GRL: a generic C++ reinforcement learning library

Wouter Caarls <mailto:wouter@caarls.org>

October 5th, 2015

1 Introduction

GRL is a C++ reinforcement learning library that aims to easily allow evaluating different algorithms through a declarative configuration interface.

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)
|       |-- solvers           MDP solvers (VI, rollout-based)
|       |-- 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
|   |-- lqr                   Linear Quadratic Regulator solver
|   |-- 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
|   `-- taskmaster            Taskmaster parameter study example
|-- 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++)
- Eigen
- GLUT
- ZLIB
- QT4 (including the OpenGL bindings)
- TinyXML
- MuParser
- ODE, the Open Dynamics Engine
- Python (including Tkinter and the yaml reader)
- Lua

On Ubuntu 16.04, these may be installed with the following command:

```
wcaarls@vbox:~$ git cmake g++ libeigen3-dev \
libgl1-mesa-dev freeglut3-dev libz-dev libqt4-opengl-dev \
libtinyxml-dev libmuparser-dev libode-dev python-yaml python-tk \
liblua5.1-dev
```

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

5 Running

The most important executables in grl are the deployer (`grld`) and configurator (`grlc`). The configurator allows you to generate configuration files easily. To see an example, run

```
wcaarls@vbox:~/src/grl/bin$ ./grlc ../cfg/pendulum/sarsa_tc.yaml
```

More information on the configurator can be found in Section 8. Once you have configured your experiment, you can either run it directly from the configurator, or save it and run it using the deployer. For example:

```
wcaarls@vbox:~/src/grl/build$ ./grld ../cfg/pendulum/sarsa_tc.yaml
```

6 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.

7 Class structure

Most classes in grl derive from `Configurable`, a base class that standardizes configuration such that the object hierarchy may be constructed declaratively in a configuration file. Directly beneath `Configurable` are the abstract base classes defining the operation of various parts of the reinforcement learning environment, being:

`Agent` RL-GLUE[1] style agent interface, receiving observations in an episodic manner and returning actions.

`Discretizer` Provides a list of discrete points spanning a continuous space.

`Environment` RL-GLUE style environment interface, receiving actions and returning observations.

`Experiment` Top-level interface, which typically calls the agent and environment in the correct manner, but may in general implement any experiment.

`Optimizer` Black-box optimization of control policies, suggesting policies and acting on their cumulative reward.

`Policy` Basic control policy that implements the state-action mapping.

`Predictor` Basic reinforcement learning interface that uses transitions to predict a value function or model.

`Projector` Projects an observation onto a feature vector, represented as a `Projection`.

`Representation` Basic supervised learning interface that uses samples to approximate a function. As such, it generally supports reading, writing and updating of any vector-to-vector mapping.

`Sampler` (Stochastically) chooses an item from a vector of (generally unnormalized) values.

`Trace` Stores a trace of projections with associated eligibilities that can be iterated over.

[1] `http://http://glue.rl-community.org`

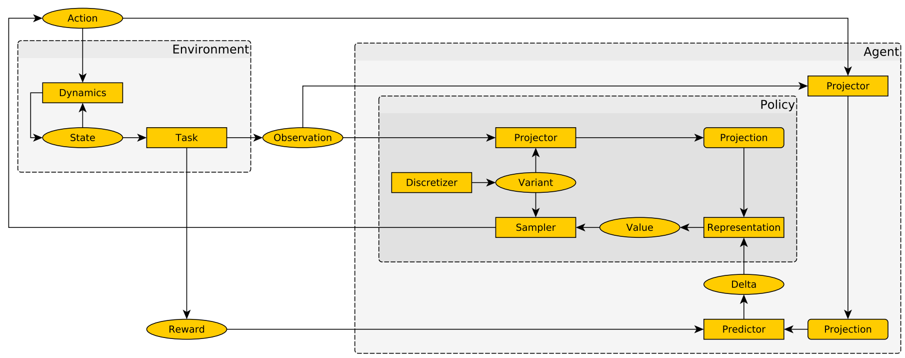

Figure 1: Information flow diagram for regular TD control. Rectangles (and dashed rectangles) are `Configurable` objects, while the others are the data passed between them.

`Visualization` Draws on the screen to visualize some aspect of the learning process.

`Visualizer` Keeps track of visualizations and provides the interface to the graphics subsystem.

Each abstract base class is generally implemented in various concrete classes, with or without additional hierarchy. A list can be requested by running

```
wcaarls@vbox:~/src/grl/bin$ ./grlq
```

and is also available in the appendices of this document.

A typical example of the information flow between the various classes can be seen in Figure 1, which depicts the standard TD control setting.

7.1 Configuration

Each `Configurable` subclass must define its type and a short description using the `TYPEINFO` macro:

```
class OnlineLearningExperiment : public Experiment
{
  public:
    TYPEINFO("experiment/online_learning", "Interactive learning experiment")

  /* ... */
};
```

This textual description of the type is used to facilitate user configuration by limiting the selection of parameter values, as well as enforcing the type hierarchy.

In general, the textual description should follow the C++ class hierarchy, but this is not obligatory.

The basic `Configurable` interface has three important functions:

7.1.1 `request`

```
virtual void request(ConfigurationRequest *config);
```

`request` is called by the configurator to find out which parameters the object requires to be set, and which parameters it exports for other objects to use. To do this, it should extend the given `ConfigurationRequest` by pushing configuration request parameters (`CRP`s). A basic `CRP` has the following signature:

```
CRP(string name, string desc, TYPE value)
```

where TYPE is one of int, double, Vector, or string. For example:

```
config->push_back(CRP("steps", "Number of steps per learning run", steps_));
config->push_back(CRP("output", "Output base filename", output_));
```

The `value` argument is used both to determine the type of the parameter and the default value suggested by the configurator. `request` may also be called while the program is running, in which case it is expected to return the current value of all parameters.

To use other `Configurable` objects as parameters, use

```
CRP(string name, string type, string desc, Configurable *value)
```

The extra `type` field restricts which `Configurable` objects may be used to configure this parameter. Only objects whose `TYPEINFO` starts with the given `type` are eligible. For example:

```
config->push_back(CRP("policy", "policy/parameterized",

                      "Control policy prototype", policy_));
```

restricts the `"policy"` parameter to classes derived from `ParameterizedPolicy`. Note that this extra type hierarchy is related to, but not derived from the actual class hierarchy. Care must therefore be taken in the correct usage of `TYPEINFO`.

Some parameters are not requested, but rather *provided* by an object. In that case. These have the following signature:

```
CRP(string name, string type, string desc, CRP::Provided)
```

Examples of provided parameters are the number of observation dimensions (provided by `Tasks`) or the current system state (provided by some `Environments`).

7.1.2 `configure`

```
virtual void configure(Configuration *config);
```

`configure` is called after all parameters (including other `Configurable` objects) have been initialized. The parameter values may be accessed using mapping syntax (`config["parameter"]`). Note that `Configurable` objects are passed as void pointers and must still be cast to their actual class:

```
steps_ = config["steps"];
output_ = config["output"].str();
policy_ = (ParameterizedPolicy*)config["policy"].ptr();
```

Note the use of `.str()` and `.ptr()` for strings and objects, respectively. Provided parameters should be written to the configuration instead of read, like so:

```
config.set("state", state_);
```

7.1.3 `reconfigure`

```
virtual void reconfigure(const Configuration *config);
```

Some parameters may be defined as reconfigurable by appending `CRP::Online` to the respective `CRP` signature. In the case of a reconfiguration, `reconfigure` will be called with the new values of those parameters in `config`. `reconfigure` may also be used for general messaging, equivalent to RL-GLUE's `message` calls. In that case, it is often helpful to reconfigure all objects in the object hierarchy, which can be done using

```
void Configurable::walk(const Configuration &config);
```

Examples are resetting the hierarchy for a new run (`config["action"] = "reset"`) or saving the current state of all memories (`config["action"] = "save"`). In the latter case, `Configurable::path()` may be used to determine an object's location in the object hierarchy.

7.2 Roles

While using the configurator, the user often has to select previously defined objects as the value of certain parameters. If all such previously defined objects are presented as possibilities, the list would quickly grow very large. To make setting these parameters easier, a class may have various *roles* while providing the same interface. In that case, only previously defined objects with a role that starts with the requested role are valid choices.

An example is a `Representation`, which may represent a state-value function, action-value function, control policy or model. Each has a different number of inputs and outputs, and chosing the wrong representation will result in mismatches. An object requesting a `Representation` may therefore request a certain role. For example:

```
config->push_back(CRP("representation", "representation.value/action",

                      "Q-value representation", representation_));
```

requests any representation that represents action-values. A newly defined `representation` will do, of course, but from the previously defined ones only the ones with the right role are eligible.

The same strategy is used for basic types, for example:

```
config->push_back(CRP("outputs", "int.action_dims",
                      "Number of outputs", outputs_, CRP::System));
```

make sure the only suggested previously defined values for the `"outputs"` parameter are ones with the `"action_dims"` role. As an added convenience, if the parameter is defined as a *system parameter* (`CRP::System`), meaning that the choice is not free but rather defined by the structure of the configuration, and only a single value was previously defined, that value is automatically used.

The role that needs to be requested may depend on the role of the requesting object itself. In that case, the following signature for `request` should be used:

```
virtual void request(const std::string &role, ConfigurationRequest *config);
```

8 Configurator

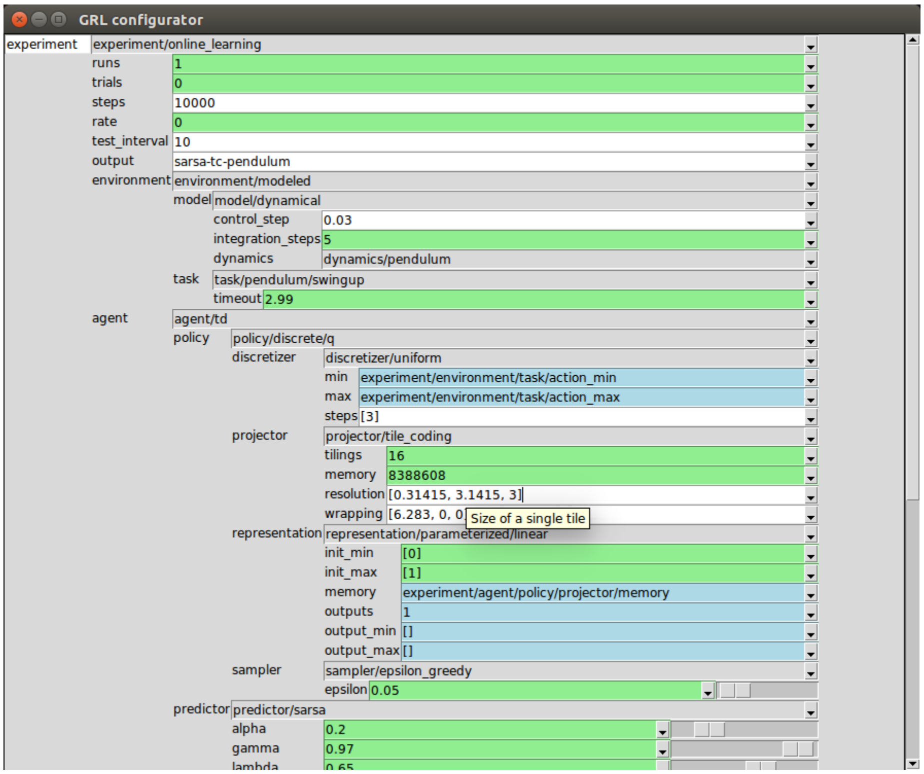

Figure 2: Python configurator user interface

9 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`.

9.1 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, d] = grl_env('step', a);
```

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

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

9.2 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', d, r, o);
```

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

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

A Agents

A.1 agent/black_box

Agent that learns from the cumulative reward of complete rollouts

episodes	int	Number of episodes to evaluate policy
optimizer	optimizer	Policy optimizer

A.2 agent/dyna

Agent that learns from both observed and predicted state transitions

planning_steps	int	Number of planning steps per control step
planning_horizon	int	Planning episode length
asynchronous	int	Asynchronous planning (actual planning_steps depends on control ste
policy	policy	Control policy
predictor	predictor	Value function predictor
model	observation_model	Observation model used for planning
model_predictor	predictor/model	Model predictor
model_agent	agent	Agent used for planning episodes

Provided parameters

state	state	Current observed state of planning

A.3 agent/fixed

Fixed-policy agent

policy	policy	Control policy

A.4 agent/master/exclusive

Master agent that selects one sub-agent to execute

gamma	double	Discount rate
agent1	agent/sub	First subagent
agent2	agent/sub	Second subagent

A.5 agent/master/sequential

Master agent that executes sub-agents sequentially

agent1	agent	First subagent, providing the suggested action
agent2	agent	Second subagent, providing the final action

A.6 agent/solver

Agent that successively solves learned models of the environment

interval	int	Episodes between successive solutions (0=asynchronous)
policy	policy	Control policy
predictor	predictor	Optional (model) predictor
solver	solver	Model-based solver

A.7 agent/sub/compartmentalized

Sub agent that is valid in a fixed state-space region

min	vector.observation_min	Minimum of compartment bounding box
max	vector.observation_max	Maximum of compartment bounding box
agent	agent	Sub agent

A.8 agent/td

Agent that learns from observed state transitions

policy	policy	Control policy
predictor	predictor	Value function predictor

B Discretizers

B.1 discretizer/peaked

Peaked discretizer, with more resolution around center

min	vector	Lower limit
max	vector	Upper limit
steps	vector	Discretization steps per dimension
peaking	vector	Extra resolution factor around center (offset by 1/factor at edges)

B.2 discretizer/uniform

Uniform discretizer

min	vector	Lower limit
max	vector	Upper limit
steps	vector	Discretization steps per dimension

C Dynamics

C.1 dynamics/acrobot

Acrobot dynamics

C.2 dynamics/cart_pole

Cart-pole dynamics from Barto et al.

C.3 dynamics/pendulum

Pendulum dynamics based on the DCSC MOPS

C.4 dynamics/rbdl

RBDL rigid body dynamics

file	string	RBDL Lua model file

C.5 dynamics/tlm

Two-link manipulator dynamics

D Environments

D.1 environment/leo2

LEO/2 environment

port	string	Device ID of FTDI usb-to-serial converter
bps	int	Bit rate

Provided parameters

state	state	Current state of the robot

D.2 environment/modeled

Environment that uses a state transition model internally

model	model	Environment model
task	task	Task to perform in the environment (should match model)
exporter	exporter	Optional exporter for transition log (supports time, state, observation, action, reward, t

Provided parameters

state	state	Current state of the model

D.3 environment/ode

Open Dynamics Engine simulation environment

xml	string	XML configuration filename

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

E Experiments

E.1 experiment/approx_test

Approximator test experiment (supervised learning)

train_samples	int	Number of training samples
test_samples	int	Number of test samples
file	string	Output file (csv format)
input_min	vector	Lower limit for drawing samples
input_max	vector	Upper limit for drawing samples
projector	projector	Projector (should match representation)
representation	representation	Learned representation
mapping	mapping	Function to learn

E.2 experiment/batch_learning

Batch learning experiment using randomly sampled experience

runs	int	Number of separate learning runs to perform
batches	int	Number of batches per learning run
batch_size	int	Number of transitions per batch
rate	int	Test trial control step frequency in Hz
output	string	Output base filename
model	model	Model in which the task is set
task	task	Task to be solved
predictor	predictor	Learner
test_agent	agent	Agent to use in test trials after each batch
observation_min	vector.observation_min	Lower limit for observations
observation_max	vector.observation_max	Upper limit for observations
action_min	vector.action_min	Lower limit for actions
action_max	vector.action_max	Upper limit for actions

Provided parameters

state	state	Current observed state of the environment

E.3 experiment/online_learning

Interactive learning experiment

runs	int	Number of separate learning runs to perform
trials	int	Number of episodes per learning run
steps	int	Number of steps per learning run
rate	int	Control step frequency in Hz
test_interval	int	Number of episodes in between test trials
output	string	Output base filename
environment	environment	Environment in which the agent acts
agent	agent	Agent
test_agent	agent	Agent to use in test trials

Provided parameters

state	state	Current observed state of the environment
curve	state	Learning curve

F Exporters

F.1 exporter/csv

Comma-separated values exporter

file	string	Output base filename
fields	string	Comma-separated list of fields to write
style	string	Header style

G Importers

G.1 importer/csv

Comma-separated values importer

file	string	Input base filename

H Mappings

H.1 mapping/multisine

Sum of sines mapping

inputs	int	Number of input dimensions
outputs	int	Number of output dimensions
sines	int	Number of sines

I Models

I.1 model/compass_walker

Simplest walker model from Garcia et al.

control_step	double.control_step	Control step time
integration_steps	int	Number of integration steps per control step

I.2 model/dynamical

State transition model that integrates equations of motion

control_step	double.control_step	Control step time
integration_steps	int	Number of integration steps per control step
dynamics	dynamics	Equations of motion

I.3 model/pinball

Model of a ball on a plate

control_step	double.control_step	Control step time
integration_steps	int	Number of integration steps per control step
restitution	double	Coefficient of restitution
radius	double	Ball radius

I.4 model/windy

Sutton & Barto's windy gridworld model

J Observation_models

J.1 observation_model/approximated

Observation model based on observed transitions

jacobian_step	double	Step size for Jacobian estimation
control_step	double.control_step	Control step time (0 = estimate using SMDP approximator)
differential	int.differential	Predict state deltas
wrapping	vector.wrapping	Wrapping boundaries
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
stddev_limit	double	Maximum standard deviation of acceptable predictions, as fract
projector	projector.pair	Projector for transition model (—S—+—A— dimensions)
representation	representation.transition	Representation for transition model (—S—+2 dimensions)

J.2 observation_model/fixed

Observation model based on known state transition model

jacobian_step	double	Step size for Jacobian estimation
model	model	Environment model
task	task	Task to perform in the environment (should match model)

J.3 observation_model/fixed_reward

Observation model based on observed transitions but known task

jacobian_step	double	Step size for Jacobian estimation
control_step	double.control_step	Control step time (0 = estimate using SMDP approximator)
differential	int.differential	Predict state deltas
wrapping	vector.wrapping	Wrapping boundaries
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
stddev_limit	double	Maximum standard deviation of acceptable predictions, as fract
projector	projector.pair	Projector for transition model (—S—+—A— dimensions)
representation	representation.transition	Representation for transition model (—S—+2 dimensions)
task	task	Task to perform in the environment

K Optimizers

K.1 optimizer/cma

Coverance matrix adaptation black-box optimizer

population	int	Population size
sigma	vector	Initial standard deviation (a single-element vector will be replicated for
policy	policy/parameterized	Control policy prototype

L Policies

L.1 policy/action

Policy based on a direct action representation

sigma	vector	Standard deviation of exploration distribution
output_min	vector.action_min	Lower limit on outputs
output_max	vector.action_max	Upper limit on outputs
projector	projector.observation	Projects observations onto representation space
representation	representation.action	Action representation

L.2 policy/action_probability

Policy based on an action-probability representation

discretizer	discretizer	Action discretizer
projector	projector	Projects observation-action pairs onto representation space
representation	representation	Action-probability representation

L.3 policy/discrete/q

Q-value based policy

discretizer	discretizer.action	Action discretizer
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	Action-value representation
sampler	sampler	Samples actions from action-values

L.4 policy/discrete/q/bounded

Q-value based policy with bounded action deltas

bound	vector	Maximum action delta
discretizer	discretizer.action	Action discretizer
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	Action-value representation
sampler	sampler	Samples actions from action-values

L.5 policy/discrete/q/ucb

UCB1 policy

discretizer	discretizer.action	Action discretizer
projector	projector.pair	Projects observation-action pairs onto representation spac
representation	representation.value/action	Q-value representation
visit_representation	representation.value/action	Visit count representation
c_p	double	UCB1 exploration term

L.6 policy/discrete/random

Policy that chooses discrete random actions

discretizer	discretizer.action	Action discretizer

L.7 policy/discrete/v

State-value based policy

gamma	double	Discount rate
discretizer	discretizer.action	Action discretizer
model	observation_model	Observation model
projector	projector.observation	Projects observations onto representation space
representation	representation.value/state	State-value representation
sampler	sampler	Samples actions from state-values

L.8 policy/mcts

Monte-Carlo Tree Search policy

model	observation_model	Observation model used for planning
discretizer	discretizer.action	Action discretizer
gamma	double	Discount rate
epsilon	double	Exploration rate
horizon	int	Planning horizon
budget	double	Computational budget

L.9 policy/nmpc

Nonlinear model predictive control policy using the MUSCOD library

model_path	string	Path to MUSCOD model library
model_name	string	Name of MUSCOD model library
outputs	int.action_dims	Number of outputs

L.10 policy/parameterized/action

Parameterized policy based on a direct action representation

sigma	vector	Standard deviation of exploration distribution
output_min	vector.action_min	Lower limit on outputs
output_max	vector.action_max	Upper limit on outputs
projector	projector.observation	Projects observations onto representation space
representation	representation/parameterized.action	Action representation

L.11 policy/parameterized/pid

Parameterized policy based on a proportional-integral-derivative controller

setpoint	vector	Setpoint
outputs	int.action_dims	Number of outputs
p	vector	P gains ([out1_in1, ..., out1_inN, ..., outN_in1, ..., outN_inN])
i	vector	I gains
d	vector	D gains (use P gain on velocity instead, if available)
il	vector	Integration limits

L.12 policy/parameterized/state_feedback

Parameterized policy based on a state feedback controller

operating_state	vector	Operating state around which gains are defined
operating_action	vector	Operating action around which gains are defined
gains	vector	Gains ([in1_out1, ..., in1_outN, ..., inN_out1, ..., inN_outN])
output_min	vector.action_min	Lower action limit
output_max	vector.action_max	Upper action limit

L.13 policy/random

Policy that chooses continuous random actions

output_min	vector.action_min	Lower action limit
output_max	vector.action_max	Upper action limit

L.14 policy/uct

Monte-Carlo Tree Search policy using UCB1 action selection

model	observation_model	Observation model used for planning
discretizer	discretizer.action	Action discretizer
gamma	double	Discount rate
epsilon	double	Exploration rate
horizon	int	Planning horizon
budget	double	Computational budget

M Predictors

M.1 predictor/ac/action

Actor-critic predictor for direct action policies

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation,
alpha	double	Critic learning rate
beta	double	Actor learning rate
gamma	double	Discount rate
lambda	double	Trace decay rate
critic_projector	projector.observation	Projects observations onto critic representation space
critic_representation	representation.value/state	Value function representation
critic_trace	trace	Trace of critic projections
actor_projector	projector.observation	Projects observations onto actor representation space
actor_representation	representation.action	Action representation
actor_trace	trace	Trace of actor projections

M.2 predictor/ac/probability

Actor-critic predictor for action-probability policies

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observatio
alpha	double	Critic learning rate
beta	double	Actor learning rate
gamma	double	Discount rate
lambda	double	Trace decay rate
critic_projector	projector.observation	Projects observations onto critic representation space
critic_representation	representation.value/state	Value function representation
critic_trace	trace	Trace of critic projections
actor_projector	projector.pair	Projects observation-action pairs onto actor representatio
actor_representation	representation.value/action	Action-probability representation
actor_trace	trace	Trace of actor projections
discretizer	discretizer.action	Action discretizer

M.3 predictor/advantage

Advantage learning off-policy value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, acti
alpha	double	Learning rate
gamma	double	Discount rate
lambda	double	Trace decay rate
kappa	double	Advantage scaling factor
discretizer	discretizer.action	Action discretizer
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	A-value representation
trace	trace	Trace of projections

M.4 predictor/expected_sarsa

Expected SARSA low-variance on-policy value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, acti
alpha	double	Learning rate
gamma	double	Discount rate
lambda	double	Trace decay rate
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	Q-value representation
policy	policy/discrete/q	Q-value based policy
sampler	sampler	Target distribution
trace	trace	Trace of projections

M.5 predictor/fqi

Fitted Q-iteration predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, a
gamma	double	Discount rate
transitions	int	Maximum number of transitions to store
iterations	int	Number of policy improvement rounds per episode
reset_strategy	string	At which point to reset the representation
macro_batch_size	int	Number of episodes/batches after which prediction is rebuilt
discretizer	discretizer.action	Action discretizer
projector	projector.pair	Projects observations onto critic representation space
representation	representation.value/action	Value function representation

M.6 predictor/full/qi

Deterministic model-based action-value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, acti
gamma	double	Discount rate
model	observation_model	Observation model used for planning
discretizer	discretizer.action	Action discretizer
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	Action-value function representation

M.7 predictor/full/vi

Deterministic model-based state-value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, actio
gamma	double	Discount rate
model	observation_model	Observation model used for planning
discretizer	discretizer.action	Action discretizer
projector	projector.observation	Projects observations onto representation space
representation	representation.value/state	State-value function representation

M.8 predictor/ggq

Greedy-GQ off-policy value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, acti
alpha	double	Learning rate
eta	double	Relative secondary learning rate (actual is alpha*eta)
gamma	double	Discount rate
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	(Q, w) representation
policy	policy/discrete/q	Greedy target policy

M.9 predictor/model

Observation model predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, action
differential	int.differential	Predict state deltas
wrapping	vector.wrapping	Wrapping boundaries
projector	projector.pair	Projector for transition model (—S—+—A— dimensions)
representation	representation.transition	Representation for transition model (—S—+2 dimensions)

M.10 predictor/qv

QV on-policy value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, ac
alpha	double	State-action value learning rate
beta	double	State value learning rate
gamma	double	Discount rate
lambda	double	Trace decay rate
q_projector	projector.pair	Projects observation-action pairs onto representation space
q_representation	representation.value/action	State-action value representation (Q)
v_projector	projector.observation	Projects observations onto representation space
v_representation	representation.value/state	State value representation (V)
trace	trace	Trace of projections

M.11 predictor/sarsa

SARSA on-policy value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, acti
alpha	double	Learning rate
gamma	double	Discount rate
lambda	double	Trace decay rate
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	Q-value representation
trace	trace	Trace of projections

M.12 predictor/td

TD value function predictor

importer	importer	Optional importer for pre-training
exporter	exporter	Optional exporter for transition log (supports observation, actio
alpha	double	Learning rate
gamma	double	Discount rate
lambda	double	Trace decay rate
projector	projector.observation	Projects observations onto representation space
representation	representation.value/state	State value representation
trace	trace	Trace of projections

N Projectors

N.1 projector/fourier

Fourier basis function projector

input_min	vector	Lower input dimension limit (for scaling)
input_max	vector	Upper input dimension limit (for scaling)
order	int	Order of approximation (bases per dimension)
parity	string	Whether to use odd or even bases

Provided parameters

memory	int.memory	Feature vector size

N.2 projector/grid

Standard discretization

input_min	vector	Lower input dimension limit
input_max	vector	Upper input dimension limit
steps	vector	Grid cells per dimension

Provided parameters

memory	int.memory	Grid size

N.3 projector/identity

Simply returns the input vector

N.4 projector/pre/normalizing

Preprocesses projection onto a normalized [0, 1] vector

input_min	vector	Lower input dimension limit (for scaling)
input_max	vector	Upper input dimension limit (for scaling)
projector	projector.	Downstream projector

N.5 projector/pre/peaked

Preprocesses projection for more resolution around center

peaking	vector	Extra resolution factor around center (offset by 1/factor at edges)
input_min	vector	Lower input dimension limit (for scaling)
input_max	vector	Upper input dimension limit (for scaling)
projector	projector.	Downstream projector

N.6 projector/pre/scaling

Preprocesses projection onto a scaled vector

scaling	vector	Scaling vector
projector	projector.	Downstream projector

N.7 projector/sample/ann

Projects onto samples found through approximate nearest-neighbor search

samples	int	Maximum number of samples to store
neighbors	int	Number of neighbors to return
locality	double	Locality of weighing function
bucket_size	int	?
error_bound	double	?
inputs	int	Number of input dimensions

N.8 projector/sample/ertree

Projects onto samples found through the Extra-trees algorithm by Geurts et al.

samples	int	Maximum number of samples to store
trees	int	Number of trees in the forest
splits	int	Number of candidate splits
leaf_size	int	Maximum number of samples in a leaf
inputs	int	Number of input dimensions
outputs	int	Number of output dimensions

N.9 projector/tile_coding

Hashed tile coding projector

tilings	int	Number of tilings
memory	int.memory	Hash table size
resolution	vector	Size of a single tile
wrapping	vector.wrapping	Wrapping boundaries (must be multiple of resolution)

O Representations

O.1 representation/llr

Performs locally linear regression through samples

ridge	double	Ridge regression (Tikhonov) factor
order	int	Order of regression model
input_nominals	vector	Vector indicating which input dimensions are nominal
output_nominals	vector	Vector indicating which output dimensions are nominal
outputs	int	Number of output dimensions
output_min	vector	Lower output limit
output_max	vector	Upper output limit
projector	projector/sample	Projector used to generate input for this representation

O.2 representation/parameterized/ann

Parameterized artificial neural network representation

inputs	int	Number of input dimensions
output_min	vector	Lower limit on outputs
output_max	vector	Upper limit on outputs
hiddens	int	Number of hidden nodes
steepness	double	Steepness of activation function
bias	int	Use bias nodes
recurrent	int	Feed hidden activation back as input

O.3 representation/parameterized/linear

Linear-in-parameters representation

init_min	vector	Lower initial value limit
init_max	vector	Upper initial value limit
memory	int.memory	Feature vector size
outputs	int	Number of outputs
output_min	vector	Lower output limit
output_max	vector	Upper output limit

P Samplers

P.1 sampler/epsilon_greedy

Maximum search with a uniform random chance of non-maximums

epsilon	double	Exploration rate

P.2 sampler/greedy

Maximum search

P.3 sampler/softmax

Softmax (Gibbs/Boltzmann) sampler

tau	double	Temperature of Boltzmann distribution

Q Solvers

Q.1 solver/agent

Solver that uses a simulated agent

steps	int	Number of planning steps before solution is returned
horizon	int	Planning episode length
start	vector	Starting state for planning
model	observation_model	Observation model used for planning
agent	agent	Agent used for planning episodes

Provided parameters

state	state	Current observed state of planning

Q.2 solver/lqr

Linear Quadratic Regulator solver

operating_state	vector	Operating state around which to linearize
operating_action	vector	Operating action around which to linearize
q	vector	Q (state cost) matrix diagonal
r	vector	R (action cost) matrix diagonal
model	observation_model	Observation model
policy	policy/parameterized/state_feedback	State feedback policy to adjust

Q.3 solver/vi

Value iteration solver

sweeps	int	Number of planning sweeps before solution is returned
parallel	int	Perform backups in parallel (requires reentrant representation)
discretizer	discretizer.observation	State space discretizer
predictor	predictor/full	Predictor to iterate

R Tasks

R.1 task/acrobot/balancing

Acrobot balancing task

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.2 task/cart_pole/balancing

Cart-pole balancing task

timeout	double	Episode timeout

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.3 task/cart_pole/swingup

Cart-pole swing-up task

timeout	double	Episode timeout
randomization	int	Start state randomization
shaping	int	Whether to use reward shaping
gamma	double	Discount rate for reward shaping

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.4 task/compass_walker/walk

Compass walker walking task

timeout	double	Episode timeout

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.5 task/lua

User-provided task specification in LUA

file	string	Lua task file
options	string	Option string to pass to task configuration function

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.6 task/pendulum/swingup

Pendulum swing-up task

timeout	double	Episode timeout
randomization	double	Level of start state randomization

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.7 task/pinball/movement

Pinball movement task

tolerance	double	Goal tolerance

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.8 task/tlm/balancing

Two-link manipulator balancing task

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

R.9 task/windy/movement

Windy gridworld movement task

Provided parameters

observation_dims	int.observation_dims	Number of observation dimensions
observation_min	vector.observation_min	Lower limit on observations
observation_max	vector.observation_max	Upper limit on observations
action_dims	int.action_dims	Number of action dimensions
action_min	vector.action_min	Lower limit on actions
action_max	vector.action_max	Upper limit on actions
reward_min	double.reward_min	Lower limit on immediate reward
reward_max	double.reward_max	Upper limit on immediate reward

S Traces

S.1 trace/enumerated/accumulating

Accumulating eligibility trace using a queue of projections

S.2 trace/enumerated/replacing

Replacing eligibility trace using a queue of projections

T Visualizations

T.1 visualization/acrobot

Acrobot visualization

state	state	Acrobot state to visualize

T.2 visualization/cart_pole

Cart-pole visualization

state	state	Cart-pole state to visualize

T.3 visualization/compass_walker

Compass walker visualization

state	state	Compass walker state to visualize

T.4 visualization/field/policy/action

Visualizes a policy over a field of states

field_dims	vector	Dimensions to visualize
input_min	vector	Lower input dimension limit
input_max	vector	Upper input dimension limit
points	int	Number of points to evaluate
savepoints	int	Number of points to evaluate when saving to file ('s')
projection	string	Method of projecting values onto 2d space
policy	policy	Control policy
output_dim	int	Action dimension to visualize

T.5 visualization/field/policy/value

Visualizes the value of a policy over a field of states

field_dims	vector	Dimensions to visualize
input_min	vector	Lower input dimension limit
input_max	vector	Upper input dimension limit
points	int	Number of points to evaluate
savepoints	int	Number of points to evaluate when saving to file ('s')
projection	string	Method of projecting values onto 2d space
projector	projector.pair	Projects observation-action pairs onto representation space
representation	representation.value/action	Q-value representation
policy	policy/discrete/q	Q-value based control policy

T.6 visualization/field/value

Visualizes an approximation over a field of states

field_dims	vector	Dimensions to visualize
input_min	vector	Lower input dimension limit
input_max	vector	Upper input dimension limit
points	int	Number of points to evaluate
savepoints	int	Number of points to evaluate when saving to file ('s')
projection	string	Method of projecting values onto 2d space
output_dim	int	Output dimension to visualize
projector	projector	Projects inputs onto representation space
representation	representation	Value representation

T.7 visualization/pendulum

Pendulum visualization

state	state	Pendulum state to visualize

T.8 visualization/pinball

Pinball visualization

state	state	Pinball state to visualize

T.9 visualization/sample

Visualizes a sample-based approximation

field_dims	vector	Dimensions to visualize
field_min	vector	Lower visualization dimension limit
field_max	vector	Upper visualization dimension limit
output_dim	int	Output dimension to visualize
points	int	Texture size
projector	projector/sample	Sample projector whose store to visualize

T.10 visualization/sample/random

Visualizes an approximation over randomly sampled states

field_dims	vector	Dimensions to visualize
input_min	vector	Lower input dimension limit
input_max	vector	Upper input dimension limit
output_dim	int	Output dimension to visualize
points	int	Texture size
projector	projector	Projects inputs onto representation space
representation	representation	Value representation

T.11 visualization/state

Plots state values

input_dims	vector	Input dimensions to visualize
input_min	vector	Lower input dimension limit
input_max	vector	Upper input dimension limit
memory	int	Number of data points to draw
state	state	State to visualize

T.12 visualization/tlm

Two-link manipulator visualization

state	state	Two-link manipulator state to visualize

T.13 visualization/windy

Windy gridworld visualization

state	state	Windy gridworld state to visualize

U Visualizers

U.1 visualizer/glut

Visualizer based on the GLUT library