Practical assignment of Intelligent Robotics II

Cognitive architectures in autonomous robotics. Deliberative systems

The practical task of the subject consists in the *implementation in Python of a cognitive architecture with a deliberative decision system*. This architecture must allow a robot to act in an environment and fulfill its objectives independently of its initial position or orientation and independently of the change of the positions of any object.

The goal of the project is for the robot to be able to **discover its own goals in the environment** and, once found, to be able to learn how to reach them in a consistent manner. To make this possible, the operation of the architecture's deliberative system will be as shown in Fig. 1.

This deliberative system should generate a series of candidate actions that, after being evaluated in a world model, will yield a series of candidate states (predicted states resulting from the execution of the actions). These predicted states must be evaluated, at each instant of time, by a utility model (based on intrinsic or extrinsic motivations). In this way, the robot must choose the action that leads to the best future state (highest utility), with the ultimate objective of guiding the robot to the achievement of its goal.

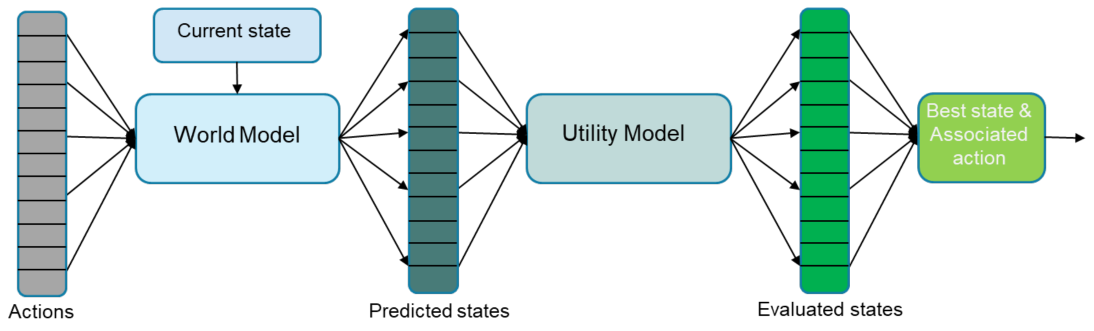

Fig. 1. Example of deliberative decision system

SIMULATOR AND ROBOT:

The practical exercise will be carried out using the simulation version of the Robobo robot:

https://theroboboproject.com

The Robobo Sim simulator will be used for testing and algorithm tuning:

https://github.com/mintforpeople/robobo-programming/wiki/Unity

You must program it using the robobopy library:

https://github.com/mintforpeople/robobo-programming/wiki/python-doc

In order to carry out the task, each group must choose a world from those included in the simulator in which there are 3 cylinders (red, green and blue), as shown in Fig. 2. This red cylinder will be the goal that the robot must **discover and learn how to reach**.

For the proper operation of the architecture, **it will also be necessary to define some environment rules that condition the performance of the robot**, so that it can avoid colliding with the walls or the rest of the objects (if any) of the scenario.

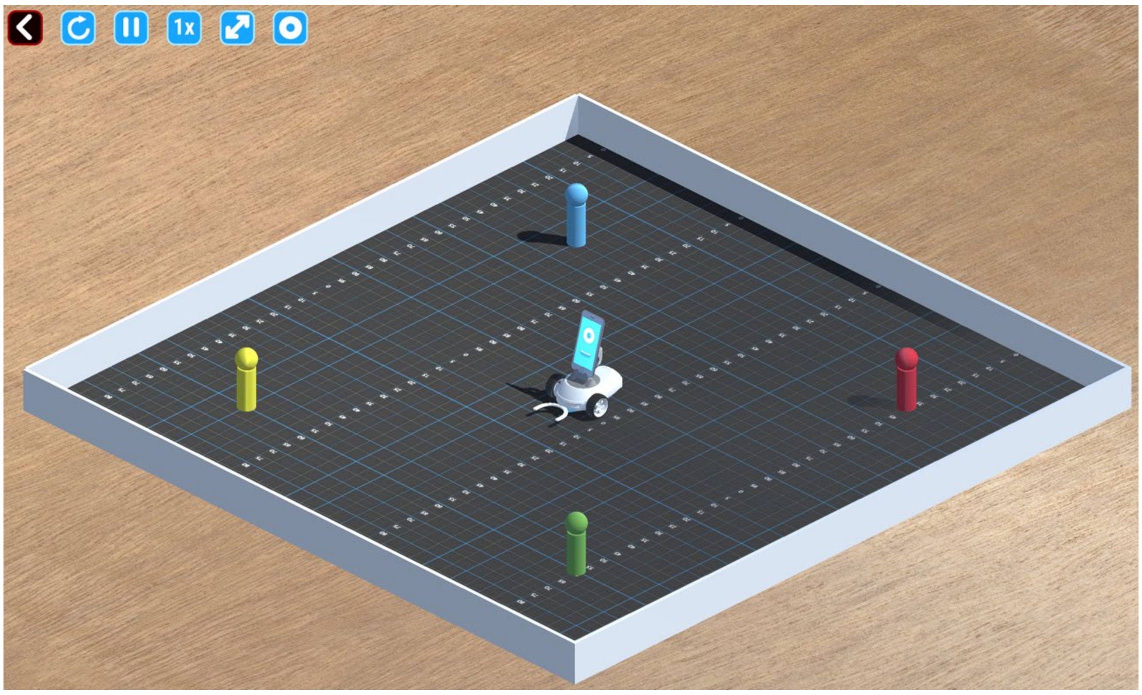

Fig. 2. RoboboSim in the world "Four cylinders"

To control the simulation, it is necessary to use also the ***robobosim*** library.

https://github.com/mintforpeople/robobo-programming/wiki/robobosimpy

With this library, we can control the position of the objects, measure distances, etc.

Sensors:

The robot's perceptions (and the input of the utility models) at each instant of time will be:

P(t) = (red, green, blue)

How to compute these perceptions depends on the complexity/realism level of your solution. For instance:

- *Simple case: red* refers to the distance from the red cylinder to the robot, *green* to the distance from the green cylinder to the robot, and *blue* to the distance from the blue cylinder to the robot. These distances can be obtained from the robobosim library, which provides the coordinates of the various objects.

- *Complex case:* obtaining distance and angle with respect to each color from the camera (x, y, size)

Actions:

The actions A(t) control the movement of the Robobo, which will move freely around the scenario. Again, the action space could be simple or complex:

- *Simple case:* the robot will move at constant speed and the actions will change its orientation in the range of angles between -90º and 90º (negative actions will move to the left and positive actions will move to the right). To simplify the problem, this range of actions will be discretized into 5 possible values: -0º, -45º, 0º, 45º and 90º.
- *Complex case:* the robot can move freely in a continuous range using the motor commands of the library (left and right power of each wheel)

World model implementation details:

The world model should be learned using the following procedure:

- Random actions are excuted by the robot in a discretized way
- P(t) are captured and stored in a memory
- A regression model is created (**one model vs many models**)
    - Linear/Non-linear regression
    - Other machine learning model (ANN)
- The model is evaluated in new random actions

If you are no table to learn the model, you can program it by hand (lower grade)

Utility model implementation details:

The choice of the utility model to be used (based on intrinsic or extrinsic motivations) will depend on the following:

- In the initial stages of learning, when the robot has not yet found its goal, a model based on **intrinsic motivation** will be used. This will guide the robot's behavior towards the exploration of the environment, i.e., towards the discovery of unvisited perceptual states. This behavior will allow the robot to discover its goal and generate training data to learn the extrinsic utility model.
- Once the goal has been discovered and the extrinsic utility model has been learned, the latter can be used to evaluate the candidate states. This model will lead the robot towards its goal and thus represents the **extrinsic motivation**.

Intrinsic part (cognitive)

This model will be implemented manually, i.e., it is not necessary to learn it. The model will be based on *Novelty*, i.e., how novel the future state is with respect to the already known states.

The objective of this intrinsic part is to explore the environment looking for potential goals and to generate traces to achieve those goals. This information will then be used to produce an extrinsic utility model associated with the goal.

The robot's actions (the candidate states that are prospectively generated by them) are evaluated using *Novelty*. Formally, the novelty of the *k*-th candidate state *Sc,k* is

$$Nov_k = \frac{1}{M}\sum_{i=1}^{M} dist\left(S_{c,k} - S_i\right)^n \ con\ k = 1\ to\ N$$

where *n* is a coefficient that regulates the balance between the relevance of distant and near states, S_i is the *i*-th state in the traces memory and *N* is the number of candidate states considered.

- **Extrinsic part (operational)**

One way to represent extrinsic utility models is by means of artificial neural networks (ANNs). These networks can be learned (online) from the traces experienced by the robot when it reaches its goal.

Thus, we can use the information contained in these traces (visited states) to reach the goal as an initial training set to train an ANN that serves as a utility model to reach that goal.

Fig. 3 represents this process in a simple setup. Obviously, a single trace will provide a poor model. Therefore, the robot will need to reach the goal from different areas of the state space to produce more traces for more complete training. This process of acquiring traces must be done with some method of exploration. Once a sufficiently large set of traces with well-evaluated states has been collected, the training of the utility model should be optimal.

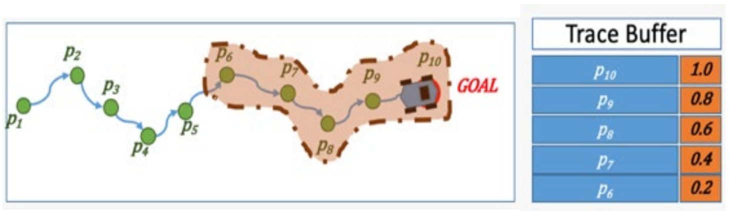

Fig. 3. Schematic representation of the learning process of the utility model. The points in the state space (perceptions) are denoted by p_i. The assigned utility goes from 1 at the point where the goal is and decreases to 0 along the trace followed.

EVALUATION CRITERIA:

- Problem definition and setup (up to 3,5 points):
    - Coding of actions
    - Perception
    - Handling the experiment using the RoboboSim library

    - Implementation of obstacle avoidance behavior
- Implementation of deliberative system based on intrinsic motivations (up to 3,5 points):
    - Implementation of world model
    - Implementation algorithm based on Novelty
- Implementation of deliberative system based on extrinsic motivations (up to 3 points):
    - Utility model learning from robot
    - Utility model use

The performance of the system will be evaluated, as well as the originality and complexity of the proposal and the solution.

It will be a positive value if the world model is also learned.

DEADLINE: The deadline for the submission of the practical work will be **Friday 30th of May, at 10:30**. It will be necessary to send by mail to the professors of the subject (martin.naya@udc.es, richard@udc.es):

1. A document containing at least:
    a. Introduction to deliberative systems in cognitive architectures.
    b. Brief description of the problem to be solved and the proposed solution.
    c. Existing problems or failures and a possible solution to them.
    d. Conclusions and bibliography.

   The maximum length will be 5 pages (excluding cover page).
2. The source code of the programming done.
3. A commented video showing the operation of the system in the simulator (maximum 5 minutes).