# GRL: a generic C++ reinforcement learning library

Wouter Caarls <mailto:wouter@caarls.org>

June 11th, 2015

## 1 Introduction

## 2 Directory structure

```
.
|-- base                    Base library
|   |-- include             Header files
|   `-- src                 Source files
|       |-- agents          Agents (fixed, black box, td)
|       |-- discretizers    Action discretizers
|       |-- environments    Environments (pendulum, cart-pole)
|       |-- experiments     Experiments (online, batch)
|       |-- policies        Control policies (PID, Q-based)
|       |-- predictors      Value function predictors (SARSA, AC)
|       |-- projectors      State projectors (tile coding, fourier)
|       |-- representations Representations (linear, ann)
|       |-- samplers        Action samplers (greedy, e-greedy)
|       |-- traces          Elibility traces (accumulating, replacing)
|       `-- visualizations  Visualizations (value function, policy)
|-- addons                  Optional modules
|   |-- cma                 CMA-ES black-box optimizer
|   |-- gl                  OpenGL-based visualizations
|   |-- glut                GLUT-based visualizer
|   |-- llr                 Locally linear regression representation
|   |-- matlab              Matlab interoperability
|   |-- muscod              Muscod interoperability
|   |-- odesim              Open Dynamics Engine environment
|   |-- rbdl                Rigid Body Dynamics Library dynamics
|   `-- ros                 ROS interoperability
|-- bin                     Python binaries (configurator)
|-- externals               Imported external library code
|-- cfg                     Sample configurations
```

```
|-- share                    Misc files
|-- tests                    Unit tests
|-- CMakeLists.txt           CMake instructions to build everything
`-- grl.cmake                CMake helper functions
```

# 3 Prerequisites

GRL requires some libraries in order to compile. Which ones exactly depends on which agents and environments you would like to build, but the full list is

- Git
- GCC (including g++)
- Boost (for shared_ptr)
- Eigen
- GLUT
- QT4 (including the OpenGL bindings)
- TinyXML
- MuParser
- ODE, the Open Dynamics Engine
- Python (including Tkinter and the yaml reader)

On Ubuntu 14.04, these may be installed with the following command:

```
wcaarls@vbox:~$ git cmake g++ libboost-dev1 libeigen3-dev1 \
libgl1-mesa-dev-lts-utopic freeglut3-dev1 libqt4-opengl-dev \
libtinyxml-dev libmuparser-dev libode-dev1 python-yaml python-tk1 \
```

# 4 Building

GRL may be built with or without ROS's catkin. When building with, simply merge `grl.rosinstall` with your catkin workspace

```
wcaarls@vbox:~$ mkdir indigo_ws
wcaarls@vbox:~$ cd indigo_ws
wcaarls@vbox:~/indigo_ws$ rosws init src /opt/ros/indigo
wcaarls@vbox:~/indigo_ws$ cd src
wcaarls@vbox:~/indigo_ws/src$ rosws merge /path/to/grl.rosinstall
wcaarls@vbox:~/indigo_ws/src$ rosws up
wcaarls@vbox:~/indigo_ws/src$ cd ..
wcaarls@vbox:~/indigo_ws$ catkin_make
```

Otherwise, follow the standard CMake steps of (in the `grl` directory)

```
wcaarls@vbox:~/src/mprl$ mkdir build
wcaarls@vbox:~/src/mprl$ cd build
wcaarls@vbox:~/src/mprl/build$ cmake ..
-- The C compiler identification is GNU 4.8.2
...
wcaarls@vbox:~/src/mprl/build$ make
Scanning dependencies of target yaml-cpp
...
```

# 5 Build environment

The whole `grl` system is built as a single package, with the exception of `mprl_msgs`. This is done to facilitate building inside and outside catkin. There is one `CMakeLists.txt` that is used in both cases. The ROS interoperability is selectively built based on whether `cmake` was invoked by `catkin_make` or not.

Modules are built by calling their respective `build.cmake` scripts, which is done by `grl_build_library`. The include directory is set automatically, as is an `SRC` variable pointing to the library's source directory.

The build system has a simplistic dependency management scheme through `grl_link_libraries`. This calls the `link.cmake` files of the libraries on which the current library depends. Typically they will add some `target_link_libraries` and add upstream dependencies. `grl_link_libraries` also automatically adds the upstream library's include directory.

# 6 Class structure

## 6.1 Class factories

## 6.2 Configuration

# 7 Matlab interface

If Matlab is installed (and can be found on the path), a MEX interfaces for the agents and environments is built. If you want to use these, make sure that you're building with a compatible compiler, both by setting the `CC` and `CXX` variables in your call to `cmake` and by correctly configuring `mex`.

## 7.1 Environments

To initialize an environment, call

```
>> spec = grl_env('cfg/matlab/pendulum_swingup.yaml');
```

Where the argument specifies a configuration file that has a top-level 'environment' tag. `spec` gives some information about the environment, such as number of dimensions, minimum and maximum values, etc. Next, retrieve the first observation of an episode with

```
>> o = grl_env('start');
```

where `o` is the observation from the environment. All following steps should be called using

```
>> [o, r, t] = grl_env('step', a);
```

where `a` is the action suggested by the agent, `r` is the reward given by the environment and `t` signals termination of the episode. If `t` is 2, the episode ended in an absorbing state. When all episodes are done, exit cleanly with

```
>> grl_env('fini');
```

## 7.2 Agents

To initialize the agent, use

```
>> grl_agent('init', 'cfg/matlab/sarsa.yaml');
```

Where the argument specifies a configuration file that has a top-level 'agent' tag. Next, give the first observation of an episode with

```
>> a = grl_agent('start', o);
```

where `o` is the observation from the environment and `a` is the action suggested by the agent. All following steps should be called using

```
>> a = grl_agent('step', r, o);
```

where `r` is the reward given by the environment. To signal the end of an episode (absorbing state), use

```
>> a = grl_agent('end', r);
```

To end an episode without an absorbing state, simply start a new one. To exit cleanly after all epsiodes are finished (which also allows you to reinitialize the agent with different options), call

```
>> grl_agent('fini');
```