Chapter 9: Control Lego Mindstorms NXT Robots

9 Ch Mindstorms NXT Control Package

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

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

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

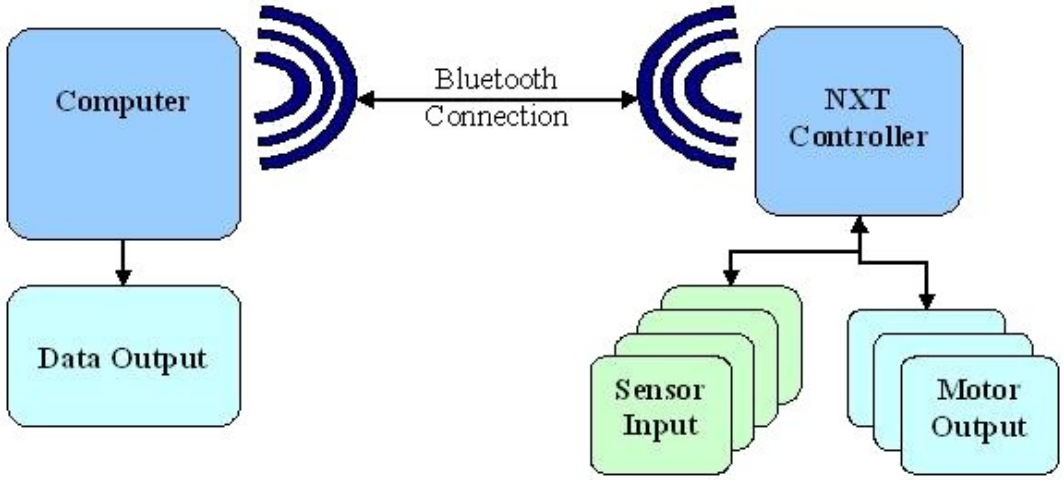

Figure 1: Communication Diagram of NXT

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

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

9.1 Basics of a Ch Mindstorms NXT program

To successfully control the Mindstorm NXT using Ch, it is important to practice good coding habits. The format of the Ch Mindstorm code is very similar to how a normal C code would be written, with the inclusion of some Ch specific functions and header files that are used to connect and control the NXT. The basic structure of a Ch NXT robot program is shown in the flow diagram in Figure **??** below.

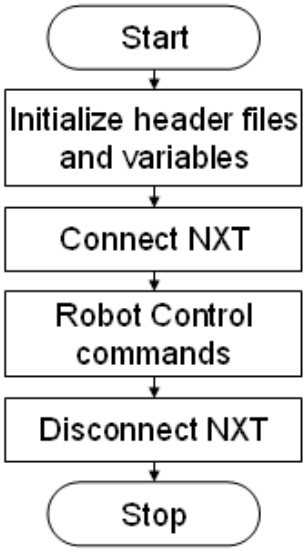

Figure 2: Flow Diagram of a basic NXT program

9.2 Controlling a NXT Vehicle

Figure 3: NXT Vehicle

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

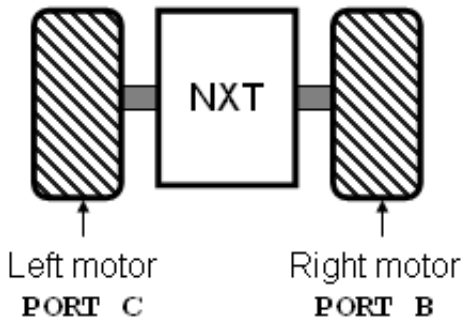

Figure 4: Motor configuration of the NXT Vehicle

9.2.1 Initialization

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

```
#include <conio.h>
#include <ch_nxt.h>
#include <unistd.h>
```

The `conio.h` is an important header file that include the bluetooth connection functions that is required to establish connection between your PC and your NXT. This header also include the _kbhit() function, which is used to detect for hit "anykey" on keyboard if you want to pause your program. The `ch_nxt.h` is the essential header file for Ch NXT function and variables. The `unistd.h` header is needed when using the `sleep()` function, which is used when delays must be placed into a program.

9.2.2 Connect the NXT and Checking Connection Status

The NXT status, sensor/encoder data, and input/output protocols are stored in a structure called `nxt_remote`. This structure must be created in every NXT program in order to connect. Therefore, in beginning of your code, you must define a `nxt_remote` structure and use the `nxt_connect()` function to connect to the NXT. The `nxt_connect()` function will return a 0 if no connection is established, so you will have to terminate your program if no connection is established. An example of how to create the `struct` and how to retrieve data are shown below:

```
struct nxt_remote nxtr;

//Check status of NXT connection
if (!nxt_connect(&nxtr)){
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(0);
}
```

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

9.2.3 Moving the Robot Forward

After creating a connection with the NXT, you can begin your program to control the NXT. You can move the NXT vehicle forward by using the `nxt_forward()` and selecting a speed. However the speed of the motors are limited, and can only be set from 1 to 100. After using

the `nxt_forward()` function, you might want to allow some time for the motor to run by pausing the program for a few seconds. To stop the program from running for a few seconds, you can use the `sleep()` function (each unit is 1 second, i.e. `sleep(1)` = 1 second pause). Figure **??** shows the NXT vehicle moving forward by actuating both wheels forward with the same speed, which is how the function `nxt_forward()` works.

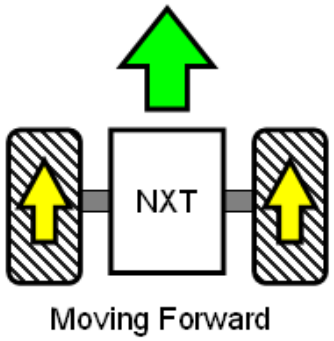

Figure 5: Top-down view of NXT vehicle with two wheels

The following shows how the lego vehicle robot is moved forward in the `forward.ch` program:

```
//Define variables
int speed = 25;
//Commands:
//Move foward
nxt_forward(speed);
//Pause program for a while
sleep(5);
```

9.2.4 Ending your program

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

```
//Stop the motors
nxt_stop();

//Disconnect NXT
nxt_disconnect();
```

9.2.5 forward.ch

Now that the components of the program have been covered, `forward.ch` can now be run. The code can be written using C or Ch syntax. For simplicity, the following code is presented as a Ch program.

```
/*************************************************************
forward.ch
By Michael Schirle

The purpose of this demo is to introduce the CH Mindstorms
Control Package syntax to new users by moving the robot
forward.
*************************************************************/
#include <conio.h>
#include <ch_nxt.h>
#include <unistd.h>

//Define speed variable, can be set [-100 to 100]
int speed = 25; //the speed of the motors.

//Connect to NXT
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)) {
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(0);
}

//Turn the motors on
nxt_forward(speed);

//Add time delay before next command
sleep(5);

//Turn the motors off
nxt_stop();

//Disconnect NXT
nxt_disconnect();
```

Program 1: Ch program to move the NXT vehicle forward.

9.2.6 How to make your NXT move backward

For a two wheeled NXT vehicle, you can make the vehicle move backward by using `nxt_backward()`. The function works like `nxt_forward()`, moving the robot backwards by entering a speed between 0 to 100. The function works by actuationg both wheels backward at the same speed, as shown in Figure **??**. Using our new function, we can add the following code fragment to our first program to make the robot move backwards. The code fragment is shown below:

```
//Move backward
nxt_backward(speed);
sleep(5);
```

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

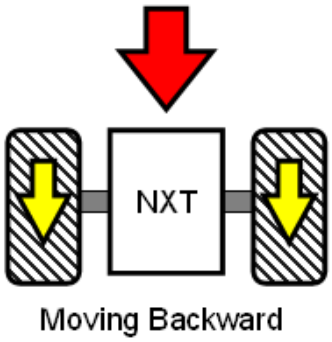

Figure 6: NXT vehicle moving backwards

9.2.7 How to make your NXT move foward and left/right

To make a two wheeled NXT vehicle to turn left or right while moving, one wheel must be moving faster than the other one. For example, to move the NXT forward and turn left, both wheels will have to turn at a positive direction, and the right wheel must run faster than the left wheel. Figure **??** shows the NXT vehicle moving forward and left or right by actuating both wheels forward with different speed.

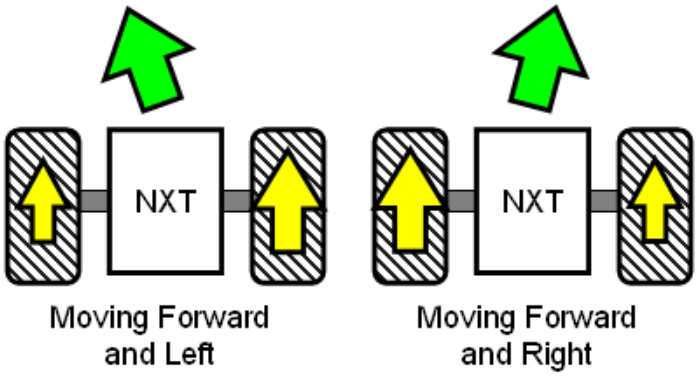

Figure 7: NXT vehicle moving forward and left/right

To make the NXT move forward and left/right, we can use the functions `nxt_turnleft()` and `nxt_turnright()`. Like the previous NXT functions, a speed must be specified when using the function. An example is shown below:

```
//Move Left
nxt_turnleft(speed);
sleep(5);

//or

//Move Right
nxt_turnright(speed);
sleep(5);
```

The modified programs are called: `turnleft.ch` and `turnright.ch`.

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

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

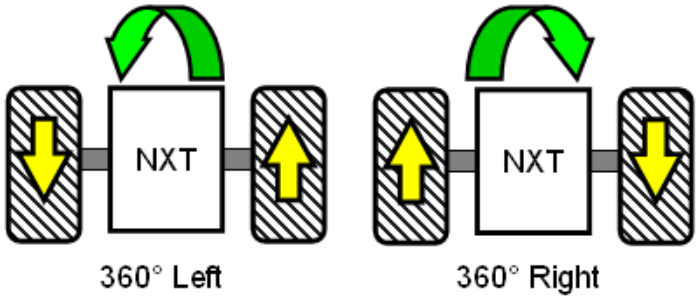

Figure 8: NXT vehicle turning 360 degrees

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

```
//rotate left
nxt_rotateleft(speed);
sleep(5);

// or

//rotate right
nxt_rotateright(speed);
sleep(5);
```

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

9.3 Advance Mindstorm Motor Control

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

9.4 Motor Control Functions

The function that is used to control the NXT motors is `nxt_motor()`. To use the function, you need the motor port and the motor speed. The speed of the motors are limited, and can only ranged from -100 to +100. Like the previous functions, you will need to use the `sleep()` function to leave the motors on for the desired amount of the time. Due to the setup of the NXT vehicle, forward motion can be achieved by turning the motors on to the same speed. For example:

```
//Move foward
nxt_motor(PORT_B, speed, RUN_IDLE);
nxt_motor(PORT_C, speed, RUN_IDLE);
//Pause program for a while
sleep(5);
```

This will move the NXT vehicle forward at the value the variable speed was set to. To get the NXT vehicle to move in reverse, the same code can be used by changing the sign on speed. The following shows the `forwardBackward.ch` rewritten to use the individual motor control.

```
/*************************************************************
forwardBackward.ch
By Michael Schirle

The purpose of this program is to introduce how to get the
nxt_robot to reverse direction.
*************************************************************/
#include <conio.h>
#include <ch_nxt.h>
#include <unistd.h>

//Define Speed Variable
int speed = 25; //the speed of the motors.

//Connect to NXT
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)) {
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(0);
}

//Move the Robot Forward
nxt_forward(speed);
sleep(5);

//Stop the robot in place
nxt_stop();
sleep(1);

//Move the Robot Backwards
nxt_backward(speed);
sleep(5);

//Turn the motors off
```

```
nxt_stop();

//Disconnect NXT
nxt_disconnect();
```

Program 2: Ch program to move the NXT vehicle forward and backward.

9.5 Turning Using Single Motor Control

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

To turn the NXT vehicle left, the right wheel must be moving faster in forward direction then the left wheel. The function `nxt_turnleft()` works by making the left wheel move at 0.7 of the speed that the right wheel is set to. To implement it with the single motor control functions, you would do the following:

```
//Move foward-left
nxt_motor(PORT_B, speed, RUN_IDLE);
nxt_motor(PORT_C, 0.7 * speed, RUN_IDLE);
//Pause program for a while
sleep(5);
```

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

```
//Rotate left in place
nxt_motor(PORT_B, speed, RUN_IDLE);
nxt_motor(PORT_C, -speed, RUN_IDLE);
//Pause program for a while
sleep(5);
```

The `nxt_turnright()` and `nxt_rotateright()` work similarly to `nxt_turnleft()` and `nxt_rotateleft()`, with only the commands to the motors reversed. Once you understand how the single motor commands work, more advance movements can be done, such as a move back and left motion. The single motor commands can also be used to add a third motor attachment to the NXT vehicle, or used to control alternate robot designs that move or act differently.

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

```
//Move foward-left
nxt_motor(PORT_B, 1.2 * speed, RUN_IDLE);
nxt_motor(PORT_C, speed, RUN_IDLE);
//Pause program for a while
sleep(5);
```

While this would work for slower motor speeds, a bug would occur if the NXT vehicle speed was set too high. Can you spot why? As previously discussed, the valid motor speeds for the NXT motors are between -100 to +100. If the speed variable is set to 100, the command to control the left motor (Port C) will function correctly. The right motor (Port B) will recieve a command telling it to set the motor speed to 1.2 x 100 = 120. This is an invalid motor command, and if you tried to run it on the NXT, it will cause the motor to reverse direction, making the NXT vehicle turn right instead of left. It is important to be watchful of these type of errors while writing code to control the NXT.

9.5.1 Manual Real Time Control Program

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

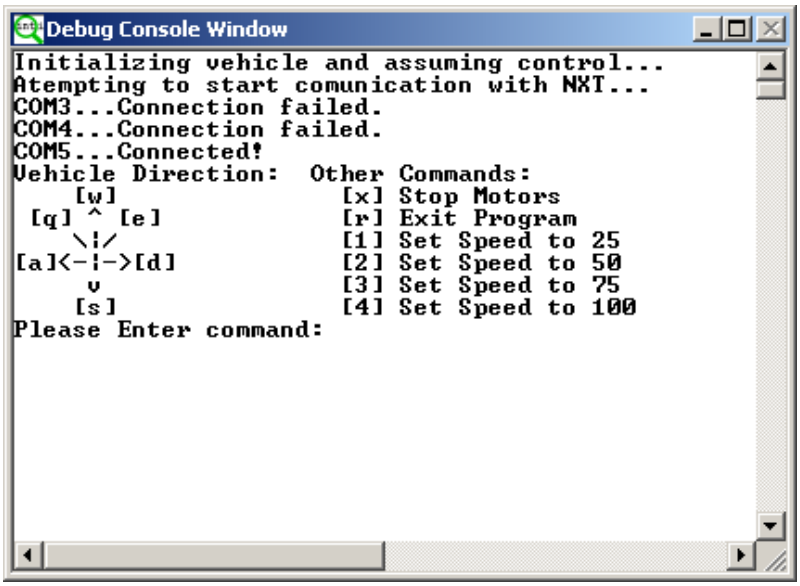

Figure 9: NXT vehicle RTC User Interface

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

- The key `"w"` is to control the NXT to move forward.
- The key `"s"` is to control the NXT to move backward.
- The key `"a"` is to control the NXT to turn left.

- The key "d" is to control the NXT to turn right.
- The key "x" is to stop the NXT motors.
- The key "1" is to set the NXT motor speed to 25.
- The key "r" is to exit the manual RTC program.

In the robot control code block, a while loop is implemented to allow the user to control the NXT continuously until the program is terminated. Within the while loop, the program grabs the user's input and decide what to do with it using the `switch` and `case` statements. The whole NXT vehicle RTC program is shown in Program 2. Please make sure your NXT vehicle are configured according to to Figure **??** to run Program 2. In the rest of this section, we are going to explain the whole program in detail.

```
/*************************************************************
vehicle_rtc.ch
Vehicle Robot Real Time Control Demo
By Joshua Galbraith and Harry H. Cheng

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 <windows.h>
#include <conio.h>
#include <stdio.h>
#include <ch_nxt.h>

struct nxt_remote nxtr;
int speed = 25,//speed of the motors. (default to 25)
        quit = 0,//used by quit case to exit the loop
               status1,//used to check for errors
               status2; //used to check for errors
char key = 'x',//stores the input from the user
        movemode = 'x';//stores the last movement command

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

//GUI display
printf("Vehicle Direction:  Other Commands:");
printf("\n    [w]               [x] Stop Motors");
printf("\n [q] ^ [e]            [r] Exit Program");
printf("\n    \\|/               [1] Set Speed to 25");
printf("\n[a]<-|->[d]           [2] Set Speed to 50");
printf("\n     v                [3] Set Speed to 75");
printf("\n    [s]               [4] Set Speed to 100\n");
printf("Please Enter command:");
```

```
//Control loop. Interprets user command and does action
while (quit != 1 ) {
    key = _getch();
    switch (key) {
        case 'w': //up
            nxt_motor(PORT_B, speed, RUN_IDLE);
            nxt_motor(PORT_C, speed, RUN_IDLE);
            movemode = 'w';
            break;
        case 's': //down
            nxt_motor(PORT_B, -speed, RUN_IDLE);
            nxt_motor(PORT_C, -speed, RUN_IDLE);
            movemode = 's';
            break;
        case 'd': //right
            nxt_motor(PORT_B, -speed, RUN_IDLE);
            nxt_motor(PORT_C, speed, RUN_IDLE);
            movemode = 'd';
            break;
        case 'a': //left
            nxt_motor(PORT_B, speed, RUN_IDLE);
            nxt_motor(PORT_C, -speed, RUN_IDLE);
            movemode = 'a';
            break;
        case 'q': //forward-left
            nxt_motor(PORT_B, speed, RUN_IDLE);
            nxt_motor(PORT_C, speed*0.6, RUN_IDLE);
            movemode = 'q';
            break;
        case 'e': //forward-right
            nxt_motor(PORT_B, speed*0.6, RUN_IDLE);
            nxt_motor(PORT_C, speed, RUN_IDLE);
            movemode = 'e';
            break;
        case 'x': //stop
            nxt_motor(PORT_B, 0, OFF_IDLE);
            nxt_motor(PORT_C, 0, OFF_IDLE);
            movemode = 'x';
            break;
        case 'r': //quit
            printf("\nExiting program.\n");
            quit = 1;
            break;
        case '1': //speed 25
            speed = 25;
            ungetch(movemode);
            break;
        case '2': //speed 50
            speed = 50;
            ungetch(movemode);
            break;
        case '3': //speed 75
            speed = 75;
            ungetch(movemode);
```

```
            break;
        case '4': //speed 100
            speed = 100;
            ungetch(movemode);
            break;
        default:
            printf("\nInvalid Input!\n");
    }//switch(key)
}
nxt_disconnect();   //stop interfacing. This also stops the motors.
printf("Press any key to exit.\n");
while (!_kbhit()); //wait for key press
exit 0;
```

Program 2: Manual Real Time Control Program for NXT vehicle.

Header files

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

```
#include <conio.h>
#include <stdio.h>
#include <ch_nxt.h>
#include <unistd.h>
```

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

Declaring variables

After including the headers, the C program will start the `main()` program. Like any C programs, you must declare variables before you can use them. On the top few lines of the `main()` program, variables are declared.

```
struct nxt_remote nxtr;
int speed = 25,//speed of the motors. (default to 25)
    quit = 0,//used by quit case to exit the loop
    status1,//used to check for errors
    status2; //used to check for errors
char key = 'x',//stores the input from the user
    movemode = 'x';//stores the last movement command
```

- The `nxt_remote` struct stores the connection status, sensor data, and motor counter data of the NXT.
- The integer variable `speed` stores the speed of the motor.

- The integer variable `quit` is used to check if the user wants to quit the program.
- The integer variable `status1` and `status2` are used to check the sensor connection.
- The character variable `key` stores the input from the user.
- The character variable `movemode` stores the last command that the user used.

Checking connection

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

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

User interface

Before the beginning of the real time control, the user must be able to know the function of the key they are pressing. To do this, the program print out a user interface for the user to read. In segment of the Program 4 shown below, the NXT vehicle RTC program used `printf()` command is used to display the user interface for the user to read.

```
//GUI display
printf("Vehicle Direction:  Other Commands:");
printf("\n    [w]               [x] Stop Motors");
printf("\n [q] ^ [e]            [r] Exit Program");
printf("\n    \\|/               [1] Set Speed to 25");
printf("\n[a]<-|->[d]           [2] Set Speed to 50");
printf("\n     v                [3] Set Speed to 75");
printf("\n    [s]               [4] Set Speed to 100\n");
printf("Please Enter command:");
```

Real time control

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

The program fragment shown below shows the beginning and the end of the while loop for Program 4:

```
while (quit != 1 ) {
    key = _getch();
    switch (key){
```

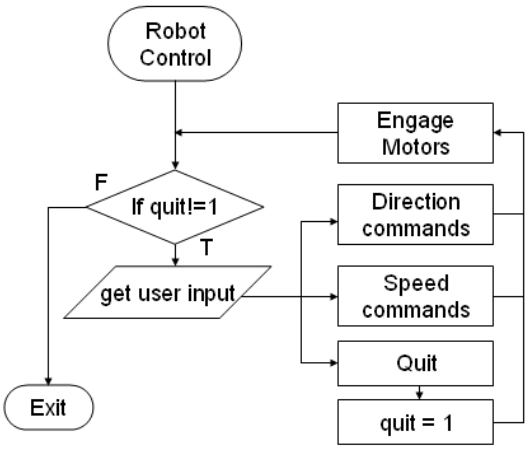

Figure 10: NXT vehicle RTC Control Loop Flow Diagram

```
        ...

        }
    }
```

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.

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

Directional commands

The movements are controlled using the 'w', 's', 'a', and 'd' format. In Figure **??**, 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 `switch()` function will run the codes for the case 'w'. The program fragment for case 'w' is shown below:

```
case 'w': //up
    nxt_motor(PORT_B, speed, RUN_IDLE);
    nxt_motor(PORT_C, speed, RUN_IDLE);
    movemode = 'w';
    break;
```

In case 'w', the program will run the motor in `PORT_B` and `PORT_C` forward at the velocity `speed`, and then set the motor to run idle mode. Next, 'w' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle. Basically, the case 'w' will move the NXT vehicle forward with velocity `speed`.

When the key 's' has been pressed, the `switch()` function will run the codes for the case 's'. The program fragment for case 's' is shown below:

```
case 's': //down
    nxt_motor(PORT_B, -speed, RUN_IDLE);
    nxt_motor(PORT_C, -speed, RUN_IDLE);
    movemode = 's';
    break;
```

In case 's', the program will run the motor in `PORT_B` and `PORT_C` backward at the velocity `speed`, and then set the motor to run idle mode. Next, 's' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle. Basically, the case 's' will move the NXT vehicle backward with velocity `speed`.

When the key 'a' has been pressed, the `switch()` function will run the codes for the case 'a'. The program fragment for case 'a' is shown below:

```
case 'a': //left
    nxt_motor(PORT_B, speed, RUN_IDLE);
    nxt_motor(PORT_C, -speed, RUN_IDLE);
    movemode = 'a';
```

In case 'a', the program will actuate motor in `PORT_B` forward at velocity `speed` and actuate motor in `PORT_C` backward at velocity `speed`, and then set the motor to run idle mode. Next, 'a' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.

When the key 'd' has been pressed, the `switch()` function will run the codes for the case 'd'. The program fragment for case 'd' is shown below:

```
case 'd': //right
    nxt_motor(PORT_B, -speed, RUN_IDLE);
    nxt_motor(PORT_C, speed, RUN_IDLE);
    movemode = 'd';
    break;
```

In case 'd', the program will actuate motor in `PORT_B` backward at velocity `speed` and actuate motor in `PORT_C` forward at velocity `speed`, and then set the motor to run idle mode. Next, 'd' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.

When the key 'q' has been pressed, the `switch()` function will run the codes for the case 'q'. The program fragment for case 'q' is shown below:

```
case 'q': //forward-left
    nxt_motor(PORT_B, speed, RUN_IDLE);
    nxt_motor(PORT_C, speed*0.6, RUN_IDLE);
    movemode = 'q';
    break;
```

In case 'q', the program will actuate motor in `PORT_B` forward at velocity `speed` and actuate motor in `PORT_C` forward at velocity 0.6*`speed`, and then set the motor to run idle mode. Next, 'q' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.

When the key 'e' has been pressed, the `switch()` function will run the codes for the case 'e'. The program fragment for case 'e' is shown below:

```
case 'e': //forward-right
    nxt_motor(PORT_B, speed*0.6, RUN_IDLE);
    nxt_motor(PORT_C, speed, RUN_IDLE);
    movemode = 'e';
    break;
```

In case 'e', the program will actuate motor in `PORT_B` forward at velocity 0.6*`speed` and actuate motor in `PORT_C` forward at velocity `speed`, and then set the motor to run idle mode. Next, 'e' key is stored in the variable `movemode`, which will be used to indicate the current mode for NXT vehicle.

When the key 'x' has been pressed, the `switch()` function will run the codes for the case 'x'. The program fragment for case 'x' is shown below:

```
case 'x': //stop
    nxt_motor(PORT_B, 0, OFF_IDLE);
    nxt_motor(PORT_C, 0, OFF_IDLE);
    movemode = 'x';
    break;
```

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

Speed control

The speed of the motor is controlled by the number key '1', '2', '3', and '4'. In Figure **??**, the user interface shows that each key has a specific speed. For example, key '1' indicates 25 speed, and key '2' indicate 50 speed. As shown below, each of the key has its fragment of code.

```
case '1': //speed 25
    speed = 25;
    ungetch(movemode);
    break;
case '2': //speed 50
    speed = 50;
    ungetch(movemode);
    break;
case '3': //speed 75
    speed = 75;
    ungetch(movemode);
    break;
case '4': //speed 100
    speed = 100;
    ungetch(movemode);
    break;
```

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

Functions for other keys

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

```
case 'r': //quit
    printf("\nExiting program.\n");
    quit = 1;
    break;
```

When the `'r'` key is pressed, the program will print out the statement 'exiting program' and set `quit` variable is set to 1, which allows the program to exit out of the while loop.
For the keys that is not specified to any cases, a default case is used for the time when the user input a wrong key. The program fragment for the default case is shown below:

```
default:
    printf("\nInvalid Input!\n");
```

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

Disconnecting your NXT

After the while loop, the NXT vehicle RTC program ends with the four lines shown below.

```
nxt_disconnect();   //stop interfacing. This also stops the motors.
printf("Press any key to exit.\n");
while (!_kbhit()); //wait for key press
```

These lines of code stop the NXT motors, disconnect the NXT from the computer, ask for the user to press a key, and terminate the program.

9.6 Using NXT Sensors

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

9.6.1 Using your touch sensor

After you have connected your NXT to your PC, you will need to set up the sensor if you would like to include sensors in your NXT program. For example, if you want to add components like light sensor, ultrasonic sensor, and/or touch sensor, you will need to set up their connection using the `nxt_setsensor()` function. The `nxt_setsensor()` function will return 0 if no connection is established, so you can terminate the program if a sensor connection is not established. An example connection check for adding the Touch and Ulstrasonic sensors to the NXT vehicle is as follow:

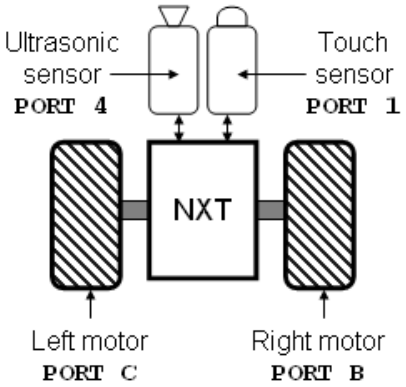

Figure 11: Sensor/Motor configuration of the NXT Vehicle

```
struct nxt_remote nxtr;

//Check status of NXT connection
if (!nxt_connect(&nxtr)){
    exit(0);
}

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

//Save status of PORT_1, and PORT_4
status1=nxt_setsensor(PORT_1,TOUCH,BOOLEANMODE);
status2=nxt_setsensor(PORT_4,ULTRASONIC,RAWMODE);

//Check connection status of PORT_1
if(status1==0) {
    exit(0);
}

//Check connection status of PORT_4
if(status2==0) {
    exit(0);
}
```

As in previous examples, the nxt_remote stucture, `nxtr`, is created, and the initial connection to the NXT is made. The next step is to initialize the NXT sensor ports to the correct types and ensure the sensors are working correctly. The int variables `status1` and `status2` are used to check the return value of the function `nxt_setsensor()`. // A common application of touch sensors for a vehicle robot is for obstacle detection. In order to demonstrate the use of the touch sensor, consider the touch sensor demo in Program 3.

```
/*************************************************************
touchsensor.ch
Vehicle moving forward demo
By Wai Hong Tsang
```

```
The purpose of this demo is to demonstrate the CH Mindstorms
Control Package's ability to control the NXT Mindstorm to use
the touch sensor.
************************************************************/
#include <conio.h>
#include <ch_nxt.h>
#include <unistd.h>

struct nxt_remote nxtr;

//Connect to NXT
if (!nxt_connect(&nxtr)) {
    exit(0);
}

//Set sensor types
int status;
status = nxt_setsensor(PORT_1, TOUCH, BOOLEANMODE);
if (status == 0) {
    exit(0);
}

//Define the variables
int  speed = 25; //the speed of the motors. (default to 25)

//Move Robot Forward
nxt_motor(PORT_B, speed, RUN_IDLE);
nxt_motor(PORT_C, speed, RUN_IDLE);

//Commands:
while (1) {
    //Get touch sensor data
    nxt_getsensor(PORT_1);
    //If touch sensor is triggered
    if (nxtr.NXT_sensorvalraw[PORT_1] < 500) {
        //Move backward
        nxt_motor(PORT_B, -speed, RUN_IDLE);
        nxt_motor(PORT_C, -speed, RUN_IDLE);
        sleep(5);
        //Quit the while loop
        break;
    }
}

//Stop the motors
nxt_motor(PORT_B, 0, OFF_IDLE);
nxt_motor(PORT_C, 0, OFF_IDLE);

//Disconnect NXT
nxt_disconnect();
```

Program 3: Ch NXT touch sensor demonstration program.

Checking touch sensor connection

Program 3 is similar to Program 1, the only changes is the addition of the use of the touch sensor. Program 3 makes the NXT move forward until the touch sensor is triggered, after which it will back up and stop. One of the additions that is added in Program 3 is the initialization of the touch sensor. The fragment of the initialization of the touch sensor is shown below.

```
//Set sensor types
int status;
status = nxt_setsensor(PORT_1, TOUCH, BOOLEANMODE);
if (status == 0){
    exit(0);
}
```

In this fragment, the program use the `nxt_setsensor()` command to set the touch sensor to `PORT 1`. The connection status between the NXT and the sensor is then returned in the variable called status. Next, the if statement check if the connection to the sensor is good. If the variable status is equal to 0, which means no sensor connection, the program will be exited and the rest of the codes will not be executed.

Using while loop

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

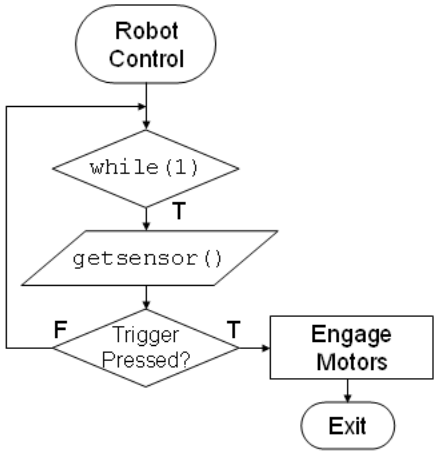

Figure 12: Flow Diagram of the while loop in Program 3

In Program 3, 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 below 500. Then the program will move the NXT backward and disconnect the NXT. The while loop of the touch sensor demonstration program is described below:

```
while(1){
    //Get touch sensor data
```

```
        nxt_getsensor(PORT_1);
        //If touch sensor is triggered
        if (nxtr.NXT_sensorvalraw[PORT_1] < 500){
            //Move backward
            nxt_motor(PORT_B, -speed, RUN_IDLE);
            nxt_motor(PORT_C, -speed, RUN_IDLE);
            sleep(5);
            break;
        }
    }
```

In this while loop, the program uses the `nxt_getsensor()` command to get the data from `PORT 1`, which is the port for the touch sensor according to Figure **??**. Next, the program checks the touch sensor if it is pressed with the if statement. If the touch sensor has been triggered, the codes in the if statement will run. The codes inside the if statement is the same command for moving the NXT backward. After moving the NXT backward, the command break allows the program to exit out of the while loop. This while loop will never exit until the touch sensor is triggered and the break command is used.

9.6.2 Using your ultrasonic sensor

In order to demonstrate the use of the ultrasonic sensor, consider the ultrasonic sensor demo in Program 4. NOTE: If your robot gets stuck, put your hands in front of the ultrasonic sensor to quit the loop.

```
/*************************************************************
ultrasonicsensor.ch
Vehicle moving forward controlled by ultra sonic sensor demo
By Wai Hong Tsang

The purpose of this demo is to demonstrate the CH Mindstorms
Control Package's ability to control the NXT Mindstorm to use
the ultra sonic sensor.
*************************************************************/
#include <conio.h>
#include <ch_nxt.h>
#include <unistd.h>

struct nxt_remote nxtr;

//Connect to NXT
if (!nxt_connect(&nxtr)) {
    printf("Error: Cannot connect to Lego Mindstorm NXT.\n");
    exit(0);
}

//Set sensor types
int status;
status = nxt_setsensor(PORT_4, ULTRASONIC, RAWMODE);
if (status == 0) {
    exit(0);
}

//Define the variables
```

```
int  speed = 25; //the speed of the motors. (default to 25)

//Commands:
while (1) {
    //Get ultrasonic sensor data
    nxt_getsensor(PORT_4);
    //If obstacle is really close
    if (nxtr.NXT_sensorvalraw[PORT_4] < 20) {
        speed = 25;
        //Move backward
        nxt_motor(PORT_B, -speed, RUN_IDLE);
        nxt_motor(PORT_C, -speed, RUN_IDLE);
        sleep(3);
        //Quit the while loop
        break;
    }
    //Else if the obstacle is close
    else if (nxtr.NXT_sensorvalraw[PORT_4] < 60) {
        speed = 25;
    }
    //Else if the obstacle is not close
    else if (nxtr.NXT_sensorvalraw[PORT_4] < 100) {
        speed = 50;
    }
    //Else if there is no obstacle in sight
    else if (nxtr.NXT_sensorvalraw[PORT_4] < 200) {
        speed = 75;
    }
    //Sensor value larger than 200
    else {
        speed = 75;
    }
    //Move forward (constantly)
    nxt_motor(PORT_B, speed, RUN_IDLE);
    nxt_motor(PORT_C, speed, RUN_IDLE);
}

//Stop the motors
nxt_motor(PORT_B, 0, OFF_IDLE);
nxt_motor(PORT_C, 0, OFF_IDLE);

//Disconnect NXT
nxt_disconnect();
```

Program 4: Ch NXT ultrasonic sensor demonstration program.

Checking touch sensor connection

Like Program 3, Program 4 is meant to be a brief demonstration of one method of using an ultrasonic sensor with a vehicle NXT. Similar to the initialization of the touch sensor, the initialization of the ultrasonic sensor is shown below:

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

```
status = nxt_setsensor(PORT_4, ULTRASONIC, RAWMODE);
if (status == 0){
    exit(0);
}
```

In this program fragment, the `nxt_setsensor()` command sets the ultrasonic sensor to `PORT 4`. Next, it checks the return value of the sensor connection status. The program will quit if the return value is 0, meaning there is no connection between the sensor and the NXT.

Contents in the while loop

In Program 4, the while loop uses the ultrasonic sensor to detect distances between the NXT and the obstacles in front of the NXT. The ultrasonic sensor will detect distances and the program code reacts to the data by slowing down or speeding up. The flow diagram of the while loop of Program 4 is shown in Figure **??**.

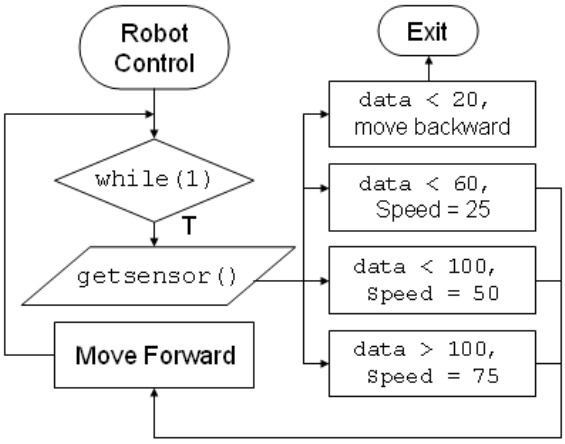

Figure 13: Flow Diagram of the while loop in Program 3

The while loop code block for Program 4 is shown below:

```
//Commands:
while(1){
    //Get ultrasonic sensor data
    nxt_getsensor(PORT_4);
    //If obstacle is really close
    if (nxtr.NXT_sensorvalraw[PORT_4] < 20){
        speed = 25;
        //Move backward
        nxt_motor(PORT_B, -speed, RUN_IDLE);
        nxt_motor(PORT_C, -speed, RUN_IDLE);
        sleep(3);
        //Quit the while loop
        break;
    }
    //Else if the obstacle is close
    else if(nxtr.NXT_sensorvalraw[PORT_4] < 60){
        speed = 25;
```

```
        }
        //Else if the obstacle is not close
        else if(nxtr.NXT_sensorvalraw[PORT_4] < 100){
            speed = 50;
        }
        //Else if there is no obstacle in sight
        else if(nxtr.NXT_sensorvalraw[PORT_4] < 200){
            speed = 75;
        }
        //Sensor value larger than 200
        else{
            speed = 75;
        }
        //Move forward (constantly)
        nxt_motor(PORT_B, speed, RUN_BRAKE);
        nxt_motor(PORT_C, speed, RUN_BRAKE);
    }
```

Similar to Program 3, the while loop in Program 4 also gathers the sensor data using the `nxt_getsensor()` command to gather data from the ultrasonic sensor in `PORT 4`. Next, the if-else statement block determine what to do depending on the distance data from ultrasonic sensor.

- If the sensor value is below 20, which means the vehicle is very close to an obstacle, the program tells the vehicle to reverse, and then break out of the while loop.
- If the sensor value is above 20 and below 60, which means the vehicle is close to an obstacle, the program set the speed variable to 25.
- If the sensor value is above 60 and below 100, which means the vehicle is not close to an obstacle, the program set the speed variable to 50.
- If the sensor value is above 100 and below 200, which means there is nothing in front of the vehicle, the program set the speed variable to 75.
- If the sensor value is other value that is not mentioned above, the program set the speed variable to 75.

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

9.6.3 Autonomous Control Program

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

- 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.

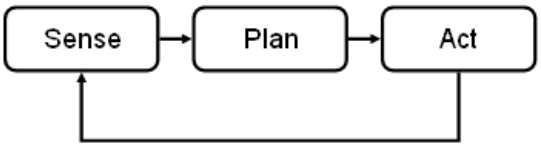

Figure 14: Sense Plan Act Diagram

- Act is to act with robot's surrounding.

The main difference between the manual RTC program and the autonomous program is the content inside the while loop. In the manual RTC program, the codes inside the while loop scan for user's input and the robot acts on the input that the user provided. In the autonomous program, the codes inside the while loop perform the sense-plan-act cycle similarly to the diagram shown in Figure **??**. Every cycle, the information is gathered from the sensor and send back to the computer. The computer will decide what to do depending on the sensor data. The autonomous control program for the NXT vehicle is described in Program 5.

```
/*************************************************************
vehicle_auto.ch
Vehicle Robot Autonomous Exploration Example
By Joshua Galbraith and Harry H. Cheng

The purpose of this demo is to demonstrate the CH Mindstorms
Control Package's ability to control the machine robot model
autonomously, as well as demonstrate how use sensor data from
the NXT to controle it's actions.
*************************************************************/
#include <windows.h>
#include <conio.h>
#include <stdio.h>
#include <ch_nxt.h>


struct nxt_remote nxtr;
int speed = 25, //speed of the motors. (default to 25)
status1 = 2, //used to check for errors
status2 = 2; //used to check for errors
char key = 'x',  //stoes user input
            movemode = 'x'; //stores last movement command
//Connect to NXT
printf("Initializing vehicle and assuming control...");
if (!nxt_connect(&nxtr)){
    printf("\nPress any key to exit.\n");
    while (!_kbhit()); //wait for keypress
    return(0);
}

//Set sensor types
status1 = nxt_setsensor(PORT_1, TOUCH, BOOLEANMODE);
status2 = nxt_setsensor(PORT_4, ULTRASONIC, RAWMODE);
```

```
if ((status1 == 0) || (status2 == 0)){
    printf("\nError initializing sensors.\nPress any key to exit.\n");
    while (!_kbhit()); //wait for keypress
    return(0);
}

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

    //get touch sensor. If pressed reverse and turn left
    nxt_getsensor(PORT_1);
    if (nxtr.NXT_sensorvalraw[PORT_1] < 500){
        nxt_motor_rotate(PORT_B, -25, 720);
        nxt_motor_rotate(PORT_C, -25, 720);
        Sleep(1000);
        nxt_motor(PORT_B, 0, RUN_BRAKE);
        nxt_motor(PORT_C, 0, RUN_BRAKE);
        Sleep(750);
        nxt_motor_rotate(PORT_C, 50, 720);
    }

    // get distance from UltraSonic sensor,
    // set speed according to distance. Turn left if really close.
    nxt_getsensor(PORT_4);
    if (nxtr.NXT_sensorvalraw[PORT_4] < 10){
        nxt_motor_rotate(PORT_B, -25, 720);
        nxt_motor_rotate(PORT_C, -25, 720);
        Sleep(1000);
        nxt_motor(PORT_B, 0, RUN_BRAKE);
        nxt_motor(PORT_C, 0, RUN_BRAKE);
        Sleep(750);
        nxt_motor_rotate(PORT_C, 50, 720);
        speed = 0;
        Sleep(750);
    }
    else if (nxtr.NXT_sensorvalraw[PORT_4] < 30)
        speed = 25;
    else if (nxtr.NXT_sensorvalraw[PORT_4] < 50)
        speed = 50;
    else if (nxtr.NXT_sensorvalraw[PORT_4] < 100)
        speed = 75;
    else if (nxtr.NXT_sensorvalraw[PORT_4] < 200)
        speed = 100;
    else
        speed = 75;

    // Turn motors on (drive forward)
    nxt_motor(PORT_B, speed, RUN_BRAKE);
```

```
    nxt_motor(PORT_C, speed, RUN_BRAKE);
}
nxt_disconnect();   //stop interfacing. This also stops the motors.
printf("\nPress any key to exit.\n");
while (!_kbhit()); //wait for keypress
```

Program 5: Autonomous Control Program for NXT vehicle.

In Program 5, the sensors that is used are the touch sensor and the ultra sonic sensor. These sensors are located in the front of the vehicle so that when the vehicle encounters an obstacle, the program will control the robot to avoid or steer away from it. A diagram of the vehicle and its sensor and actuators of the program is shown in Figure **??**. Please make sure your NXT vehicle are configured according to to Figure **??** to run Program 5.

Exiting the while loop

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

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

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

Touch sensor

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

```
//get touch sensor. If pressed reverse and turn left
nxt_getsensor(PORT_1);
if (nxtr.NXT_sensorvalraw[PORT_1]<500){
    nxt_motor_rotate(PORT_B,-25,720);
    nxt_motor_rotate(PORT_C,-25,720);
    sleep(1);
    nxt_motor(PORT_B,0,RUN_BRAKE);
    nxt_motor(PORT_C,0,RUN_BRAKE);
    sleep(.750);
    nxt_motor_rotate(PORT_C,50,720);
}
```

In the first line of this fragment, the NXT gathers data from `PORT_1`, which is the port for the touch sensor. Next, it checks the value for the touch sensor data with an if statement. If the value of the touch sensor data is less than 500, which means the touch sensor has been triggered, the program will execute the obstacle avoidance commands inside the if statement. The commands in the if statement control the NXT vehicle to reverse, then stop, and then steer left.

Ultrasonic sensor

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

```
// get distance from UltraSonic sensor,
// set speed according to distance. Turn left if really close.
nxt_getsensor(PORT_4);
if(nxtr.NXT_sensorvalraw[PORT_4]<10){
    nxt_motor_rotate(PORT_B,-25,720);
    nxt_motor_rotate(PORT_C,-25,720);
    sleep(1);
    nxt_motor(PORT_B,0,RUN_BRAKE);
    nxt_motor(PORT_C,0,RUN_BRAKE);
    sleep(.750);
    nxt_motor_rotate(PORT_C,50,720);
    speed=0;
    sleep(.750);
}
else if (nxtr.NXT_sensorvalraw[PORT_4]<30)
    speed =25;
else if (nxtr.NXT_sensorvalraw[PORT_4]<50)
    speed =50;
else if (nxtr.NXT_sensorvalraw[PORT_4]<100)
    speed =75;
else if (nxtr.NXT_sensorvalraw[PORT_4]<200)
    speed =100;
else
    speed =75;
```

In the first line of this fragment, the NXT gathers data from `PORT_4`, 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. Here is the list of sensor value threshold and its commands.

- If the sensor value is below 10, which means the vehicle is very close to an obstacle, the program tells the vehicle to reverse, stop, and then steer left.
- If the sensor value is above 10 and below 30, which means the vehicle is close to an obstacle, the program set the speed variable to 25.
- If the sensor value is above 30 and below 50, which means the vehicle sees an incoming obstacle, the program set the speed variable to 50.

- If the sensor value is above 50 and below 100, the program set the speed variable to 75.
- If the sensor value is above 100 and below 200, which means there is nothing in front of the vehicle, the program set the speed variable to 100.
- If the sensor value is other value that is not mentioned above, the program set the speed variable to 75.

Running forward

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

```
// Turn motors on (drive forward)
nxt_motor(PORT_B,speed,RUN_BRAKE);
nxt_motor(PORT_C,speed,RUN_BRAKE);
```

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

10 Controlling Non-Vehicle NXT Robots

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

10.1 Controlling NXT Machine

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

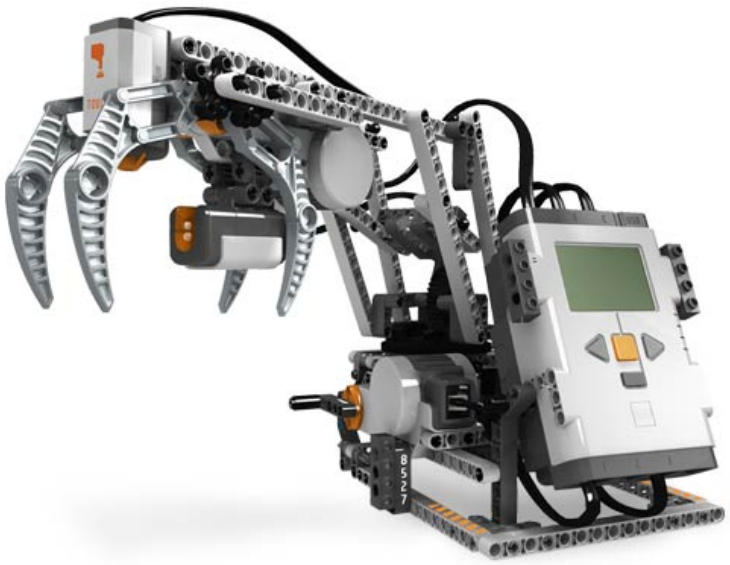

Figure 15: NXT Machine

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

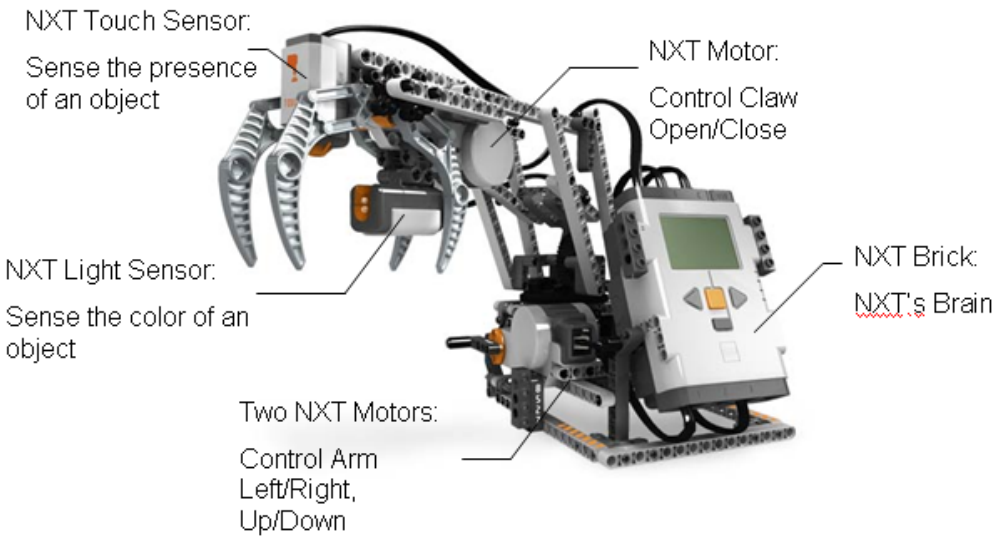

Figure 16: Components of the NXT Machine

10.1.1 Manual Real Time Control Program

In this section the manual real time control for the NXT machine will be introduced and described. The manual RTC for the NXT machine allows the user to control the NXT machine manually via the keyboard. The user interface of the RTC program for the NXT machine is shown in Figure **??** The user interface and the user commands for the NXT machine RTC program is a bit different compared to the NXT vehicle RTC program. Instead of controlling the direction by pressing one key, the user will be required to press a key for direction or command, and then enter a number to set the angle or speed. When a specific key is pressed during the execution of the NXT machine RTC program, the program uses a `switch()` statement to performs a fragment of code that send commands to the NXT to move in a direction or perform a task. If a directional key or aset 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:

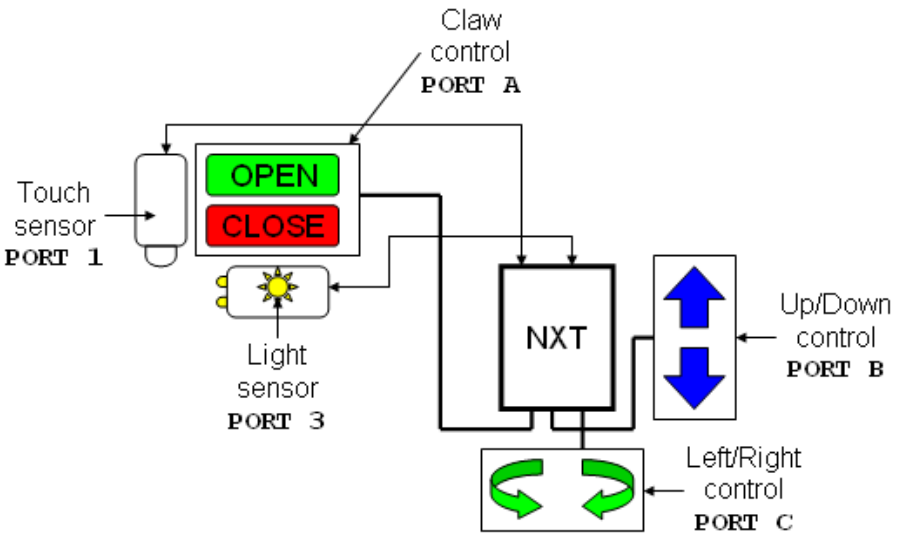

Figure 17: Sensor/Motor configuration of the NXT Machine

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

The NXT machine RTC program is described in Program 6. In the rest of this section, we are going to explain important parts of the manual rtc program in detail.

```
/*************************************************************
machine_rtc.ch
Machine Robot Real Time Control Demo
By Joshua Galbraith

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 <ch_nxt.h>         //this is the control package.
```

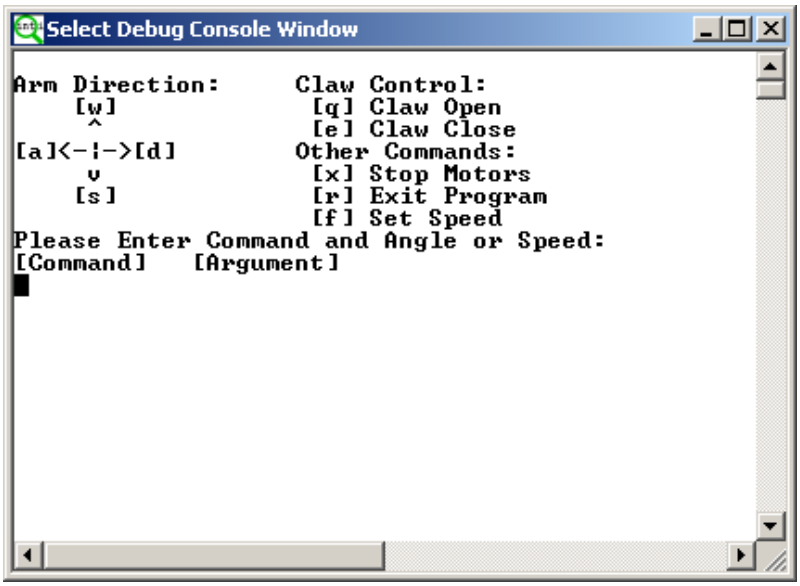

Figure 18: NXT vehicle RTC User Interface

```
struct nxt_remote nxtr; //structure that holds NXT information.
int speed = 100;        //used to control the motor speed.
int angle = 0;          //stores the angle input from the user.
int quit = 0;           //used to break out of the control loop
int result = 0;
double gearratio = (8.0 / 56) * (1.0 / 24); //gear ratio on the arm
char dir = 0;           //stores "direction to move" input from user.
char color[5];
char temp[20];
char *temp_loc;
printf("gear ratio: %f", gearratio);

//call nxt_connect function and check for success
printf("\nInitializing arm and assuming control...");

result = nxt_connect(&nxtr);
if (result == 0) {
    while (!_kbhit());  //wait for key press
    nxt_disconnect();   //stop interfacing. This also stops the motors.
    exit(0);
}//if (result ==0)


//Initialize sensor and check for success
result = nxt_setsensor(PORT_1, TOUCH, BOOLEANMODE);
if (result == 0) {
    printf("\nSensor Setup failed. Exiting program.");
    while (!_kbhit());  //wait for key press
    nxt_disconnect();   //stop interfacing. This also stops the motors.
    exit(0);
```

```
}//if (result ==0)

result = nxt_setsensor(PORT_3, LIGHT_ACTIVE, RAWMODE);
if (result == 0) {
    printf("\nSensor Setup failed. Exiting program.");
    while (!_kbhit());  //wait for key press
    nxt_disconnect();   //stop interfacing. This also stops the motors.
    exit(0);
}//if (result ==0)


/*This is the user input loop, that gets the user input and
sends the commands to the NXT accordingly.
w,s move arm up and down
a,d move arm left and right
q,e open and close the claw
x stops the motor
r quits the program
f sets the speed (default to 100)    */

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

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

    // scanf("%c %d",&dir, &angle);

    switch (dir) {
        case 'a': //Arm rotate left.
            nxt_motor_rotate(PORT_C, speed, (int)(angle / gearratio));
            break;
        case 'd': //Arm rotate right.
            nxt_motor_rotate(PORT_C, -speed, (int)(angle / gearratio));
            break;
        case 'w': //Raise arm up.
            nxt_motor_rotate(PORT_B, speed, angle);
            break;
        case 's': //lower arm down
            nxt_motor_rotate(PORT_B, -speed, angle);
```

```
            break;
        case 'q': //claw open
            nxt_motor(PORT_A, -15, RUN_IDLE);
            Sleep(1000);
            nxt_motor(PORT_A, 0, RUN_BRAKE);
            break;
        case 'e': //claw close
            nxt_motor(PORT_A, 15, RUN_IDLE);
            Sleep(1000);
            nxt_motor(PORT_A, 0, RUN_BRAKE); //puts motor in break mode
            break;
        case 'x': //stop
            nxt_motor_zero();
            nxt_motor(PORT_A, 0, OFF_IDLE);
            nxt_motor(PORT_B, 0, OFF_IDLE);
            nxt_motor(PORT_C, 0, OFF_IDLE); //puts motor in idle mode
            break;
        case 'r': //quit
            printf("\nQuit.");
            quit = 1;
            break;
        case 'f': //set speed
            speed = angle;
            printf("\nSpeed set to %d.", speed);
            break;
        default:
            printf("\n");
    }//switch(dir)
    Sleep(200);
    nxt_getsensor(PORT_1);
    if (nxtr.NXT_sensorvalraw[PORT_1] < 500) {
        printf("    The Ball was grabbed ");
        nxt_getsensor(PORT_3);
        if (nxtr.NXT_sensorvalraw[PORT_3] < 500) {
            printf("and the color is red\n");
        }
        else {
            printf("and the color is blue\n");
        }
    }

}


nxt_disconnect(); //stop interfacing. This also stops the motors.
while (!_kbhit()); //wait for key press
```

Program 6: Manual Real Time Control Program for NXT machine.

How does it work?

The initialization and the termination of the NXT machine RTC program is very similar to the NXT vehicle RTC program. The biggest difference between the two programs is in the

while loop of the program. In this section, we will focus on how the while() loop of the NXT machine RTC program work internally.

While loop

Similar to the while loop of the NXT vehicle RTC program, the whileloop of the NXT machine RTC program scans for the variable quit to see if it is set to 1. If the 'r' key has been pressed, the variable quit will be set to 1 and the while loop 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 or a set speed key, the program ask for the user to input a speed or 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') || (dir == 'f')){
        printf("  Enter angle or Speed: ");
        scanf("%d", &angle);
    }
    switch (dir){

    ...

    }
}
```

Depending on what key was pressed, the program will run a fragment of code that sends commands to the NXT using the `switch()` command.

Directional commands

The directional movements are controlled using the 'w', 's', 'a', and 'd' keys. In Figure **??**, 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 `switch()` function will run the codes for the case 'a'. The program fragment for case 'a' is shown below:

```
case 'a': //Arm rotate left. Adjusted angle.
    nxt_motor_rotate(PORT_C, speed, (int)(angle / gearratio));
    break;
```

In case 'a', the program will run the motor in `PORT_C` at velocity `speed`, and to an `angle` divided by the gear ratio that the user has entered. Basically, for case 'a' the program will rotate the NXT machine arm left to an adjusted angle that the user has entered.
When the key 'd' has been pressed, the `switch()` function will run the codes for the case 'd'. The program fragment for case 'd' is shown below:

```
case 'd': //Arm rotate right. Adjusted angle.
    nxt_motor_rotate(PORT_C, -speed, (int)(angle / gearratio));
    break;
```

In case 'd', the program will run the motor in `PORT_C` at velocity `-speed`, and to an `angle` divided by the gear ratio that the user has entered. Basically, for case 'd' the program will rotate the NXT machine arm right to an adjusted angle that the user has entered.
When the key 'w' has been pressed, the `switch()` function will run the codes for the case 'w'. The program fragment for case 'w' is shown below:

```
case 'w': //Raise arm up.
    nxt_motor_rotate(PORT_B, speed, angle);
    break;
```

In case 'w', the program will run the motor in `PORT_B` at velocity `speed`, and to a prescribed `angle` that the user has entered. Basically, for case 'w' the program will move the NXT machine arm upward to a prescribed angle that the user entered to an angle at a set speed.
When the key 's' has been pressed, the `switch()` function will run the codes for the case 's'. The program fragment for case 's' is shown below:

```
case 's': //lower arm down
    nxt_motor_rotate(PORT_B, -speed, angle);
    break;
```

In case 's', the program will run the motor in `PORT_B` at velocity `-speed`, and to a prescribed `angle` that the user has entered. Basically, for case 'w' the program will move the NXT machine arm downward to a prescribed angle that the user entered to an angle at set speed.

Discrete commands

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

```
case 'q': //claw open
    nxt_motor(PORT_A, -15, RUN_IDLE);
    sleep(1);
    nxt_motor(PORT_A, 0, RUN_BRAKE);
    break;
```

In case 'q', the program will run the motor in `PORT_A` at velocity `-15`, and run idly for 1000 milliseconds. Next, the program will hold the position of the motor at that spot, thus keeping the machine claw open. Basically, for case 'q' the program will open the machine claw.
When the key 'e' has been pressed, the `switch()` function will run the codes for the case 'e'. The program fragment for case 'e' is shown below:

```
case 'e': //claw close
    nxt_motor(PORT_A, 15, RUN_IDLE);
    sleep(1);
    nxt_motor(PORT_A, 0, RUN_BRAKE); //puts motor in break mode
    break;
```

In case 'e', the program will run the motor in `PORT_A` at velocity `15`, and run idly for 1000 milliseconds. Next, the program will hold the position of the motor at that spot, thus keeping the machine claw close. Basically, for case 'e' the program will close the machine claw. When the key 'x' has been pressed, the `switch()` function will run the codes for the case 'x'. The program fragment for case 'x' is shown below:

```
case 'x': //stop
    nxt_motor_zero();
    nxt_motor(PORT_A, 0, OFF_IDLE);
    nxt_motor(PORT_B, 0, OFF_IDLE);
    nxt_motor(PORT_C, 0, OFF_IDLE); //puts motor in idle mode
    break;
```

In case 'x', the program will stop all the motors, and turn the mode to off and idle.
When the key 'r' has been pressed, the `switch()` function will run the codes for the case 'r'. The program fragment for case 'r' is shown below:

```
case 'r': //quit
   printf("\nQuit.");
   quit = 1;
```

In case 'x', 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 setup

To set the speed of the movement of the arm, the user can enter the key 'f' and enter the speed. The speed of the motor are in the range of integer of -100 to 100 (negative meaning rotating backward). The program fragment of case 'f' is shown below:

```
case 'f': //set speed
    speed = angle;
    printf("\nSpeed set to %d.", speed);
```

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:

```
sleep(.200);
nxt_getsensor(PORT_1);
if (nxtr.NXT_sensorvalraw[PORT_1] < 500){
    printf("    The Ball was grabbed ");
    nxt_getsensor(PORT_3);
    if (nxtr.NXT_sensorvalraw[PORT_3] < 500){
        printf("and the color is red\n");
    }
    else{
        printf("and the color is blue\n");
    }
}
```

In this fragment, the program will sleep for 200 milliseconds and then grab the sensor value stored in `PORT_1` using the `nxt_getsensor()` command, which is the touch sensor. Afterward, the if statement checks if the claw has grabbed an object. If the sensor value for the touch sensor is greater than 500, the touch sensor is not triggered, so there is the NXT detected that there is no object in its claw. If the sensor value for the touch sensor is less than 500, the touch sensor is triggered and the NXT detected that the claw has grabbed an

object.
When the touch sensor is trigged, the program will continue in the if statement, and the program will prints out that "The Ball was grabbed" in the screen and it get the sensor value stored in `PORT_3`, which is the light sensor. Next, the program will determine the color of the ball using the light sensor value. If the light sensor value is less than 500, the NXT will detect that the ball that was grabbed is red and program will print out "and the color is red". If the light sensor value is greater than 500, the NXT will detect that the ball that was grabbed is blue and program will print out "and the color is blue".

10.1.2 Autonomous Control Program

In this section, the autonomous control program for the NXT machine will be introduced. This autonomous program uses the NXT machine arm to scan it's surrounding. It performs this task by rotating its arm by an angle step and 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 NXT machine automatic control program is described in Program 7. In the rest of this section, we are going to explain important parts of the automatic control program in detail.

```
/*************************************************************
machine_auto.ch
Machine Robot Autonomous Data Collection Example
By Joshua Galbraith

The purpose of this demo is to demonstrate the CH Mindstorms
Control Package's ability to control the machine robot model
autonomously, as well as demonstrate how to collect and plot
sensor data from the NXT.
*************************************************************/



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

FILE *stream;

class CPlot plot;
struct nxt_remote nxtr;
//int speed=30;
int quit = 0,    //used to exit for loop
        i,           //counter variable
        result;      //stores result of function
double gearratio = (8.0 / 56) * (1.0 / 24);
enum {numpoints = 90};  //desired number of data points
const int anglestep = 2; //angle moved between steps
double angle[numpoints];//angle calculated from the tachometer
int distance[numpoints];//data received from the ultrasonic sensor

//Connect to NXT exit if failure
if (!nxt_connect(&nxtr)) {
```

```
    printf("\nPress any key to quit.");
    while (!_kbhit()); //wait for key press
    exit(0);
}

//Initialize arm. (Set sensor types and initialize variables)
printf("\nInitializing arm for autonomous control...\n");
result = nxt_setsensor(PORT_3, ULTRASONIC, RAWMODE);
if (result == 0) {
    printf("\nSensor Setup failed. Exiting program.");
    while (!_kbhit());  //wait for key press
    nxt_disconnect(); //stop interfacing. This also stops the motors.
    exit(0);
}
for (i = 0; i < numpoints; i++) {
    angle[i] = 0;
    distance[i] = 0;
}

//print usage information to the user
printf("\n%d Data points will be collected with a"
       "step size of %d.", numpoints, anglestep);
printf("\nPlease ensure that the arm can rotate"
       "%d degrees from its current position.", (numpoints*anglestep));
printf("\nPress any key to continue. Press q at any time to quit.");
if (getch() == 'q') {
    printf("\nQuitting program.");
    Sleep(1500);
    exit(0);
}

//begin Autonomous loop
for (i = 0; i < numpoints; i++) {

    //get sensor data, if success print data, else print error
    if (nxt_getsensor(PORT_3)) {
        distance[i] = nxtr.NXT_sensorvalraw[PORT_3];
        if (nxt_gettacho(PORT_C)) {
            angle[i] = (nxtr.NXT_motor_pos_cum[PORT_C] * gearratio);
            printf("\nSample: %d,  distance: %d,  Angle: %f",
                   i, distance[i], angle[i]);
        }
    }
    else printf("\nError!");


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

```
    //rotate arm by anglestep (rotate motor anglestep/gear ratio)
    nxt_motor_rotate(PORT_C, 100, (anglestep / gearratio));
    Sleep(1000);

} //end of for(i=0;i<numpoints-1;i++)


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


//log data to the file output.csv
stream = fopen("output.csv", "w");
for (i = 0; i < numpoints; i++) {
    fprintf(stream, "%f,%d\n", angle[i], distance[i]);
}
fclose(stream);

//plot data in CH using a polar graph
plot.polarPlot(PLOT_ANGLE_DEG);
plot.data2D(angle, distance);
plot.sizeRatio(1);
plot.grid(PLOT_ON);
plot.plotting();

//wait for user and exit program
printf("Data plotted.\nPress any key to exit.");
while (!_kbhit()); //wait for key press
```

Program 7: Automatic Control Program for NXT machine.

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

How does it work?

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

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

The variable `numpoints` determines how many number of times the for loop will run and the variable `angles` determines how much the arm rotates in degrees of angle.
Another feature this program has is the usage information printout described in the program fragment below.

```
    printf("\n%d Data points will be collected with a"
```

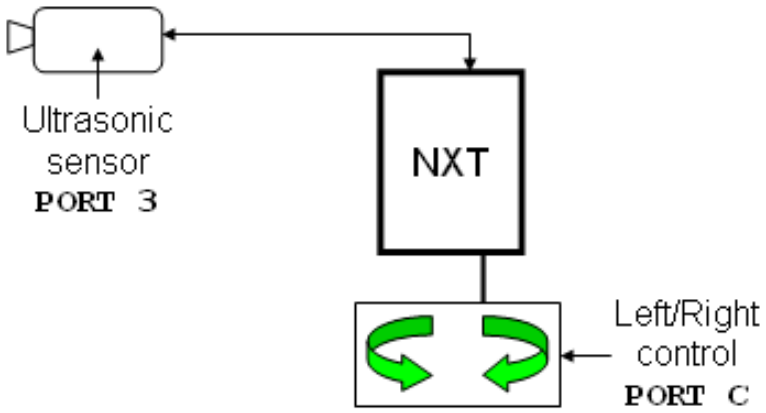

Figure 19: NXT vehicle RTC User Interface

```
        "step size of %d.",numpoints,anglestep);
    printf("\nPlease ensure that the arm can rotate"
        "%d degrees from its current position.",(numpoints*anglestep));
    printf("\nPress any key to continue. Press q at any time to quit.");
    if(getch()=='q'){
        printf("\nQuitting program.");
        sleep(1.500);
        exit(0);
}
```

In this program fragment, the program will print out how many data point will be collected with the angle step size. Next, it will calculate and print out the full rotation angle of the robot arm and ask the user to ensure that the arm can rotate that amount 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 NXT machine. The for loop uses a series of if-else statements to perform data collection and arm movement. There are three sections for this for loop. The first section is to collect, store, and print ultrasonic data and angle rotation data. The second section scans for user input to see if the user has pressed 'q' to quit the program. The last section is to rotate the arm by an angle step. The program fragment below is the for loop for the autonomous program for the NXT machine.

```
for(i=0;i<numpoints;i++){
    //get sensor data, if success print data, else print error
    if(nxt_getsensor(PORT_3)){
    distance[i]=nxtr.NXT_sensorvalraw[PORT_3];
        if (nxt_gettacho(PORT_C)){
            angle[i]= (nxtr.NXT_motor_pos_cum[PORT_C]*gearratio);
            printf("\nSample: %d,  distance: %d,  Angle: %f",
```

```
                i, distance[i],angle[i]);
        }
    }
    else printf("\nError!");
    //check if q was pressed and if so exit program
    if (!_kbhit){
        if(getch()=='q'){
            printf("\nQuitting program.");
            break;
        }
    }
    //rotate arm by anglestep (rotate motor anglestep/gear ratio)
    nxt_motor_rotate(PORT_C,100,(anglestep/gearratio));
    sleep(1);
}
```

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 store 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 `nxt_motor_rotate()` command. Lastly, the program freezes for 1000 milliseconds by using the `Sleep()` command.

Creating and writing data file

Before the termination of the autonomous control program for the NXT machine, the program creates a data file and stores the collected data in that file. The program fragment shown below is the program codes for creating and storing a data file for the data that was collected:

```
stream=fopen("output.csv","w");
for(i=0;i<numpoints;i++){
    fprintf(stream,"%f,%d\n",angle[i],distance[i]);
}
fclose(stream);
```

In this fragment, the program first creates a file called output.csv and set it to write mode and store the file pointer to a variable called stream. Next, the for loop writes the angle data and the distance data onto the file one set at a time. At last, the program closes the file by using the function `fclose()`.

Plotting data

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

the data in a polar diagram.

```
//plot data in CH using a polar graph
plot.polarPlot(PLOT_ANGLE_DEG);
plot.data2D(angle,distance);
plot.sizeRatio(1);
plot.grid(PLOT_ON);
plot.plotting();
```

In this fragment, the program uses the CPlot class commands to do its plotting. First, the program use the function `polarPlot()` to set the plot to polar and degrees in angle. Next, the program use the function `data2D()` to insert the collected data onto the polar plot. Afterward, the program uses the function `sizeRatio()` function and `grid()` command to correct the size of the plot and to add grid to the plot. Finally, the program creates the plot by using the function `plotting()`.

10.2 Controlling NXT Humanoid

Figure 20: NXT Humanoid

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

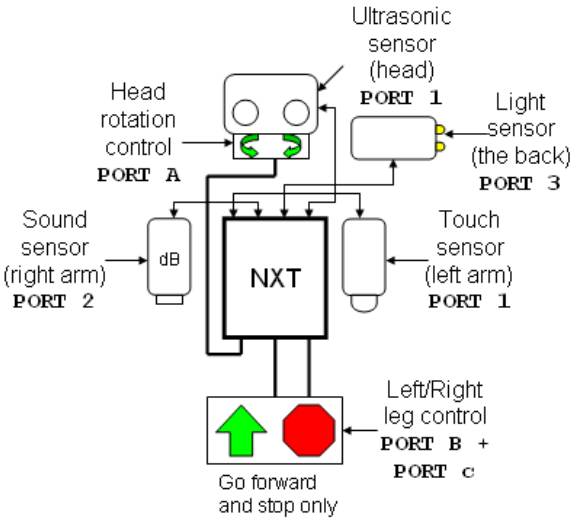

Figure 21: Sensor/Motor configuration of the NXT Humanoid

10.2.1 Manual Real Time Control Program

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

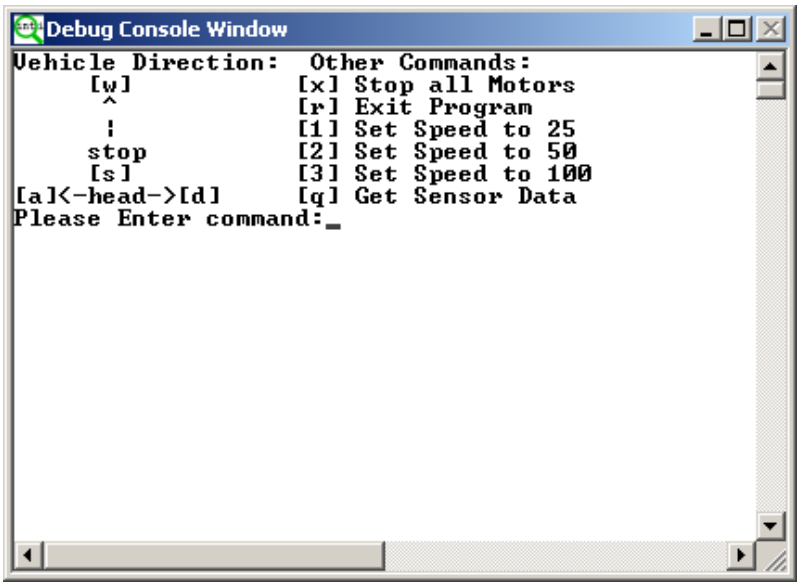

Figure 22: NXT Humanoid RTC User Interface

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

The NXT humanoid real time control program is described in Program 8. In the rest of this section, we are going to explain important parts of the manual RTC program in detail.

```
/*************************************************************
humanoid_rtc.ch
Humanoid Robot Real Time Control Demo
By Joshua Galbraith

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 <windows.h>
#include <conio.h>
#include <stdio.h>
#include <ch_nxt.h>

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

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


//Set sensor types
status1 = nxt_setsensor(PORT_1, TOUCH, BOOLEANMODE);
status2 = nxt_setsensor(PORT_2, SOUND_DB, RAWMODE);
status3 = nxt_setsensor(PORT_4, LIGHT_INACTIVE, RAWMODE);
status4 = nxt_setsensor(PORT_4, ULTRASONIC, RAWMODE);
if ((status1 == 0) || (status2 == 0) || (status3 == 0)
        || (status4 == 0)) {
    printf("\nError initializing sensors.\nPress any key to exit.\n");
    while (!_kbhit()); //wait for key press
    exit(0);
}

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

//Control loop. Interprets user command and does action
while (quit != 1)
{
    key = _getch();
```

```
    switch (key) {
        case 'w': //up
            nxt_motor(PORT_B, speed, RUN_IDLE);
            nxt_motor(PORT_C, speed, RUN_IDLE);
            movemode = 'w';
            break;
        case 's': //down
            nxt_motor(PORT_B, 0, OFF_IDLE);
            nxt_motor(PORT_C, 0, OFF_IDLE);
            movemode = 's';
            break;
        case 'd': //right
            nxt_motor(PORT_A, 30, RUN_IDLE);
            break;
        case 'a': //left
            nxt_motor(PORT_A, -30, RUN_IDLE);
            break;
        case 'q': //print sensor
            printsensor(&nxtr);
            break;
        case 'x': //stop
            nxt_motor(PORT_B, 0, OFF_IDLE);
            nxt_motor(PORT_C, 0, OFF_IDLE);
            movemode = 'x';
            break;
        case 'r': //quit
            printf("\nExiting program.\nPress any key to exit.");
            quit = 1;
            break;
        case '1': //speed 25
            speed = 25;
            ungetch(movemode);
            break;
        case '2': //speed 50
            speed = 50;
            _ungetch(movemode);
            break;
        case '3': //speed 75
            speed = 75;
            ungetch(movemode);
            break;
        case '4': //speed 100
            speed = 100;
            ungetch(movemode);
            break;

        default:
            printf("\nInvalid Input!\n");
    }//switch(key)

}
nxt_disconnect();   //stop interfacing. This also stops the motors.
while (!_kbhit()); //wait for key press
```

```
int printsensor(struct nxt_remote *nxtr) {
    int touch = 0;
    int ultra = 0;
    int sound = 0;
    int light = 0;

    nxt_getsensor(PORT_1);
    nxt_getsensor(PORT_2);
    nxt_getsensor(PORT_3);
    nxt_getsensor(PORT_4);
    touch = nxtr->NXT_sensorvalraw[PORT_1];
    sound = nxtr->NXT_sensorvalraw[PORT_2];
    light = nxtr->NXT_sensorvalraw[PORT_3];
    ultra = nxtr->NXT_sensorvalraw[PORT_4];

    if (touch < 500)
        printf("\n\n\nThe touch sensor has been activated.\n", touch);
    else
        printf("\nThe touch sensor has not been activated.\n");

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

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

Program 8: Manual Real Time Control Program for NXT machine.

How does it work?

The the main difference between the RTC program for the NXT humanoid and the NXT vehicle is the content in the switch cases. Other than the switch cases, the while loop of the RTC program is the same as the previous RTC program. When the key 'r' has been pressed, the `switch()` function will run the codes for the case 'r'. In the case 'r', the variable quit will be to 1, which will allow the program to exit the while loop, thus quitting the program. When the key 'w' has been pressed, the `switch()` function will run the codes for the case 'w'. The program fragment for case 'w' is shown below:

```
case 'w': //up
    nxt_motor(PORT_B, speed, RUN_IDLE);
    nxt_motor(PORT_C, speed, RUN_IDLE);
    movemode='w';
    break;
```

In case 'w', the program will run the motor in `PORT_B` and `PORT_C` at velocity `speed`. Basically, for case 'w' the program will control the NXT humanoid to walk forward at a speed.
When the key 's' has been pressed, the `switch()` function will run the codes for the case 's'. The program fragment for case 's' is shown below:

```
case 's': //down
    nxt_motor(PORT_B, 0, OFF_IDLE);
    nxt_motor(PORT_C, 0, OFF_IDLE);
    movemode='s';
    break;
```

In case 's', the program will stop the motor in `PORT_B` and `PORT_C` and turn them to off idle mode. Basically, for case 's' the program will stop the NXT humanoid from walking forward.
When the key 'a' has been pressed, the `switch()` function will run the codes for the case 'a'. The program fragment for case 'a' is shown below:

```
case 'a': //left
    nxt_motor(PORT_A, -30, RUN_IDLE);
    break;
```

In case 'a', the program will run the motor in `PORT_A` at -30 speed. Basically, for case 'a' the program will rotate the NXT humanoid head left.
When the key 'd' has been pressed, the `switch()` function will run the codes for the case 'd'. The program fragment for case 'd' is shown below:

```
case 'a': //left
    nxt_motor(PORT_A, -30, RUN_IDLE);
    break;
```

In case 'd', the program will run the motor in `PORT_A` at 30 speed. Basically, for case 'd' the program will rotate the NXT humanoid head right.
When the key 'q' has been pressed, the `switch()` function will run the codes for the case 'q'. The program fragment for case 'q' is shown below:

```
case 'q': //print sensor
    printsensor(&nxtr);
    break;
```

In case 'q', the program will run the function `printsensor()`, which will get sensor data from each data and print these data out for the user. The details of the `printsensor()` function will be discussed later in this section. // When the key 'x' has been pressed, the `switch()` function will run the codes for the case 'x'. The program fragment for case 'x' is shown below:

```
case 'x': //stop
    nxt_motor(PORT_B, 0, OFF_IDLE);
    nxt_motor(PORT_C, 0, OFF_IDLE);
    movemode='x';
    break;
```

In case 'x', the program will stop all the motors connected to the NXT and set them to off idle mode. //

Printing sensor data

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

```
int printsensor(struct nxt_remote *nxtr){
    int touch=0;
    int ultra=0;
    int sound=0;
    int light=0;

    nxt_getsensor(PORT_1);
    nxt_getsensor(PORT_2);
    nxt_getsensor(PORT_3);
    nxt_getsensor(PORT_4);
    touch=nxtr->NXT_sensorvalraw[PORT_1];
    sound=nxtr->NXT_sensorvalraw[PORT_2];
    light=nxtr->NXT_sensorvalraw[PORT_3];
    ultra=nxtr->NXT_sensorvalraw[PORT_4];

    if (touch<500)
        printf("\n\n\nThe touch sensor has been activated.\n",touch);
    else
        printf("\nThe touch sensor has not been activated.\n");

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

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

In the beginning of this function, the program get sensor data from each sensor and store them in its corresponding variable. Next, if else statements are used to print the correct statement if the sensor is triggered or not. For example, if the touch sensor is triggered, the sensor data retrieved is below 500, 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.

11 Appendix

11.1 User Functions

Functions	Description
`nxt_connect()`	Connects to the Mindstorms NXT using blue tooth.
`nxt_forward()`	Moves the NXT vehicle forward.
`nxt_backward()`	Moves the NXT vehicle backward.
`nxt_turnleft()`	Turns the NXT vehicle left.
`nxt_turnright()`	Turns the NXT vehicle right.
`nxt_rotateleft()`	Rotates the NXT vehicle to the left in place.
`nxt_rotateright()`	Rotates the NXT vehicle to the right in place.
`nxt_stop()`	Stops all motors on the NXT.
`nxt_motor()`	Sets the motor output for NXT.
`nxt_motor_rotate()`	Rotate the motor for NXT for given degree.
`nxt_setsensor()`	Set up the sensors to collect data for the NXT.
`nxt_getsensor()`	Get sensor data from NXT.
`nxt_printmess()`	Prints the last message send to or receive from NXT.
`nxt_disconnect()`	Disconnect the NXT.
`nxt_gettacho()`	Get tachometer counts from NXT.
`nxt_motor_zero()`	Reset tachometer count to zero for all motors.

11.2 Functions

Name	Value	Description
		Motor Macros
Motor Ports:		
PORT_A	0	Select output PORT A.
PORT_B	1	Select output PORT B.
PORT_C	2	Select output PORT C.
Motor Mode:		
RUN_IDLE	0	Sets motor to run, idle.
RUN_BRAKE	1	Sets motor to run, brake.
OFF_IDLE	3	Sets motor to off, idle.
		Sensor Macros
Sensor Ports:		
PORT_1	0	Select sensor input PORT 1.
PORT_2	1	Select sensor input PORT 2.
PORT_3	2	Select sensor input PORT 3.
PORT_4	3	Select sensor input PORT 4.
Sensor Type:		
TOUCH	1	Sets to Touch sensor.
TEMPERATURE	2	Sets to Temperature Sensor.
LIGHT_ACTIVE	5	Sets to Light Sensor in active mode(LED on).
LIGHT_INACTIVE	6	Sets to Light Sensor in inactive mode(LED off).
SOUND_DB	7	Sets to Sound Sensor in dB.
SOUND_DBA	8	Sets to Sound Sensor in dBa.
ULTRASONIC	11	Sets to Ultrasonic Sensor.
Sensor Mode:		
RAWMODE	0	
BOOLEANMODE	50	

nxt_connect

Synopsis

```
#include <ch_nxt.h>
int nxt_connect(struct nxt_remote *nxt_user);
```

Purpose

Connects to the Mindstorms NXT using bluetooth.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`*nxt_user`	The address of the structure `nxt_remote` that the user created in the main function.

Description

This function connects to the Mindstorms NXT using blue tooth. The parameter `*nxt_user` is the address of the structure `nxt_remote` that stores the input/output data and port addresses. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_disconnect();
```

Output

None

---

nxt_forward

Synopsis

```
#include <ch_nxt.h>
int nxt_forward(int speed);
```

Purpose

Move the NXT vehicle forward at the specified speed.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`speed`	Speed of the motors, (range is from 1 to 100).

Description

The function is meant for use only with the NXT vehicle designed robot, and may not function correctly with other designs. The function sets the output of the motors based on the parameter `speed`. The values are based off of the speed that the individual motors of the NXT can be set to. The NXT will continue to move forward at the set speed until the motors are stopped.

Example

```
#include <ch_nxt.h>
#include <stdio.h>
#include <unistd.h>

int speed;
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_forward(speed);
sleep(5);
nxt_stop();

nxt_disconnect();
```

Output

None

---

nxt_backward

Synopsis

```
#include <ch_nxt.h>
int nxt_backward(int speed);
```

Purpose

Move the NXT vehicle backward at the specified speed.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`speed`	Speed of the motors, (range is from 1 to 100).

Description

The function is meant for use only with the NXT vehicle designed robot, and may not function correctly with other designs. The function sets the output of the motors based on the parameter `speed`. The values are based off of the speed that the individual motors of the NXT can be set to. The NXT will continue to move backward at the set speed until the motors are stopped.

Example

```
#include <ch_nxt.h>
#include <stdio.h>
#include <unistd.h>

int speed;
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_backward(speed);
sleep(5);
nxt_stop();

nxt_disconnect();
```

Output

None

---

nxt_turnleft

Synopsis

```
#include <ch_nxt.h>
int nxt_turnleft(int speed);
```

Purpose

Move the NXT vehicle forward and turn left at the specified speed.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`speed`	Speed of the motors, (range is from 1 to 100).

Description

The function is meant for use only with the NXT vehicle designed robot, and may not function correctly with other designs. The function sets the right wheel to the `speed` value and the left wheel to 0.7 * `speed` value. The motors will continue to rotate until a motor speed is changed or the motors are stopped.

Example

```
#include <ch_nxt.h>
#include <stdio.h>
#include <unistd.h>

int speed;
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_turnleft(speed);
sleep(3);
nxt_stop();

nxt_disconnect();
```

Output

None

---

nxt_turnright

Synopsis

```
#include <ch_nxt.h>
int nxt_turnright(int speed);
```

Purpose

Move the NXT vehicle forward and turn right at the specified speed.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`speed`	Speed of the motors, (range is from 1 to 100).

Description

The function is meant for use only with the NXT vehicle designed robot, and may not function correctly with other designs. The function sets the left wheel to the `speed` value and the right wheel to 0.7 * `speed` value. The motors will continue to rotate until a motor speed is changed or the motors are stopped.

Example

```
#include <ch_nxt.h>
#include <stdio.h>
#include <unistd.h>

int speed;
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_turnright(speed);
sleep(3);
nxt_stop();

nxt_disconnect();
```

Output

None

---

nxt_rotateleft

Synopsis

```
#include <ch_nxt.h>
int nxt_rotateleft(int speed);
```

Purpose

Rotate the NXT vehicle to the left in place at the specified speed.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`speed`	Speed of the motors, (range is from 1 to 100).

Description

The function is meant for use only with the NXT vehicle designed robot, and may not function correctly with other designs. The right wheel is set to the value of `speed` and the left wheel is set to negative `speed`. The NXT vehicle will continue to rotate in place until until a motor speed is changed or the motors are stopped.

Example

```
#include <ch_nxt.h>
#include <stdio.h>
#include <unistd.h>

int speed;
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_rotateleft(speed);
sleep(3);
nxt_stop();

nxt_disconnect();
```

Output

None

---

nxt_rotateright

Synopsis

```
#include <ch_nxt.h>
int nxt_rotateright(int speed);
```

Purpose

Rotate the NXT vehicle to the right in place at the specified speed.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`speed`	Speed of the motors, (range is from 1 to 100).

Description

The function is meant for use only with the NXT vehicle designed robot, and may not function correctly with other designs. The left wheel is set to the value of `speed` and the right wheel is set to negative `speed`. The NXT vehicle will continue to rotate in place until until a motor speed is changed or the motors are stopped.

Example

```
#include <ch_nxt.h>
#include <stdio.h>
#include <unistd.h>

int speed;
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_rotateright(speed);
sleep(3);
nxt_stop();

nxt_disconnect();
```

Output

None

---

nxt_stop

Synopsis

```
#include <ch_nxt.h>
int nxt_stop();
```

Purpose

Stops all motors on the Mindstorm NXT.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

None

Description

The function stops all the motors on the Mindstorm NXT. This function should not be used if you only want to stop one motor in your program.

Example

```
#include <ch_nxt.h>
#include <stdio.h>
#include <unistd.h>

int speed;
struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}

nxt_forward(speed);
sleep(3);
nxt_stop();

nxt_rotateright(speed);
sleep(3);
nxt_stop();

nxt_disconnect();
```

Output

None

---

nxt_motor

Synopsis

```
#include <ch_nxt.h>
int nxt_motor(int port, int speed, int brake);
```

Purpose

Sets the motor output for NXT.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`port`	motor port connection to NXT.
`speed`	Speed of the motor, (range is from -100 to 100).
`brake`	The mode of the motor.

Description

This function sets the motor output for NXT. The parameter `port` indicate which motor port you would like to control. The `MACRO`s available for the parameter `port` are `PORT_A`, `PORT_B`, and `PORT_C`, which refers to Port A, Port B, and Port C of the NXT brick. The speed of the motor is set by the parameter `speed`. The minimum value for `speed` is -100, which refer to run motor at full speed in reverse. The maximum value for `speed` is 100, which refer to run motor at full speed forward. If the `speed` is set to zero, then the motor will not run. The mode of the motor is set by parameter `brake`. The `MACRO`s available for parameter are `RUN_IDLE`, `RUN_BRAKE`, `OFF_IDLE`. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
int speed = 25;
nxt_motor(PORT_B, speed, RUN_IDLE);
nxt_motor(PORT_C, speed, RUN_IDLE);
//Allow the NXT to run for the next 5 seconds
Sleep(5000);
nxt_disconnect();
```

Output

None

---

nxt_motor_rotate

Synopsis

```
#include <ch_nxt.h>
int nxt_motor_rotate(int port, int speed, int degrees);
```

Purpose

Rotate the motor for NXT for given degree.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`port`	motor port connection to NXT.
`speed`	Speed of the motor, (range is from -100 to 100).
`degrees`	The degree which the motor will rotate.

Description

This function will rotate a motor at a given speed at a given degree. The parameter `port` indicate which motor should the NXT set output to. The `MACRO`s available for the parameter `port` are `PORT_A`, `PORT_B`, and `PORT_C`, which refers to Port A, Port B, and Port C of the NXT brick. The speed of the motor is set by the parameter `speed`. The minimum value for `speed` is -100, which refer to run motor at full speed in reverse. The maximum value for `speed` is 100, which refer to run motor at full speed forward. The degree which the motor will rotate is set by the parameter `degree`. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
int speed = 25;
int degree = 360;
nxt_motor_rotate(PORT_B, speed, degree);
nxt_motor_rotate(PORT_C, speed, degree);
//Allow the NXT to run for the next 5 seconds
Sleep(5000);
nxt_disconnect();
```

Output

None

---

nxt_setsensor

Synopsis

```
#include <ch_nxt.h>
int nxt_setsensor(int port, int sensortype, int sensormode)
```

Purpose

Set up the sensors to collect data for the NXT.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`port`	sensor input port connection to NXT.
`sensortype`	Type of sensor that is connected.
`sensormode`	Sensor mode.

Description

This function set up the sensors to collect data for the NXT. The parameter `port` select which port is configured within the function. The `MACRO`s available for the parameter `port` are `PORT_1`, `PORT_2`, `PORT_3`, and `PORT_4`, which refers to Port 1, Port 2, Port 3, and Port 4 of the NXT brick. The type of sensor can be configured using the parameter `sensortype`. The `MACRO`s available for parameter `sensortype` are `TOUCH`, `TEMPERATURE`, `LIGHT_ACTIVE`, `LIGHT_INACTIVE`, `SOUND_DB`, `SOUND_DBA`, and `UNLTRASONIC`. The mode for the sensor can be selected with the parameter `sensormode`. The `MACRO`s available for `sensormode` are `RAWMODE` and `BOOLEANMODE`. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
int status_1=2;
int status_2=2;

status_1=nxt_setsensor(PORT_1,TOUCH,BOOLEANMODE);
status_2=nxt_setsensor(PORT_4,ULTRASONIC,RAWMODE);
if(status_1==0){
    printf("Connection to touch sensor has failed");
}
if(status_2==0){
```

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

nxt_disconnect();
```

Output

None

---

nxt_getsensor

Synopsis

```
#include <ch_nxt.h>
nxt_getsensor(int port);
```

Purpose

Retrieve sensor data from NXT.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`port`	The sensor port that you want to gather data from.

Description

This function allow the user to retrieve sensor data from a selected port. The parameter `port` allow the user to select which sensor to gather data from. The `MACRO`s available for the parameter `port` are `PORT_1`, `PORT_2`, `PORT_3`, and `PORT_4`, which refers to Port 1, Port 2, Port 3, and Port 4 of the NXT brick. The result is stored in `NXT_sensorvalraw[port]`. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
int status_1=2;
int status_2=2;

status_1=nxt_setsensor(PORT_1,TOUCH,BOOLEANMODE);
status_2=nxt_setsensor(PORT_4,ULTRASONIC,RAWMODE);
if(status_1==0){
    printf("Connection to touch sensor has failed");
}
if(status_2==0){
    printf("Connection to ultrasonic sensor has failed");
}
//get sensor data
nxt_getsensor(PORT_1);
nxt_getsensor(PORT_4);
```

```
//store sensor data in variables
int port_1_data;
int port_4_data;
port_1_data = nxtr.NXT_sensorvalraw[PORT_1];
port_4_data = nxtr.NXT_sensorvalraw[PORT_4];

nxt_disconnect();
```

Output

None

---

nxt_printmess

Synopsis

```
#include <ch_nxt.h>
int nxt_printmess();
```

Purpose

Print the last message sent to or receive from the NXT.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

None

Description

Allow the user to print the last message sent to or receive from the NXT. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
nxt_printmess();
```

Output

None

---

nxt_disconnect

Synopsis

```
#include <ch_nxt.h>
int nxt_disconnect();
```

Purpose

Disconnect the NXT from your computer.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

None

Description

This function allow the user to disconnect the NXT from your computer. This function also turns all motors to off/idle, and turns all sensors to none. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
//Disconnect NXT
nxt_disconnect();
```

Output

None

---

nxt_gettacho

Synopsis

```
#include <ch_nxt.h>
int nxt_gettacho(int port);
```

Purpose

Retrieve the tachometer count for the motor.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

Parameter	Description
`port`	The motor port that you want to gather tachometer count from.

Description

This function allow the user to retrieve the tachometer count for the motor. The parameter `port` indicate which motor should the system gather tachometer count from. The `MACRO`s available for the parameter `port` are `PORT_A`, `PORT_B`, and `PORT_C`, which refers to Port A, Port B, and Port C of the NXT brick. The result is stored in `nxt_motor_pos_raw[port]` and `nxt_motor_pos_cum[port]`. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
int speed = 25;
int degree = 360;
int count;

//Rotate motor
nxt_motor_rotate(PORT_B, speed, degree);
//Allow the NXT to run for the next 5 seconds
Sleep(5000);
//Get tachometer count
nxt_gettacho(PORT_B);

//Store tachometer count in variable
count = nxtr.NXT_motor_pos_cum[PORT_B];

nxt_disconnect();
```

Output

None

---

nxt_motor_zero

Synopsis

```
#include <ch_nxt.h>
int nxt_motor_zero();
```

Purpose

Reset the tachometer count for all motors.

Return Value

Returns 1 if successful, returns 0 if failed.

Parameters

None

Description

This function allow the user to reset the tachometer count for all of the motor. The function will return 1 if successful and return 0 if failed.

Example

```
#include <ch_nxt.h>
#include <stdio.h>

struct nxt_remote nxtr;
if (!nxt_connect(&nxtr)){
    printf("Connection to NXT has failed.");
    exit(0);
}
int speed = 25;
int degree = 360;
int count;

//Rotate motor
nxt_motor_rotate(PORT_B, speed, degree);
//Allow the NXT to run for the next 5 seconds
Sleep(5000);
//Get tachometer count
nxt_gettacho(PORT_B);

//Store tachometer count in variable
count = nxtr.NXT_motor_pos_cum[PORT_B];

//Reset the tachometer count
nxt_motor_zero();
nxt_disconnect();
```

Output

None