Robot Projects for RobotBASIC
Volume I: The Fundamentals

Project 5: Remote Control

In earlier Projects, we examined how to move the robot with **`rForward`** and **`rTurn`**, and how to read and utilize the **`rFeel`** sensors. In this project we will learn to control the robot's movements with the keyboard and the mouse. This gives the user control over the robot, but we would like to prevent them from driving the robot into an obstacle. For that reason, we will even give the robot a little intelligence so that it can refuse to obey instructions that might cause a collision.

Reading the Keyboard

Before we actually work with the robot, we need to understand how to determine when someone presses a key on the keyboard. Normally we could use an **`INPUT`** statement to capture whatever is typed at the keyboard, but that will not do in this case because the **`INPUT`** statement requires the user to press the **`ENTER`** key when they are finished typing. Instead, we would prefer the robot to respond to single key-strokes. Note: If you would like to learn more about INPUT, refer to the RobotBASIC HELP.

Look at the program in Figure 5.1. The RobotBASIC statement **`getkey`** watches the keyboard and places a zero in the variable specified (**`k`** in this case) if no key is pressed. If a key is pressed though, it assigns the code for that key to the variable specified. The **`print`** statement prints the value of **`k`**, so you can see it. If you just hold a key down, the program will continue to print the value until the key is released (at which time it will start printing zero's).

```
while TRUE
  getkey k
  print k
  delay 100
wend
```

Figure 5.1: This program prints the codes used to represent keys.

Enter the program in Figure 1 and run it. You will see zero's being printed down the left hand side of the screen – when the bottom is reached, the screen will automatically clear and the printing will start again at the top of the screen. If you press the letter **`a`**, for example, you will see 97 print. Similarly, **`b`** will print 98, **`c`** will print 99, etc. If you use the upper-case letters, they will be 32 less than the codes for lower-case letters (**`A`** will be

65). These codes are called ASCII codes and are used internally on nearly all computers. There are ASCII codes for all standard numbers, letters, and symbols.

If we use the principle shown by Figure 5.1, we could write a program to control the robot's movements. For example, we might have the robot move forward when we press the **`F`** key, turn left when we press the **`L`** key, etc. This would work, but it might be a little more natural if the user could use the arrow keys (the up-arrow could be forward, the right-arrow could turn right, etc.). If we try pressing the arrow keys with the program in Figure 5.1, though, it just prints zeros. This happens because the arrow keys are not considered to be standard keys and therefore do not have assigned ASCII codes (early computers did not have arrow keys).

If we use the statement **`getkeye`** instead of **`getkey`**, the specified variable will be assigned an *extended* code. This will let you detect when the arrow keys are pressed as well as any other key on your keyboard (such as arrow keys or even the shift key by itself). Some of the codes for normal letters will be standard ASCII, but others (capital letters for example) will be different. For our purposes we only need the codes for the arrow keys. If you modify Figure 5.1 to use **`getkeye`**, and press the arrow keys you will see that they produce the following codes.

left-arrow	37
up-arrow	38
right-arrow	39
down-arrow	40

Controlling the Robot

Figure 5.2 shows how we can use this information to build a program that allows the user to control the simulated robot with the arrow keys. Notice the delay in the program has been shortened to 10ms to let the robot move at an appropriate speed. If you find your robot is moving too fast or too slow, adjust the amount of delay.

```
rLocate 400,400
while true
  getkeye k
  if k=37 then rTurn -1
  if k=38 then rForward 1
  if k=39 then rTurn 1
  if k=40 then rForward -1
  delay 10
wend
```

Figure 5.2: This program allows the user to move the robot with the arrow keys.

If you test the program in Figure 5.2 for a while, you will see that you can cause an error by driving the robot into a wall (or other obstacle you draw on the screen). Since a computer is controlling our robot, we should expect it to be smart enough to avoid obstacles even if we are dumb enough to cause an accident (many new cars will now warn you or even apply the brakes for you if you try to drive into something). We can implement a simple version of this idea by replacing the **IF** statement controlling the **`rForward 1`** command in Figure 2 with the following statement.

```
if (k=38) and (rFeel()=0) then rForward 1
```

The above statement will only move the robot forward if **`k`** is equal to 38 *and* the feel sensors are all zero. Notice that each of the two parts of the decision parameter are enclosed in parenthesis. This is not always necessary, but using the parenthesis helps clarify the meaning for humans and ensures that computer performs the intended action. Try the program and you will see that in nearly all situations the program prevents you from causing a collision by driving the robot forward. Future projects will demonstrate better ways to achieve this goal, but for now this works reasonably well.

Using the Mouse

Keyboard control is nice, but using the mouse could be even better. The RobotBASIC command **`ReadMouse x,y,b`** reads the current status of the mouse and places the screen coordinates of the mouse's current position into the variables **`x`** and **`y`** (any variable names can be used). It also places a number in the variable **`b`** that indicates the status of the mouse buttons. A zero indicates no buttons are pressed while a 1 indicates the left button and a 2 indicates the right. The program in Figure 5.3 demonstrates this idea.

```
while true
  readmouse x,y,z
  print x;y;z
  delay 100
wend
```

Figure 3: This program shows how to read mouse information and display it.

If you run the program in Figure 3 and move the mouse around or push the left or right buttons, you will see the numbers change. It is hard to observe these numbers though, because they are continually scrolling down the screen. To solve this problem, replace the print statement in the program with the following line. It allows printing at a specific position so that new values just replace old ones.

```
xyString 10,10,x;y;z
```

The 10,10 in the above line specifies the **`x,y`** position for the printing. The rest of the statement is just like the parameters in the **`print`** statement. Remember, if you are not familiar with any programming statements just look them up in the HELP system. Modify the program in Figure 5.3 and run it. As you move the mouse or press buttons, you will easily see the changes in the numbers being displayed on the screen. Knowing the **`x,y`** position of the mouse and the status of the buttons can be used in a variety of ways to help us control the robot. We could, for example, make the robot turn left when the mouse is on the left side of the screen. This could work, but let's try something a little more creative. The program in Figure 5.4 will produce the shape shown in Figure 5.5.

```
Rectangle 300,500,500,530,GREEN,GREEN
Rectangle 300,560,500,590,BLUE,BLUE
Rectangle 300,530,375,560,RED,RED
Rectangle 425,530,500,560,LightRED,LightRED
```

Figure 5.4: This program produces the shape shown in Figure 5.5.

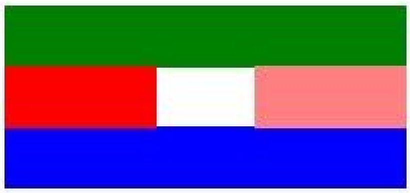

Figure 5.5: This diagram can allow us to control the robot with the mouse.

We can use the diagram in Figure 5.5 to control the robot. If the mouse is over the green area, for example, we could have the robot move forward. Moving the mouse over the blue area could request the robot to backup. Likewise, the red and light red areas could be used to turn the robot left and right. To perform these actions, we need a way to read the color at the mouse cursor position. The command **`ReadPixel x,y,c`** can do that for us. It will read the color of the screen at position **`x,y`** and place the value of that color in the variable **`c`** (again, any variables can be used).

Figure 5.6 shows a complete program that allows the mouse to control the robot's movements. If you run the program, you will find that you can control the robot by moving the mouse over the appropriate areas in the diagram.

```
Rectangle 300,500,500,530,GREEN,GREEN
Rectangle 300,560,500,590,BLUE,BLUE
Rectangle 300,530,375,560,RED,RED
Rectangle 425,530,500,560,LIGHTRED,LIGHTRED

rLocate 400,400
while true
  readmouse x,y,b  // find out where the mouse is
  readpixel x,y,c  // read the color where the mouse is
  if c=RED then rTurn -1
  if c=GREEN then rForward 1
  if c=LightRED then rTurn 1
  if c=BLUE then rForward -1
  delay 10
wend
```

Figure 5.6: This program controls the robot by moving the mouse over different areas of the shape shown in Figure 5.5.

Suggestions for Study

Modify the program of Figure 5.6 so that the robot will not cause a collision even if you try to move it into a wall or obstacle. Verify that it works as desired.

Chose either the program in Figure 5.2 or the program in Figure 5.6 and modify it so that it can control the RobotBASIC Robot in addition to the simulation. It is suggested that you make use of subroutines to ensure that your final program is well organized and easy to read.

More advanced users might want to improve on the mouse version of this program. In the current version, the robot either moves forward or backward in a straight line or it rotates left and right. It is possible to make the robot move in an arc using both **`rTurn`** and **`rForward`** statements as demonstrated by the code in Figure 5.7.

```
rLocate 400,400
while true
  rTurn 1
  rForward 1
wend
```

Figure 5.7: This short program demonstrates how to move the robot in an arc.

You can change the arc produced by Figure 5.7 by adding more **`rTurn`** Statements or more **`rForward`** statements. If the GREEN box in Figure 5.5 had ends that were different colors you could modify the program so that moving the mouse to those areas would make the robot turn gently left and right. Try to modify the program in this way.