toddhester / **rl-texplore-ros-pkg**

Automatically exported from code.google.com/p/rl-texplore-ros-pkg

72 commits | **2** branches | **0** releases | **2** contributors

Branch: **master** | **New pull request** | **Find file** | **Clone or download**

toddhester Update package.xml — Latest commit `9c927f8` on 6 Sep 2015

src	Update package.xml	2 years ago
.catkin_workspace	catkinized version of rl-texplore-ros-pkg	2 years ago
.gitignore	catkinized version of rl-texplore-ros-pkg	2 years ago
README	Update Readme	2 years ago

README

rl-texplore-ros-pkg

Reinforcement Learning Framework and Repository

Todd Hester

4 May 2011

July 2015 Update

The indigo-devel branch of this repository contains a catkinized version of the original codebase. If you are using ROS Groovy or newer, you should check out this version.

The stacks directory contains a ROS stack for reinforcement learning (RL). The code in this repository provides agents, environments, and multiple ways for them to communicate (through ROS messages, or by including the agent and environment libraries). There are 5 packages in the repository:

```
rl_common:     Some files that are common to both agents and environments.
rl_msgs:       Definitions of ROS messages for agents and envs to communicate.
rl_agent:      A library of some RL agents including Q-Learning and TEXPLORE.
rl_env:        A library of some RL environments such as Taxi and Fuel World.
rl_experiment: Code to run some RL experiments without ROS message passing.
```

Documentation for these packages is here:

http://www.ros.org/wiki/reinforcement_learning

And there is a more thorough tutorial here:

http://www.ros.org/wiki/reinforcement_learning/Tutorials/Reinforcement%20Learning%20Tutorial

Working with the ROS build system is difficult at first. Expect a steep learning curve. Working through some of the ROS tutorials is definitely worthwhile when starting out. Documentation:

http://www.ros.org/wiki/ROS.

The interfaces themselves are quite good. So are the visualization and debugging tools. And, there is an impressive amount of useful robotics software already available in the ROS package system.

Directions:

(1) Check out a copy of the RL code base with ROS interfaces

Our packages have their own Github project. Check out the trunk of rl-texplore-ros-pkg:

$ git clone https://github.com/toddhester/rl-texplore-ros-pkg.git

This creates a copy of the source tree without commit access.

To commit changes yourself, create a github account and e-mail the project owners requesting commit access.

(2) Install ROS as in http://www.ros.org/wiki/ROS/Installation

This is time-consuming, but not especially difficult. Be careful

```
    to follow the directions exactly.  It works fine on Ubuntu and
    can be made to work for Mac OSX with some effort.  Windows will
    probably not work, but there are future plans to support it.

    Add the ROS environment setup to the end of your .bashrc:

    $ echo "source ~/ros/setup.sh" >> ~/.bashrc

    Append our packages as well as the ROS ones (this example does it
    from the shell, or you can use any editor):

    $ cat <<'EOF' >> ~/.bashrc
    export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:~/svn/rl-texplore-ros-pkg
    EOF


(3) Log out and log back in.


(4) Try to run some ROS commands:

    $ rospack find rl_agent
    $ roscd rl_agent
    $ rosmake rl_agent

    This should compile cleanly.


(5) Try to run an RL experiment. First compile the code:

    $ rosmake rl_agent
    $ rosmake rl_env
```

Now we want to run the code to run our experiment. We need to start both an agent and an environment, and they will interact by passing rl_msg messages back and forth. Open three tabs in your terminal. Run each of these commands in a different tab:

```
$ roscore
$ rosrun rl_agent agent --agent qlearner
$ rosrun rl_env env --env taxi
```

The second line starts an Q-Learner agent. There are other options to start different agents. The third line starts a taxi environment. There are other options here as well to start different environments. Options for both of these can be seen with the -h option. In the agent window, you'll see it print out the sum of rewards at the end of each episode.

(6) ROS has some nice tools for plotting and recording data. To look at the messages being passed during the experiment, type:

```
$ rostopic list
```

This prints a list of the current messages to the screen. Let's view a particular message:

```
$ rostopic echo /rl_env/rl_state_reward
```

Here we can see the contents of the state reward message being sent by the environment, which contains the current state vector, the reward, and a boolean of whether this is a terminal state or not.

Another thing we can do using ros is make live plots. To make a live plot

```
    of reward per episode, type:

    $ rxplot /rl_agent/rl_experiment_info/episode_reward

    One more thing that can be done is to record all the messages being passed
    in order to view, replay, or plot them later. To record all the messages
    sent by the agent, type:

    $ rosbag record /rl_agent/rl_action /rl_agent/rl_experiment_info

    Now we can play them back in place of an agent, and just have the saved
    messages control the agent in the environment. Kill the agent, and let's
    play back the recorded actions:

    $ rosbag play <bagfile>


(7) Finally, we can also run experiments without passing ROS messages using
    the rl_experiment package. It includes both the agent and env libraries
    directly, instead of doing message passing. First, compile this package:

    $ rosmake rl_experiment

    To run an experiment, we pass an agent and an environment on the
    command line, and we can also pass other options. Once it starts running,
    it will print out the sum of rewards per episode for episodic tasks, or
    the per step rewards for non-episodic tasks. Let's try running R-Max with
    M = 1 on deterministic Fuel World:

    $ rosrun rl_experiment experiment --agent rmax --m 1 --env fuelworld --deterministic
```