

Robot Control Using Genetic Programming

By: Abraham L. Howell

Abstract:

This paper will discuss a software application named “Abe Howell’s GP Robot Control”, which utilizes genetic programming (GP) to evolve a suitable obstacle avoidance program for a low-cost mobile robot equipped with Bluetooth® connectivity. What is unique about this application is the fact that genetic programming evolves the optimum program while controlling the low-cost robot wirelessly. The topics to be covered include the low-cost robot and Bluetooth modules, overall architecture of the application, how GP was implemented, and how to use the software.

Low-cost Robot and Bluetooth Modules:

The low-cost robot used in this project is the NewCDBot, which is an educational robot kit supplied by the author’s company, Abe Howell’s Robotics¹. The NewCDBot was chosen due to its low cost of \$85, ease of use and programming, and ultra-compact footprint, which is approximately the size of a compact disc. Bluetooth technology was chosen to create the wireless connection between the NewCDBot and laptop due to its relatively low cost and minimal power requirements². An EB500 Transceiver is installed on the NewCDBot and an Orange-Micro USB Bluetooth dongle on the laptop³. A picture of a NewCDBot with an EB500 Transceiver is shown below in figure 1.

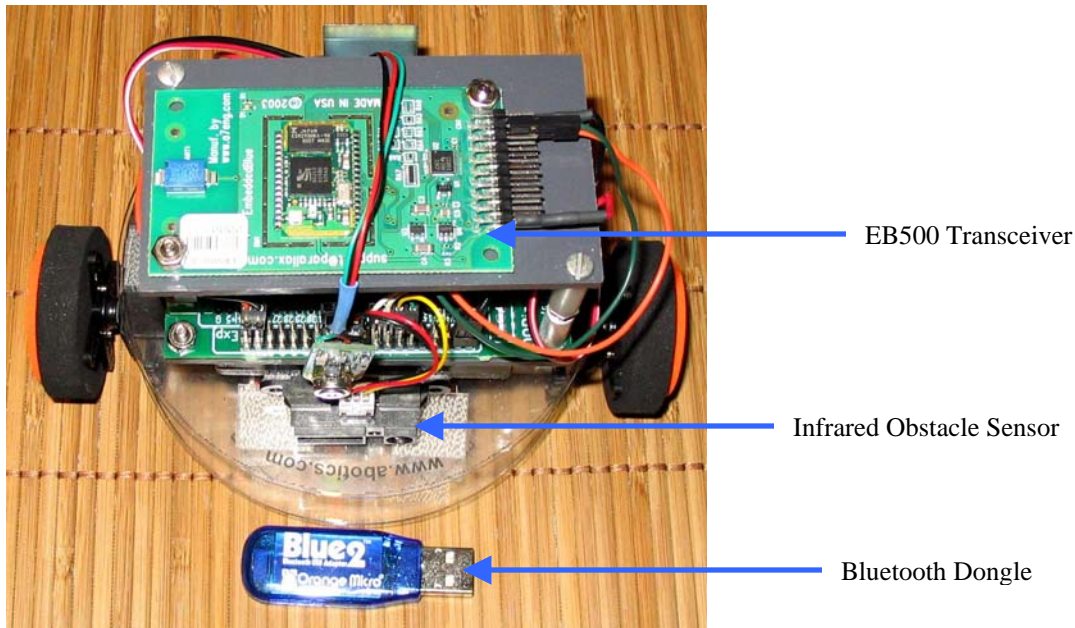


Figure 1. NewCDBot with EB500 Transceiver.

A Sharp GP2D120 infrared obstacle sensor is installed on the NewCDBot and provides an 8-bit value, which corresponds to the detection distance. The 'Brains' of the NewCDBot is an OOPic II+ micro-controller⁴. The OOPic II+ was utilized due to its object orientated programming, but also because it support Serial Control Protocol (SCP). SCP allows the robot to be reprogrammed on the fly and controlled wirelessly as well.

Overall Application Architecture:

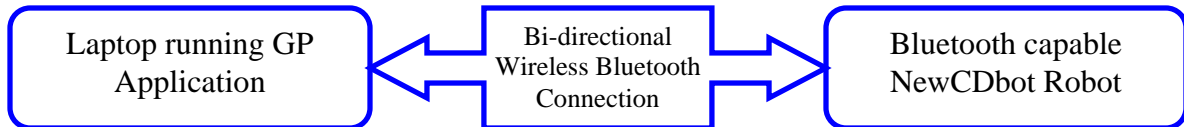


Figure2. Overall application architecture.

As can be seen from the figure2 diagram above the laptop has a bi-directional wireless connection with the robot. Therefore, the laptop and the robot can both transmit and receive data from one another. Transmitted and received data is limited to 8-bit packets, but is more than sufficient for this application. If larger packets of data need to be sent a simple packet scheme can be devised. Basically, the GP application hooks onto a Virtual Bluetooth Serial Port to communicate with the robot. I will not go into exactly how this connection is created, but will supply the following web link for additional edification: http://www.abotics.com/buildit_yourself.htm Once this connection has been established the GP application can control and request sensor readings from the robot. To accomplish this task a simple polling program has been created and uploaded to the robot. This polling program basically monitors the serial port for requests or commands from the GP application and then acts upon them as they are received. Below is snippet of code from the polling program.

```

Do
  If TxRx.Received=cvTrue then
    rcvbuff=TxRx

    Select Case rcvbuff.Value
    Case 97 'a
      TxRx=FrontEye

    Case 98 'b
      Back
      OOPic.Delay=100
      STOP
      TxRx.String="d"

    Case 102 'f
      FWD
      OOPic.Delay=100
      STOP
      TxRx.String="d"

    Case 108 'l
      SpinLeft
      OOPic.Delay=25
      STOP
      TxRx.String="d"
  
```

```

Case 114 'r
  SpinRight
  OOPic.Delay=25
  STOP
  TxRx.String="d"
End Select
End If
Loop

```

As can be seen from the above code-snippet the robot reads in an 8-bit value and then uses a Select Case method to determine the appropriate action to be taken. Obviously, there is additional code that wasn't shown in the above snippet and this code would contain the motion functions and initialization routines. The full code listing will be provided at the end of the paper and will be included in the GP source-code.

GP Implementation:

The entire GP application was coded in Visual Basic.Net® except for the polling program, which was created using the OOPic Multi-Language Compiler IDE. A screenshot of the application's main form is shown below in figure3.

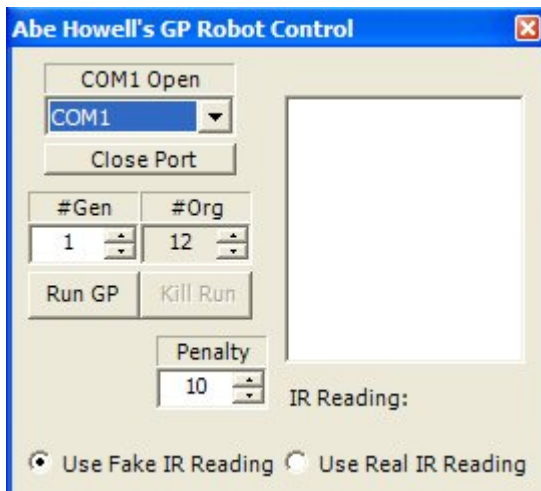


Figure3. GP application screenshot.

The above screenshot is the only means of running the GP process and evolving an optimum obstacle avoidance program for the robot. You have the ability to change the number of generations, number of organisms, backup penalty, and whether to use a fake or real sensor reading. The actual use of the application will be discussed in the next section.

I. Genetic Population

The genetic population for this application consists of linear programs, which have the following terminal and function set.

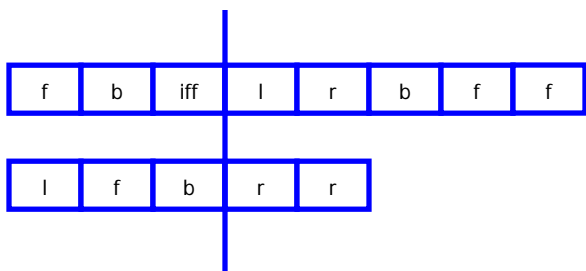
T={FrontEye} Fnc={Forward, Left, Right, Back, IF obstacle in front}

Each of the linear programs represents an organism in the population and can have a minimum length of 2 and a maximum of 10. The initial population is created using the ramped half-half method, whereby programs can vary in length and are randomly generated. The built-in Rnd() function is used to generate all the required random numbers. Once the initial population has been created the actual GP process can commence.

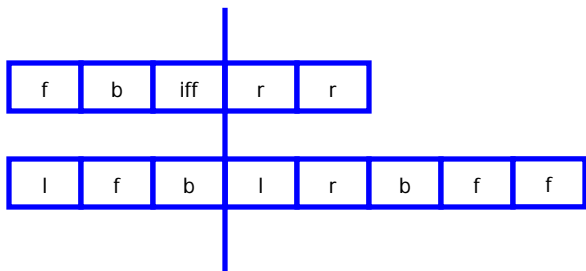
II. Executing the GP process:

During the GP process organisms will be run and evaluated in 4-organism tournaments. A single generation will consist of looping through the entire population two at a time starting with organism 1 and ending with the last two organisms. The tournaments are run such that two of the four organisms are picked sequentially as described above, but the remaining two organisms are randomly chosen. During the actual tournament each of the organisms will be executed and evaluated on the robot one by one. At the end of a single tournament the two best organisms will be copied into the two worst. Next, the two worst organisms will undergo crossover and finally mutation. The crossover process is illustrated below in figure4.

Organisms before crossover.



Organisms after crossover.



The crossover point is randomly chosen and is a number from 1 to the length of the shorter organism. This helps to simplify the crossover process, but is still effective. After crossover is complete, each organism undergoes mutation. A random number between 1 and the organism length is generated and used to pick the program instruction to mutate. Once the mutations are complete, each of the newly created organisms is copied into the population and the GP process continues by copying the best of the four organisms into the Best of Generation (BOG). The best of tournament organism will get copied into the BOG only if it has a lower score than the previous BOG. This means that the BOG could actually be found in generation one even though a total of 10 generations might be

executed. The score for an organism is based on the final obstacle sensor reading and backup penalty. The backup penalty was implemented to try and force the evolution of programs that would favor moving the robot forward rather than backwards. This unfavorable phenomenon could have been avoided if an additional sensor were installed on the rear of the robot.

III. Simplified Organism Evaluation:

Typically, GP applications are coded using List Processing (LISP) syntax. LISP is chosen for its built-in program evaluation capability. If programming languages other than LISP are used for GP, the program evaluation process is completely left up to the developer and can be a rather arduous and unwelcome task. This application has been coded in Visual Basic.Net and overcomes this technical hurdle in a unique way. Since each of the programs are directly run and evaluated on the robot in real-time using a wireless connection, there isn't a need to deal with creating a means for evaluation the code. Additionally, this problem is even further simplified since a simple polling program has been implemented on the robot and provides control using simple commands. The use of a polling program also offloads the computationally intensive tasks to a more than capable laptop or desktop pc.

How to use the software:

This section will cover how to make use of the software GUI. There are two ways to run the actual GP process. The first is an offline method and basically generates fake sensor readings so that the program can be run exclusive of an EB500 equipped NewCDbot and USB Bluetooth dongle, but will obviously generate false results. The second method requires all the necessary hardware and will evolve an appropriate obstacle avoidance program. I will explain the steps required to run the program using fake sensor readings. First, select the "Use Fake IR Reading" option and a valid COMPORT as shown below in figure5.

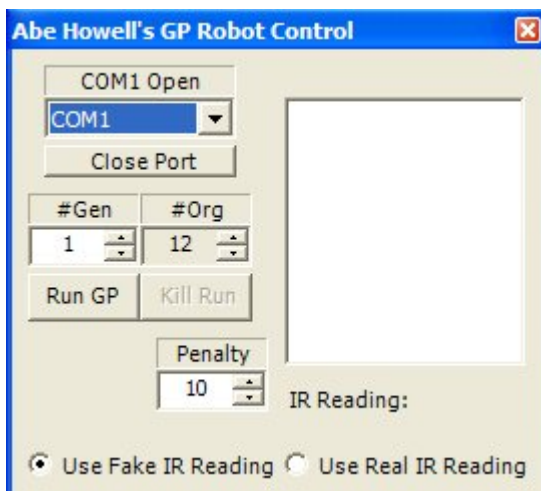


Figure5. Select "Use Fake IR Reading."

For this simple illustration we'll leave the number of generations at one and the set the number of organisms to four as shown in figure6. We'll also leave the backup penalty value at 10.

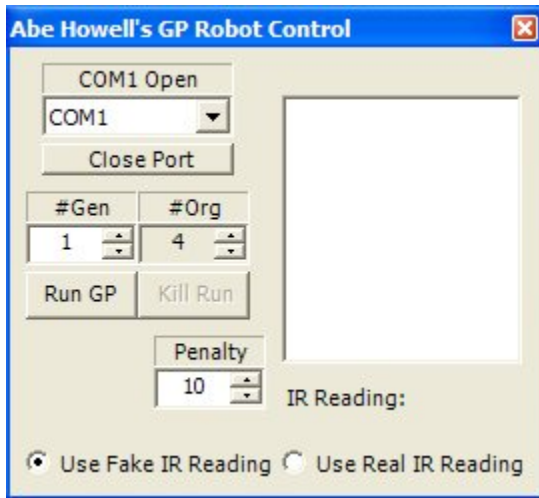


Figure6. Set number of organisms.

We are finally ready to click the “Run GP” and actually run the GP process. After clicking the “Run GP” button you’ll notice that the “Kill Run” button has become enabled and will allow you to terminate a run. The process should start running and evaluating the programs one by one live. Once the run is complete you should be left with the BOG as shown below in figure7.

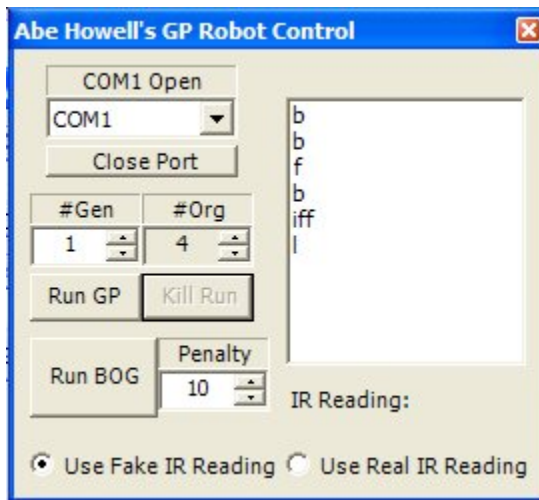


Figure7. Final BOG result.

It can be seen that even though this run only went through one generation with four organisms using fake sensor readings it still produced a somewhat reasonable result. If this program were uploaded to the robot and run it would cause the robot to drive backwards for 1 second, go forward for 0.5 seconds, go back for 0.5 seconds, and check the front IR sensor and spin left for 0.5 seconds if blocked. You can play around with the number of generations/organisms and observe the results.

Conclusion:

Even though this application uses simple linear programs with a maximum length of 10 it can be observed that the results are successful and provide some insight into the use of GP for evolving robotic artificial intelligence. Another benefit of this project is its ultra-low cost, which is well below \$250. A brief discussion regarding a similar GP application can be found in a book by John Koza. The discussion in Koza's book is actually the foundation for this project. The robot used in Koza's discussion is a Khepera robot, which is valued at approximately \$3000. This project is roughly 12 times less expensive than the Khepera implementation, but still accomplishes the same goal of evolving an obstacle avoidance program. Future work on this project might include using a more complex program structure and even uploading entire programs to the robot for evaluation purposes instead of using the simple polling program.

Bibliography

1. Howell, Abraham L., NewCDBot Educational Robot Kit, 2003, Abe Howell's Robotics, http://www.abotics.com/abotics_newcdbot_kit.htm
2. Bluetooth Technology, 2003, www.bluetooth.com
3. Parallax Inc., EB500 Transceiver, 2003, www.parallax.com.
4. Savage, Scott. OOPic Multi-Language Compiler. Savage Innovations Inc, 2003 <http://www.oopic.com>
5. Koza, John, "Genetic Programming. On the programming of computers by means of natural selection.", MIT Press, MA, 1992.
6. Koza, John, "The Genetic Programming Paradigm", Wiley, 1992.

Simple Polling Program Listing:

'The Following Code Was Developed for the NewCDBot, but
'can be used for any robot that utilizes the OOPic
'micro-controller and the specified sensors.
'Written by Abraham L. Howell
'If there are any errors or comments please send to:
'(abe@abotics.com)

```
Dim LeftServo as new oServo
Dim RightServo as new oServo
Dim FrontEye as new oA2D
Dim LeftSrvStp as new oByte
Dim RightSrvStp as new oByte
Dim speed as new oByte
Dim TxRx as new oSerial
Dim rcvbuff as new oByte
```

'Sub Routine in which Infrared sensors, Servos, bumper switches,
'Temperature sensor, and E1 EEPROM are Configured
Sub Init()

```
    'Turn on OOPic Pullup resistors
    Oopic.Pullup=cvTrue
    'Set Voltage Reference
    Oopic.ExtVRef=0
```

```
    'Connect FrontEye to I/O Line#3(Pin#11)
    FrontEye.IOLine=4
    'Turn on FrontEye
    FrontEye.Operate=cvTrue
```

```
    'Connect LeftServo to I/O Line#5(Pin#15)
    LeftServo.IOLine=5
    'Connect RightServo to I/O Line#6(Pin#17)
    RightServo.IOLine=6
    LeftSrvStp=53
    RightSrvStp=53
    speed=5
```

```
    'Setup TxRx Baudrate
    TxRx.Baud=cv9600
    TxRx.Mode=0
    TxRx.Operate=cvTrue
```

End Sub

'Sub That Makes Robot Go Forward
Sub FWD()

```
    LeftServo=LeftSrvStp-speed
    RightServo=RightSrvStp+speed
    LeftServo.Operate=cvTrue
    RightServo.Operate=cvTrue
```

End Sub

'Sub That Makes Robot Go BackWard

```

Sub BACK()
    LeftServo=LeftSrvStp+speed
    RightServo=RightSrvStp-speed
    LeftServo.Operate=cvTrue
    RightServo.Operate=cvTrue
End Sub
'-----
'-----
'Sub That Makes Robot SpinLeft
Sub SpinLeft()
    LeftServo=LeftSrvStp-speed
    RightServo=RightSrvStp-speed
    LeftServo.Operate=cvTrue
    RightServo.Operate=cvTrue
End Sub
'-----
'-----
'Sub That Makes Robot SpinRight
Sub SpinRight()
    LeftServo=LeftSrvStp+speed
    RightServo=RightSrvStp+speed
    LeftServo.Operate=cvTrue
    RightServo.Operate=cvTrue
End Sub
'-----
'-----
'Sub That Makes Robot Stop
Sub STOP()
    LeftServo.Operate=cvFalse
    RightServo.Operate=cvFalse
End Sub
'-----
'-----
'Sub That Initializes and Runs the Robot
Sub Main()

Init

LeftSrvStp=53
RightSrvStp=53

OOPic.delay=100
RunRobot

End Sub
'-----
'-----
'Sub that Runs the Robot with the User's Own Code!
Sub RunRobot()
    speed= 10
Do
    If TxRx.Received=cvTrue then
        rcvbuff=TxRx

        Select Case rcvbuff.Value

            Case 97 'a

```

```
TxRx=FrontEye

Case 98 'b
  Back
  OOPic.Delay=100
  STOP
  TxRx.String="d"

Case 102      'f
  FWD
  OOPic.Delay=100
  STOP
  TxRx.String="d"

Case 108      'l
  SpinLeft
  OOPic.Delay=25
  STOP
  TxRx.String="d"

Case 114      'r
  SpinRight
  OOPic.Delay=25
  STOP
  TxRx.String="d"

End Select

End If

Loop
End Sub
```