The Open-Source TEXPLORE Code Release for Reinforcement Learning on Robots

Todd Hester and Peter Stone

Department of Computer Science,
University of Texas at Austin,
Austin, TX, 78712
{todd,pstone}@cs.utexas.edu

Abstract. The use of robots in society could be expanded by using reinforcement learning (RL) to allow robots to learn and adapt to new situations on-line. RL is a paradigm for learning sequential decision making tasks, usually formulated as a Markov Decision Process (MDP). For an RL algorithm to be practical for robotic control tasks, it must learn in very few samples, while continually taking actions in real-time. In addition, the algorithm must learn efficiently in the face of noise, sensor/actuator delays, and continuous state features. In this paper, we present the TEXPLORE ROS code release, which contains TEXPLORE, the first algorithm to address all of these challenges together. We demonstrate TEXPLORE learning to control the velocity of an autonomous vehicle in real-time. TEXPLORE has been released as an open-source ROS repository, enabling learning on a variety of robot tasks.

Keywords: Reinforcement Learning, Markov Decision Processes, Robots.

1 Introduction

Robots have the potential to solve many problems in society by working in dangerous places or performing unwanted jobs. One barrier to their widespread deployment is that they are mainly limited to tasks where it is possible to hand-program behaviors for every situation they may encounter. Reinforcement learning (RL) [19] is a paradigm for learning sequential decision making processes that could enable robots to learn and adapt to their environment on-line. An RL agent seeks to maximize long-term rewards through experience in its environment.

Learning on robots poses at least four distinct challenges for RL:

1. The algorithm must learn from very few samples (which may be expensive or time-consuming).
2. It must learn tasks with continuous state representations.
3. It must learn good policies even with unknown sensor or actuator delays (i.e. selecting an action may not affect the environment instantaneously).
4. It must be computationally efficient enough to take actions continually in real-time.

S. Behnke et al. (Eds.): RoboCup 2013, LNAI 8371, pp. 536–543, 2014.

Addressing these challenges not only makes RL applicable to more robotic control tasks, but also many other real-world tasks.

While algorithms exist that address various subsets of these challenges, we are not aware of any that are easily adapted to address all four issues. RL has been applied to a few carefully chosen robotic tasks that are achievable with limited training and infrequent action selections (e.g. [11]), or allow for an off-line learning phase (e.g. [13]). However, to the best of our knowledge, none of these methods allow for continual learning on the robot running in its environment.

In contrast to these approaches, we present the TEXPLORE algorithm, the first algorithm to address all four challenges at once. The key insights of TEXPLORE are 1) to learn multiple domain models that generalize the effects of actions across states and target exploration on uncertain and promising states; and 2) to combine Monte Carlo Tree Search and a parallel architecture to take actions continually in real-time. TEXPLORE has been released publicly as an open-source ROS repository at: `http://www.ros.org/wiki/rl-texplore-ros-pkg`.

2 Background

We adopt the standard Markov Decision Process (MDP) formalism for this work [19]. An MDP consists of a set of states S, a set of actions A, a reward function $R(s,a)$, and a transition function $P(s'|s,a)$. In many domains, the state s has a factored representation, where it is represented by a vector of n state variables $s = \langle x_1, x_2, ..., x_n \rangle$. In each state $s \in S$, the agent takes an action $a \in A$. Upon taking this action, the agent receives a reward $R(s,a)$ and reaches a new state s', determined from the probability distribution $P(s'|s,a)$.

The value $Q^*(s,a)$ of a given state-action pair (s,a) is an estimate of the future reward that can be obtained from (s,a) and is determined by solving the Bellman equation: $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$, where $0 < \gamma < 1$ is the discount factor. The goal of the agent is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agent's lifetime. The optimal policy π is then $\pi(s) = \text{argmax}_a Q^*(s,a)$.

Model-based RL methods learn a model of the domain by approximating $R(s,a)$ and $P(s'|s,a)$ for each state and action. The agent can then plan on this model through a method such as value iteration [19] or UCT [10], effectively updating the Bellman equations for each state using their model. RL algorithms can also work without a model, updating the values of actions only when taking them in the real task. Generally model-based methods are more sample efficient than model-free methods, as their sample efficiency is only constrained by how many samples it takes to learn a good model.

3 TEXPLORE

In this section, we describe TEXPLORE [9], a sample-efficient model-based real-time RL algorithm. We describe how TEXPLORE returns actions in real-time in Section 3.1, and its approach to model learning and exploration in Section 3.2.

3.1 Real-Time Architecture

In this section, we describe TEXPLORE's real-time architecture, which can be used for a broad class of model-based RL algorithms that learn generative models. Most current model-based RL methods use a sequential architecture, where the agent receives a new state and reward; updates its model with the new transition $\langle s, a, s', r \rangle$; plans exactly on the updated model (i.e. by computing the optimal policy with a method such as value iteration); and returns an action from its policy. Since both the model learning and planning can take significant time, this algorithm is not real-time. Alternatively, the agent may update its model and plan on batches of experiences at a time, but this requires long pauses for the batch updates to be performed. Making the algorithm real-time requires two modifications to the standard sequential architecture: 1) utilizing sample-based approximate planning and 2) developing a novel parallel architecture called the Real-Time Model-Based Architecture (RTMBA) [7].

First, instead of planning exactly with value iteration, RTMBA uses UCT [10], a sample-based anytime approximate planning algorithm from the Monte Carlo Tree Search (MCTS) family. MCTS planners simulate trajectories (rollouts) from the agent's current state, updating the values of the sampled actions with the reward received. The agent performs as many rollouts as it can in the given time, with its value estimate improving with more rollouts. These methods can be more efficient than dynamic programming approaches in large domains because they focus their updates on states the agent is likely to visit soon rather than iterating over the entire state space.

In addition, we developed a Real-Time Model Based Architecture (RTMBA) that parallelizes the model learning, planning, and acting such that the computation-intensive processes (model learning and planning) are spread out over time. Actions are selected as quickly as dictated by the robot control loop, while still being based on the most recent models and plans available.

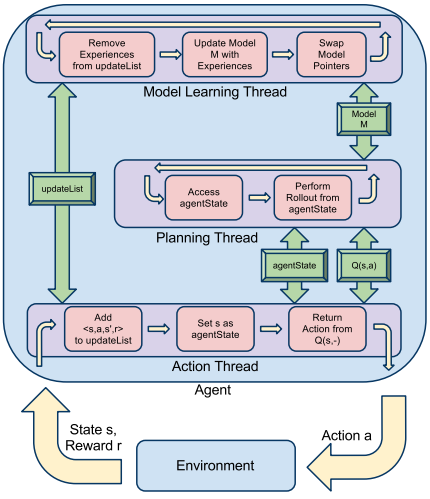

Fig. 1. Parallel Architecture

Since both model learning and planning can take significant computation (and thus wall-clock time), RTMBA places both of those processes in their own parallel threads in the background, shown in Figure 1. A third thread interacts with the environment, receiving the agent's new state and reward and returning the action given by the agent's current policy. The threads communicate through shared data structures protected by mutex locks. By de-coupling the action thread from the time-consuming model-learning and planning processes, RTMBA releases the algorithm from the need

to complete the model update and planning between actions. The full details of this architecture are described in [7,9].

3.2 Model Learning

While the parallel architecture we just presented enables TEXPLORE to operate in real-time, the algorithm must learn an accurate model of the domain quickly to learn the task with high sample efficiency. While tabular models are a common approach, they require the agent to take every action from each state once (or multiple times in stochastic domains), since they learn a prediction for each state-action separately. If we assume that the transition dynamics are similar across state-action pairs, we can improve upon tabular models by incorporating *generalization* into the model learning. TEXPLORE achieves high sample efficiency by combining this generalization with targeted exploration to improve the model as quickly as possible.

TEXPLORE approaches model learning as a supervised learning problem with (s, a) as the input and s' and r as the outputs the supervised learner is predicting. The supervised learner can make predictions about the model for unseen or infrequently visited states based on the transitions it has been trained on. TEXPLORE uses C4.5 decision trees [15] as the supervised learner to learn models of the transition and reward functions. The algorithm learns a model of the domain by learning a separate prediction for each of the n state features and reward. For continuous domains, the algorithm uses the M5 regression tree algorithm [16], which learns a linear regression model in each leaf of the tree, enabling it to better model continuous dynamics by building a piecewise linear model.

Each tree makes predictions for the particular feature or reward it is given based on a vector containing the n features of the state s along with the action a: $\langle x_1, x_2, ..., x_n, a\rangle$. To handle robots, which commonly have sensor and actuator delays, we provide the model with the past k actions, so that the model can learn which of these past actions is relevant for the current prediction.

Using decision trees to learn the model of the MDP provides us with a model that can be learned quickly with few samples. However, it is important that the algorithm focuses its exploration on the state-actions most likely to be relevant to the the task. To drive exploration, TEXPLORE builds multiple possible models of the domain in the form of a random forest [3]. The random forest model is a collection of m decision trees, where each tree is trained on only a subset of the agent's *experiences* ($\langle s, a, s', r\rangle$ tuples). Each tree in the random forest represents a hypothesis of what the true domain dynamics are. TEXPLORE then plans on the average of these predicted distributions, so that TEXPLORE balances the models predicting overly positive outcomes with the ones predicting overly negative outcomes. More details, including an illustrative example of this exploration, are provided in [9].

4 ROS Code Release

The TEXPLORE algorithm and architecture presented in this paper has been fully implemented, empirically tested, and released publicly as a Robot Operating System (ROS) repository at: `http://www.ros.org/wiki/rl-texplore-ros-pkg`.

ROS[1] is an open-source middleware operating system for robotics. It includes tools and libraries for commonly used functionality for robots such as hardware abstraction and messaging. In addition, it has a large ecosystem of users who release software packages for many common robot platforms and tasks as open source ROS repositories. With TEXPLORE released as a ROS repository, it can be easily downloaded and applied to a learning task on any robot running ROS with minimal effort. The goal of this algorithm and code release is to encourage more researchers to perform learning on robots using state-of-the-art algorithms.

The code release contains five ROS packages:

1. **rl_common**: This package includes files that are common to both reinforcement learning agents and environments.
2. **rl_msgs**: The **rl_msgs** package contains a set of ROS messages (`http://www.ros.org/wiki/rl_msgs`) that we have defined to communicate with reinforcement learning algorithms. The package defines an *action* message that the learning algorithm publishes, and a *state/reward* message that it subscribes to. To interface the learning algorithm with a robot already running ROS, one only needs to write a single node that translates *action* messages to actuator commands for the robot and translates robot sensor messages into *state/reward* messages for the learning algorithm. The advantages of this design are that we can apply RL on a wide variety of tasks by simply creating different RL interface nodes, while the actual learning algorithms and the robot's sensors/actuators interfaces are cleanly abstracted away.
3. **rl_agent**: This package contains the TEXPLORE algorithm. In addition, it includes several other commonly used reinforcement learning algorithms, such as Q-LEARNING [21], SARSA [17], R-MAX [2], and DYNA [18]. The agent package also includes a variety of model learning and planning techniques for implementing different model-based methods.
4. **rl_env**: This package includes a variety of benchmark RL domains such as Fuel World [8], Taxi [4], Mountain Car [19], Cart-Pole [19], Light World [12], and the simulated car velocity control domain presented in Section 5.
5. **rl_experiment**: This package contains code to run RL experiments without ROS message passing, by compiling both the experiment and agent together into one executable. This package is useful for running simulated experiments that do not require message passing to a robot.

5 Example Application

In this section, we present more details on how to interface the TEXPLORE algorithm with a robot already running ROS. In particular, we will demonstrate the ability of TEXPLORE to learn velocity control on our autonomous vehicle [1]. This task has a continuous state space, delayed action effects, and requires learning that is both sample efficient (to learn quickly) and computationally efficient (to learn on-line while controlling the car).

[1] `www.ros.org`

The task is to learn to drive the vehicle at a desired velocity by controlling the pedals. For learning this task, the RL agent's 4-dimensional state is the desired velocity of the vehicle, the current velocity, and the current position of the brake and accelerator pedals. The agent's reward at each step is -10.0 times the error in velocity in m/s. Each episode is run at 10 Hz for 10 seconds. The agent has 5 actions: one does nothing (no-op), two increase or decrease the desired brake position by 0.1 while setting the desired accelerator position to 0, and two increase or decrease the desired accelerator position by 0.1 while setting the desired brake position to 0. While these actions change the desired positions of the pedals immediately, there is some delay before the brake and accelerator reach their target positions.

The vehicle was already using ROS [14] as its underlying middleware. The **rl_msgs** package defines an *action* and *state/reward* message for the agent to communicate with an environment. To connect the agent with the robot, we wrote a ROS node that translates actions into actuator commands and sensor information into state/reward messages. The agent's action message is an integer between 0-4 as the action. The interface node translates this action into messages that provide commanded positions for the brake and throttle of the car. The interface node then reads sensor messages that provide the car's velocity and the true positions of the pedals, and uses these to create a state vector and reward for the agent and publishes a state/reward message. Figure 2 visualizes how the learning algorithm interfaces with the robot.

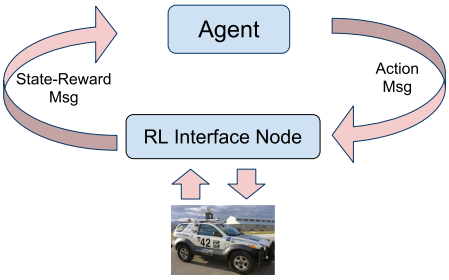

Fig. 2. ROS RL Interface

We ran five trials of Continuous TEXPLORE (using M5 regression trees) with $k = 2$ delayed actions on the physical vehicle learning to drive at 5 m/s from a start of 2 m/s. Figure 3 shows the average rewards over 20 episodes. In all five trials, the agent learned the task within 11 episodes, which is less than 2 minutes of driving time. This experiment shows that our TEXPLORE code release can be used to learn a robotic task that has continuous state and actuator delays in very few samples while selecting actions continually in real-time. In addition to learning to control the velocity of an autonomous vehicle, a variant of TEXPLORE has also been used to learn how to score penalty kicks in the Standard Platform League of RoboCup [6]. More results with comparisons against other state-of-the-art algorithms are available in [9,5].

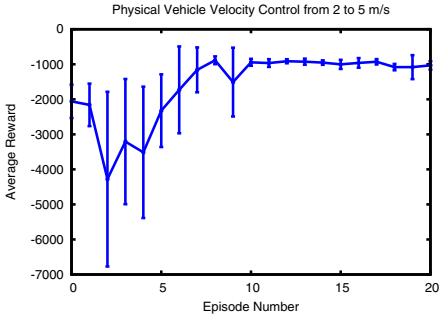

Fig. 3. Average rewards of TEXPLORE learning to control the physical vehicle from 2 to 5 m/s

6 Related Work

Since TEXPLORE is addressing many challenges, there is ample related work on each individual challenge, although no other methods address all four challenges. Work related to the algorithmic components of TEXPLORE is detailed in [9,5]. In this section, we look at related reinforcement learning code releases.

Similar to our **rl_msgs** package which defines ROS messages for an agent to communicate with an environment, RL-GLUE [20] defines similar messages for general use. The RL-LIBRARY[2] builds off of this as a central location for sharing RL-GLUE compatible RL projects. However, currently the library only contains the SARSA(λ) [17] algorithm, and does not have any algorithms which focus on learning on robots.

RLPARK[3] is a Java-based RL library that includes both learning methods and methods for the real-time display of learning data. RLPARK contains a variety of algorithms for both control and prediction, both on-line or off-line. It does contain some algorithms that target learning on robots, however it does not provide a ROS interface for connecting with other robots.

The York Reinforcement Learning Library (YORLL)[4] focuses on multi-agent learning, however it works for single-agent learning as well. It contains a few basic algorithms like Q-LEARNING [21] and SARSA [17], and has various options for handling multiple agents. However, it does not have any algorithms which focus on learning on robots.

Teaching Box[5] is a learning library that is focused on robots. It contains some algorithms for reinforcement learning as well as learning from demonstration. It uses RL-GLUE to interface the agent and environment, rather than using ROS.

7 Conclusion

We identify four properties required for RL to be practical for continual, on-line learning on a broad range of robotic tasks: it must (1) be sample-efficient, (2) work in continuous state spaces, (3) handle sensor and actuator delays, and (4) learn while taking actions continually in real-time. This article presents the code release of TEXPLORE, the first algorithm to address all of these challenges. Note that there are other challenges relevant to robotics that TEXPLORE does not address, such as *partial observability* or *continuous actions*, which we leave for future work.

The code release provides the TEXPLORE algorithm along with a variety of other commonly used RL algorithms. It also contains a number of common benchmark tasks for RL. The release includes a set of ROS messages for RL which define how an RL agent can communicate with a robot. Using these defined messages, it is easy to interface TEXPLORE or the other algorithms provided in the code release with robots already running ROS.

[2] `library.rl-community.org/wiki/Main_Page`
[3] `rlpark.github.com/`
[4] `www.cs.york.ac.uk/rl/software.php`
[5] `amser.hs-weingarten.de/en/teachingbox.php`

Acknowledgements. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-0917122), ONR (N00014-09-1-0658), and the FHWA (DTFH61-07-H-00030).

References

1. Beeson, P., O'Quin, J., Gillan, B., Nimmagadda, T., Ristroph, M., Li, D., Stone, P.: Multiagent interactions in urban driving. Journal of Physical Agents 2(1), 15–30 (2008)
2. Brafman, R., Tennenholtz, M.: R-Max - a general polynomial time algorithm for near-optimal reinforcement learning. In: IJCAI (2001)
3. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
4. Dietterich, T.: The MAXQ method for hierarchical reinforcement learning. In: ICML, pp. 118–126 (1998)
5. Hester, T.: TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains. PhD thesis, Department of Computer Science, University of Texas at Austin, Austin, TX (December 2012)
6. Hester, T., Quinlan, M., Stone, P.: Generalized model learning for reinforcement learning on a humanoid robot. In: ICRA (May 2010)
7. Hester, T., Quinlan, M., Stone, P.: RTMBA: A real-time model-based reinforcement learning architecture for robot control. In: ICRA (2012)
8. Hester, T., Stone, P.: Real time targeted exploration in large domains. In: ICDL (August 2010)
9. Hester, T., Stone, P.: TEXPLORE: Real-time sample-efficient reinforcement learning for robots. Machine Learning 87, 10–20 (2012)
10. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
11. Kohl, N., Stone, P.: Machine learning for fast quadrupedal locomotion. In: AAAI Conference on Artificial Intelligence (2004)
12. Konidaris, G., Barto, A.G.: Building portable options: Skill transfer in reinforcement learning. In: IJCAI (2007)
13. Ng, A., Kim, H.J., Jordan, M., Sastry, S.: Autonomous helicopter flight via reinforcement learning. In: NIPS (2003)
14. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.: ROS: An open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
15. Quinlan, R.: Induction of decision trees. Machine Learning 1, 81–106 (1986)
16. Quinlan, R.: Learning with continuous classes. In: 5th Australian Joint Conference on Artificial Intelligence, pp. 343–348. World Scientific, Singapore (1992)
17. Rummery, G., Niranjan, M.: On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166. Cambridge University Engineering Department (1994)
18. Sutton, R.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: ICML, pp. 216–224 (1990)
19. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
20. Tanner, B., White, A.: RL-Glue: Language-independent software for reinforcement-learning experiments. JMLR 10, 2133–2136 (2009)
21. Watkins, C.: Learning From Delayed Rewards. PhD thesis. University of Cambridge (1989)