SafeMDP Documentation

Release 1.0

Matteo Turchetta, Felix Berkenkamp, Andreas Krause

November 18, 2016

CHAPTER 1

API Documentation

The *safemdp* package implements tools for safe exploration in finite MDPs.

1.1 Main classes

These classes provide the main functionality for the safe exploration

`SafeMDP`(graph, gp, S_hat0, h, L[, beta])	Base class for safe exploration in MDPs.
`link_graph_and_safe_set`(graph, safe_set)	Link the safe set to the graph model.
`reachable_set`(graph, initial_nodes[, out])	Compute the safe, reachable set of a graph
`returnable_set`(graph, reverse_graph, ...[, out])	Compute the safe, returnable set of a graph

1.1.1 SafeMDP

class `safemdp.`**`SafeMDP`**(*graph*, *gp*, *S_hat0*, *h*, *L*, *beta=2*)

Base class for safe exploration in MDPs.

This class only provides basic options to compute the safely reachable and returnable sets. The actual update of the safety feature must be done in a class that inherits from *SafeMDP*. See *safempd.GridWorld* for an example.

Parameters graph: networkx.DiGraph

The graph that models the MDP. Each edge has an attribute *safe* in its metadata, which determines the safety of the transition.

gp: GPy.core.GPRegression

A Gaussian process model that can be used to determine the safety of transitions. Exact structure depends heavily on the usecase.

S_hat0: boolean array

An array that has True on the ith position if the ith node in the graph is part of the safe set.

h: float

The safety threshold.

L: float

The lipschitz constant

beta: float, optional

The confidence interval used by the GP model.

Methods

add_gp_observations(x_new, y_new)	Add observations to the gp mode.
compute_S_hat()	Compute the safely reachable set given the current safe_set.

add_gp_observations(*x_new*, *y_new*)
Add observations to the gp mode.

compute_S_hat()
Compute the safely reachable set given the current safe_set.

1.1.2 link_graph_and_safe_set

safemdp.**link_graph_and_safe_set**(*graph*, *safe_set*)
Link the safe set to the graph model.

Parameters graph: nx.DiGraph()

safe_set: np.array

Safe set. For each node the edge (i, j) under action (a) is linked to safe_set[i, a]

1.1.3 reachable_set

safemdp.**reachable_set**(*graph*, *initial_nodes*, *out=None*)
Compute the safe, reachable set of a graph

Parameters graph: nx.DiGraph

Directed graph. Each edge must have associated action metadata, which specifies the action that this edge corresponds to. Each edge has an attribute ['safe'], which is a boolean that indicates safety

initial_nodes: list

List of the initial, safe nodes that are used as a starting point to compute the reachable set.

out: np.array

The array to write the results to. Is assumed to be False everywhere except at the initial nodes

Returns reachable_set: np.array

Boolean array that indicates whether a node belongs to the reachable set.

1.1.4 returnable_set

safemdp.**returnable_set**(*graph*, *reverse_graph*, *initial_nodes*, *out=None*)
Compute the safe, returnable set of a graph

Parameters graph: nx.DiGraph

Directed graph. Each edge must have associated action metadata, which specifies the action that this edge corresponds to. Each edge has an attribute ['safe'], which is a boolean that indicates safety

reverse_graph: nx.DiGraph

The reversed directed graph, *graph.reverse()*

initial_nodes: list

List of the initial, safe nodes that are used as a starting point to compute the returnable set.

out: np.array

The array to write the results to. Is assumed to be False everywhere except at the initial nodes

Returns returnable_set: np.array

Boolean array that indicates whether a node belongs to the returnable set.

1.2 Grid world

Some additional functionality specific to gridworlds.

GridWorld(gp, world_shape, step_size, beta, ...)	Grid world with Safe exploration
states_to_nodes(states, world_shape, step_size)	Convert physical states to node numbers.
nodes_to_states(nodes, world_shape, step_size)	Convert node numbers to physical states.
draw_gp_sample(kernel, world_shape, step_size)	Draws a sample from a Gaussian process distribution over a user
grid_world_graph(world_size)	Create a graph that represents a grid world.
grid(world_shape, step_size)	Creates grids of coordinates and indices of state space
compute_true_safe_set(world_shape, altitude, h)	Computes the safe set given a perfect knowledge of the map
compute_true_S_hat(graph, safe_set, ...[, ...])	Compute the true safe set with reachability and returnability.
compute_S_hat0(s, world_shape, n_actions, ...)	Compute a valid initial safe seed.
shortest_path(source, next_sample, G)	Computes shortest safe path from a source to the next state-action pair
path_to_boolean_matrix(path, graph, S)	Computes a S-like matrix for approaches where performances is based on
safe_subpath(path, altitudes, h)	Computes the maximum subpath of path along which the safety constrain

1.2.1 GridWorld

class safemdp.**GridWorld**(*gp*, *world_shape*, *step_size*, *beta*, *altitudes*, *h*, *S0*, *S_hat0*, *L*, *update_dist=0*)

Grid world with Safe exploration

Parameters gp: GPy.core.GP

Gaussian process that expresses our current belief over the safety feature

world_shape: shape

Tuple that contains the shape of the grid world n x m

step_size: tuple of floats

Tuple that contains the step sizes along each direction to create a linearly spaced grid

beta: float

Scaling factor to determine the amplitude of the confidence intervals

altitudes: np.array

It contains the flattened n x m matrix where the altitudes of all the points in the map are stored

h: float

Safety threshold

S0: np.array

n_states x (n_actions + 1) array of booleans that indicates which states (first column) and which state-action pairs belong to the initial safe seed. Notice that, by convention we initialize all the states to be safe

S_hat0: np.array or nan

n_states x (n_actions + 1) array of booleans that indicates which states (first column) and which state-action pairs belong to the initial safe seed and satisfy recovery and reachability properties. If it is nan, such a boolean matrix is computed during initialization

noise: float

Standard deviation of the measurement noise

L: float

Lipschitz constant to compute expanders

update_dist: int

Distance in unweighted graph used for confidence interval update. A sample will only influence other nodes within this distance.

Methods

add_gp_observations(x_new, y_new)	Add observations to the gp mode.
add_observation(node, action)	Add an observation of the given state-action pair.
compute_S_hat()	Compute the safely reachable set given the current safe_set.
compute_expanders()	Compute the expanders based on the current estimate of S_hat.
plot_S(safe_set[, action])	Plot the set of safe states
target_sample()	Compute the next target (s, a) to sample (highest uncertainty within
update_confidence_interval([jacobian])	Updates the lower and the upper bound of the confidence intervals
update_sets()	Update the sets S, S_hat and G taking with the available observation

add_gp_observations(*x_new*, *y_new*)

Add observations to the gp mode.

add_observation(*node*, *action*)

Add an observation of the given state-action pair.

Observing the pair (s, a) means adding an observation of the altitude at s and an observation of the altitude at f(s, a)

Parameters **node: int**

Node index

action: int

Action index

compute_S_hat()
Compute the safely reachable set given the current safe_set.

compute_expanders()
Compute the expanders based on the current estimate of S_hat.

plot_S(*safe_set*, *action=0*)
Plot the set of safe states

Parameters **safe_set: np.array(dtype=bool)**

n_states x (n_actions + 1) array of boolean values that indicates the safe set

action: int

The action for which we want to plot the safe set.

target_sample()
Compute the next target (s, a) to sample (highest uncertainty within G or S_hat)

Returns node: int

The next node to sample

action: int

The next action to sample

update_confidence_interval(*jacobian=False*)
Updates the lower and the upper bound of the confidence intervals using then posterior distribution over the gradients of the altitudes

Returns l: np.array

lower bound of the safety feature (mean - beta*std)

u: np.array

upper bound of the safety feature (mean - beta*std)

update_sets()
Update the sets S, S_hat and G taking with the available observation

1.2.2 states_to_nodes

safemdp.**states_to_nodes**(*states*, *world_shape*, *step_size*)
Convert physical states to node numbers.

Parameters **states: np.array**

States with physical coordinates

world_shape: tuple

The size of the grid_world

step_size: tuple

The step size of the grid world

Returns nodes: np.array

The node indices corresponding to the states

1.2.3 nodes_to_states

`safemdp.nodes_to_states(nodes, world_shape, step_size)`

Convert node numbers to physical states.

Parameters **nodes: np.array**

Node indices of the grid world

world_shape: tuple

The size of the grid_world

step_size: np.array

The step size of the grid world

Returns states: np.array

The states in physical coordinates

1.2.4 draw_gp_sample

`safemdp.draw_gp_sample(kernel, world_shape, step_size)`

Draws a sample from a Gaussian process distribution over a user specified grid

Parameters **kernel: GPy kernel**

Defines the GP we draw a sample from

world_shape: tuple

Shape of the grid we use for sampling

step_size: tuple

Step size along any axis to find linearly spaced points

1.2.5 grid_world_graph

`safemdp.grid_world_graph(world_size)`

Create a graph that represents a grid world.

In the grid world there are four actions, (1, 2, 3, 4), which correspond to going (up, right, down, left) in the x-y plane. The states are ordered so that *np.arange(np.prod(world_size)).reshape(world_size)* corresponds to a matrix where increasing the row index corresponds to the x direction in the graph, and increasing y index corresponds to the y direction.

Parameters **world_size: tuple**

The size of the grid world (rows, columns)

Returns graph: nx.DiGraph()

The directed graph representing the grid world.

1.2.6 grid

`safemdp.grid(world_shape, step_size)`

Creates grids of coordinates and indices of state space

Parameters **world_shape: tuple**

Size of the grid world (rows, columns)

step_size: tuple

Phyiscal step size in the grid world

Returns states_ind: np.array

(n*m) x 2 array containing the indices of the states

states_coord: np.array

(n*m) x 2 array containing the coordinates of the states

1.2.7 compute_true_safe_set

safemdp.**compute_true_safe_set**(*world_shape*, *altitude*, *h*)

Computes the safe set given a perfect knowledge of the map

Parameters **world_shape: tuple**

altitude: np.array

1-d vector with altitudes for each node

h: float

Safety threshold for height differences

Returns true_safe: np.array

Boolean array n_states x (n_actions + 1).

1.2.8 compute_true_S_hat

safemdp.**compute_true_S_hat**(*graph*, *safe_set*, *initial_nodes*, *reverse_graph=None*)

Compute the true safe set with reachability and returnability.

Parameters **graph: nx.DiGraph**

safe_set: np.array

initial_nodes: list of int

reverse_graph: nx.DiGraph

graph.reverse()

Returns true_safe: np.array

Boolean array n_states x (n_actions + 1).

1.2.9 compute_S_hat0

safemdp.**compute_S_hat0**(*s*, *world_shape*, *n_actions*, *altitudes*, *step_size*, *h*)

Compute a valid initial safe seed.

s: int or nan Vector index of the state where we start computing the safe seed from. If it is equal to nan, a state is chosen at random

world_shape: tuple Size of the grid world (rows, columns)

n_actions: int Number of actions available to the agent

altitudes: np.array It contains the flattened n x m matrix where the altitudes of all the points in the map are stored

step_size: tuple step sizes along each direction to create a linearly spaced grid

h: float Safety threshold

S_hat: np.array Boolean array n_states x (n_actions + 1).

1.2.10 shortest_path

`safemdp.`**`shortest_path`**(*source*, *next_sample*, *G*)

Computes shortest safe path from a source to the next state-action pair the agent needs to sample

Parameters **source: int**

Staring node for the path

next_sample: (int, int)

Next state-action pair the agent needs to sample. First entry is the number that indicates the state. Second entry indicates the action

G: networkx DiGraph

Graph that indicates the dynamics. It is linked to S matrix

Returns path: list

shortest safe path

1.2.11 path_to_boolean_matrix

`safemdp.`**`path_to_boolean_matrix`**(*path*, *graph*, *S*)

Computes a S-like matrix for approaches where performances is based on the trajectory of the agent (e.g. unsafe or random exploration) Parameters ——— path: np.array

Contains the nodes that are visited along the path

graph: networkx.DiGraph Graph that indicates the dynamics

S: np.array Array describing the safe set (needed for initialization)

Returns bool_mat: np.array

S-like array that is true for all the states and state-action pairs along the path

1.2.12 safe_subpath

`safemdp.`**`safe_subpath`**(*path*, *altitudes*, *h*)

Computes the maximum subpath of path along which the safety constraint is not violated Parameters ——— path: np.array

Contains the nodes that are visited along the path

altitudes: np.array 1-d vector with altitudes for each node

h: float Safety threshold

Returns subpath: np.array

Maximum subpath of path that fulfills the safety constraint

1.3 Utilities

The following are utilities to make testing and working with the library more pleasant.

`DifferenceKernel`(kernel)	A fake kernel that can be used to predict differences two function values.
`max_out_degree`(graph)	Compute the maximum out_degree of a graph

1.3.1 DifferenceKernel

class `safemdp.`**`DifferenceKernel`**(*kernel*)

A fake kernel that can be used to predict differences two function values.

Given a gp based on measurements, we aim to predict the difference between the function values at two different test points, X1 and X2; that is, we want to obtain mean and variance of f(X1) - f(X2). Using this fake kernel, this can be achieved with *mean, var = gp.predict(np.hstack((X1, X2)), kern=DiffKernel(gp.kern))*

Parameters **kernel: GPy.kern.***

The kernel used by the GP

Methods

`K`(x1[, x2])	Equivalent of kern.K
`Kdiag`(x)	Equivalent of kern.Kdiag for the difference prediction.

`K`(*x1*, *x2=None*)

Equivalent of kern.K

If only x1 is passed then it is assumed to contain the data for both whose differences we are computing. Otherwise, x2 will contain these extended states (see PosteriorExact._raw_predict in GPy/inference/latent_function_inference0/posterior.py)

Parameters **x1: np.array**

x2: np.array

`Kdiag`(*x*)

Equivalent of kern.Kdiag for the difference prediction.

Parameters **x: np.array**

1.3.2 max_out_degree

`safemdp.`**`max_out_degree`**(*graph*)

Compute the maximum out_degree of a graph

Parameters **graph: nx.DiGraph**

Returns max_out_degree: int

The maximum out_degree of the graph

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index