Evolutionary Robotics Workshop – CS765 S1 2017

Motivation

From neural networks, to philosophy of mind, from numerical integration methods to genetic algorithms, Evolutionary Robotics involves a wide variety of skill sets, each of which can be challenging to learn. Given the limited amount of time we have available, I have put together a (more or less) complete evolutionary robotics project, with the hope that it will allow you to "get your hands dirty" and learn from experience while investigating and experimenting with evolutionary robotics. Like the *Robot Psychoanalysis* Workshop, this learning exercise is new and experimental. Your feedback (and your understanding) will be most appreciated. Parts of the project may be too easy or too hard, and I will take this into account when marking.

Once you get the software running, below is a set of questions that are intended to guide your investigation into the software. It is my hope that you will not just stick to this guide, but that you will also follow your nose a bit, and take looks at parts of the code that you find interesting. As before with the *Robot Psychoanalysis Workshop*, it is my hope that you don't worry so much about getting a good mark, and that you instead focus on exploring and enjoy the learning process.

I expect that in addition to the time spent in class today, you will require approximately 1-2 hours to respond to all of the questions below.

Instructions

1. **Download, install and run "Processing" (http://processing.org/).** This is a Java based programming environment that is designed to facilitate visualization and interaction.

2. **Download EvoRob from my canvas**. This is a program that uses artificial evolution to design controllers for simulated lightweight two-wheeled robots (comparable to Khepera robots http://en.wikipedia.org/wiki/Khepera_mobile_robot). You should be able to run this program by running processing, File → Open **ERWS/ERWS.pde**, and then clicking on the play button.

   There are three modes that this program can be in.

   - pressing "e" enters **evolve-mode**.
   - pressing "b" enters **demonstrate-best** mode, showing the best individual found (so far) by the evolutionary process.
   - pressing "p" enters **demonstrate-population** mode, showing the behaviour of the entire population all at once.

In addition:

- pressing “shift-R” **resets** the population, randomizing the genome of every individual,
- pressing the number keys 1-5 changes the frame rate from (slow to fast)
- pressing “s” when in evolve mode changes the lower-right plot to show the value of the sensors (top) and batteries (bottom)
- pressing “q” **quits**.

3. **For now, press “e” to put it into evolve mode**, and let it stay in that mode in the background while you read the following description of the the software.

Description

This software is using an evolutionary algorithm to train simple two-wheeled robots to seek out two different types of resource: food (green) and water (blue), while avoiding traps (red). The agents (a.k.a. animats) and the challenge are directly inspired by the paper by Anil Seth that we already read for this class:

Seth, Anil K. (1998) *"Evolving action selection and selective attention without actions, attention, or selection."* From Animals to Animats – Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior. Vol. 5.

To review what we already know from Seth’s article:

Each agent has 6 sensors: 3 on each side, corresponding to a food-detector, a water-detector, and a trap-detector.

Each agent has two batteries, one that is replenished when the agent encounters food, and the other when it encounters water. The agent “dies” when it encounters traps or when either of its batteries goes empty.

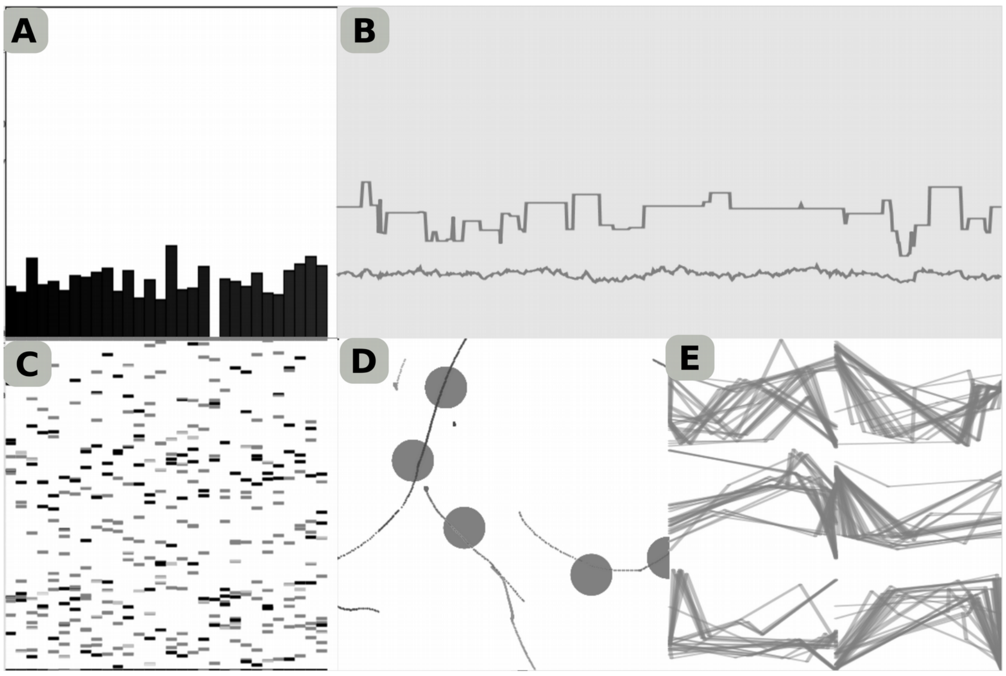
Each agent has a set of sensorimotor “links” or “activation functions.” Each link takes one of the sensory inputs and maps that to a motor output (another scalar value). The form of this map is defined by piecemeal-linear functions where the parameters are determined by artificial evolution, just as in Seth’s paper. For each motor, the sum of all incoming links determines the motor activity.

There is one link for each sensor-motor combination (*6 sensors X 2 motors = 12 links)* but we assume that the robots links have bilateral symmetry (allowing us to only have to evolve 6 links).

Finally, I should point out that the agents in this simulation are a bit simpler that in the original paper in that their activation functions are not influenced by their battery levels.

Questions

1. Press ‘e’ to go into **evolve-**mode, and without looking at the code, try to figure out what each plot/visualzation is. Annotate the figure below with brief descriptions of what you’ve figured out, labelling each area of the visualization (A-E). Don’t worry if you don’t get them all at this stage, some of the exercises below may help you figure them out later. One thing that might help you do this is to consider the time-scale of the plots (how quickly are things changing, relative to other things).



2. Open up **EvolveMode.pde** and find where the evolutionary algorithm is written. Describe how this code works in broad strokes. Ignore the clearly-marked drawing or data-tracking code. The genetic algorithm is different than the pseudo-code I went over in class – how so?

3. How is fitness calculated, and what are its approximate minimum and maximum values? (You could insert some println() statements into the appropriate place in the code to try to figure this out.) Why is the minimum fitness never as low as 0?

4. Why does the peak fitness of the entire population sometimes go down?

5. What does the parameter **N_TRIALS** affect (at the top of the file **EvolveMode.pde**)? Change its value and restart the program. How has this influenced the population’s fitness dynamics? What are advantages and disadvantages increasing this parameter’s value?

6. In this code, the fitness is the average of all of the trials. An alternative approach would be to take the product of the fitness from each trial. What would this encourage / why might one do this? To answer this question, think about the range of possible fitness values, and think about some hypothetical example trial scores that might be encountered with **N_TRIALS** set to 5.

7. Restart the program and watch the the LOWER LEFT and LOWER RIGHT plots, what happens over the first 20 generations? This dynamic is called “population convergence”. What is population convergence and why is it happening?
8. Find the parameters **MU_1** and **MU_2** in the mutate() method inside the **Brains.pde** file.
    1. What do each of the these parameters control?
    2. What happens when you set both to 0?
    3. Why might you want both types of mutation (i.e. why might you want both of these values set to non-zero values)?
    4. What would be a disadvantage of having either of these values very high (e.g. 0.5)?
9. The fitness of a population will generally increase either in a smooth gradual way, or in big leaps. Why might fitness jump up suddenly?
10. What does the parameter N_POINTS (in **Brains.pde**) determine? What would a potential advantage be of increasing the number of intermediate points be? What might a disadvantage be?
11. Calculate the total number of genes, and make a table breaking them down (10 genes are used to do X, 5 genes are used to do Y). How big is the space being searched by the GA (i.e. how many dimensions)? What are the implications of having lots of genes?
12. As mentioned above, the controller for the agents in this simulation are more simple that in the original paper, in that their activation functions (referred to as “links” in the original paper) are not influenced by battery level. How does this limit the possible behaviour of the evolved robots?
13. We also have imposed bilateral symmetry on the controller. Can you think of any ways that this constrains the controller, making finding a high quality solution harder?
14. Returning to the topic of “convergence.” (question 7), how would you expect (I) mutation rate, and (ii) population size to influence the rate at which a population converges?
15. EXTRA CREDIT: Identify some portion of the code that interests you.
    1. Briefly describe how it works (2-3 sentences).
    2. Perform a basic investigation, where you modify that bit of code (e.g. vary a parameter).
    3. Explain what you thought / hoped might happen.
    4. Explain what did happen (if anything noticeable did occur).