RobotC Tutorial Packet I

Before you use this document:

Unless otherwise noted, Storming Robots retains an “All Rights Reserved” copyright, pursuant from the day this document was published by Storming Robots. This means that you are NOT allowed to copy them and use them on your own site or other publication without permission. This is SOLELY used for you to view, but NOT for redistribution for any purpose.

Scope of this document:

This tutorial will introduce only important basics, including fundamental programming structure, and control structure with loop and if in ANSI-C, as well as basic robotics intrinsic motor functions and variables commonly used in RobotC.

Assumptions:

This tutorial assumes:

- You know the very basic operation using NXT, such as how to select turn on/off, scroll through menu to select, etc.
- You have access to a basic NXT kit to perform all the exercises in this packet.
- You must have a workable copy of RobotC. You may download the trial version online. If you choose to purchase it, you may want to go through our website to purchase it, as it will entitle you 20% refund (off from the full retail cost) from Storming Robots.
- You should have felt somewhat comfortable to use a computer, start a program, and basic navigation in your system.

Best practice:

- Examine all samples. **DO NOT** just copy, compile and run.
- Stay curious! Try out various actions.
- Use the help panel (on the left of RobotC)
- Practice K.I.S.S. Principle. “Complicated” does not necessarily mean “cool”.
- Often elegant and efficient means simplicity.
- Write “comment” for your code

**Online Location:** robotc.stormingrobots.com
# Contents

## Chapter 1 – Basic C Fundamentals in RobotC ................................................................. 5

1.1 Using RobotC the first time .................................................................................. 5  
   1.1.1 The IDE (Integrated Development Environment) ........................................ 5

1.2 Your First Program ................................................................................................. 7  
   1.2.1 Download firmware .................................................................................. 7
   1.2.2 Write your first program ........................................................................... 8
   1.2.3 Syntax Errors ............................................................................................ 9
   1.2.4 Learn from Samples ................................................................................ 9
   1.2.5 Practice Exercise ..................................................................................... 10

1.3 Introduction to Variables ......................................................................................... 12
   1.3.1 Variable/Data Types ............................................................................... 12
   1.3.2 To display variables ............................................................................... 12
   1.3.3 Local vs. Global Variables ...................................................................... 13
   1.3.4 RobotC Intrinsic Variables .................................................................... 13

1.4 Basics in conditional / Boolean structure .............................................................. 14
   1.4.1 Logical/Boolean Operators ..................................................................... 14
   1.4.2 Logical/Boolean Expressions ................................................................ 14

1.5 Introduction to Loop Control Structure ................................................................ 16
   1.5.1 “while” loop .......................................................................................... 16
   1.5.2 “for” loop .............................................................................................. 16
   1.5.3 Introduction to Unary Operation ............................................................ 17

1.6 Introduction to Array ................................................................................................. 18

1.7 More in control structure ......................................................................................... 19
   1.7.1 Tenary Operation ................................................................................... 19
   1.7.2 switch... case... statements .................................................................. 19

1.8 To look up available commands: ............................................................................ 19

1.9 What is a Debugger ................................................................................................ 20

1.10 Learn from the samples ......................................................................................... 21  
    Samples List ....................................................................................................... 21

1.11 MIni-challenge Exercises ..................................................................................... 22

## Chapter II – More about structure design in RobotC .................................................. 24

2.1 Primitive C vs. RobotC intrinsic types ...................................................................... 24

2.2 RobotC Intrinsic Functions ...................................................................................... 25
Chapter III – Motion Navigation ................................................................. 28

3.1 Access and Modify Motors Data .............................................................. 28
  3.1.1 Move the motors with simple movements ........................................... 28
  3.1.1a Learning from the samples ................................................................. 28

3.2 Mini-challenge Exercises ...................................................................... 29

Chapter IV – Basic Sensor Operation ......................................................... 30

How to access basic nxt sensors data ......................................................... 30

4.1 Touch Sensor ......................................................................................... 31
  4.1.1 Know about your sensor ................................................................. 31
  4.1.2 Learn from the samples ................................................................. 31

4.2 Light Sensor .......................................................................................... 32
  4.2.1 Know about your sensor ................................................................. 32
  4.2.2 Practice Calculating Thresholds ....................................................... 33
  4.2.3 Learn from the samples ................................................................. 33

4.3 Ultra Sonic Sensor ................................................................................ 34
  4.3.1 Know about your sensor ................................................................. 34
  4.3.2 Learn from the samples ................................................................. 34

4.4 Motor Encoder – for a distance ............................................................. 35
  4.4.1 Know about your sensor ................................................................. 35
  4.4.2 Move with Encoder Feedback .......................................................... 35
  4.4.2a Learn from the samples ............................................................... 36
  4.4.3 Minimize Overshooting .................................................................. 36
  4.4.3a Learn from the sample codes: ...................................................... 36

4.5 Mini-challenge Exercises ...................................................................... 37

Chapter V – Applied physics – Mechanical Gear Math ............................. 39

5.1 Intro to Basic Gears Transmission .......................................................... 39

5.2 Calculating Gear Ratio ......................................................................... 40

5.3 Convert Distance Traveled to Encoder degrees ...................................... 41
  5.3.1 To find EncPerCM for Gear Ratio 1:1 .............................................. 41
  5.3.2 To find EncPerCM for Gear Ratio not 1:1 ........................................ 42
  5.4 So the Formula .................................................................................... 42

5.4 Convert Body Rotation Degrees to Encoder Degrees ............................ 43
  5.4.1 Learn from samples ........................................................................ 44

5.5 Mini-challenge Exercises ...................................................................... 44
Chapter VI – motor feedback control ......................................................... 45
  6.1  Power vs. Speeds .............................................................................. 45
  6.2  Sync motors .................................................................................. 45
  6.3  Minimize overshooting .................................................................. 46
  6.4  Learn from samples ........................................................................ 47
  6.5  Mini-challenge Exercises ............................................................... 47

chapter 7  Functions .................................................................................. 48
  7.2  Learn from samples ........................................................................ 49
  7.2  Mini-challenge Exercises .................................................................. 49

chapter 8  Buttons Control ....................................................................... 50
  8.1  Learn from samples ........................................................................ 50
  8.2  Mini-challenge Exercises .................................................................. 50
CHAPTER 1 – BASIC C FUNDAMENTALS IN ROBOTC

1.1 USING ROBOTC THE FIRST TIME

1.1.1 The IDE (Integrated Development Environment)

Using the software is quite intuitive. The following is the IDE window.

Basic operations: Here lists some commonly used menu selections:

To create a new file, e.g. hello.c

Click on "New" to create a file. This is the editor window where you will type your program in.

To Compile and Download:

In order for NXT to understand your program, you will have to:

"Compile": to ensure it is grammatically correct. (F7)

"Download": copy the compiled code into NXT (F5)
To have your robot to Execute/ Run

You will need to find and select your program to run.

If you experience program execute program, you should consult NXT operation menu.
1.2 YOUR FIRST PROGRAM

1.2.1 Download firmware

You must download firmware before you can start using your NXT controller.

![Screenshot of RoboC software showing firmware download process](image)

- **Refresh the connected list**
- **Rename your NXT**
- **Download the firmware.**
1.2.2 Write your first program

Type this program into the IDE Window.

Connect your NXT to your computer using the USB cable.

Press “F5” to “compile”, and “download.

**Entry point: task main()**

This is an entry point when a program is executed.

One program must have only a single “task main()”

A task must have its own “{“ and “}” pair.

RobotC function to display some data on the LCD screen. This one will display “Hello You!” on line 2.

You can display maximum 7 lines.

RobotC function to play a beep sound.

Tell the system to wait for 5 seconds.

Note that all functions must have its own “(“ “)” pair.

1.2.2 { … } | […] | (...) | ;

For every {, there should have its matching closing }.;

For every [, there should have its matching closing ].

For every {, there should have its matching closing }.

All expressions must end with “;”, except for Boolean expressions and loop structure. We’ll cover this in the later section.
1.2.3 Syntax Errors

What if your code generates a lot of errors after you click on “compile”? This means your code is not compliant to the grammatical rules.

Compiler is a complex pattern matcher. It looks at your code more or less like a spoken language; which imposes a strict set of grammatical rules. Once it finds mismatch of an expected pattern, it generates an error.

You will need to know how to systematically find the syntax errors. The scope of this packet will not cover the diagnostic method extensively. However, here are a few tips that you may use to help you locate the errors.

Example:

Steps:
1. To locate the error: double click on the error line. The system will locate the line which prompts the compiler to generate the error.
2. Should start with the first error.
3. Read the error message.

4. If you have missing “brace”, use the “Format Whole File” may help you to find missing ‘{‘ ‘(‘ or ‘)’.

1.2.4 Learn from Samples

- 01-Hello.c
- 02-PlaySound01.c
- 03-Display.c
- 04-Display.c
- 05-Display.c
- 06-Boolean.c
- 07-DRAW.C
1.2.5 Practice Exercise

1) Let’s have some fun drawing something on the screen.

Before you can do this, you must learn a bit about the NXT LCD screen.

- Measured 100 wide by 64 pixels high display.

Draw a line and rectangle.

a) Type the code in the window.
b) Compile and Download your code.
c) Run your code at your NXT.

Notice the ‘//’. It prefixes a line of comment. This is a great way to make your code easier to understand and for trouble shooting as well.

To comment a block, use ‘*/’ and ‘*/’.

e.g.

```c
/* program name: abc.c
   Date: such and such
   Author: Smiley Smith */
```

2) Create a rectangle with 10 pixels base, 5 pixels height.

   e.g. nxtDrawRect(10, 50, 20, 45);

3) Animate the rectangle that you created, e.g. moving it down 5 pixels at time until it disappears off the screen.

   e.g.
   ```c
   nxtDrawRect(10, 50, 20, 45);
   ```
4) Create a Smiley face.

Before you do this, you should explore other Draw functions by looking up available Draw functions, and create a circle with 30 pixels as its diameter.

a) Change yourself to Super User.

b) Select “Function Library View”

c) Click on the Display on the left. Click on Commands. A long list of RobotC display functions show up.
1.3 **INTRODUCTION TO VARIABLES**

### 1.3.1 Variable/Data Types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Sample expressions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>a data type only has two value value range: true or false</td>
<td>bool done; bool tooLong=false;</td>
</tr>
<tr>
<td>int</td>
<td>a data type can hold 2 bytes long integer value, value range: An integer value with the range from -32767 to 32767.</td>
<td>int x; int y=98;</td>
</tr>
<tr>
<td>float</td>
<td>a data type can hold 2 bytes long float value allowing decimal places value range: An decimal value with the range from -32767.0 to 32767.0.</td>
<td>float fraction; float amount = 9.7512;</td>
</tr>
<tr>
<td>string</td>
<td>a data type can hold sequence of characters. The value must be represented in “ “; e.g. string str = “Storming Robots”;</td>
<td>string name; string city=”Branchburg”</td>
</tr>
</tbody>
</table>

e.g.:

```c
int xyz;    // this means “declaring” variable xyz as an integer type.
int abc=10;  // this means “declaring” as an integer type, and
             // “initializing” the abc with the value of 10.
```

### 1.3.2 To display variables

Sample expressions:

```c
int i=10, j=20;
nxtDisplayTextLine(1, "this is robotc!"); // display at line 1 with “this is robotc”
nxtDisplayTextLine(2, "i = %d; j = %d", i , j ); // display value of “i” and “j”.
```

```
i = 10; j = 20
```

// display value of “i” and “j” with each one in a column of 4 spaces, right justified.
```
nxtDisplayTextLine(2, "i = %4d; j = %4d", i , j );
```

```
i = 10; j = 20
```

// display value of “i” and “j” with each one in a column of 4 spaces, left justified.
```
nxtDisplayTextLine(2, "i = %-4d; j = %-4d", i , j );
```

```
i = 10 ; j = 20
```
1.3.3 Local vs. Global Variables

- Understanding of ideas of “scope”

```c
===
task main()
{
    int i=4; // this is a local
    i++;
}
===
int i = 4; // this is a global

task main()
{
    i++;
    ...
}
===
task main()
{
    int i=29;
    {
        int i=4;
        nxtDisplayTextLine(2, "%d", i); // this display 4
    }
    nxtDisplayTextLine(2, "%d", i); // this display 29
}
```

1.3.4 RobotC Intrinsic Variables

There are many variables facilitated by RobotC. You should never name your own variables using RobotC intrinsic variables. The following includes two major ones.

**Program time vs System Time**

<table>
<thead>
<tr>
<th>int nPgmTime</th>
<th>int nSysTime;</th>
</tr>
</thead>
<tbody>
<tr>
<td>The lower 16-bits of the internal 1-millisecond clock.</td>
<td>The lower 16-bits of the internal 1-millisecond clock.</td>
</tr>
<tr>
<td>Contains the value of the user program elapsed time.</td>
<td>Contains the value of the system clock elapsed Time since NXT is powered on.</td>
</tr>
<tr>
<td>Is reset to 0 when user program starts running.</td>
<td>Is reset to 0 when NXT is first powered on.</td>
</tr>
<tr>
<td>Does not increment during even debugging mode.</td>
<td>Increments during even debugging mode.</td>
</tr>
</tbody>
</table>
1.4 BASICS IN CONDITIONAL / BOOLEAN STRUCTURE

1.4.1 Logical/Boolean Operators

<table>
<thead>
<tr>
<th>Basic logical operators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>== Equal (the same)</td>
</tr>
<tr>
<td>( do not use “=” as “=” means assignment)</td>
</tr>
<tr>
<td>!= Not Equal</td>
</tr>
<tr>
<td>&lt; Less than</td>
</tr>
<tr>
<td>&gt; Greater than</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>&amp;&amp; And</td>
</tr>
</tbody>
</table>

Sample Expressions Boolean variables:

```c
bool incomplete = false;
bool done = false;
```

1.4.2 Logical/Boolean Expressions

a) if ... else if ... else -----------------------------

```c
If ( boolean expressions ) {
    .... if block
}
Where <boolean expressions> will contain a single or complex expressions to be evaluated.

The <if block> means a block of codes which will be executed ONLY If the <boolean expressions> are evaluated to be true.
Sample Expressions: (code segment only)

```c
int x = 0;
    x = x + 5;
if (x == 0)
    { PlaySound(soundBeepBeep);
else if (x < 0)
    { PlaySoundFile("Woops.rso");
else
    { nxtDisplayTextLine(3, "Ha! x = %d", x);
}
```

```c
int x = 0;
    x = x + 5;
if (x > 0)
    { PlaySoundFile("soundBeepBeep");
if (x < 5)
    { nxtDisplayTextLine(3, "Ha! x less than %d", x);
else
    { nxtDisplayTextLine(3, "Well! x not = 5");
}
else
    { nxtDisplayTextLine(3, "x > 0");
}
```

**Caution!** Watch your "{" and it’s matching "}". To search for the matching "{" and "}", start with the innermost set. We often refer this to conditional block.

**More Boolean expressions samples:**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if (x == 10)</code></td>
<td>if x equal to 10</td>
</tr>
<tr>
<td><code>if (x &lt;= 10)</code></td>
<td>if x less than and equal to 10</td>
</tr>
<tr>
<td>`if (x &gt; 10</td>
<td></td>
</tr>
<tr>
<td><code>if (x &lt;= 10 &amp;&amp; y &lt;= 20)</code></td>
<td>if x &lt;= 10 and y &lt;= 20</td>
</tr>
<tr>
<td>`if (!(x &gt; 10</td>
<td></td>
</tr>
<tr>
<td>`if (!((x &lt;= 10</td>
<td></td>
</tr>
</tbody>
</table>

Notice that these statements express the same set of conditions.
1.5 INTRODUCTION TO LOOP CONTROL STRUCTURE

What is a loop: to repeat a set of instructions.

1.5.1 “while” loop

Expression format:

```c
while ( conditional expression )
{
    ... while block
    ... this may contain one to many instructions
}
```

e.g.

```c
int i;
while (i < 20)
{
    ... do something ...
    i = i+1;
}
```

Sample expressions:

```c
int i=0;
while (i < 20)
{
    nxtDisplayTextLine(i, “i =%d”, i);
    i = i+1;
}
```

Declares and initializes variable ‘i’ equal to zero
This is the conditional expression. The expressions inside the block will be executed while i < 20 is true.
increments i by one after each iteration

1.5.2 “for” loop

Expression format:

```c
for ( <initial setup> ; <conditional expression> ; <increment or decrement> )
{
    ... for block
}
```

Sample expression:

```c
int i;
for (i = 0; i < 20; i=i+1)
{
    nxtDisplayTextLine(i, “i =%d”, i);
}
```

Initializes variable ‘i’ equal to zero
This is the conditional expression. It continues to do the expressions inside the block while i < 20 is true.
increments i by one after each iteration
1.5.3 Introduction to Unary Operation

A **unary operation** is an operation with only one operand, i.e. a single input. They are evaluated before other operations containing them.

In mathematics: e.g.

\[ N! = \text{factorial of } N \]

or

\[ \sin(x), \text{ etc.} \]

In programming:

\[ ++i \quad \text{same as} \quad i = i + 1 \quad \text{same as } i +=1 \]

\[ --i \quad \text{same as} \quad i = i - 1 \quad \text{same as } i -=1 \]

\[ i /=2; \quad \text{same as} \quad i = i / 2 \]

\[ i *=2; \quad \text{same as} \quad i = i * 2 \]

\[ i %=2; \quad \text{same as} \quad i = i \% 2 \]
1.6 **INTRODUCTION TO ARRAY**

- systematic representation for a sequence of same kind of data.

Example 1:

```c
char grades[5]; // allocate 5 elements for "grades". Each grade takes up one byte (size of "char" type)
```

```
grades[0] = 'A'; grades[1] = 'B'; ... etc.
```

Example 2:

```c
int age[10]; // allocate 10 elements for "age". Each age takes up 4 bytes (size of "int" type)
```

```
age[0] = 50; grades[1] = 60; ... etc.
```

```c
short motor[3]; // allocate 5 elements for "grades". Each grade takes up one byte (size of "char" type)
```

```
```

Beware, do NOT use more than you allocate. It will crash your program or generate unpredictable results.

**Practice Exercise**

Exercises pertaining array will be provide in later chapters when you use Array for motors and sensors.

More about array data structure will be covered in Tutorial Packet II.
1.7 MORE IN CONTROL STRUCTURE

(skip this section if you are absolute beginner)

1.7.1 Tenary Operation

\[
\text{result} = (a > b \ ? \ x : y);
\]

\[
\text{if} \ (a > b) \{ \\
\quad \text{result} = x; \\
\} \\
\text{else} \{ \\
\quad \text{result} = y; \\
\}
\]

1.7.2 switch... case... statements

(skip this section if you are absolute beginner)

\[
\text{switch} \ (x) \{ \\
\quad \text{case} 1: \\
\quad \quad \text{nxtDisplayTextLine}(2, \ "x==1"); \\
\quad \quad \text{break}; \\
\quad \text{case} 2: \\
\quad \quad \text{nxtDisplayTextLine}(2, \ "x==2"); \\
\quad \quad \text{break}; \\
\quad \text{Default:} \\
\quad \quad \text{nxtDisplayTextLine}(2, \ "x==%d", x); \\
\quad \quad \text{break}; \\
\}
\]

The following structure will provide the same logic as the “switch” structure on the left.

\[
\text{if} \ (x == 1) \{ \\
\quad \text{nxtDisplayTextLine}(2, \ "x==1"); \\
\} \\
\text{else if} \ (x==2) \{ \\
\quad \text{nxtDisplayTextLine}(2, \ "x==2"); \\
\} \\
\text{else} \{ \\
\quad \text{nxtDisplayTextLine}(2, \ "x==%d", x); \\
\}
\]

1.8 TO LOOK UP AVAILABLE COMMANDS:
1.9 WHAT IS A DEBUGGER

Being able to debug a robotics program real-time is extremely helpful to locate logical errors in your code.

RobotC’s debugger feature makes RobotC more superior than most other robotics development environment available. It is not common to find a robotics IDE which also comes with a user-friendly and easy to use debugger. This debugger does not support multi-tasking though. Here is a simple overview:

- Press F5 to compile and download.
- Click on “Step into”. You will see an arrow pointing at the current execute expression.
- Look at the “Output” dialog windows at the bottom and watch the variables that you are interested to watch.
- If the output windows did not show up, click on the “Robot” tab to select the “Global Variables”.
- You may even select the “NXT Remote Screen” to view the LCD screen display as well.
- Do not use others as they are for more advanced usage.
1.10 LEARN FROM THE SAMPLES

If you have yet done all the sample codes from Chapter 1 downloaded from from http://robotc.stormingrobots.com, do it now.

IMPORTANT STEPS TO FOLLOW FOR BEGINNERS:

- If you have trouble remember the basic syntax, you should go back to read every single line of codes to understand the very basic and simple grammar/syntax.

HOW TO WORK WITH THE SAMPLES:

1. **Understand each single expression!** Read the sample code to ensure you understand every single expression
2. **Hand type the samples!** Especially for beginners, type in the sample. DO NOT use the computer “copy and paste”. Type them manually.
3. **Use your time wisely!** Compile, Run them and see result makes sense to you. If you simply copy and paste without understanding every single line of expression, you are wasting your time.
4. **Be inquisitive and Experiment!** Look into each single line of expression, make a couple of meaningful changes, then compile and download it again to your NXT.
5. **Be resourceful!** Use the “Function Library Panel” often to find available functions/commands/system variables.
6. **Use Debugger** to view how values are being changed. The debugger is truly a great self-tutorial tool.

Samples List

- o 01-Hello.c
- o 02-PlaySound01.c
- o 03-Display.c
- o 04-Display.c
- o 05-Display.c
- o 06-Boolean.c
- o 07-Draw.c
- o 08-Variable01.c
- o 09-PlaySoundWhile.c
- o 10-VariablesWhile.c
- o 11-VariablesWhile.c
- o 12-DisplaySysParm01.c
- o 12-VariablesFor.c
- o 13-PgmTime.c
1.11 Mini-Challenge Exercises

**Challenge 1:** Write a single program to do the following:
- Display your name.
- Play a sound. Wait for one second.
- Display your school name on the next line.
- Play another sound. Wait for one second.
- Display your favorite event, followed by another sound. Wait for another one second.
- End the program.

**Challenge 2:** Write a program to create a variable called “ct”. Create a loop so that your screen will display the following:

```
1
2
3
4
5
6
7
8
```

**Challenge 3:**
1) Draw a small rectangle with coordinates with top left point at (10, 20) and bottom right point at (25, 5).
   e.g. `nxtDrawRect(10,20, 25, 5);`
2) Draw a rectangle with base 10 pixels and height 5 pixels with the top left point at (10,20).
3) Draw a rectangle with base 10 pixels and height 5 pixels. This rectangle must be situated right at the center of your screen.

**Challenge 4:** Download the sample code for performing factorial, simpleFactorialError.c. There is a syntax error and one logic error. Correct the error, compile, download and execute.

**Challenge 5:** Given the radius. Generate Circumference and Area of a circle. Only 2 decimal places. Do it in one’s increment four times. E.g.
**Challenge 6:** Given the width and length of a rectangle. Generate the area of the biggest circle to fit inside this rectangle. E.g.

<table>
<thead>
<tr>
<th>R</th>
<th>Cir</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.00</td>
<td>43.96</td>
<td>314.15</td>
</tr>
<tr>
<td>14.00</td>
<td>65.94</td>
<td>153.86</td>
</tr>
<tr>
<td>21.00</td>
<td>131.95</td>
<td>1385.44</td>
</tr>
<tr>
<td>28.00</td>
<td>175.93</td>
<td>2463.01</td>
</tr>
</tbody>
</table>

**Challenge 7:** Draw the rectangle, as well as the painted complete circle fit inside the rectangle.

**Challenge 8:** Given the measurement of base and height of a right angle triangle, find out the measurement of the hypotenuse of the triangle. *(Pythagorean Theorem)*

<table>
<thead>
<tr>
<th>B</th>
<th>H</th>
<th>Hyp</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>60</td>
<td>1962.50</td>
</tr>
</tbody>
</table>

---

tutorial samples: robotc.stormingrobots.co  
www.stormingrobots.com  
Tel: (908) 595-1010
CHAPTER II – MORE ABOUT STRUCTURE DESIGN IN ROBOTC

Steps:

- Review each section.
- After you have read the section, Download the sample codes as recommended in each section.
- Always review and understand the purpose of the sample program. DO NOT just copy and paste.
- Compile, and test them with the controller to see if it runs as you have expected.
- For chapter II, you only need to work with the controller (the programming brick along), without building the robot.

2.1 PRIMITIVE C VS. ROBOTC INTRINSIC TYPES

Primitive C types

- int : integer (2 bytes)
- short : short integer (1 byte)
- float : number with decimal places (2 bytes)
- bool : true or false  (1 byte)
- char : a single character field (1 byte)

RobotC specific data types

- RobotC provides a rich set of its own data types. The most common data types:
  - tMotor : motor type
  - tSensors : sensor type
  - string : maximum 17 characters field

Others

There are many others pertaining to:

- File I/O Access (will be covered in RobotC Tutorial Packet II)
- Bluetooth access (will be covered in Bluetooth Tutorial Packet)
- LCD Graphics types (will not be covered in our tutorial. However, you may review RobotC samples codes. )
- Many others (not covered as they are for platforms other than NXT)
2.2. **ROBOTC INTRINSIC FUNCTIONS**

RobotC provide a rich set of robotics functions.

To view them all:

```
Functions template:

<Data type>  <function name> ( parameters )
{   Instructions...
}

e.g.  void wait1Msec (5000)
    wait1Msec : is the function name
    5000 : parameter
    void : return data type is void. This means it does not return anything.

e.g.  void nxtDrawRect (10, 20, 35, 6)
    nxtDrawRect : is the function name
    10, 20, 35, 6: four parameters which are all "int" type
    void : return data type is void. This means it does not return anything.

e.g.  int strcmp("aaaa", "aaab")
    strcmp : is the function name
    "aaaa", "aaab": two string parameters
    int : return data type is int. This means it returns integer value
          which equals value difference between "aaaa" and "aaab".
```

In this section, you will do more practice to reinforce your basic programming skill before using the actual motor navigation and sensors feedback.
2.3 **Writing Pseudo-Code**

Pseudo Code is:

- natural language constructs modeled to look like statements available in many programming languages
- design a sequence of instructions to perform some task(s)
- should always do this before you start programming
- e.g. calculating factorial for N!

```plaintext
set result to 1
for I from N to 1
    result = result * I
```

- e.g. Euclidean algorithm to calculate G.C.D. for number N, & M

```plaintext
if M > N
    swap M and N
while N != 0
    remainder = N % M
    M = remainder
    N = M
end while
Answer = M
```

Why is it important?

- Demonstrate the detailed steps to perform an algorithm.
- Break down the abstract of an algorithm
- Allow us to focus on developing the algorithm without worrying about the actual coding semantics.
- Is very useful for error tracking, trouble-shooting, etc.
- Help to modularize or dissemble a complex task into small tasks

2.4 **Learn from Samples**

This exercises are only for those who have the proper math level skill and want to take on more programming challenge. You may skip this.

Remember the Best Practice rules thumb stated at the beginning of chapter 1.

- 01Factorial.c: change the factorial number and run it again
- 02FactorialError.c: find the error and correct it.
- 03Triangle.c: practice you loop structure. Involving nested loops. Need design skill.
- 04DrawSpiral.c: draw spiral. This requires knowledge in trigonometry, design skill, and loops structure.
2.5 **MINI-CHALLENGE EXERCISES**

Note: If you have absolutely no idea how to get started with the following projects, you should do the following:

- go back the samples, study the codes
- be inquisitive and make relevant changes
- do not be afraid of errors
- step through the codes with the debugger window on.
- Look at the variables value and see they behave as you expected.

**Challenge 1: Change the sound and delay.**

a) Modify `p1PlaySimpleSound.c` to play Beep sound 5 times with 1 beep in-between.
b) Modify the `p1Display02.c` to display the following using variable like this:

```
<table>
<thead>
<tr>
<th>I</th>
<th>I+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
</tbody>
</table>
```
c) Modify `p1Display02.c` using loop.

**Hint:**
```
Display... (num+2, "%d %d", num, num*num);
```

**Challenge 2: Write a program to calculate factorial starting with value 3 and display like the following:**

<table>
<thead>
<tr>
<th>At program starts up</th>
<th>After 1 second</th>
<th>After 1 second</th>
<th>After 1 second</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X!</td>
<td>X!</td>
<td>X!</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
</tbody>
</table>

Pseudo code:

```
set x to 1
while (x <= 6)
    perform X! (i.e. factorial of ‘X’)
    update the display with both value of X and X! (factorial of X)
wait 1 sec
```

**Challenge 3: Write the Euclidean algorithm**

- to calculate G.C.D. for number 36 & 96 based on the pseudo code provided in section 2.3.
CHAPTER III – MOTION NAVIGATION

3.1 ACCESS AND MODIFY MOTORS DATA

3.1.1 Move the motors with simple movements

NXT controller supports 3 motor ports.

RobotC variables for motor ports: motorA, motorB, motorC
RobotC data type: tMotor
Valid data range: -100 to 100 (reverse to forward direction of -100% to 100% power level)
0 == stop

For example:
    motor[motorA] = 50; // start motor A with 50% power level

3.1.1a Learning from the samples

(May get help from the Chapter 3 samples downloaded from the online tutorial page.)

- Simple movements samples p1SimpleMovements.c: forward & backward
- p1SimpleTurns.c:
  - point turn: 90° left and 90° right
  - drag turn: 90° left and 90° right
- Repeat movements:
  - p1MotorWVar01.c
  - p1MotorWVar02.c
3.2 **Mini-Challenge Exercises**

1) Write a program to make your robot run in a square using Wait1Msec(...).

2) Write a program to wiggle left and right for 5 times and back to the starting position.

Pseudo code:

```
    tm = 500
    Left for tm milliseconds
    While ct < 5
        right for tm*2 msec
        left for tm*2 msec
        ct + 1
    left for tm sec
```

3) Program your robot to run to a target distance as fast as possible, but be able to stop almost right at the target without overshooting. Set target to be 50cm, then try 100cm, then try 200cm.

Note: You may wonder what encoder value should be used for 50cm, or 100cm, or 200cm. For now, you will just use any intuitive number(s) with trial and errors method. You will learn how to perform the gear math to convert the distance to encoder value.
CHAPTER IV – BASIC SENSOR OPERATION

HOW TO ACCESS BASIC NXT SENSORS DATA

There are three things you should always start with:

- Know which sensor port you are using
- Know RobotC data type for sensors
- Know valid data range

Commonly used RobotC intrinsic:

<table>
<thead>
<tr>
<th>Sensor ports:</th>
<th>S1, S2, S3, S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RobotC data type:</td>
<td>tSensors</td>
</tr>
<tr>
<td>Valid data range:</td>
<td>Various sensors have various valid range</td>
</tr>
</tbody>
</table>

Steps that you should do in order to get the data from the basic sensors:

**1st step:** Set the sensor type.

This allows the RobotC to generate normalized (or commonly known) value of a sensor.

You must use a value sensor type provided by RobotC. Some commonly used RobotC “sensors type” constants:

```c
sensorTouch // for both RCX & NXT touch sensors
sensorReflection // for RCX light sensors
sensorRotation // for RCX rotation sensors
sensorLightActive // NXT light sensor only; sensor LED light on
sensorLightInactive // NXT light sensor only; sensor LED light off
sensorI2CCustomFast // for other I2Cbase sensors (will be covered in advanced Packets)
```

**2nd step:** Access/Get the value.

In the following sections, you will learn how to program to utilize feedback from the basic NXT sensors available from the NXT package. They are:

- Touch sensor
- Light sensor
- Ultrasonic sensor

How about the rotation sensor?

Yes, it comes with the rotation sensor, but it is embedded inside each motor. We should call it as Encoder. This has been covered in the previous section.
4.1 TOUCH SENSOR

4.1.1 Know about your sensor

Sensor Type: sensorTouch

Valid data range: 0 means released 1 means pushed in

Examples:

Sample 1:

```c
SensorType[S1] = sensorTouch;
While (SensorValue[S1]==0)
{
    nxtDisplayTextLine(4, "Released!");
}
nxtDisplayTextLine(4, "Pushed!");
```

Sample 2:

```c
const tSensor leftBumper = S1;
int value=0;

SensorType[leftBumper] = sensorTouch;
while (value==0)
{
    nxtDisplayTextLine(4, "Released!");
    value = SensorValue[leftBumper];
}
nxtDisplayTextLine(4, "Pushed!");
```

Sample 3:

```c
while ( (value=SensorValue[leftBumper]) == 0)
nxtDisplayTextLine(4, "Released!");
nxtDisplayTextLine(4, "Pushed!");
```

4.1.2 Learn from the samples

Again, you should review the code to understand the actions.

The following samples do not use the motors. It shows how you may use the touch sensors. Run the two programs and pay attention the difference and find out why.

- p1Touch01.c
- p1Touch02.c

The following samples require motors:

- p1Bumper01.c
- p1Bumper02.c
4.2 LIGHT SENSOR

4.2.1 Know about your sensor

The light sensor measures the amount of light that it sees. It reports the amount of “reflection” represented by a number between 0 (total darkness) and 100 (very bright).

The light sensor uses its own light source, a red light emitting diode (LED), to illuminate a small area in front of its receiver, called Photo-transistor.

Sensor Type: sensorLightActive

Valid data range: 0 <= value <= 100; 0==darkest and 100==brightest

Examples:

1) If S1 sees darker than
   e.g.
   if (SensorValue[S1] < 43)
     { ...; }

2) while both S1 and S2 see brighter than
   e.g.
   while (SensorValue[S1] > 43 && SensorValue[S2] > 45)
     { ...; }

3) If see brighter than
   e.g.
   if (SensorValue[S1] > 43)
     { ...; }

4) while see brighter than
   e.g.
   if (SensorValue[S1] > 43)
     { ...; }

p1Bumper03.c : Pay attention the number count.
4.2.2 Practice Calculating Thresholds

Before you program your light sensor, you must find out the proper threshold. One simple way is to take the average of the darkest and the brightest.

| value of black: ______ | value of white: ______ | average : \[_______\] |

Think About this:

The light value can easily be affected by the lighting around the robot.

Try the following 3 things:

1) Read the light value of the black tape in well-lit classroom
2) Turn off the light, read the light value of the same black area.
3) Take your robot and this paper outside under the sun, read the light value of the same black area again.

One simple rule of thumb using light sensor is to locate your sensor in such a way that:

- Light source is consistent
- Distance from it to your target surface consistent.

4.2.3 Learn from the samples


- p1GoTilDark.c
- p1LineTrack01.c: Line Tracking with one light sensor
- p1LineTrack02.c: Line Tracking until it bumps into something (with a touch sensor bumper)
- p1LineTrack01wTimer.c : line tracking for a period of time
- p1LineTrackWObstacle01.c: line tracking until hitting an obstacle
- p1LineTrackWObstacle02.c: line tracking until hitting an obstacle using macros to represent motors
4.3 ULTRA SONIC SENSOR

4.3.1 Know about your sensor

It uses sound propagation technology. The Ultrasonic Sensor uses the same scientific principle as bats: it measures distance by calculating the time it takes for a sound wave to hit an object and come back – just like an echo.

The Ultrasonic Sensor provide feedback in either centimeters or inches. It is able to measure distances from 0 to 2.5 meters with error margin of +/- 3 cm.

Large-sized objects with hard surfaces provide the best readings. Objects made from soft fabrics, such as carpet, or from curved objects (e.g. a ball), or from very thin and small objects can be difficult for the sensor to produce reading reasonable to us.

The sonar can only detect range from 7 to 50 reliably. When it sees objects 50cm+ away, it most likely will return 255. That means it cannot detect anything. Some claim it can read up to 45cm. Honestly, I have not had such luck.

Actual distance: Look at the following diagram, the distance is closer to the length of the hypotenuse.

![Ultrasonic Sensor Diagram]

Valid data range: 0<=value<=50 ; 255 for anything farther.

Note that battery level may affect the reliability, even within 50. We have had experienced that I could not read up to 35 when the battery is running low.

Sensor Type: SensorSONAR

Special Notes:

if SensorValue[S1] == 12

where S1 is the US port. It means the object most likely is <12cm away from your robot, i.e. ~= the hypotenuse.

Caution: Sound wave will not work on surface which absorbs sound waves, such as carpet.

4.3.2 Learn from the samples

- p1Sonar01.c: beep if sees something within 10cm.
- p1Sonar02.c: go forward until 7 cm close to an object
- p1Sonar03.c: whenever it sees object within 7cm, turn 90° left. Do this 4 times.
4.4 MOTOR ENCODER – FOR A DISTANCE

4.4.1 Know about your sensor

**Valid data range:** \(-32767 \leq \text{value} \leq 32767\) (can be in centimeters or Inches)

**Sensor Type:** N/A

The NXT motor has a built in tachometer that keeps track of the current angle (in degrees) of the motor axle. Or, people usually called it rotation sensor or encoder. The terms are interchangeable.

Tachometer is the technical mechanic name.

RobotC uses a variable **nMotorEncoder** to hold the rotation degree for the motor.

e.g.

```c
nMotorEncoder[motorA] = 0; // reset encoder for motor A
nMotorEncoder[motorB] = 0; // reset encoder for motor B

if(nMotorEncoder[motorA] < 1800)
{
    ... // if encoder rotates < 1800, do step(s) in the "{
    ...
}
while(nMotorEncoder[motorA] < 1800) // while encoder A rotates < 1800,
{
    ... // if encoder rotates < 1800, do step(s) in the "{
    Go back to checking if encoder still rotates < 1800
}

while(nMotorEncoder[motorA] < 1800 || nMotorEncoder[motorB] > -1800) // while encoder A < 1800 or encoder B > -1800
{
    ... // if the condition is true, do step(s) in the "{
    Go back to check whether the condition still true
}
```

**Important:** The degree is the true degree of rotation of the embedded encoder (rotation sensing) inside the motor, not the robot physical body itself.

4.4.2 Move with Encoder Feedback

This will allow you to tell how many degrees your motor turns. This will be covered in the next chapter when we cover sensors.

RobotC intrinsics variables: nMotorEncoder[ ... ]

e.g. nMotorEncoder[motorA] = 0; // reset the rotational degrees

```c
if ( nMotorEncoder[motorA] < 360) // if the motor A rotates less than 360 degrees
```
4.4.2a Learn from the samples

- p1SimpleEncoder.c
- p1GoForwardWEnc.c: catch the display of the “target” encoder vs. the “actual” encoder after full stop
- p1GoLeftWEnc01.c
- p1GoLeftWEnc02.c: Does the same thing like p1GoLeftWEnc01.c. Does one perform better than another one?

4.4.3 Minimize Overshooting

This will allow you to define the target encoder values and minimize overshooting.

RobotC intrinsic: nMotorEncoderTarget[ ... ]

e.g.

```c
nMotorEncoderTarget[motorA] = 360;  // preset the target encoder value to 360
// however, this does not start the motor
motor[motorA] = 50;                        // make sure you will start the motor
// NXT will decelerate and stop your robot for you robot
// right about at 360 encoder value
```

You need the capability to check the status of the motor, i.e., stops already or still running. You need to use:

```c
nMotorRunState[ ... ]
```

and

```c
runStateIdle
```

e.g.

```c
if ( nMotorRunState[motorA] == runStateIdle )  // if motor A stops
if ( nMotorRunState[motorA] != runStateIdle )   // if motor A is still moving
```

4.4.3a Learn from the sample codes:

- p1SimpleMotorTarget.c
- p1WatchTheEncoder.c: Watch the encoder value
- p1MoveWTargetEnc.c: you can change the target encoder value and check.
4.5 **Mini-Challenge Exercises**

**Challenge 1:**
- `goBackWEnc.c` : run backward for any specific distance based on the conversion using target control.
- `goRightWEnc.c` : make right turn for any specific distance based on the conversion using target control.

**Challenge 2:**
- modify all the forward/backward/right/left into individual functions and integrated into a single program.
- `runSquare.c` : program your robot to run in a square using the 4 functions stated above. Use conversion routines
- rewrite the forward/backward functions to a single `goStraight` function which can do either forward or backward
- rewrite the right/left turn functions to a single turn function which can do either right or left turn.

**Challenge 3:**
Line tracking until see an obstacle. Then, it should get it and get back on the line behind the obstacle.

Pseudo Code:

Modularize your tasks!

Step 1) Write a high level analysis:

1. While sonar sees no obstacle (say >15cm)
2. { Line trace
3. }
4. Back up a few cm
5. While sonar < 30
6. { Slowly turn left
7. }
8. Stop
9. Turn left more
   (Depends on your robot side. This step is to allow enough room for your robot go by the obstacle)
10. While light sensor does not see dark
11. { Get around the object
12. }
13. Stop

Step 2) Broke down the large task into small ones. You may not need to do this for simple steps like “Turn left more”. However, you definitely should have one for the “Get around the object”. Here is one example:
While light sensor does not see dark
  • If sonar > X cm (depending on your robot size)
    • Shift right
      (such as: Left motor 60%
             Right motor 25%)
  • Else
    • Shift left
      (such as: Right motor 60%
             Left motor 25%)

Note: The power levels may very likely change depending on your gears system, tire, chassis sides, etc.

The purpose of tracing the object instead of "dead reckoning" because you would never know the size of the object. However, just for simplicity sake, you can use a fixed sized object.
CHAPTER V – APPLIED PHYSICS – MECHANICAL GEAR MATH

5.1 INTRO TO BASIC GEARS TRANSMISSION

Basics Terms
A gear is a set of toothed wheels (gear wheels or cog-wheels) that work together to transmit movement.

Many of the everyday mechanisms and devices we commonly use contain gear wheels. These include bicycles, cars, and can-openers.

Functions of gears:
1. To change the position of a rotating movement. (This is sometimes called applying the rotation at a distance.)
2. To change the direction of rotation.
3. To increase or decrease speed of rotation.
4. To increase turning force (This is sometimes called torque.)

Driver/Input
The name for a gear wheel that is turned by an outside force (such as that from a motor or from a person turning a handle) and that also turns at least one other gear wheel.

Driven/Follower/Output
The name for a gear wheel that is turned by another gear wheel.

Gear Ratio
A proportion used to compare how two meshed gear wheels move relative to each other. For gears, use the number of teeth for calculation. For pulleys, use its diameter for calculation.

Gearing Down
An arrangement in which a small driver turns a large follower, resulting in a slowing down of the turning. Gearing down produces a powerful turning force (torque).

Gearing Up
As arrangement in which a large driver turns a small follower, resulting in a speeding up of the turning. Gearing up reduces the turning force.

Idler Gear
The name for a gear wheel that is meshed between a driver and a follower. It does not mean it does not move. It is called idler gear because it does not affect the final gear ratio.

Warm up information
- Two meshed gear wheels turn in opposite directions.
When two gear wheels are mounted on the same axle, they both turn at the same speed, regardless of their sizes.

- Gears have a trade-off with turning force (torque) and turning speed.
- In general, \( \text{torque} \uparrow \text{speed} \downarrow \) when \( \text{torque} \downarrow \text{speed} \uparrow \)

## 5.2 Calculating Gear Ratio

Torque (twisting or turning force) is inversely proportional to speed.

In order to determine both the speed and force of rotating axles, we need to calculate the Gear Ratio.

The gear ratio is the ratio of the number of teeth on each gear. Here is a gear with 8 teeth meshed with a gear with 40 teeth.

What does this gear ratio \( \frac{5}{1} \) tell us?

- The Input-driver gear will rotate 5X when the output-follower gear rotates 1X
- The Input-driver gear will rotate 5X faster than the output-follower gear.
- This contraption is meant to increase torque
5.3 CONVERT DISTANCE TRAVELED TO ENCODER DEGREES

In order to convert a target distance, e.g 10cm, to a required encoder value, you need to know:

“How many encoder degrees to run 1cm?”. Let call it EncPerCM.

5.3.1 To find EncPerCM for Gear Ratio 1:1

step 1: Find the tire’s circumference:

Let \( \text{tireD} = \) diameter of the tire in centimeter

One tire revolution = tireD \( \times \) \( \pi \) cm

Figure 1

step 2: Find the Gear Ratio (GR)

Gear Ratio = \( \frac{\text{follower gear}}{\text{driver gear}} \)

See the sample setup on the right:

GR = 1 : 1 or 1/1

Step 3: Find encoder degrees per 1 unit of distance

Say, we are interested in “cm”, centimeter. Thus, you need find out:

1 cm = ? encoder degrees. Let’s call it EncDegreesPerCM

\[ \therefore \text{One tire revolution} = \text{tireD} \times \pi \text{ cm} \]
\[ = 360^\circ \]

\[ \therefore \text{tireD} \times \pi \text{ cm} = 360^\circ \]

\[ 1 \text{ cm} = \frac{360^\circ}{\text{tireD} \times \pi} \]

so in your program, you need to create a variable to hold this value. For example:

```c
float tireD = 8; \// if the tire diameter = 8 cm
int EncDegreesPerCM = 360 / (tireD * PI);
```

So, In order to travel 20 cm, total encoder value = \( 20 \times \text{EncDegreePerCM} \);
5.3.2 To find EncPerCM for Gear Ratio not 1:1

Motor rotates 2X == Tire rotates 1X
GR = 2/1
∴ 360° * GR = tireD \pi cm

EncDegreesPerCM = 360° * GR / (tireD\pi)

∴ In order to travel 20 cm, rotational degrees = 20 * 360° * 2 / \pi

5.4.3 So the Formula

The formula to convert distance traveled to encoder value (rotational degrees in this case):

Postulate 1: Encoder pre CM = 360 * GearRatio / (tireD\pi)

∴ To travel \( x \) cm, motor requires to turn \( x * 360 \times \text{GearRatio} / (\text{tireD}\pi) \) encoder degrees
5.4 Convert Body Rotation Degrees to Encoder Degrees

Remember: It is all about "distance" it needs to travel, no matter what shape the travel pattern is.

Therefore, the 1st step is to find out the distance for Chassis Turning.

Steps:

1) Find out the traveled distance:
   a) Measure the wheel base (W).
   b) Calculate the distance traveled by one full chassis revolution, circumference with the wheel base as the diameter of the circle it travels.
      \[ W \times \pi \]  (let’s create a variable called, FullTurnCM)
   c) Get the fraction of the body turns out of a 360 full body rotation, e.g. \( 90^\circ = \frac{1}{4} \) of 360°
      - i.e. \( \text{fraction} = \text{target degree of turn} / 360 \)
   d) FullTurnCM * fraction

2) Calculate Encoder degrees per cm, (let’s call it EncDegreesPerCM).
   - \[ 360 * \text{GR} / (\text{tireD} * \pi) \]

3) Total degrees (encoder value) needed to travel for 90 degrees
   = FullTurnCM * fraction * EncDegreesPerCM

You can easily generate another formula:

One full body point turn rotation = \( W \times \pi \) cm (refer to figure 3)

\[
\therefore 1 \text{ cm} = 360 * \text{GearRatio}/ (\text{tireD} * \pi) \quad \text{(from postulate 1)}
\]

\[
\text{Full body point turn rotation} = W * \pi \text{ cm}
\]

\[
\therefore = W * \pi * 360 * \text{GearRatio} / (\text{tireD} * \pi)
\]

\[
= W * 360 * \text{GearRatio} / \text{tireD}
\]

1 body point turn rotation = \( (W * 360 * \text{GearRatio} / \text{tireD}) / 360 \)

\[
= W * \text{GearRatio} / \text{tireD}
\]

The formula to convert body rotation to encoder value:

Postulate 2: \( 1^\circ \) of body point turn rotation = \( W * \text{GearRatio} / \text{tireD} \) encoder degrees
So for \( X^\circ = X * W / (D * \text{GearRatio}) \) encoder degrees

\[
\therefore \text{ To turn } X^\circ \text{ turn left turn:}
\]

- Right motor requires to turn \( X * W * \text{GearRatio} / \text{tireD} \) encoder degrees
- Left motor requires to turn \( -1 * X * W * \text{GearRatio} / \text{tireD} \) encoder degrees
5.4.1 Learn from samples

- p1GoForwardWMath.c
- p1GoLeftWMath.c

5.5 MINI-CHALLENGE EXERCISES

Now, write 2 functions:

- goBackwardWMath()
- goRightWMath()

Hints:

- You may write separate programs or functions within a single program.
- Make a function to be capable of handle both going forward and backward
CHAPTER VI – MOTOR FEEDBACK CONTROL

6.1 POWER VS. SPEEDS

You can set the Motor power with motor[], not speeds.

Not all motors are created equal. Various factors in the robot’s construction, and the manufacturing process for the motors themselves cause different amounts of energy to be lost to friction in each motor.

This means that even though both motors start with the same power at the plug, the amount of power that reaches the wheel to move the robot can vary quite a bit. Even with the same power being applied, speeds may differ.

In order to make two motors run the same speeds, you need to do some work, that is called motor feedback control.

6.2 SYNC MOTORS

be able to travel a straight line. This is because both motors are not rotating at exactly the same speed at all time. It requires a motor feedback controller process to make this happen.

RobotC provides such feature to synchronize the power levels between the two motors.

Important note:

- Once you activate the feedback control, you must not modify the secondary motor.
- Do set master motor to “100” power level. If it is already 100, the sync process will not be able to give the secondary motor higher power level.

Steps:

1. Reset both motor encoder values
2. To activate the feedback control:

   e.g. nSyncedMotors = synchAC; // Activate the feedback control.
   // Motor ‘A’ is the master, ‘C’ is the secondary.
3. To set “nSyncedTurnRatio”:

```c
nSyncedTurnRatio=100;
// means secondary motor power level = 100% of master motor power level;
// i.e. go forward
nSyncedTurnRatio=-100;
// means secondary motor power level =-100% of master motor power level;
// i.e. spin around
```

4. Start your motor for a specific # of encoder value.
   - **Important:** by activating the feedback control, the process assumes duty to adjust the power of the secondary motor. Therefore, you MUST not make any modification to the secondary encoder, nor its direction.

5. Stop ONLY the master motor.
6. To deactivate:

   ```c
   nSyncedMotors = synchNone; // No motor synchronization
   ```

### 6.3 Minimize Overshooting

This has been briefly covered in earlier section. Let’s review it further.

By now, you should have noticed that there is always a certain amount of overshooting at the complete stop. In order to minimize this, you will need to perform the proper deceleration by adjusting the power based on the remaining distance and a few features about the mechanic aspects of the motor.

This is part of the motor feedback control process.

In mechanical engineering, you will come across a topics called “Proportion, Integral, and Derivative – P.I.D.”. Put in a much simpler form, P. is adjustment based on current data, I. is for adjustment based on past data, and D. is adjustment based on prediction.

Again, RobotC provides this feature for you with intrinsic variable: `nMotorEncoderTarget[<motor>]`.

Note that you do not need to stop the motor. The internal feedback control process will stop it for you at the specific encoder value you specified.

```c
e.g. nMotorEncoderTarget[motorA] = 1000; // move 1000 encoder counts and stop
motor[motorA] = 75; // start your motor
// motor[motorA] = 0; is not necessary.
```

**Important note:**

- `nMotorEncoderTarget[... ]` does not start the motor.
- The controller decreases the power level a lot at the end of the few degrees of rotation. If your battery level is not at its optimum condition, e.g. 7.3V+, the motor may seem to stop before it reaches the target.
If battery level is low...

Refer to the sample on the right:
Motor seems to stop at, say, 998. Thus, if you wait until it finishes to get done, it won’t; and your robot will appear to be stuck.

<table>
<thead>
<tr>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>nMotorEncoderTarget[motorA] = 1000; while (nMotorEncoder[motorA]&lt; 1000) { motor[motorA] = 50; } PlaySound(BeepBeep); // even the robot stops, it never beeps because the process is stuck at the conditional expression.</td>
</tr>
</tbody>
</table>

Workaround to avoid being stuck.

<table>
<thead>
<tr>
<th>Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>nMotorEncoderTarget[motorA] = 1000; motor[motorA] = 75; if ( nMotorRunState[motorA]==runStateIdle</td>
</tr>
</tbody>
</table>

### 6.4 Learn from Samples

- goForwardWSync.c
- goForwardWSyncTargetControl.c
- goLeftWSync.c
- goLeftWSyncTargetControl.c

### 6.5 Mini-Challenge Exercises

Now, write 2 functions:

- goBackWSyncTargetControl.c
- goRightWSyncTargetControl.c

**Challenge 1:**

- goBackWEnc.c : run backward for any specific distance based on the conversion using target control.
- goRightWEnc.c : make right turn for any specific distance based on the conversion using target control.
A function is a segment of codes which encapsulate a set of actions. Its characteristics:

- Reusable
- Self-contained, if designed properly
- Perform a coherent set of actions for achieving a particular functionality
- May take in variables / arguments which go through the same of instructions
- May return a specific value or none.

Structure of a function:

A function takes in 2 arguments, and return a value. In this example, it takes in two individual “int” and return another “int”. Note that you can any data types, such char, byte, string, etc.

```c
int largerNum( int x, int y)
{
    if (x >y)
        return x;
    return y;
}
```

A function runs the robot forward until it sees dark and stop.

```c
void goUntilSeeDark()
{
    while(SensorValue[S1]>35)
    { // bot on white
        motor[motorC] = 50;
        motor[motorB] = 50;
    }
    motor[motorC] = 0;
    motor[motorB] = 0;
}
```

For example: to calculate factorial of a number:

```c
int factorial(int number)
{   int i, result=1;
    for (i=number; i>1 ; i--)
    {    result = result * i;
    }
    return result;
}
```
7.2 LEARN FROM SAMPLES

- p1SimpleFunc.c
- p1factorialFunc.c

7.2 MINI-CHALLENGE EXERCISES

1. Modify the 4 sample projects in Ch 6.4 to become 4 separate functions in a single program.

   Sample skeleton code:

   ```c
   void goForward(int distInCM )
   {
       ...
   }

   void goBackward(int distInCM )
   {
       ...
   }

   void goLeft(float chassisDegrees)
   {
       ...
   }

   void goRight(float chassisDegrees)
   {
       ...
   }

   task main()
   {
       ...
   }
   ```

2. Create a “goStraight(int distance)” function which can perform either forward/backward.

3. Create a “goTurn(int degrees)” function which can perform either right or left turn.

4. Write runSquare.c : program your robot to run in a square using the 2 functions done in (2) and (3) stated above. Note: For accurate navigation, you should use do the gear math conversion, as well as motor feedback control introduced in chapter 6.
CHAPTER 8  BUTTONS CONTROL

You can hijack the buttons for specific purpose.

Intrinsic variables:

\( nNxtButtonPressed \) can be one of the following values:

- \( kNoButtons = \) nothing is pressed
- \( kLeftButton = \) left button is pressed
- \( kRightButton = \) right button is pressed
- \( kEnterButton = \) enter button is pressed
- \( kExitButton = \) exit button is pressed

8.1 LEARN FROM SAMPLES

▷ p1Button01.c
▷ p1Button02.c

8.2 MINI-CHALLENGE EXERCISES

**CHALLENGE 1:** Change the sound and delay.

a) Modify 09-PlaySoundWhile.c from chapter 1 to play Upward Tone when right button is pressed, but downward Tone when left button is pressed.

b) Modify the 03-Display.c from chapter 1 to display the following using variable like this:

| 1 | I
|---|---
| 2 | I*I
| 3 | 4
| 4 | 9
| 5 | 16

For this exercise, the next number shows only when the right button is pressed. Program ends after the sixth times when a button is pressed.

**CHALLENGE 2:**

Play "Do Re Mi Fa So La Ti " using the buttons... Right buttons go from Do to Re, from Re to Mi, etc.

Left button goes backward, such as Ti to La, La to So, etc.

✈ The End of Packet I ✈