CS295/CS395/CSYS395: Evolutionary Robotics

Programming Assignment 10 of 10

Assigned: Monday, November 4, 2011

Due: Monday, November 11, 2011 by midnight

Description: In this assignment you will integrate the evolutionary algorithm you developed in Python with the robot simulator. This requires six steps:

1. From the Python code copied from assignment 3, output a matrix of synaptic weights to a text file.
2. When the ODE code starts, it creates the robot, reads in this matrix, and sets the synaptic weights from the sensors to the motors.
3. The ODE code runs for 1000 time steps, after which the robot's displacement from the origin is written into a text file.
4. This number is read in by the Python code, and assigned as the fitness of the parent synaptic weight matrix.
5. A mutated, child synaptic weight matrix is created and sent to the ODE simulation.
6. In ODE the robot's position is set back to the origin, the new synaptic weights are read in, and so on.

These steps are to be realized as follows.

1. Back up Assignment_9 on a flash drive or another computer so that you can always return to your completed eighth assignment.
2. Copy directory Assignment_9, which contains your submitted document and the entire ODE folder. Rename the new directory Assignment_10.
3. Copy the Python code you created in assignment 3 into this new directory.
4. In the Python code, change the dimension of the synaptic weight matrices so that they are 4×8, and ensure that the values range in $[-1, 1]$.
5. After the parent matrix is created, write out the matrix to a text file called `weights.dat`. Make sure this file is stored in the directory where your ODE simulation will run. This directory is usually `ode-0.11.1\lib\debugdoubledll` on a Windows machine, but will be different on other platforms.

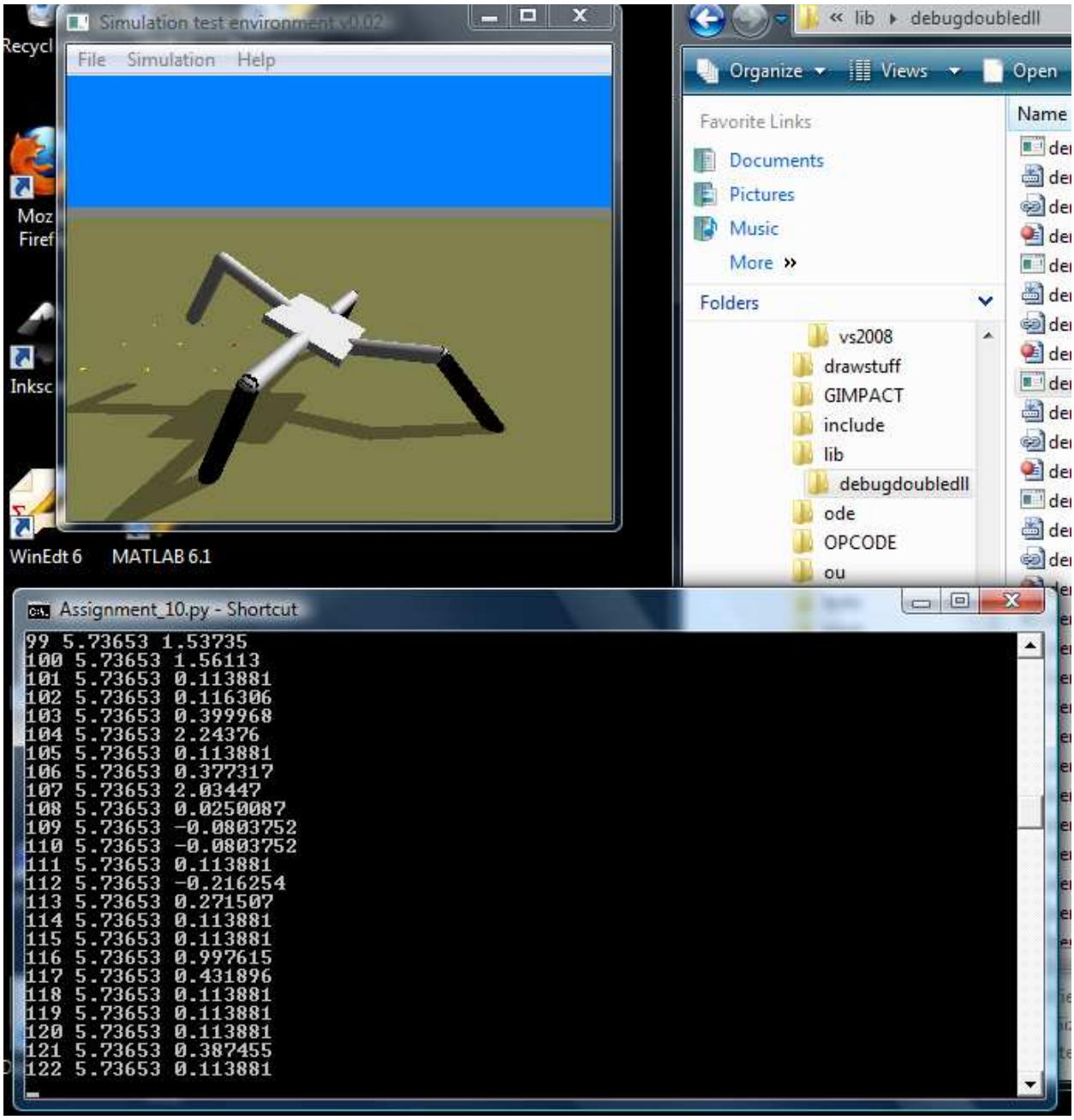

Figure 1: Display of the robot's behavior (graphics window) and the evolutionary algorithm running in Python (text window).

6. The Python code should then periodically check for the existence of the second file, which will report the fitness of the parent matrix. You can implement this using the `path.os.exists` and `sleep` Python functions:
```
fileName = 'fit.dat';
while ( os.path.exists(fileName)==False ):
.    sleep(0.2);
```

7. Once this part of the Python code is working, set it aside. In the ODE simulation, create a new variable `timer` that is set to zero, and incremented during each pass through `simLoop` when it is unpaused.

8. When the timer reaches 1000—that is, 1000 iterations through `simLoop` have been performed—set the timer to zero again. Also, reset the robot by destroying all of its body parts and joints, and recreating them at the origin:
```
internalTimer++;
if ( internalTimer == 1000 ) {
.  internalTimer = 0;
.  DeleteRobot();
.  CreateRobot();
}
```

9. Run the ODE code; you should see the robot performing the same behavior repeatedly for 1000 time steps. This is because you are not resetting the synaptic weight matrix yet.

10. Add an additional line to `CreateRobot()`
`WeightsRead()`
that looks for the text file `weights.dat`. Once it exists, it reads in the values and stores them in the matrix `weights` that you created in the last assignment. Finally, this function deletes the file. Check any online C tutorial to learn how to check whether a file exists, and how to delete it. When you run the ODE simulation now, it should simulate the robot once, and then freeze, because when it gets to the `if` clause in `simLoop` when `CreateRobot` is called for the second time, there is no file for it to read in.

11. Now add an additional line that calls a function `Fitness_Save()` to the `if` clause
```
internalTimer++;
if ( internalTimer == 1000 ) {
.  internalTimer = 0;
.  Fitness_Save();
.  DeleteRobot();
.  CreateRobot();
}
```
This function should get the position of the robot's main body using
`dBodyGetPosition()`
(see the ODE documentation). It should take the horizontal component of this final position, and write it to a text file `fit.dat`.

12. Run your ODE simulation, and make sure that it stores a single number in `fit.dat`. Now, update your Python code so that when the file `fit.dat` exists, it reads in the value, stores it in `parentFit`, and deletes the file. This can be done in Python using `os.remove('fit.dat');`

13. In the Python code, once `parentFit` has been set and `fit.dat` has been deleted, a child matrix can be created, and its values stored in `weights.dat`. The Python code should then pause again until the fitness of the child matrix is returned by the ODE simulation. When it is, the Python code should print the current generation, the fitness of the parent, and the fitness of the child.

14. Now you should be able to run your Python code, and, while it is running, start up your ODE code. Make sure that the following actions occur in the right order: (1) the Python code outputs a weight matrix; (2) the ODE simulation reads in and deletes this matrix; (3) the ODE code simulates the robot's behavior while the Python code waits; (4) the ODE code saves the robot's fitness; (5) the Python code reads in this fitness and deletes the file; and (6) the Python code writes another synaptic weight matrix.

15. If your two applications communicate with one another correctly, you should see something like in Fig. 1. The graphics window shows the behavior of the robot using the current synaptic weight matrix, and the text window corresponding to the Python code outputs the results of each 'generation'.

16. Make sure to do a screencapture that captures both your robot in action and the results of your evolutionary algorithm after a few generations. The T.A. will look to verify two aspects of your screenshot. First, that when the child has a higher fitness than the parent (the third number in a row is higher than the second number in a row), the child becomes the parent (the third number in row i becomes the second number in row $i + 1$. Second, make sure to capture one of the better of the child solutions moving to the right: if you select for the horizontal component of the robot's final position, the robot should start to move to your right. Note how in Fig. 1 the robot is to the right of the dots at the left of the graphics window (these dots indicate the $(0, 0, 0)$ origin).

17. Do a screencapture until you get such a combined image. Copy it into your document and send it to the T.A.