Theory & Practice of AI: Affective Robotics Practical 3
Affective decision making: Motivated architectures

Lola Cañamero and Matthew Lewis
L.Canamero@herts.ac.uk / M.Lewis4@herts.ac.uk

In these exercises, we will put into practice some of the concepts described in lecture 3(b) "Affective Decision Making in Robots". In particular, we will construct a motivated decision-making architecture, and consider how different ways of computing the robot's motivation can lead to different robot behaviours.

We will first see how we can turn a reactive decision-making robot into an active decision-making robot. The Braitenberg Vehicles seen previously are reactive decision-making robots: they simply react to their sensor inputs. We will now convert these into an active decision-making robot that makes decisions based on their internal motivations (see Affective Robotics lecture 3 slides, pages 38–46, for background on motivated systems and examples of how motivations can be calculated).

Exercise 1

For this exercise, you will nee to use the Playground environment named **resources.playground** (along with the **resources.png** image file) on StudyNet.

Modify the code for Vehicle 3a from Practical Sheet 2 (either from Exercise 1 or Exercise 2) as follows:

1. Add two internal **essential variables** for "energy" and the "hydration" of the robot with some initial value (say 500 each). These variables should behave as follows: when the energy/hydration is zero, the robot is dead, and the value of the variables will slowly decrease (decay) over time, meaning that the robot needs to replenish both energy and hydration variables to stay alive. Add a subroutine to decrease the variables periodically, and to call a "dead" subroutine if either of the variables fall to 0 or below. Call this "decay" code periodically, either from an already existing event handler, or by adding a new timer to call the subroutine. The "dead" subroutine, should stop the decay of the variables and should also stop robot from moving.
   Check: the variables should decay leading to the robot's death.
2. Add a way for the robot to replenish its energy. Use the central front proximity sensor (prox.horizontal[2]) to detect objects and add a subroutine that gradually increases the energy (up to some maximum, say 1000) if an object is detected close to this sensor (a value of prox.horizontal[2] greater than 500). As with the energy decay above, this code should be called periodically.
3. Now add way for the robot to gradually rehydrate by moving over blue patches on the ground. This should use the ground sensors (prox.ground.delta[0/1]) to detect the reflected IR. A value of less than 500 can be interpreted as a blue area.
4. Add code to calculate the *error* (sometimes called the deficit) in the energy and the hydration, that is the difference between the ideal value (sometimes called the setpoint) and the current value (see slide 45). If you used a maximum value of 1000 above, then use that value as the ideal value/setpoint. The error in this case should always be positive.
5. Add motivation variables: motivation to hydrate and motivation to recharge (replenish energy). Initially, these motivations will be calculated as being equal to the energy error/deficit. We will try different values for the motivation below.
6. Instead of calling the Braitenberg code to approach objects every decision-making step (e.g. every call of the prox event handler), call instead a decision-making subroutine. This subroutine will use a winner-take-all method, comparing the values of the competing motivations in the robot and calling a relevant behaviour depending on which motivation wins. In the case of the energy "motivation to recharge", the behaviour should be the Braitenberg 3a behaviour (to approach objects and stay near them).

7. If the winning motivation is the motivation to hydrate, then the robot should behave as follows. If it is over a blue patch, then it should stop in place (allowing it to rehydrate). If it is not over a blue patch then it should "wander" (move around the environment avoiding obstacles).

Now run your code and observe how the robot behaves. Watch how the values of the essential variables change. How long does the robot spend recharging/rehydrating when it finds a resource? Is it spending long enough? Will the robot survive for a long time, or will it die after a few minutes?

As described above, the initial implementation of the motivations calculated their value using just the current error value. This uses only internal signals. Now consider some different ways of calculating the motivation (Affective Robotics lecture 3 slides, page 39). Try to modify your current calculation of the motivation (step 5 above) to some of the alternatives and note what happens:

- External signals only: calculate the motivations based only on perception. Make them equal to the size of the external perception that corresponds to the resource (objects for energy; blue patches for hydration).
- Behaviour that is currently being executed. Add a variable to keep track of which behaviour (seek to hydrate or seek to recharge was executed last). If the last behaviour was to seek to hydrate, add a constant to the motivation to hydrate; if the last behaviour was to seek to recharge, add a constant to the motivation to recharge. What effect does increasing the constant that is added have?
- Combine internal and external signals. You can try adding the external signal to the internal error. You should also try the equation

  $$\text{motivation} = \text{deficit} + (\text{deficit} \times \text{stimulus})$$

  (See Affective Robotics lecture 3 slides, pages 44 & 52).

  You will need to be careful that your stimulus is not too large, or it will overwhelm the motivation. If required you can divide it by some factor.

Exercise 2

Adapt Exercise 1 above to create your own two-resource problem. You will need to define your own essential variables, resources, motivations and behaviours. Regarding behaviours, you need to write appropriate consummatory (e.g. recharge, drink) and appetitive behaviours (e.g. "find charger", "find water"), instead of than using Braitenberg vehicles.

Hints:

1. For this exercise you will probably need to adapt/design an environment. The .playground file is an XML file that you can create/edit in a text editor. You can look at the XML source of the example.playground and resources.playground files provided above and in the first practical for guidance and ideas. A typical environment might look like those used for the two-resource problem presented in lectures (Lecture 3 slides, page 48).
2. If the robot cannot locate a resource, it may use a "wander" behaviour to move around the environment. This will need to include a system for avoiding the arena walls (either a separate behaviour, or it can be built-in to the wander behaviour). You will want this behaviour to explore the entire arena and not just remain in the area around the walls.

Explore the different ways of calculating motivation used in Exercise 1 above.