Reinforcement Learning using Stage ROS

Gabriel Paludo Licks
Graduate Program in Computer Science - School of Technology
Pontifical Catholic University of Rio Grande do Sul - PUCRS
Porto Alegre, Brazil
gabriel.paludo@acad.pucrs.br

Abstract

Reinforcement learning is an approach to learn agent policies in stochastic environments where the goal is to control the system in a way that the expected accumulated reward is maximized over time. In the context of robotics, reinforcement learning offers a framework for the design of behaviors in an environment where this machine learning techniques can be applied. Though approaches to reinforcement learning using ROS robots exist, most of them require heavy processing power. In this paper, we report the development of a reinforcement learning approach to train ROS robots using Stage simulation. We have implemented the Q-Learning algorithm for training the robot and instantiated the training to two environments where the robot has a task to accomplish, one with a free path and another with an obstacle to avoid. Results show that the free path environment easily converges to an optimal policy, whether the path with obstacle environment converges yet to a sub-optimal policy.

Introduction

Reinforcement learning is an approach to learn agent policies in stochastic environments modelled as Markov Decision Processes (Bellman 1978). An environment is framed as a set of states and a stochastic transition system whose transitions the agent influences by choosing from a set of actions. The goal is to control the system in a way that the expected accumulated reward is maximized over time (Sutton and Barto 2018a). This imitates the trial-and-error method used by humans to learn, which consists of taking actions and receiving positive or negative feedback.

In the context of robotics, reinforcement learning offers a framework for the design of behaviors (Kober, Bagnell, and Peters 2013). The challenge is to build a simple environment where this machine learning techniques can be validated. Packages such as the OpenAI_ROS[1] provide a structure to make it easy to train ROS robots using Gazebo simulation, however training is mostly not a feasible task due to physics simulation complexity resulting in a time consuming training and a large state space that has not been discretized.

[1]

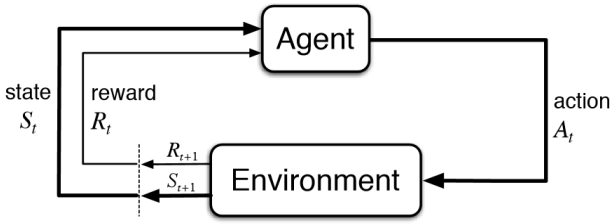

Figure 1: Agent-environment interaction in a MDP

In this paper, we report the development of a reinforcement learning approach to train ROS robots using Stage simulation, which is a lighter simulation in terms of processing for multiple trials during training. We have instantiated the training algorithm to two environments where the robot has a task to cross a corridor from one side to another, one of them having an obstacle to be avoided. Results show that the corridor environment easily converges to an optimal policy, whether the corridor with obstacle environment converges yet to a sub-optimal policy.

Approach

This work relies on an implementation of a reinforcement learning agent, which will act over an environment to explore it and find policies for transitioning in an environment. Most reinforcement learning problems can be formulated as a Markov Decision Processes (MDPs). Figure 1, from Sutton and Barto, illustrates the agent-environment interaction in a MDP. Thus, there are a few aspects that need to be defined: the environment in which the agent will be acting in; how the state of this environment will be represented; which actions our agent will be able to execute; and a reward measure as a feedback from the environment to the agent.

In our case, the environment is the map in which the robot agent will interact, which is instantiated using the Stage simulator. Our environment state will be defined by the current coordinate position where the robot is standing in the map. The available actions in this environment correspond to the possible movements the robot can perform, which is to go FORWARD, LEFT or RIGHT in order to cross the map to reach the destination point. Finally, one of the tasks of RL is to use

observed rewards to learn an optimal (or nearly optimal) policy for the environment (Russell and Norvig 2016), therefore our agent needs a reward function.

We model our reward function based on the information we get from the environment, which in this case is the robot position and the distance from the robot to the obstacles (walls). We assign a higher reward for FORWARD actions and a slightly smaller value for LEFT and RIGHT. Then, we calculate an euclidean distance from the current robot position to the goal and increse this reward as the robot gets closer to the destination. Another factor taken into consideration is how close the robot is to a wall or obstacle, since we want to avoid it. Thus, we punish the reward value as the robot move closer to walls by the information received from the laser sensor.

Training algorithm

The algorithm used for training the robots is the Q-learning, which is a popular reinforcement learning algorithm. This method learns the values of stateaction pairs, denoted by $Q(s, a)$, which represents the value of action a in state s (Sutton and Barto 2018b). The basic idea of the algorithm is to use the update function of Equation 1 to incrementally estimate values of $Q(s, a)$ based on reward signals from each action taken. We store these action-values learned for each state-action pair, and a good policy can be achieved if each state-action pair is visited as many times as possible (Watkins and Dayan 1992). The Q-Learning pseudo-code we use is shown in Algorithm 1

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a)) \quad (1)$$

The agent training is organized in episodes, each of which is composed of a number of steps in the environment. The number of episodes and steps in an episode can be defined through a parameters file. Each step consists of the agent choosing an action, executing it in the environment and letting the environment return the reward for then learning its state-action pair value (using the equation we specified above). These steps are repeated until the robot (agent) reaches a terminal state (goal position or an invalid position). If a terminal state is reached, the environment is reset and the robot starts a new episode from the initial position state.

Environments and robot trained

The robot we used for training is the Turtlebot 2, a classic in ROS robot platforms, which is a two-wheeled differential drive robot. We have then modelled two environments for training, which are in the shape of a corridor. The first, named "Corridor" environment, is a free path where the robot has to learn how to cross from one side to another without crashing into the walls. The second, named "Corridor with Obstacle" environment, is the same shape as the first however containing a pointing wall in the middle of the corridor, which the robot has to avoid in order to cross to the other side and reach destination point. We show the first "Corridor"environment in Figure 3 and the second "Corridor with Obstacle" environment in Figure 4.

Algorithm 1: Q-Learning pseudo-code

Input: percept, a percept indicating the current state s' and reward signal r'
Output: An action a.
Persistent: Q, a table of action value indexed by state and action, initially zero ;
N_{sa}, a table of frequencies for state-action pairs, initially zero ;
s, the previous state, initially null;
a, the previous action, initially null ;
r, the previous reward, initially null;
if $isTerminal(s')$ **then**
  $Q[s', None] \leftarrow r'$;
end
if *s is not null* **then**
  $N[s, a] \leftarrow N[s, a] + 1$;
  $Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$;
end
$s \leftarrow s'$; $r \leftarrow r'$;
$a \leftarrow \arg\max_{a'} f(Q[s', a'], N[s', a'])$;
return a

Experiments

We trained the Turtlebot 2 robot in the environments and tasks we modelled using the Stage simulator and observed their behavior throughout the episodes. We trained the robot in the first "Corridor" environment for the course of 500 episodes, which is a sufficient number of episodes to train the robot for accomplishing such task without obstacles. The accumulated reward per episode is shown in Figure 2. Initially, the steps performed do not result in a high reward, due to a higher number of random actions taken in order to explore the environment. The rewards grow over episodes and show convergence after 300 episodes of training, resulting in an optimal policy performed afterwards.

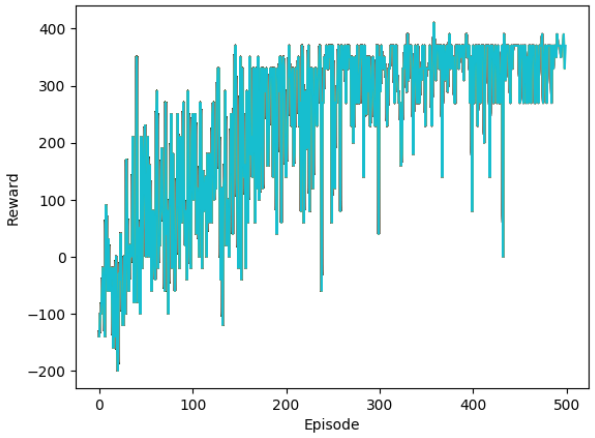

Figure 2: Training in "Corridor"

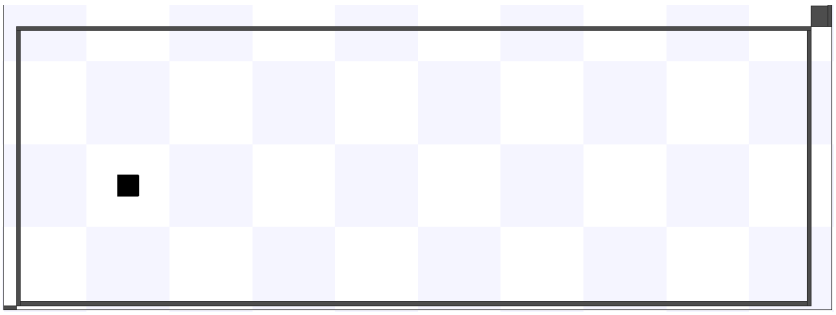

Figure 3: Corridor environment

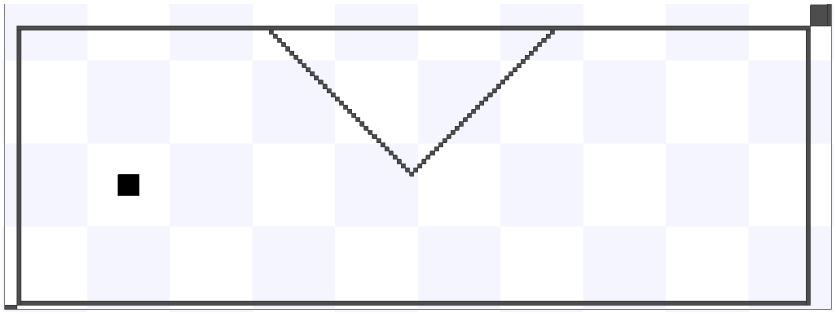

Figure 4: Corridor with Obstacle environment

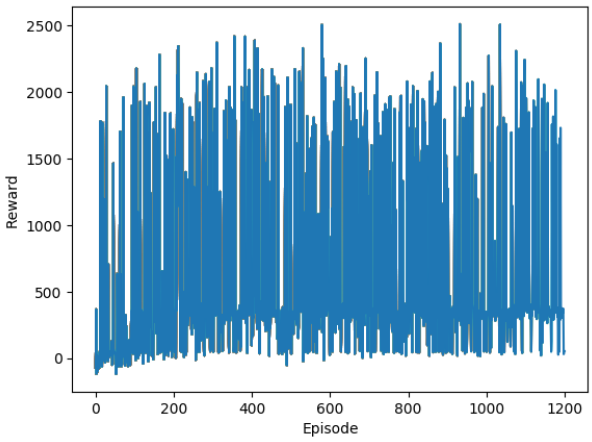

Figure 5: Training in "Corridor with Obstacle"

In the second "Corridor with Obstacle" environment, we trained the robot over the course of 1200 episodes. The accumulated reward per episode in this environment is shown in Figure 5. In this training, the robot has achieved a sub-optimal policy and performs an unstable behavior. Most of the times, the robot reaches the destination point (which is shown in rewards values above 1500). There is a learning curve in the rewards plotted, however it did not stabilize yet with the iterations trained. For a proper convergence in this environment, there might be necessary a different modelling of a reward function that pushes the agent to have a better understanding of possible obstacles.

Final considerations

We have explored the use of Stage simulator and ROS-based robots for accomplishing tasks using reinforcement learning. As a limitation we encountered, Stage simulation tends to be unstable and inaccurate if accelerated. Increasing the simulation speed too much causes the robot to crash into the walls, since there is not enough time to interpret the commands sent by the algorithm and stop the robot. We have also noticed that sensor readings are also inaccurate if simulation speed is too high, causing the algorithm to receive wrong or noisy laser values at the specific time it receives from the topic subscribed. Either way, Stage simulation is lighter and requires less processing power if compared to Gazebo simulation, which contains a higher level of complexity.

We trained the Turtlebot 2 robot in the environments we modelled. In the first "Corridor" environment, the agent shows easy convergence and learns an optimal policy. In the second "Corridor with Obstacle" environment, the task requires a more complex reward function model, yet it still achieves a sub-optimal policy reaching goal destination a considerable number of times. Nonetheless, it seems an interesting tool for arousing the use of reinforcement learning in the context of robotics in order to simplify the evaluation of different algorithms to solve tasks.

References

Bellman, R. 1978. *An introduction to artificial intelligence: Can computers think?* The University of Michigan: Boyd & Fraser Pub. Co.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11):1238–1274.

Russell, S. J., and Norvig, P. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Sutton, R. S., and Barto, A. G. 2018a. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., and Barto, A. G. 2018b. *Reinforcement learning: An introduction*. Cambridge, MA: MIT press. chapter Temporal-Difference Learning, 119–138.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.