Reinforcement Learning using OpenAI_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. The OpenAI Gym is a toolkit for training algorithms in standardized environment that lately has grown popularity. This toolkit is adapted for robotics in a ROS package named OpenAI_ROS that integrates standardized environments modelled in Gazebo for robot training using ROS. In this paper, we report the usage of this ROS package through installing and training robots in the provided environments for the predefined tasks. Throughout these experiments, we find limitations and constraints that may often cause the training process unfeasible. Though we could performed training using the resources available, the policies executed by the robots were sub-optimal due to the constraints identified.

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. The OpenAI Gym, which has recently gained popularity in the machine learning community (Brockman et al. 2016), is a toolkit that provides such environments in order to test different reinforcement learning algorithms to perform a predefined task.

Unfortunately, even if the Gym allows to train robots, it does not provide environments to train ROS based robots using Gazebo simulations. Ezquerro, Rodriguez, and Tellez have created a ROS package for this purpose, which provides common ground environments using ROS and Gazebo in order to execute reinforcement learning algorithms for robots to act in the environments. In this paper, we report experiments using this package and the limitations evidenced in such approach of using ROS and Gazebo for reinforcement learning.

The OpenAI_ROS package

The OpenAI_ROS package is a structure to make it easy to use OpenAI with ROS and Gazebo. The package provides environments for most of the common robots used in ROS, so that the user only have to focus on the AI, and not to worry on the connection to the simulated robot and environment formalization. The package is basically composed of the following main elements (Ezquerro, Rodriguez, and Tellez 2018): the training environments and the training script.

Training environments: these are the ones in charge of providing to the learning algorithm all the data needed in order to make the robot learn. They inherit from the OpenAI Gym official environment, so they are completely compatible and use the normal training procedure of the Gym. The OpenAI_ROS package already provides many environments for robots and tasks, so it is likely to use the desired ones and concentrate on the learning algorithm (Ezquerro, Rodriguez, and Tellez 2018).

There are different classes that compose the training environments:

- *Task Environment*: This is the class that allows to specify the task that the robot has to learn.
- *Robot Environment*: This is the class that specifies the robot to use on the task.
- *Gazebo Environment*: This is the class that specifies the connection to the simulator.

Training script: it defines and sets up the learning algorithm that is going to be used in order to train the robot. This is where the reinforcement learning algorithm sits. The most important aspect to understand from the training script is that it is totally independent from the environments. This means

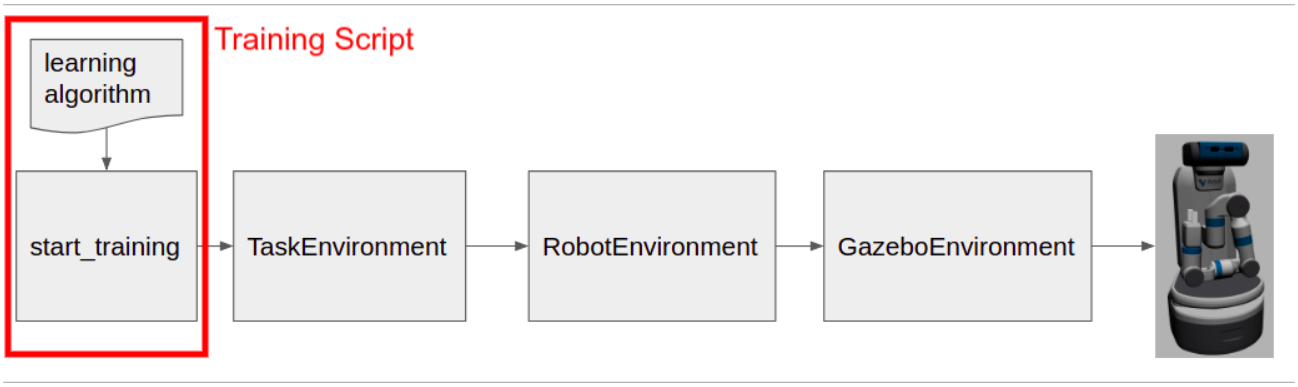

Figure 1: OpenAI_ROS package pipeline. Source: wiki.ros.org/openai_ros

that, it is possible to change the algorithm used to learn in the training script without having to worry about modifying the environments structure (Ezquerro, Rodriguez, and Tellez 2018). The overall structure of these elements composing the package and how they connect to each other is illustrated in Figure 1.

Experiments

We have installed the OpenAI_ROS package and executed training in three different environments using two different robots. We start by explaining the algorithm used for training the robots (first subsection), then we introduce the robots and the environments we used for training (second subsection) and, lastly, we discuss the results we obtained through the training process (third subsection).

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).

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

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 robots trained

The first robot we tested for training is the Turtlebot 2, a classic in ROS robot platforms. The environments provided by the OpenAI_ROS package for the Turtlebot 2 are two: a circuit maze (Figure 2); and an empty world with a wall (Figure 3). In the circuit maze environment, the task is for the robot to learn how to move around the maze and get back to the starting point. In the wall environment, the task is to cross by the side of the wall and reach a center point in the other side of the wall. In these environments, the robot locates itself through laser sensor information.

The second robot used is the WAM-V USV robot, which is a boat used in challenges such as the Maritime RobotX Challenge. The environment provided for training this robot is the sandisland world (Figure 4). In this environment, the task is to remain inside the buoys that delimit a corridor for the boat and cross from one side to another. In this environment, the boat locates itself by its current XYZ position on the environment, thus it knows when it reaches an area out of the buoys and also knows if it reaches the goal position.

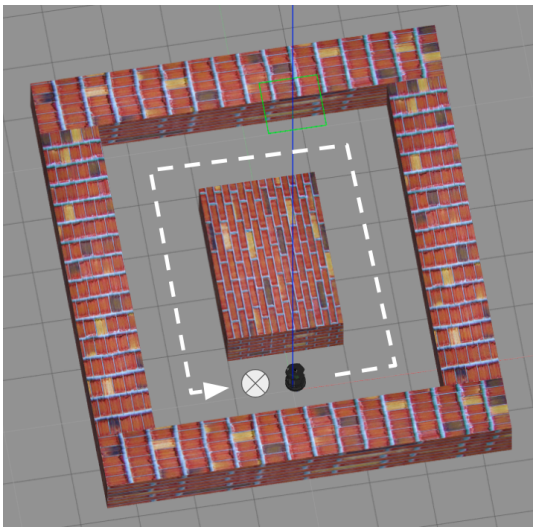

Figure 2: Turtlebot Maze Environment

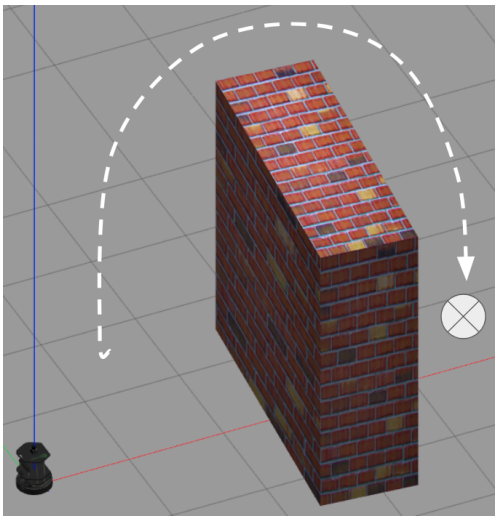

Figure 3: Turtlebot Wall Environment

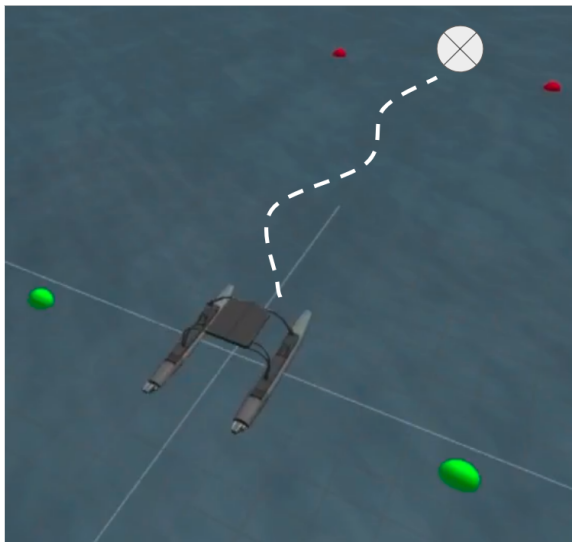

Figure 4: WAM-V Boat Environment

Results and discussion

We trained these robots in their respective available environments and observed their behavior throughout the episodes. We trained the Turtlebot in both environments provided by the package for at least 2000 episodes each environment, with no decent accumulated reward per episode. For the WAM-V boat, we started to get decent accumulated rewards per episode after training over approximately 24 hours, which resulted in 6000 episodes of training. However, these iterations do not seem to bring the algorithm close to convergence. The learning process is exploration-intensive and these environments provide a large state space, making it hard for the algorithm to explore all states and learn its state-action values. Thus, the robots have learned suboptimal policies, normally with unstable behavior.

There are two reasons that strongly relate to this result: the number of iterations and/or the environment modelling. The number of iterations is fundamental for learning optimal policies by exploring all states and actions in an environment. If an agent does not iterate enough, the policies tend to be always suboptimal due to lack of environment knowledge. In our case, it is likely that the environment modelling provided by the package results in a large state space, as every XYZ point location in the map represents a new state for the robot agent. For this reason, there might not be an estimated number of iterations needed in order to learn optimal policies and learn consistent state-action pair values.

Limitations

Throughout the installation of the OpenAI_ROS package, we encountered a number of constraints. The installation is not a clear process to perform, since the authors do not clearly specify dependencies and packages that are need for running the training and the models used in the environments. The package documentation lacks a consistent structure of steps for installing and running experiments with the package, often providing misleading information and wrong links to repositories.

Instead of installing the OpenAI_ROS package and its dependencies in the user's local machine, the authors tend to recommend their web platform called ROS Development Studio [1]. In this platform, they provide the infrastructure and a ready-to-use ROS environment with all dependencies met. However, it is only available for free at a low CPU power and for a 30 hours limited session. Thus, the use of this platform is out of the scope of this report.

The main limitation encountered during the experiments is due to the trainings duration. We found that the simulation could not be accelerated significantly for speeding up the training process. In addition, Gazebo simulation tends to be unstable and inaccurate if accelerated. When testing with a Turtlebot, increasing the real time factor of the simulation caused the robot to crash into the walls. The reason for this to happen is most likely to the lack of processing power for physics and not interpreting the laser signals to stop the robot. Note that, this package still only supports working with Gazebo.

Another reason that may cause slow convergence is the large state space along with non-deterministic transitions that are due to the simulator's physics. The state space of the environment is represented by each XYZ point in the map, which considerably raises the number of combinations for state-action pairs that have to be learned by the agent through iterations. The transition from one state to another is also non-deterministic, which means that choosing an action in a state does not guarantee arriving in the expected state, due to the physics of the environment (e.g. the wind, in the WAM-V environment).

[1] Information available at http://www.theconstructsim.com/rds-ros-development-studio/

Final considerations

We have explored using the OpenAI_ROS package in a local installation and we accomplished the integration between the package and its dependencies, being able to execute robots training in the environments provided. However, training in reinforcement learning is often a time consuming task and is directly related to the size of the state space. Though the agent reaches suboptimal policies, it is necessary for the training process to be held in a more powerful environment than a local machine in order to converge faster.

Overall, the OpenAI_ROS is yet a tentative solution for integrating OpenAI with ROS simulated robots that has recently started to be developed. Definitely, there are improvements necessary for the environments available in order to speed up training processes and also explicit the constraints that may arise throughout its use. Nonetheless, it is an important 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.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

Ezquerro, A.; Rodriguez, M.; and Tellez, R. 2018. OpenAI ROS package.

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.

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.