Project report
Academic year 2023/2024

Intelligent robotics II

Cognitive architecture for an autonomous robot in Robobo Sim

Ettore Caputo
Stefano Iannicelli

October 10, 2024

Contents

Introduction

This document is a report about a project of a Cognitive Architecture proposed for the Intelligent robotics II course. Inside this document, we'll discuss about our project decisions, we'll illustrate our solution and how it works. The entire work is based on *Python*, the *Robobo Simulator* and its correlated python libraries. The goal of the project is for a robot to be able to reach its goals in an hypothetical environment, in our case the simulator, basing its behavior on a Cognitive Architecture.

1 Description of the problem

The problem consists in designing a cognitive architecture that allows a robot to be autonomous and follow its goals. To achieve this we follow the deliberative decision system architecture showed in Figure 1. Thus, we've implemented the *World Model* and the *Utility Model*.

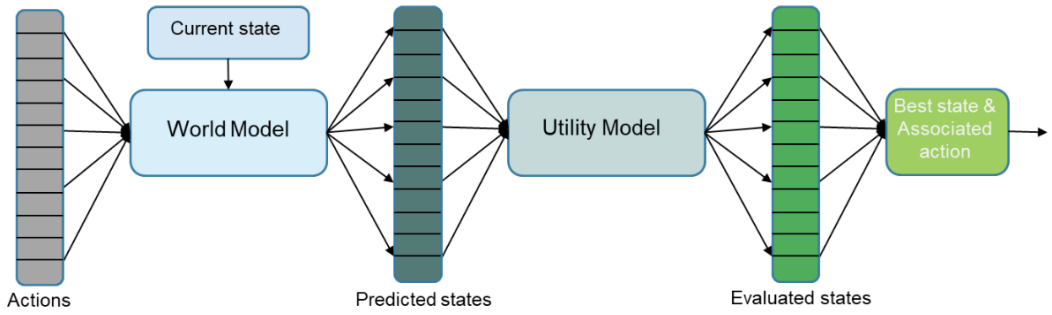

Figure 1: Example of deliberative decision system

2 Perform action

The first thing that we did was to create some primitive for perform the robot action, all the rotation of the robot were made with the gyroscope present on the robot. This sensor return an angle between -180, 180 so for use that we implement the following method that allow the robot rotation from every angle:

Case $action \in [Left30, Left15]$:

$$sign = -1$$
$$range_1 = [\alpha - 180, 180]$$
$$range_2 = [-180, \alpha - 180]$$

Case $action \in [Right15, Right30]$:

$$sign = 1$$
$$range_1 = [-180, 180 - \alpha]$$
$$range_2 = [180 - \alpha, 180]$$

$$\theta_{target} = \theta + sign * \alpha \tag{1}$$
$$\theta_{target} = (-sign * 180) + sign * (sign * \theta - (180 - \alpha)) \tag{2}$$

When $\theta \in range_1$ the equation (1) is used, otherwise when $\theta \in range_2$ the equation (2) is used. These equations are used to compute the angle θ_{target} the robot must reach.

3 Proposed solution

To resolve the problem defined before we present our architecture, a first macro representation is showed in Figure 2. Starting from the left you can see the input given to the World Model. This input is the current state, it is composed by two principal part:

Sensing information. This part represent the information from the robot's sensors, its position and its orientation. We can represent the *sensing information* with the following tuple: $\langle x, y, theta, ir, red\_pos, red\_size \rangle$.

One-hot Action. This part is a one-hot representation of the possible actions that the robot can apply in the environment. In detail the possible robot actions are five, $[Left30, Left15, Forward, Right15, Right30]$.

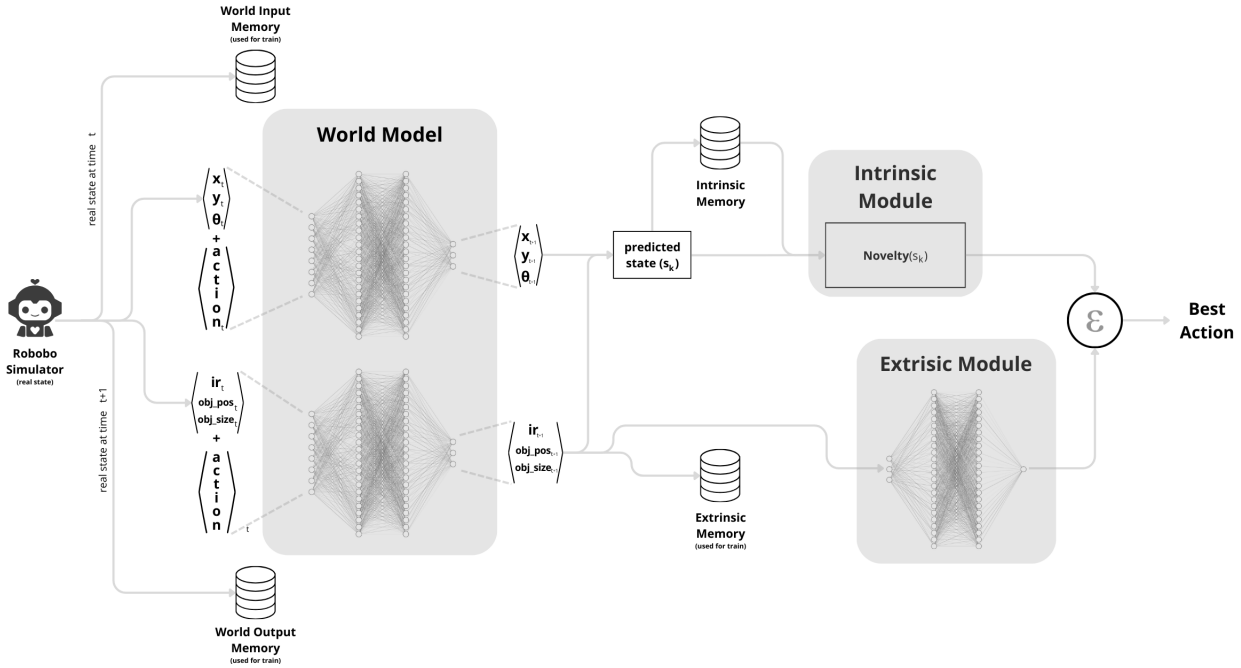

Figure 2: This figure show the architecture proposed. In this case the World Model is composed of Neural Networks. In the simple version of our propose is implemented with mathematical operations.

3.1 World Model

The World Model represent the abstraction of the physical behaviors in the Robobo environment. We've implemented the World Model in two different versions. The first one is based on mathematical operations, then to provide the predicted states from the sensing in input we use some simple algebra operations. The second one use a Neural Network, in this case the NN have to learn to predict well the future states. We implement the obstacle avoidance behaviour, because if the robot have an obstacle in front, the number of predicted states will be four not five, because the forward action isn't available.

3.1.1 Mathematical version

The state in input s_t to the World Model at time t, is identified by the six-tuple $\langle x, y, theta, ir, red\_pos, red\_size \rangle$. In the world model we try to predict the states at time $t+1$ after perform all the possible actions that the robot can perform at time t. The state at time $t+1$ is $\langle x_{t+1}, y_{t+1}, \theta_{t+1}, ir_{t+1} \rangle$ computed in the following way, for the forward action will be updated:

- $x_{t+1} = x_t + 60 * cos(\theta_t)$, where 60 is the movement at 10 of speed
- $y_{t+1} = y_t + 60 * sin(\theta_t)$
- $ir_{t+1} = ir_t + 50$, if $ir_t > 0$, else $ir_{t+1} = ir_t$
- $red\_size_{t+1} = red\_size_t + 50$, if $red\_size_t > 0$, else $red\_size_{t+1} = red\_size_t$

for the actions that turn the robot of an angle α will be updated:

- $\theta_{t+1} = \theta_t + \alpha$

3.1.2 Neural Network version

To generalize well the physical behaviors in the Robobo environment, we tried to use a Neural Network architecture for the World Model. As you can see in Figure 2 is composed of two NNs, one for the *positioning* of the robot and the other one for the *sensing*. The NNs are the same, they are composed of one input layer $input\,layer \in \mathbb{R}^8$, **two** $hidden\,layer \in \mathbb{R}^{64}$ and one $output\,layer \in \mathbb{R}^3$. The positioning NN has the task of, given the triple $\langle x_t, y_t, \theta_t \rangle$ in input and the one-hot encoding of the action, predict the next values state: $\langle x_{t+1}, y_{t+1}, \theta_{t+1} \rangle$. The sensing NN has the task of, given the triple $\langle ir_t, obj\_pos_t, obj\_size_t \rangle$ in input and the one-hot encoding of the action, predict the next values state: $\langle ir_{t+1}, obj\_pos_{t+1}, obj\_size_{t+1} \rangle$. We use the network 5 times for predict all the possible next state of the robot.

3.2 Utility Model

3.2.1 Intrinsic Module

The intrinsic model takes in input the predicted state computed by the world model, is implemented with the novelty function:

$$Nov_k = 1/m \sum_{i=1}^{M} dist(S_{c,k} - S_i)^n$$

where $dist(S_{c,k} - S_i)$ is the L1 norm between the predicted state k and the M older state saved in the intrinsic memory. At the end of this we have the an array of 5 element, each element correspond at the novelty value for the state k in input. The index of the array that contains the max novelty value is the action that the robot will perform.

3.2.2 Extrinsic Module

The extrinsic module, as requested in the assignment was implemented with a Neural Network, as you can see in Figure 2, is composed by one input layer $input\,layer \in \mathbb{R}^3$, **two** $hidden\,layer \in \mathbb{R}^{64}$ and one $output\,layer \in \mathbb{R}^1$. The input to the neural network is a triple $\langle ir_{t+1}, red\_pos_{t+1}, red\_size_{t+1} \rangle$, this information came directly from World Model output. The network is composed of two hidden dense layer, each of 64 units. The output layer is only one unit and return a value in the range $[0, 1]$. For this reason we used the sigmoid activation function in the last layer. We evaluate each predicted state, given from the world model, by passing it to the neural network, at the the action that we chose is the one that produce the higher value.

Loss function. The loss function for the Extrinsic Module is the *Binary Cross Entropy* loss, defined as follow:

$$L(x, y) = -w[y * log(x) + (1 - y) * log(1 - x)]$$

We decided to use this loss because our reward values given to each action are between 0 and 1, as explained below.
Given a path $p = [s_{p_0}, ..., s_{p_n}]$, where the state s_{p_n} is the nearest to the goal, we used an exponential law reward function to assign the value to each state in the path p. The states near the goal have reward close to 1, the further away the state is, the closer the value will be to 0.

$$reward(i) = e^{-i/16}$$

where $i \in [0, n]$ is the index of the state that we want to assign the reward in the path p.

4 Train

We choose the ϵ-greedy policy for train the model, at the start the action is chosen by the Novelty with probability 1, every time that the robot find the goal this probability is decreased by 0.04. Therefore, as the robot finds the goal, the probability that the actions performed will be chosen by the neural network will increase. With the mathematical world model the robot learn a good policy in about 10 minutes.

5 Existing problems and possible solution

The neural World Model doesn't seem to works well. After many trials, different configurations, some other small changes we can say that is not able to predict the next state well. We think that the problem is in the *obj_size* prediction, because the world model probably not has learned to increase the value of *obj_size* in the generated predicted state when the action is: $Forward$. For this reason at a certain point the robot starts to turn left or right when it sees the object in front of it, without ever moving forward. This because the robot is driven more from the *obj_pos* value, that produce a positive increment when the object change position on the screen.

Conclusions

In this project we implement a simple cognitive architecture, using the Robobo simulator, we create the World Model, mathematical and neural, the intrinsic motivation based on the Novelty and the extrinsic based on a neural network. After the train process we obtain good result with the mathematical World Model, infact, the robot, after an initial exploration phase, learn how to reach the goal.