Ch Mindstorms Package User’s Guide

Version 0.9.5

Most engaging to learn math
Simplest to program a robot
Easiest to control multiple robots

Integration Engineering Laboratory
University of California, Davis
June 6, 2016
1 Introduction 1

2 Configuring Lego Mindstorms NXT/EV3 for Remote Control 2
   2.1 Check Bluetooth Support on Computers . . . . . . . . . . . . . . . . . . . . . . . . . 2
   2.2 Enable Bluetooth on NXTs/EV3s . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   2.3 Launch “Ch Robot Controller” . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   2.4 Configure Robots . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
      2.4.1 Scan Robots . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
      2.4.2 Add and Pair a Robot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
      2.4.3 Rename a Robot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3 Control NXTs/EV3s via the GUI Controller 11
   3.1 Assemble a NXT/EV3 with Motors and Sensors . . . . . . . . . . . . . . . . . . . . 11
   3.2 Connect and Disconnect to NXTs/EV3s from the Ch Mindstorms Controller . . . . 12
   3.3 Robot Control Option . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
   3.4 Single Vehicle Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
      3.4.1 Motion Control Panel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
      3.4.2 Sensors Panel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
      3.4.3 Sensor Control Panel for EV3 . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
   3.5 Other Control Panels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
   3.6 Two Vehicle Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   3.7 General Vehicle Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

4 Get Started with Ch Mindstorms Package 22
   4.1 Introduction for CMindstorms Package . . . . . . . . . . . . . . . . . . . . . . . . . 22
   4.2 A Basic Ch Mindstorms Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
   4.3 Controlling the Speed of Motors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
   4.4 Making Motors Move . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
      4.4.1 Blocking and Non-Blocking Functions . . . . . . . . . . . . . . . . . . . . . . . 26
   4.5 Retrieving a Motor Angle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
   4.6 Use Sensors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
      4.6.1 Use Touch Sensor and Ultrasonic Sensor . . . . . . . . . . . . . . . . . . . . . 28
   4.7 How to Use Other Sensors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
   4.8 Advanced sensor functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
   4.9 Control a Group of Robots With Identical Movements . . . . . . . . . . . . . . . . . 34
   4.10 EV3 Only Features . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
      4.10.1 Drive Four Motors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
      4.10.2 Use More Sensors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

5 Controlling a Vehicle 36
   5.1 How to Make Your Robotic Vehicle Move Forward . . . . . . . . . . . . . . . . . . . 37
      5.1.1 Initialization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
      5.1.2 Declare a Robot Object . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
      5.1.3 Drive the Robot by Angle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
      5.1.4 Ending Your Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
   5.2 How to Make the Vehicle Move Backward . . . . . . . . . . . . . . . . . . . . . . . . 38
5.3 How to Make the Vehicle Turn Left/Right .............................................. 38
5.4 Advanced Mindstorms Motor Control .................................................. 39
  5.4.1 Manual Real Time Control Program .............................................. 39
5.5 Using Sensors ..................................................................................... 47
  5.5.1 Using Touch Sensor .................................................................. 48
  5.5.2 Using Ultrasonic Sensors ............................................................. 50
  5.5.3 Obstacle-Avoidance Vehicle ....................................................... 51
  5.5.4 Autonomous Control Program ................................................... 52
6 Controlling Non-Vehicle Lego Mindstorms Robots ................................. 56
  6.1 Controlling Robot in Machine Configuration .................................... 56
    6.1.1 Manual Real Time Control Program ....................................... 58
    6.1.2 Autonomous Control Program .............................................. 64
  6.2 Controlling Lego Mindstorms Robots in Humanoid ......................... 68
    6.2.1 Manual Real Time Control Program ....................................... 69
7 Play Music with Robots ....................................................................... 74
  7.1 Play Melody in Software .................................................................. 74
  7.2 Music Basics .................................................................................... 74
  7.3 Defining a Note in Software ............................................................. 75
  7.4 Playing an Array of Notes ................................................................. 76
  7.5 Write and Play Customized Song File .............................................. 77
A CMindstorms Class API ....................................................................... 78
  A.1 Data Types Used in CMindstorms Class ....................................... 78
    A.1.1 robotJointId_t ....................................................................... 78
    A.1.2 robotSensorPort_t .................................................................. 78
  A.2 Member Functions Available for General Purpose ........................... 78
B Member Functions Available only for Vehicle Configuration ............... 91
C Member Functions Available in CMindstormsGroup ............................ 93
D Notes Defined in Ch ........................................................................... 94
1 Introduction

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

The Ch Mindstorms 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 controller. The API converts the complex messaging tasks required to communicate with robots into easy to use functions, allowing users to focus their efforts on their robotic application, rather than the details of communication. The API of the Ch Mindstorms Package was designed to support and augment all of the functionality found in the LEGO Mindstorms controller. The Ch package further enhances the capabilities of the Lego Mindstorms controller by adding data collection and plotting capabilities. Additionally a 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 robot controller, the sensors, and the motors can be described in Figure 1. Once a robot is connected to a computer and a controlling program has started, the program instructions are sent from the computer to the robot. The robot controller will process these instructions perform appropriate tasks by sending commands to the motors or receiving data from the sensors. The robot 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 Lego Mindstorms Robot.](image)

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

In this guide, we will go over the basics of Ch Mindstorms programs. Lastly we will describe how to control non-vehicle configured Lego robots. After reading this guide, you will be ready to write your own Ch program to control your Lego Mindstorms robots.
2 Configuring Lego Mindstorms NXT/EV3 for Remote Control

Before you can use Ch Robot Controller or Ch Mindstorms package to control robots, you will need to enable Bluetooth on both computer and robot sides. Then NXTs/EV3s can be paired with the computer through Bluetooth. After successfully paired with the computer, robots are able to be connected and controlled by the computer.

2.1 Check Bluetooth Support on Computers

A Bluetooth-enabled computer should have a Bluetooth icon, as figure 2 shown, locating at the lower right corner of the “Desktop”. The bluetooth icons might have a little different for different operating systems such as Windows, Mac OS X, etc. If there is no bluetooth support on your computer, you need a Bluetooth dongle like Figure 3 shown.

![Figure 2: The Bluetooth icon.](image)

![Figure 3: A Bluetooth dongle.](image)

2.2 Enable Bluetooth on NXTs/EV3s

If Bluetooth is enabled and visible to other devices, the robot should show the symbol on the top left corner of its display. Otherwise, you will need to turn Bluetooth on manually on the NXT/EV3 brick.

Bluetooth status in NXTs/EV3s  NXTs/EV3s have the following Bluetooth status displayed on their screen,

- Bluetooth is enabled but not connected or visible to other Bluetooth devices.

- Bluetooth is enabled and visible to other Bluetooth devices but not connected.

- Bluetooth is enabled, visible to other Bluetooth devices and connected to other Bluetooth devices.

If non-of these status shows on your NXT/EV3 bricks, you need to enable Bluetooth manually.

Enable Bluetooth on NXT Bricks

- On the main screen of a NXT, use right and left buttons to select the word “Bluetooth” as the following figure shown and click the orange button.
• Then use the right and left buttons to select the word “On/Off” as the following figure shown and click the orange button.

• Use right and left buttons to select the word “On” as the following figure shown and push the orange button.
Enable Bluetooth on EV3 Bricks

- On the main screen of an EV3, use the right and left buttons to switch to “Settings” panel and use up and down buttons to select the word “Bluetooth” as the following figure shown and push the square button in the middle.

- In the popoped dialog, use up and down buttons to select the word “Bluetooth” as the following figure shown and push the square button in the middle. Please make sure both the word “Visibility” and “Bluetooth” has a little check mark at the end of the word.
2.3 Launch “Ch Robot Controller”

In Windows, you can launch the “Ch Robot Controller” by double clicking the application icon (Figure 4) on the Desktop of your computer. On the Mac OS X system, the “Ch Robot Controller” application is located inside the “Applications” folder in Finder.

Figure 4: “Ch Robot Controller” Icon.
2.4 Configure Robots

Figure 5: The Main Window of the “Ch Robot Controller” for EV3.

Figure 6: The Main Window of the “Ch Robot Controller” for NXT.
2.4.1 Scan Robots

After launching the “Ch Robot Controller”, the Graphical User's Interface (GUI) should display as Figure 5 for EV3 and Figure 6 for NXT. Click the “Scan Robot” button (Figure 7) on the GUI to start the robot scanner as Figure 8 shown.

Figure 7: The Robot Scanner Button.

Figure 8: The Robot Scanner Dialog.

Click “Scan” button on the dialog to search nearby robots and a list of robots should be listed. Move the robot closer to the computer and scan again if your robot is not listed.

2.4.2 Add and Pair a Robot

Select a robot from the list and click “Add” button to add the robot to the control panel. You will need to pair the robot with your computer for the first time. A dialog will pop up requiring a pairing code as the following figure shows.

Figure 9: A Dialog Requiring Pairing Code.
On the robot, you will need to accept the connection from the computer and type in the same code. It may take a few seconds or more before the pairing code dialog or connection confirmation dialog shows on the robot display. For NXTs, you can directly type in the pairing code as Figure 10 shows.

Figure 10: Type in pairing code on a NXT.

For EV3s, you need to confirm the connection (Figure 11a) before you can type in the pairing code (Figure 11b).

(a) Confirm Bluetooth Connection on a EV3. (b) Type in pairing code on a EV3.

Afterwards, the system will pair the robot automatically, and the robot will be added into the “Ch Robot Controller” control panel.

2.4.3 Rename a Robot

After a robot has been successfully added into the control panel, the name of the robot will show on the list as shown in Figure 12.
Figure 12: The List of Robot.

The default name of a robot is “EV3” for Lego Mindstorms EV3 and “NXT” for Lego Mindstorms NXT. The name will also display on the robot side as following figures show.

However, if multiple robots have been added, it is better to give each robot a unique name. Select a robot from the list and click “Rename” button on the “Ch Robot Controller” control panel. A dialog will pop up requiring a new name for the selected robot as the following figure shows.
Figure 14: A Dialog Requiring New Name for a Robot.

It may take a few seconds to rename a robot.
3 Control NXTs/EV3s via the GUI Controller

In this section, the Graphical User Interface (GUI) for controlling NXT/EV3 bricks will be introduced. The Ch Mindstorms controller consists of robot manager, robot controller and global settings.

3.1 Assemble a NXT/EV3 with Motors and Sensors

In order to use the Ch Mindstorms controller, the NXT/EV3 brick needs to be equipped with three motors and four sensors. As shown in Figure 15, three motors need to be physically connected to the ports with letters, such as A, B, and C, on the NXT/EV3 brick by cables provided by Lego. Similarly, sensors also need to be physically connected to the ports on the NXT/EV3 via cables, but the ports are in numbers, such as 1, 2, 3, and 4.

Figure 15: NXT/EV3 brick connects with three motors and four sensors.
3.2 Connect and Disconnect to NXTs/EV3s from the Ch Mindstorms Controller

![Image of the Robot Manager](image)

After adding robots from the robot scanner, a list of robots will show in the robot manager. Select a robot from the list as shown in Figure 16. The dot color on the left of the address indicates the connection status of the corresponding robot.

- red – not connected.
- yellow – connecting.
- green – connected.

Click “Connect” button to connect the selected robot. The red dot should turn green as soon as the robot is connected. On the robot side, the symbol ![Bluetooth Symbol] should display on the top left corner. By clicking “Disconnect” button, the connected robot will disconnect from the computer. **Please note that in order to run a Ch program that controls NXTs/EV3s, the NXTs/EV3s should not currently be connected to any programs on other computers.** Furthermore, the Bluetooth devices have a maximum limit of connected devices. The maximum limit is 7 devices connected simultaneously.

3.3 Robot Control Option

Figure 17 shows the “robot control option”. It allows users to choose three different control options, which are “Single Vehicle Control”, “Two Vehicle Control” and “General Vehicle Control”. Each control option has unique features which will be introduced in following sections.
3.4 Single Vehicle Control

The Single Vehicle Control contains five different control panels. They are “Motion Control”, “Sensors”, “Vehicle Control”, “Vehicle Turning” and “Vehicle Segment Control” and they will be introduced in this section.

3.4.1 Motion Control Panel

(a) The “Motion Control” Panel for NXT

(b) The “Motion Control” Panel for EV3

Figure 18: The “Motion Control” Panel of Ch Mindstorms Controller

Once a robot is connected by the Ch Mindstorms Controller, the motor angles and speeds of the robot are displayed as shown in Figure 18. This dialog is located under the first tab of Ch Mindstorms Controller, labeled “Motion Control”. The “Motion Control” tab can be used not only to display the information about robot’s motor positions but also to control the speeds and positions of the motors. The tab is divided up into 5 sections, including the “robot figure”, the “Individual Joint Control” section, the “Joint Position Control” section, the “Move Joints” section and the “Joint Speed Control” section.

Individual Joint Control Section  In the “Individual Joint Control” section, there are three sets of buttons as shown in Figure 19. Each set of buttons can be used to control the corresponding motor of the connected robot. When the up or down arrows are clicked, the robot begins to move the corresponding motor in either the positive or negative direction. The motor will move continuously until the stop button (located between the up and down arrows) is clicked.
The "Joint Position Control" section, shown in Figure 20, displays and controls the positions of each of motors of a robot. The motor positions are displayed in the numerical text browser located above each dial panel. The displayed motor positions are in units of degrees. The dial panels also display the positions of motors. Users can also drag the dial panels and the motor will move to the dropped position. The motor angles are updated automatically when motors are controlled either from the control panel or a Ch program. However, in Mac OS X, this section is not updated when a Ch program is running due to the communication performance issue.

The "Move Joints" section, as shown in Figure 21, contains three editors for typing angles for corresponding motors to move and buttons for starting moving motors. Once the button "Move" is clicked, the robot will move each motor by the corresponding angle submitted. The stop button causes the robot to stop where it is and the "zero" button causes each motor reset back to zero position.

The "Joint Speed Control" section, as Figure 22, displays and controls the motor speeds of the robot. The joint speeds are in units of degrees per second and a valid speed should be in the range of \([-650, 650]\) degree/second. To set a specific desired motor speed for a particular motor, the motor speed may be typed directly into the edit boxes below the sliders, and the "Set Speed" button should be clicked. Also, dragging each slider can also modify the speed of corresponding motor.

The "Sensors" panel displays relevant information regarding any sensors the NXT has access to. As Figure 23 shows, there are eight types of sensors the user can select. A sensor can be selected by clicking the check box on the left of the corresponding sensor. Only four sensors can be selected at the same time with different sensor ports on the NXT brick. The port number can be selected in the radio box, in the port section, for each sensor type. The value section of each sensor type will display the value from the corresponding sensor.
3.4.3 Sensor Control Panel for EV3

Different from sensor control panel for NXT, the one for EV3 will display information and status of sensors attached at four sensor ports on the EV3. If there is no sensor attached at a certain port, the block, corresponding to the port, will keep blank.
Some sensors have more than one mode so one can select modes for each sensor from the combo box in each sensor block on the sensor panel. For example, the color sensor can be used as color mode or light mode. If the color sensor is running as a color mode, it will return a number between 0 to 7 to indicate different colors. When the color sensor is used as a light mode, it will return the light intensity of the environment by percentage.
3.5 Other Control Panels

Figure 25 shows the “Vehicle Control” Panel. In this panel, the robot can be controlled as a vehicle. The “Direct Control” section can drive the vehicle forward and backward, turn the vehicle left and right and stop the vehicle. In addition, the linear speed of the vehicle can be set in the “Speed” section. The wheel radius of the vehicle can be set in “Wheel Radius” section. In the combo box, default wheel radius for the vehicle is listed and you can also customize the wheel radius by selecting “Custom” option in the box. Then a spin box will show up so that you can adjust the wheel radius manually. In the “Drive the Vehicle by Distance” section and the “Drive the Vehicle by Time” section, you can specify the distance and time you want the vehicle to drive. Also, by clicking the “Show Ch Code” button, a dialog pops up with corresponding Ch code for you to learn how Ch programs control robots.
Figure 26: The “Vehicle Turning” Panel

<table>
<thead>
<tr>
<th>Wheel Radius (in)</th>
<th>Track Width (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.10</td>
<td>4.54</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Turn Options</th>
<th>Turn Angle (deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn Left</td>
<td>90.00</td>
</tr>
<tr>
<td>Turn Right</td>
<td></td>
</tr>
</tbody>
</table>

- Motor B
- Motor C
- Track Width
“Vehicle Turning” panel (Figure 26) and “Vehicle Control Segments” have the similar interface with the “Vehicle Control” Panel.

In “Vehicle Turning” panel, you will be able to specify an angle for robot to turn. Also, some variables, such as wheel radius and track width, need to be set correctly to make it turn precisely.

“Vehicle Segment Control” allows you to set three segments and each segment has similar settings as “Vehicle Control” panel. In each segment, you can set a delay time.

### 3.6 Two Vehicle Control

You can switch to “Two Vehicle Control” from the Robot Control Option. The Figure 28 shows the “Two Vehicle Control” panel. In “Two Vehicle Control” panel, you are able to control two connected vehicles.
3.7 General Vehicle Control

Figure 29 shows the “General Vehicle Control” panel. In “General Vehicle Control”, you can specify different wheel sizes and speeds for each wheel of the vehicle.
Figure 29: The “General Vehicle Control” Panel
4 Get Started with Ch Mindstorms Package

In this chapter, the basics of controlling an NXT/EV3 via Ch program will be discussed. The basics include controlling motors, setting up sensors, and getting information from motors and sensors. The basic structure of a Ch robot program is shown in Figure 30 as a flowchart.

Figure 30: Flow Diagram of a basic program

Also, to successfully control Lego Mindstorms robots using Ch, it is important to practice good coding habits. The format of the Ch Mindstorms 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 robots. To help the user become acquainted with the Ch Mindstorms programs, sample programs will be presented in this section to illustrate the basics and minimum requirements of a Ch Mindstorms control program. The sample programs are located at CHHOME/package/chmindstorm/demos, where CHHOME is the Ch home directory, such as C:\Ch for Windows and /usr/local/Ch for Mac OS X. Therefore, for Windows, the demos are located at C:\Ch\package\chmindstorm\demos and /usr/local/ch/package/chmindstorms/demos for Mac OS X by default.

4.1 Introduction for CMindstorms Package

Each NXT has three motors while each EV3 has four. Both NXT and EV3 has four ports for sensors. In the CMindstorms Package, we have special symbols to represent the motors and the sensors, which are presented in the following tables.

Symbols for motors

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOINT1</td>
<td>PortA on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>JOINT2</td>
<td>PortB on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>JOINT3</td>
<td>PortC on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>JOINT4</td>
<td>PortD on the Lego Mindstorms EV3.</td>
</tr>
</tbody>
</table>
Symbols for sensors

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT1</td>
<td>PORT1 on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>PORT2</td>
<td>PORT2 on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>PORT3</td>
<td>PORT3 on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>PORT4</td>
<td>PORT4 on the Lego Mindstorms NXT/EV3.</td>
</tr>
</tbody>
</table>

For the motors, you will need to specify arguments like the speed ratios, direction of rotation or the target 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.

### 4.2 A Basic Ch Mindstorms Program

The first demo presents a minimal program which connects to a Lego Mindstorms robot and moves all the motors.

#### Source Code

```ch
/* File name: start.ch

* Move joints 2 and 3 by 360 degrees.*/
#include <mindstorms.h>
CMindstorms robot;

/* Move joints 2 and 3 (motors B and C) by 360 degrees */
robot.move(NaN, 360, 360);
```

Program 1: `start.ch` Source Code

#### Header files

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

```ch
#include <mindstorms.h>
```

#### Initialization

The following line initializes a new variable named `robot` which represents a Lego Mindstorms robot which we wish to control. The special variable is actually an instance of the `CMindstorms` class, which contains its own set of functions called “methods”, “member functions”, or simply “functions”.

```ch
CMindstorms robot;
```

The next line,

```ch
robot.move(NaN, 360, 360);
```

cause joints 2 and 3 of connected robot move 360 degrees. The member function `move()` expects input angles in degrees. If you want to use angles in radians, the conversion needs to be done via the function `radian2degree()`. The function is implemented in Ch with the code
```c
#include <math.h> /* For M_PI */
double radian2degree(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, `degree2radian()`.

### 4.3 Controlling the Speed of Motors

Now that we have already discussed how to include related header files, define variables for the program, connect/disconnect from robots and also set motors to zero positions. In this section, we will present a demo program, `setSpeed.ch`, to illustrate how to set speed ratios for the motors.

#### Source code

```c
/* File name: setSpeedRatios.ch
 * set speed ratios for joints of robot. */
#include <mindstorms.h>
CMindstorms robot;

/* set speed ratios and move */
robot.setJointSpeedRatios(0, 0.4, 0.4);
robot.setJointSpeedRatio(JOINT1, 0.5);
robot.move(360, 360, 360);

/* set speed and move */
robot.setJointSpeeds(0, 100, 100);
robot.setJointSpeed(JOINT1, 60);
robot.move(360, 360, 360);
```

Program 2: `setSpeed.ch` Source Code

#### Explanation

The first several lines,

```c
#include <mindstorms.h>
CMindstorms robot;
```

initializes the program, the “robot” variable, and connects to the robot. The next three lines,

```c
/* set speed ratios and move */
robot.setJointSpeedRatios(0, 0.4, 0.4);
robot.setJointSpeedRatio(JOINT1, 0.5);
robot.move(360, 360, 360);
```

sets the speed ratio for motors and makes motors move. The function `setJointSpeedRatios()` sets the speed ratio for JOINT1 as 0 and sets the speed ratios for JOINT2 and JOINT3 as 0.4. The function `setJointSpeedRatio()`, which can only set a speed ratio for one motor at a time, sets the speed ratio for JOINT1 as 0.5. The range of speed ratios is from 0 to 1. The next few lines,

```c
/* set speed and move */
robot.setJointSpeeds(0, 100, 100);
robot.setJointSpeed(JOINT1, 60);
robot.move(360, 360, 360);
```
set motors to a specified speed in degrees per second and make motors move. The function `setJointSpeeds()` sets the speed for JOINT1 to 0 degrees/second and sets the speed for JOINT2 and JOINT3 to 100 degrees/second. The function `setJointSpeed()`, which can only set a speed for one motor at a time, sets the speed for JOINT1 to 60 degrees/second.

4.4 Making Motors Move

Now that we have already discussed how to connect/disconnect and sets the speed ratios for the motors, this section will presents a full series of moving functions. In this section, a simple demo program will be presented to illustrate the series of functions for moving motors.

Source Code

```c
/* File name: move.ch
 * illustrate the full series of moving function */
#include <mindstorms.h>
CMindstorms robot;

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

/* Move a joint by user specified angle */
robot.moveJoint(JOINT1, 360);

/* Move a joint to absolute angle */
robot.moveJointTo(JOINT1, 720);

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

/* Move all joints to absolute angles */
robot.moveTo(360, 360, 360);
```

Program 3: move.ch Source Code

Explanation

The first part of code,

```c
#include <mindstorms.h>
CMindstorms robot;
```

initializes the program, connects to the robot. The next line,

```c
robot.resetToZero();
```

uses the function `resetToZero()` which is a member function of class `CMindstorms`. The function causes all motors of the connected robot reset to the zero position, which means the absolute angles of all motors will be set to zero. The next five lines,

```c
/* Move a joint by user specified angle */
robot.moveJoint(JOINT1, 360);

/* Move a joint to absolute angle */
robot.moveJointTo(JOINT1, 720);
```

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 */
robot.move(180, 360, 360);

/* Move all joints to absolute angles */
robot.moveTo(360, 360, 360);

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

4.4.1 Blocking and Non-Blocking Functions

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

Source Code

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

#include <mindstorms.h>
CMindstorms robot;

/* Non-blocking Function */
robot.moveJointNB(JOINT1, 360);
printf("This message will be printed on 
" "the screen when motor A is moving.\n");
robot.moveWait();

/* Blocking Function */
robot.moveJoint(JOINT1, 360);
printf("This message will be printed on 
" "the screen after motor A stopped moving.\n");
```

Program 4: blockNonblock.ch Source Code

Explanation

These five lines,

```c
/* Non-blocking Function */
robot.moveJointNB(JOINT1, 360);
printf("This message will be printed on 
" "the screen when motor A is moving.\n");
robot.moveWait();
```

use the non-blocking function moveJointNB(). The function printf() will print a message onto the screen while the motor A is moving. However,

```c
/* Blocking Function */
robot.moveJoint(JOINT1, 360);
printf("This message will be printed on 
" "the screen after motor A stopped moving.\n");
```

will only print out the message after the function moveJoint() finished, meaning that motor A has stopped. Most moving functions have both a blocking and non-blocking version. In Table 3, we list some simple blocking functions and their corresponding non-blocking functions.
### Table 3: Block and Non-block Functions

<table>
<thead>
<tr>
<th>Block Functions</th>
<th>Non-block Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>moveJoint()</td>
<td>moveJointNB()</td>
</tr>
<tr>
<td>moveJointTo()</td>
<td>moveJointToNB()</td>
</tr>
<tr>
<td>move()</td>
<td>moveNB()</td>
</tr>
<tr>
<td>moveTo()</td>
<td>moveToNB()</td>
</tr>
</tbody>
</table>

Also, detailed information for both blocking and non-blocking versions of the movement functions can be found in Appendix A on page 78.

## 4.5 Retrieving a Motor Angle

This demo demonstrates how to get a motor’s current angle in a Ch Mindstorms program. The angle is the absolute position in degrees.

### Source Code

```ch
/* Filename: getJointAngle.ch
 * Find the current joint angle of a joint */
#include <mindstorms.h>
CMindstorms robot;

/* Get the joint angle of the first joint */
double angle;
robot.getJointAngle(JOINT1, angle);

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

Program 5: getJointAngle.ch Source Code

### Explanation

The following lines,

```ch
/* Get the joint angle of the first joint */
double angle;
robot.getJointAngle(JOINT1, angle);
```

retrieve the current angle of motor A. JOINT1, defined in the header file mindstorms.h, represents portA on the robot. Detailed information for all ports descriptions can be found in Appendix A.1 on page 78. Finally, the last part of the program,

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

prints the value of the variable onto the screen.

## 4.6 Use Sensors

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

Source Code

```ch
/* File name: sensor.ch 
   * A brief introduction of using sensors of a robot. */

#include <mindstorms.h>

CMindstorms robot;

/* Variables to store values gotten from robot */
int touchValue;
double ultraValue;

/* set touch and ultrasonic sensor */
robot.setSensorTouch(PORT1, "Touch");
robot.setSensorUltrasonic(PORT4, "Centimeter");

/* get values collected by sensors from robot */
robot.getSensorTouch(PORT1, touchValue);
robot.getSensorUltrasonic(PORT4, ultraValue);

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

Program 6: sensor.ch Source Code

Explanation

The first part of the code,

```ch
#include <mindstorms.h>

CMindstorms robot;

/* Variables to store values gotten from robot */
int touchValue;
double ultraValue;
```

initializes the program, declares the variables and connects to the robot. The next part of code,

```ch
/* set touch and ultrasonic sensor */
robot.setSensorTouch(PORT1, "Touch");
robot.setSensorUltrasonic(PORT4, "Centimeter");
```

initializes two sensors for the connected robot. The function we used to initialize the sensors are `setSensorTouch()` and `setSensorUltrasonic()`, which takes two arguments. The first argument represents the port number on the robot, which is a variable type `robotSensorPort_t`. The second argument the function takes is the mode of a sensor. The mode for a touch sensor can be “Touch” or “Bumps” mode. In “Touch” mode, the value of 1 will be returned if the sensor is pressed while 0 will be returned if the sensor is not pressed. In “Bumps” mode, the value of 1 will be returned if the sensor is pressed and released otherwise 0 will be returned. The mode for an ultrasonic sensor can be “Centimeter” or “Inch”. The distance retrieved from the sensor is in centimeters if mode “Centimeter” is set. On the other hand, the distance is in inch if mode “Inch” is set. The next several lines,
```
/* get values collected by sensors from robot */
robot.getSensorTouch(PORT1, touchValue);
robot.getSensorUltrasonic(PORT4, ultraValue);
```

gets the collected values from the robot by using functions `getSensorTouch()` and `getSensorUltrasonic()`. In order to use the function, two arguments are necessary. One is the sensor port and another is the variable used to store the value retrieved from the robot. The last part,

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

displays the values retrieved from the robot onto the screen.

### 4.7 How to Use Other Sensors

Besides the touch sensor and ultrasonic sensor discussed in the previous section, the Ch package also supports several other sensors, which are light sensors, sound sensors, color sensors, angle sensors, force sensors and accelerometers. In this section, we will introduce the usage of those sensors.

#### Light Sensors and Sound Sensors

When setting up a sensor such as a light or sound sensor, the mode of the sensor will set the output as a percentage of full scale. Therefore, the values of light and sound sensors should range from 0 to 100 percent.

Light sensors can be used in two modes, which are “Ambient” and “Reflect”. In “Ambient” mode, the light sensor measures the strength of light from the environment. In “Reflect” mode, the sensor turns on a red light on it and measures the intensity of light reflected back from the red light.

Sound sensors have two modes, which are “dB” and “dBA”. “dB” is the abbreviation of decibels, a unit which can be used to express intensity of sound. In “dBA” mode, A-weighting is applied on measuring sound pressure level in effort to account for the relative loudness perceived by the human ear.

#### Color Sensor

The color sensor has three modes, which are “Color”, “Ambient” and “Reflect”. In “Color” mode, it can be used to detect different colors. The color sensor recognizes seven colors including black, blue, green, yellow, red, white, brown and no color. Each enumerate data retrieved represents a color as the following figure shown.

![Color Sensor Chart](image)

Figure 31: Color Sensor Chart.

The color sensor can be used as a light sensor if the mode “Ambient” or “Reflect” is set. Detailed information of these two modes can be found in the light sensor section.

#### Force Sensor

The force sensor will return a raw value from 0 to 800 and this number can be converted to grams. The relationship between the raw value and the force in grams is illustrated in the following figure.
The force sensor supported is a NXT Force Sensor (NFS1074) from the HiTechnic www.hitechnic.com.

Angle Sensor  Two modes, “Degree” and “Radian”, can be used. The angle sensor will return an angle value between 0 and 360 degrees if “Degree” mode is set while the value will be in radian if “Radian” mode is set. One thing that needs to be noticed is that the angle increments are in 2 degrees. The angle sensor supported is a NXT Angle Sensor (model NAA1030) from the HiTechnic www.hitechnic.com.

Temperature Sensor  The temperature sensor has a 6.4 cm-long probe and measures the temperature at the tip of the probe. It measures the temperature in Celsius from −20°C to 120°C or in Fahrenheit from −4°F to 248°F.

Gyroscope  A gyroscope is a sensor that detects rotational motions. The EV3 gyroscope measures both rotated angle in degree and rate of rotation in degree per second on a single axis shown on the sensor.

Accelerometer  The accelerometer uses the same function to initialize it. However, unlike other sensors, the function getSensorAccelerometer(port, x, y, z) is used to get values for three axises. The accelerometer supported is NXT Acceleration/Tilt Sensor (model NAC1040) from the HiTechnic www.hitechnic.com.

4.8 Advanced sensor functions

In the last section, the basic usage of sensors have been introduced. In order to satisfy some advanced usage of sensors, six sensor functions, listed below, are introduced in this section.

- waitUntil(port, sign, value)
- waitUntilAnd(port, sign1, value1, sign2, value2)
• waitUntilOr(port, sign1, value2, sign2, value2)
• repeatUntil(function, port, sign, value)
• repeatUntilAnd(function, port, sign1, value1, sign2, value2)
• repeatUntilOr(function, port, sign1, value2, sign2, value2)

**waitUntil(port, sign, value)**
This function is a function waiting for a sensor to return a value that satisfies the given condition. For example:

```c
#include <mindstorm.h>

int main()
{
  CMindstorms robot;

  robot.setSensorTouch(PORT1, "Touch");

  printf("wait touch sensor pressed.\n");
  robot.waitUntil(PORT1, "==", 1);
  robot.move(NaN, 360, 360);

  return 0;
}
```

In the example above, a robot is connected and port1 is initialized as a touch sensor. Then a program is blocked once the `waitUntil` function is called. The function will wait until the value returned from the sensor attached at port1 equals to “1”. The first argument of `waitUntil` is the port number, at which the sensor is attached. The second argument can be “==”, “!="”, “>”, “<”, “>=” and “<=". The last argument is the critical value of the waiting condition for the function.

**waitUntilAnd(port, sign1, value1, sign2, value2)**
This function is another function blocking a program until the value returned by the sensor attached to a given port that satisfies both two given conditions. For example:

```c
#include <mindstorm.h>
#include <string.h>

int main()
{
  CMindstorms robot;

  robot.setSensorUltrasonic(PORT4, "Centimeter");
  robot.setJointSpeedRatios(0, -0.2, -0.2);
  robot.moveForeverNB();

  printf("wait ultrasonic sensor.\n");
  robot.waitUntilAnd(PORT4, ">", 10, "<", 20);

  return 0;
}
```

In the example above, a robot is connected to port4 as an ultrasonic sensor. Then the program is blocked once the `waitUntilAnd` function is called. The function will wait until the ultrasonic sensor returns a value which is greater than 10 and less than 20. So the function will block a program until the value, returned by the sensor attached to a given port, is greater than “value1” and less than “value2”.

31
waitUntilOr(port, sign1, value1, sign2, value2)
Same as the waitUntilAnd function, waitUntilOr is a waiting function, which has two conditions.
However, the function will unblock a program once either condition is satisfied. For example:

```c
#include <mindstorm.h>
#include <string.h>

int main(){
    CMindstorms robot;

    robot.setSensorUltrasonic(PORT4, "Centimeter");
    robot.setJointSpeedRatios(0, -0.2, -0.2);
    robot.moveForeverNB();

    printf("wait ultrasonic sensor.\n");
    robot.waitUntilOr(PORT4, ">", 20, "<<", 10);

    return 0;
}
```
In the example above, a robot is connected to port4 as an ultrasonic sensor. Then a program is blocked once the waitUntilOr function is called. The function will wait until the ultrasonic sensor returns a value which is greater than 20 or less than 10. So the function will block the program until the value, returned by the sensor attached to the given port, is greater than “value1” or less than “value2”.

repeatUntil(function, port, sign, value)
This function repeats the given function until a sensor return a value that satisfies the given condition. For example:

```c
#include <mindstorm.h>

void dowork(){
    robot.driveAngle(30);
}

int main(){
    CMindstorms robot;

    robot.setSensorTouch(PORT1, "Touch");
    printf("wait touch sensor pressed.\n");
    robot.repeatUntil(dowork, PORT1, "==", 1);

    return 0;
}
```
In the example above, a robot is connected and port1 is initailized as touch sensor. Then a program is blocked once the repeatUntil function is called. The first argument is the name of the function for repeating. The given function will be repeated until the value returned from the sensor attached at port1 equals “1”. The second argument of waitUntil is the port number, where the sensor is attached. The third argument can be “==”, “!”, “>”, “<”, “>=” and “<=”. The last argument is the critical value of the waiting condition for the function.

repeatUntilAnd(function, port, sign1, value1, sign2, value2)
This function also repeats the given function until a sensor return a value that satisfies the given
two conditions are required to be satisfied to unblock the program. For example:

```c
#include <mindstorm.h>
#include <string.h>

void dowork()
{
    robot.driveAngle(30);
}

int main()
{
    CMindstorms robot;
    
    robot.setSensorUltrasonic(PORT4, "Centimeter");
    
    printf("wait ultrasonic sensor.\n");
    robot.repeatUntilAnd(dowork, PORT4, ">", 10, "<", 20);
    
    return 0;
}
```

In the example above, a robot is connected to port4 as an ultrasonic sensor. Then a program is blocked and the “dowork” function will be repeated once the `repeatUntilAnd` function is called. The function will wait until the ultrasonic sensor returns a value which is greater than 10 and less than 20. So the function will blocks a program until the value, returned by the sensor attached to a given port, is greater than “value1” and less than “value2”.

`repeatUntilOr(function, port, sign1, value1, sign2, value2)`

Same as the `repeatUntilAnd` function, `repeatUntilOr` is a repeating function, which has two conditions. However, the function will unblock a program once either condition is satisfied. For example:

```c
#include <mindstorm.h>
#include <string.h>

void dowork()
{
    robot.driveAngle(30);
}

int main()
{
    CMindstorms robot;
    
    robot.setSensorUltrasonic(PORT4, "Centimeter");
    
    printf("wait ultrasonic sensor.\n");
    robot.repeatUntilOr(dowork, PORT4, ">", 20, "<", 10);
    
    return 0;
}
```

In the example above, a robot is connected to port4 as an ultrasonic sensor. Then the program is blocked and the “dowork” function will be repeated once the `repeatUntilOr` function is called. The function will wait until the ultrasonic sensor returns a value which is greater than 20 or less than 10. So the function will blocks a program until the value, returned by the sensor attached to a given port, is greater than “value1” or less than “value2”.

33
4.9 Control a Group of Robots With Identical Movements

In previous sections, we have learned how to control a single robot. In this section, we will learn how to control multiple robots as a group.

Source Code

```c++
/* File name: group.cpp */
* Drive a group of robots forward by 360 degrees */

#include <mindstorms.h>

/* Declare robots and a group */
CMindstorms robot1, robot2;
CMindstormsGroup group;

/* Add two robots into a group */
group.addRobot(robot1);
group.addRobot(robot2);

/* Drive the group of robots forward by 360 degrees */
group.driveForward(360);
```

Program 7: group.ch Source Code

Declare Robots and a Group Object

The lines,

```c++
CMindstorms robot1, robot2;
CMindstormsGroup group;
```
defines two robot objects as we learned before and a `CMindstormsGroup` object `group`. Before we can use the group object, robots need to be added into the group. The following lines of code,

```c++
group.addRobot(robot1);
group.addRobot(robot2);
```
add two robots into the group. The last line of code,

```c++
group.driveForward(360);
```
drives the group of robots forward by 360 degrees.

4.10 EV3 Only Features

Since the EV3s have new features different from NXTs, we will learn how to use functions for EV3 in this section.

4.10.1 Drive Four Motors

Source Code

```c++
/* File name: ev3_move.ch */
* Move motors and get motor angles for EV3 */

#include <mindstorms.h>

/* Declare robot and other variables */
CMindstorms robot;
```
Move Four Motors

The following line of code,

```java
robot.move(360, 360, 360, 360);
```

moves the four motors by 360 degrees. The `move()` function can have the fourth argument for the fourth motor on the EV3.

Get Four Motor Angles

The following lines of code,

```java
robot.getJointAngles(angle1, angle2, angle3);
robot.getJointAngle(JOINT4, angle4);
```

get motor angles from the EV3. Unlike the `move()` function, `getJointAngles()` can only have three arguments. In order to get the angle for fourth motor, the function `getJointAngle()` need to be used to get the last motor angle.

4.10.2 Use More Sensors

The infrared sensor is only supported for EV3. The infrared sensor uses the light waves reflected back from an object to estimate the distance between the sensor and that object. The distance is measured in percentage between 0 (very close) to 100 (far away).
5 Controlling a Vehicle

The actuators available are motors. Normally, you can only control the speed and direction of the connected motors. For a two wheeled vehicle, there are two ways that the vehicle can be controlled. In addition to moving the robot by controlling the individual motors as shown in Figure 33, you can also use a set of Ch functions written specifically to control a robot in vehicle configuration. The diagram of the vehicle and the motor ports is shown in Figure 34. 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 Mindstorms program to move the robot forward. Please make sure your robotic vehicle are configured according to to Figure 34 to run our demonstration programs.
5.1 How to Make Your Robotic Vehicle Move Forward

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

```c
#include <mindstorms.h>
CMindstorms robot;

/* Set Speed Ratio */
robot.setJointSpeedRatios(NaN, 0.25, 0.25);

/* drive the motor B and C by 360 degree */
robot.driveAngle(360);
```

Program 9: forward.ch Source Code

5.1.1 Initialization

In the beginning of a Ch Mindstorms robot 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 for running the program. Essential header files for the robot include:

```c
#include <mindstorms.h>
```

The `mindstorm.h` is the essential header file for Ch Mindstorms control functions and variables.

5.1.2 Declare a Robot Object

The robot status, sensor/encoder data, and input/output protocols are stored in a C++ class called `CMindstorms`. Objects in this class must be created in every Ch program for controlling Lego Mindstorms robots. Therefore, in the beginning of your code, you must define a `CMindstorms` class and it will connect to the robot automatically. The following line of code creates the variable `robot` that is used to store data and control the robot.

```c
CMindstorms robot;
```

5.1.3 Drive the Robot by Angle

The following code:

```c
/* Set Speed Ratio */
robot.setJointSpeedRatios(NaN, 0.25, 0.25);

/* drive the motor B and C by 360 degree */
robot.driveAngle(360);
```

sets motor speed ratios for Motor B and C and drives the robot by angle. Figure 35 shows the robotic vehicle moving forward by actuating both wheels with the same speed ratio, which is how
the function `driveAngle()` works. By default, the ports for the two wheels on the vehicle are `MotorB (JOINT2)` and `MotorC (JOINT3)`.

![Figure 35: Top-down view of robotic vehicle with two wheels](image)

5.1.4 Ending Your Program

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

5.2 How to Make the Vehicle Move Backward

The function works by actuating both wheels backward at the same speed, as shown in Figure 36. The code fragment is shown below:

```java
robot.driveAngle(-360);
```

In order to drive the robot backward by angle, a negative angle should be put in the function `driveAngle()`. The modified program is called: `backward.ch`.

5.3 How to Make the Vehicle Turn Left/Right

To make your robotic vehicle turn or rotate in place, the wheels must be spun in the opposite direction at the same speed. For example, to rotate the vehicle to the left, the right wheel must be spun forward, while the left wheel spins at the same speed in reverse. Figure 37 shown below shows the vehicle turning left or right by actuating the wheels in opposite direction.
To make the vehicle rotate in place, we can use the functions `turnLeft()` and `turnRight()`. An example of how to use the functions are shown below:

```c
/* set speed ratio */
robot.setJointSpeedRatios(NaN, 0.50, 0.50);

/* rotate left */
robot.turnLeft(90, 1, 4.5);

// or

/* set speed ratio */
robot.setJointSpeedRatios(NaN, 0.50, 0.50);

/* rotate right */
robot.turnRight(90, 1, 4.5);
```

The first parameter in the functions is the desired turn angle of robot. The second parameter is the radius of the wheels and the last parameter is distance between two wheels.

### 5.4 Advanced Mindstorms Motor Control

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

#### 5.4.1 Manual Real Time Control Program

Manual real time control program allows you to control your robotic 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 vehicle’s motion. The vehicle real time control (RTC) program, `vehicle_rtc.ch`, prints out a user interface for the user to use while executing the program. Figure 38 is the user interface of the robotic vehicle RTC program.

In Figure 38, the user interface of the 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 vehicle RTC program, the program uses a `if-else if-else` statement to performs a fragment of code that send commands
The key ‘‘w’’ is to control the vehicle to move forward.

The key ‘‘s’’ is to control the vehicle to move backward.

The key ‘‘a’’ is to control the vehicle to turn left.

The key ‘‘d’’ is to control the vehicle to turn right.

The key ‘‘x’’ is to stop the vehicle motors.

The key ‘‘1’’ is to set the vehicle motor speed ratio from 0.25.

The key ‘‘2’’ is to set the vehicle motor speed ratio from 0.5.

The key ‘‘3’’ is to set the vehicle motor speed ratio from 0.75.

The key ‘‘4’’ is to set the vehicle 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 vehicle 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 if-else statements. The whole vehicle RTC program is shown in Program 10. Please make sure your vehicle are configured according to Figure 34 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
 * The purpose of this demo is to 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>
```c
#include <stdio.h>
#include <mindstorms.h>

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

/* 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\|\[-[d] [2] Set SpeedRatio to 0.50 ");
printf("\n \| [3] Set SpeedRatio to 0.75 ");
printf("\n[v] [s] [4] Set SpeedRatio to 1
");
printf("Please Enter command:");

/* Control loop. Interprets user command and does action */
while (!quit){
    key = _getch();
    if(key == 'w'){
        // up
        robot.setJointSpeedRatios(NaN, speedRatio, speedRatio);
        robot.moveJointForeverNB(JOINT2);
        robot.moveJointForeverNB(JOINT3);
        movemode = 'w';
    }
    else if(key == 's'){
        // down
        robot.setJointSpeedRatios(NaN, -speedRatio, -speedRatio);
        robot.moveJointForeverNB(JOINT2);
        robot.moveJointForeverNB(JOINT3);
        movemode = 's';
    }
    else if(key == 'd'){
        // right
        robot.setJointSpeedRatios(NaN, -speedRatio, speedRatio);
        robot.moveJointForeverNB(JOINT2);
        robot.moveJointForeverNB(JOINT3);
        movemode = 'd';
    }
    else if(key == 'a'){
        // left
        robot.setJointSpeedRatios(NaN, speedRatio, -speedRatio);
        robot.moveJointForeverNB(JOINT2);
        robot.moveJointForeverNB(JOINT3);
        movemode = 'a';
    }
    else if(key == 'q'){
        // forward-left
        robot.setJointSpeedRatios(NaN, speedRatio, 0.7*speedRatio);
        robot.moveJointForeverNB(JOINT2);
        robot.moveJointForeverNB(JOINT3);
        movemode = 'q';
    }
    else if(key == 'e'){
        // forward-right
        robot.setJointSpeedRatios(NaN, 0.7*speedRatio, speedRatio);
        robot.moveJointForeverNB(JOINT2);
        robot.moveJointForeverNB(JOINT3);
        movemode = 'e';
    }
    else if(key == 'x'){
        // stop and hold joints
    }
}
```
robot.holdJoints();
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 three lines.

```c
#include <conio.h>
#include <stdio.h>
#include <mindstorms.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 user input.
- The header `nxt.h` provides the program with general functions of the Ch Mindstorms Control Package.

Declaring variables After including the headers, variables are declared.

```c
CMindstorms robot;
double speedRatio = 0.25; //speedRatio of the motors. (default to 25)
int quit = 0;            //used by quit case to exit the loop
char key = 'x',        //stores the input from the user
    movemode = 'x';  //stores the last movement command
```

- The `CMindstorms` class stores the connection status, sensor data, and motor data of the robot. Also, the class includes the functions for controlling the robot.
• The double variable \texttt{speedRatio} stores the speed ratio of the motor.
• The integer variable \texttt{quit} is used to check if the user wants to quit the program.
• The character variable \texttt{key} stores the input from the user.
• The character variable \texttt{movemode} stores the last command that the user used.

\textbf{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. The vehicle RTC program used \texttt{printf()} command is used to display the user interface for the user to read.

\begin{verbatim}
/* 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:");
\end{verbatim}

\textbf{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 vehicle begins with the robot control code block. In the robot control code block, a \texttt{while} loop is implemented to allow the user to control the vehicle continuously until the program is terminated. Within the \texttt{while} loop, the program grabs an input from the user, and then decide what to do with the input using the \texttt{if-else if-else} statement. A flowchart for the vehicle RTC control program is shown in Figure 39.

The program fragment below shows the beginning and the end of the \texttt{while} loop for Program 10:

\begin{verbatim}
/* Control loop. Interprets user command and does action */
while (!quit){
    key = _getch();
    if(key == 'w'){ //up
        robot.setJointSpeedRatios(NaN, speedRatio, speedRatio);
        robot.moveJointForeverNB(JOINT2);
    }
    // Other commands...
}
\end{verbatim}

Figure 39: A flowchart for the vehicle RTC Control Program.
robot.moveJointForeverNB(JOINT3);
movemode = 'w';
}

else if(key == 's'){
    // down
    robot.setJointSpeedRatios(NaN, -speedRatio, -speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 's';
}

else if(key == 'd'){
    // right
    robot.setJointSpeedRatios(NaN, -speedRatio, speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'd';
}

else if(key == 'a'){
    // left
    robot.setJointSpeedRatios(NaN, speedRatio, -speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'a';
}

else if(key == 'q'){
    // forward-left
    robot.setJointSpeedRatios(NaN, speedRatio, 0.7*speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'q';
}

else if(key == 'e'){
    // forward-right
    robot.setJointSpeedRatios(NaN, 0.7*speedRatio, speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'e';
}

else if(key == 'x'){
    // stop and hold joints
    robot.holdJoints();
    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");
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.

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 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 robot.

**Directional commands**  The movements are controlled using the ‘w’, ‘s’, ‘a’, and ‘d’ format. As shown in Figure[38] 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 if-else statement will run the codes for the case ‘w’. The program fragment for case ‘w’ is shown below:

```java
if(key == 'w'){ // up
    robot.setJointSpeedRatios(NaN, speedRatio, speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'w';
}
```

In case ‘w’, the program will run joints JOINT2 and JOINT3 forward at the velocity speedRatio. Next, the ‘w’ key is stored in the variable movemode, which will be used to indicate the current mode for the vehicle.

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

```java
else if(key == 's'){ // down
    robot.setJointSpeedRatios(NaN, -speedRatio, -speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 's';
}
```

In case ‘s’, the program will run the joints JOINT2 and JOINT3 backward at the velocity speedRatio. Next, the ‘s’ key is stored in the variable movemode, which will be used to indicate the current mode for the vehicle.

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

```java
else if(key == 'd'){ // right
    robot.setJointSpeedRatios(NaN, -speedRatio, speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'd';
}
```

In case ‘d’, the program will actuate joint JOINT2 backward at velocity speedRatio and actuate joint JOINT3 forward at velocity speedRatio. Next, the ‘d’ key is stored in the variable movemode, which will be used to indicate the current mode for vehicle.

When the key ‘a’ has been pressed, the if-else if-else statement will run the code for the case ‘a’. The program fragment for case ‘a’ is shown below:
else if(key == 'a') {
    // left
    robot.setJointSpeedRatios(NaN, speedRatio, -speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'a';
}

In case ‘a’, the program will actuate joint JOINT2 forward at velocity speedRatio and actuate joint JOINT3 backward at velocity speedRatio. Next, the ‘a’ key is stored in the variable movemode, which will be used to indicate the current mode for vehicle.

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

else if(key == 'q') {
    // forward-left
    robot.setJointSpeedRatios(NaN, speedRatio, 0.7*speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'q';
}

In case ‘q’, the program will actuate joint JOINT2 forward at velocity speedRatio and actuate joint JOINT3 forward at velocity 0.7*speedRatio. Next, the ‘q’ key is stored in the variable movemode, which will be used to indicate the current mode for vehicle.

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

else if(key == 'e') {
    // forward-right
    robot.setJointSpeedRatios(NaN, 0.7*speedRatio, speedRatio);
    robot.moveJointForeverNB(JOINT2);
    robot.moveJointForeverNB(JOINT3);
    movemode = 'e';
}

In case ‘e’, the program will actuate joint JOINT2 forward at velocity 0.7*speedRatio and actuate joint JOINT3 forward at velocity speedRatio. Next, the ‘e’ key is stored in the variable movemode, which will be used to indicate the current mode for vehicle.

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

else if(key == 'x') {
    // stop and hold joints
    robot.holdJoints();
    movemode = 'x';
}

In case ‘x’, the program will set the motor in JOINT2 and JOINT3 to zero velocity, and then set the motor to idle mode. Next, the ‘x’ key is stored in the variable movemode, which will be used to indicate the current mode for vehicle. The case ‘x’ will stop the motor and keep it hold 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 38, 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 keys has its own fragment of code.

else if(key == '1') {
    // speed ratio 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);
}

For each of the cases, 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. The speed keys allow the program to change the speed of the 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 if-else statement will perform a fragment of code for case ‘r’. The program fragment for case ‘r’ is shown below:

```c
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 are 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:

```c
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 Sensors

Sensors takes a physical quantity and converts it to signals which can be read by the robot or the computer. Sensors allow communication between the outside environment and the robot. The Lego Mindstorms robots have 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 vehicle are configured according to Figure[40] to run these demonstration programs.
5.5.1 Using Touch Sensor

After you have connected a Lego Mindstorms robot to your PC, you will need to set up the sensor you would like to include. Examples of adding touch and ultrasonic sensors to the vehicle is introduced in the section.

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

```c
/* File name: touchsensor.ch

The purpose of this demo is to demonstrate the CH Mindstorms Control Package’s ability to control the robot to use the touch sensor. */

#include <mindstorms.h>

CMindstorms robot;
int touchValue = 0;

/* Set sensor types */
robot.setSensorTouch(PORT1, "Touch");

/* set joint speed ratios */
robot.setJointSpeedRatios(NaN, 0.25, 0.25);

/* Move Robot Forward */
robot.moveJointForeverNB(JOINT2);
robot.moveJointForeverNB(JOINT3);

/* Commands: */
while(1){
    /* Get touch sensor data */
    robot.getSensorTouch(PORT1, touchValue);
    /* Iftouch sensor is triggered */
    if(touchValue){
        /* Move backward */
```
Checking Touch Sensor Connection

Program 11 is similar to Program 9, the only change is the addition of the use of the touch sensor. In Program 11, the robotic vehicle will move forward until the touch sensor is triggered, after which it will back up and stop. The fragment of the initialization of the touch sensor is shown below.

```c
/* Set sensor types */
robot.setSensorTouch(PORT1, "Touch");
```

In this fragment, the program use the `setSensorTouch()` function to set the touch sensor to PORT1 with mode “Touch”.

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 decides what to do with the data. A flowchart of the `while` loop for the touch sensor demo program is shown in Figure 41.

![Figure 41: A flowchart of the while loop in Program 11](image)

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 robot. Then the program will move the robotic vehicle backward and disconnect the robot. The `while` loop of the touch sensor demonstration program is described below:

```c
/* Commands: */
while(1){
    /* Get touch sensor data */
    robot.getSensorTouch(PORT1, touchValue);
```
/* If touch sensor is triggered */
if (touchValue) {
    /* Move backward */
    robot.setJointSpeedRatios(NaN, -0.25, -0.25);
    robot.moveJointTimeNB(JOINT2, 2);
    robot.moveJointTime(JOINT3, 2);
    /* quit the while loop */
    break;
}
}

In this while loop, the program uses the getSensorTouch() function to get the data from PORT1, which is the port for the touch sensor according to Figure 40. 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 vehicle backward. After moving the vehicle 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 Ultrasonic Sensors

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

```c++
/* File name: ultrasonicsensor.ch */
/* The purpose of this demo is to demonstrate the CH Mindstorms Control Package’s ability to control the robot to use the ultrasonic sensor. */

#include <mindstorms.h>

CMindstorms robot;
double ultraValue;

/* Set ultrasonic sensor with centimeter mode */
robot.setSensorUltrasonic(PORT4, "Centimeter");

/* Commands: */
while(1){
    /* Move forward (constantly)*/
    robot.driveForeverNB();

    /* Get distance from ultrasonic sensor.
    * If obstacle is really close,
    * drive backward for 360 degree and quit */
    robot.getSensorUltrasonic(PORT4, ultraValue);
    if (ultraValue < 10) {
        robot.driveAngle(-360);
        break;
    }
}
```

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 robotic vehicle. Similar to the initialization of the touch sensor, the initialization of the ultrasonic sensor is shown below:

```c
/* Set ultrasonic sensor with centermeter mode */
robot.setSensorUltrasonic(PORT4, "Centimeter");
```

In this program fragment, the `setSensorUltrasonic()` function initializes the ultrasonic sensor to PORT4 with mode “Centimeter”.

Contents in the While Loop

In Program 12, the while loop uses the ultrasonic sensor to detect distances between the vehicle and the obstacles in front of the vehicle. The ultrasonic sensor will detect distances and the program code reacts to the data. The `while` loop code block for Program 12 is shown below:

```c
/* Commands : */
while(1){
    /* Move forward (constantly)*/
    robot.driveForeverNB();

    /* Get distance from ultrasonic sensor.
    * If obstacle is really close,
    * drive backward for 360 degree and quit */
    robot.getSensorUltrasonic(PORT4, ultraValue);
    if (ultraValue < 10) {
        robot.driveAngle(-360);
        break;
    }
}
```

The program sets the robot to move forward. Similar to Program 11, the `while` loop in Program 12 also gathers the sensor data using the `getSensorUltrasonic()` function from the ultrasonic sensor attached at PORT4. 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.

5.5.3 Obstacle-Avoidance Vehicle

A demo program is presented in this section to illustrate how to use the advanced sensor functions to make an obstacle-avoidance vehicle.

Source Code

```c
/* File name: obstacleAvoid.ch
   To illustrate waitUntil functions */

#include <mindstorms.h>

CMindstorms robot;

/* Initialize Ultrasonic Sensor */
robot.setSensorUltrasonic(PORT4, "Centimeter");

/* Set Motor Speed Ratio */
robot.setJointSpeedRatios(NaN, 0.2, 0.2);```
/* Move forward and avoid obstacle */
while(1){
    robot.driveForeverNB();
    robot.waitUntil(PORT4, "<", 20.0);
    robot.driveAngle(-360);
    robot.turnLeft(90, 1, 4.5);
}

Program 13: obstacleAvoid.ch Source Code

Initialize the Program
The first part of the code is:

```c
#include <mindstorms.h>

CMindstorms robot;
/* Initialize Ultrasonic Sensor */
robot.setSensorUltrasonic(PORT4, "Centimeter");
/* Set Motor Speed Ratio */
robot.setJointSpeedRatios(NaN, 0.2, 0.2);
```

which declares a Lego Mindstorms robot variable, connects to the robot and initializes a ultrasonic
sensor attached at port 4 and sets the joint speed ratio of motors.

Contents in the while loop
The main part of the program:

```c
/* Move forward and avoid obstacle */
while(1){
    robot.driveForeverNB();
    robot.waitUntil(PORT4, "<", 20.0);
    robot.driveAngle(-360);
    robot.turnLeft(90, 1, 4.5);
}
```

is a while loop, in which vehicle will keep driving forward and checking if any obstacle in the front.
If the distance to the obstacle is less than the given value, the vehicle move backward by 360 degrees
and turn left by 90 degrees.

5.5.4 Autonomous Control Program

In the previous sections, we thoroughly covered the manual real time control program, which allows
you to remote control your robotic vehicle with your keyboard. In this section, we will talk about the
autonomous control program for the vehicle. In an autonomous control program, the robot, which
is the vehicle, must be able to move around by itself without human commands or interventions.
In order to achieve such task, the vehicle 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 42.

- 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.
Figure 42: A diagram for sense plan act.

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 42. Every cycle, the information is gathered from the sensor and sent back to the computer. The computer will decide what to do depending on the sensor data. The autonomous control program for the robotic vehicle is described in Program 14.

Source Code

```c
#include <conio.h>
#include <stdio.h>
#include <mindstorms.h>

CMindstorms robot;
int touchValue; // used to store touch sensor data
double ultraValue; // used to store ultrasonic sensor data
char key = 'x', // stores user input
        movemode = 'x'; // stores last movement command

/* Set sensor types */
robot.setSensorTouch(PORT1, "Touch");
robot.setSensorUltrasonic(PORT4, "Centimeter");

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 */
    robot.getSensorTouch(PORT1, touchValue);
    if (touchValue){
        robot.driveAngle(-720);
        robot.turnLeft(90, 1.1, 4.5);
    }
    /* Get distance from UltraSonic sensor. */
}
```

Source Code

/* File name: vehicle_auto.ch
   The purpose of this demo is to demonstrate the CH Mindstorms
   Control Package's ability to control the vehicle robot model
   autonomously, as well as demonstrate how use sensor data from
   the robot to control it's actions. */

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

CMindstorms robot;
int touchValue; // used to store touch sensor data
double ultraValue; // used to store ultrasonic sensor data
char key = 'x', // stores user input
        movemode = 'x'; // stores last movement command

/* Set sensor types */
robot.setSensorTouch(PORT1, "Touch");
robot.setSensorUltrasonic(PORT4, "Centimeter");

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 */
    robot.getSensorTouch(PORT1, touchValue);
    if (touchValue){
        robot.driveAngle(-720);
        robot.turnLeft(90, 1.1, 4.5);
    }
    /* Get distance from UltraSonic sensor. */
}
In Program 14, the sensors 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 sensors and actuators for the program is shown in Figure 40. Please make sure the robotic vehicle are configured according to to Figure 40 to run Program 14.

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 robot. 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.

```c
/* 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 vehicle. When the vehicle detect obstacles in the front, the touch sensor will be triggered. The autonomous program will notice that the touch sensor is triggered and command the vehicle to steer away from the obstacle. The program fragment of the touch sensor is shown below.

```c
/* get touch sensor. If pressed reverse and turn left */
robot.getSensorTouch(PORT1, touchValue);
if (touchValue){
    robot.driveAngle(-720);  
    robot.turnLeft(90, 1.1, 4.5);
}
```

In the first line of this fragment, the vehicle gathers data from PORT1, 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 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 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.

```java
/* Get distance from UltraSonic sensor.
 * Turn left if really close.*/
robot.getSensorUltrasonic(PORT4, ultraValue);
if (ultraValue < 10){
    robot.turnLeft(90, 1.1, 4.5);
}
```

In the first line of this fragment, the vehicle gathers data from PORT4, 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, the 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.

```java
/* drive vehicle forward forever */
robot.driveForeverNB();
```

This program fragment commands the vehicle to move forward continuously by setting both motors rotating positively at the variable speedRatio.
6 Controlling Non-Vehicle Lego Mindstorms Robots

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

6.1 Controlling Robot in Machine Configuration

![Lego Mindstorms Robot in Machine Configuration](image)

Figure 43: Lego Mindstorms Robot in Machine Configuration.

The robot also comes with two other forms, one of these forms is the machine form as shown in Figure 43. In this section, a demo program for controlling machine configuration will be discussed. Compared to the vehicle configuration, the machine configuration uses three motors to manipulate its arm and two sensors for detection. The location and description of the components of the machine configuration is shown in Figure 44.
As shown in Figure 44, 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 machine uses the light sensor to sense the color of the object it is handling and uses the touch sensor to sense if it has successfully grabbed an object. Please use the sensor/motor port configuration shown in Figure 45. The ch demo programs for the machine configuration is described in this section.
6.1.1 Manual Real Time Control Program

In this section the manual real time control for the machine will be introduced and described. The manual RTC for the machine allows the user to control the machine manually via the keyboard. The user interface of the RTC program for the machine is shown in Figure 46. The user interface and the user commands for the machine RTC program is a bit different compared to the vehicle RTC program. Instead of controlling the direction by pressing one key, the user will be required to press a key for 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 machine RTC program, the program uses an if-else if-else statement to performs a fragment of code that send commands to move motors performing a task. If a directional key or a set speed key has been pressed, program will ask for an user input for a number to set the arm to move at an angle or set the speed. If a discrete task key is pressed, like open or close claw, the program will not ask for an 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 arm to move up.
- The key "s" is to control the arm to move down.
- The key "a" is to control the arm to turn left.
- The key "d" is to control the arm to turn right.
- The key "q" is to control the claw to open.
- The key "e" is to control the claw to close.
- The key "x" is to stop the motors.
- The key "r" is to exit the manual RTC program.
- The key "f" is to set the motor speed.

The machine RTC program is described in Program 15. 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

The purpose of this demo is to 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 <mindstorms.h> // this is the control package.

CMindstorms robot;

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 touchValue, colorValue;

double gear_ratio = (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\n", gear_ratio);

/* Initialize sensor */
robot.setSensorTouch(PORT1, "Touch");
robot.setSensorColor(PORT3, "Color");

/* This is the user input loop, that gets the user input and
sends the commands to the robot 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 (0 to 1) */

printf("\nArm Direction: Claw Control:\n");
printf("[w] [q] Claw Open\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 == 'q') || (dir == 's') || (dir == 'd')) {
    printf("Enter angle: ");
    scanf("%d", &angle);
  }
  if(dir == 'f'){
    printf("Enter speed ratio: ");
    scanf("%f", &speedRatio);
  }
  if(dir == 'a'){
    //Arm rotate left.
  }
}

59
robot.moveJoint(JOINT3, (angle / gearratio));
}
else if(dir == 'd') { // Arm rotate right.
    robot.moveJoint(JOINT3, (int)(-angle / gearratio));
}
else if(dir == 'w') { // Arm rotate down
    robot.moveJoint(JOINT2, (int)(angle / gearratio));
}
else if(dir == 's') { // lower arm down
    robot.moveJoint(JOINT2, (int)(-angle / gearratio));
}
else if(dir == 'q') {
    /* open claw */
    robot.setJointSpeedRatio(JOINT1, speedRatio);
    robot.moveJointTime(JOINT1, 1);
}
else if(dir == 'e') {
    /* close claw */
    robot.setJointSpeedRatio(JOINT1, -speedRatio);
    robot.moveJointTime(JOINT1, 1);
}
else if(dir == 'x') { // stop and hold joints
    robot.holdJoints();
}
else if(dir == 'r') { // quit
    printf("\nQuit.\n");
    quit = 1;
}
else if(dir == 'f') {
    robot.setJointSpeedRatios(speedRatio, speedRatio, speedRatio);
    printf("\nSpeed ratio set to %d.\n", speedRatio);
}
else
    printf("Invalid command\n");

robot.delaySeconds(0.2);
robot.getSensorTouch(PORT1, touchValue);
if (touchValue < 0) {
    printf("The Ball was grabbed ");
    robot.getSensorColor(PORT3, colorValue);
    if (colorValue == 5) {
        printf("and the color is red\n");
    } else if(colorValue == 2){
        printf("and the color is blue\n");
    }
}
}

Program 15: machine_rtc.ch Source Code

How does it work?

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

Similar to the while loop of the vehicle RTC program, the while loop of the 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 will terminate 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 a robot using the if-else if-else command.

Directional commands

The directional movements are controlled using the ‘w’, ‘s’, ‘a’, and ‘d’ keys. In Figure 46, 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 if-else statement will run the codes for the case ‘a’. The program fragment for case ‘a’ is shown below:

```
if(dir == 'a'){
    //Arm rotate left.
    robot.moveJoint(JOINT3, (angle / gearratio));
}
```

In case ‘a’, the program will run the joint JOINT3 at velocity speedRatio, and to an angle divided by the gear ratio that the user has entered. For case ‘a’ the program will rotate the machine arm left to an adjusted angle that the user has entered.

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

```
else if(dir == 'd'){
    //Arm rotate right.
    robot.moveJoint(JOINT3, (int)(-angle / gearratio));
}
```

In case ‘d’, the program will run the joint JOINT3 at velocity -speedRatio, and to an angle divided by the gear ratio that the user has entered. For case ‘d’ the program will rotate the machine arm right to an adjusted angle that the user has entered.
When the key ‘w’ has been pressed, the `if-else if-else` statement will run the codes for the case ‘w’. The program fragment for case ‘w’ is shown below:

```java
else if (dir == 'w') {
    robot.moveJoint(JOINT2, (int)(angle / gearratio));
}
```

In case ‘w’, the program will run the joint JOINT2 at velocity speedRatio, and to a prescribed angle that the user has entered. For case ‘w’ the program will move the 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 if-else` statement will run the codes for the case ‘s’. The program fragment for case ‘s’ is shown below:

```java
else if (dir == 's') {
    robot.moveJoint(JOINT2, (int)(-angle / gearratio));
}
```

In case ‘s’, the program will run the joint JOINT2 at velocity -speedRatio, and to a prescribed angle that the user has entered. For case ‘w’ the program will move the 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 do 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 if-else` statement will run the code for the case ‘q’. The program fragment for case ‘q’ is shown below:

```java
else if (dir == 'q') {
    /* open claw */
    robot.setJointSpeedRatio(JOINT1, speedRatio);
    robot.moveJointTime(JOINT1, 1);
}
```

In case ‘q’, the program will run the joint JOINT1 at velocity speedRatio for 1 seconds. Next, the program will hold the position of the motor at that spot, thus keeping the machine claw open. For case ‘q’ the program will open the machine claw.

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

```java
else if (dir == 'e') {
    /* close claw */
    robot.setJointSpeedRatio(JOINT1, -speedRatio);
    robot.moveJointTime(JOINT1, 1);
}
```

In case ‘e’, the program will run the joint JOINT1 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. for case ‘e’ the program will close the machine claw.

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

```java
else if (dir == 'x') {
    /* do nothing */
}
```
else if (dir == 'x'){
    // stop and hold joints
    robot.holdJoints();
}

In case ‘x’, the program will stop and hold all motors.

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

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

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.

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 (dir == 'f'){
    robot.setJointSpeedRatios(speedRatio, speedRatio, speedRatio);
    printf("\nSpeed ratio set to %d.", 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:

robot.getSensorTouch(PORT1, touchValue);
if (touchValue < 0) {
    printf(" The Ball was grabbed ");
    robot.getSensorColor(PORT3, colorValue);
    if (colorValue == 5) {
        printf(" and the color is red\n");
    } else if (colorValue == 2) {
        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 PORT1 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 0, the touch sensor is not triggered, so there is the robot detected that there is no object in its claw. If the sensor value for the touch sensor is 1, the touch sensor is triggered meaning 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 PORT3, which is the light sensor. Next, the program will determine the color of the ball using the light sensor value. The light sensor returns a number less than 50 means the ball grabbed is
red and program will print out "and the color is red". On the other hand, if the number returned is greater than 50, the claw grabs a blue ball 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 machine will be introduced. This autonomous program uses the machine arm to scan its surrounding. It performs this task by rotating its arm by an angle step and collect 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 machine automatic control program is presented in Program 16. In the rest of this section, we are going to explain important parts of the automatic control program in detail.

Source Code

```c
#include <conio.h>
#include <stdio.h>
#include <chplot.h>
#include <mindstorms.h>

CPlot plot;
CMindstorms robot;
double speedRatio = 0.30;
int quit = 0, // used to exit for loop
    i; // counter variable

double ultraValue, position;
double gearratio = (8.0/56)*(1.0/24);
int numpoints = 10; // desired number of data points
int anglestep = 2; // angle moved between steps
double angle[numpoints]; // angle retrieved from the tachometer
double distance[numpoints]; // data received from the ultrasonic sensor

/* Initialize arm. (Set sensor types and initialize variables) */
printf("\nInitializing arm for autonomous control...\n");

robot.setSensorUltrasonic(PORT4, "Centimeter");
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\n", 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.\n");

if(getch() == 'q'){
```
/* begin Autonomous loop*/
for (i = 0; i < numpoints; i++){
    /* get sensor data, if success print data, else print error*/
    robot.getSensorUltrasonic(PORT4, ultraValue);
    distance[i] = ultraValue;
    robot.getJointAngle(JOINT1, angle[i]);
    printf("\nSample: %d, distance: %lf, Angle: %lf",
           i, distance[i], angle[i]);

    /* 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)*/
    robot.moveJoint(JOINT1, anglestep / gearratio);
}

/* plot data in CH */
plot.data2DCurve(angle, distance, numpoints);
plot.grid(PLOT_ON);
plot.plotting();

Program 16: machine_auto.ch Source Code

Please use the Figure 47 to connect your Lego Mindstorms robots for the autonomous control program.

Figure 47: Robotic Machine Autonomous Control 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:

```c
double ultraValue, position;
double gearratio = (8.0/56)*(1.0/24);
int numpoints = 10; // desired number of data points
int anglestep = 2; // angle moved between steps
double angle[numpoints]; // angle retrieved from the tachometer
double 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 `angle` determines how much the arm rotates in degrees of angle.

In the next part of the program,

```c
/* print usage information to the user*/
printf("%d Data points will be collected with a" "step size of %.d.", numpoints, anglestep);
printf("Please ensure that the arm can rotate" "%d degrees from its current position.",
(numpoints*anglestep));
printf("Press any key to continue. Press q at any time to quit.\n");
if (getch() == 'q') {
    printf("Quitting program.\n");
    exit(0);
}
```

The program print out how many data point will be collected with the angle step size. Also, it calculates and prints out the full rotation angle of the robot arm and ask the user to ensure that the arm can rotate that amount of angle. 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 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 machine.

```c
/* begin Autonomous loop*/
for(i = 0; i < numpoints; i++){
    /* get sensor data, if success print data, else print error*/
    robot.getSensorUltrasonic(PORT4, ultraValue);
    distance[i] = ultraValue;
    robot.getJointAngle(JOINT1, angle[i]);
    printf("Sample: %d, distance: %lf, Angle: %lf",
        i, distance[i], angle[i]);

    /* check if q was pressed and if so exit program*/
    if (!_kbhit) {
        if (getch() == 'q') {
            printf("Quitting program.\n");
        }
    }
}
```
The first section of the for loop begins at the line after the first comment and ends at the line before the second comment. The first section begins by getting the ultrasonic sensor data and stores 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 error.

The second section of the for loop begins at the line after the second comment and ends at the line before the third comment. In this 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.

The third section of the for loop begins at the line after the third comment and ends at the end of the for loop. In this 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.

```c
/* plot data in CH */
plot.data2DCurve(angle, distance, numpoints);
plot.grid(PLT_ON);
plot.plotting();
```

In this fragment, the program uses the CPlot class commands to do its plotting. The program use the function data2DCurve() to insert the collected data onto the polar plot. Afterward, the program uses the function grid() command 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.

![Plot collected data in Ch](image)

Figure 48: Plot collected data in Ch
6.2 Controlling Lego Mindstorms Robots in Humanoid

Figure 49: Lego Mindstorms Humanoid

The third form of the Lego Mindstorms is a humanoid robot as shown in Figure 49. The humanoid uses two of its motors to perform the walking motion. Also, the humanoid uses its last motor to control the rotation of its head. The 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 humanoid will be discussed. Please configure sensors and motors according to Figure 50.

Figure 50: Sensor/Motor configuration of the Humanoid
6.2.1 Manual Real Time Control Program

The real time control program of the humanoid is similar to the real time control program of the vehicle. The RTC program of the 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 collected by sensors. Figure 51 shows the user interface of the humanoid.

![Humanoid RTC User Interface](image)

Figure 51: Humanoid RTC User Interface

The list below is the list of commands:

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

The humanoid real time control program is described in Program 17. In the rest of this section, we are going to explain important parts of the manual RTC program in detail.

Source Code

```c
/* File name: humanoid_rtc.ch
 * The purpose of this demo is to 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 <mindstorms.h>
int printGUI();
int printSensor(CMindstorms &robot);
CMindstorms robot;
double direction = 1, speed = 25;
int quit = 0; // used by quit case to exit the loop
char key = 'x', // stores the input from the user
    movemode = 'x'; // stores the last movement command

/* Set sensor types */
robot.setSensorTouch(PORT1, "Touch");
robot.setSensorSound(PORT2, "dBA");
robot.setSensorLight(PORT3, "Ambient");
robot.setSensorUltrasonic(PORT4, "Centimeter");

/* print control menu */
printGUI();

/* Control loop. Interprets user command and does action */
while (quit != 1){
    robot.setJointSpeeds(NaN, direction * speed, direction * speed);
    key = _getch();
    if(key == 'w'){ // forward
        direction = 1;
        robot.moveForeverNB();
        movemode = 'w';
    }
    else if(key == 's'){ // backward
        direction = -1;
        robot.moveForeverNB();
        movemode = 's';
    }
    else if(key == 'd'){ // right
        robot.moveJointForeverNB(JOINT1);
    }
    else if(key == 'a'){ // left
        robot.moveJointForeverNB(JOINT1);
    }
    else if(key == 'q'){ // print sensor
        printSensor(robot);
    }
    else if(key == 'x'){ // stop and hold joints
        robot.holdJoints();
        movemode = 'x';
    }
    else if(key == 'r'){ // quit
        printf("\nExiting program.\nPress any key to exit.\n");
        quit = 1;
    }
    else if(key == '1'){ // speedRatio .25
        speed = 25;
        ungetch(movemode);
    }
    else if(key == '2'){ // speedRatio .50
        speed = 50;
        ungetch(movemode);
    }
    else if(key == '3'){ // speedRatio .75
        speed = 75;
    }
}
else if(key == '4') { // speedRatio 1
    speed = 100;
    ungetch(movemode);
}

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

int printGUI(){
    /* 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 \[s\] \[3\] Set Speed to 75"
    printf("\n\[a\]<-head->[d] \[q] Get Sensor Data"
    printf("Please Enter command:");

    return 0;
}

int printSensor(CMindstorms &robot) {
    int touchValue = 0;
    int soundValue = 0;
    int lightValue = 0;
    double ultraValue = 0;

    /* get values from robot */
    robot.getSensorTouch(PORT1, touchValue);
    robot.getSensorSound(PORT2, soundValue);
    robot.getSensorLight(PORT3, lightValue);
    robot.getSensorUltrasonic(PORT4, ultraValue);

    /* display the values */
    if (touchValue < 0) {
        printf("\nThe touch sensor has been pressed.\n");
    } else {
        printf("\nThe touch sensor has not been pressed.\n");
    }

    printf("The distance is %lf\n", ultraValue);
    printf("The light level is %d\n", lightValue);
    printf("The sound level is %ddBA\n\n", soundValue);

    /* print control menu */
    printGUI();

    return 0;
}

Program 17: humanoid_rtc.ch Source Code

How does it work?
The the main difference between the RTC program for the humanoid and the 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 if-else statement
will run the code 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.

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

```java
if(key == 'w'){
    //forward
direction = 1;
    robot.moveForeverNB();
    movemode = 'w';
}
```

In case ‘w’, the program will run joints JOINT2 and JOINT3 at velocity speedRatio. For case ‘w’
the program will control the humanoid to walk forward at a speed ratio.

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

```java
else if(key == 's'){
    //backward
direction = -1;
    robot.moveForeverNB();
    movemode = 's';
}
```

In case ‘s’, the program will stop joints JOINT2 and JOINT3 and turn them to off. For case ‘s’ the
program will stop the humanoid from walking forward.

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

```java
else if(key == 'd'){
    //right
    robot.moveJointForeverNB(JOINT1);
}
```

In case ‘d’, the program will run the joint JOINT1 at a speed ratio of 0.3. For case ‘d’ the program
will rotate the humanoid head right.

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

```java
else if(key == 'a'){
    //left
    robot.moveJointForeverNB(JOINT1);
}
```

In case ‘a’, the program will run JOINT1 at a speed ratio of 0.3. For case ‘a’ the program will rotate
the humanoid head left.

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

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

In case ‘q’, the program will run the function printSensor(), which will retrieve sensor 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 ‘x’ has been pressed, the if-else if-else statement will run the code for the case
‘x’. The program fragment for case ‘x’ is shown below:
else if (key == 'x') {
    // stop and hold joints
    robot.holdJoints();
    movemode = 'x';
}

In case ‘x’, the program will stop and hold all motors.

### Printing sensor data

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

```c
int printSensor(CMindstorms &robot) {
    int touchValue = 0;
    int soundValue = 0;
    int lightValue = 0;
    double ultraValue = 0;

    /* get values from robot */
    robot.getSensorTouch(PORT1, touchValue);
    robot.getSensorSound(PORT2, soundValue);
    robot.getSensorLight(PORT3, lightValue);
    robot.getSensorUltrasonic(PORT4, ultraValue);

    /* display the values */
    if (touchValue < 0) {
        printf("The touch sensor has been pressed.\n");
    } else {
        printf("The touch sensor has not been pressed.\n");
    }

    printf("The distance is %lf.\n", ultraValue);
    printf("The light level is %d.\n", lightValue);
    printf("The sound level is %d\n\n\n\n", soundValue);

    /* print control menu */
    printGUI();

    return 0;
}
```

In the beginning of this function, the program get sensor data from each sensor and store them in the 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.
Play Music with Robots

Have you ever listened to a song and wondered how its made? How did the musician string a bunch of sounds together, while following a beat? Let’s do the this with robots in Ch program.

7.1 Play Melody in Software

There’s a method we can use to play a bunch of songs already built into the Ch. It’s called the playMelody method. `playMelody(note_t song(int), double speedFactor)`.

There’s quite a few to choose from like Twinkle Twinkle Little Star, or Row Your Boat. The song names are abbreviated and implemented as functions; these are the names we will use when calling our playMelody function. The table to the left has all of the songs and their abbreviations.

<table>
<thead>
<tr>
<th>Name of Song</th>
<th>Abbreviation of Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-I-N-G-O</td>
<td>Bingo</td>
</tr>
<tr>
<td>Do Re Mi</td>
<td>DoReMi</td>
</tr>
<tr>
<td>Happy Birthday</td>
<td>HappyBirthday</td>
</tr>
<tr>
<td>The Ice Cream Truck Jingle</td>
<td>IceCream</td>
</tr>
<tr>
<td>Jingle Bells</td>
<td>JingleBells</td>
</tr>
<tr>
<td>Marry Had A Little Lamb</td>
<td>LittleLamb</td>
</tr>
<tr>
<td>Twinkle Twinkle Little Star</td>
<td>LittleStar</td>
</tr>
<tr>
<td>Merry Christmas</td>
<td>MerryChristmas</td>
</tr>
<tr>
<td>Old Mc Donald Had A Farm</td>
<td>OldMcDonald</td>
</tr>
<tr>
<td>A Typical Phone Ring Tone</td>
<td>RingTone</td>
</tr>
<tr>
<td>Row Your Boat</td>
<td>RowYourBoat</td>
</tr>
<tr>
<td>Dance Music</td>
<td>Techno</td>
</tr>
<tr>
<td>The Wheels On The Bus</td>
<td>WheelsOnTheBus</td>
</tr>
</tbody>
</table>

Here’s an example below that will play Jingle Bells.

```c
#include <mindstorms.h>
CMindstorms robot;
robot.playMelody(JingleBells, 1);
```

If we wanted to play a different song all we would have to do is change the name of the song in playMelody to something else. Try out all of the songs, some of the best are Happy Birthday, Row Your Boat, and Merry Christmas.

Of course, you can try creating your songs in Ch. We will introduce how to do it step by step in the following sections.

7.2 Music Basics

Let’s get started with some basic mesic theory. Actually, all the sounds we hear during a song can be represented by notes played for a certain duration.

Notes are the building blocks of music. Each note represents a certain tone that corresponds to a certain frequency. Remember, frequency is measured in Hertz (Hz) which means the number of vibrations per second. Sound waves cause vibrations in the air which our ears detect and we hear as sounds. For example, a note of 100Hz means the air vibrates 100 times per second. A human
ear can typically hear from $20 \text{ Hz to } 20,000 \text{ Hz}$. So, a 100Hz note would sound fairly low pitched while a 10,000 one would be sound much sharper and higher pitched.

Musicians dont bother talking about notes in Hertz, as you can imagine it would get complicated rather quickly. Instead, they use letters to describe notes, and each letter corresponds to a certain frequency. These letters are in the table below.

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>33</td>
</tr>
<tr>
<td>D1</td>
<td>37</td>
</tr>
<tr>
<td>E1</td>
<td>41</td>
</tr>
<tr>
<td>F1</td>
<td>44</td>
</tr>
<tr>
<td>G1</td>
<td>49</td>
</tr>
<tr>
<td>A1</td>
<td>55</td>
</tr>
<tr>
<td>B1</td>
<td>62</td>
</tr>
</tbody>
</table>

You may notice that a lot of frequencies are not listed. This is because, in music, musicians use scales. Each letter can have a number after it indicating its scale, the lower the number, the lower the frequency, and vice versa. The notes on the left are in the first scale (notice the tiny letter next to them). For example, the note C2 has a frequency of 65Hz and thus is higher pitched than the note C1. Scales make the biggest difference in pitch, a note with a lower scale will always be a lower frequency than any note in a higher scale. In addition to scales every C, D, F, G, and A notes can also be “sharp” (denoted by a #) which increases their frequencies, but not as much going from one letter to another letter. If you look at C# (35Hz), it has a higher frequency than C but is still lower pitched than a D note.

Something interesting you may realize is the difference in frequency between notes is not linear. C1 and D1 differ by 4Hz while A1 and B1 differ by 7Hz. This is not a mistake but is actual because the human ear senses change in pitch differently. If a note is very high pitched its hard for humans to notice a change in frequency from one note to another. This is why we have to make the difference in frequency between notes larger when the note is higher pitched. Other senses work this way too. For example, if a light is off and it turns on a little bit the human eye sees this as a big change, but if the light is already really bright, the eye wont be able to detect small changes in its brightness.

Luckily musicians dont worry about frequencies and Hertz when designing a song. Instead, they’ll string a bunch of notes together at different scales and durations to make a tune. Next, we’ll learn how to program our robots to do this.

### 7.3 Defining a Note in Software

Creating a note in our code is simple. Check out the example below.

```c
#include <pitches.h>

note_t note1;
note1.frequency = NOTE_A4;
note1.duration = 4;

note_t note2;
note2 = {NOTE_C3S, 2};
```

In the example, we created two notes in slightly different ways. Remember, a note corresponds to a certain frequency which can be represented by a letter, and we can also play a note for a
certain duration.

In Ch, notes are a structure of type note_t containing frequency, duration. Frequency in Hertz, and duration in length of notes. A duration of 1 would mean the length is one note, while a duration of 4 means the length is $\frac{1}{4}$ of a full note. This relationship means the bigger the number, the smaller amount of time your note will play for. Typically, we say the length of one note is in seconds, so a note with a duration of 4 will only play for $\frac{1}{4}$ the length of a full note, or 0.25 seconds.

If we want to create a note we need to declare a note_t object. In line three of the code, we declare note1, similar to how we would create a robot object or any other variable. Next, we must specify note1’s frequency and duration. We can access these values by typing note1.frequency or note1.duration. In pitches.h there are a bunch of letters for notes we can use already set up. We just have to type NOTE followed by the letter and scale. Here we set note1 to have a frequency of A in the number 4 scale. We could have also used an actual frequency like 1,000Hz. All notes defined in Ch is listed in Appendix D. Lastly, we specify the duration.

Note2 is declared in much the same way. But rather than assigning the frequency and duration individually, we do it all at once. We have to use , and then put the frequency first followed by a comma and then the duration.

Remember a song is just a sequence of notes. We can represent a sequence of notes in software by using an array.

```
note_t song[] = {{ NOTE_C5 , 4} , { NOTE_G4 , 8} , { NOTE_A4 , 4} ,
                 { NOTE_G4 , 4} , {0 , 4} , { NOTE_B4 , 4} , { NOTE_C5 , 4}};
```

The code above is how we would declare an array of notes. Its the same syntax just like declaring an array of ints or chars. And since its an array of notes we have to specify what each note sounds like, just like we did with note2 in the previous example. Also, when we declare an array like this we separate each note by a comma and enclose the whole thing in curly braces.

### 7.4 Playing an Array of Notes

It’s easy to play an array of notes in Ch! There’s already a function that can do this for us. It’s called playNotes. It’s implemented like this `playNotes(note_t song[], double speedFactor, int pin)`. We can see that, in order to use the function we first give it an array of notes to play, followed by how fast to play them (a number less than one means the notes will play faster, and if greater than one they will play slower), lastly we have to specify the pin of our speaker. See the example below,

```c
#include <mindstorms.h>

CMindstorms robot;

note_t song[] = {{ NOTE_C4 , 8} , { NOTE_C4 , 8} , { NOTE_D4 , 4} , { NOTE_C4 , 4} ,
                 { NOTE_F4 , 4} , { NOTE_E4 , 4} , {0 , 4} , { NOTE_C4 , 8} , { NOTE_G4 , 8} , { NOTE_A4 , 4} ,
                 { NOTE_D4 , 4} , { NOTE_G4 , 4} , { NOTE_E4 , 4} , {0 , 4} , { NOTE_C4 , 8} , { NOTE_C4 , 8} ,
                 { NOTE_C5 , 4} , { NOTE_A4 , 4} , { NOTE_F4 , 8} , { NOTE_F4 , 8} , { NOTE_E4 , 4} ,
                 { NOTE_D4 , 4} , {0 , 4} , { NOTE_A4 , 8} , { NOTE_A4 , 8} , { NOTE_A4 , 4} , { NOTE_F4 , 4} ,
                 { NOTE_G4 , 4} , { NOTE_F4 , 4}};

robot.playNotes(song, 1);
```

We don’t need to include pitches.h to use the notes because it already comes with the robot header file. `playNotes()` is part of our robot class, so we have to call it using the robot. Then we declare our song array, and lastly call it with `robot.playNotes(song, 1)` passing in our song array! We will play the song array at normal speed, and our speaker will be plugged into pin 8.

Our song array could be anything! The one above are the notes for Happy Birthday.
7.5 Write and Play Customized Song File

We have introduced how to define a song as an array of note_t and play the song using the method playNotes. However, you may want to play the same song in different Ch program or share with others. It is better to store the song in an independent file. In this section, we will introduce how to create customized song files.

Remember that we use the method playMelody to play a pre-defined song, HappyBirthday. The song is saved as a Ch function file. Now, let’s start to create your Happy Birthday song file.

The first open a new file in Ch and create a function with the prototype note_t functionName(int i). The function can be anything in theory. However, they can not be repeated. Therefore, you can not reuse the function name for those pre-defined songs. In the following example, we will create a new happy birthday function called myHappyBirthday.

```c
note_t myHappyBirthday(int i)
{
    int len = 0;
    note_t note;
    note_t song[] = {{NOTE_C4, 8}, {NOTE_C4, 8}, {NOTE_D4, 4}, {NOTE_C4, 4},
                     {NOTE_F4, 4}, {NOTE_E4, 4}, {0, 4}, {NOTE_C4, 8}, {NOTE_C4, 8},
                     {NOTE_D4, 4}, {NOTE_C4, 4}, {NOTE_G4, 4}, {NOTE_E4, 4}, {0, 4},
                     {NOTE_C4, 8}, {NOTE_C4, 8}, {NOTE_G5, 4}, {NOTE_A4, 4}, {NOTE_F4, 8},
                     {NOTE_F4, 8}, {NOTE_F4, 4}, {NOTE_D4, 4}, {0, 4}, {NOTE_A5, 8},
                     {NOTE_A5, 8}, {NOTE_A4, 4}, {NOTE_F4, 4}, {NOTE_G4, 4}, {NOTE_F4, 4}};

    len = sizeof(song)/sizeof(note_t);
    if(i < len) {
        note.frequency = song[i].frequency;
        note.duration = song[i].duration;
    } else {
        note.frequency = -1;
        note.duration = -1;
    }

    return note;
}
```

In the function, we defined three variables, len, note and texttt song. The note_t array song stores all the notes of the song which is the same as we introduced in the previous section. The variable len is used to store the number of notes in song. The ith note stored in song will be retrieved and assigned to the variable note which will be returned. A if statement is used to check if the argument i is over the length of notes stored in song. If it passed the length, a null note would be assigned to the note which will terminate the song. After you finish writing your song file, don not forget to save it with the name of functionName.chf. In this case, the function file will be saved as myHappyBirthday.chf.

To play the song file, simply call the method playMelody by passing the song function your just created. For example,

```c
#include <mindstorms.h>
CMindstorms robot;

robot.playMelody(myHappyBirthday, 1);
```

One thing needs to be noted that the song file (*.chf) need to be in the same folder of where your main program is. Otherwise, the program is not able to find where the function is.
A  CMindstorms Class API

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

A.1  Data Types Used in CMindstorms Class

The data types defined in the header file mindstorms.h are described in this appendix. These data types are used by the Ch Mindstorms library to represent certain values, such as joint id’s and motor directions.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>robotJointId_t</td>
<td>An enumerated value that indicates ports on a robot for motors.</td>
</tr>
<tr>
<td>robotSensorId_t</td>
<td>An enumerated value that indicates ports on a robot for sensors.</td>
</tr>
<tr>
<td>robotSensorType_t</td>
<td>An enumerated value that indicates the type of a sensor of a Lego Mindstorms robot.</td>
</tr>
<tr>
<td>robotRecordData_t</td>
<td>Recorded time, angle, distance, x or y coordinate.</td>
</tr>
</tbody>
</table>

A.1.1  robotJointId_t

This data type is an enumerated type used to identify ports on the Lego Mindstorms robots for motors. Valid values for this type are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOINT1</td>
<td>Port A on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>JOINT2</td>
<td>Port B on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>JOINT3</td>
<td>Port C on the Lego Mindstorms NXT/EV3.</td>
</tr>
<tr>
<td>JOINT4</td>
<td>Port D on the Lego Mindstorms EV3</td>
</tr>
</tbody>
</table>

A.1.2  robotSensorPort_t

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT1</td>
<td>sensor PORT 1 on the robot.</td>
</tr>
<tr>
<td>PORT2</td>
<td>sensor PORT 2 on the robot.</td>
</tr>
<tr>
<td>PORT3</td>
<td>sensor PORT 3 on the robot.</td>
</tr>
<tr>
<td>PORT4</td>
<td>sensor PORT 4 on the robot.</td>
</tr>
</tbody>
</table>

A.2  Member Functions Available for General Purpose

Member functions available in CMindstorms class are listed in this section.

```c
void robot.connect()
```

connects a robot to be controlled.
void robot.connectWithAddress(string_t addr)
item connects a robot to be controlled using the bluetooth address. The program then does not depend on the configuration set up. For example,

```
robot.connectWithAddress("00:16:53:14:7b:b4");
```

void robot.delaySeconds(int seconds)
item makes a program pause for the number of seconds specified before moving to the next line of code. For example,

```
robot.delaySeconds(3);
```

void robot.enableRecordDataShift()
item enables the shifting of recorded data. By default, the record functions only record data while the robot is in motion (data shifted).

void robot.disableRecordDataShift()
disables the shifting of recorded data. By default, the record functions only record data while the robot is in motion (data shifted).

void robot.getDistance(double distance, double radius)
records the distance that a vehicle has moved using a specified wheel radius. For example,

```
robot.getDistance(distance, 1.75);
```

void robot.getAccelerometerData(double &x, double &y, double &z)
measures the magnitude of gravitational forces in the X, Y and Z directions and stores values in variables x, y, and z.

```
robot.getAccelerometer(x, y, z);
```

void robot.getBatteryVoltage(double &voltage)
measures the battery voltage and stores value in variable voltage.

```
robot.getBatteryVoltage(voltage);
```

void robot.getLEDColor(string_t color)
passes the color of the robots LED to the variable color. For example,

```
string_t color;
robot.getLEDColor(color);
```
void robot.getLEDColorRGB(int &r, int &g, int &b)
passes the color of the robots LED in RGB values to variables r, g, and b. For example,

```c
robot.getLEDColorRGB(r, g, b);
```

void robot.getJointAngle(int id, double &angle)
gets the motor angle of a robot 10 times and stores the average of the 10 motor angles (in degrees) in the variable angle. id is used to specify which motor to monitor and is written as JOINT1, JOINT2, JOINT3, or JOINT4 for the respective motors. For example,

```c
robot.getJointAngle(JOINT1, angle);
```

void robot.getJointAngles(double &angle1, double &angle2, double &angle3)
is the same as getJointAngle(), but it gets the average angle value for all movable motors of a robot. Values for the average angle of each motor will be stored in the respective variables angle1, angle2, angle3, and angle4. For example,

```c
robot.getJointAngles(angle1, angle2, angle3);
```

void robot.getJointAngleInstant(int id, double &angle)
gets a motor angle of a CMindstorms (in degrees) in the variable angle. id is used to specify which motor to monitor and is written as JOINT1, JOINT2, or JOINT3 for the respective motors. For example,

```c
robot.getJointAngleInstant(JOINT1, angle);
```

void robot.getJointAnglesInstant(double &angle1, double &angle2, double &angle3)
is the same as getJointAngleInstant(), but it gets the angle values for all movable motors of a robot. Values for the angle of each motor will be stored in the respective variables angle1, angle2, angle3, angle4. For example,

```c
robot.getJointAnglesInstant(angle1, angle2, angle3);
```

void robot.getJointSpeed(int id, double &speed)
gets the speed (deg/sec) of a robot's motor and stores the value in the variable speed.

```c
robot.getJointSpeed(JOINT3, speed);
```
void robot.getJointSpeeds(double &speed1, double &speed2, double &speed3)
gets the speed of all motors of a robot and stores value in the respective variable for that motor. Please use `getJointSpeed(JOINT4, speed)` to get the joint speed of JOINT4 for EV3. For example,

```c
robot.getJointSpeeds(speed1, speed2, speed3);
```

void robot.getJointSpeedRatio(int id, double &ratio)
gets the speed ratio of a robot’s motor and records value in the variable ratio. For example,

```c
robot.getJointSpeedRatio(JOINT3, speedRatio);
```

void robot.getJointSpeedRatios(double &speedRatio1, double &speedRatio2, double &speedRatio3)
gets the speed ratio of all motors of a robot and stores value in the respective variable for that motor. For example,

```c
robot.getJointSpeedRatios(speedRatio1, speedRatio2, speedRatio3);
```

void getSensorInfrared(int port, int &value);
gets the value of Infrared sensor attached to the specified port on a robot.

```c
robot.getSensorIR(PORT4, value);
```

void getSensorColor(int port, int &value);
gets the value of color sensor attached to the specified port on a robot.

```c
robot.getSensorColor(PORT4, value);
```

void getSensorUltrasonic(int port, double &value);
gets the value of ultrasonic sensor attached to the specified port on a robot.

```c
robot.getSensorUltrasonic(PORT4, value);
```

void getSensorTouch(int port, int &value);
gets the value of touch sensor attached to the specified port on a robot.

```c
robot.getSensorTouch(PORT4, value);
```
void getSensorSound(int port, int &value);
gets the value of sound sensor attached to the specified port on a robot.
robot.getSensorSound(PORT4, value);

void getSensorLight(int port, int &value);
gets the value of light sensor attached to the specified port on a robot.
robot.getSensorLight(PORT4, value);

void getSensorAngle(int port, int &value);
gets the value of angle sensor attached to the specified port on a robot.
robot.getSensorAngle(PORT4, value);

void getSensorForce(int port, int &value);
gets the value of force sensor attached to the specified port on a robot.
robot.getSensorForce(PORT4, value);

void getSensorAccelerometer(int port, double &x, double &y, double &z);
gets the value of accelerometer attached to the specified port on a robot.
robot.getSensorAccelerometer(PORT4, x, y, z);

void getSensorTemperature(int port, double &value);
gets the value from the temperature sensor attached to the specified port on a robot.
robot.getSensorTemperature(PORT4, value);

void getSensorGyroscope(int port, int &value);
gets the value from the gyroscope attached to the specified port on a robot.
robot.getSensorGyroscope(PORT4, value);

void robot.holdJoint(int id)
holds a motor. For example,
robot.holdJoint(JOINT1);
void robot.holdJoints()
holds all motors. For example,

    robot.holdJoints();

void robot.holdJointsAtExit()
holds all motors at exit. For example,

    robot.holdJointsAtExit();

int isConnected()
checks the connection of a robot. The function returns 1 if the robot is connected. Otherwise, 0
will be returned.

int isJointMoving(int joint_id);
checks moving statement of a motor. The function returns 1 if the specified motor is moving. Otherwise, 0 will be returned.

int isMoving();
checks moving statement of the robot. The function returns 1 if the robot is moving. Otherwise, 0
will be returned.

int isNotMoving();
checks moving statement of the robot. The function returns 1 if the robot is not moving. Otherwise, 0 will be returned.

void robot.move(double angle1, double angle2, double angle3)
void robot.move(double angle1, double angle2, double angle3, double angle4)
moves a robot’s motors relative to their current positions by corresponding values of angle1, angle2, angle3. If a fourth argument is given, then the function will move the fourth motor of the robot by the value of the argument. If an argument is given as the symbol ‘NaN’, the function will ignore the corresponding motor. For example,

    robot.move(NaN, 90, -90);
    robot.move(NaN, 30, 90, 20);
```c
void robot.moveForeverNB()

moves joints forever, while allowing the next lines of code to begin before the function moveForeverNB() has finished executing. For example,

```robot.moveForeverNB();```

```c
void robot.moveJoint(int id, double angle)

is the same as the move() function but allows you to specify just one motor to move. For example,

```robot.moveJoint(JOINT3, 90);```

```c
void robot.moveJointNB(int id, double angle)

A non-blocking version of moveJoint().

```c
void robot.moveJointForever(int id)

moves a motor forever. For example,

```robot.moveJointForever(JOINT1);```

```c
void robot.moveJointForeverNB(int id)

A non-blocking version of moveJointForever().

```c
void robot.moveJointTime(int id, int seconds)

moves a motor for specific time. For example,

```robot.moveJointTime(JOINT1, 10);```

```c
void robot.moveJointTimeNB(int id, int seconds)

A non-blocking version of moveJointTime().

```c
void robot.moveJointTo(int id, double angle)

is the same as the moveTo() function but allows you to specify just one joint to move. For example,

```robot.moveJointTo(JOINT3, 90);```

```c
void robot.moveJointToNB(int id, double angle)

A non-blocking version of moveJointTo().

```
```c
void robot.moveJointWait(int id)

pauses the program until the movement of the specified motor id, has stopped its current motion. For example,

```robot.moveJointWait(JOINT1);```

```c
void robot.moveNB(double angle1, double angle2, double angle3)
void robot.moveNB(double angle1, double angle2,
                  double angle3, double angle4)

A non-blocking version of move().

```c
void robot.moveTime(int seconds)

moves joints for a specified time in seconds. For example,

```robot.moveTime(5);```

```c
void robot.moveTimeNB(int seconds)

A non-blocking version of moveTime().

```c
void robot.moveTo(double angle1, double angle2, double angle3)
void robot.moveTo(double angle1, double angle2,
                  double angle3, double angle4)

moves the respective motors of a robot to the absolute position of the specified angles. This is different from the move() function which moves the motor an angle relative to its current position. If a fourth argument is given, then the function will move the fourth motor of the robot by the value of the argument. If an argument is given as the symbol ‘NaN’, the function will ignore the corresponding motor. For example,

```robot.moveTo(NaN, 120, -120);
robot.moveTo(NaN, 120, -120, 90);```

```c
void robot.moveToNB(double angle1, double angle2, double angle3)
void robot.moveToNB(double angle1, double angle2,
                     double angle3, double angle4)

A non-blocking version of moveTo().

```c
void robot.moveWait()

pauses the program until the movement of all the joints of a robot have finished moving.
```
void robot.recordAngleBegin(int id, double* &timedata,
                          double* &angledata, double timeinterval)

begins recording data points of time and motor angle stored in the variables, timedata and angledata. id specifies which motor to record data for and timeinterval is used to tell how often the program should record data points (in seconds). For example,

  robot.recordAngleBegin(JOINT1, timedata, angledata, 0.1);

void robot.recordAngleEnd(int id, int numDataPoints)

ends the recording started by the member function recordAngleBegin() and passes the number of data points collected in the variable numDataPoints. For example,

  robot.recordAngleEnd(JOINT1, numDataPoints);

void robot.recordAnglesBegin(double* &timedata, double* &angledata2
                          double* &angledata3, double timeinterval)

begins recording data points of time and motor angle 2 and 3 stored in the variables, timedata, angledata2 and angledata3. timeinterval is used to tell how often the program should record data points (in seconds). For example,

  robot.recordAnglesBegin(timedata, angledata2, angledata3, 0.1);

void robot.recordAnglesEnd(int numDataPoints)

ends the recording started by the member function recordAnglesBegin() and passes the number of data points collected in the variable numDataPoints. For example,

  robot.recordAnglesEnd(numDataPoints);

void robot.relaxJoint(int id)

relaxes a motor. For example,

  robot.relaxJoint(JOINT1);

void robot.relaxJoints()

relaxes all motors. For example,

  robot.relaxJoints();

void robot.resetToZero()

resets the robot’s joints to their absolute zero positions.
void robot.resetToZeroNB()
A non-blocking version of resetToZero().

void robot.setBuzzerFrequencyOff()
turns off the robot buzzer. For example,

```c
robot.setBuzzerFrequencyOff();
```

void robot.setBuzzerFrequencyOn(int frequency)
turns on the robot buzzer with a specified frequency (in Hz). For example,

```c
robot.setBuzzerFrequencyOn(450);
```

void robot.setLEDColor(const string_t* color)
sets the robot LED to the specified color. For example,

```c
robot.setLEDColor("blue");
```

void robot.setLEDColorRGB(int r, int g, int b)
sets the color of the robot’s LED in RGB values. For example,

```c
robot.setLEDColorRGB(160, 32, 240);
```

void robot.setJointSpeed(int id, double speed)
sets the speed of a robot motor given the value in degrees per second. For example,

```c
robot.setJointSpeed(JOINT1, 90);
```

void robot.setJointSpeeds(double speed1, double speed2, double speed3)
void robot.setJointSpeeds(double speed1, double speed2, double speed3, double speed4)
sets the speed in degrees per second of all motors. If a fourth argument is given, then the function will set speed for the fourth motor of the robot by the value of the argument. If an argument is given as the symbol ‘NaN’, the function will ignore the corresponding motor. For example,

```c
robot.setJointSpeeds(NaN, speed2, speed3);
```

void robot.setJointSpeedRatio(int id, double ratio)
sets the speed ratio of robot’s motor to the value of the variable ratio. For example,

```c
robot.setJointSpeedRatio(JOINT3, speedratio);
```
void robot.setJointSpeedRatios(double ratio1, double ratios, double ratio3)
void robot.setJointSpeedRatios(double ratio1, double ratios, double ratio3, double ratio4)

sets the speed ratio of all motors at a time. For example, If a fourth argument is given, then the
function will set speed ratio for the fourth motor of the robot by the value of the argument. If an
argument is given as the symbol ‘NaN’, the function will ignore the corresponding motor.

robot.setJointSpeedRatios(ratio1, ratio2, ratio3);

void robot.setJointPower(int id, int power)
sets the motor power of a specified motor. The specified power can be an integer from -100 to
+100. For example,

robot.setJointPower(JOINT1, 50);

void setSensorInfrared(int port);
sets the Infrared sensor attached at the specified port on a robot.

robot.setSensorInfrared(PORT4);

void setSensorColor(int port, string_t mode);
sets the color sensor attached at the specified port on a robot with a specified mode. The mode for
color sensor can be “Color”, “Ambient” or “Reflect”. For example,

robot.setSensorColor(PORT4, "Color" );
// or
robot.setSensorColor(PORT4, "Ambient" );
// or
robot.setSensorColor(PORT4, "Reflect" );

void setSensorUltrasonic(int port, string_t mode);
sets the ultrasonic sensor attached at the specified port on a robot with a specified mode. The
mode for ultrasonic sensor can be “Centimeter” or “Inch”. For example,

robot.setSensorUltrasonic(PORT4, "Centimeter" );
// or
robot.setSensorUltrasonic(PORT4, "Inch" );

void setSensorTouch(int port, string_t mode);
sets the touch sensor attached at the specified port on a robot with a specified mode. The
mode for touch sensor can be “Touch” or “Bumps”. For example,

robot.setSensorTouch(PORT4, "Touch" );
// or
robot.setSensorTouch(PORT4, "Bumps" );
```cpp
void setSensorSound(int port, string_t mode);
sets the sound sensor attached at the specified port on a robot with a specified mode. The mode for sound sensor can be “dB” or “dBA”. For example,
```n
```cpp
robot.setSensorSound(PORT4, "dB");
// or
robot.setSensorSound(PORT4, "dBA");
```

```cpp
void setSensorLight(int port, string_t mode);
sets the light sensor attached at the specified port on a robot with a specified mode. The mode for light sensor can be “Ambient” or “Reflect”. For example,
```n
```cpp
robot.setSensorLight(PORT4, "Ambient");
// or
robot.setSensorLight(PORT4, "Reflect");
```

```cpp
void setSensorAngle(int port, string_t mode);
sets the angle sensor attached at the specified port on a robot with a specified mode. The mode for angle sensor can be “Radian” or “Degree”. For example,
```n
```cpp
robot.setSensorAngle(PORT4, "Radian");
// or
robot.setSensorAngle(PORT4, "Degree");
```

```cpp
void setSensorForce(int port);
sets the force sensor attached at the specified port on a robot. For example,
```n
```cpp
robot.setSensorForce(PORT4);
```

```cpp
void setSensorAccelerometer(int port);
sets the accelerometer attached at the specified port on a robot. For example,
```n
```cpp
robot.setSensorAccelerometer(PORT4);
```

```cpp
void setSensorTemperature(int port, string_t mode);
sets the temperature sensor attached at the specified port on a robot. The mode for the temperature sensor can be “Celsius” or “Fahrenheit”. For example,
```n
```cpp
robot.setSensorTemperature(PORT4, "Celsius");
// or
robot.setSensorTemperature(PORT4, "Fahrenheit");
```
void setSensorGyroscope(int port, string_t mode);
sets the gyroscope attached at the specified port on a robot. The mode for the gyroscope can be “Angle” or “Rate”. In the “Angle” mode, it measures the rotated angle in degree related to the zero position in range of [-90, 90]. In the “Rate” mode, it measures the rate of rotation in degree per second. For example,

```c
robot.setSensorTemperature(PORT4, "Celsius");
// or
robot.setSensorTemperature(PORT4, "Fahrenheit");
```

void robot.systemTime(double &time)
records the time in seconds since the system last started (Windows) or since 00:00:00 January 1, 1970 (Mac OS X and Linux). For example,

```c
robot.systemTime(time1);
/* Code to be timed */
robot.systemTime(time2);
elapsedtime = time2 - time1;
```

void robot.waitUntil(int port, string_t sign, double value)
blocks a program once the function is called until the sensor, attached at given port, satisfies the given condition. for example,

```c
robot.waitUntil(PORT1, "==", 1);
```

void robot.waitUntilAnd(int port, string_t sign1, double value1, string_t sign2, double value2)
blocks a program once the function is called until the sensor, attached at given port, greater than value1 and less than value2. for example,

```c
robot.waitUntilAnd(PORT4, ">", 10, "<", 20);
```

void robot.waitUntilOr(int port, string_t sign1, double value1, string_t sign2, double value2)
blocks a program once the function is called until the sensor, attached at given port, greater than value1 or less than value2. for example,

```c
robot.waitUntilOr(PORT4, ">", 20, "<", 10);
```

void robot.repeatUntil(void (*function)(), int port, string_t sign, double value)
repeat the given function until the sensor, attached at given port, satisfies the given condition. for example,

```c
void function(){
    robot.driveAngle(30);
}
robot.repeatUntil(function, PORT1, "==", 1);
```
void robot.repeatUntilAnd(function, int port, string_t sign1, double value1, string_t sign2, double value2)

repeat the given function until the sensor, attached at given port, greater than value1 and less than value2. For example,

```c
void function(){
    robot.driveAngle(30);
}
robot.repeatUntilAnd(function, PORT4, ">", 10, "<<", 20);
```

void robot.repeatUntilOr(function, int port, string_t sign1, double value1, string_t sign2, double value2)

repeat the given function until the sensor, attached at given port, greater than value1 or less than value2. For example,

```c
void function(){
    robot.driveAngle(30);
}
robot.repeatUntilOr(function, PORT4, ">", 20, "<<", 10);
```

### B Member Functions Available only for Vehicle Configuration

Member functions available only for vehicle configuration are listed in this section.

**void robot.driveAngle(double angle)**

moves both wheels of a vehicle by a specified angle. For example,

```c
robot.driveAngle(360);
// or
robot.driveAngle(-360);
```

**robot.driveAngleNB(angle)**

A non-blocking version of driveAngle().

**void robot.driveDistance(double distance, double radius)**

moves a two wheel robot a specified distance given its wheel radius. For example,

```c
robot.driveDistance(10, 1.75);
```

**void robot.driveDistanceNB(double distance, double radius)**

A non-blocking version of driveDistance().
void robot.driveTime(int seconds)
moves all motors for specific time. For example,

```c
robot.driveTime(10);
```

void robot.driveTimeNB(int seconds)
A non-blocking version of driveTime().

void robot.driveForever()
moves all motors forever. For example,

```c
robot.driveForever();
```

void robot.driveForeverNB()
A non-blocking version of driveForever().

void robot.recordDistanceBegin(double* &time, double* &distance,
                              double radius, double timeInterval)
begins recording the time and distance of a robot motor in variables time and distance (of type robotRecordData_t) given the specified wheel radius. timeInterval is the time interval between measurements.

void robot.recordDistanceEnd(int &numDataPoints)
ends the recordDistanceBegin() function and passes the number of data points collected in the variable numDataPoints. For example,

```c
robot.recordDistanceBegin(timedata, distances, 1.75, 0.1);
robot.driveTime(4);
robot.recordDistanceEnd(numDataPoints);
```

void robot.recordDistanceOffset(double offset)
creates an offset so that when the recordDistanceBegin() function is called, each distance value will add an offset value. For example,

```c
robot.recordDistanceOffset(2);
```

void robot.turnLeft(double angle, double radius, double trackwidth)
turns a two wheel robot left by a specified angle, given the wheel radius and trackwidth (distance between the two wheels). For example,

```c
robot.turnLeft(90, 1.75, 3.69);
```
\textbf{void robot.turnLeftNB(double angle)}

A non-blocking version of turnLeft().

\textbf{void robot.turnRight(double angle, double radius, double trackwidth)}

turns a two wheel robot left or right by a specified angle, given the wheel radius and trackwidth. For example,

\begin{verbatim}
robot.turnRight(90, 1.75, 3.69);
\end{verbatim}

\textbf{void robot.turnRightNB(double angle)}

A non-blocking version of turnRight().

\section{Member Functions Available in CMindstormsGroup}

\textbf{void group.addRobot(CMindstorms robot)}

adds a robot using the robot’s declared name to the class CMindstormsGroup. For example,

\begin{verbatim}
group.addRobot(robot1);
\end{verbatim}

\textbf{void group.addRobots(CMindstorms robot[])}

adds an array of robots to a group using the robot array’s declared name. For example,

\begin{verbatim}
CMindstorms robot[4];
group.addRobots(robot);
\end{verbatim}

\begin{table}[h]
\centering
\caption{Member Functions Available in Class CMindstormsGroup}
\begin{tabular}{|l|l|}
\hline
Member Function Name & Description \\
\hline
addRobot() & Add a robot as a member of the robot group. \\
addRobots() & Add robots to the robot group. \\
driveAngle() & Drive each robot in the group a certain angle. \\
driveAngleNB() & Drive each robot in the group a certain angle. \\
driveDistance() & Drive each robot in the group a certain distance. \\
driveDistanceNB() & Identical to \texttt{driveDistance()} but non-blocking. \\
driveForeverNB() & Drive each robot forever until stopped. \\
driveTime() & Drive each robot for a specified time. \\
driveTimeNB() & Identical to \texttt{driveTime()} but non-blocking. \\
holdJoint() & Hold a joint for all robots in the group. \\
holdJoints() & Hold all the joints for all robots in the group. \\
holdJointsAtExit() & Hold all the joints at exit for all robots in the group. \\
\hline
\end{tabular}
\end{table}

93
**isConnected()** Check if a robot is connected to a program.

**isMoving()** Check if any joint of a robot is moving.

**move()** Move two joints of each robot by specified angles.

**moveForeverNB()** Move joints forever till stopped.

**moveNB()** Identical to **move()** but non-blocking.

**moveJoint()** Move a joint from its current position by an angle.

**moveJointNB()** Identical to **moveJoint()** but non-blocking.

**moveJointForeverNB()** Move a joint forever. A joint will move until stopped.

**moveJointTime()** Move a joint for a specified time.

**moveJointTimeNB()** Identical to **moveJointTime()** but non-blocking.

**moveJointTo()** Set the desired joint position for a joint.

**moveJointToNB()** Identical to **moveJointTo()** but non-blocking.

**moveJointWait()** Wait until the specified motor has stopped moving.

**moveTime()** Move joints for a specified time.

**moveTimeNB()** Identical to **moveJointTime()** but non-blocking.

**moveTo()** Move two joints of each robot to specified absolute angles.

**moveToNB()** Identical to **moveTo()** but non-blocking.

**moveToZero()** Instructs all motors to go to their zero positions.

**moveToZeroNB()** Identical to **moveToZero()** but non-blocking.

**moveWait()** Wait until all motors have stopped moving.

**relaxJoint()** Relax a joint of the robot.

**relaxJoints()** Relax all the joints of the robot.

**setJointSpeedRatio()** Set a joint speed to a fraction of its maximum speed.

**setJointSpeedRatios()** Set joint speeds to a fraction of their maximum speeds.

---

### D  Notes Defined in Ch

Table 8 lists all macros defined in Ch for notes. One thing need to be noticed that not all frequencies can be played due to the limitation of a robot. For example, Lego NXTs can only accept frequencies from 200 Hz to 14000 Hz while EV3 can only play notes from C4 to C7 (262 Hz to 2093 Hz). Therefore, check the frequency range of the your robot before composing a melody. Otherwise, out-of-range notes played by your robot will make no sense to you.
<table>
<thead>
<tr>
<th>Ch Macro</th>
<th>Letter</th>
<th>Frequency (Hz)</th>
<th>Ch Macro</th>
<th>Letter</th>
<th>Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTE_B0</td>
<td>B0</td>
<td>31</td>
<td>NOTE_GS4</td>
<td>GS4</td>
<td>415</td>
</tr>
<tr>
<td>NOTE_C1</td>
<td>C1</td>
<td>33</td>
<td>NOTE_A4</td>
<td>A4</td>
<td>440</td>
</tr>
<tr>
<td>NOTE_CS1</td>
<td>CS1</td>
<td>35</td>
<td>NOTE_AS4</td>
<td>AS4</td>
<td>466</td>
</tr>
<tr>
<td>NOTE_D1</td>
<td>D1</td>
<td>37</td>
<td>NOTE_B4</td>
<td>B4</td>
<td>494</td>
</tr>
<tr>
<td>NOTE_DS1</td>
<td>DS1</td>
<td>39</td>
<td>NOTE_C5</td>
<td>C5</td>
<td>523</td>
</tr>
<tr>
<td>NOTE_E1</td>
<td>E1</td>
<td>41</td>
<td>NOTE_CS5</td>
<td>CS5</td>
<td>554</td>
</tr>
<tr>
<td>NOTE_F1</td>
<td>F1</td>
<td>44</td>
<td>NOTE_D5</td>
<td>D5</td>
<td>587</td>
</tr>
<tr>
<td>NOTE_FS1</td>
<td>FS1</td>
<td>46</td>
<td>NOTE_DS5</td>
<td>DS5</td>
<td>622</td>
</tr>
<tr>
<td>NOTE_G1</td>
<td>G1</td>
<td>49</td>
<td>NOTE_E5</td>
<td>E5</td>
<td>659</td>
</tr>
<tr>
<td>NOTE_GS1</td>
<td>GS1</td>
<td>52</td>
<td>NOTE_F5</td>
<td>F5</td>
<td>698</td>
</tr>
<tr>
<td>NOTE_A1</td>
<td>A1</td>
<td>55</td>
<td>NOTE_FS5</td>
<td>FS5</td>
<td>740</td>
</tr>
<tr>
<td>NOTE_AS1</td>
<td>AS1</td>
<td>58</td>
<td>NOTE_G5</td>
<td>G5</td>
<td>784</td>
</tr>
<tr>
<td>NOTE_B1</td>
<td>B1</td>
<td>62</td>
<td>NOTE_CS5</td>
<td>CS5</td>
<td>831</td>
</tr>
<tr>
<td>NOTE_C2</td>
<td>C2</td>
<td>65</td>
<td>NOTE_A5</td>
<td>A5</td>
<td>880</td>
</tr>
<tr>
<td>NOTE_CS2</td>
<td>CS2</td>
<td>69</td>
<td>NOTE_AS5</td>
<td>AS5</td>
<td>932</td>
</tr>
<tr>
<td>NOTE_D2</td>
<td>D2</td>
<td>73</td>
<td>NOTE_B5</td>
<td>B5</td>
<td>988</td>
</tr>
<tr>
<td>NOTE_DS2</td>
<td>DS2</td>
<td>78</td>
<td>NOTE_C6</td>
<td>C6</td>
<td>1047</td>
</tr>
<tr>
<td>NOTE_E2</td>
<td>E2</td>
<td>82</td>
<td>NOTE_CS6</td>
<td>CS6</td>
<td>1109</td>
</tr>
<tr>
<td>NOTE_F2</td>
<td>F2</td>
<td>87</td>
<td>NOTE_D6</td>
<td>D6</td>
<td>1175</td>
</tr>
<tr>
<td>NOTE_FS2</td>
<td>FS2</td>
<td>93</td>
<td>NOTE_DS6</td>
<td>DS6</td>
<td>1245</td>
</tr>
<tr>
<td>NOTE_G2</td>
<td>G2</td>
<td>98</td>
<td>NOTE_E6</td>
<td>E6</td>
<td>1319</td>
</tr>
<tr>
<td>NOTE_GS2</td>
<td>GS2</td>
<td>104</td>
<td>NOTE_F6</td>
<td>F6</td>
<td>1397</td>
</tr>
<tr>
<td>NOTE_A2</td>
<td>A2</td>
<td>110</td>
<td>NOTE_FS6</td>
<td>FS6</td>
<td>1480</td>
</tr>
<tr>
<td>NOTE_AS2</td>
<td>AS2</td>
<td>117</td>
<td>NOTE_G6</td>
<td>G6</td>
<td>1568</td>
</tr>
<tr>
<td>NOTE_B2</td>
<td>B2</td>
<td>123</td>
<td>NOTE_CS6</td>
<td>CS6</td>
<td>1661</td>
</tr>
<tr>
<td>NOTE_C3</td>
<td>C3</td>
<td>131</td>
<td>NOTE_A6</td>
<td>A6</td>
<td>1760</td>
</tr>
<tr>
<td>NOTE_CS3</td>
<td>CS3</td>
<td>139</td>
<td>NOTE_AS6</td>
<td>AS6</td>
<td>1865</td>
</tr>
<tr>
<td>NOTE_D3</td>
<td>D3</td>
<td>147</td>
<td>NOTE_B6</td>
<td>B6</td>
<td>1976</td>
</tr>
<tr>
<td>NOTE_DS3</td>
<td>DS3</td>
<td>156</td>
<td>NOTE_C7</td>
<td>C7</td>
<td>2093</td>
</tr>
<tr>
<td>NOTE_E3</td>
<td>E3</td>
<td>165</td>
<td>NOTE_CS7</td>
<td>CS7</td>
<td>2217</td>
</tr>
<tr>
<td>NOTE_F3</td>
<td>F3</td>
<td>175</td>
<td>NOTE_D7</td>
<td>D7</td>
<td>2349</td>
</tr>
<tr>
<td>NOTE_FS3</td>
<td>FS3</td>
<td>185</td>
<td>NOTE_DS7</td>
<td>DS7</td>
<td>2489</td>
</tr>
<tr>
<td>NOTE_G3</td>
<td>G3</td>
<td>196</td>
<td>NOTE_E7</td>
<td>E7</td>
<td>2637</td>
</tr>
<tr>
<td>NOTE_GS3</td>
<td>GS3</td>
<td>208</td>
<td>NOTE_F7</td>
<td>F7</td>
<td>2794</td>
</tr>
<tr>
<td>NOTE_A3</td>
<td>A3</td>
<td>220</td>
<td>NOTE_FS7</td>
<td>FS7</td>
<td>2960</td>
</tr>
<tr>
<td>NOTE_AS3</td>
<td>AS3</td>
<td>233</td>
<td>NOTE_G7</td>
<td>G7</td>
<td>3136</td>
</tr>
<tr>
<td>NOTE_B3</td>
<td>B3</td>
<td>247</td>
<td>NOTE_GS7</td>
<td>GS7</td>
<td>3322</td>
</tr>
<tr>
<td>NOTE_C4</td>
<td>C4</td>
<td>262</td>
<td>NOTE_A7</td>
<td>A7</td>
<td>3520</td>
</tr>
<tr>
<td>NOTE_CS4</td>
<td>CS4</td>
<td>277</td>
<td>NOTE_AS7</td>
<td>AS7</td>
<td>3729</td>
</tr>
<tr>
<td>NOTE_D4</td>
<td>D4</td>
<td>294</td>
<td>NOTE_B7</td>
<td>B7</td>
<td>3951</td>
</tr>
<tr>
<td>NOTE_DS4</td>
<td>DS4</td>
<td>311</td>
<td>NOTE_C8</td>
<td>C8</td>
<td>4186</td>
</tr>
<tr>
<td>NOTE_E4</td>
<td>E4</td>
<td>330</td>
<td>NOTE_CS8</td>
<td>CS8</td>
<td>4435</td>
</tr>
<tr>
<td>NOTE_F4</td>
<td>F4</td>
<td>349</td>
<td>NOTE_D8</td>
<td>D8</td>
<td>4699</td>
</tr>
<tr>
<td>NOTE_FS4</td>
<td>FS4</td>
<td>370</td>
<td>NOTE_DS8</td>
<td>DS8</td>
<td>4978</td>
</tr>
</tbody>
</table>

Table 8: Notes Defined in Ch
Index

CMindstorms

- `connect()`, 78
- `connectWithAddress()`, 79
- `delaySeconds()`, 79
- `disableRecordDataShift()`, 79
- `driveAngle()`, 91
- `driveAngleNB()`, 91
- `driveDistance()`, 91
- `driveDistanceNB()`, 91
- `driveForever()`, 92
- `driveForeverNB()`, 92
- `driveTime()`, 92
- `driveTimeNB()`, 92
- `enableRecordDataShift()`, 79
- `getAccelerometerData()`, 79
- `getBatteryVoltage()`, 79
- `getDistance()`, 79
- `getJointAngle()`, 80
- `getJointAngleInstant()`, 80
- `getJointAngles()`, 80
- `getJointAnglesInstant()`, 80
- `getJointSpeed()`, 80
- `getJointSpeedRatio()`, 81
- `getJointSpeedRatios()`, 81
- `getJointSpeeds()`, 81
- `getLEDColor()`, 79
- `getLEDColorRGB()`, 80
- `getSensorAccelerometer()`, 82
- `getSensorAngle()`, 82
- `getSensorColor()`, 81
- `getSensorForce()`, 82
- `getSensorGyroscope()`, 82
- `getSensorInfrared()`, 81
- `getSensorLight()`, 82
- `getSensorSound()`, 82
- `getSensorTemperature()`, 82
- `getSensorTouch()`, 81
- `getSensorUltrasonic()`, 81
- `holdJoint()`, 82
- `holdJoints()`, 83
- `holdJointsAtExit()`, 83
- `isConnected()`, 83
- `isJointMoving()`, 83
- `isMoving()`, 83
- `isNotMoving()`, 83
- `move()`, 83
- `moveForeverNB()`, 84
- `moveJoint()`, 84
- `moveJointForever()`, 84
- `moveJointForeverNB()`, 84
- `moveJointNB()`, 84
- `moveJointTime()`, 84
- `moveJointTimeNB()`, 84
- `moveJointTo()`, 84
- `moveJointToNB()`, 84
- `moveJointWait()`, 84
- `moveNB()`, 85
- `moveTime()`, 85
- `moveTimeNB()`, 85
- `moveTo()`, 85
- `moveToNB()`, 85
- `moveWait()`, 85
- `repeatUntil()`, 90
- `repeatUntilAnd()`, 91
- `repeatUntilOr()`, 91
- `resetToZero()`, 86
- `resetToZeroNB()`, 87
- `setBuzzerFrequencyOff()`, 87
- `setBuzzerFrequencyOn()`, 87
- `setJointPower()`, 88
- `setJointSpeed()`, 87
- `setJointSpeedRatio()`, 87
- `setJointSpeedRatios()`, 88
- `setJointSpeeds()`, 87
- `setLEDColor()`, 87
- `setLEDColorRGB()`, 87
- `setSensorAccelerometer()`, 89
- `setSensorAngle()`, 89
- `setSensorColor()`, 88
- `setSensorForce()`, 89
- `setSensorGyroscope()`, 90
- `setSensorInfrared()`, 88
- `setSensorLight()`, 89
- `setSensorSound()`, 89
- `setSensorTemperature()`, 89
- `setSensorTouch()`, 88
- `setSensorUltrasonic()`, 88
- `systemTime()`, 90
- `turnLeft()`, 92
- `turnLeftNB()`, 93
- `turnRight()`, 93
- `turnRightNB()`, 93
- `waitUntil()`, 90
waitUntilAnd(), 90
waitUntilOr(), 90
CMindstormsGroup
addRobot(), 93
addRobots(), 93

robotJointId_t
  JOINT1, 78
  JOINT2, 78
  JOINT3, 78
  JOINT4, 78

robotSensorId_t
  PORT1, 78
  PORT2, 78
  PORT3, 78
  PORT4, 78