Point-to-Point Navigation with Deep Reinforcement Learning

Paul-Antoine Le Tolguenec[1,✉]

[1]ENSTA Bretagne, Brest

Nowadays the problem of the autonomous vehicle is one of the most developed subjects in the world of research. Although there are different methods, methods based on deep learning are more and more used in this field. This article illustrates a solution based on Deep Reinforcement Learning.

Deep learning | Reinforcement | Control | Autonomous car

Correspondence: ***paul-antoine.le_tolguenec@ensta-bretagne.org***

Supplementary Note 1: Introduction

In recent years, mobile robotics has experienced unprecedented growth. Most of the methods concerning trajectory estimation are based on automatic laws. A very robust method is the linearizing loop control method. But in some cases this method has to be reworked, because it has to be linearised around a working point or other. Another very robust method is that of potential fields, since it also allows to avoid certain obstacles that could appear spontaneously on the way to the robot. But for this type of method it is often necessary to implement other layers that allow to find the path to reach a point in the defined space. Most of the time it is also necessary to develop a finite state machine to manage the high level of the robot. Usual methods are therefore very time-consuming to implement and can lead to mistakes. Moreover, once the control of the robot is set up, optimisation is difficult. With the progress of deep learning, a new control approach has emerged: deep reinforcement learning. The advantage of this method is that once the method is set up it can be generalised to many problems and it is only necessary to change certain parameters. Also this method is permanently optimized. In this article, I expose my work which has allowed me to train a model to make decisions to orientate a car so that it is able to go from point A to point B as quickly as possible by avoiding objects that appear as they go along. To train the model I use the A2C (Advantage Actor Critic) algorithm which uses the principle of reinforcement learning. Part II illustrates the modelling I have carried out and within which I have trained the network. Part III presents the implementation of the method (network architecture, theory ...). Finally, part IV illustrates the experimentation phase.

Supplementary Note 2: Modelisation

A. Formalisme. The objective of our work is to automate a car so that it is able to go from point A to point B while avoiding obstacles. The kinematics of the car is modelled by the following equations:

$$\begin{cases} (i) & \dot{x} = v\cos\phi \\ (ii) & \dot{x} = v\sin\phi \\ (iii) & \phi = u_{\pi_\theta} \\ (iv) & v = k \end{cases} \quad (1)$$

At first, I tried to control only the steering angle of the car. Thus u_{π_θ} represents the ϕ output taken via a policy π_θ. The robot is also equipped with three sonars, one frontal and two lateral, which allow to visualize its environment.

B. Simulation. Once the kinematics were defined, I had to simulate the robot and obtain a visual rendering. I used the Kivy library of python. Kivy is a free and open source Python framework for developing mobile apps and other multitouch application software with a natural user interface. The objective was to have a visual rendering of the robot's evolution, but also to be able to interact with the environment via a man-machine interface. Once the simulation is launched, a window opens and the robot evolves in its environment. The operator can interact with the robot's environment by creating walls that he can draw with the mouse. There is also a button to erase all the walls on the map to rebuild others and see if the robot is able to generalise its behaviour in any type of environment.

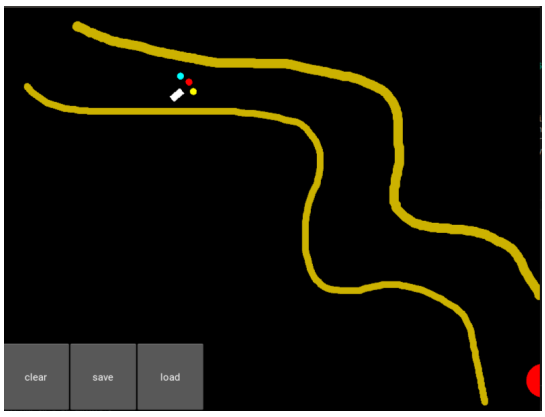

Fig. 1. Simulation of the system

As can be seen in the figure. The car is represented by a white square. The sonars of the car are represented by the three circles at the front of the car.
The walls are represented by the yellow lines. And the objective of the robot is represented by the red dot. At first, the robot can cross walls but when it does so its speed decreases

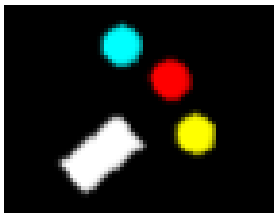

Fig. 2. Robot sonars

and it gets a negative reward because it has crossed the wall and because it will arrive less quickly at its target.

Supplementary Note 3: Proposed method

The objective of our work is to train an agent to be able to choose the orientation angle of the car according to his perception of the environment.

A. network structure. The structure of the network is quite simple. The network takes as input: the distances output by the sonars, and the angle formed by the direction of the robot and the line passing through the centre of the robot and target. To simplify the teaching process, we have discretised the robot's exit space. The robot can make three decisions: do nothing, increment its angle by +20 degrees or -20 degrees. Of course, making the exit area discreet means poorer results in the long term. But it allows for a quicker convergence towards a more stable policy.

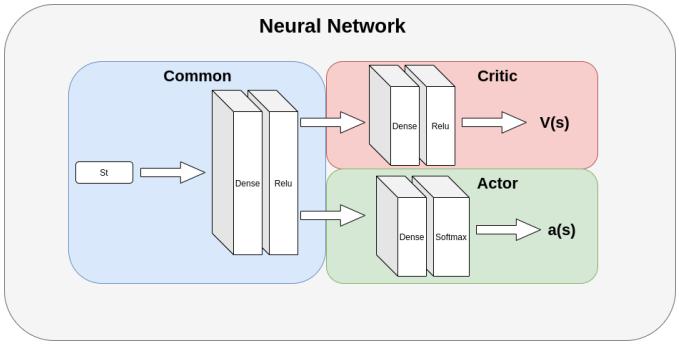

Fig. 3. Neural Network Structure

As you can see the network is divided into two parts:

- Critic

  The Critic part is used to determine the value of the state in which the agent is in. This part corresponds to the value-based part of the learning. It allows to stabilize the learning and to converge towards a global maximum of the policy.

- Actor

  The actor part corresponds to the part of the agent that takes action. It is the policy-based part of learning. The actor part pulls out a probability distribution $\pi_\theta(s)$ which makes it possible to determine what probability is for the agent to take this or that action given the state.

The purpose of learning is to change the probability distribution in the direction that gives the maximum reward. The value part allows to compare the quality of an action carried out in relation to the value of a state already estimated to know if this action is more optimised than what has been done before.

B. Advantage Actor Critic algorithm. To update the network parameters we used reinforcement learning. We used one of the most powerful reinforcement learning algorithms before the A3C. A2C is an algorithm halfway between value based learning and policy based learning.

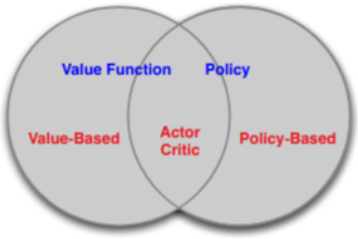

Fig. 4. Actor Critic

The Advantage Actor Critic algorithm takes the following form :

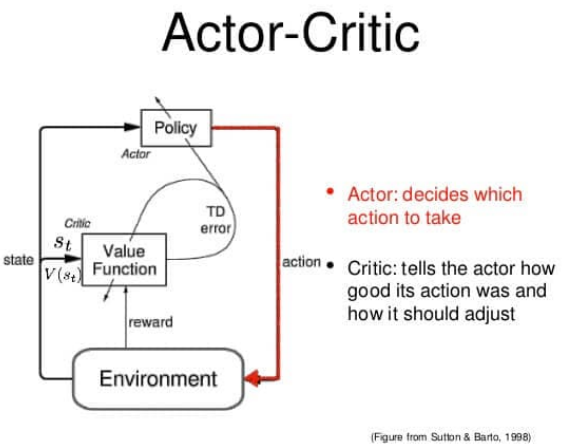

Fig. 5. Actor Critic

As can be seen in figure 4, the principle of the critical actor is to take an action thanks to the actor and to criticise the action achieved thanks to the critic. But to criticise this action you need something to compare this action with the critic. This is the Advantage part. There are several ways of expressing Advantage. But the most common method is to take as the advantage : $Rt - V_{\pi_\theta(s,a)}$
with $Rt = \gamma^N V_{end} + \sum_{k=0}^{N-1} \gamma^k (r_{t+k})$.
As can be seen in figure 4, the advantage part of the algorithm can also be called the TD error.
In the literature one initially finds $Rt = \gamma V_{t+1} + r_t$ but recent work on replay experience has shown that taking into account more rewards makes it easier to converge towards a global minimum of the cost function estimating the MDP (Markovian decision process). On the other hand, using this

Algorithm 1 N-step Advantage Actor-Critic

procedure N-STEP ADVANTAGE ACTOR-CRITIC
  Start with policy model π_θ *and value model* V_ω
  repeat:
    Generate an episode $S_0, A_0, r_0, \ldots, S_{T-1}, A_{T-1}, r_{T-1}$
    for t *from* $T-1$ *to* 0:
      $V_{end} = 0$ if $(t+N \geq T)$ *else* $V_\omega(s_{t+N})$
      $R_t = \gamma^N V_{end} + \sum_{k=0}^{N-1} \gamma^k \, (r_{t+k}$ *if* $(t+k<T)$ *else* $0)$
    $L(\theta) = \frac{1}{T} \sum_{i=0}^{T-1} (R_t - V_\omega(S_t)) \log \pi_\theta(A_t|S_t)$
    $L(\omega) = \frac{1}{T} \sum_{i=0}^{T-1} (R_t - V_\omega(S_t))^2$
    Optimize π_θ *using* $\nabla L(\theta)$
    Optimize V_ω *using* $\nabla L(\omega)$
end procedure

method implies a certain volatility of the estimator since a stock affects the value of a state further in time, and it often takes longer to converge. It is therefore a question of finding the best hyper-parameters, which are often found empirically.

C. Reward process. In reinforcement learning methods the reward process is very important as it allows the robot to perceive its environment. More importantly, it corresponds to the MRP (Markovian reward process) that we try to estimate thanks to the neural network (the critical part). So if the reward process does not correspond exactly to the environment, the agent will not correctly estimate the behaviour it has to adopt.

Supplementary Note 4: Experimental discussion and results

Considering that the rope remains continuously under tension while the robot is moving, then the evolution of the angle x_4 of the sled relative to the towing vehicle is described by the *Differential Equation* **??**.
We are able to find the trajectory of the sled by computing the interval containing the angle x_4, considering that initially x_4 belongs to $[-\pi/2; \pi/2]$. Then by applying the *Differential Equation* **??** on this interval, knowing the control vector u, we end up obtaining a fine interval framing the real angle x_4, independently of the initial angle as we can see on the FIGURE **??**.