RSLGym

Release 0.2.0

Takahiro Miki, Joonho Lee, Yuntao Ma, Pascal Egli

Mar 11, 2022

CONTENTS

This is the reinforcement learning framework from the Robotics System Lab (RSL) at ETH Zurich.

It provides an interface to train reinforcement learning agents that are simulated in the RaiSim physics engine.

For efficiency, experience generation in RaiSim is parallelized using a vectorized environment in C++. The vectorized environment is wrapped using pybind11 such that it can be used with RL algorithms implemented in python.

Currently, we provide examples for training agents with custom PPO and TRPO implementations and the algorithms provided by PFRL (https://github.com/pfnet/pfrl) which are implemented using pyTorch.

CHAPTER

ONE

INSTALL DEPENDENCIES

1.1 Folder Setup

To avoid conflicts with other libs it is recommended to install everything locally. Therefore you will need two folders:

- **WORKSPACE**: where you clone all the git repos (e.g. ~/rslgym_ws)
- **LOCAL_INSTALL**: where you install all the libs (e.g. ~/rslgym_build)

Define the LOCAL_INSTALL variable in your .bashrc and add it to the library search path, i.e. add the following lines to **~/.bashrc**:

```
export LOCAL_INSTALL=/home/<user_account>/rslgym_build
export LD_LIBRARY_PATH=$LOCAL_INSTALL/lib:$LD_LIBRARY_PATH
```

1.2 RaiSimLib

Install the RaiSim rigid body simulator.

- Clone repo and install additional dependencies:

  ```
  cd $WORKSPACE
  git clone https://github.com/raisimTech/raisimLib.git
  sudo apt install cmake
  ```

- Install python3 depencency:

  ```
  sudo apt install python3-dev
  ```

- Follow RaiSim installation instructions: https://raisim.com/sections/Installation.html
- To use RaiSim you need a valid license. You can apply for it here: https://raisim.com/sections/License.html
- Place the license in **~/.raisim** and rename the file to **activation.raisim** (create the folder if necessary)
  - The license can also be placed at another location. This needs to be specified when instantiating the vectorized environment in python, see rslgym/ wrapper/ script/ RaisimGymVecEnv.py.

1.3 RaiSimOgre

Install the ogre visualizer for RaiSim.

- Follow the instructions here: https://github.com/raisimTech/raisimOgre

1.4 PyBind11

Install the PyBind11 library:

```
cd $WORKSPACE/raisimLib/thirdParty/pybind11
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=$LOCAL_INSTALL -DPYBIND11_
↪TEST=FALSE
make install
```

CHAPTER

TWO

INSTALL RSLGYM

2.1 Virtualenv

To avoid conflicts with other libs it is recommended to install everything locally in a **virtual environment**:

```
pip3 install virtualenv
mkdir ~/.virtualenvs
pip3 install virtualenvwrapper
```

Add the following lines to **~/.bashrc**:

```
export WORKON_HOME=~/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source ~/.local/bin/virtualenvwrapper.sh
```

Open a new terminal tab and create your virtual environment:

```
source ~/.profile
mkvirtualenv --system-site-packages rslgym
```

2.2 PyTorch

Activate the virtualenvironment (if not already active):

```
workon rslgym
```

Install the latest stable version of PyTorch using pip following the instructions here https://pytorch.org/get-started/locally/

2.3 RSLGym

Clone and install RSLGym (inside the virtual environment):

```
cd $WORKSPACE
git clone <rslgym_repo>
cd rslgym
pip3 install -e .
```

2.4 Other Dependencies

Yaml-cpp for hyperparameter loading:

```
sudo apt install libyaml-cpp-dev
```

Dependencies for openAI examples:

```
sudo apt install ubuntu-restricted-extras swig
```

Valgrind for debugging **environment.hpp**:

```
sudo apt install valgrind
```

CHAPTER

THREE

EXAMPLES

We provide examples for training RL agents that are simulated in RaiSim and the openAI Gym.

The examples are located in rslgym/examples/envs.

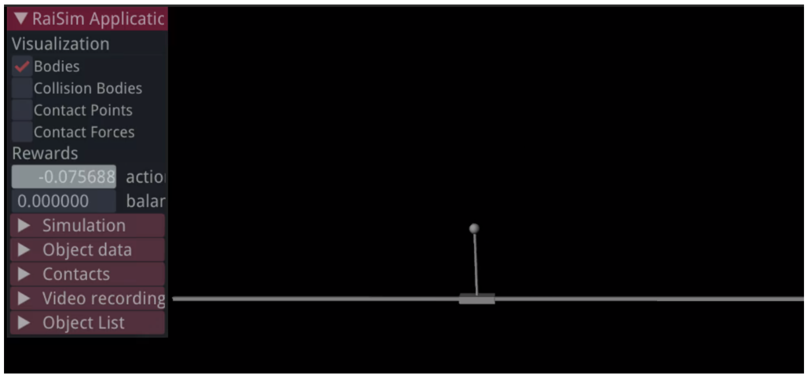
To run the examples that use PFRL algorithms install PFRL in your virtual environment:

```
pip3 install pfrl
```

To run the openAI box2D examples install box2D:

```
pip3 install box2d-py box2d-kengz
```

3.1 RaiSim Cart Pole



Simple cart pole balancing example using RaiSim.

With the provided hyperparameters the agent is able to balance the pole. However, the performance can always be improved with more tuning. Feel free to contribute if you find better settings.

1. Activate your virtual python environment:

```
workon <name-of-rslgym-virtualenv>
```

2. Build the cart pole environment. Inside **rslgym/examples/envs/cart_pole** run:

```
rslgym build . --name cart_pole --CMAKE_PREFIX_PATH $LOCAL_INSTALL
```

3. Train the cart pole agent running the ***_train.py** scrips in cart_pole/scripts.

The scripts regularly test the current performance of the agent and store the weights of the neural networks and a video in **cart_pole/runs/<time_and_data_when_training_started>**.

The training configuration and hyperparameters can be configured in the **cfg.yaml** file:

```
./rsl_ppo_train.py --cfg_name=<optional-path-to-cfg-default:cfg.yaml>
```

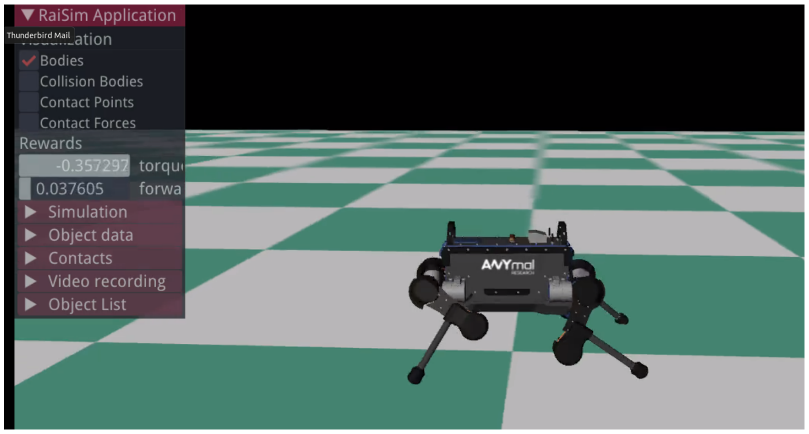
4. The learning progress can be monitored using **tensorboard**. In another terminal, activate your virtual environment and run:

```
tensorboard --logdir=<absolute-path-to-folder-containing-training-weights>
```

5. The performance of the agent can also be tested using the ***_test.py** scripts. Videos of the tests are stored in the folder where the policy weights are stored. In the folder **testing_<tested_iteration>**:

```
./rsl_ppo_test.py --weight <path_to_folder_with_weights> -i <iteration_to_test>
```

3.2 RaiSim ANYmal Bear

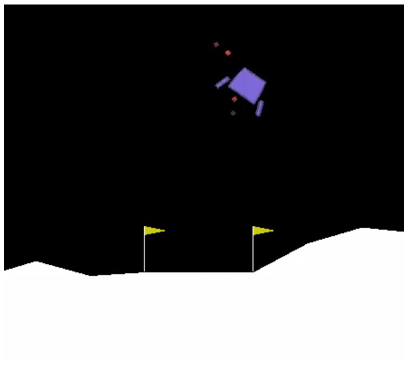


Simple locomotion example with ANYmal Bear in RaiSim.

To train the agent, follow the same steps as in the cart_pole example above. Build the environment with the name **anymal**:

```
rslgym build . --name anymal --CMAKE_PREFIX_PATH $LOCAL_INSTALL
```

3.3 OpenAI Gym



Example of training openAI gym environments with PPO.

No need to compile anything. Just run the training scripts in your virtual environment.

CHAPTER

FOUR

CREATE YOUR OWN ENVIRONMENT

You need to implement a single environment instance. The frameworks parallelizes this single environment by creating a **vectorized environment**.

Your environment needs to have at least Environment.hpp which inherits from **rslgym/rslgym/wrapper/include/RaisimGymEnvBase.hpp**:

```
virtual void init() = 0;
virtual void reset() = 0;  // resets environment at the beginning or when terminated
virtual void setSeed(int seed) = 0;  // sets random number generator(s) seed(s)
virtual void observe(Eigen::Ref<EigenVec> ob) = 0;  // returns environment observations
virtual float step(const Eigen::Ref<EigenVec>& action) = 0;  // returns reward
virtual bool isTerminalState(float& terminalReward) = 0;  //substitutes terminal reward
virtual void setInfo(const std::unordered_map<std::string, EigenVec>& info) = 0;  //␣
↪transfer info to the environment
virtual void updateInfo() {};  //transfer info from the environment
```

setInfo() and **updateInfo()** allow you to transfer information from the python code to the environment and the other way around. This can be useful to monitor variables in the environment or to update curriculum factors in the environment.

You can define any additional info to this variable like below. This will be a dictionary on the python side:

```
void updateInfo() final {
  info_["gc"] = gc_.cast<float>();
  info_["gv"] = gv_.cast<float>();
  info_["rewards"] = Eigen::Vector2d(torqueReward_, forwardVelReward_).cast<float>();
}
```

In the same way, you can get python dictionary data from python side in Environment.hpp:

```
void setInfo(const std::unordered_map<std::string, EigenVec>& info) {
  for (auto &kv: info) {
    const auto& key = kv.first;
    const auto& value = kv.second;
    // do whatever based on info from python
    // if(key == "curriculum")
    //    updateCurriculum(value);
  }
}
```

4.1 Building, Naming and Usage

You can define the name under which the python module will be built in the **environment.hpp**.

```
#define ENVIRONMENT_NAME <my-env-name>
```

When building the environment, you can pass the name of the python package which will be called rslgym_wrapper_<your_package_name>:

```
rslgym build . --name <your_package_name> --CMAKE_PREFIX_PATH $LOCAL_INSTALL
```

Then, in python, you can include the environment using:

```
from rslgym_wrapper_<your_package_name> import <my-env-name>
```

For more information about the **build** function, call:

```
rslgym build --help
```

4.2 Debugging

To debug your environment.hpp and catch e.g. nasty segmentation faults you can build your environment using the **–debug** flag:

```
rslgym build . --name cart_pole --debug --CMAKE_PREFIX_PATH $LOCAL_INSTALL
```

This will create a c++ executable which you can debug with **valgrind** using this command:

```
rslgym debug <render/no_render> --name cart_pole --resource <relative-path-to-rsc-folder-
↪default:/rsc> --cfg <relative-path_to_cfg-default:/cfg.yaml>
```

For more information about the **debug** function, call:

```
rslgym debug --help
```

4.3 Additional Libraries

RSLGym includes and links **raisim**, **eigen3** and **OpenMP** libraries. If you want to use **additional libraries** in your environment, you can add prebuilt or custom libraries to the CMake variable *EXTRA_LIBS* through a CMake include file. Note that the libraries must be built with **-fPIC** option:

```
rslgym build . --name <your_package_name> --CMAKE_INCLUDE_FILE <path_to_my_cmake_include_
↪file>
```

An example of a CMake include file to build custom libraries can be found below.

```
add_library(<MY_LIBRARY_NAME> ${CMAKE_CURRENT_LIST_DIR}/<path_to_my_source_code_relative_
↪to_this_file>.cpp)
target_include_directories(<MY_LIBRARY_NAME> PUBLIC ${CMAKE_CURRENT_LIST_DIR}/<path_to_
↪my_include_directories>)
```

(continues on next page)

(continued from previous page)

```
target_compile_options(<MY_LIBRARY_NAME> PRIVATE -mtune=native -fPIC -O3)
set(EXTRA_LIBS <MY_LIBRARY_NAME>)
```