch NXT Package

In this chapter, the basics of controlling an NXT via Ch programing will be discussed. The basics include controlling an NXT joint, setting up an NXT sensor, and also getting information from joints and sensors. The basic structure of a Ch NXT robot program is shown in a flowchart in Figure 14.

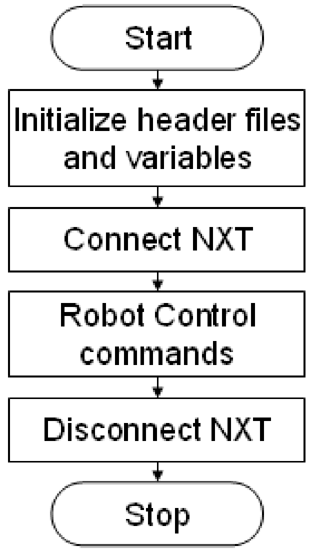

Figure 14: Flow Diagram of a basic NXT program

Also, to successfully control the Mindstorm NXT using Ch, it is important to practice good coding habits. The format of the Ch Mindstorm code is very similar to how a normal C code would be written, with the inclusion of some Ch specific functions and header files that are used to connect and control the NXT. To help the user become acquainted with the Ch NXT programs, sample programs will be presented in this section to illustrate the basics and minimum requirements of a Ch NXT control program. The sample programsare located at `CHHOME/package /chnxt/demos`, where `CHHOME` is the Ch home directory, such as `C :\Ch` for Windows. Therefore, for Windows, the demos are located at `C:\Ch\package\chnxt\demos` by default.

4.1 Introduction for ChNXT Package

Each NXT brick has three ports for motors and four ports for sensors. Therefore, in the ChNXT Package, we have our own symbols to represent the motors and the sensors, which are presented in the following tables.

Symbols for motors

`NXT_JOINTA`	PortA on the Lego Mindstorms NXT.
`NXT_JOINTB`	PortB on the Lego Mindstorms NXT.
`NXT_JOINTC`	PortC on the Lego Mindstorms NXT.

Symbols for sensors

NXT_SENSORPORT1	PORT1 on the Lego Mindstorms NXT.
NXT_SENSORPORT2	PORT2 on the Lego Mindstorms NXT.
NXT_SENSORPORT3	PORT3 on the Lego Mindstorms NXT.
NXT_SENSORPORT4	PORT4 on the Lego Mindstorms NXT.

For the motors, you need to specify the speed or speed ratios, direction of rotation or the angle for moving. However, the sensors have more complicated arguments, which are the types of sensors and the modes of sensors. The tables below show the types and the modes of sensors.

Sensor types

NXT_SENSORTYPE_SWITCH	Set to a switch type sensor. Touch sensor is a switch type sensor.
NXT_SENSORTYPE_LIGHT_ACTIVE	Set to Light Sensor in light active mode(LED on).
NXT_SENSORTYPE_LIGHT_INACTIVE	Set to Light Sensor in light inactive mode(LED off).
NXT_SENSORTYPE_SOUND_DB	Set to Sound Sensor in dB.
NXT_SENSORTYPE_SOUND_DBA	Set to Sound Sensor in dB with adjusted.
NXT_SENSORTYPE_LOWSPEED_9V	Set to ISP type sensor with 9 Voltage. The ultrasonic sensor belongs to this type of sensor.
NXT_SENSORTYPE_COLORFULL	Set to Color Sensor in color detector mode.
NXT_SENSORTYPE_COLORRED	Set to Color Sensor in lightsensor mode with red light on.
NXT_SENSORTYPE_COLORGREEN	Set to Color Sensor in lightsensor mode with green light on.
NXT_SENSORTYPE_COLORBLUE	Set to Color Sensor in lightsensor mode with blue light on.
NXT_SENSORTYPE_COLORNONE	Set to Color Sensor in lightsensor mode with no light on.

Before using NXT sensors, the type of the sensor has to been specified. There are many different kind of sensors, but in our package so far, we only support touch sensor, light sensor, sound sensor, color sensor, and ultrasonic sensor. Also, each sensor has a data mode, which needs to be specified before using.

Sensor modes

NXT_SENSORMODE_RAWMODE	Get sensor value as raw mode.
NXT_SENSORMODE_BOOLEANMODE	Get sensor value as boolean mode.
NXT_SENSORMODE_PCTFULLSCALEDMODE	Get sensor value as percentage of full scale reading for configured sensor type.

In our ChNXT package, we only support three modes for the sensors. For ultrasonic sensor and color sensor, we use raw mode. For touch sensor, we use boolean mode, which has only two value of 0 and 1 to indicate true or false. For the light sensor, sound sensor and color sensor, we use the percentage of full scaled reading mode.

In the future, we will support more kinds of sensors and relavant sensor modes. In the next section, we will start to control an NXT brick with Ch code.

4.2 A Basic Ch NXT Program

The first demo presents a simple program which connects to a Lego Mindstorms NXT and moves joints B and C.

Source Code

```
/* File name: start.ch
 *
 * Move the NXT all joints by 360 degrees.*/

#include <nxt.h>

ChNXT nxt;

/* Connect to the paired NXT */
nxt.connect();

/* Set the robot to "home" position,
 * where all joint angles are 0 degrees.*/
nxt.moveToZero();

/* Rotate joint 2 and 3 by 360 degrees */
nxt.move(360, 360, 360);
```

Program 1: `start.ch` Source Code

Explanation

The beginning of every program will include necessary header files. Each header file imports functions used for a number of tasks, such as displaying messages on the screen or controlling the Lego Mindstorms NXT. The header file `nxt.h`, which contains all `ChNXT` class and other related functions for controlling the NXT, should be included in each Ch NXT program.

```
#include <nxt.h>
```

The following line initializes a new variable named nxt which represents the remote Lego Mindstorm NXT which we wish to control. The special variable is actually an instance of the ChNXT class, which contains its own set of functions called "methods", "menber functions", or simply "functions".

```
ChNXT nxt;
```

The next line,

```
nxt.connect();
```

will connect our computer to the remote NXT, which is related to the new variable `nxt`.

Another way to get connected with the NXT is using the function `connectWithAddress()`, and the usesage is as following:

```
nxt.connectWithAddress("11:22:33:44:55:66");
```

The string `"11:22:33:44:55:66"` represents the Bluetooth address of the Lego Mindstorms NXT you wish to connect. Detailed documentation for `connect()` and `connectWithAddress()` are presented in Appendix A on page 68. The next line,

```
nxt.moveToZero();
```

uses the function `moveToZero()` which is a member function of class `ChNXT`. The function causes all joints of the connected NXT move to zero position, which means the absolute angle will be all zero.

The next line of code will cause all joints of the connected NXT rotate 360 degrees.

```
nxt.move(360, 360, 360);
```

The member function `move()` expects input angles in degrees. If you want to use angles in radians, the conversion need to be done via the function `rad2deg()`. The function is implemented in Ch with the code

```
#include <math.h> /* For M_PI */
double rad2deg(double radians){
    double degrees;
    degrees = radians * 180.0 / M_PI;
    return degrees;
}
```

If desired, values in radians may also be converted to degrees using the counterpart function, `deg2rad()`. Detailed information for function `rad2deg()` and function `deg2rad()` can be found in Appendix B on Page 103.

4.3 Setting the Zero Positions for NXT Joints

In the last section, we introduced a very basic demo of Ch NXT programs. In this section, another simple demo will be presented to illustrated how to set absolute zero positions for joints.

Source Code

```
/* File name: setZero.ch
 * presents how to set zero positions for NXT joints. */

#include <nxt.h>

ChNXT nxt;

/* Connect to the NXT */
nxt.connect();

/* Set new zero positions */
nxt.setJointToZero(NXT_JOINTA);

/* Move to zero */
nxt.moveToZero();
```

Program 2: `setZero.ch` Source Code

Explanation

The first several lines of the code,

```
#include <nxt.h>

ChNXT nxt;
```

```
/* Connect to the NXT */
nxt.connect();
```

initializes the program, declares the variable and connects to the remote device. The next line,

```
/* Set new zero positions */
nxt.setJointToZero(NXT_JOINTA);
```

sets `NXT_JOINTA` to the new zero position. In order to set new zero positions, user should move the desired position before calling the function `setJointToZero()`. The function `setJointToZero()` can only set zero position for one joint. The argument it takes is the target joint. Another function called `setToZero()`, which takes no arguments, can set new zero positions for all joints. The next line,

```
/* Move to zero */
nxt.moveToZero();
```

makes all joints move to new zero positions.

4.4 Controlling the Speed of NXT Joints

Another simple Ch program, `setSpeedRatios.ch`, illustrates how to set speed ratios for the NXT's joints.

Source Code

```
/* File name: setSpeedRatios.ch
 * set speed ratios for joints of NXT. */

#include <nxt.h>
ChNXT nxt;

/* Connect to the paired NXT */
nxt.connect();

/* set speed ratios */
nxt.setJointSpeedRatios(0, 0.4, 0.4);
nxt.setJointSpeedRatio(NXT_JOINTA, 0.5);

/* make NXT joints move */
nxt.move(360, 360, 360);
```

Program 3: `setSpeedRatios.ch` Source Code

Explanation

The fisrt several lines,

```
#include <nxt.h>
ChNXT nxt;

/* Connect to the paired NXT */
nxt.connect();
```

initializes the program, variable, and connect to the NXT. The next three lines,

```
/* set speed ratios */
nxt.setJointSpeedRatios(0, 0.4, 0.4);
nxt.setJointSpeedRatio(NXT_JOINTA, 0.5);
```

sets the speed ratios setting for all joints on the NXT. The function `setJointSpeedRatios()` sets the speed ratio for `NXT_JOINTA` as 0 and sets the speed ratios for `NXT_JOINTB` and `NXT_JOINTC` as 0.4. The function `setJointSpeedRatio()`, which can only set speed ratio for one joint, sets the speed ratio for `NXT_JOINTA` as 0.5.

The next line,

```
/* make NXT joints move */
nxt.move(360, 360, 360)
```

makes three joints of NXT move at setted speed ratios. The usage of the function `move()` is just as the demo discussed in Section 4.2 on Page 13.

4.5 Making NXT Joints Move

Now that we have already discussed connect/disconnect and set speed ratios for joints, this section will presents some of the moving functions.

4.5.1 A Demo Program for Movement Functions

In this section, a simple demo program will be presented to illustrate the series functions of moving NXT joints.

Source Code

```
/* File name: move.ch
 * illustrate the full series of moving function */

#include <nxt.h>
ChNXT nxt;

/* Connect to the NXT */
nxt.connect();

/* Set speed ratios */
nxt.setJointSpeedRatios(0.5, 0.5, 0.5);

/* move joint to zero position */
nxt.moveToZero();

/* move a joint by user specified angle */
nxt.moveJoint(NXT_JOINTA, 360);

/* move a joint to absolute angle */
nxt.moveJointTo(NXT_JOINTA, 360);

/* move all joints by specified angles */
nxt.move(180, 360, 360);

/* move all joints to absolute angles */
nxt.moveTo(360, 360, 360);
```

Program 4: `move.ch` Source Code

Explanation

The first part of code,

```
#include <nxt.h>
ChNXT nxt;

/* Connect to the NXT */
nxt.connect();

/* Set speed ratios */
nxt.setJointSpeedRatios(0.5, 0.5, 0.5);
```

initializes the program, connects to the NXT and sets speed ratios for joints. The next couple lines,

```
/* move joint to zero position */
nxt.moveToZero();
```

moves the joint to the absolute zero positions for all joints. The function `moveToZero()` is used to move joints to absolute zero positions. The next four lines,

```
/* move a joint by user specified angle */
nxt.moveJoint(NXT_JOINTA, 360);

/* move a joint to absolute angle */
nxt.moveJointTo(NXT_JOINTA, 360);
```

includes two functions `moveJoint()` and `moveJointTo()`, which makes one joint move a specified angle relatively and absolutely, respectively. Similarly, the next four lines,

```
/* move all joints by specified angles */
nxt.move(180, 360, 360);

/* move all joints to absolute angles */
nxt.moveTo(360, 360, 360);
```

include functions `move()` and `moveTo()`, which makes all joints move specified angles relatively and absolutely, respectively.

4.5.2 Blocking and Non-Blocking Functions

The movement functions described in previous demo are all blocking functions. Once the blocking movement functions are called, the functions will hang, or "block", until all the joints have stopped moving. However, the movement functions also have "non-blocking" version, which means the function returns immediately and the function `moveWait()` can be used to wait for the movement to stopping. `NB` at the end of non-blocking functions indicates that the functions are non-blocking version, such as `moveNB()`. A simple example will be presented in the following.

Example

```
/* File name: blockNonblock.ch
   To illustrate the block and non-block functions */

#include <nxt.h>

ChNXT nxt;

/* Connect to the NXT */
nxt.connect();
```

```
/* Non-blocking Function */
nxt.moveJointNB(NXT_JOINTA, 360);
printf("This message will be printed on \
        the screen when joint1 is moving.\n");
nxt.moveWait();

/* Blocking Function */
nxt.moveJoint(NXT_JOINTA, 360);
printf("This message will be printed on \
        the screen after joint1 stopped moving.\n");
```

Program 5: `blockNonblock.ch` Source Code

Explanation

```
/* Non-blocking Function */
nxt.moveJointNB(NXT_JOINTA, 360);
printf("This message will be printed \
        on the screen when jointA is moving.\n");
nxt.moveWait();
```

uses the non-blocking function `moveJointNB()`. The function `printf()` will print a message onto the screen while the jointA is moving. However, the blocking part below,

```
/* Blocking Function */
nxt.moveJoint(NXT_JOING1, 360);
printf("This message will be printed on \
        the screen after jointA stopped moving.\n");
```

will print out the message after the function `moveJoint()` finishes, which means after jointA stops moving. Most moving functions have both blocking and non-blocking versions. However, there are still some exceptions. The function `moveJointContinuousNB()` and function `moveContinuousNB()` have only non-blocking version and the function `moveContinuousTime()` is only blocking. In Table 5, we list all blocking functions and their corresponding non-blocking functions.

Table 5: Block and Non-block Functions

Blocking Functions	Non-blocking Functions
`moveJoint()`	`moveJointNB()`
`moveJointTo()`	`moveJointToNB()`
`move()`	`moveNB()`
`moveTo()`	`moveToNB()`
`moveToZero()`	`moveToZeroNB()`
`vehicleRollForward()`	`vehicleRollForwardNB()`
`vehicleRollBackward()`	`vehicleRollBackwardNB()`
`vehicleRotateLeft()`	`vehicleRotateLeftNB()`
`vehicleRotateRight()`	`vehicleRotateRightNB()`

Also, detailed information for both blocking and non-blocking versions of movement functions can be found in Appendix A on Page 68.

4.6 Retrieving a Joint Angle

This demo presents how to get a current joint angle in a Ch NXT program. The angle is the absolute postion in degrees.

Source Code

```
/* Filename: getJointAngle.ch
 * Find the current joint angle of a joint */

#include <nxt.h>
ChNXT nxt;

/* Connect to a NXT */
nxt.connect();

/* Get the joint angle of the first joint */
double angle;
nxt.getJointAngle(NXT_JOINTA, angle);

/* Print out the joint angle */
printf("The current joint angle for joint 1 is %lf degrees.\n", angle);
```

Program 6: `getJointAngle.ch` Source Code

Explanation

```
/* Get the joint angle of the first joint */
double angle;
nxt.getJointAngle(NXT_JOINTA, angle);
```

retrieve the current angle of joint 1. `NXT_JOINTA` is an enumerated value defined in the header file `nxt.h`. Detailed information for all enumerated values defined in `nxt.h` can be found in Appendix A.1 on page 68. Finally, the last part of the program,

```
/* Print out the joint angle */
printf("The current joint angle for joint 1 is %lf degrees.\n", angle);
```

prints the value of the variable onto the screen.

4.7 Setting Sensors for NXT

We have finished discussing the connection and movement functions in previous sections. In this section, we will start to discuss how to use the sensors of the NXT. The following demo code will present how to setup a sensor for the NXT and how to get values collected by the sensor from the NXT.

4.7.1 Use Touch Sensor and Ultrasonic Sensor

In this section, a demo program will be presented to demonstrate how to use touch sensors and ultrasonic sensors with the NXT bricks by using Ch code.

Source Code

```
/* File name: sensor.ch
 * A breif introduction of using sensors of NXT.*/

#include <nxt.h>

ChNXT nxt;

/* Setup sensors and check sensor connection */
int status1=2, status2=2;

/* Variables to store values gotten from NXT */
int touchValue, ultraValue;

/* Connect to NXT */
nxt.connect();

/* Save status of NXT_SENSORPORT1, and NXT_SENSORPORT4 */
status1 = nxt.setSensor(NXT_SENSORPORT1,
            NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);

status2 = nxt.setSensor(NXT_SENSORPORT4,
            NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);

/* Check connection status sensors connection */
if(status1) {
    printf("Fail to setup sensors.\n");
    exit(-1);
}

if(status2) {
    printf("Fail to setup sensors.\n");
    exit(-1);
}

/* get values collected by sensors from NXT */
nxt.getSensor(NXT_SENSORPORT1, touchValue);
nxt.getSensor(NXT_SENSORPORT4, ultraValue);

/* display the values we got onto the screen */
printf("Touch sensor: %d\n", touchValue);
printf("Ultrasonic sensor: %d\n", ultraValue);
```

Program 7: `sensor.ch` Source Code

Explanation

The first part of the code,

```
#include <nxt.h>

ChNXT nxt;

/* Setup sensors and check sensor connection */
int status1=2, status2=2;

/* Variables to store values gotten from NXT */
```

```
int touchValue, ultraValue;

/* Connect to NXT */
nxt.connect();
```

initializes the program, declares the variables and connects to the NXT. Here, we declared two more sets of variables, where one set is used to check the connection status of sensors and another set is used to store the values collected by sensors. The next part of code,

```
/* Save status of NXT_SENSORPORT1, and NXT_SENSORPORT4 */
status1 = nxt.setSensor(NXT_SENSORPORT1,
            NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);

status2 = nxt.setSensor(NXT_SENSORPORT4,
            NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);
```

setups two sensors for the connected NXT. The function we used to setup sensors is `setSensor()`, which takes three arguments. The first argument represents the port number on the NXT, which is in type `nxtSensorPort_t`. The second argument the function takes is the type of a sensor, which is in type `nxtSensorType_t`. The last argument is the working mode of a sensor, which is in type `nxtSensorMode_t`. Detailed information for the three variable types can be found in Appendix A.1 on Page A.1. The next several lines,

```
/* Check connection status sensors connection */
if(status1) {
    printf("Fail to setup sensors.\n");
    exit(-1);
}

if(status2) {
    printf("Fail to setup sensors.\n");
    exit(-1);
}
```

check the status of each sensor to see if they were setup correctly. If it fails to setup sensors, the program will exit automatically. The function `setSensor()` returns 0 when run successfully. The next part of code,

```
/* get values collected by sensors from NXT */
nxt.getSensor(NXT_SENSORPORT1, touchValue);
nxt.getSensor(NXT_SENSORPORT4, ultraValue);
```

gets the collected values from the NXT by using the function `getSensor()`. In order to use the function, two arguments are neccessary. One is the sensor port in type `nxtSensorType_t`. Another argument is the variable used to store value gotten from the NXT. The last part,

```
/* display the values we got onto the screen */
printf("Touch sensor: %d\n", touchValue);
printf("Ultrasonic sensor: %d\n", ultraValue);
```

displays the values gotten from the NXT onto the screen. Since the touch sensor sets as boolean mode, there only two values. One is 1, which means true or touched, while the other is 0, which means false or untouched. The value of the ultrasonic sensor ranges from 0 to 80 with units of centimeter.

4.7.2 How to use other sensors

Besides the touch sensors and ultrasonic sensors discussed in the previous section, the Ch package also supports several other sensors, such as light sensors, sound sensors, and color sensors. In this section, we will introduce the usage of those sensors.

Light Sensors and Sound Sensors

For the light sensor and the sound sensor, we use the same functions as we used in last section, which are `setSensor()` and `getSensor()`. The function `setSensor` is used to setup a sensor and the function `getSensor()` is used for getting the value of the sensor from the NXT. However, when setting up a sensor as a light or sound sensor, the mode of the sensor should be percentage of full scaled, which is `NXT_SENSORTYPE_PCTFULLSCALEDMODE`. Therefore, the values of light and sound sensors should range from 0 to 100 percent.

Furthermore, there is a LED on the light sensor. The user can decide to turn on the LED or not by their sensor types, which are `NXT_SENSORTYPE_LIGHT_ACTIVE` and `NXT_SENSORTYPE_LIGHT_INACTIVE`, which indicats the LED should be used. Also, the sound sensor has two sensor types too. They are `NXT_SENSORTYPE_SOUND_DB` and `NXT_SENSORTYPE_SOUND_DBA`. The `DB` in both types is "Decibel", which is the unit for measuring sound level. The `A` after `DB` in the second type means "adjusted", which indicates that the value is more precise.

Color Sensor

The color sensor of the Lego Mindstorms has two modes. It can be used as a color sensor to detect different colors and it also can be used as a light sensor. When the color sensor is used as a light sensor, it works the same as the light sensor. However, it has three colors of LEDs rather than one LED in simple light sensors. Therefore, you can choose red, blue, green or no light. The options are `NXT_SENSORTYPE_COLORRED`, `NXT_SENSORTYPE_COLORBLUE`, `NXT_SENSORTYPE_COLORGREEN` and `NXT_SENSORTYPE_COLORNONE`.

```
nxt.setSensor(NXT_SENSORPORT3, NXT_SENSORTYPE_COLORRED,
    NXT_SENSORMODE_PCTFULLSCALEDMODE);

nxt.setSensor(NXT_SENSORPORT3, NXT_SENSORTYPE_COLORGREEN,
    NXT_SENSORMODE_PCTFULLSCALEDMODE);

nxt.setSensor(NXT_SENSORPORT1, NXT_SENSORTYPE_COLORBLUE,
    NXT_SENSORMODE_PCTFULLSCALEDMODE);

nxt.setSensor(NXT_SENSORPORT1, NXT_SENSORTYPE_COLORNONE,
    NXT_SENSORMODE_PCTFULLSCALEDMODE);
```

There is another option type called `NXT_SENSORTYPE_COLORFULL`, which will be used to setup the color sensor to detect different colors. In this case, the sensor mode should be `NXT_SENSORMODE_RAWMODE`, which is the same as the ultrasonic sensor.

5 Controlling a NXT Vehicle

Figure 15: NXT Vehicle

The NXT comes with three actuator output ports. The actuators available are the NXT motors. Normaly, you can only control the speed and direction of the connected motors. For a two wheeled NXT vehicle, there are two ways that the NXT vehicle can be controlled. In addition to moving the NXT by controlling the individual motors as shown in Figure 15, you can also use a set of Ch mindstorm functions writen specificly to control an NXT vehicle. The diagram of the vehicle and the motor ports is shown in Figure 16. When controlling the individual motors, you would need to define the speed and direction of each motor. The functions in the following examples require only a speed. In this section, we will show a basic Ch NXT program to move the robot forward. Please make sure your NXT vehicle are configured according to to Figure 16 to run our demonstration programs.

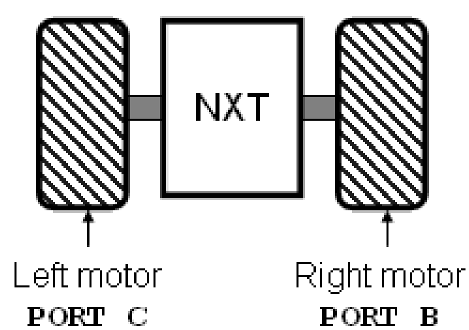

Figure 16: Motor configuration of the NXT Vehicle

5.1 How to make your NXT move forward

To help the user become acquainted with the Ch NXT package, the example `vehicleRollForward.ch` will be presented in the following section to illustrate the basics and minimum requirements of a Ch NXT control program.

```
/* File name: forward.ch
 *
 * Introduce the CH Mindstorms control Package syntax
 * to new users by moving the robot forward. */

#include <nxt.h>

ChNXT nxt;

/* Connect to NXT */
if (nxt.connect()) {
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(-1);
}

/* Set Speed Ratio */
nxt.setJointSpeedRatios(0, 0.25, 0.25);

/* Turn the motors on */
nxt.vehicleRollForward(360);
```

Program 8: `forward.ch` Source Code

5.1.1 Initialization

In the beginning of a Ch Mindstorms NXT program or any C program, you must include proper header files to run the program properly. Without proper header files, the program will not have the specific libraries or source codes to run the program. Essential header files for the NXT includes:

```
#include <nxt.h>
```

The `nxt.h` is the essential header file for Ch NXT control functions and variables.

5.1.2 Connect the NXT and Checking Connection Status

The NXT status, sensor/encoder data, and input/output protocols are stored in a C++ class called `ChNXT`. This class must be created in every NXT program in order to connect. Therefore, in the beginning of your code, you must define a `ChNXT` class and use the `connect()` function to connect to the NXT. The `connect()` function will return a 1 if no connection is established, so you will have to terminate your program if no connection is established. An example of how to create the `class` and how to retrieve data are shown below:

```
ChNXT nxt;

/* Check status of NXT connection */
if (nxt.connect()){
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(-1);
}
```

The line `ChNXT nxt;` creates the class `nxt` that is used to store data and control the NXT robot. The function `nxt.connect()` called in the if statement is used to terminate the program in the event no connection is established. The `printf()` is included to print out an error message if `nxt.connect` fails. To end the program if connection fails, the function `exit()` is used. The use of `exit(-1)` is similar to the C function `return 0;`, that can be used when no `main()` function is present.

5.1.3 Moving the Robot Forward

After estabilishing the connection between a computer and an NXT, you can move the NXT vehicle forward by using the `vehicleRollForward()`. After using the `vehicleRollForward()` function, the program will wait for the motion stopping. Figure 17 shows the NXT vehicle moving forward by actuating both wheels forward with the same speed ratio, which is how the function `vehicleRollForward()` works. By default, the ports for the two wheels on the NXT are `NXT_JOINTB` and `NXT_JOINTC`.

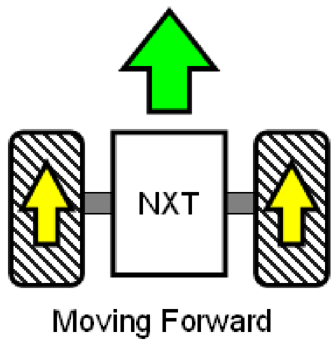

Figure 17: Top-down view of NXT vehicle with two wheels

The following part is the maint code to make the NXT vehicle robot move forward in the program `vehicleRollForward.ch`:

```
/* set speed ratio */
nxt.setSpeedRatios(0, 0.25, 0.25);

/* Move foward */
nxt.vehicleRollForward(360);
```

5.1.4 Ending your program

After you finish your program, you must end your program properly by stopping all the motors and disconnect the NXT from your computer. You can stop the motors using the `stopAllJoints()` function, which stops all of the NXT motors. To disconnect the nxt, use the `disconnect()` function. For example:

```
/* Stop the motors */
nxt.stopAllJoints();

/* Disconnect NXT */
nxt.disconnect();
```

The disconnection process is not neccessary since the program will kill the connection between the computer and the remote device automatically when it finishes execution.

5.2 How to make your NXT move backward

To make a NXT vehicle move backward, the function `vehicleRollBackward()` can be used. The function works smilarly to `vehicleRollForward()`, moving the robot backwards. The function works by actuationg both wheels backward at the same speed, as shown in Figure 18.
Using our new function, we can add the following code fragment to our first program to make the robot move backwards. The code fragment is shown below:

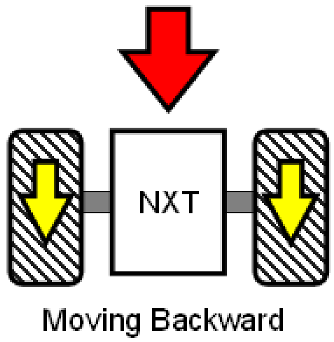

Figure 18: NXT vehicle moving backwards

```
/* set speed ratio */
nxt.setJointSpeedRatios(0, 0.25, 0.25);

/* Move backward */
nxt.vehicleRollBackward(360);
```

The modified program is called: `vehicleRollBackward.ch`.

5.3 How to make your NXT turn in place left/right

To make your NXT vehicle turn or rotate in place, the NXT vehicle wheels must be spun in the opposite direction at the same speed. For example, to rotate the NXT vehicle to the left, the right wheel must be spun forward, while the left wheel spins at the same speed in reverse. Figure 19 shown below shows the NXT vehicle turning in place left or right by actuating the wheels in opposite direction.

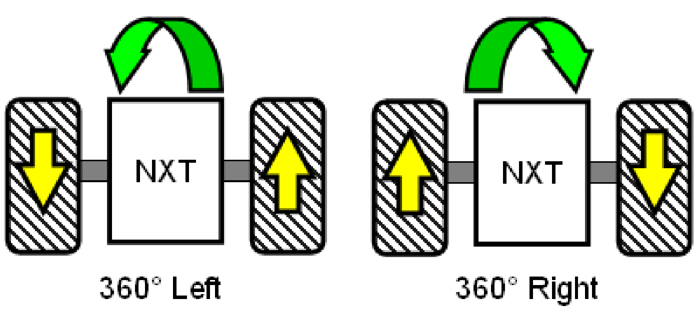

Figure 19: NXT vehicle turning 360 degrees

To make the NXT rotate in place, we can use the functions `vehicleRotateLeft()` and `vehicleRotateRight()`. An example of how to use the functions are shown below:

```
/* set speed ratio */
nxt.setJointSpeedRatios(0, 0.50, 0.50);

/* rotate left */
nxt.vehicleRotateLeft(360);

// or

/* set speed ratio */
nxt.setJointSpeedRatios(0, 0.50, 0.50);

/* rotate right */
nxt.vehicleRotateRight(360);
```

An example program is called: `vehicleRotate.ch`.

5.4 Advanced Mindstorm Motor Control

The previous section showed simplified controls for an NXT vehicle robot. To control alternate NXT designs, or to perform more advanced movements with the NXT vehicle, the NXT motors must be controlled individually. The following sections shows how to control the individual motors, and how the previously presented NXT vehicle actions can be done by controlling the individual motors.

5.4.1 Motor Control Functions

The function that is used to control the NXT motors is `moveJointContinuousNB()`. To use the function, you need the motor port and the move direction. The speed of the motors are limited, and can only ranged from 0 to 1 a speed ratio. For the direction, the data type `nxtJointState_t` is defined. In the data type, there are two values indicate the move directions. One is call `NXT_FORWARD`, which means the positive direction and the other is called `NXT_BACKWARD` means the negative direction. The `NB` indicates that the function is non-blocking. Therefore, you will need to use the `delay()` function to leave the motors on for the desired amount of the time. Otherwise, the program will go to the next statements directly. Due to the setup of the NXT vehicle, forward motion can be achieved by turning the motors on at the same speed ratio. For example:

```
/* set speed ratio */
nxt.setJointSpeedRatios(0, speedRatio, speedRatio);

/* Move foward */
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);

/* Pause program for 5 seconds */
delay(5);
```

This will move the NXT vehicle forward at the value the variable speed was set to. To get the NXT vehicle to move in reverse, the same code can be used by changing `NXT_FORWARD` to `NXT_BACKWARD`. The program `forwardBackward.ch` moves the NXT forward and backward each with 360 degrees for its wheels.

```
/* File name: forwardBackward.ch
 *
 * Introduce how to get the nxt robot
 * to reverse direction.*/
```

```
#include <nxt.h>

ChNXT nxt;
double speedRatio = 0.25;

/* Connect to NXT */
if (nxt.connect() != 0) {
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(-1);
}

/* set speed via pass the speed ratio*/
nxt.setJointSpeedRatios(0, speedRatio, speedRatio);

/* Move the Robot Forward */
nxt.vehicleRollForward(360);

/* Move the Robot Backwards */
nxt.vehicleRollBackward(360);
```

Program 9: `forwardBackward.ch` Source Code

5.4.2 Turning Using Single Motor Control

Turning and rotation movements can also be achieved using the `moveJointContinuousNB()` functions. As previously discussed, to turn our NXT vehicle, one motor must be rotating at a faster speed then the other, or the motors must be spinning in the opposite direction. For the following discussion, Figures 19 maybe useful.

To turn the NXT vehicle left, the right wheel must be moving faster in the forward direction then the left wheel. If you want to move forward and turn left, let the left wheel move at 0.7 of the speed that the right wheel is set to. To implement it with the single motor control functions, you would do the following:

```
/* set speed ratio */
nxt.setJointSpeedRatios(0, speedRatio, 0.7*speedRatio);

/* Move foward-left */
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);

/* Pause program for a while */
delay(5);
```

The value of 0.7 is somewhat arbitrary, and other constant values could be used to test the resulting NXT vehicle response. The function `vehicleRotateLeft()` works similarly, instead setting the left wheel at the negative speed of the right wheel. An example is shown below:

```
/* set speed ratio */
nxt.setJointSpeedRatios(0, speedRatio, speedRatio);

/* Rotate left in place */
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);

/* Pause program for a while */
delay(5);
```

To make the functions `vehicleRotateLeft()` and `vehicleRotateRight()` work, you need to make the speed ratio opposite. Once you understand how the single motor commands work, more advance movements can be done, such as a move back and left motion. The single motor commands can also be used to add a third motor attachment to the NXT vehicle, or used to control alternate robot designs that move or act differently.

One alternative approach to implementing turning is to increase the speed of a wheel, instead of decreasing a wheel speed. For example, to implement a left turn, we could increase the speed of the right wheel by multiplying by a constant, such as 1.2. The resulting code would look like:

```
/* set speed ratios */
nxt.setJointSpeedRatios(0, 1.2*speedRatio, speedRatio);

/* Move foward-left */
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);

/* Pause program for a while */
delay(5);
```

While this would work for slower motor speeds, a bug would occur if the NXT vehicle speed was set too high. Can you spot why? As previously discussed, the valid motor speed ratio for NXT motors are between 0 to 1. If the speed variable is set to 1, the command to control the left motor (NXT_JOINTC) will function correctly. The right motor (NXT_JOINTB) will recieve a command telling it to set the motor speed to 1.2 x 1 = 1.2. This is an invalid motor command, but if the speed ratio is set to greater than 1, the maximun speed ratio of 1 will be used. Similarly, if you try to set a speed ratio less than zero, the minimum speed ratio of zero will be used.

5.4.3 Manual Real Time Control Program

Manual real time control program allows you to control your NXT vehicle with your keyboard like a remote control. For a manual control program, a user interface is usually used to display all the possible option that a user can input into the program. The user interface allow the user to know how to control the NXT's motion. The NXT vehicle real time control (RTC) program, `vehicle_rtc.ch` prints out a user interface for the user to use while executing the program. Figure 20 is the user interface of the NXT vehicle RTC program.

In Figure 20, the user interface of the NXT vehicle RTC program display all the possible key that the user can use. In addition, the user interface also indicate the functionality of the key that is being pressed. When a specific key is pressed during the execution of the NXT vehicle RTC program, the program uses a **if-else** statement to performs a fragment of code that send commands to the NXT. For example:

- The key `"w"` is to control the NXT to move forward.
- The key `"s"` is to control the NXT to move backward.
- The key `"a"` is to control the NXT to turn left.
- The key `"d"` is to control the NXT to turn right.
- The key `"x"` is to stop the NXT motors.
- The key `"1"` is to set the NXT motor speed ratio from 0.25.
- The key `"2"` is to set the NXT motor speed ratio from 0.5.
- The key `"3"` is to set the NXT motor speed ratio from 0.75.

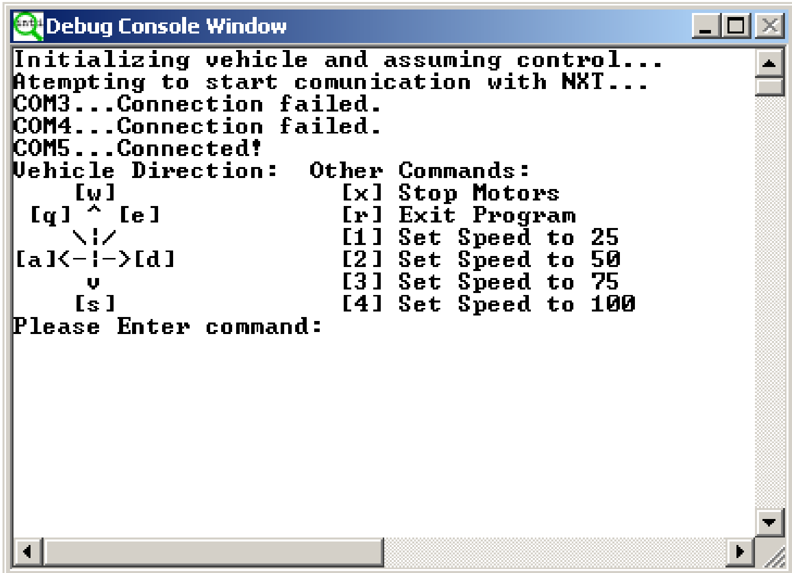

Figure 20: NXT vehicle RTC User Interface

- The key "4" is to set the NXT motor speed ratio from 1.
- The key "r" is to exit the manual RTC program.

In the robot control code block, a **while** loop is implemented to allow the user to control the NXT continuously until the program is terminated. Within the **while** loop, the program grabs the user's input and decide what to do with it using the **if-else** statements. The whole NXT vehicle RTC program is shown in Program 10. Please make sure your NXT vehicle are configured according to Figure 16 to run Program 10. In the rest of this section, we are going to explain the whole program in detail.

```
/* File name: vehicle_rtc.ch
 *
 * Demonstrate the CH Mindstorms Control Package's ability
 * to control the machine robot model, as well as demonstrate
 * how to get and use sensor data. */

#include <conio.h>
#include <stdio.h>
#include <nxt.h>

ChNXT nxt;
double speedRatio = 0.25;        //speedRatio of the motors. (default to 25)
int quit = 0,                    //used by quit case to exit the loop
    status2;                     //used to check for errors
char key = 'x',                  //stores the input from the user
     movemode = 'x';             //stores the last movement command

/* Connect to NXT */
printf("Initializing vehicle and assuming control...");
if (nxt.connect()) {
    printf("\nPress any key to exit.\n");
    while (!_kbhit());           //wait for keypress
    exit(0);
```

```
}

/* GUI display */
printf("Vehicle Direction:  Other Commands:");
printf("\n    [w]                [x] Stop Motors");
printf("\n [q] ^ [e]             [r] Exit Program");
printf("\n    \\|/                 [1] Set SpeedRatio to 0.25");
printf("\n[a]<-|->[d]            [2] Set SpeedRatio to 0.50");
printf("\n     v                 [3] Set SpeedRatio to 0.75");
printf("\n    [s]                [4] Set SpeedRatio to 1\n");
printf("Please Enter command:");

/* Control loop. Interprets user command and does action*/
while (quit != 1 ) {
    key = _getch();
    if(key == 'w'){             //up
        nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
        movemode = 'w';
    }else if(key == 's'){       //down
        nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
        movemode = 's';
    }else if(key == 'd'){       //right
        nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
        movemode = 'd';
    }else if(key == 'a'){       //left
        nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
        movemode = 'a';
    }else if(key == 'q'){       //forward-left
        nxt.setJointSpeedRatios(0, speedRatio, 0.7*speedRatio);
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
        movemode = 'q';
    }else if(key == 'e'){       //forward-right
        nxt.setJointSpeedRatios(0, 0.7*speedRatio, speedRatio);
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
        movemode = 'e';
    }else if(key == 'x'){       //stop
        nxt.stopOneJoint(NXT_JOINTB);
        nxt.stopOneJoint(NXT_JOINTC);
        movemode = 'x';
    }else if(key == 'r'){       //quit
        printf("\nExiting program.\n");
        quit = 1;
    }else if(key == '1'){       //speedRatio 0.25
        speedRatio = 0.25;
        ungetch(movemode);
    }else if(key == '2'){       //speedRatio 0.50
        speedRatio = 0.50;
        ungetch(movemode);
    }else if(key == '3'){       //speedRatio 0.75
```

```
        speedRatio = 0.75;
        ungetch(movemode);
    }else if(key == '4'){       //speedRatio 1
        speedRatio = 1;
        ungetch(movemode);
    }else{
        printf("\nInvalid Input!\n");
    }
}
```

Program 10: `vehicle_rtc.ch` Source Code

Header files

Similar to any C program, you will have to include necessary header files, which is described in the first four lines.

```
#include <conio.h>
#include <stdio.h>
#include <nxt.h>
```

- The header `conio.h` provides a function for the program to detect a key press for the `-press a key-` command.
- The header `stdio.h` provides input and output function for the program. These input and output function allows the program to display output for the user or ask for the user input.
- The header `nxt.h` provides the program with general functions of the Ch Mindstorm Control Package.

Declaring variables

After including the headers, variables are declared.

```
ChNXT nxt;

double speedRaio = 0.25;//speed ratio of the motors. (default to 0.25)

int quit = 0,            //used by quit case to exit the loop
    status1,             //used to check for errors
    status2;             //used to check for errors

char key = 'x',          //stores the input from the user
     movemode = 'x';     //stores the last movement command
```

- The `ChNXT` class stores the connection status, sensor data, and motor counter data of the NXT. Also, the class includes the functions for controling the NXT.
- The double variable `speedRatio` stores the speed ratio of the motor.
- The integer variable `quit` is used to check if the user wants to quit the program.
- The integer variable `status1` and `status2` are used to check the sensor connection.
- The character variable `key` stores the input from the user.
- The character variable `movemode` stores the last command that the user used.

Checking connection

After declaring variables, the connection of the NXT needs to be checked. In the next 7 lines of the program, the program checks for the connection of the NXT to the computer. If the NXT connection fails, the program will quit.

```
/* Connect to NXT */
printf("Initializing vehicle and assuming control...");
if (nxt.connect()){
    printf("\nPress any key to exit.\n");
    while (!_kbhit()); //wait for keypress
        exit(-1);
}
```

User interface

Before the beginning of the real time control, the user must be able to know the function of the key they are pressing. To do this, the program print out a user interface for the user to read. In segment of the Program 12 shown below, the NXT vehicle RTC program used `printf()` command is used to display the user interface for the user to read.

```
/* GUI display */
printf("Vehicle Direction:  Other Commands:");
printf("\n    [w]              [x] Stop Motors");
printf("\n [q] ^ [e]           [r] Exit Program");
printf("\n    \|/              [1] Set Speed Ratio to 0.25");
printf("\n[a]<-|->[d]          [2] Set Speed Ratio to 0.50");
printf("\n     v               [3] Set Speed Ratio to 0.75");
printf("\n    [s]              [4] Set Speed Ratio to 1\n");
printf("\Please Enter command:");
```

Real time control

After completing the initiation of the code, which include adding header files, declaring variables, checking connection, and displaying the user interface, the real time control of the NXT begins with the robot control code block. In the robot control code block, a **while** loop is implemented to allow the user to control the NXT continuously until the program is terminated. Within the **while** loop, the program grabs an input from the user, and then decide what to do with the input using the `if-else` statement. A flowchart for the NXT RTC control program is shown in Figure 21.

The program fragment below shows the beginning and the end of the **while** loop for Program 10:

```
while (quit != 1 ) {
    key = _getch();
    if(key == 'w'){
        ...
    }else if(key == 's'){
        ...
    }else{
        ...
    }
}
```

When the program reaches this stage, the real time control begins. The **while** loop allows the program to keep asking the user's input until the `'r'` key is pressed. When the `'r'` key is pressed, the program will set `quit` variable is set to 1, which allows the program to exit out of the **while** loop.

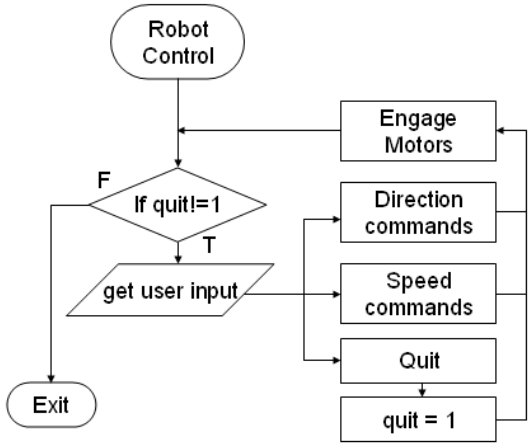

Figure 21: A flowchart for the NXT vehicle RTC Control Program.

Inside the first line of the **while** loop, the program use the `_getch()` command to obtain a userinput and store the user input to the variable `key`. After obtaining the user's input in a variable, the program use a `if-else` statement to check which key was pressed. Depending on what key was pressed, the program will run a fragment of code that sends commands to the NXT.

Directional commands

The movements are controlled using the 'w', 's', 'a', and 'd' format. As shown in Figure 20, the user interface used arrows to indicate the movement direction and associate each direction with a specific key. The available buttons for movements are 'w', 's', 'a', 'd', 'q', 'e', and 'x'.
When the key 'w' has been pressed, the `if-else` statement will run the codes for the case 'w'. The program fragment for case 'w' is shown below:

```
if(key == 'w'){//up
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
    movemode = 'w';
}
```

In case 'w', the program will run joints `NXT_JOINTB` and `NXT_JOINTC` forward at the velocity `speedRatio`. Next, 'w' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle. Basically, the case 'w' will move the NXT vehicle forward at velocity `speedRatio`.

When the key 's' has been pressed, the **if-else** statement will run the codes for the case 's'. The program fragment for case 's' is shown below:

```
else if(key == 's'){//down
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
    movemode = 's';
}
```

In case 's', the program will run the joints `NXT_JOINTB` and `NXT_JOINTC` backward at the velocity `speedRatio`. Next, 's' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle. Basically, the case 's' will move the NXT vehicle backward with velocity `speedRatio`.

When the key ‘a’ has been pressed, the **if-else** statement will run the codes for the case ‘a’. The program fragment for case ‘a’ is shown below:

```
else if(key == 'a'){//left
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
    movemode = 'a';
}
```

In case ‘a’, the program will actuate joint `NXT_JOINTB` forward at velocity `speedRatio` and actuate joint `NXT_JOINTC` backward at velocity `speedRatio`. Next, ‘a’ key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.
When the key ‘d’ has been pressed, the **if-else** statement will run the codes for the case ‘d’. The program fragment for case ‘d’ is shown below:

```
else if(key == 'd'){//right
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
    movemode = 'd';
}
```

In case ‘d’, the program will actuate joint `NXT_JOINTB` backward at velocity `speedRatio` and actuate joint `NXT_JOINTC` forward at velocity `speedRatio`. Next, ‘d’ key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.
When the key ‘q’ has been pressed, the **if-else** statement will run the codes for the case ‘q’. The program fragment for case ‘q’ is shown below:

```
else if(key == 'q'){//forward-left
    nxt.setJointSpeedRatios(0, speedRatio, 0.7*speedRatio);
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
    movemode = 'q';
}
```

In case ‘q’, the program will actuate joint `NXT_JOINTB` forward at velocity `speedRatio` and actuate joint `NXT_JOINTC` forward at velocity 0.7* `speedRatio`. Next, ‘q’ key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.
When the key ‘e’ has been pressed, the **if-else** statement will run the codes for the case ‘e’. The program fragment for case ‘e’ is shown below:

```
else if(key == 'e'){//forward-right
    nxt.setJointSpeedRatios(0, 0.7*speedRatio, speedRatio);
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
    movemode = 'e';
}
```

In case ‘e’, the program will actuate joint `NXT_JOINTB` forward at velocity 0.7*`speedRatio` and actuate joint `NXT_JOINTC` forward at velocity `speedRatio`. Next, ‘e’ key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.
When the key ‘x’ has been pressed, the **if-else** statement will run the codes for the case ‘x’. The program fragment for case ‘x’ is shown below:

```
else if(key == 'x'){//stop
    nxt.stopOneJoint(NXT_JOINTB);
    nxt.stopOneJoint(NXT_JOINTC);
    movemode = 'x';
}
```

In case 'x', the program will set the motor in `NXT_JOINTB` and `NXT_JOINTC` to zero velocity, and then set the motor to off idle mode. Next, 'x' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle. Basically, the case 'x' stops the motor and keep it turned off until another key is pressed.

Speed control

The speed of the motor is controlled by the number key '1', '2', '3', and '4'. In Figure 20, the user interface shows that each key has a specific speed ratio. For example, key '1' indicates 0.25 speed ratio, and key '2' indicate 0.50 speed ratio. As shown below, each of the key has its fragment of code.

```
else if(key == '1'){
    speedRatio = 0.25;
    ungetch(movemode);
}else if(key == '2'){
    speedRatio = 0.50;
    ungetch(movemode);
}else if(key == '3'){
    speedRatio = 0.75;
    ungetch(movemode);
}else if(key == '4'){
    speedRatio = 1;
    ungetch(movemode);
}
```

For each of the case, the fragment of code changes the variable `speedRatio` and performs an `ungetch()` command. The `ungetch()` command allows the program run the mode that it was previously saved in the variable `movemode` before the speed keys are pressed. Basically, the speed keys allow the program to change the speed of the NXT motor without changing the movement mode that it was in.

Functions for other keys

As mentioned before, the **while** loop allows the program to keep asking the user's input until the variable `quit` is set to 1. To quit the **while** loop, we must have a special case that sets the variable `quit` to 1. When the key 'r' is pressed, the **if-else** statement will perform a fragment of code for case 'r'. The program fragment for case 'r' is shown below:

```
else if(key == 'r'){//quit
    printf("\nExiting program.\n");
    quit = 1;
}
```

When the `'r'` key is pressed, the program will print out the statement 'exiting program' and set `quit` variable is set to 1, which allows the program to exit out of the **while** loop.

For the keys that is not specified to any cases, a default case is used for the time when the user input a wrong key. The program fragment for the default case is shown below:

```
else{
    printf("\nInvalid Input!\n");
}
```

This fragment prints 'Invalid Input!' to the user to indicate the key they just pressed is an invalid input.

5.5 Using NXT Sensors

Sensors convert a physical quantity and convert it to signals which can be read by the NXT or the computer. Sensors allow the communication between the outside environment to the NXT. The NXT is equipped with four sensor input ports and you can equip each port with a variety of different sensors. In this section, we are going to discuss about how to use the touch sensor and the ultrasonic sensor. In these discussion, two demonstration programs will be presented. Please make sure your NXT vehicle are configured according to Figurer 22 to run these demonstration programs.

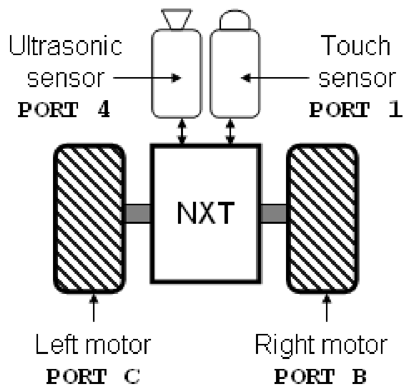

Figure 22: Sensor/Motor configuration of the NXT Vehicle

5.5.1 Using your touch sensor

After you have connected the NXT to your PC, you will need to set up the sensors if you would like to use in the NXT program. For example, if you want to add components like the light sensor, ultrasonic sensor, and/or touch sensor, you will need to set up their connection using the **setSensor()** function. The **setSensor()** function will return 1 if no connection is established, so you can terminate the program if a sensor connection is not established. An example connection check for adding the Touch and Ulstrasonic sensors to the NXT vehicle is as follow:

```
ChNXT nxt;

/* Check status of NXT connection */
if (nxt.connect()){
    exit(-1);
}

/* Setup sensors and check sensor connection */
int status1 = 2, status2 = 2;

/* Save status of NXT_SENSORPORT1, and NXT_SENSORPORT4 */
status1 = nxt.setSensor(NXT_SENSORPORT1,
            NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);

status2 = nxt.setSensor(NXT_SENSORPORT4,
            NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);

/* Check connection status of NXT_SENSORPORT1 */
if(status1) {
```

```
    exit(-1);
}

/* Check connection status of NXT_SENSORPORT4 */
if(status2) {
    exit(-1);
}
```

As in previous examples, the class **nxt**, is created, and the initial connection to the NXT is made. The next step is to initialize the NXT sensor ports to the correct types and ensure the sensors are working correctly. The variables `status1` and `status2` are used to check the return value of the function `setSensor()`.

A common application of touch sensors for a vehicle robot is for obstacle detection. In order to demonstrate the use of the touch sensor, consider the touch sensor demo in Program 11.

Source Code

```
/* File name: touchsensor.ch
 *
 * Demonstrate the CH Mindstorms Control Package's ability
 * to control the NXT Mindstorm to use the touch sensor. */

#include <nxt.h>

ChNXT nxt;
int status;
int touchValue;

/* Connect to NXT */
if (nxt.connect()) {
    printf("Fail to connect!\n");
    exit(-1);
}

/* Set sensor types */
status = nxt.setSensor(NXT_SENSORPORT1,
        NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);
if (status) {
    printf("Fail to setup sensors.\n");
    exit(-1);
}

/* set joint speed ratios */
nxt.setJointSpeedRatios(0, 0.25, 0.25);

/* Move Robot Forward */
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);

/* Commands: */
while (1) {
    /* Get touch sensor data */
    nxt.getSensor(NXT_SENSORPORT1, touchValue);
    /* If touch sensor is triggered */
    if (touchValue == 1) {
        /* Move backward */
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
```

```
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
        delay(2);
        /* quit the while loop */
        break;
    }
}

/* Stop the motors */
nxt.stopTwoJoints(NXT_JOINTB, NXT_JOINTC);
```

Program 11: `touchsensor.ch` Source Code

Checking touch sensor connection

Program 11 is similar to Program 8, the only change is the addition of the use of the touch sensor. Program 11 makes the NXT move forward until the touch sensor is triggered, after which it will back up and stop. One of the additions in Program 11 is the initialization of the touch sensor. The fragment of the initialization of the touch sensor is shown below.

```
/* Set sensor types */
int status;

status = nxt.setSensor(NXT_SENSORPORT1,
            NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);
if (status){
    printf("Failed to setup sensors\n");
    exit(-1);
}
```

In this fragment, the program use the `setSensor()` command to set the touch sensor to `NXT_SENSORPORT1`. The connection status between the NXT and the sensor is then returned in the variable called `status`. Next, the **if** statement check if the connection to the sensor is good. If the variable status is equal to 1, which means no sensor connection, the program will exit and the rest of the codes will not be executed.

Using while loop

A **while** loop is a common method that is used for sensor data gathering. For every iteration of the **while** loop, the program checks the data gathered by the touch sensor. After gathering the data, the program decide what to do with the data. A flowchart of the **while** loop of the touch sensor demo program is shown in Figure 23

In Program 11, the **while** loop checks for the data of the touch sensor. If the touch sensor is triggered, the data of the touch sensor will be set to 1 by the NXT. Then the program will move the NXT backward and disconnect the NXT. The **while** loop of the touch sensor demonstration program is described below:

```
while(1){
    /* Get touch sensor data and save into a variable*/
    nxt.getSensor(NXT_SENSORPORT1, touchValue);

    /* If touch sensor is triggered */
    if (touchValue == 1){
        /* Move backward */
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
        delay(2);
        /* quit the while loop */
```

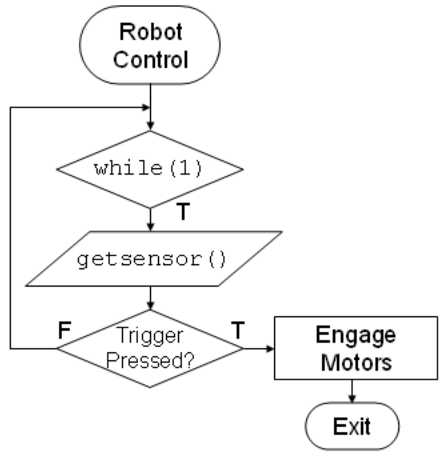

Figure 23: A flowchart of the **while** loop in Program 11

```
        break;
    }
}
```

In this **while** loop, the program uses the `getSensor()` command to get the data from `NXT_SENSORPORT1`, which is the port for the touch sensor according to Figure 22. Next, the program checks the touch sensor if it is pressed with the **if** statement. If the touch sensor has been triggered, the codes in the **if** statement will run. The codes inside the **if** statement is the same command for moving the NXT backward. After moving the NXT backward, the command break allows the program to exit out of the **while** loop. This **while** loop will never exit until the touch sensor is triggered and the break command is used.

5.5.2 Using your ultrasonic sensor

In order to demonstrate the use of the ultrasonic sensor, consider the ultrasonic sensor demo in Program 12. NOTE: If your robot gets stuck, put your hands in front of the ultrasonic sensor to quit the loop.

Source Code

```
/* File name: ultrasonicsensor.ch
 *
 * Demonstrate the Ch Mindstorms Control Package's ability
 * to control the NXT Mindstorm to use the ultrasonic sensor.*/

#include <nxt.h>

ChNXT nxt;
int status;
int ultraValue;
double speedRatio;

/* Connect to NXT */
if (nxt.connect()) {
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(-1);
```

```
}

/* Set sensor types */
status = nxt.setSensor(NXT_SENSORPORT4,
        NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);
if (status) {
    printf("Fail to setup sensors\n");
    exit(-1);
}

/* Commands: */
while (1) {
    /* get ultrasonic sensor data */
    nxt.getSensor(NXT_SENSORPORT4, ultraValue);

    /* If obstacle is really close */
    if (ultraValue < 20 && ultraValue >0) {
        speedRatio = 0.25;
        /* Move backward */
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
        sleep(3);
        /* Quit the while loop */
        break;
    }/* Else if the obstacle is close */
    else if (ultraValue < 40) {
        speedRatio = .5;
    }/* Else if the obstacle is not close */
    else if (ultraValue < 80) {
        speedRatio = .75;
    }/* Else if there is no obstacle in sight */
    else {
        speedRatio = 1.0;
    }
    /* Move forward (constantly)*/
    nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
}

/* Stop the motors */
nxt.stopTwoJoints(NXT_JOINTB, NXT_JOINTC);
```

Program 12: `ultrasonicsensor.ch` Source Code

Checking ultrasonic sensor connection

Like Program 11, Program 12 is meant to be a brief demonstration of one method of using an ultrasonic sensor with a vehicle NXT. Similar to the initialization of the touch sensor, the initialization of the ultrasonic sensor is shown below:

```
/* Set sensor types */
int status;

status = nxt.setSensor(NXT_SENSORPORT4,
            NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);

if (status){
```

```
    printf("Fail to setup sensors.\n");
    exit(-1);
}
```

In this program fragment, the `setSensor()` command sets the ultrasonic sensor to `NXT_SENSORPORT4`. Next, it checks the return value of the sensor connection status. The program will quit if the return value is 1, meaning there is no connection between the sensor and the NXT.

Contents in the while loop

In Program 12, the while loop uses the ultrasonic sensor to detect distances between the NXT and the obstacles in front of the NXT. The ultrasonic sensor will detect distances and the program code reacts to the data by slowing down or speeding up. The flowchart of the **while** loop of Program 12 is shown in Figure 24.

The **while** loop code block for Program 12 is shown below:

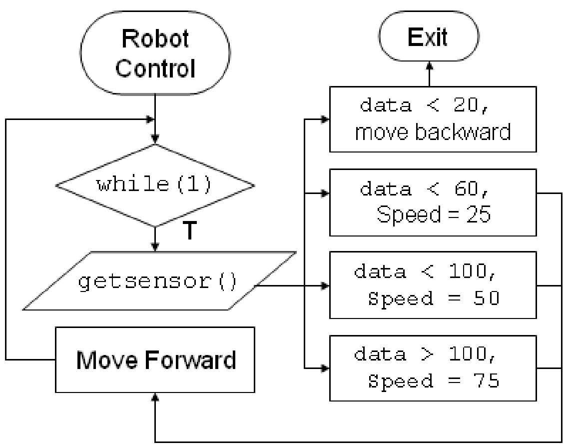

Figure 24: Flow Diagram of the while loop in Program 11

```
/* Commands: */
while(1){
    /* Get ultrasonic sensor data */
    nxt.getSensor(NXT_SENSORPORT4, ultrasonicValue);

    /* If obstacle is really close */
    if (ultrasonicValue < 20){
        speedRatio = 0.25;
        /* Move backward */
        nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
        nxt.vehicleRollBackward(360);
        /* Quit the while loop */
        break;
    }/* Else if the obstacle is close */
    else if(ultrasonicValue < 60){
        speedRatio = 0.25;
    }/* Else if the obstacle is not close */
    else if(ultrasonicValue < 100){
        speedRatio = 0.50;
    }/* Else if there is no obstacle in sight */
    else if(ultrasonicValue < 200){
```

```
        speedRatio = 0.75;
    }/* Sensor value larger than 200 */
    else{
        speedRatio = 0.75;
    }
    /* Move forward (constantly) */
    nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
}
```

Similar to Program 11, the **while** loop in Program 12 also gathers the sensor data using the `getSensor()` command to gather data from the ultrasonic sensor in `NXT_SENSORPORT4`. Next, **the if-else** statement block determine what to do depending on the distance data from ultrasonic sensor.

- If the sensor value is below 20, which means the vehicle is very close to an obstacle, the program tells the vehicle to reverse, and then break out of the **while** loop.
- If the sensor value is above 20 and below 60, which means the vehicle is close to an obstacle, the program sets the speed raio variable to 0.25.
- If the sensor value is above 60 and below 100, which means the vehicle is not close to an obstacle, the program sets the speed ratio variable to 0.50.
- If the sensor value is above 100 and below 200, which means there is nothing in front of the vehicle, the program sets the speed ratio variable to 0.75.
- If the sensor value is other value that is not mentioned above, the program sets the speed variable to 0.75.

After the **if-else** statement block, the program sets the robot to move forward with velocity set at the speed variable. Then the program returns back to the beginning of the **while** loop, which is to gather data from the sensor again.

5.5.3 Autonomous Control Program

In the previous sections, we thoroughly covered the manual real time control program, which allows you to remote control your NXT vehicle with your keyboard. In this section, we will talk about the autonomous control program for the NXT vehicle. In an autonomous control program, the robot must be able to move around by itself without human commands or interventions. In order to achieve such task, the NXT must be able to detect obstacles using its sensors and steer away from the obstacle using its actuators. A typical autonomous control scheme is to sense, plan, and act, which is shown in Figure 25.

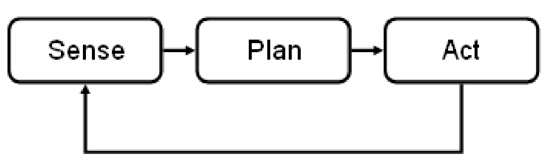

Figure 25: A diagram for sense plan act.

- Sense is to gather data from the robot's surrounding.

- Plan is to plan the interaction between the robot and its surrounding using gathered data.
- Act is to act with robot's surrounding.

The main difference between the manual RTC program and the autonomous program is the content inside the **while** loop. In the manual RTC program, the codes inside the **while** loop scan for user's input and the robot acts on the input that the user provided. In the autonomous program, the codes inside the **while** loop perform the sense-plan-act cycle similarly to the diagram shown in Figure 25. Every cycle, the information is gathered from the sensor and send back to the computer. The computer will decide what to do depending on the sensor data. The autonomous control program for the NXT vehicle is described in Program 13.

Source Code

```
/* File name: vehicle_auto.ch
 *
 * Demonstrate the CH Mindstorms Control Package's ability
 * to control the machine robot model autonomously, as well
 * as demonstrate how use sensor data from the NXT to controle it's actions. */

#include <conio.h>
#include <stdio.h>
#include <nxt.h>

ChNXT nxt;
double speedRatio = 0.25;       //speed ratio of the motors. (default to .25)
int status1 = 2;                //used to check for errors
int status2 = 2;                //used to check for errors
int touchValue, ultraValue;     //used to store sensor data
char key = 'x',                 //stoes user input
     movemode = 'x';            //stores last movement command

/* Connect to NXT */
printf("Initializing vehicle and assuming control...");
if (nxt.connect()){
    printf("\nPress any key to exit.\n");
    while (!_kbhit());          //wait for keypress
    exit(-1);
}

/* Set sensor types */
status1 = nxt.setSensor(NXT_SENSORPORT1,
        NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);
status2 = nxt.setSensor(NXT_SENSORPORT4,
        NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);
if ((status1) || (status2)){
    printf("\nError initializing sensors.\nPress any key to exit.\n");
    while (!_kbhit());          //wait for keypress
    exit(-1);
}

while (1) {
    /* check user input 'q' to quit */
    if (kbhit()){
        if (getch() == 'q'){
            printf("\nExiting.");
            break;
        }
```

```
    }

    /* get touch sensor. If pressed reverse and turn left */
    nxt.getSensor(NXT_SENSORPORT1, touchValue);
    if (touchValue == 1){
        nxt.moveJoint(NXT_JOINTB, 720);
        nxt.moveJoint(NXT_JOINTC, 720);
        delay(1);
        nxt.moveJoint(NXT_JOINTB, -720);
        nxt.moveJoint(NXT_JOINTC, 720);
    }

    /* get distance from UltraSonic sensor,
       set speed according to distance. Turn left if really close.*/
    nxt.getSensor(NXT_SENSORPORT4, ultraValue);
    if (ultraValue < 10){
        nxt.moveJoint(NXT_JOINTB, -720);
        nxt.moveJoint(NXT_JOINTC, -720);
        delay(1);
        nxt.moveJoint(NXT_JOINTB, 720);
        nxt.moveJoint(NXT_JOINTC, -720);
        delay(0.75);
    } else if (ultraValue < 20){
        speedRatio = 0.25;
    }else if (ultraValue < 40){
        speedRatio = 0.50;
    }else if (ultraValue< 80){
        speedRatio = 0.75;
    }else{
        speedRatio = 1;
    }

    /* Turn motors on (drive forward) */
    nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
}
```

Program 13: `vehicle_auto.ch` Source Code

In Program 13, the sensors that is used are the touch sensor and the ultra sonic sensor. These sensors are located in the front of the vehicle so that when the vehicle encounters an obstacle, the program will control the robot to avoid or steer away from it. A diagram of the vehicle and its sensor and actuators of the program is shown in Figure 22. Please make sure your NXT vehicle are configured according to to Figure 22 to run Program 13.

Exiting the while loop

In the autonomous program, there must be codes that allow the user to quit the autonomous program. Otherwise, the robot will roam forever until the batteries run out or until a deliberate shut down of the program. In the beginning of the **while** loop, the program checks for the user's input. If the user's input is 'q' to quit, then the program will break out of the **while** loop and safely disconnects the NXT. If the user's input is not 'q' or if the user did not input anything, the program will continue to the next section of the **while** loop. The program fragment for exiting the autonomous program is shown below.

```
/* check user input 'q' to quit */
if(kbhit()){
```

```
    if (getch()=='q'){
        printf("\nExiting.");
        break;
    }
}
```

In this program fragment, an if statement is used to check if a keyboard key has been hit. Next, if a keyboard key has been hit, another if statement checks if the input is 'q'. If both conditions are satisfied, the break statement will break out of the **while** loop of the program.

Touch sensor

The next section of the **while** loop uses the touch sensor to control the NXT vehicle. When the NXT comes into contact with some obstacle in the front, the touch sensor will be triggered. The autonomous program will notice that the touch sensor is triggered and command the NXT to steer away from the obstacle. The program fragment of the touch sensor is shown below.

```
/* get touch sensor. If pressed reverse and turn left */
nxt.getSensor(NXT_SENSORPORT1, touchValue);
if (touchValue == 1){
    nxt.vehicleRollBackward(180);
    nxt.vehicleRotateLeft(360);
}
```

In the first line of this fragment, the NXT gathers data from `NXT_SENSORPORT1`, which is the port for the touch sensor. Next, it checks the value for the touch sensor data with an **if** statement. If the value of the touch sensor data is less than 500, which means the touch sensor has been triggered, the program will execute the obstacle avoidance commands inside the **if** statement. The commands in the **if** statement control the NXT vehicle to reverse, then stop, and then steer left.

Ultrasonic sensor

The next part of the **while** loop uses the ultrasonic sensor to control the speed of the NXT vehicle. The ultrasonic sensor is used to detects the distance between itself to an incoming obstacle. The distance between the ultrasonic sensor and the incoming obstacle will tell the vehicle if it should slow down or speed up. For example, if the sensor senses nothing in front of the vehicle, the program will tell the vehicle to speed up; and if the sensor senses there is an obstacle in front, the program will tell the vehicle to slow down. The program fragment of the ultrasonic sensor is shown below.

```
/*
 * get distance from UltraSonic sensor,
 * set speedRatio according to distance. Turn left
 * if really close.
 */

nxt.getSensor(NXT_SENSORPORT4, ultrasonicValue);
if(ultrasonicValue < 10){
    nxt.vehicleRollBackward(360);
    nxt.vehicleRotateRight(180);
    speedRatio=0;
} else if (ultraValue < 20)
    speedRatio = 0.25;
else if (ultraValue < 40)
    speedRatio = 0.50;
else if (ultraValue < 80)
    speedRatio = 0.75;
else
    speedRatio = 1.0;
```

In the first line of this fragment, the NXT gathers data from `NXT_SENSORPORT4`, which is the port for the ultrasonic sensor. Afterwards, there is a block of **if-else** statement to determine what speed is used for the sensor data gathered. The **if-else** block changes the speed variable of the vehicle depending on the ultrasonic sensor value. sensor value threshold and its commands, which are the same as described in Section 5.5.2.

Running forward

The autonomous program does not work if the robot is stationary. The last portion of the **while** loop sets the robot to be running forward if it is not performing other tasks. The program fragment for running forward is shown below.

```
/* Turn motors on (drive forward) */
nxt.setJointSpeedRatios(0, speedRatio, speedRatio);
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
```

This program fragment commands the vehicle to move forward continuously by setting both motors rotating positively at the variable `speedRatio`.

6 Controlling Non-Vehicle NXT Robots

Previously, the focus has been on controlling a vehicle NXT design. Ch Mindstorms NXT Control Package can also be used to control alternate NXT robot configurations. The following sections demonstrates Ch code that controls the Lego Machine NXT Robot and the Lego Humanoid Robot. These examples should give you a sufficient background using Ch to program the NXT to create codes for any Lego NXT creation you may make.

6.1 Controlling NXT Machine

Figure 26: NXT Machine

The NXT mindstorm also comes with two other forms, one of these forms is the machine form as shown in Figure 26. In this section, the NXT machine form and the NXT machine demo program will be discussed. Compared to the NXT vehicle, the NXT machine uses three motors to manipulate its arm and two sensors for detection. The location and description of the components of the NXT machine is shown in Figure 27.

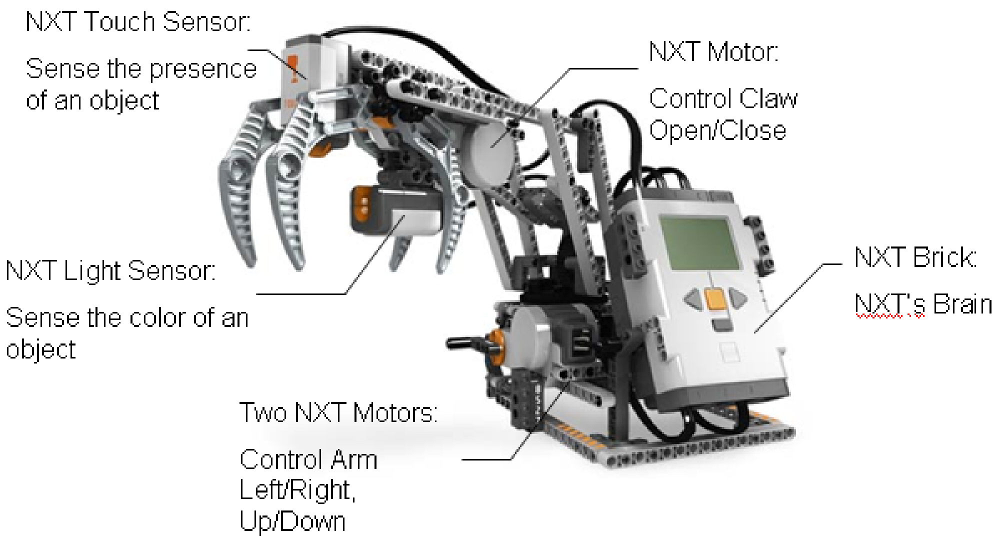

Figure 27: Components of the NXT Machine

As shown in Figure 27, one of its motor is responsible for moving its arm left and right. Another motor is responsible for moving its arm up and down. The last motor is responsible for controlling its claws open and close. There are two sensors mounted on the claw, they are the light sensor and the touch sensor. The NXT machine uses the light sensor to sense the color of the object it is handling. The NXT uses the touch sensor to sense if it has successfully grabbed an object. Please use the sensor/motor port configuration shown in Figure 28 for the Ch NXT machine demos programs described in later in this section.

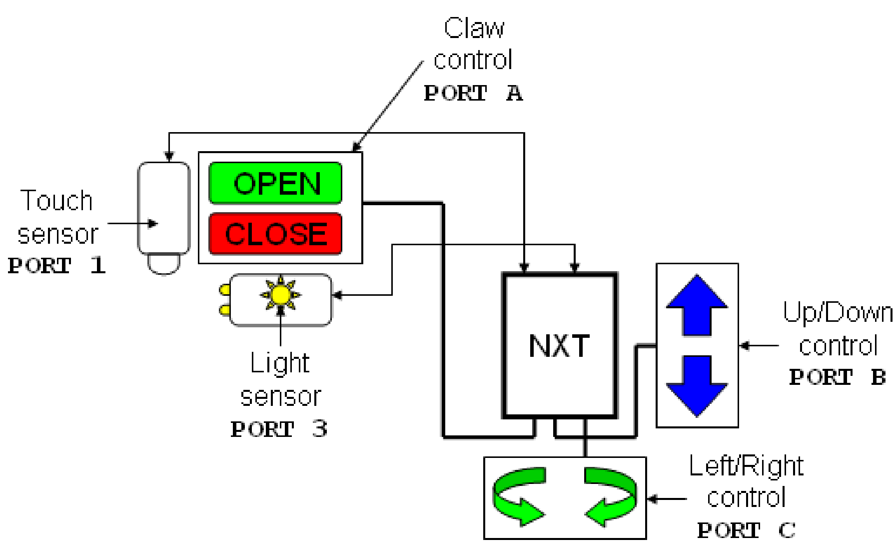

Figure 28: Sensor/Motor configuration of the NXT Machine

6.1.1 Manual Real Time Control Program

In this section the manual real time control for the NXT machine will be introduced and described. The manual RTC for the NXT machine allows the user to control the NXT machine manually via the keyboard. The user interface of the RTC program for the NXT machine is shown in Figure 29

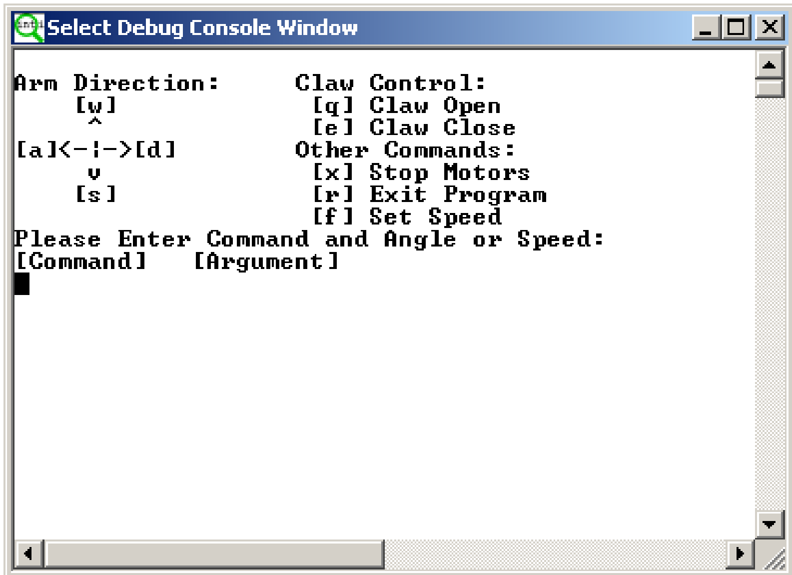

Figure 29: NXT vehicle RTC User Interface

The user interface and the user commands for the NXT machine RTC program is a bit different compared to the NXT vehicle RTC program. Instead of controlling the direction by pressing one key, the user will be required to press a key for a direction or command, and then enter a number to set the angle or speed. When a specific key is pressed during the execution of the NXT machine RTC program, the program uses an `if-else` statement to perform a fragment of code that sends commands to the NXT to move in a direction or perform a task. If a directional key or a set speed key has been pressed, the program will ask the user to input a number to set the arm to move at the specified angle or speed. If a discrete task key is pressed, like open or close claw, the program will not ask for a user input for a number. The list below is the list of commands and a short description of each commands:

- The key `"w"` is to control the NXT arm to move up.
- The key `"s"` is to control the NXT arm to move down.
- The key `"a"` is to control the NXT arm to turn left.
- The key `"d"` is to control the NXT arm to turn right.
- The key `"q"` is to control the NXT claw to open.
- The key `"e"` is to control the NXT claw to close.
- The key `"x"` is to stop the NXT motors.
- The key `"r"` is to exit the manual RTC program.
- The key `"f"` is to set the NXT motor speed.

The NXT machine RTC program is described in Program 14. In the rest of this section, we are going to explain important parts of the manual rtc program in detail.

Source Code

```
/* File name: machine_rtc.ch
 *
 * Demonstrate the CH Mindstorms Control Package's ability
 * to control the machine robot model, as well as demonstrate
 * how to set up and get sensor data. */

#include <conio.h>
#include <stdio.h>
#include <nxt.h>

ChNXT nxt;
double speedRatio = 1;          //used to control the motor speedRatio.
int angle = 0;                  //stores the angle input from the user.
int quit = 0;                   //used to break out of the control loop
int status = 0;
int touchValue, lightValue;
double gearratio = (8.0 / 56) * (1.0 / 24);//gear ratio on the arm
char dir = 0;                   //stores "direction to move" input from user.
char color[5];
char temp[20];
char *temp_loc;
printf("gear ratio: %f", gearratio);

/* call nxt_connect function and check for success */
printf("\nInitializing arm and assuming control...");
status = nxt.connect();
if (status) {
    while (!_kbhit());          //wait for key press
    nxt.disconnect();           //stop interfacing. This also stops the motors.
    exit(0);
}

/* Initialize sensor and check for success*/
status = nxt.setSensor(NXT_SENSORPORT1,
        NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);
if (status) {
    printf("\nSensor Setup failed. Exiting program.");
    while (!_kbhit());          //wait for key press
    nxt.disconnect();           //stop interfacing. This also stops the motors.
    exit(0);
}

status = nxt.setSensor(NXT_SENSORPORT3,
        NXT_SENSORTYPE_LIGHT_ACTIVE, NXT_SENSORMODE_RAWMODE);
if (status) {
    printf("\nSensor Setup failed. Exiting program.");
    while (!_kbhit());          //wait for key press
    nxt.disconnect();           //stop interfacing. This also stops the motors.
    exit(0);
}

/*This is the user input loop, that gets the user input and
sends the commands to the NXT accordingly.
w,s move arm up and down
a,d move arm left and right
q,e open and close the claw
x stops the motor
```

```
r quits the program
f sets the speedRatio (default to 1)    */

printf("\nArm Direction:     Claw Control:\n");
printf("    [w]              [q] Claw Open\n");
printf("     ^               [e] Claw Close\n");
printf("[a]<-|->[d]         Other Commands:\n");
printf("     v               [x] Stop Motors\n");
printf("    [s]              [r] Exit Program\n");
printf("                     [f] Set Speed\n");
printf("Please Enter Command and Angle or Speed:\n");
printf("[Command]   [Argument]\n");

while (quit != 1){
    printf("\nEnter command: ");
    dir = getche();
    if ((dir == 'w') || (dir == 'a') || (dir == 's') ||
            (dir == 'd')) {
        printf("  Enter angle: ");
        scanf("%d", &angle);
    }
    if(dir == 'f'){
        printf("   Enter speed ratio:");
        scanf("%lf", &speedRatio);
    }

    if(dir == 'a'){             //Arm rotate left.
        nxt.moveJoint(NXT_JOINTC, angle / gearratio);
    }else if(dir == 'd'){       //Arm rotate right.
        nxt.moveJoint(NXT_JOINTC, -angle / gearratio);
    }else if(dir == 'w'){
        nxt.moveJoint(NXT_JOINTB, angle / gearratio);
    }else if(dir == 's'){       //lowdder arm down
        nxt.moveJoint(NXT_JOINTB, -angle / gearratio);
    }else if(dir == 'q'){       //claw open
        nxt.moveJointContinuousNB(NXT_JOINTA, NXT_BACKWARD);
        delay(1);
        nxt.stopOneJoint(NXT_JOINTA);
    }else if(dir == 'e'){       //claw close
        nxt.moveJointContinuousNB(NXT_JOINTA, NXT_FORWARD);
        delay(1);
        nxt.stopOneJoint(NXT_JOINTA);
    }else if(dir == 'x'){       //stop
        nxt.stopAllJoints();
    }else if(dir == 'r'){       //quit
        printf("\nQuit.");
            quit = 1;
    }else if(dir == 'f'){
        nxt.setJointSpeeds(speedRatio, speedRatio, speedRatio);
        printf("\nSpeed ratio set to %d.", speedRatio);
    }else
        printf("\n");

    delay(0.2);
    nxt.getSensor(NXT_SENSORPORT1, touchValue);
    if (touchValue == 1) {
        printf("    The Ball was grabbed ");
        nxt.getSensor(NXT_SENSORPORT3, lightValue);
        if (lightValue < 50) {
```

```
            printf("and the color is red\n");
        }else {
            printf("and the color is blue\n");
        }
    }
}
```

Program 14: `machine_rtc.ch` Source Code

How does it work?

The initialization and the termination of the NXT machine RTC program is very similar to the NXT vehicle RTC program. The biggest difference between the two programs is in the **while** loop of the program. In this section, we will focus on how the **while()** loop of the NXT machine RTC program work internally.

While loop

Similar to the **while** loop of the NXT vehicle RTC program, the **while** loop of the NXT machine RTC program scans for the variable quit to see if it is set to 1. If the 'r' key has been pressed, the variable quit will be set to 1 and the **while** loop and the machine RTC program will be terminated. In the **while** loop, the user will be required to enter different types of command. Some commands are directional command, where the user needs to enter a number after the command. Some commands are discrete command, where the user does not need to enter another number. Instead of simply scanning for a key, the **while** loop has an additional if statement that scans for which key was pressed. If the key is a directional key, the program ask for the user to input an angle using the `scanf()` command. The fragment of the **while** loop is shown below:

```
while (quit != 1 ) {
    printf("\nEnter command: ");
    dir = _getch();
    if ((dir == 'w') || (dir == 'a') || (dir == 's')
            || (dir == 'd')){
        printf("  Enter angle: ");
        scanf("%d", &angle);
    }
    if(key == 'a'){
        ...
    }else if(key == 'd'){
        ...
    }else{
        ...
    }
}
```

Depending on what key was pressed, the program will run a fragment of code that sends commands to the NXT using the **if-else** command.

Directional commands

The directional movements are controlled using the 'w', 's', 'a', and 'd' keys. In Figure 29, the user interface used arrows to indicate the movement direction and associate each direction with a specific key.

When the key 'a' has been pressed, the **if-else** statement will run the codes for the case 'a'. The program fragment for case 'a' is shown below:

```
if(key == 'a'){
    nxt.moveJoint(NXT_JOINTC, angle / gearratio);
}
```

In case ‘a’, the program will run the joint `NXT_JOINTC` at velocity `speedRatio`, and to an `angle` divided by the gear ratio that the user has entered. Basically, for case ‘a’ the program will rotate the NXT machine arm left to an adjusted angle that the user has entered.
When the key ‘d’ has been pressed, the **if-else** statement will run the codes for the case ‘d’. The program fragment for case ‘d’ is shown below:

```
else if(key == 'd'){
    nxt.moveJoint(NXT_JOINTC, -angle / gearratio);
}
```

In case ‘d’, the program will run the joint `NXT_JOINTC` at velocity `-speedRatio`, and to an `angle` divided by the gear ratio that the user has entered. Basically, for case ‘d’ the program will rotate the NXT machine arm right to an adjusted angle that the user has entered.
When the key ‘w’ has been pressed, the **if-else** statement will run the codes for the case ‘w’. The program fragment for case ‘w’ is shown below:

```
else if(key == 'w'){
    nxt.moveJoint(NXT_JOINTB, angle);
}
```

In case ‘w’, the program will run the joint `NXT_JOINTB` at velocity `speedRatio`, and to a prescribed `angle` that the user has entered. Basically, for case ‘w’ the program will move the NXT machine arm upward to a prescribed angle that the user entered to an angle at a set speed ratio.
When the key ‘s’ has been pressed, the **if-else** statement will run the codes for the case ‘s’. The program fragment for case ‘s’ is shown below:

```
else if(key == 's'){
    nxt.moveJoint(NXT_JOINTB, -angle);
}
```

In case ‘s’, the program will run the joint `NXT_JOINTB` at velocity `-speedRatio`, and to a prescribed `angle` that the user has entered. Basically, for case ‘s’ the program will move the NXT machine arm downward to a prescribed angle that the user entered to an angle at a set speed ratio.

Discrete commands

The discrete commands are commands that performs a specific task that can either be on or off. For example, open or close the claw, turn on or off the motors, or quit the program. For this program, the discrete commands does not require another parameter, so the user does not need to input another number for using these commands. These commands are accessed by entering the ‘q’, ‘e’, ‘x’, ‘r’ keys.
When the key ‘q’ has been pressed, the **if-else** statement will run the codes for the case ‘q’. The program fragment for case ‘q’ is shown below:

```
else if(key == 'q'){
    nxt.moveJointContinuousNB(NXT_JOINTA, NXT_BACKWARD);
    delay(1);
    nxt.stopOneJoint(NXT_JOINTA);
}
```

In case ‘q’, the program will open the machine claw. First, the program will run the joint `NXT_JOINTA` at velocity `speedRatio` for 1 seconds. Next, the program will hold the position of the joint at that spot, thus keeping the machine claw open.

In case 'e', the program will close the machine claw. The program will run the joint `NXT_JOINTA` at velocity `speedRatio` for 1 seconds. Next, the program will hold the position of the motor at that spot, thus keeping the machine claw close.

When the key 'e' has been pressed, the **if-else** statement will run the codes for the case 'e'. The program fragment for case 'e' is shown below:

```
else if(key == 'e'){
    nxt.moveJointContinuousNB(NXT_JOINTA, NXT_FORWARD);
    delay(1);
    nxt.stopOneJoint(NXT_JOINTA);
}
```

In case 'x', the program will stop all the motors, and turn the mode to off. When the key 'x' has been pressed, the **if-else** statement will run the codes for the case 'x'. The program fragment for case 'x' is shown below:

```
else if(key == 'x'){
    nxt.stopAllJoints();
}
```

In case 'r', the program will print the string "Quit." and set the variable `quit` to 1. By setting the variable `quit` to 1, the **while** loop will be terminated, thus quitting the program. When the key 'r' has been pressed, the **if-else** statement will run the codes for the case 'r'. The program fragment for case 'r' is shown below:

```
else if(key == 'r'){
    printf("\nQuit.");
    quit = 1;
}
```

Speed ratio setup

To set the speed ratio of the movement of the arm, the user can enter the key 'f' and enter the speed ratio. The speed ratio of joints are in the range of 0 to 1. The program fragment of case 'f' is shown below:

```
else if(key == 'f'){
    printf("    Enter the speed ratio (0 to 1):");
    scanf("%lf", &speedRatio);
    printf("\nSpeed ratio set to %lf.", speedRatio);
}
```

Using the sensors

After the switch cases, the **while** loop will use the sensors to detect if the claw has grabbed an object or not. If the object has been detected, the program will also try to determine the color of the object using its light sensor In this program, the object that is grabbed is assumed to be a ball, and the color of the ball is assumed to be red or blue. The program fragment for this task is described below:

```
delay(0.2);
nxt.getSensor(NXT_SENSORPORT1, touchValue);
if (touchValue == 1){
    printf("    The Ball was grabbed");
    nxt.getSensor(NXT_SENSORPORT3, colorValue);
    if (colorValue < 50){
```

```
        printf("and the color is red\n");
    }else{
        printf("and the color is blue\n");
    }
}
```

In this fragment, the program will delay for 0.2 seconds and then grab the sensor value stored in `NXT_SENSORPORT1` using the `getSensor()` command, which is the touch sensor. Afterward, the if statement checks if the claw has grabbed an object. If the sensor value for the touch sensor is greater than 50, the touch sensor is not triggered, so there is the NXT detected that there is no object in its claw. If the sensor value for the touch sensor is 1, the touch sensor is triggered and the NXT detected that the claw has grabbed an object.

When the touch sensor is trigged, the program will continue in the **if** statement, and the program will prints out that "The Ball was grabbed" in the screen and it get the sensor value stored in `NXT_SENSORPORT3`, which is the light sensor. Next, the program will determine the color of the ball using the light sensor value. If the light sensor value is less than 50, the NXT will detect that the ball that was grabbed is red and program will print out "and the color is red". If the light sensor value is greater than 50, the NXT will detect that the ball that was grabbed is blue and program will print out "and the color is blue".

6.1.2 Autonomous Control Program

In this section, the autonomous control program for the NXT machine will be introduced. This autonomous program uses the NXT machine arm to scan it's surrounding. It performs this task by rotating its arm by an angle step and collecting distance data with an ultrasonic sensor as it is rotating. At the end of the program, the collected data will be stored in a data file called 'output.csv', and a polar diagram of the data will be display for the user. The NXT machine automatic control program is presented in Program 15. In the rest of this section, we are going to explain important parts of the automatic control program in detail.

Source Code

```
/* File name: machine_auto.ch
 *
 * Demonstrate the CH Mindstorms Control Package's ability
 * to control the machine robot model autonomously, as well
 * as demonstrate how to collect and plot sensor data from the NXT.*/

#include <conio.h>
#include <stdio.h>
#include <chplot.h>
#include <nxt.h>

CPlot plot;
ChNXT nxt;
double speedRatio = 0.30;
int quit = 0,                   //used to exit for loop
    i,                          //counter variable
    status;                     //stores status of function
int ultraValue, position;
double gearratio = (8.0 / 56) * (1.0 / 24);
enum {numpoints = 20};          //desired number of data points
const int anglestep = 2;        //angle moved between steps
double angle[numpoints];        //angle calculated from the tachometer
```

```
double distance[numpoints];      //data received from the ultrasonic sensor

/* Connect to NXT exit if failure */
if (nxt.connect()) {
    printf("\nPress any key to quit.");
    while (!_kbhit());           //wait for key press
    exit(-1);
}

/* Initialize arm. (Set sensor types and initialize variables) */
printf("\nInitializing arm for autonomous control...\n");
status = nxt.setSensor(NXT_SENSORPORT4,
        NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);
if (status) {
    printf("\nSensor Setup failed. Exiting program.");
    while (!_kbhit());           //wait for key press
    nxt.disconnect();            //stop interfacing. This also stops the motors.
    exit(-1);
}

for (i = 0; i < numpoints; i++) {
    angle[i] = 0;
    distance[i] = 0;
}

/* print usage information to the user*/
printf("\n%d Data points will be collected with a"
       "step size of %d.", numpoints, anglestep);
printf("\nPlease ensure that the arm can rotate"
       "%d degrees from its current position.",
       (numpoints*anglestep));
printf("\nPress any key to continue. Press q at any time to quit.");

if (getch() == 'q') {
    printf("\nQuitting program.");
    delay(1.5);
    exit(0);
}

/* begin Autonomous loop*/
for (i = 0; i < numpoints; i++) {
    printf("hello\n");
    /* get sensor data, if success print data, else print error*/
    if ((nxt.getSensor(NXT_SENSORPORT4, ultraValue)) == 0) {
        distance[i] = ultraValue;
        if ((nxt.getJointAngle(NXT_JOINTA, angle[i])) == 0) {
            printf("\nSample: %d,  distance: %d,  Angle: %lf",
                   i, distance[i], angle[i]);
        }
    }else
        printf("\nError!");

    /* check if q was pressed and if so exit program*/
    if (!_kbhit) {
        if (getch() == 'q') {
            printf("\nQuitting program.");
            break;
        }
    }
```

```
    /* rotate arm by anglestep (rotate motor anglestep/gear ratio)*/
    nxt.moveJoint(NXT_JOINTA, anglestep / gearratio);
    delay(1);
}

/* Stop interfacing. This also stops the motors and sensors.*/
nxt.disconnect();
printf("\n");

/* plot data in Ch */
plot.polarPlot(PLOT_ANGLE_DEG);
plot.data2DCurve(angle, distance, numpoints);
plot.sizeRatio(1);
plot.grid(PLOT_ON);
plot.plotting();
```

Program 15: `machine_auto.ch` Source Code

Please use the Figure 30 to connect your NXT devices for the autonomous control program.

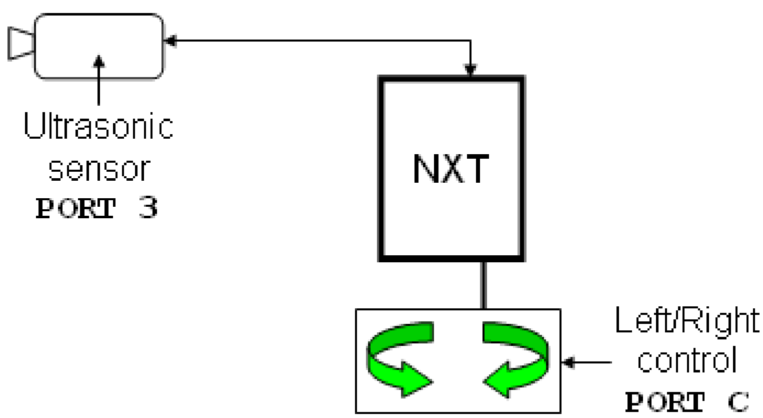

Figure 30: NXT vehicle RTC User Interface

How does it work?

In this autonomous program, a **for** loop is used instead of a **while** loop. The **for** loop will collect data, print out data, and rotate the arm at an angle step for every loop. Some of the parameters are hardcoded in the program, for example, number of loops and angle steps, so the user cannot change them as the autonomous program is executed. These parameters are shown in the program fragment below:

```
enum {numpoints=90};     //desired number of data points
const int anglestep=2;  //angle moved between steps
double angle[numpoints];//angle calculated from the tachometer
int distance[numpoints];//data received from the ultrasonic sensor
```

The variable **numpoints** determines how many number of times the **for** loop will run and the variable **angles** determines how much the arm rotates in degrees of angle.

Another feature this program has is the usage information printout described in the program fragment below.

```
printf("\n%d Data points will be collected with a",
        "step size of %d.", numpoints, anglestep);

printf("\nPlease ensure that the arm can rotat",
        "%d degrees from its current position.",(numpoints*anglestep));

printf("\nPress any key to continue. Press q at any time to quit.");

if(getch()=='q'){
    printf("\nQuitting program.");
    delay(1.5);
    exit(-1);
}
```

In this program fragment, the program will print out how many data point will be collected with the angle step size. Next, it will calculate and print out the full rotation angle of the robot arm and ask the user to ensure that the arm can rotate that amount. Lastly, the program asks the user to continue or quit the program.

For loop

The **for** loop is the main part of the autonomous program for the NXT machine. The **for** loop uses a series of if-else statements to perform data collection and arm movement. There are three sections for this **for** loop. The first section is to collect, store, and print ultrasonic data and angle rotation data. The second section scans for user input to see if the user has pressed ‘q’ to quit the program. The last section is to rotate the arm by an angle step. The program fragment below is the **for** loop for the autonomous program for the NXT machine.

```
for(i=0;i<numpoints;i++){
    /* get sensor data, if success print data, else print error*/
    if((nxt.getSensor(NXT_SENSORPORT4, ultraValue))){
        distance[i]= ultraValue;
        if (nxt.getJointAngle(NXT_SENSORPORT3, angle[i] == 0)){
            printf("\nSample: %d,  distance: %d,  Angle: %lf",
                        i, distance[i],angle[i]);
        }
    }else
        printf("\nError!");

    /* check if q was pressed and if so exit program */
    if (!_kbhit){
        if(getch()=='q'){
            printf("\nQuitting program.");
            break;
        }
    }

    /* rotate arm by anglestep (rotate motor
       anglestep/gear ratio)*/
    nxt.moveJoint(NXT_JOINTC, anglestep/gearratio);
    delay(1);
}
```

The first section begins by getting the ultrasonic sensor data and storing it in an array. If the program is able to retrieve the data, the program will also get the data from the tachometer and convert it to angle. The calculated angle will be stored in another array. Next, the program will print the sample number, distance detected by ultrasonic sensor, and angle rotated by the motor. If the program is unable to retrieve the data, the program will print an error message.

In the next section, the program checks if a key has been hit by the user, if no key has been hit, this section is skipped. If a key has been hit and it happened to be the 'q' key, the program will print 'Quitting program' and the program will be aborted.

In the third section, the program controls the motor to rotate at a given angle using the `moveJoint()` command. Lastly, the program pauses for 1 second by using the `delay()` command.

Plotting data

In addition to creating and storing a data file, the program also plots a polar diagram for the user to visualize the data. The program fragment below are the commands for plotting the data in a polar diagram.

```
/* plot data in Ch */
plot.polarPlot(PLOT_ANGLE_DEG);
plot.data2DCurve(angle, distance, numpoints);
plot.sizeRatio(1);
plot.grid(PLOT_ON);
plot.plotting();
```

In this fragment, the program uses the CPlot class commands to do its plotting. First, the program use the function `polarPlot()` to set the plot to polar and degrees in angle. Next, the program use the function `data2DCurve()` to insert the collected data onto the polar plot. Afterward, the program uses the function `sizeRatio()` function and `grid()` command to correct the size of the plot and to add grid to the plot. Finally, the program creates the plot by using the function `plotting()`. After executing this program, we will get a plot as the following.

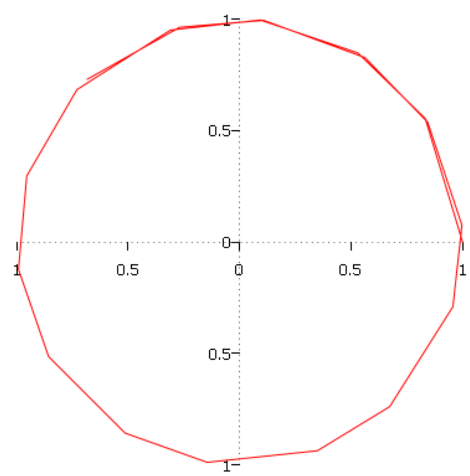

Figure 31: Plot collected data in Ch

6.2 Controlling NXT Humanoid

Figure 32: NXT Humanoid

The third form of the NXT mindstorm is a humanoid robot as shown in Figure 32. The NXT humanoid uses two of its motors to perform the walking motion. Also, the NXT humanoid uses its last motor to control the rotation of its head. The NXT humanoid is equipped with four sensors, a sound sensor on its right hand, a touch sensor on its left hand, a light sensor in the back, and an ultrasonic sensor on its head. In the next section, the real time control program for the NXT humanoid will be discussed. Please configure your NXT sensors and motors according to Figure 33.

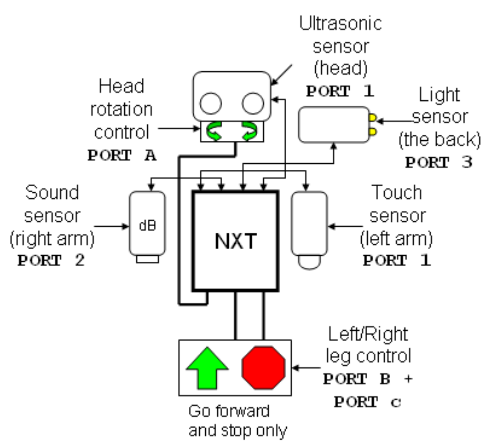

Figure 33: Sensor/Motor configuration of the NXT Humanoid

6.2.1 Manual Real Time Control Program

The real time control program of the NXT humanoid is similar to the real time control program of the NXT vehicle. The RTC program of the NXT humanoid allows the user to control the robot's leg movement and head rotation using the keyboard. In addition, the RTC program allow the user to print out data that has been collected by the NXT sensor. Figure 34 shows the user interface of the NXT humanoid. The list below is the list of commands:

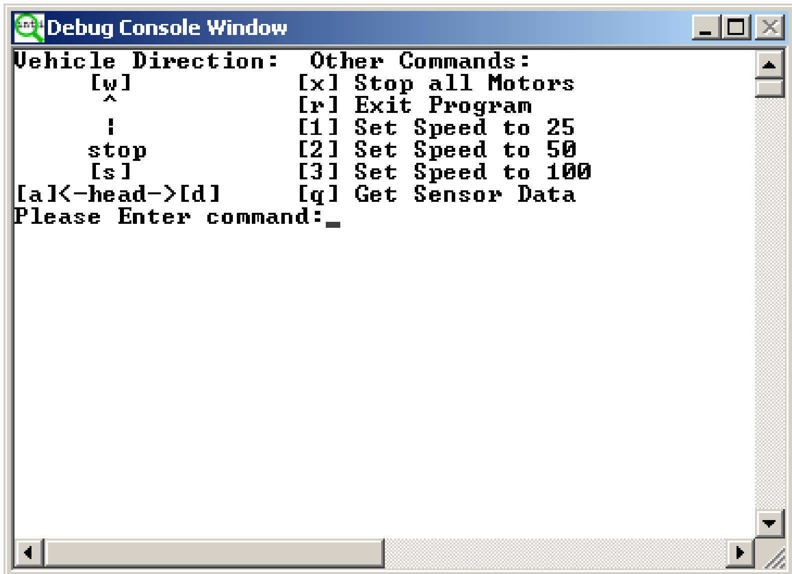

Figure 34: NXT Humanoid RTC User Interface

- The key `"w"`is to control the NXT humanoid to walk forward.
- The key `"s"` is to control the NXT humanoid to stop.
- The key `"a"` is to control the NXT humanoid head to turn left.
- The key `"d"` is to control the NXT humanoid head to turn right.
- The key `"q"` is to print out sensor data.
- The key `"x"` is to stop all of the NXT motors.
- The key `"r"` is to exit the RTC program.
- The number keys are used to set the NXT motor speed.

The NXT humanoid real time control program is described in Program 16. In the rest of this section, we are going to explain important parts of the manual RTC program in detail.

Source Code

```
/* File name: humanoid_rtc.ch
 *
 * Demonstrate the CH Mindstorms Control Package's ability
 * to control the Humanoid Robot Model, as well as demonstrate
 * how to get sensor data. */
```

```
#include <conio.h>
#include <stdio.h>
#include <nxt.h>

ChNXT nxt;
double speedRatio = 0.25;//speedRatio of the motors. (default to 25)
int quit = 0,            //used by quit case to exit the loop
    status1,             //used to check for errors
    status2,             //used to check for errors
    status3,             //used to check for errors
    status4;             //used to check for errors
char key = 'x',          //stores the input from the user
     movemode = 'x';     //stores the last movement command

/* Connect to NXT */
printf("Initializing vehicle and assuming control...");
if (nxt.connect()) {
    printf("\nPress and key to exit.\n");
    while (!_kbhit());   //wait for keypress
    exit(0);
}

/* Set sensor types */
status1 = nxt.setSensor(NXT_SENSORPORT1,
        NXT_SENSORTYPE_TOUCH, NXT_SENSORMODE_BOOLEANMODE);
status2 = nxt.setSensor(NXT_SENSORPORT2,
        NXT_SENSORTYPE_SOUND_DB, NXT_SENSORMODE_RAWMODE);
status3 = nxt.setSensor(NXT_SENSORPORT3,
        NXT_SENSORTYPE_LIGHT_INACTIVE, NXT_SENSORMODE_RAWMODE);
status4 = nxt.setSensor(NXT_SENSORPORT4,
        NXT_SENSORTYPE_ULTRASONIC, NXT_SENSORMODE_RAWMODE);
if ((status1) || (status2) || (status3)
        || (status4)) {
    printf("\nError initializing sensors.\nPress any key to exit.\n");
    while (!_kbhit());   //wait for key press
    exit(0);
}

/* GUI display */
printf("Vehicle Direction:  Other Commands:");
printf("\n     [w]           [x] Stop all Motors");
printf("\n      ^            [r] Exit Program");
printf("\n      |            [1] Set Speed Ratio to 0.25");
printf("\n     stop          [2] Set Speed Ratio to 0.50");
printf("\n     [s]           [3] Set Speed Ratio to 0.75");
printf("\n[a]<-head->[d]     [q] Get Sensor Data\n");
printf("Please Enter command:");

/* Control loop. Interprets user command and does action*/
while (quit != 1){
    nxt.setJointSpeedRatios(0.3, speedRatio, speedRatio);
    key = _getch();
    if(key == 'w'){      //up
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
        movemode = 'w';
    }else if(key =='s'){ //down
        nxt.moveJointContinuousNB(NXT_JOINTB, NXT_BACKWARD);
```

```
        nxt.moveJointContinuousNB(NXT_JOINTC, NXT_BACKWARD);
        movemode = 's';
    }else if(key == 'd'){//right
        nxt.moveJointContinuousNB(NXT_JOINTA, NXT_FORWARD);
    }else if(key == 'a'){//left
        nxt.moveJointContinuousNB(NXT_JOINTA, NXT_BACKWARD);
    }else if(key == 'q'){//print sensor
        printSensor(&nxt);
    }else if(key == 'x'){//stop
        nxt.stopAllJoints();
        movemode = 'x';
    }else if(key == 'r'){//quit
        printf("\nExiting program.\nPress any key to exit.");
        quit = 1;
    }else if(key == '1'){//speedRatio .25
        speedRatio = 0.25;
        ungetch(movemode);
    }else if(key == '2'){//speedRatio .50
        speedRatio = 0.50;
        ungetch(movemode);
    }else if(key == '3'){//speedRatio .75
        speedRatio = 0.75;
        ungetch(movemode);
    }else if(key == '4'){//speedRatio 1
        speedRatio = 1;
        ungetch(movemode);
    }else{
        printf("\nInvalid Input!\n");
    }
}

int printSensor(ChNXT *nxt) {
    int touchValue = 0;
    int ultraValue = 0;
    int soundValue = 0;
    int lightValue = 0;

    /* get values from NXT */
    nxt->getSensor(NXT_SENSORPORT1, touchValue);
    nxt->getSensor(NXT_SENSORPORT2, soundValue);
    nxt->getSensor(NXT_SENSORPORT3, lightValue);
    nxt->getSensor(NXT_SENSORPORT4, ultraValue);

    /* display the values */
    if (touchValue == 1)
        printf("\n\n\nThe touch sensor has been activated.\n",
                touchValue);
    else
        printf("\nThe touch sensor has not been activated.\n");

    printf("The distance reported by the ultrasonic sensor is %d.\n",
                        ultraValue);
    /*
    if (light<50)  printf("\nThe touch sensor has been activated\n");
    else    printf("\nThe touch sensor has been activated\n");
    */
    printf("The light level is %d.\n", lightValue);
    printf("The Sound level is %dDb\n\n\n", soundValue);
```

```
    /* GUI display */
    printf("Vehicle Direction:  Other Commands:");
    printf("\n     [w]           [x] Stop all Motors");
    printf("\n      ^            [r] Exit Program");
    printf("\n      |            [1] Set Speed to 25");
    printf("\n     stop          [2] Set Speed to 50");
    printf("\n     [s]           [3] Set Speed to 75");
    printf("\n[a]<-head->[d]     [q] Get Sensor Data\n");
    printf("Please Enter command:");
    return 0;
}
```

Program 16: `humanoid_rtc.ch` Source Code

How does it work?

The the main difference between the RTC program for the NXT humanoid and the NXT vehicle is the content in the switch cases. Other than the switch cases, the **while** loop of the RTC program is the same as the previous RTC program. When the key 'r' has been pressed, the **if-else** statement will run the codes for the case 'r'. In the case 'r', the variable quit will be to 1, which will allow the program to exit the **while** loop, thus quitting the program.
In case 'w', the program will run joints `NXT_JOINTB` and `NXT_JOINTC` at velocity `speedRatio`. For case 'w', the program will control the NXT humanoid to walk forward at a specified speed ratio.

When the key 'w' has been pressed, the **if-else** statement will run the codes for the case 'w'. The program fragment for case 'w' is shown below:

```
if(key == 'w')//up
    nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
    nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
    movemode='w';
    break;
```

In case 's', the program will stop joints `NXT_JOINTB` and `NXT_JOINTC` and turn them to off. For case 's', the program will stop the NXT humanoid from walking forward.

When the key 's' has been pressed, the **if-else** statement will run the codes for the case 's'. The program fragment for case 's' is shown below:

```
else if(key == 's'){//down
    nxt.stopOneJoint(NXT_JOINTB);
    nxt.stopOneJoint(NXT_JOINTC);
    movemode='s';
}
```

In case 'a', the program will run joints `NXT_JOINTA` at a speed ratio of 0.3. For case 'a', the program will rotate the NXT humanoid head left.

When the key 'a' has been pressed, the **if-else** statement will run the codes for the case 'a'. The program fragment for case 'a' is shown below:

```
else if(key == 'a'){//left
    nxt.setJointSpeedRatio(NXT_JOINTA, 0.3);
    nxt.moveJointContinuousNB(NXT_JOINTA, NXT_BACKWARD);
}
```

In case 'd', the program will run the joint NXT_JOINTA at a speed ratio of 0.3. For case 'd', the program will rotate the NXT humanoid head right.

When the key 'd' has been pressed, the **if-else** statement will run the codes for the case 'd'. The program fragment for case 'd' is shown below:

```
else if(key == 'd'){//right
    nxt.setJointSpeedRatio(NXT_JOINTA, 0.3);
    nxt.moveJointContinuousNB(NXT_JOINTA, NXT_FORWARD);
}
```

In case 'q', the program will run the function `printSensor()`, which will get sensor data from each data and print these data out for the user. The details of the `printSensor()` function will be discussed later in this section.

When the key 'q' has been pressed, the **if-else** statement will run the codes for the case 'q'. The program fragment for case 'q' is shown below:

```
else if(key == 'q'){//print sensor
    printSensor();
}
```

In case 'x', the program will stop all the motors connected to the NXT and set them to off idle mode. When the key 'x' has been pressed, the **if-else** statement will run the codes for the case 'x'. The program fragment for case 'x' is shown below:

```
else if(key == 'x'){//stop
    nxt.stopOneJoint(NXT_JOINTB);
    nxt.stopOneJoint(NXT_JOINTC);
    movemode = 'x';
}
```

Printing sensor data

The function definition for the function `printSensor` is shown in the program fragment below:

```
int printSensor(ChNXT *nxt){
    int touchValue = 0;
    int ultraValue = 0;
    int soundValue = 0;
    int lightValue = 0;

    nxt->getSensor(NXT_SENSORPORT1, touchValue);
    nxt->getSensor(NXT_SENSORPORT2, ultraValue);
    nxt->getSensor(NXT_SENSORPORT3, soundValue);
    nxt->getSensor(NXT_SENSORPORT4, lightValue);

    if (touch == 1)
        printf("\n\n\nThe touch sensor has been activated.\n", touchValue);
    else
        printf("\nThe touch sensor has not been activated.\n");

    printf("The distance reported by the ultrasonic sensor is %d.\n",
                    ultraValue);
    /*
    if (light<500)
        printf("\nThe touch sensor has been activated\n");
    else
        printf("\nThe touch sensor has been activated\n");
```

```
    */
    printf("The light level is %d.\n",lightValue);
    printf("The Sound level is %dDb\n\n\n",soundValue);

    //GUI display
    printf("Vehicle Direction:  Other Commands:");
    printf("\n     [w]            [x] Stop all Motors");
    printf("\n      ^             [r] Exit Program");
    printf("\n      |             [1] Set Speed Ratio to 0.25");
    printf("\n     stop           [2] Set Speed Ratio to 0.50");
    printf("\n     [s]            [3] Set Speed Ratio to 0.75");
    printf("\n[a]<-head->[d]      [q] Get Sensor Data\n");
    printf("Please Enter command:");
    return 0;
}
```

In the beginning of this function, the program get sensor data from each sensor and store them in its corresponding variable. Next, **if else** statements are used to print the correct statement if the sensor is triggered or not. For example, if the touch sensor is triggered, the sensor data retrieved is equal to 1, then the program will print ‘The touch sensor has been activated’. Next, the program will print out the distance collected by the ultrasonic sensor, then the light sensor and lastly the sound sensor. Finally, the function prints the user interface again for the user to read.

A ChNXT Class API

The header file nxt.h defines all the data types, macros and function prototypes for th Lego Mindstorms NXT API library. The header file declares a class called ChNXT which contains member functions which may be used to control the Lego Minstorms NXT.

A.1 Data Types Used in ChNXT Class

The data types defined in the header file `nxt.h` are described in this appendix. These data types are used by the NXT library to represent certain values, such as joint id's and motor directions.

Data Type	Description
`nxtJointId_t`	An enumerated value that indicates a nxt joints.
`nxtJointState_t`	The current state of a nxt joint.
`nxtSensorId_t`	An enumerate value that indicates a nxt's sensors.
`nxtSensorType_t`	An enumerate value that indicates the type of a sensor of a nxt.
`nxtSensorMode_t`	An enumerate value that indicates the mode of a sensor for getting value.

A.1.1 `nxtJointId_t`

This data type is an enumerated type used to identify a joint on the Lego Mindstorms NXT. Valid values for this type are:

```
typedef enum nxt_joints_e {
  NXT_JOINTA = 0,
  NXT_JOINTB = 1,
  NXT_JOINTC = 2,
} nxtJointId_t;
```

Value	Description
`NXT_JOINTA`	PortA on the Lego Mindstorms NXT.
`NXT_JOINTB`	PortB on the Lego Mindstorms NXT.
`NXT_JOINTC`	PortC on the Lego Mindstorms NXT.

A.1.2 `nxtJointState_t`

This datatype is an enumerated type used to designate the current movement state of a joint. The values may be retrieved from the robot with the `getJointState()` function and may be set with the `moveContinuous()` family of functions. Valid values are:

```
typedef enum nxt_joint_state_e {
    NXT_FORWARD   = 1,
    NXT_BACKWARD  = -1,
} nxtJointState_t;
```

Value	Description
`NXT_FORWARD`	This value indicates that the joint is currently moving forward.
`NXT_BACKWARD`	This value indicates that the joint is currently moving backward.

A.1.3 nxtSensorPort_t

This datatype is an enumerated type used to identify a sensor on the Lego Mindstorms NXT. Valid values for this type are:

```c
typedef enum nxt_sensors_e{
    NXT_SENSORPORT1 = 0,
    NXT_SENSORPORT2 = 1,
    NXT_SENSORPORT3 = 2,
    NXT_SENSORPORT4 = 3
} nxtSensorId_t;
```

Value	Description
NXT_SENSORPORT1	Select sensor input PORT 1 on the NXT.
NXT_SENSORPORT2	Select sensor input PORT 2 on the NXT.
NXT_SENSORPORT3	Select sensor input PORT 3 on the NXT.
NXT_SENSORPORT4	Select sensor input PORT 4 on the NXT.

A.1.4 nxtSensorType_t

This data type is used to identify the type of a sensor for the Lego Mindstorms NXT. Valid values for this type are:

```c
typedef enum nxt_sensor_type_e{
    NXT_SENSORTYPE_SWITCH           = 0x01,
    NXT_SENSORTYPE_TEMPERATURE      = 0x02,
    NXT_SENSORTYPE_LIGHT_ACTIVE     = 0x05,
    NXT_SENSORTYPE_LIGHT_INACTIVE   = 0x06,
    NXT_SENSORTYPE_SOUND_DB         = 0x07,
    NXT_SENSORTYPE_SOUND_DBA        = 0x08,
    NXT_SENSORTYPE_LOWSPEED         = 0x0A,
    NXT_SENSORTYPE_9V               = 0x0B,
    NXT_SENSORTYPE_HIGHSPEED        = 0x0C, /* No useage now */
    NXT_SENSORTYPE_COLORFULL        = 0x0D,
    NXT_SENSORTYPE_COLORRED         = 0x0E,
    NXT_SENSORTYPE_COLORGREEN       = 0x0F,
    NXT_SENSORTYPE_COLORBLUE        = 0x10,
    NXT_SENSORTYPE_COLORNONE        = 0x11
} nxtSensorType_t;

#define NXT_SENSORTYPE_TOUCH      NXT_SENSORTYPE_SWITCH
#define NXT_SENSORTYPE_ULTRASONIC NXT_SENSORTYPE_LOWSPEED_9V
```

Value	Description
NXT_SENSORTYPE_SWITCH	Set to a switch type sensor. Touch sensor is a switch type sensor.
NXT_SENSORTYPE_TEMPERATURE	Set to Temperature Sensor.
NXT_SENSORTYPE_LIGHT_ACTIVE	Set to Light Sensor in light active mode(LED on).
NXT_SENSORTYPE_LIGHT_INACTIVE	Set to Light Sensor in light inactive mode(LED off).
NXT_SENSORTYPE_SOUND_DB	Set to Sound Sensor in dB.
NXT_SENSORTYPE_SOUND_DBA	Set to Sound Sensor in dB with adjusted.
NXT_SENSORTYPE_LOWSPEED	Set to ISP type sensor.
NXT_SENSORTYPE_LOWSPEED_9V	Set to ISP type sensor with 9 Voltage. The ultrasonic sensor belongs to this type of sensor.

NXT_SENSORTYPE_HIGHSPEED	Not avialable now.
NXT_SENSORTYPE_COLORFULL	Set to Color Sensor in color detector mode.
NXT_SENSORTYPE_COLORRED	Set to Color Sensor in lightsensor mode with red light on.
NXT_SENSORTYPE_COLORGREEN	Set to Color Sensor in lightsensor mode with green light on.
NXT_SENSORTYPE_COLORBLUE	Set to Color Sensor in lightsensor mode with blue light on.
NXT_SENSORTYPE_COLORNONE	Set to Color Sensor in lightsensor mode with no light on.

A.1.5 nxtSensorMode_t

This data type is used to identify the mode for a sensor to get value for the Lego Mindstorms NXT. Valid values for this type are:

```
typedef enum nxt_sensor_mode_e{
    NXT_SENSORMODE_RAWMODE            = 0x00,
    NXT_SENSORMODE_BOOLEANMODE        = 0x20
    NXT_SENSORMODE_TRANSITIONCNTMODE = 0x40,
    NXT_SENSORMODE_PERIODCOUNTERMODE = 0x60,
    NXT_SENSORMODE_PCTFULLSCALEMODE  = 0x80,
    NXT_SENSORMODE_CELSIUSMODE        = 0xA0,
    NXT_SENSORMODE_FAHRENHEITMODE     = 0xC0
} nxtSensorMode_t;
```

Value	Description
NXT_SENSORMODE_RAWMODE	Get sensor value as raw mode.
NXT_SENSORMODE_BOOLEANMODE	Get sensor value as boolean mode.
NXT_SENSORMODE_TRANSITIONCNTMODE	Get sensor value as a number of transitions between TRUE and FALSE.
NXT_SENSORMODE_COUNTERMODE	Get sensor value as a number of transitions from FALSE to TRUE, then back to FALSE.
NXT_SENSORMODE_PCTFULLSCALEDMODE	Get sensor value as percentage of full scale reading for configured sensor type.
NXT_SENSORMODE_CELSIUSMODE	Get sensor value of temperature in degrees Celsius.
NXT_SENSORMODE_FAHRENHEITMODE	Get sensor value of temperature in degrees Fahrenheit.

A.2 ChNXT Class API Overview

Table 12: Functions for Communication

Function	Description
connect()	Connects to the Mindstorms NXT using bluetooth (Read bluetooth address from the configuration file automaticly).
connectWithAddress()	Connects to the Mindstorms NXT via bluetooth (Read bluetooth address from users' input).
disconnect()	Disconnects from the Mindstorms NXT.

Table 13: Functions for Checking Status

Function	Description
isConnected()	Check the connection to a robot.

`isMoving()`	Check if any joints are currently in motion.

Table 14: Functions for Sensors

Function	Description
`setSensor()`	Setup the sensors to collect data from the environment.
`getSensor()`	Get the data collected by the sensors from the NXT.

Table 15: Functions for Joints

Functions	Description
`setJointZero()`	Set the tachometer to zero for a single joint.
`setToZero()`	Set the tachometer to zero for all joints.
`setJointSpeedRatio()`	Set up a single joint speed.
`setJointSpeedRatios()`	Set up all joints' speed.
`moveJointContinuousNB()`	Make a single joint move continuously.
`moveJoint()`	Make a single joint move a user-specified angle.
`moveJointNB()`	Identical to `moveJoint()` but non-blocking.
`moveJointTo()`	Make a single joint move to an absolute angle.
`moveJointToNB()`	Identical to `moveJointTo()` but non-blocking.
`move()`	Make all joints move a user-specified angles.
`moveNB()`	Identical to `move()` but non-blocking.
`moveTo()`	Make all joints move to absolute angles.
`moveToNB()`	Identical to `moveTo()` but non-blocking.
`moveToZero()`	Make all joints move to absolute zero positions.
`moveToZeroNB()`	Identical to `moveToZero()` but non-blocking.
`moveContinuousNB()`	Make all joints move continuously.
`moveContinuousTime()`	Make all joints move continuously for a certain amount of time.
`moveWait()`	Wait untill all motors have stopped moving.
`getJointAngle()`	Get tachometer counts from NXT.
`getJointSpeedRatio()`	Get a motor's speed ratio from NXT.
`getJointSpeedRatios()`	Get all motors' speed ratios from NXT.
`stopOneJoint()`	Make a single joint stop moving.
`stopTwoJoints()`	Make two joints stop moving.
`stopAllJoints()`	Make all joints stop moving.

Table 16: Functions for the Vehicle Configuration

Functions	Description
`vehicleRollForward()`	Moves the NXT vehicle forward.
`vehicleRollForwardNB()`	Identical to `vehicleRollForward()`, but non-blocking.
`vehicleRollBackward()`	Moves the NXT vehicle backward.
`vehicleRollBackwardNB()`	Identical to `vehicleRollBackward()`, but non-blocking.
`vehicleRotateLeft()`	Rotates the NXT vehicle left.
`vehicleRotateLeftNB()`	Identical to `vehicleRotateLeft()`, but non-blocking.

`vehicleRotateRight()`	Rotates the NXT vehicle right.
`vehicleRotateRightNB()`	Identical to `vehicleRotateRight()`, but non-blocking.
`vehicleMotionWait()`	Wait until vehicle stop moving.

Table 17: Functions for the Humanoid Configuration

Functions	Description
`humanoidWalkForward()`	Moves the NXT humanoid forward.
`humanoidWalkForwardNB()`	Identical to `humanoidWalkForward()`, but non-blocking.
`humanoidWalkBackward()`	Moves the NXT humanoid backward.
`humanoidWalkForwardNB()`	Identical to `humanoidWalkBackward()`, but non-blocking.
`humanoidMotionWait()`	Wait until humanoid robot stop moving.

A.3 ChNXT Class API Details

ChNXT::connect()

Synopsis

```
#include <nxt.h>
int ChNXT::connect();
```

Purpose
Connect to a Mindstorms NXT via Bluetooth.

Return Value
The function returns 0 on success, and non-zero otherwise.

Parameters
None.

Description
This function is used to connect to a Mindstorms NXT via Bluetooth. The Bluetooth address is gotten from the configuration file.

Example

```
#include <nxt.h>

ChNXT nxt;
if (nxt.connect()){
    printf("Connection to NXT has failed.");
    exit(0);
}
```

See Also
`connectWithAddress()`, `disconect()`

ChNXT::connectWithAddress()

Synopsis

```
#include <nxt.h>
int ChNXT::connectWithAddress(char usr_addr[18]);
```

Purpose
Connect to a Mindstorms NXT via Bluetooth.

Return Value
The function returns 0 on success, and non-zero otherwise.

Parameters

`usr_address` The Bluetooth address of the Mindstorm NXT.

Description
This function is used to connect to a Mindstorms NXT via Bluetooth. The Bluetooth address is gotten from the configuration file.

Example

```
#include <nxt.h>

ChNXT nxt;
if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(0);
}
```

See Also

connect(), disconnect()

ChNXT::disconnect()

Synopsis

```
#include <nxt.h>
int ChNXT::disconnect();
```

Purpose

Disconnect from a Lego mindstorms NXT.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function is used to disconnect from a connected Mindstorms NXT. A call to this function is not necessary before the terminationof a program. It is only neccessary if another connection will be established within the same program at a later time.

Example

```
#include <nxt.h>

ChNXT nxt;
if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt.disconnect();
```

See Also

connect(), connectWithAddress

ChNXT::getJointAngle()

Synopsis

```
#include <nxt.h>
int ChNXT::getJointAngle(nxtJointId_t id, double &angle);
```

Purpose

Retrieve a robot joint's current angle.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number. This is an enumerated type discussed in Section A.1.1 on page 68.
`angle`	A variable to store the current angle of the robot motor. The contents of this variable will be overwritten with a value that represents the motor's angle in degrees.

Description
This function gets the current motor angle of a NXT's motor. The angle returned is in units of degrees and is accurate to roughly ± 1 degrees. The function `getJointAngle()` always returns an angle from 0 to ∞ degrees.

Example

```
#include <nxt.h>

ChNXT nxt;
double angle;

nxt.connect();

nxt.getJointAngle(NXT_JOINTA, angle);

printf("The angle of jointA is: %lf\n", angle);
```

See Also

ChNXT::getJointSpeedRatio()

Synopsis

```
#include <nxt.h>
int ChNXT::getJointSpeedRatio(nxtJointId_t id, double &ratio);
```

Purpose
Get the speed ratio settings of a joint on the Lego Mindstorms NXT.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	Retrieve the speed ratio setting of this joint. This is an enumerated type discussed in Section A.1.1 on page 68.
`ratio`	A variable of type double. The value of this variable will be overwritten with the current speed ratio setting of the joint.

Description
This function is used to find the speed ratio setting of a joint. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1. In other words, if the ratio is set to 0.5, the joint will turn at 50% of its maximum angular velocity while moving continuously or moving to a new goal position.

Example

```
#include <nxt.h>

ChNXT nxt;
double ratio;

nxt.connect();

nxt.getJointSpeedRatio(NXT_JOINTA, ratio);

printf("The speed ratio of jointA is: %lf\n", ratio);
```

See Also
`setJointSpeedRatio(), getJointSpeedRatio()`

ChNXT::getJointSpeedRatios()

Synopsis

```
#include <nxt.h>
int ChNXT::getJointSpeedRatios(double &ratio1, double &ratio2, double &ratio3);
```

Purpose
Get the speed ratio settings of all joints on the Lego Mindstorms NXT.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
- `ratio1` The address of a variable to store the speed ratio of joint A.
- `ratio2` The address of a variable to store the speed ratio of joint B.
- `ratio3` The address of a variable to store the speed ratio of joint C.

Description
This function is used to retrieve all four joint speed ratio settings of a Lego Mindstorms NXT simultaneously. The speed ratios are as a value from 0 to 1.

Example

```
#include <nxt.h>

ChNXT nxt;
double ratio1, ratio2, ratio3;

nxt.connect();

nxt.getJointSpeedRatio(ratio1, ratio2, ratio3);

printf("The speed ratio of jointA is: %lf\n", ratio1);
printf("The speed ratio of jointB is: %lf\n", ratio2);
printf("The speed ratio of jointC is: %lf\n", ratio3);
```

See Also
`getJointSpeedRatio(), setJointSpeedRatio()`

ChNXT::getJointState()

Synopsis

```
#include <nxt.h>
int ChNXT::getJointState(nxtJointId_t id, nxtJointState_t &state);
```

Purpose
Determine whether a motor is moving or not.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number. This is an enumerated type discussed in Section A.1.1 on Page 68.
`state`	An integer variable which will be overwritten with the current state of the motor. This is an enumerated type discussed in Section A.1.2 on Page 68.

Description
This function is used to determine the current state of a motor. Valid states are listed below.

Value	Description
NXT_FORWARD	This value indicates that the joint is currently moving forward.
NXT_BACKWARD	This value indicates that the joint is currently moving backward.

Example

```
#include <nxt.h>

ChNXT nxt;
nxtJointState_t status;

nxt.connect();

nxt.getJointState(NXT_JOINTA, status);
if(status == 0)
    printf("JointA is not moving.\n");
else
    printf("JointA is moving.\n");
```

See Also
`isMoving()`

ChNXT::getSensor()

Synopsis

```
#include <nxt.h>
int ChNXT::getSensor(nxtSensorId_t id, double &value);
```

Purpose
Retrieve a NXT sensor's current value.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The sensor port. This is an enumerated type discussed in Section A.1.3 on Page 69.
`value`	A variable to store the current value of the NXT sensor. The contents of this variable will be overwritten with the sensor's value.

Description
This function gets the current sensor's value of a NXT.

Example

```
#include <nxt.h>

ChNXT nxt;
int value;

nxt.connect();

nxt.setSensor(NXT_SENSORPORT4, NXT_SENSORTYPE_ULTRASONIC,
                NXT_SENSORMODE_RAWMODE);

nxt.getSensor(NXT_SENSORPORT4, value);

printf("The value of sensor 4 is: %d\n", value);
```

See Also

ChNXT::humanoidWalkBackward()
ChNXT::humanoidWalkBackwardNB()

Synopsis

```
#include <mobot.h>
int ChNXT::humanoidWalkBackward(double angle);
int ChNXT::humanoidWalkBackwardNB(double angle);
```

Purpose
Make the Lego Mindstorms NXT in the humanoid configuration walk backward.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`angle`	The angle to turn the joints, specified in degrees.

Description

ChNXT::humanoidWalkBackward()
This function is used to make the Lego Mindstorms walk backward in the humanoid configuration. The amount to roll the joints is specified by the argument, `angle`.

ChNXT::humanoidWalkBackwardNB()
This function is used to make the Lego Mindstorms walk backward in the humanoid configuration. The amount to roll the joints is specified by the argument, `angle`.

This function has both a blocking and non-blocking version. The blocking version, `humanoidWalkBackward()`, will block until the robot motion has completed. The non-blocking version, `humanoidWalkBackwardNB()`, will return immediately, and the motion will be performed asynchronously.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Blocking function */
nxt.humanoidWalkBackward(360);

/* Non-blocking function */
nxt.humanoidWalkBackwardNB(360);
nxt.humanoidMotionWait();
```

See Also

`humanoidWalkBackward()`

ChNXT::humanoidWalkForward()
ChNXT::humanoidWalkForwardNB()

Synopsis

```
#include <mobot.h>
int ChNXT::humanoidWalkForward(double angle);
int ChNXT::humanoidWalkForwardNB(double angle);
```

Purpose

Make the Lego Mindstorms NXT in the humanoid configuration walk forward.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`angle`	The angle to turn the joints, specified in degrees.

Description

ChNXT::humanoidWalkForward()
This function is used to make the Lego Mindstorms walk forward in the humanoid configuration. The amount to roll the joints is specified by the argument, `angle`.

ChNXT::humanoidWalkForwardNB()
This function is used to make the Lego Mindstorms walk forward in the humanoid configuration. The amount to roll the joints is specified by the argument, `angle`.

This function has both a blocking and non-blocking version. The blocking version, `humanoidWalkForward()`, will block until the robot motion has completed. The non-blocking version, `humanoidWalkForwardNB()`, will return immediately, and the motion will be performed asynchronously.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Blocking function */
nxt.humanoidWalkForward(360);

/* Non-blocking function */
nxt.humanoidWalkForwardNB(360);
nxt.humanoidMotionWait();
```

See Also
`humanoidWalkBackward()`

ChNXT::humanoidMotionWait()

Synopsis

```
#include <nxt.h>
int ChNXT::humanoidMotionWait();
```

Purpose
Wait for a motion to complete execution in humanoid configuration.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function is used to wait for a motion function to fully complete its cycle.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Non-blocking function */
nxt.humanoidRotateRightNB(360);

/* wait until non-blocking motion stops */
nxt.humanoidMotionWait();
```

See Also
`humanoidWalkForward()`, `humanoidWalkBackward()`

ChNXT::isConnected()

Synopsis

```
#include <nxt.h>
int ChNXT::isConnected(void);
```

Purpose
Check to see if currently connected to a Lego Mindstorms NXT via Bluetooth.
Return Value
The function returns zero if it is not currently connected to a Lego Mindstorms NXT or if an error has occured, or 1 if the Lego Mindstorms NXT is connected.

Parameters
None.

Description
This function is used to check if the software is currently connected to a Lego Mindstorms NXT.

Example

```
#include <nxt.h>

ChNXT nxt;
int connectStatus;

nxt.connect();
connectStatus = nxt.isConnected();

if(connectStatus)
    printf("Connected!\n");
else
    printf("Not connected!\n");
```

See Also

ChNXT::isMoving()

Synopsis

```
#include <nxt.h>
int ChNXT::isMoving(void);
```

Purpose
Check to see if a Lego Mindstorms NXT is currently moving any of its joints.

Return Value
This function returns 0 if none of the joints are being driven or if an error has occured, or 1 if any joint is being driven.

Parameters
None.

Description
This function is used to determine if a robot is currently moving any of its joints.

Example

```
#include <nxt.h>

ChNXT nxt;
int moveStatus;

nxt.connect();
```

```
moveStatus = isMoving(NXT_JOINTA);

if(moveStatus)
    printf("JointA is moving!\n");
else
    printf("JointA is not moving!\n");
```

See Also
`getJointState()`

ChNXT::move()
ChNXT::moveNB()

Synopsis

```
#include <nxt.h>
int ChNXT::move(double degreesA, double degreesB, double degreesC);
int ChNXT::moveNB(double degreesA, double degreesB, double degreesC);
```

Purpose
Move all joints of Lego Mindstorms NXT by specified degrees.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`angle1`	The amount to move joint A relative to the current position.
`angle2`	The amount to move joint B relative to the current position.
`angle3`	The amount to move joint C relative to the current position.

Description

ChNXT::move()
This function moves all of the joints of a Lego Mindstorms NXT by the specified number of degrees from their current positions.

ChNXT::moveNB()
This function moves all of the joints of a Lego Mindstorms NXT by the specified number of degrees from their current positions.

The function `moveNB()` is the non-blocking version of the `move()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more information on blocking and non-blocking functions, please refer to Section 4.5.2 on page 17.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(0);
}
```

```
/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move by the non-blocking function*/
nxt.moveNB(360, 360, 360);
nxt.moveWait();

/* move by the blocking function*/
nxt.move(360, 360, 360);
```

See Also
`moveTo()`, `moveToNB()`

ChNXT::moveContinuousNB()

Synopsis

```
#include <nxt.h>
int ChNXT::moveContinuousNB(nxtJointState_t dirA,
                            nxtJointState_t dirB,
                            nxtJointState_t dirC);
```

Purpose
Move the joints of a robot continuously in the specified directions.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
Each parameter specifies the direction the joint should move. The types are enumerated in `mobot.h` and have the following values:

Value	Description
NXT_FORWARD	This value indicates that the joint is currently moving forward.
NXT_BACKWARD	This value indicates that the joint is currently moving backward.

More documentation about these types may be found at Section A.1.2 on page 68.

Description
This function causes joints of a robot to begin moving at the previously set speed. The joints will continue moving until the joint hits a joint limit, or the joint is stopped by setting the speed to zero. This function is a non-blocking function.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(0);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move all joints on the Mindstorm NXT */
nxt.moveContinuousNB(NXT_FORWARD, NXT_FORWARD, NXT_FORWARD);
```

See Also

ChNXT::moveContinuousTime()

Synopsis

```
#include <nxt.h>
int ChNXT::moveContinuousTime(nxtJointState_t dirA,
                              nxtJointState_t dirB,
                              nxtJointState_t dirC,
                              double seconds);
```

Purpose
Move the joints of a robot continuously in the specified directions for some amount of time.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
Each direction parameter specifies the direction the joint should move. The types are enumerated in `nxt.h` and have the following values:

Value	Description
NXT_FORWARD	This value indicates that the joint is currently moving forward.
NXT_BACKWARD	This value indicates that the joint is currently moving backward.

The `seconds` parameter is the time to perform the movement, in seconds.

Description
This function causes joints of a robot to begin moving. The joints will continue moving until the joint hits a joint limit, or the time specified in the `seconds` parameter is reached. This function will block until the motion is completed.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(0);
}

/* setup speed for all three joints */
nxt.setJointSpeedRatios(0.4, 0.4, 0.4);

/* move all joints on the Mindstorm NXT in 10 seconds*/
nxt.moveContinuousTime(NXT_FORWARD, NXT_FORWARD,
                       NXT_FORWARD, 10);
```

See Also

ChNXT::moveJoint()

ChNXT::moveJointNB()

Synopsis

```
#include <nxt.h>
int ChNXT::moveJoint(nxtJointId_t id, double angle);
int ChNXT::moveJointNB(nxtJointId_t id, double angle);
```

Purpose

Move a joint on the robot by a specified angle with respect to the current position.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to move.
`angle`	The desired angle the joint need to move.

Description

ChNXT::moveJoint()
This function commands the motor to move by an angle relative to the joint's current position at the joints current speed setting. The current motor speed ratio may be set with the `setJointSpeedRatio()` member function. Please note that if the motor speed is set to zero, the motor will not move after calling the `moveJoint()` function.

ChNXT::moveJointNB()
This function commands the motor to move by an angle relative to the joint's current position at the joints current speed setting. The current motor speed ratio may be set with the `setJointSpeedRatio()` member function. Please note that if the motor speed is set to zero, the motor will not move after calling the `moveJointNB()` function.

The function `moveJointNB()` is the non-blocking version of the `moveJoint()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 4.5.2 on page 17.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(0);
}

/* setup speed for joint1 */
nxt.setJointSpeedRatio(NXT_JOINTA, 40);

/* move joint1 360 degrees */
nxt.moveJointNB(NXT_JOINTA, 360);
nxt.moveJointWait();

/* move joint1 360 degrees */
nxt.moveJoint(NXT_JOINTA, 360);
```

See Also

ChNXT::moveJointContinuousNB()

Synopsis

```
#include <nxt.h>
int ChNXT::moveJointContinuousNB(robotJointId_t id, int dir);
```

Purpose
Move the specific joint on the Mindstorm NXT continuously in a direction.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint of the Mindstorm NXT.
`dir`	The move direction of the joint. 1 means positive and 0 means negative direction.

Description
This function is used to rotate the specific joint of the Mindstorm NXT. The joint will move continuously until the function `stopOneJoint()` is called. The variable `dir` indicates the move direction of the joint.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move the joint 1 on the Mindstorm NXT */
nxt.moveJointContinuousNB(NXT_JOINTA, NXT_FORWARD);
```

See Also

ChNXT::moveJointTo()
ChNXT::moveJointToNB()

Synopsis

```
#include <nxt.h>
int ChNXT::moveJointTo(nxtJointId_t id, double angle);
int ChNXT::moveJointToNB(nxtJointId_t id, double angle);
```

Purpose
Move a joint on the Lego Mindstorms NXT to an absolute position.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to move.
`angle`	The absolute angle in degrees to move the motor to.

Description

ChNXT::moveJointTo()
This function commands the motor to move to a position specified in degrees at the current motor's speed. The current motor speed may be set with the `setJointSpeedRatio()` member function. Please note that if the motor speed is set to zero, the motor will not move after calling the `moveJointTo()` function.

ChNXT::moveJointToNB()
This function commands the motor to move to a position specified in degrees at the current motor's speed. The current motor speed may be set with the `setJointSpeedRatio()` member function. Please note that if the motor speed is set to zero, the motor will not move after calling the `moveJointToNB()` function.

The function `moveJointToNB()` is the non-blocking version of the `moveJointTo()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 4.5.2 on page 17.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeedRatio(NXT_JOINTA, 40);

/* move joint1 360 degrees with the non-blocking function */
nxt.moveJointToNB(NXT_JOINTA, 360);
nxt.moveJointWait(NXT_JOINTA);

/* move joint1 360 degrees with the blocking function */
nxt.moveJointTo(NXT_JOINTA, 360);
```

See Also

ChNXT::moveJointWait()

Synopsis

```
#include <nxt.h>
int ChNXT::moveJointWait(nxtJointId_t id);
```

Purpose
Wait for a joint to stop moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to wait for.

Description
This function is used to wait for a joint motion to finish. Functions such as `moveNB()` and `moveJointNB()` do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveJointWait()` function is used to wait for a robotic joint motion to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example
See Also
`moveWait()`

ChNXT::moveTo()
ChNXT::moveToNB()

Synopsis

```
#include <nxt.h>
int ChNXT::moveTo(double angleA, double angleB, double angleC);
int ChNXT::moveToNB(double angleA, double angleB, double angleC);
```

Purpose
Move all joints of Lego Mindstorms NXT to the specified position.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`angle1`	The absolute position to move joint A, expressed in degrees.
`angle2`	The absolute position to move joint B, expressed in degrees.
`angle3`	The absolute position to move joint C, expressed in degrees.

Description

ChNXT::moveTo()
This function moves all of the joints of a robot to the specified absolute positions.

ChNXT::moveToNB()
This function moves all of the joints of a robot to the specified absolute positions.

The function `moveToNB()` is the non-blocking version of the `moveTo()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 4.5.2 on page 17.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(40, 40, 40);

/* move by the non-blocking function*/
nxt.moveToNB(360, 360, 360);
nxt.moveWait();

/* move by the blocking function*/
nxt.moveTo(360, 360, 360);
```

See Also

`move()`, `moveNB()`

`ChNXT::moveToZero()`
`ChNXT::moveToZeroNB()`

Synopsis

```
#include <nxt.h>
int ChNXT::moveToZero();
int ChNXT::moveToZeroNB();
```

Purpose

Move all joints of Lego Mindstorms NXT to their absolute zero position.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

ChNXT::moveToZero()
This function moves all of the joints of a NXT to their zero position.

ChNXT::moveToZeroNB()
This function moves all of the joints of a NXT to their zero position.

The function `moveToZeroNB()` is the non-blocking version of the `moveToZero()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 4.5.2 on page 17.

Example

```
#include <nxt.h>
```

```
ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move to zero by the non-blocking function*/
nxt.moveToZeroNB();
nxt.moveWait();

/* move to zero by the blocking function*/
nxt.moveToZero();
```

See Also

ChNXT::moveWait()

Synopsis

```
#include <nxt.h>
int ChNXT::moveWait();
```

Purpose
Wait for all joints to stop moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function is used to wait for all joint motions to finish. Functions such as `move()` and `moveTo()` do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveWait()` function is used to wait for robotic motions to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example

```
#include <nxt.h>

ChNXT nxt;

if (!nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move by the non-blocking function*/
nxt.moveNB(360, 360, 360);

/* wait until motors stop moving */
```

```
nxt.moveWait();
```

See Also
`moveJointWait()`

ChNXT::setJointToZero()

Synopsis

```
#include <nxt.h>
int ChNXT::setJointToZero(nxtJointId_t id);
```

Purpose
Reset the tachometer count for a single joint on Mindstorms NXT.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The port of the joint locate on the Mindstorm NXT.

Description
This function is used to reset the tachometer count for a joint on the Mindstorms NXT.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

nxt.setJointToZero(NXT_JOINTA);
```

See Also
`setJointToZeros()`

ChNXT::setToZero()

Synopsis

```
#include <nxt.h>
int ChNXT::setToZero(void);
```

Purpose
Reset the tachometer count for all motors.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function is used to reset the tachometer count for all joints of the Mindstorms NXT.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

nxt.setToZero();
```

See Also
setJointToZero()

ChNXT::setSensor()

Synopsis

```
#include <nxt.h>
int ChNXT::setSensor(nxtSensorId_t id,
                     nxtSensorType type,
                     nxtSensorMode mode);
```

Purpose
Set up a sensor of Lego mindstorms NXT.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

id	The port of the sensor locate on the Mindstorm NXT.
type	The type of the sensor.
mode	The mode of the sensor.

NXT Sensor Types

Value	Description
NXT_SENSORTYPE_SWITCH	Set to a switch type sensor. Touch sensor is a switch type sensor.
NXT_SENSORTYPE_LIGHT_ACTIVE	Set to Light Sensor in light active mode(LED on).
NXT_SENSORTYPE_LIGHT_INACTIVE	Set to Light Sensor in light inactive mode(LED off).
NXT_SENSORTYPE_SOUND_DB	Set to Sound Sensor in dB.
NXT_SENSORTYPE_SOUND_DBA	Set to Sound Sensor in dB with adjusted.
NXT_SENSORTYPE_LOWSPEED	Set to ISP type sensor.
NXT_SENSORTYPE_LOWSPEED_9V	Set to ISP type sensor with 9 Voltage. The ultrasonic sensor belongs to this type of sensor.
NXT_SENSORTYPE_COLORFULL	Set to Color Sensor in color detector mode.
NXT_SENSORTYPE_COLORRED	Set to Color Sensor in lightsensor mode with red light on.

NXT_SENSORTYPE_COLORGREEN	Set to Color Sensor in lightsensor mode with green light on.
NXT_SENSORTYPE_COLORBLUE	Set to Color Sensor in lightsensor mode with blue light on.
NXT_SENSORTYPE_COLORNONE	Set to Color Sensor in lightsensor mode with no light on.

NXT Sensor Modes

Value	Description
NXT_SENSORMODE_RAWMODE	Get sensor value as raw mode.
NXT_SENSORMODE_BOOLEANMODE	Get sensor value as boolean mode.
NXT_SENSORMODE_TRANSITIONCNTMODE	Get sensor value as a number of transitions between TRUE and FALSE.
NXT_SENSORMODE_COUNTERMODE	Get sensor value as a number of transitions from FALSE to TRUE, then back to FALSE.
NXT_SENSORMODE_PCTFULLSCALEDMODE	Get sensor value as percentage of full scale reading for configured sensor type.

NXT Sensor Range

Sensor	Range
Touch Sensor	0 and 1.
Light Sensor	0 to 100 (percentage).
Sound Sensor	0 to 100 (percentage).
Ultrasonic Sensor	0 to 80 cm.
Color Sensor	0 to 6.

Description

This function is used to set up a sensor on the Mindstorm with a specific type and mode of the sensor. Function `getSensor()` can be used to get the sensor values for each type of sensor.

Example

```
#include <nxt.h>

ChNXT nxt;
int status_1=2;
int status_2=2;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup port 1 as touch sensor */
status_1=nxt.setSensor(SENSOR_PORT1,
        SENSOR_TYPE_TOUCH, SENSOR_MODE_BOOLEANMODE);
if(status_1){
    printf("Connection to touch sensor has failed");
}

/* setup port4 as ultrasonic sensor */
status_2=nxt.setSensor(SENSOR_PORT4,
        SENSOR_TYPE_ULTRASONIC, SENSOR_MODE_RAWMODE);
```

```
if(status_2){
    printf("Connection to ultrasonic sensor has failed");
}
```

See Also
`getSensor()`

ChNXT::setJointSpeedRatio()

Synopsis

```
#include <nxt.h>
int ChNXT::setJointSpeedRatio(nxtJointId_t id, double ratio);
```

Purpose
Set the speed ratio setting of a joint on the Lego Mindstorms NXT.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The port of the sensor locate on the Mindstorm NXT.
`ratio`	A variable of type double with a value from 0 to 1.

Description
This function is used to set the speed ratio setting of a joint on the Lego Mindstorms NXT. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1. In other words, if the ratio is set to 0.5, the joint will turn at 50% of its maximum angular velocity while moving continuously or moving to a new goal position.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* set the speed ratio setting for joint1 on port0 */
nxt.setJointSpeedRatio(NXT_JOINTA, 0.5);
```

See Also
`setJointSpeedRatios()`

ChNXT::setJointSpeedRatios()

Synopsis

```
#include <nxt.h>
int ChNXT::setJointSpeedRatios(double ratioA, double ratioB, double ratioC);
```

Purpose
Set the speed ratio setting of all joints on the Lego Mindstorms NXT.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`ratioA`	The speed ratio setting for the first joint.
`ratioB`	The speed ratio setting for the second joint.
`ratioC`	The speed ratio setting for the third joint.

Description
This function is used to simultaneously set the speed ratio settings of all joints on the Lego Mindstorms NXT. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* set the speed ratio settings for all joints*/
nxt.setJointSpeedRatios(0.5, 0.4, 0.4);
```

See Also
`setJointSpeedRatio()`

ChNXT::stopOneJoint()

Synopsis

```
#include <nxt.h>
int ChNXT::stopOneJoint(nxtJointId_t id);
```

Purpose
Stop a joint moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint of the Mindstorm NXT.

Description
This function is used to stop the specific joint on the Mindstorm NXT.

Example

```
#include <nxt.h>

ChNXT nxt;
```

```
if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move joint1 360 degrees */
nxt.moveJointContinuousNB(NXT_JOINTA, NXT_FORWARD);
delay(5);

/* stop joint1 */
nxt.stopOneJoint(NXT_JOINTA);
```

See Also
`stopTwoJoints()`, `stopAllJoints()`

ChNXT::stopTwoJoints()

Synopsis

```
#include <nxt.h>
int ChNXT::stopTwoJoints(nxtJointId_t id1, nxtJointId_t id2);
```

Purpose
Stop two joints moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id1`	A joint of the Mindstorm NXT.
`id2`	Another joint of the Mindstorm NXT.

Description
This function is used to stop two specific joints on the Mindstorm NXT.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move joint2 and joint3 forward */
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
delay(5);
```

```
/* stop joint2 and joint3 */
nxt.stopTwoJoints(NXT_JOINTB, NXT_JOINTC);
```

See Also
stopOneJoint(), stopAllJoints()

ChNXT::stopAllJoints()

Synopsis

```
#include <nxt.h>
int ChNXT::stopAllJoints(void);
```

Purpose
Stop all joints moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function is used to stop all joints on the Mindstorm NXT.

Example

```
#include <nxt.h>

ChNXT nxt;

if (nxt.connectWithAddress("00:16:53:12:e7:80")){
    printf("Connection to NXT has failed.");
    exit(-1);
}

/* setup speed for all three joints */
nxt.setJointSpeeds(60, 40, 40);

/* move all joints forward */
nxt.moveJointContinuousNB(NXT_JOINTA, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTB, NXT_FORWARD);
nxt.moveJointContinuousNB(NXT_JOINTC, NXT_FORWARD);
delay(5);

/* stop all joints */
nxt.stopAllJoints();
```

See Also
stopOneJoint(), stopTwoJoints()

ChNXT::vehicleRollBackward()
ChNXT::vehicleRollBackwardNB()

Synopsis

```
#include <nxt.h>
```

```
int ChNXT::vehicleRollBackward(double angle);
int ChNXT::vehicleRollBackwardNB(double angle);
```

Purpose
Make the Lego Mindstorms NXT in the vehicle configuration roll backward.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`angle`	The angle to turn the wheels, specified in degrees.

Description

ChNXT::vehicleRollBackward()
This function is used to roll the Lego Mindstorms NXT backward in the vehicle configuration. The amount to roll the wheels is specified by the argument, `angle`.

ChNXT::vehicleRollBackwardNB()
This function is used to roll the Lego Mindstorms NXT backward in the vehicle configuration. The amount to roll the wheels is specified by the argument, `angle`.

This function has both a blocking and non-blocking version. The blocking version, `vehicleRollBackward()`, will block until the robot motion has completed. The non-blocking version, `vehicleRollBackwardNB()`, will return immediately, and the motion will be performed asynchronously.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Blocking function */
nxt.vehicleRollBackward(360);

/* Non-blocking function */
nxt.vehicleRollBackwardNB(360);
nxt.vehicleMotionWait();
```

See Also
`vehicleRollForward()`

ChNXT::vehicleRollForward()
ChNXT::vehicleRollForwardNB()

Synopsis

```
#include <mobot.h>
int ChNXT::vehicleRollForward(double angle);
int ChNXT::vehicleRollForwardNB(double angle);
```

Purpose
Make the Lego Mindstorms NXT in the vehicle configuration roll forward.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`angle` The angle to turn the wheels, specified in degrees.

Description

ChNXT::vehicleRollForward()
This function is used to roll the Lego Mindstorms forward in the vehicle configuration. The amount to roll the wheels is specified by the argument, `angle`.

ChNXT::vehicleRollForwardNB()
This function is used to roll the Lego Mindstorms forward in the vehicle configuration. The amount to roll the wheels is specified by the argument, `angle`.

This function has both a blocking and non-blocking version. The blocking version, `vehicleRollForward()`, will block until the robot motion has completed. The non-blocking version, `vehicleRollForwardNB()`, will return immediately, and the motion will be performed asynchronously.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Blocking function */
nxt.vehicleRollForward(360);

/* Non-blocking function */
nxt.vehicleRollForwardNB(360);
nxt.vehicleMotionWait();
```

See Also
`vehicleRollBackward()`

ChNXT::vehicleRotateLeft()
ChNXT::vehicleRotateLeftNB()

Synopsis

```
#include <nxt.h>
int ChNXT::vehicleRotateLeft(double angle);
int ChNXT::vehicleRotateLeftNB(double angle);
```

Purpose
Rotate the Lego Mindstorms NXT left in vehicle configuration.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`angle` The angle in degrees to turn the wheels. The wheels will turn in opposite directions by the amount specifid by this argument in order to rotate the robot to the left.

Description

ChNXT::vehicleRotateLeft()
This function is used to rotate the wheels of the Lego Mindstorms NXT in vehicle configuration in opposite directions to cause the robot to rotate counter-clockwise.

ChNXT::vehicleRotateLeftNB()
This function is used to rotate the wheels of the Lego Mindstorms NXT in vehicle configuration in opposite directions to cause the robot to rotate counter-clockwise.

This function has both a blocking and non-blocking version. The blocking version, `vehicleRotateLeft()`, will block until the robot motion has completed. The non-blocking version, `vehicleRotateLeftNB()`, will return immediately, and the motion will be performed asynchronously.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Blocking function */
nxt.vehicleRotateLeft(360);

/* Non-blocking function */
nxt.vehicleRotateLeftNB(360);
nxt.vehicleMotionWait();
```

See Also

`vehicleRotateRight()`

ChNXT::vehicleRotateRight()
ChNXT::vehicleRotateRightNB()

Synopsis

```
#include <mobot.h>
int ChNXT::vehicleRotateRight(double angle);
int ChNXT::vehicleRotateRightNB(double angle);
```

Purpose

Rotate the Lego Mindstorms NXT right in vehicle configuration.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`angle` The angle in degrees to turn the wheels. The wheels will turn in opposite directions by the amount specifid by this argument in order to turn the robot to the right.

Description

ChNXT::vehicleRotateRight()
This function causes the robot to rotate the wheels of the Lego Mindstorms NXT in vehicle configuration in opposite directions to cause the robot to rotate clockwise.

ChNXT::vehicleRotateRightNB()
This function causes the robot to rotate the wheels of the Lego Mindstorms NXT in vehicle configuration in opposite directions to cause the robot to rotate clockwise.

This function has both a blocking and non-blocking version. The blocking version, `vehicleRotateRight()`, will block until the robot motion has completed. The non-blocking version, `vehicleRotateRightNB()`, will return immediately, and the motion will be performed asynchronously.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Blocking function */
nxt.vehicleRotateRight(360);

/* Non-blocking function */
nxt.vehicleRotateRightNB(360);
nxt.vehicleMotionWait();
```

See Also
`vehicleRotateRight()`

ChNXT::vehicleMotionWait()

Synopsis

```
#include <nxt.h>
int ChNXT::vehicleMotionWait();
```

Purpose
Wait for a motion to complete execution in vehicle configuration.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function is used to wait for a motion function to fully complete its cycle.

Example

```
#include <nxt.h>

ChNXT nxt;

nxt.connect();

/* Non-blocking function */
nxt.vehicleRotateRightNB(360);
```

```
/* wait until non-blocking motion stops */
nxt.vehicleMotionWait();
```

See Also

`vehicleRotateRight()`, `vehicleRotateLeft()`,
`vehicleRollForward()`, `vehicleRollBackward()`

B Miscellaneous Utility Functions

Besides the control functions described in Appendix A, some utility functions are also very useful when you are programming, such as convert a certain angle to distance with the coresponding radius of wheel. Therefore, we will introduce you some useful utility functions in this section.

B.1 Overview

Table 21: NXT Utility Functions.

Function	Description
`angle2distance()`	Calculates the angle a wheel has turned from the radius and distance traveled.
`deg2rad()`	Converts degrees to radians.
`delay()`	Puts a delay into a program.
`distance2angle()`	Calculates the distance traveled by a wheel from the wheel's radius and angle turned.
`rad2deg()`	Converts radians to degrees.

B.2 Function Details

angle2distance()

Synopsis

```
#include <nxt.h>
double angle2distance(double radius, double angle);
array double angle2distance(double radius, array double angle[:])[:];
```

Purpose

Calculate the distance a wheel has traveled from the radius of the wheel and the angle the wheel has turned.

Return Value

The value returned is the distance traveled by the wheel. If the angle argument is an array of angles, then the value returned is an array of distances. Each element of the distance array returned is the distance calculated from the respective element in the angle array.

Parameters

`radius`	The radius of the wheel.
`angle`	This value is the angle the wheel has turned. This parameter may be of `double` type, or a Ch computational array.

Description

This function calculates the angle a wheel has turned given the wheel radius and distance traveled. The equation used is

$$d = r\theta$$

where d is the distance traveled, r is the radius of the wheel, and θ is the angle the wheel has turned in radians.

Example

See Also
`distance2angle()`

---

deg2rad()

Synopsis
```
#include <nxt.h>
double deg2rad(double degrees);
array double deg2rad(double degrees[:])[:];
```

Purpose
Convert degrees to radians.

Return Value
The angle parameter converted to radians.

Parameters

`degrees`	The angle to convert, in degrees.

Description
This function converts an angle expressed in degrees into radians. Degrees and radians are two popular ways to express an angle, though they are not interchangable. The following equation is used to convert degrees to radians:

$$\theta = \delta * \frac{\pi}{180}$$

where θ is the angle in radians and δ is the angle in degrees.
Example

See Also
`rad2deg()`

---

delay()

Synopsis
```
#include <nxt.h>
void delay(double seconds);
```

Purpose
Pause a program for a set amount of time.

Return Value
None.

Parameters

`seconds`	The number of seconds to delay.

Description
This function delays or pauses a program for a number of seconds. For instance, the code

```
delay(0.5);
printf("Hello.\n");
delay(2);
printf("Goodbye.\n");
```

will pause for half a second, print the text `Hello.`, delay for 2 seconds, and then print the text `Goodbye.`. **Example**

See Also

---

distance2angle()

Synopsis
```
#include <nxt.h>
double distance2angle(double radius, double distance);
array double distance2angle(double radius, array double distance[:])[:];
```

Purpose
Calculate the angle a wheel has turned from the radius of the wheel and the distance the wheel has traveled.

Return Value
The value returned is the angle turned by the wheel in degrees. If the distance argument is an array of distances, then the value returned is an array of angles. Each element of the angle array returned is the angle calculated from the respective element in the distance array.

Parameters

`radius`	The radius of the wheel.
`distance`	This value is the distance the wheel has traveled. This parameter may be of `double` type, or a Ch computational array.

Description
This function calculates the distance a wheel has turned given the wheel radius and angle turned. The equation used is

$$\theta = \frac{d}{r}$$

where d is the distance traveled, r is the radius of the wheel, and θ is the angle the wheel has turned in radians. A further conversion is done in the code to convert the angle from radians into degrees before returning the value.

Example

See Also
`angle2distance()`

rad2deg()

Synopsis

```
#include <nxt.h>
double rad2deg(double radians);
array double rad2deg(double radians[:])[:];
```

Purpose

Convert radians to degrees.

Return Value

The angle parameter converted to degrees.

Parameters

`radians` The angle to convert, in radians.

Description

This function converts an angle expressed in radians into degrees. Degrees and radians are two popular ways to express an angle, though they are not interchangable. The following equation is used to convert radians to degrees:

$$\delta = \theta * \frac{180}{\pi}$$

where θ is the angle in radians and δ is the angle in degrees.

Example

See Also

`deg2rad()`

Index