Robot Projects for RobotBASIC
Volume I: The Fundamentals

Project 4: The Perimeter Sensors

In Project 1, the robot was moved around using the commands **`rForward`** and **`rTurn`**. Unfortunately, if the robot was commanded to move too far, it could run into an object and cause a collision with the real robot or an error with the simulator. Robots typically have sensors that allow them to examine the world around them. They might, for example, use a compass to know what direction they are facing or a ranging sensor to measure the distance to objects in its path.

Humans have many sensors too. Vision and an extensive sense of touch allow us to detect and identify objects. Expensive robots can have these abilities too, but don't expect most educational robots to sense things as well as a human, although some expensive research-based robots can outperform humans at selective tasks. RobotBASIC's simulated robot, as well as the RB-9, have numerous sensors that allow them to react to external situations in an appropriate manner. With proper programming, for example, a RobotBASIC robot can navigate through a cluttered room to find a doorway – without human intervention. Later projects will explore such situations, but for now we need to examine the fundamental principles of how sensors can be used to influence a robot's behavior.

Perimeter Proximity Sensors

In this project, we will examine perimeter sensors – that is sensors that detect objects around the perimeter of the robot (around its edges). The simulated robot actually has two types of perimeter sensors, but in this project, we will only examine the *feel* sensors.

There are five feel sensors around the front half of the robot as shown in Figure 4.1. The triangular regions extending from the robot in the figure indicate the areas where objects can be detected.

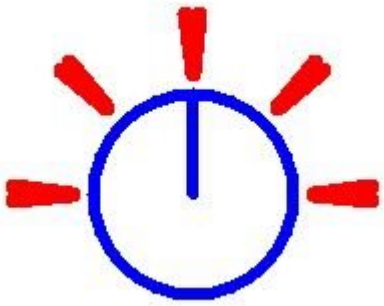

Figure 4.1: The robot has five perimeter proximity sensors.

These sensors can be read with the function **`rFeel()`**. Functions, in programming, have a value and we can extract that value and store it in the variable **`x`** as shown below.

```
x = rFeel()
```

Making Decisions

As we have seen in previous projects we can then use **`IF`** statements to test the value of a variable to determine what action we might want the robot to take. If the value of **`x`** is zero, then no objects are detected. When **`x`** is greater than zero, the value can be used to determine *which* of the five sensors are triggered. For this project, we are going to keep things simple and not worry about which sensors are seeing objects. Instead, we will just react if *any* of the sensors are triggered. The following code fragment, for example, will turn the robot around (180 degrees) if it detects an object in front of it (with any of the proximity sensors).

```
x = rFeel()
if x>0 then rTurn 180
```

We can use this principle to control our robot's movement. Let's look at a simple example to demonstrate this point. We can make the simulated robot move forward 50 pixels with the statement:

```
rForward 50
```

The problem with the above statement is that the robot will try to move the entire distance without checking to see if any obstacles are in its way. The following code fragment will cause the robot to *try* to move 50 pixels, but if an object is encountered before the distance has been transversed, the robot will stop.

```
for n = 1 to 50
  rForward 1
  x = rFeel()
  if x then break
next
```

If the loop continues till the end, the **rForward** command will be executed 50 times (once for each time through the loop). If, however, the sensors detect something, the **IF-THEN** structure will execute the **break** statement which will terminate the loop early with the program's execution continuing with the first statement following the **next**. It is worth mentioning that you can test the value of a function itself instead of storing its value in a variable. The fragment below does exactly the same thing as the previous one. Note: The **IF-THEN** structure is a simplified version of the **IF** that will execute the statement (on the same line) following the **THEN** only if the **IF** condition is true. In computer programming, any non-zero value represents a true condition so we can just use **if rFeel()** instead of saying **if rFeel()>0**, but either version is acceptable.

```
for n = 1 to 50
  rForward 1
  if rFeel() then break
next
```

We can use other types of loops to perform other actions. Both of the fragments below, for example, will continue to move the robot forward until it encounters a wall or other object.

```
repeat
  rForward 1
until rFeel()>0
```

```
while rFeel(0) = 0
  rForward 1
wend
```

Notice that one of these fragments will keep the robot moving *until something* is detected and the other moves the robot *while nothing* is detected. This may seem like a subtle difference but having a variety of looping structures often allows the programmer to organize the code in a more logical manner. If you want to try some of these *fragment* examples, do not forget to **rLocate** the robot at the beginning of your program.

We can use these principles to make the robot roam randomly around the screen and avoid objects that might cause a collision error. Look at the code in Figure 4.2. Note: the **IF** statement will be true if any of the **rFeel** sensors see an object because **rFeel** function will have a non-zero value. This statement would work equally well if you substituted either of the commented out options

```
rLocate 400,300
while true
  rForward 1
  if rFeel() then rTurn 180
  // both of the options below will also work
  // one tests for greater than 0, the other not equal to 0
  //if rFeel()>0 then rTurn 180
  //if rFeel()<>0 then rTurn 180
wend
```

Figure 4.2: This program makes the robot turn away from obstacles it encounters.

If you run the program in Figure 4.2, you might not be very impressed. The robot does indeed avoid objects, but since it always turns exactly 180° the action is very boring. You could make the program a little more interesting by making the robot turn only 140°. Modify the program and see which of the two options you like best. You could even utilize RobotBASIC's ability to generate a random number and make the robot turn a random number of degrees. Substituting the statement below, for example, will cause the robot to turn some random amount between 140 and 220 degrees.

```
if rFeel() then rTurn (140 + random(80) )
```

We can use commands like **Line**, **Circle** and **Rectangle** to create obstacles on the screen for the simulated robot to detect and avoid (in addition to the walls). These objects should be created at the beginning of the program before the robot is initialized. Modify the program in Figure 4.2 in this way and run it to verify that the robot performs appropriately. The modified program does give the robot a small amount of intelligence since it can avoid objects on its own. You will find though, that if you create an extremely cluttered environment, that the robot might make some mistakes and collide with an object or wall. In future projects we will examine ways to give the robot more information by letting it determine which sensors are actually being triggered. This will allow the robot respond more appropriately to objects within its environment. For example, if the robot detects an object only on its left, it might make sense to turn to its right.

A Roaming Real Robot

The program in Figure 4.2 can also be used to control the real robot by modifying it as shown in Figure 4.3. Notice that a couple of subroutines have been added to make the

program more organized and easier to read. Using subroutines in this manner is not a requirement, but it is highly recommended.

```
#include “RB-9.bas”
gosub InitRROScommands

Real = TRUE // set to FALSE to use simulator
PortNum = 5 // set this variable to your Bluetooth Port
Number

if Real
  gosub InitializeRealRobot
else
  gosub InitializeSimulator
endif
gosub Roam
end

InitializeSimulator:
  // draw objects in your room here (add more)
  circle 150,200,300,300,BLACK,BLACK
  rLocate 400,300
return

Roam:
  while true
    rForward 1
    if rFeel()>0 then rTurn 180
  wend
return
```

Figure 4.3: This version of Figure 4.2 can control either the real or simulated robot depending on the value of the variable **`Real`**.

Technical Information

The perimeter proximity sensors for **`rFeel()`** are implemented on the real robot with ultrasonic waves (sound waves slightly above the human hearing range). Recall that five ultrasonic sensors are mounted around the front half of the RB-9 robot as discussed in Project 3 which explained how they can measure the distance to objects.

Sometimes we need a way to quickly check to see if objects are close the robot and we don’t want to take the time to read all five of the ranging sensors and analyze their values. The RROS system inside the RB-9 robot constantly gathers the ranging data and

automatically transfers a summary of that information back to RobotBASIC every time your program tries to move the robot with an **rTurn** or an **rForward** statement. RobotBASIC provides that information to the user with the **rFeel()** statement, making it easy to determine when objects are close to the robot. Future projects will explain how to determine exactly which of the five sensors are seeing an object. In order to keep things simple for now though, we will just assume that any non-zero value means that at least one of the sensors has detected an object within range.

Limitations

It is possible for an ultrasonic wave to be absorbed by soft objects (such as a stuffed animal) or to be reflected away from the robot (instead of back toward it), if the surface of the object is not perpendicular to the wave's direction. These situations can sometimes cause faulty readings. If the robot is constantly monitoring the sensors though, then even if one reading is faulty the next reading may be fine. This means that a robot will usually respond properly as long as it moves only small amounts before obtaining new data from its sensors. Future projects will examine these problems in more detail and offer a variety of solutions.

Suggestions for Study

Try all the programs in this Project and modify them as you see fit. You might, for example, try making the robot turn different amounts and explain why you prefer one over another. If you have a RobotBASIC robot available, test your ideas with it to verify that it behaves in a similar manner to the simulation.

There is a special form of the **rFeel()** statement called **rDfeel()**. The **D** stands for debugging, as this command can sometimes help you determine why a program is not performing as you expect. The operation of the debugging version is very similar to the original function, but the simulation will draw very faint lines showing you the area being viewed by each sensor. When you use **rDfeel()**, you must provide a color like this:

```
rDfeel(RED)
```

The sensor lines will be drawn in the specified color, and that color will automatically be viewed as an Invisible color so that it does not appear to be an obstacle to the robot. Substitute the new command into your simulator programs and verify that the robot turns whenever an obstacle comes within range of the sensors. Notice that the robot moves slower when using the new command as it must perform a lot of calculations to draw the lines. For that reason, you should generally use **rFeel()** to read the proximity sensors. Future projects will explore the use of **rDfeel()** in more detail.