Robot Projects for RobotBASIC
Volume I: The Fundamentals

Project 2: Monitoring the Battery

Project 1 showed how easy it is to control the movements of both the real and simulated robots. Such actions will, of course, deplete the battery's charge. For that reason, it is important that the robot have a way to monitor the state of its battery so it can notify you when it needs to be charged. After you have learned more about robot programming, future projects will explore the idea that a reasonably intelligent robot should be able to return to its charging station (all by itself) when it determines that such an action is necessary.

As with all our projects, we will explore the principles involved using the simulator before we test them on the real robot. Before we begin though, it is a good time to discuss the value of simulation. Simulation is a vital part of many industrial situations. Car manufacturers, for example, can create simulations of airflow around a car to determine how much wind resistance the car would have, *if it were built*. This allows them to experiment with a variety of shapes and designs without the time and expense of actually constructing their ideas to test in a wind tunnel. Simulations can be very accurate, but even if they are not perfect, they can provide engineers with information vital to the design process.

RobotBASIC's simulated robot has many sensors that we will explore and utilize. For our first encounter with sensors, we will examine the battery-monitoring sensor. Basically, this sensor allows us to obtain a number that is proportional to the battery voltage. Ideally, this number will be something close to 100 when the battery is fully charged and drop as the battery depletes, but the actual number is totally dependant on the battery used.

There are many different kinds of batteries and some can be discharged further than others. In general, the battery in the RobotBASIC Robot should be recharged when it reaches 80 or so. It is certainly possible to run the battery longer, but keeping the battery charged will extend its life. We can use the function **`rChargeLevel()`** to obtain a number indicating the battery's current condition.

Most statements in a computer program are either commands or functions. Commands, such as **`rForward`**, cause something to happen while functions *return a value*. Notice that functions end with parentheses. We can utilize functions just as we would any number. For example, we can make the value of the variable **x** equal to 50 with the following statement.

```
x = 50
```

We can set **x** to the charge level in the same way as shown below.

```
x = rChargeLevel()
```

We can print this value by printing x, or just printing the function itself.

```
x = rChargeLevel()
// both of these statements print the same value
print x
print rChargeLevel()
```

The program in Figure 2.1 shows how to obtain the charge level of the simulator's battery. It also moves the robot 30 times and reprints the charge level after each move to give you an indication of how the battery declines with use.

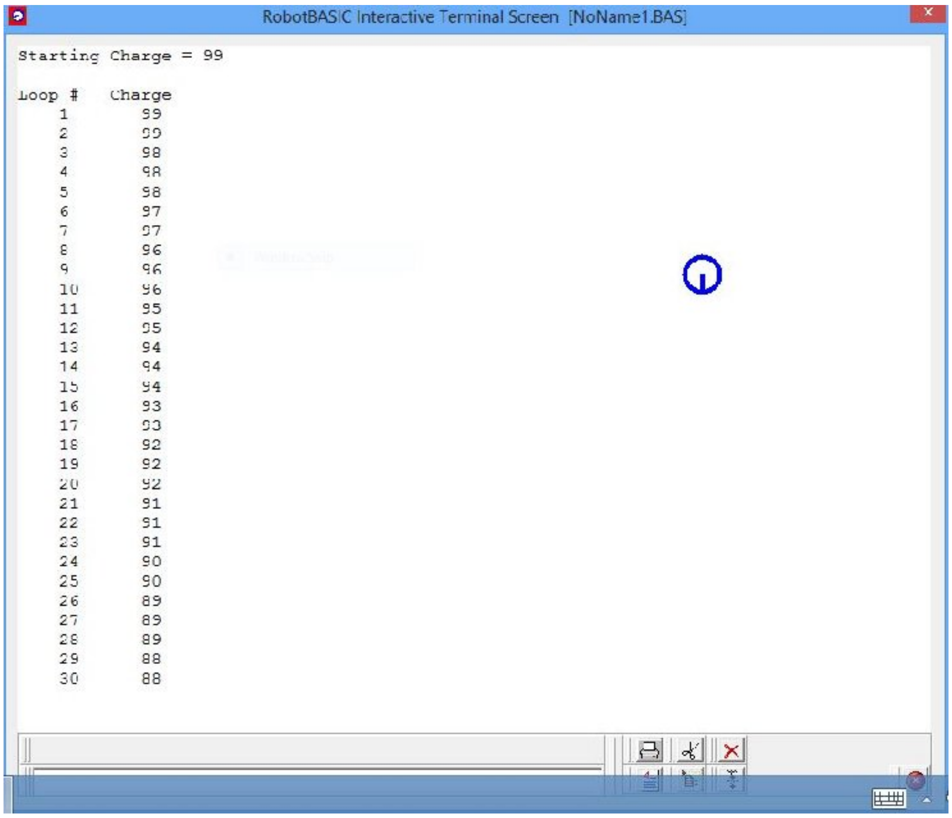
```
rLocate 300,500
rIgnoreCharge FALSE
//rCharge 10
x=rChargeLevel()
print "Starting Charge = ",x
print
print "Loop #   Charge"
for i=1 to 30
  rForward 300
  rTurn 90
  print "    ",i;rChargeLevel()
next
end
```

Figure 2.1: This program demonstrates how the battery can be monitored.

Notice that the first print statement in Figure 2.1 prints the value of **x** (the variable where the charge level was previously stored) and the second print statement prints the level directly. If you run this program, it produces the screen in Figure 2.2.

The second line in the program tells the robot not to ignore the charge level (ignoring is the default condition). Notice that the third line is commented out. Without this line, the battery automatically starts at a 99% charge. With the movements performed by the robot, the battery will have depleted to 88% after 30 times through the loop. Run the program to watch this happen in real time.

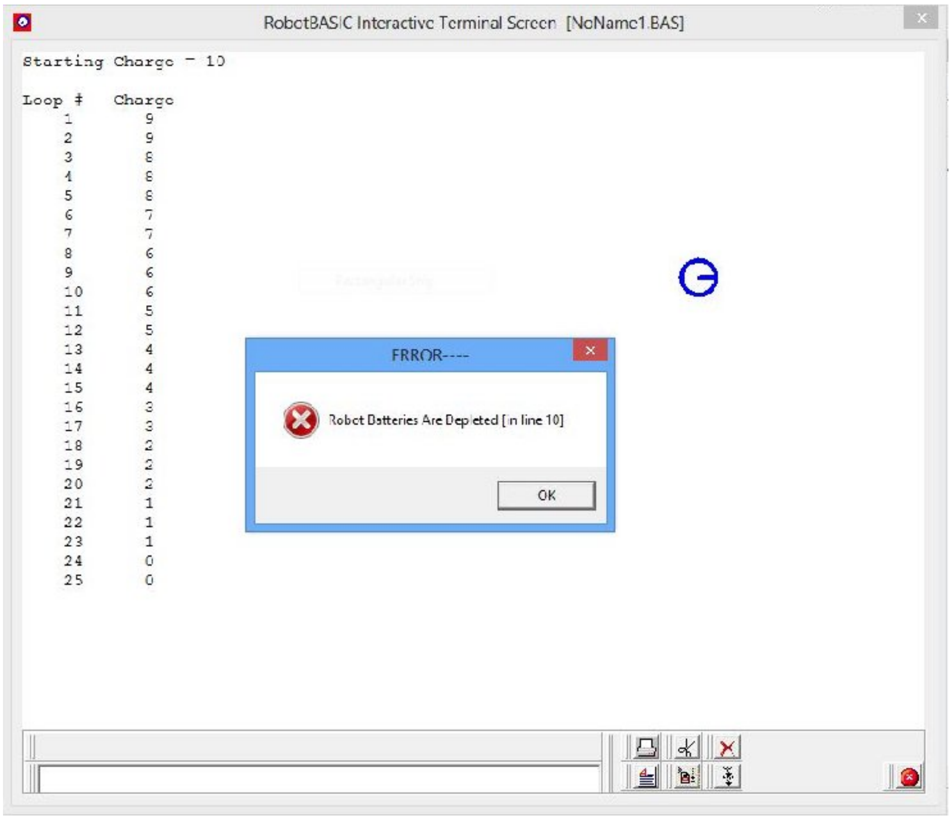
If you uncomment the third line the battery's initial charge will be set to 10%. With this low value the battery will deplete in a short amount of time. Run the modified program and you will get the results depicted in Figure 2.3. When the simulator's battery is *totally* depleted, the robot will quit moving and cause the error shown (unless the robot has been instructed to ignore the charge level). The real robot will fail at much higher levels, but this example shows how you can simulate various battery conditions. Future projects will program the robot to monitor its own battery level and stop its normal activity (whatever that may be) and make its way to a charging station to replenish the battery. Right now, such a program may seem impossible to achieve, but by the time we get to that project, you will have the knowledge to make it happen.



```
Starting Charge = 99

Loop #   Charge
   1       99
   2       99
   3       98
   4       98
   5       98
   6       97
   7       97
   8       96
   9       96
  10       96
  11       95
  12       95
  13       94
  14       94
  15       94
  16       93
  17       93
  18       92
  19       92
  20       92
  21       91
  22       91
  23       91
  24       90
  25       90
  26       89
  27       89
  28       89
  29       88
  30       88
```

Figure 2.2: After 30 movements and turns, the battery level is down to 88.



```
Starting Charge = 10

Loop #    Charge
    1         9
    2         9
    3         8
    4         8
    5         8
    6         7
    7         7
    8         6
    9         6
   10         6
   11         5
   12         5
   13         4
   14         4
   15         4
   16         3
   17         3
   18         2
   19         2
   20         2
   21         1
   22         1
   23         1
   24         0
   25         0
```

```
ERROR----
Robot Batteries Are Depleted [in line 10]
OK
```

Figure 2.3: An error occurs if the battery level becomes depleted.

Monitoring the battery in the real robot is just as easy as the simulation as shown in the program in Figure 2.4. Sometimes it makes sense to create programs that work for both the simulation and the real robot. In this example though, the program only works for the real robot.

```
#include "RB-9.bas"
gosub InitRROScommands
PortNum = 5 // set variable to your Bluetooth Port Number
gosub InitializeRealRobot
bat = rChargeLevel()
print "The current charge is ",bat
if bat>90
  print "The robot is ready for use"
elseif bat>80
  print "The battery is still good, but getting low."
else
  print "You should charge the robot"
endif
end
```

Figure 2.4: This program displays the condition of the real robot's battery.

Notice that the program in Figure 2.4 uses a different version of the **`IF`** structure. This version uses an **`elseif`** statement (all one word) to handle multiple conditions. The **`else`**-block will be executed if none of conditions in the **`if`** or **`elseif`** statements are true.

Limitations

All batteries are different, so you should change the values (80 and 90 in the example of Figure 2.4) to numbers appropriate for your battery. Older batteries, for example, might need charging much sooner than a new battery.

The real robot can perform erratically if used when the battery is low, so always keep the battery charged. Running the robot when the battery is extremely low can even damage the robot in some circumstances.

Suggestions for Study

A battery's voltage will drop when more current is drawn from it. Confirm that this is true by comparing the battery's charge level of a stationary robot to the charge level when the robot is moving.

> **Hint**: When you execute statements like **`rTurn 5`** or **`rForward 30`**, the real robot will move and stop before the next line in the program is executed. When using **`rTurn 1`** or **`rForward 1`** though, the robot to continue moving (for perhaps 200ms) even as the program continues. This means you could monitor the battery as the robot starts to move using the principle shown in the code fragment of Figure 2.5. Whenever a code *fragment* is shown, always be aware that some aspects of the code are missing. In this example, for instance, the robot has not been initialized.

Notice that the first reading in Figure 2.5 is taken while the robot is stationary. Subsequent readings are taken as the robot increases its speed. Try to guess how the readings will change before running the program. Compare the readings taken by this program with a fully charged battery compared with one that is only partially charged. Are the changes bigger or smaller with a fully charged battery? Can you speculate on why this is true?

```
print rChargeLevel()
for a=1 to 30
  rTurn 1
  print rChargeLevel()
next
```

Figure 2.5: This code fragment demonstrates how to monitor the battery as the robot starts to move.

The example in Figure 2.5 uses an **`rTurn`** instead of an **`rForward`** to generate the movement. Can you see the advantage of this approach?