Graphical RobotC Tutorial Packet

Computational Thinking and Engineering for Gr. 4 to 12

Before you Start .................................................................................................................. 4
  General Guidance to follow .......................................................................................... 4
Ch.1 – A Brief Tour of the main Screen .......................................................................... 6
  1.1) Robot Type ............................................................................................................. 6
  1.2) Basics on the tool bar ............................................................................................ 6
    a) MOTOR AND SENSORS CONFIGURATION SETUP ............................................ 6
    b) FIRMWARE DOWNLOAD ...................................................................................... 7
    c) SAVE FILE ............................................................................................................. 7
    d) COMPILE PROGRAM .......................................................................................... 7
    e) DOWNLOAD TO ROBOT ....................................................................................... 7
  1.3) Program Menu ....................................................................................................... 8
Ch.2 – Be Resourceful & Exercise Best Practice .............................................................. 9
  Utilize the Help Feature ............................................................................................... 9
  Learn from Samples ...................................................................................................... 9
  Proper steps to follow when you are tackling a challenge or project ......................... 10
  Guidelines for the most efficient coding: ..................................................................... 10
  Process of Trouble-shooting Hint: ................................................................................ 10
Ch.3 – Start Thinking Like Robot With Flowchart .......................................................... 11
  3.1) What is a flowchart? .............................................................................................. 11
  3.2) Complex to Simple and Basic Behavior ............................................................... 11

Written by: Elizabeth Mabrey, Director
Erin Young, Instructor
Last update: Nov 1st, 2017
3.3) Divide and Conquer ................................................................. 12
3.3) Basic Terms: ........................................................................... 13
3.4) Control Structures with Flowchart ........................................... 14
  3.4.1) IF-CONTROL STRUCTURE: .................................................. 14
  3.4.2) LOOP (REPEATED-NESS) CONTROL STRUCTURE ............. 17
  3.4.3) NESTED CONTROL STRUCTURE ......................................... 20
3.5) Recapture the steps: ............................................................... 24

Ch.4 - Basic Motors Operations ................................................... 25
  4.1) Learn about motors ............................................................... 25
      VALID VALUES ........................................................................... 25
      PROGRAMMING EXPRESSION ................................................. 25
  4.2) Learn from Samples ............................................................. 26
  4.3) Mini-Challenge ..................................................................... 27

Ch.5 – Work with Touch Sensor ................................................... 28
  5.1) Learn about the sensor ........................................................ 28
      VALID SENSOR VALUE ......................................................... 28
      PROGRAMMING EXPRESSION ................................................. 28
  5.2) Learn from Samples ............................................................. 28
  5.3) Mini-Challenge ..................................................................... 32

Ch.6 – Work with Light/Color Sensor ........................................... 33
  6.1) Learn about the sensor ........................................................ 33
      VALID SENSOR VALUE ......................................................... 33
      TO GET THE LIGHT SENSOR FEEDBACK FROM LIGHT SENSOR 33
      PROGRAMMING EXPRESSION ................................................. 34
  6.2) Learn from Samples ............................................................. 35
  6.3) Mini-Challenge ..................................................................... 37

Ch.7 – Work with Ultrasonic Sensor ............................................. 38
  7.1) Learn about the sensor ........................................................ 38
      VALID SENSOR VALUE ......................................................... 38
      PROGRAMMING EXPRESSION ................................................. 38
  7.2) Learn from Samples ............................................................. 39
  7.3) Mini-Challenge ..................................................................... 42

Ch.8 – Work with Gyro (EV3 Only) .............................................. 43
  8.1) Learn about the sensor ........................................................ 43
      VALID SENSOR VALUE ......................................................... 43
      PROGRAMMING EXPRESSION ................................................. 43
  8.2) Learn from Samples ............................................................. 44
  8.3) Mini-Challenge ..................................................................... 45
Ch.9 – Work with Motor Encoder................................................................. 46
  9.1) Learn about the encoder................................................................. 46
  VALID SENSOR VALUE
  PROGRAMMING EXPRESSION
  UTILIZE MOTOR FEEDBACK CONTROL
  9.2) Exercises ....................................................................................... 48
  9.3) Mini-Challenge................................................................................ 49

Ch.10 – Using variables ........................................................................... 50
  WHAT IF I WANT TO MAKE A VARIABLE EQUAL TO AN MATH EXPRESSION: 50

Ch.11 – More in Debugging ...................................................................... 52
  Button Control (for EV3 only).............................................................. 52
  Trouble shooting tips......................................................................... 53
  COMMON MISTAKE WHEN USING WAIT(...) ................................. 53

Ch.12 – Advanced Challenge ................................................................. 54
  Challenge 1 – line trace a maze filled with U-turns and 90-degrees turns. 54
  Challenge 2 – Figure out the secret Music scroll................................. 55
  Challenge 3 – Fixed Sized Obstacle Avoidance.................................. 56
  Challenge 4 – Zombie Challenge....................................................... 57
Before you Start

This tutorial is written mainly to be used in our Roboclub. If you are a completely beginner, discussion that you receive in class will be very helpful.

This tutorial is laid out in the following approach to fit our roboclub model:

<table>
<thead>
<tr>
<th>Ch. 1</th>
<th>Get to know the working environment. Just in high level tour.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ch. 2</td>
<td>Learn best practice – divide and conquer</td>
</tr>
<tr>
<td></td>
<td>Learn to be resourceful</td>
</tr>
<tr>
<td></td>
<td>- use the Help feature</td>
</tr>
<tr>
<td></td>
<td>- use the debugger</td>
</tr>
<tr>
<td></td>
<td>This should help you to become more self-sufficient.</td>
</tr>
<tr>
<td>Ch. 3</td>
<td>Learn simple flowchart design. Should start this before programming, even with young folks like Gr. 4 to 6. You may skip this chapter if you are already familiar with simple flowchart design.</td>
</tr>
<tr>
<td>Ch. 4-9</td>
<td>More indepth explanation in programming concepts by following the program menu side bar. Organized by motors, basic sensors and motor encoder with basic gear math. Since there is no support to incorporate variables, you will have to work out the math work on a piece of paper first. This is actually a good thing to reinforce mechanical gear math for the younger folks.</td>
</tr>
<tr>
<td>Ch. 10</td>
<td>Others and More in Debugging.</td>
</tr>
<tr>
<td>Ch. 11</td>
<td>Advanced challenge.</td>
</tr>
</tbody>
</table>

**General Guidance to Follow**

- **Chapter 1** should read the entirety. This is easy to read on your own.
- **Chapter 2 & 3** meant to help you become a resourceful self-learner. Assistance from discussion in class will definitely be important if you start out as a beginner. Depending on your level, instructor may skip these 2 chapters, and go right to basic motors operation. If you are experienced and a mature learner, you should cover these two chapters first.
- **Chapter 4 to 7** cover basics as well as more complex work. All more complex work is marked with (*). Instructor may choose to skip (*) depending on your level.
- **Chapter 8 to 9** jump right into more complex structure, as it assumes you have got much practice from chapter 4 and 7. At Roboclub, instructor will extract partial chapter based on your level.
Chapter 10  – This is an invaluable skill. Debugging skill will be continuously exercised in every single project. You will exercise debugging throughout the entire project process. This is documented here as your reference.

Chapter 11  – Advanced Challenge to evaluate your level of understanding by solving various problems.

Note:
RobotC/CMU Robotics Academy has release a curriculum along using their Virtual World Software – http://robotvirtualworlds.com/. This is an awesome tool for kids to learn about robotics programming without the cost of a physical kit!
Ch. 1 – A Brief Tour of the main Screen

1.1) Robot Type

1.2) Basics on the Tool Bar

a) Motor and Sensors Configuration setup

Select proper variable names for the motors and sensors ports. Instructors will discuss what rules you should go by.
b) Firmware Download

First time user, you should always download the firmware first. This must be done at least once before you the RobotC compiler works. You should not need to do this again after that. After you download a new version, you should redo this firmware download as well.

c) Save File

Save your file often! Even when you are still editing your file, you should have the habit to save your file.

d) Compile Program

Compiling your program is a must before you can download it to your robot. It is primarily used for programs that translate your code. Strictly speaking, a compiler will translate your code to a lower level language that the robot will eventually understands.

e) Download to Robot

Once compilation is successful, you will need to perform “Download to Robot”.
# 1.3) Program Menu

<table>
<thead>
<tr>
<th>Side Menu</th>
<th>About</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Flow</td>
<td>Control Structures. This is where you will get motor &amp; sensor feedback data as well.</td>
</tr>
<tr>
<td>Variables</td>
<td>Creating variables and entering math expressions</td>
</tr>
<tr>
<td>Simple Behaviors</td>
<td>default functions which encapsulate sequence of motor behaviors</td>
</tr>
<tr>
<td>Motor Commands</td>
<td>Motor Functions</td>
</tr>
<tr>
<td>Remote Control</td>
<td>Remote control functions (Only applicable If you have a joystick controller).</td>
</tr>
<tr>
<td>Timing</td>
<td>Timer functions.</td>
</tr>
<tr>
<td>Line Tracking</td>
<td>Encapsulated functions for line following</td>
</tr>
<tr>
<td>Datalog</td>
<td>Logging data for analysis *** excellent tool for analysis!</td>
</tr>
<tr>
<td>Display</td>
<td>Display Functions</td>
</tr>
<tr>
<td>EV3 LED</td>
<td>Only for EV3 – LED specifics</td>
</tr>
<tr>
<td>Gyro Sensor</td>
<td>Only for EV3 – for Gyro sensor feedback</td>
</tr>
<tr>
<td>Touch Sensor</td>
<td>Encapsulated touch “press” count (mainly to overcome the debouncing issue – see below)</td>
</tr>
<tr>
<td>Sounds</td>
<td>Basics play sound functions</td>
</tr>
</tbody>
</table>

Each item will be broken in further details in the following sections. The “Program Flow” is the most extensive one. This tutorial will skip:

- the “Line Tracking” as it encapsulates line tracking which will be shown as part of the programming sample instead.
- The “Remote controller” applies only the joystick controller.

**About the Touch Sensor:**

- A “push” is counted as one if it is ever released.
- That means, one push == one push followed by a release. Therefore, if a touch sensor is pressed down for 2 seconds and released, it is still counted as “a single” push.
Ch.2 – Be Resourceful & Exercise Best Practice

Utilize the Help Feature

- To Look up the parameters definition.
- Press F1 or Click on Help on the far right the menu bar.

Most parameters required for other icons are rather self-explanatory. The one that you will find this Help Viewer particularly useful is the “Motor Commands”. Go ahead to explore to see the available icons and their parameters.

Learn from Samples

Besides the samples provided along with this tutorial, RobotC also provides a rich array of samples.

Steps to following when using samples:

1. Must know the goal of the sample program. So, you should make sure you know what behavior the code is trying to do.
2. Review the code to at least have an intuition what the code does.
3. Write a flowchart, do the program your own when you are ready.
4. Stay inquisitive and try some slightly different scenarios
**Proper Steps to Follow When You Are Tackling a Challenge or Project**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define the objective</td>
<td>What is the main goal you are trying to achieve?</td>
</tr>
<tr>
<td>Analyze the logic</td>
<td>What is/are required? List steps to follow.</td>
</tr>
<tr>
<td>Design</td>
<td>Put logic in symbols (via flowchart). Hand check the logic</td>
</tr>
<tr>
<td>Programming</td>
<td>Follow the flowchart.</td>
</tr>
<tr>
<td>Execute</td>
<td>Observe</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>Test to maximize stability. Debugging for errors</td>
</tr>
</tbody>
</table>

**Guidelines for the Most Efficient Coding:**

- Tidy and easy to read program layout.
- Simpllicity is the best. Complex does not mean better!
- Fault tolerant. (least error prone) .. Think all possible scenarios, and test carefully.
- Sufficient meaningful comment... do not count the comment which says the obvious. For example:

```
// put a meaningful comment here
2: while (getTimer(T1, seconds) <= 5 ) {
3:   setMotor( motorA, 50 );
4:   wait( 2, seconds );
5 :}
```

- **a meaningless comment:**  ➔ turn motor A for 50% power level for 2 seconds
- **a meaningful comment:**  ➔ turn around for about 360 degrees

**Process of Trouble-Shooting Hint:**

To trouble-shoot, you need to pretend you were the robot. Then, you STRICTLY follow each of the commands in your program, but NOT WHAT YOU THINK it should be doing.

Remember: the robot cannot read your mind. It only does EXACTLY what the program tells it to do, not what the programmer wants it to do. So, if your program tells it to jump off a cliff to its demise, it will do just that.
**3.1) What is a Flowchart?**

- Is a programming tool to help you convert algorithms into computer programs which robot will understand.
- Consists of symbols which graphically depict the logical steps to carry out a task.
- Shows steps and their interconnections.
- Consists of symbols which represent a set of behaviors such as a sequence of movements or actions.
- Shows building blocks in relation to its functions.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>Start / End</td>
</tr>
<tr>
<td>🎉</td>
<td>Process/Action</td>
</tr>
<tr>
<td>🔄</td>
<td>Conditions/Decisions</td>
</tr>
<tr>
<td>🔗</td>
<td>Flow Line</td>
</tr>
<tr>
<td>🔄</td>
<td>Merge point for conditions</td>
</tr>
</tbody>
</table>

*Icons that you should know* for the samples used in this chapter, we’ll use sound/music and timer to show how to set up basic control structure samples.

Now, just for your future reference, you may hear about Pseudocode. This is English-like phrases with “Visual “ basic terms to outline the logic flow as well. For more advanced samples in the later chapters, high level design will use flowcharts, but low level design may use Psuedocode.

**3.2) Complex to Simple and Basic Behavior**

Main task: Get ready to get on the school bus to go to school in the morning.

Note: Main task (the complex behavior) is divided into sub-tasks (simple behaviors). Each task can potentially include many more steps (basic behavior) in order to complete the main task.
3.3) **Divide and Conquer**

Use some kind of algorithms to figure out a solution a lot faster.

Example: Santa’s dirty sock problem.

- Santa has to deliver 1024 boxes of presents with all equal weight to kids. However, his helpers have left one of his dirty socks (weigh 6 oz.) inside one of the boxes! Santa told his helper to try to figure out which box has the sock and remove it. However, they have to finish this up as soon as possible. They will not be able to check the weight of every single box. Santa is smart. He said to do it by Divide and Conquer - Binary method.

![Diagram showing Divide and Conquer method](image)
3.3) Basic Terms:

Some terms you should know first…

An Expression:
- An expression is like a building block. A set of them will compose a complete building.
- In your case, combination of expressions will complete your program. This program can be downloaded into the robot and run/execute.
- It may be a combination of symbols and programming terms that represents functions or value

Boolean Expression:
- An expression which provides a “true” or false value
- E.g. if (traffic light turns red) this is the boolean expression
- I stop the car
- else
- I continue driving

Boolean Operators:
- <, >, <=, >=, ==, !=
- ==: “is it equal to”. Beware:
- E.g. x==y means x the same as y
- x = y means put the value of ‘y’ into ‘x’

In this chapter, we will use the 4 timers T1, T2, T3, and T4. We won’t use sensors or motors yet because we want to draw your attention to focus on the control structures.

Sample Timer codes:

```
Timing
resetTimer
wait

Sounds
playSound
playTone
```

Drag to the working area, And select the timer to use

Play a sound. PlayTone at a frequency with duration In 10 MsecTicks play a tone at a frequency of 800 for 150 milliseconds
3.4) **CONTROL STRUCTURES WITH FLOWCHART**

3.4.1) **If- Control Structure:**

Example 1 – if- control structure

To show how “if” control structure works...

Step 1) write a flowchart for this. For example, if it is within 5 seconds, play sound and wait for 1 second.

Step 2) Do the following:

- Drag to the working area, select and type in the information as shown here on the right.

Step 3) Hit F7 to compile to see there is any syntax error. Then hit F5 or click on
Example 2 – if-else control structure

Step 1) Define the behavior (task):

If $T_1 \leq 5$ seconds
   Play Upward Tones
   Wait for 1 second
else
   Play Downward Tones
   Wait for 2 seconds

Step 2) Write your flowchart for this program

Step 3) Program the task.

```cpp
1 if(getTimer(T1, seconds) <= 5) {
2   playSound(soundUpwardTones);
3   wait(1, seconds);
4 } else {
5   playSound(soundDownwardTones);
6   wait(2, seconds);
7 }
```

Step 4) Compile and Download to your robot, and execute
Example 3 – if-else control structure with tasks outside the condition

Step 1) Define the behavior (task):
- Do the same task as Sample 2.
- After that, play BeepBeep sound. Wait for 1 second before terminate.

Step 2) Write your flowchart for this program.

Step 3) Program the task.

```c
if( getTimer(T1, seconds) <= 5 ) {
  playSound( soundUpwardTones );
  wait( 1, seconds );
} else {
  playSound( soundDownwardTones );
  wait( 2, seconds );
  wait( 1, seconds );
  playSound( soundBeepBeep );
}
```

Step 4) Compile and Download to your robot, and execute.
3.4.2) Loop (Repeated-ness) Control Structure

### repeat

- **Repeat (N)**
  - <= N times
  - Process(es)...

### repeatUntil

- **Repeat Until (Boolean Operation)**
  - Boolean operation is false
  - Process(es)...

### while

- **While (Boolean Operation)**
  - Boolean operation is true
  - Process(es)...

**Program Flow**
- repeat
- repeat (forever)
- repeatUntil
- while
Loop Examples 1

Step 1) Define your goal: Repeat playing sound “Beep Beep” 5 times.

Step 2) Write your flowchart for this program.

Step 3) Write your program code.
Unfortunately, it does not seem to play the sound. Fix it.

Step 4) Compile and Download to your robot, and execute
Loop Example 2

Step 1) Define the Task: Play Upward Tone, wait for 1 second. Then, play Downward tone, wait for 1 second. Repeat this until T1 timer is greater than 5 seconds.

Step 2) Flowchart

Step 3) Program

Answer the following question:

Note the “while” loop expression is different. The boolean condition is different. Why?
3.4.3) Nested Control Structure

**Nested Loop Example 1**

Step 1) Description of the behavior (task):

1. Your program will play “soundBeepBeep” 5 times with 1 second interval.
2. For each “soundBeepBeep”, it will play “soundShortBlip”s 4 times. There should be 500 milliseconds interval for each “soundShortBlip”.

```c
resetTimer (T1 = 0);
while (getTimer(T1, seconds) <= 5) {
    playSound(soundUpwardTones);
    wait(1, seconds);
    playSound(soundDownwardTones);
    wait(1, seconds);
}
```
Step 2) Flowchart
Since the logic may seem to be a bit complex, you need to **Divide and Conquer**.

![Flowchart](image)

Step 3) Program
3.a) Follow **the divide and conquer** method to create two separate programs:

<table>
<thead>
<tr>
<th>Sub-program 1</th>
<th>Sub-program 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.</strong> Break down the tasks into 2 small tasks as shown on the right.</td>
<td></td>
</tr>
<tr>
<td><strong>2.</strong> Write your flowchart for each of the sub-programs.</td>
<td></td>
</tr>
<tr>
<td><strong>3.</strong> For each sub-program, compile and download to your robot, and execute.</td>
<td></td>
</tr>
<tr>
<td><strong>4.</strong> Ensure it produces result as expected.</td>
<td></td>
</tr>
</tbody>
</table>

3.b) Test each sub-task to ensure they work as expected.
3.c) **Integrate (Merge)** the sub-programs into one.

1. Write a complete flowchart for entire program. (Note, usually a high level flowchart which calls the individual sub-tasks should be done first. For this sample, we do the sub-tasks first just as a more straightforward sample.)
2. Complete the integration.
3. Compile and download to your robot, and execute.
4. Ensure it produces result as expected

Step 4) Compile, Download, and Execute
Nested Loop Example 2

Step 1) Description of the task:
1. Your program will run for 5 seconds.
2. If it is during the first 2 seconds, it plays “soundBeepBeep” with 500 millisecond interval.
3. Otherwise (else), it plays “soundBlip” with also 500 milliseconds interval.

Step 2) Flowchart

Again, let’s use divide and conquer method again. However, this is a more appropriate way for designing your modules. First, it starts with a high level flowchart including names of the sub-task(s) only. After that, each sub-task should have its own detailed design flowchart.

Step 3) Program

3a) Follow the divide and conquer method to create two separate programs:

```cpp
while (getTimer(T1, seconds) <= 5) {
    // repeat while it is less than 5 seconds...
}

if (getTimer(T1, seconds) <= 2) {
    playSound(soundBeepBeep);
    wait(500, milliseconds);
} else {
    playSound(soundBlip);
    wait(500, milliseconds);
}
3b) Test each sub-task to ensure they work as expected.

3.c) ) **Integrate (Merge)** the sub-programs into one.
   i. Write a complete flowchart for entire program
   ii. Complete the integration.
   iii. Compile and download to your robot, and execute.
   iv. Ensure it produces result as expected.

Step 4) Compile, Download, and Execute

### 3.5) Recapture the Steps:

1) Define the behavior
   — Identify the requirement

2) Design with Flowchart
   If problem is complex, you must use **Divide and Conquer** method. This means:
   — Write a High level flowchart first. This does not have detailed logic
   — Break down the whole task into sub-tasks
   — Expand each sub-task into a more detailed flowchart

3) Program
   — Program each sub-task, if any.
   — Test each sub-task to ensure they work as expected.
   — Integrate (Merge) them, if there are 2 or more sub-tasks.
   — Test the whole system again.
4.1) LEARN ABOUT MOTORS

- know distance traveled using time is not accurate
- almost no two motors are identical.

Valid Values

Power level = -100 to 100, where 0 == stop

Programming Expression

<table>
<thead>
<tr>
<th>NXT</th>
<th>EV3</th>
<th>Description</th>
</tr>
</thead>
</table>
| ☑️  | ☑️  | setMotor
| ☑️  | ☑️  | May set any single one of the 4 motor ports to a certain power level. |
| ☑️  | ☑️  | setMultipleMotors
| ☑️  | ☑️  | May set multiple motors |
| ☑️  | ☑️  | stopAllMotors
| ☑️  | ☑️  | stopMotor
| ☑️  | ☑️  | stopMultipleMotors
| ☑️  | ☑️  | Display the motor encoder value. Good way for debugging. |
| ☑️  | ☑️  | displayMotorValues
| ☑️  | ☑️  | After v4.28+ |
| ☑️  | ☑️  | Skip these for now. This will be covered in the later chapter. |
| After v4.28+ | ☑️  | setMotorTarget |
4.2) Learn from Samples

Ex1) Robot goes forward for 2 seconds and backward for 3 seconds.

```
21 setMotor (motorA ←, 50 );
22 setMotor (motorC ←, 50 );
23 wait ( 2 , seconds ←);

// alternative way to move more than one motor to same power level...
25 setMultipleMotors ( -50 , motorA ←, motorC ←, ←);
26 wait ( 3 , seconds ←);
```

Ex2) Do (1) for 4 times.

Wrap around the code in the “Repeat” loop structure.

Ex3) Do (1) for 5 seconds.

a) Use Repeat Until

```
17 repeatUntil ( getTimer (T1, seconds ←) ≤ 5 ) {
18 setMultipleMotors ( 50 , leftMotor ←, rightMotor ←, ←);
19 wait ( 2 , seconds ←);
20 setMultipleMotors ( -50 , rightMotor ←, rightMotor ←, ←);
21 wait ( 3 , seconds ←);
}
```

b) Use While

```
17 while ( getTimer (T1, seconds ←) < 5 ) {
18 setMultipleMotors ( 50 , leftMotor ←, rightMotor ←, ←);
19 wait ( 2 , seconds ←);
20 setMultipleMotors ( -50 , rightMotor ←, rightMotor ←, ←);
21 wait ( 3 , seconds ←);
}
```
**Ex4)** Robots runs for 5 seconds and terminate. During half way into the 5 seconds, it goes forward. During the 2nd half of the 5 seconds, it goes backward.

4.3) **Mini-Challenge**

**Challenge 1)**
Do a pivot turn for N milliseconds so that it makes almost a perfect 90 degree left turn.

**Challenge 2)**
Do a pivot turn for N milliseconds so that it makes almost a perfect 90 degree right turn.

**Challenge 3)**
Robot goes forward for 1 second, then makes a 90 degree left turn.

**Challenge 4)**
Repeat (3) for 4 times.
   a) Use Repeat Until
   b) Use While

**Challenge 5)**
Repeat (3) and stop in 10 seconds.
   a) Use Repeat Until
   b) Use While
5.1) Learn about the sensor

Valid Sensor Value

Remember: When you work with sensor, you should know the valid range of the sensor value first.

Sensor Value of Touch Sensor
1 == means touch sensor is pushed
0 == means touch sensor is released

Programming Expression

Setup the sensor from the top menu.

You can use it in any of the control structures. Example:

```
while (Select a Value < == 0)
{
  // motor A == left; motor C == right
  // Touch sensor on S1
  int TS; // bumper (Touch on S1)
  TS = 0;
  if (TS == 1)
  {
    // Set both motors to 50% power level
    setMultipleMotors(50, leftMotor, rightMotor);
  }
  else
  {
    // Stop all motors
    stopAllMotors();
  }
}
```

5.2) Learn from Samples

Ex 1) Robot goes forward until it hits an object.

a) Use Repeat-until

<table>
<thead>
<tr>
<th>Flowchart</th>
<th>Program</th>
</tr>
</thead>
</table>
| ![Flowchart](chart.png) | ```
repeatUntil(SensorValue[touch] == 1) {
  setMultipleMotors(50, leftMotor, rightMotor);
}
stopAllMotors();
``` |
**Graphical RobotC Tutorial Packet**

**Computational Thinking and Engineering for Gr. 4 to 12**

**Storming Robots**

---

**b) Use While**

**Flowchart**

```
While touch sensor is not pushed
  TS=1
  Stop all motors

TS==0
  Set both motors to 50% power level

```

**Program**

```
while (SensorValue[touch] == 0) {
  setMultipleMotors (50, leftMotor, rightMotor);
  stopAllMotors();
}
```

---

**Ex 2) Do (1), but robot turns around for about 180 degree if your robot hits an object.**

**a) Use Repeat/ Until**

**Flowchart:**

```
Repeat until touch sensor == 1
  MotorA=50%
  MotorC=50%

sensor==0
  MotorA=50%
  MotorC=50%

```

**Program**

```
1 repeatUntil (SensorValue[touch] == 1) {
  setMultipleMotors (50, leftMotor, rightMotor);
}
2 setMotor (leftMotor, 50);
3 setMotor (rightMotor, -50);
4 wait (1.5, seconds);
5 stopAllMotors();
6 stopAllMotors();
7 stopAllMotors();
8 stopAllMotors();
```

---

**b) Do this with “While”**

Now, you should try this yourself.
Ex 3) Do (2), but robot should do some wobbling dance after it turns 180 degrees around.

a) Use Repeat/ Until

Flowchart:

```
repeatUntil (SensorValue[touch] == 1) {
  setMultipleMotors (50, leftMotor ->, rightMotor ->, ~);
}

setMotor (leftMotor ->, 50);
setMotor (rightMotor <-, -50);
wait (1.5, seconds ->);
repeat (4) {
  setMotor (leftMotor ->, 50);
  setMotor (rightMotor <-, -50);
  wait (500, milliseconds ->);
}
setMotor (leftMotor <-, -50);
setMotor (rightMotor ->, 50);
wait (500, milliseconds ->);
stopAllMotors();
```

b) Use “While”

Now, you should try this yourself.
Ex 4) Repeat (2), but terminate after 10 seconds elapse
a) Use “Repeat Until”

Flowchart:

Program:

```
repeatUntil(SensorValue[touch] == 1) {
  setMultipleMotors(50, leftMotor, rightMotor);
}
setMotor(leftMotor, 50);
setMotor(rightMotor, -50);
wait(1.5, seconds);
resetTimer(T1);
repeatUntil(getTimer(T1, seconds) == 10) {
  setMotor(leftMotor, 50);
  setMotor(rightMotor, -50);
  wait(500, milliseconds);
  setMotor(leftMotor, -50);
  setMotor(rightMotor, 50);
  wait(500, milliseconds);
}
stopAllMotors();
```

b) Use “While” in both loop.

Now, you should try this yourself.
5.3) **Mini-Challenge**

**Challenge 1)**
Program your robot to do the following forever:
- Go Forward
- If it hits an obstacle, it goes backward for 1 sec. Turn around for approximately 180 degrees.

Steps:
A) Draw a flowchart. Remember to hand test it first before you program.
B) Program it, execute, test and debug.

Hint: forever loop. Within the forever loop, there is an if-structure.

**Challenge 2)**
Do Challenge 1, but the robot will terminate after 5 seconds.

Hint: timer in a “while” loop. Inside the loop, there is an if-structure. Do not do reset timer first.

**Challenge 3)**
Do Challenge 2, “Repeat Until” structure:

Hint: timer in a “Repeat Until” loop instead of “while”. **MUST** change the Boolean expression

**Challenge 4)**
Robot is happily testing its wheel muscles by going forward for 10 seconds. However, if it detects an object, it will stop and make a sound to signal you to remove the obstacle. Then, it will continue to run forward and hope that the obstacle will be removed. It will continue to do that within the first 7 seconds. After 7 seconds elapse, if it detects an object again, it gets very angry, and decides to turn around 180 degrees and terminate.

A) Draw a flow chart first. Remember to hand test it first before you program.
B) Program it, execute, test and debug.

Hint: use “while” timer loop. Within that, use if-structure
6.1) **Learn about the Sensor**

**Valid Sensor Value**

<table>
<thead>
<tr>
<th>0%</th>
<th>50%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Led is for light source</td>
<td>The Photo-Transistor will pick up the light reflection and provide a value depending on the reflectiveness of a surface</td>
<td></td>
</tr>
</tbody>
</table>

LED: emits light which is reflected off of a surface and picked up by the photo-transistor.

Before you proceed, you must attach a light sensor onto your robot. Your light sensor must face down in order to see the floor.

It reports the amount of light to the main controller (the brain brick) as a number between 0 (total darkness) and 100 (very bright). The light sensor uses its own light source, the light emitting diode (LED), to illuminate a small area for its receiver.

For EV3: It comes with a color sensor which provides you the same functionalities as the NXT light sensor.

For best reading:
- about 1cm perpendicular to the target surface
- avoid sporadic ambient light.

**To get the light sensor feedback from Light Sensor**

1) Go thru the NXT on screen menu:
   a) Select “View”
   b) Scroll to find and select Reflective (Light) Sensor.
   c) Scroll to find and select the port where your sensor is attached to.
2) Check the reflectance rate
   a) Place the light sensor on the top the black. Record the reading:
      i. Light sensor value of the dark line = ________%
   b) Place the light sensor on the top of the white area. Record the reading:
      i. Light sensor value of white area = ________%
   c) Average of these two values = ________%

3) This is the **threshold value** you will use in order for your robot to distinguish between black and white.

**Programming Expression**

Always comment your motor and sensor configuration

The following two methods perform the same behavior

For EV3:
Example to choose a color sensor: (Again, you must first do the “Sensor Setup” from the Setup Menau.)

**Hint:** Use `displaySensorValues` to display what the feedback value is. This is very useful for debugging purpose.
6.2) Learn from Samples

Ex1) Have robot run forward toward the edge of a table.
    a) Use “Repeat Until”.

Flowchart: the same as above, except this:

Program:
```
repeatUntil (SensorValue[light] <= 37) {
  setMultipleMotors (50, leftMotor, rightMotor);
}
stopAllMotors();
```

b) Use “while”. (work on this on your own.)

Ex2) Robot will trace a circle. The circle is formed by a 1” black tape.
    a) Do this indefinitely.

Program (for forever):
```
repeat (forever) {
  if (SensorValue[light] <= 37) {
    setMotor (leftMotor, 0);
    setMotor (rightMotor, 50);
  } else {
    setMotor (leftMotor, 50);
    setMotor (rightMotor, 0);
  }
}
```

b) Repeat this for only 20 seconds - “Repeat Until”. (work on this on your own.)

    c) Do (b), but with “while” loop. (work on this on your own.)

Ex3) Do (2), except robot stops when it sees silver.
    a) Do this with “while” loop.
b) Do this with “Repeat Until”
6.3) **Mini-Challenge**

**Challenge 1)**
Robot needs to detect two cracks (marked by black tape). After it completely passes the 2nd crack, it stops.

**Hint:** use counter loop for # of cracks. Watch for what is meant by “one a complete crack”. Write flowchart to determine the details.

**Challenge 2)**
Program your robot to be a right biased line tracing. That means, when it comes to an intersection, it will follow the rightmost path.

**Hint:** still use one light sensor. Think Simplicity....

**Challenge 3)**
Do the Intense line-marathon - Go through a line maze with a lot of 90 degree turns.

**Hint:** two light sensors. Important that you will write a flowchart first.

**Challenge 4)**
Robot traces a path using one light sensor. It must stop when it reaches the silver tape.
7.1) Learn about the sensor

Valid Sensor Value

For NXT: $7 \leq \text{Reliable Value} \leq 50s$
For EV3: $5 \leq \text{Reliable Value} \leq 70s$

Accuracy will also depend on surface curvature the sound wave hits, and sufficient power source.

Programming Expression

By now, you should be familiar with the basic control structure of if-, if-else, and loop structure using “while”, “repeat until”, and “repeat”. The structure itself does not change. The changes that will need to take place will only be the Boolean expressions, where you put sensor type, and its corresponding value.

A simple if-structure using ultrasonic sensor feedback:
7.2) Learn from Samples

**Ex1) Simple Obstacle Maze.** Whenever robot detects an obstacle closer than 7 cm, it turns right 180 degrees. Continue this forever.

Flowchart

Program

```c
repeat (forever) { 
  if (SensorValue[sonar] <= 7) { 
    setMotor(leftMotor, -50);
    setMotor(rightMotor, 50); 
    wait(2, seconds -);
  } else { 
    setMultipleMotors(50, leftMotor, rightMotor); 
  } 
}
```
**Ex2) Do (1), but robot stops after it detects 4 obstacles.**

Flowchart 1

<table>
<thead>
<tr>
<th>Flowchart 2 – break the behavior down into more programmable units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat(4)</td>
</tr>
<tr>
<td>MotorA=50% MotorC=50%</td>
</tr>
<tr>
<td>Wait until ultrasonic is &lt;=7</td>
</tr>
<tr>
<td>MotorA=50% MotorC=50%</td>
</tr>
<tr>
<td>Wait 2 sec</td>
</tr>
<tr>
<td>Stop motors</td>
</tr>
</tbody>
</table>

Program:

```
repeat (4) {
    setMultipleMotors(50, leftMotor «, rightMotor «, «);
    waitUntil(SensorValue[sonar] <= 7);
    setMotor(leftMotor «, 50);
    setMotor(rightMotor «, -50);
    wait(2, seconds «);
}
stopAllMotors();
```
**Ex3) Simple Obstacle Avoidance**

Obstacle size is about 10cm in diameter. Robot goes forward and stops if it detects object within 7 cm. It should then get around the obstacle and stop at the opposite side of the obstacle.

Important: Divide and Conquer

Step 1: Robot goes forward and stops if it detects object within 7 cm.

Step 2: You may calculate the ratio of the two motors power level, or simply use trial and error method to go around the obstacle. At this step, you should focus on finding out the power levels of the two motors.

Step 3: You should estimate how long it will usually take for the robot to reach the opposite side of the obstacle. Then, create a timer loop around Step 2.

Step 4: Integrate Step 1, 2 & 3.
Integrate...

Note: you can also use “Repeat Until” expression:

7.3) MINI-CHALLENGE

**Challenge 1)**
Modify Mini-Challenges done in the Touch Sensor Chapter to use ultrasonic sensor instead of touch sensor.

**Challenge 2)**
Robot moves in a straight line towards and obstacle. When it gets within 7 cm of the obstacle, it must slowly back up until it is 20 cm away, then turn and drive quickly in the opposite direction.

**Challenge 3)**
Robot traces a black line. The end of the line is signified by an obstacle or wall 7 cm away. Robot must stop as close as possible without touching the object.
8.1) Learn about the Sensor

Valid Sensor Value

1) Degree of rotation:
   Returns the accumulated value of the gyro sensor relative to the reset point.
   Range from \(-2^{31} \leq \text{degrees} \leq 2^{31}\)

2) Headings:
   This command is useful for making movements relative to the robot’s latest reset point, instead of its current position.
   Range from -180 to +180 (in degrees).

3) Rate of change in angle:
   Returns the rate of movement for the gyro sensor in degrees per second.
   Range from -440 to +440 degrees per second

Sensor should be placed perpendicular to the plane of rotation.

During calibration, make sure you hold it still:
- When EV3 boots up
- When plug/unplug your sensor
- When resetting

Programming Expression

Reset the Gyro first: \(\text{resetGyro (S3 -)}\).

Regular loop expression sample:
8.2) LEARN FROM SAMPLES

Ex1) Simple Demo for checking Gyro Degrees of Rotation Feedback.

Program your robot to complete a 360 degree turn. It should do a small pause at 90, 180, 270 and 360 and display the motor encoder value of your chassis driving motor(s). That means, display both left and right motors if you have 2 independent motors.

```c
resetGyro (gyroSensor -);
resetMotorEncoder (leftMotor -);
resetMotorEncoder(rightMotor -);
while (getGyroDegrees(gyroSensor) <= 360 ) {
    if (getGyroDegrees(gyroSensor) == 90 ) {
        stopAllMotors();
        displayMotorValues (line3 -, rightMotor -);
        displayMotorValues (line1 -, leftMotor -);
        wait (1 , seconds -);
    }
    if (getGyroDegrees(gyroSensor) == 180 ) {
        stopAllMotors();
        displayMotorValues (line1 -, leftMotor -);
        displayMotorValues (line3 -, rightMotor -);
        wait (1 , seconds -);
    }
    if (getGyroDegrees(gyroSensor) == 270 ) {
        stopAllMotors();
        displayMotorValues (line1 -, leftMotor -);
        displayMotorValues (line3 -, rightMotor -);
        wait (1 , milliseconds -);
    }
    setMotor (leftMotor -, 50 );
    setMotor (rightMotor -, -50 );
    stopAllMotors();
    displayMotorValues (line1 -, leftMotor -);
    displayMotorValues (line3 -, rightMotor -);
    wait (5 , seconds -);
}
```
8.3) **Mini-Challenge**

**Challenge 1) with simple if-structure**
Program your robot to drive in a straight line along the edge of a table (make sure you do not allow it to fall off!). If the robot starts to veer off of its path (i.e. its relative angle increases or decreases), it must stop immediately and play a warning beep.

**Challenge 2) with simple loop structure**
Program your robot to make a perfect 90 degree turn using a while loop. Next, try 180 degrees, then 360 degrees. How accurate is the gyro sensor?

**Challenge 3) Nested loop nested with if-structure**
Program your robot to drive in the straightest line possible. If it turns more than a certain degree from its reset point, it must compensate by turning the same amount in the opposite direction. Use a measuring tape or a piece of electrical tape to test how straight the robot can go.

**Challenge 4) Nested loop nested with nested if-structure**
Program your robot to trace a line using two light sensors until it sees or hits an object. Then it should turn 60 degrees and navigate around the object until it reaches the black line again.
9.1) Learn About The Encoder

Valid Sensor Value

NXT: Range from \(-2^{15} \leq \text{encoder} \leq 2^{15}\)

EV3: Range from \(-2^{31} \leq \text{encoder} \leq 2^{31}\)

One full Motor rotation = 360 encoder value. MUST note: This denotes a count value, not type geometric rotational degrees.

Programming Expression

<table>
<thead>
<tr>
<th></th>
<th>NXT</th>
<th>EV3</th>
</tr>
</thead>
</table>
| \{ resetMotorEncoder \} & After v4.27 & \(\checkmark\) & Reset the encoder count to 0. &
<p>| { setMotorTarget }   |     |     | Set the target encoder for a single motor. This does not start the motor. You need to use setMotor( ) to start it. The internal motor control library will stop your motor for you once it reaches the specified target value. |
| { displayMotorValues } &amp; (\checkmark) &amp; (\checkmark) &amp; Display the motor encoder value. Great way for debugging purpose |</p>
<table>
<thead>
<tr>
<th>Sample Flowchart For ...</th>
<th>Sample Program Fragment</th>
</tr>
</thead>
</table>
| ![Flowchart for if only](image) | ```
if (nMotorEncoder[motorA] <= 1800) {
  setMotor(motorA, 50);
}
``` |

<table>
<thead>
<tr>
<th>Flowchart for if/else</th>
<th><img src="image" alt="Flowchart for if/else" /></th>
</tr>
</thead>
</table>
| ![Flowchart for if/else](image) | ```
if (nMotorEncoder[motorA] <= 1800) {
  setMotor(motorA, 50);
} else {
  setMotor(motorA, -50);
}
``` |

<table>
<thead>
<tr>
<th>Flowchart for while</th>
<th>Sample Program Fragment</th>
</tr>
</thead>
</table>
| ![Flowchart for while](image) | ```
while (nMotorEncoder[motorA] <= 1800) {
  setMotor(motorA, 50);
}
``` |

<table>
<thead>
<tr>
<th>Flowchart for repeatUntil</th>
<th>Program</th>
</tr>
</thead>
</table>
| ![Flowchart for repeatUntil](image) | ```
repeatUntil (nMotorEncoder[motorA] > 1800) {
  setMotor(motorA, 50);
}
``` |

This works exactly like the while above.
**Utilize Motor Feedback Control**

```c
setMotorTarget ( motor port, encoder value, power level );
```

- this will pre-set the target encoder value, and start the motor to the desired power level.
- To wait for the motor to complete its rotation before you proceed to perform other tasks, you need to do:

```c
waitUntil (getMotorMoving (motor port) == 0 );
```

- “== 0” means motor stops. “== 1” means motor is moving

Things that you should be aware. For example, you wish to run your robot with 100% power level for 1800 encoder value and stop. You chose to use RobotC’s intrinsic motor feedback control – setMotorTarget.

Do not do this:

```c
resetMotorEncoder ( armMotor );

setMotorTarget ( armMotor, 1800, 75 );

waitUntil ( getMotorEncoder (armMotor) == 1000 );

playSound ( soundBeepBeep );

displayMotorValues ( line1, armMotor );

wait ( 6, seconds );
```

Do this instead:

```c
resetMotorEncoder ( armMotor );

setMotorTarget ( armMotor, 1800, 75 );

waitUntil ( getMotorMoving (armMotor) == 0 );

playSound ( soundBeepBeep );

displayMotorValues ( line1, armMotor );

wait ( 6, seconds );
```

### 9.2) Exercises

**Ex1)** Robot runs forward until it reaches 1800 degree and stops.

**Ex2)** Program your robot to run in a square of 100cm on each side. **Hint:** use encoder to restrict it to make 90 degrees turn.

a) Use Repeat Until

b) Use While

Note: make sure you will do your Flowchart before Program

**Ex3)** Check the overshooting effect.
- Do a) Program your robots to run 100% power level “while” encoder value < 3000.
- Do b) Mark the start point and end point where robot stops at.
- Do c) Redo the program with setMotorTarget method

**Ex4) Do (3), but ...: (with nested encoder loop)**

a) Use Repeat Until  
b) Use While

---

**9.3) Mini-Challenge**

**Challenge 1) with simple if-structure**  
Program your robot to go forward for 4 seconds. At this point, it should check its encoder value. If the encoder value is >= 1200, the robot should stop and beep. Otherwise, it keeps going forward for 1 more second.

**Challenge 2) with simple loop structure**  
Estimate the encoder value needed for your robot to travel 25 cm in a straight line. Program using your estimate to see how close you got. Can you improve your estimate?

** Challenge 3) Nested loop nested with if-structure**  
Program your robot to run in a straight line for 1,800 encoders, stopping and beeping after every 600 encoders.

** Challenge 4) Nested loop nested with nested if-structure**  
Program your robot to trace a very curvy or angled line using two light sensors. The robot should stop when its left motor has reached 2,000 encoders.
Ch.10 – Using variables

Create a variable : set the value to a single number  
NOTE: all are default to “float data type”

Create a value : set the value to a device value

`Variable [Value]`

e.g.  

```
6 lineEye = SensorValue[S3];
```

Create a value : set the value by using an expression

`Variable [Expression]`

```
7 lineEye = SensorValue[S3] + 2;
```

What if I want to make a variable equal to an math expression:

```
2 tireDiameter = 10;
3 wheelsBase = 20;
4 Enc1Cm = 360.0 / (tireDiameter * PI);
5 displayVariableValues(line2, Enc1Cm);
```

```
Enc1Cm = 11.46
```

Warning: Do not use long variable names, you may have only be able to see partial number as it exceed the width size of the screen.

Advice:
Test the value of your calculation before you actually put that as part of actual program. You may utilize the “PC Emulator Mode” as well if you are not counting on any physical robot and its sensors value. (From “Robot” → “Compile Target”)

Bug at version 4.55:
Casting of data type does not seem to work properly.
Advice:
Depending on the students’ algebra level, it is actually better to mandate the students to do the calculation on their own in order to reinforce their math skill.

See! While it is more efficient to use variables, if you do not understand it, you won’t be able to utilize it in another applications properly anyway! So, the Key is to “UNDERSTAND” it, “NOT TO MEMORIZE” it.
**BUTTON CONTROL (FOR EV3 ONLY)**

Button control allows robot to provide information interactively with human, such as display values, or change actions, etc. This is a great tool for debugging purpose.

Very simple sample:

```c
repeat (forever) {
  if (getButtonPress(rightButton) == 1) {
    setLEDColor (ledGreenFlash);
    stopMotor (motorA);
  } else {
    setMotor (motorA, -50);
  }
}
```
**Trouble Shooting Tips**

1. Comment your code. Use Meaningful comment.
2. Use sound.
3. Divide your sub-modules with comment.
4. Test one sub-module at a time.
5. May set the LED color (EV3 only)
7. Use Pause, i.e wait(...)

**Common mistake when using wait(...)**

Issue: You want the robot to pause for 5 second so that you can check on something. However, you neglect that robot is still moving while it is in the “5 seconds” wait time.
Ch. 12 – Advanced Challenge

NOTE:

Before you tackle the following challenge, you should review the best practice guideline in Chapter 2. If you follow the best practice discipline, your development time will be much shorter, and obtain higher change in success.

Reminder:

1) Divide and Conquer
2) Design one high level flowchart. This flow chart should reflects most sub-tasks.
3) Design each sub-task with its own flow chart.
4) Make sure all sub-tasks are well commented with information pertaining to the context of the task or behavior.

CHALLENGE 1 – LINE TRACE A MAZE FILLED WITH U-TURNS AND 90-DEGREES TURNS.

Your robot should be able to navigate the follow path with one or two light sensor.

![Diagram of line trace maze]
**CHALLENGE 2 – FIGURE OUT THE SECRET MUSIC SCROLL**

There will be a barcode laid out so that each bar denotes a separate note, depending on its width in centimeters. Your job is to program your robot to figure out what the secret song is.

**Steps for modularization:**

1. Robot must go forward and stop when it sees the black tape. (Make a beeping noise to signify when robot has reached the black)
2. Robot must then continue forward until it sees white again.
3. Put steps (1) and (2) together and repeat until silver tape is reached.

**Now we can move on to the steps required to measure each bar:**

1. Robot must use motor encoder to determine distance traveled across each black bar. Make sure to display motor encoder value and record each time the robot passes over a bar.
2. The next step requires some pencil and paper math work. Convert the encoder value to centimeters using the following equation: \( cm = \text{encoder value} \times (\pi \times \text{tire Diameter})/360 \) **Please note that this equation assumes a 1 to 1 gear ratio.**
3. The following table can be used as the key for cm values and their corresponding notes:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 cm</td>
<td>3 cm</td>
<td>4 cm</td>
<td>5 cm</td>
<td>6 cm</td>
<td>7 cm</td>
<td>8 cm</td>
</tr>
</tbody>
</table>

**Now it’s time to figure out the secret song!**

This will require writing a separate program to play the notes that you have determined based on your centimeter calculations.

To play a note, use the command `playTone(440, 50);` where the first number (440) is the frequency of the note, and the second number (50) is the duration in 10 millisecond ticks.
Here is a table of frequencies of the notes, using middle C as a starting point:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>440 Hz</td>
<td>493.88 Hz</td>
<td>261.63 Hz</td>
<td>293.66 Hz</td>
<td>329.63 Hz</td>
<td>349.23 Hz</td>
<td>392 Hz</td>
</tr>
</tbody>
</table>

**Please note that the frequency is an integer type in the playTone command, so you must round.**

---

**CHALLENGE 3 – FIXED SIZED OBSTACLE AVOIDANCE**

Goal: Robot should perform line tracing. When robot detects on obstacle, it should get around it and stops back on the line behind the obstacle. Obstacle is a cylinder with a known size diameter.

Note: you may figure out the power levels for the left and right motor side based on various methods:

a) Ad-hoc: use trial and error, guess and check

b) Computational: calculate the ratio of the two sides power levels based on the diameter of the obstacle.

This is one possible simple flowchart:
**Challenge 4 – Zombie Challenge**

**Level 1: Goal** – Run in the field within 2 minutes, and ensure you won’t hit Zombies. Your robot must be able to reach at least once at all sides of the field, but without going out of bound.

![Diagram of zombie challenge level 1](image)

**Level 2: Goal** – Your robot will need to report distance to each zombie at each safety station. Beware, your robot must stay on the safety zone (the black line) along the whole way. Otherwise, your robot will be eaten up by the Zombie.

Each safe station, there is a silver mark. Your robot must report the distance of each zombie on the screen.

![Diagram of zombie challenge level 2](image)