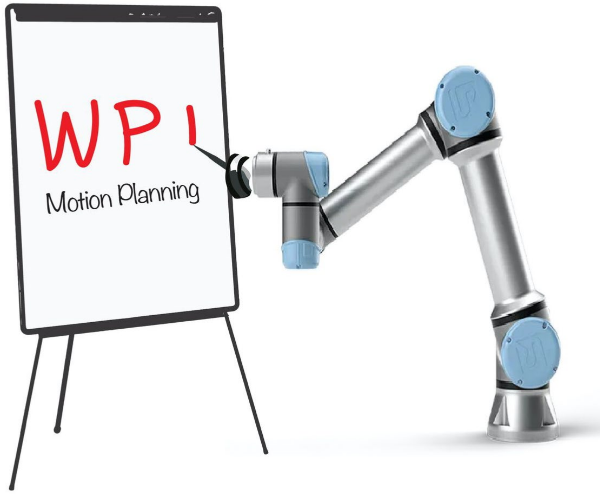
RBE550
Motion Planning
OMPL Overview



Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

Disclaimer and Acknowledgments

The slides are a compilation of work based on notes and slides mainly from Zak Kingston, , but also Ali Golestaneh

Overview

- OMPL at High Level
- OMPL Internals

OMPL at High Level

What is OMPL?

OMPL is a C++11 Library (with automatically generated Python Bindings), intended for industrial, educational, and academic use, which has:

- Core data-structures for sampling-based planners
- Commonly used heuristics, components, etc.
- Implementation of many state-of-the-art sampling-based planners
  - With details of many low-level details skipped in their corresponding papers!

What is OMPL *not*?

OMPL does not:

- Represent robot and scene geometry
- Compute forward and inverse kinematics
- Provide collision checking routines
- Simulate robot dynamics
- etc.

OMPL is an abstract library that you must provide the guts for!

What is OMPL.app?

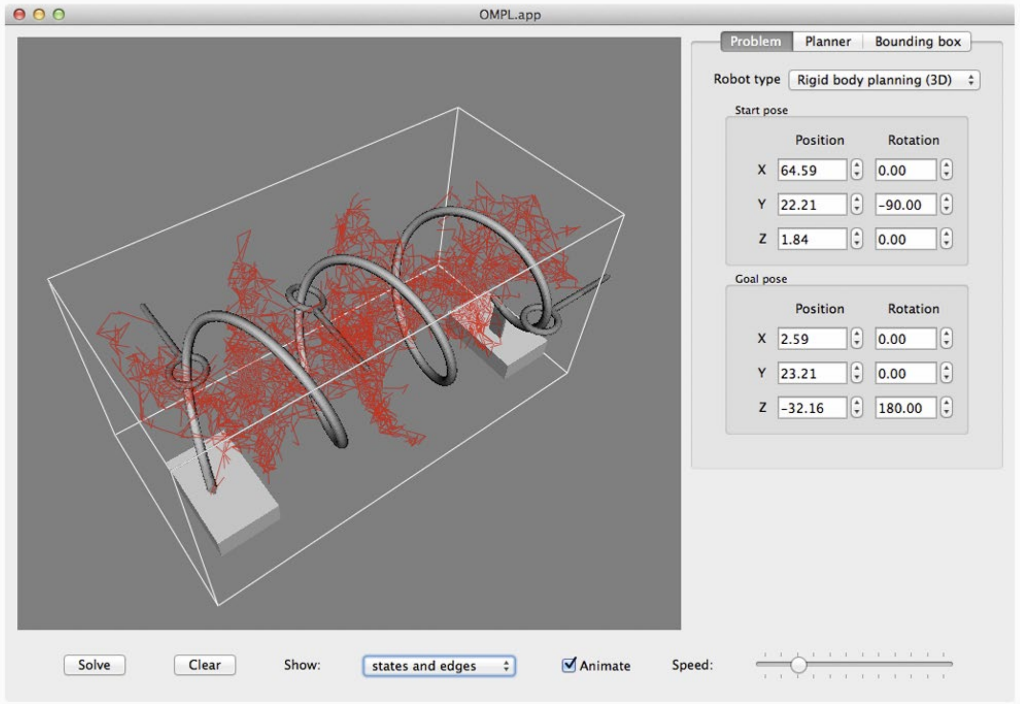


Figure 1: Spirelli Problem

OMPL.app is a lightweight GUI front-end to OMPL that uses a triangle mesh representation of free-flying robots in an environment

OMPL Internals

Sampling-Based Planners

$\textsc{GraphPlanner}(q_{\text{start}}, q_{\text{goal}})$

$\mathcal{G}.\mathtt{init}(q_{\text{start}}, q_{\text{goal}});$

while no path from q_{start} to q_{goal} **do**

$q_{\text{rand}} \leftarrow \mathtt{Sample}();$

$Q \leftarrow \mathtt{SelectNghbrs}(\mathcal{G}, q_{\text{rand}})$

for all $q_{\text{near}} \in Q$ **do**

if $\mathtt{Connect}(q_{\text{near}}, q_{\text{rand}})$ **then**

$\mathcal{G}.\mathtt{Add}(q_{\text{near}}, q_{\text{rand}})$

Figure 2: A Simple Graph-based Sampling-based Planner

OMPL is an Abstract Interface

OMPL requires you to define all the components of a motion planning problem:

State Space e.g., the configuration space

State Validator e.g., a collision checker

State Sampler e.g., a uniform sampler

Start & Goal e.g., start and goal states

Motion Planner Pick your favorite!

And so on...

These are specific to your robot and environment, your motion planner, and your problem!

OMPL is an Abstract Interface

OMPL requires you to define all the components of a motion planning problem:

State Space e.g., the configuration space

State Validator e.g., a collision checker

State Sampler e.g., a uniform sampler

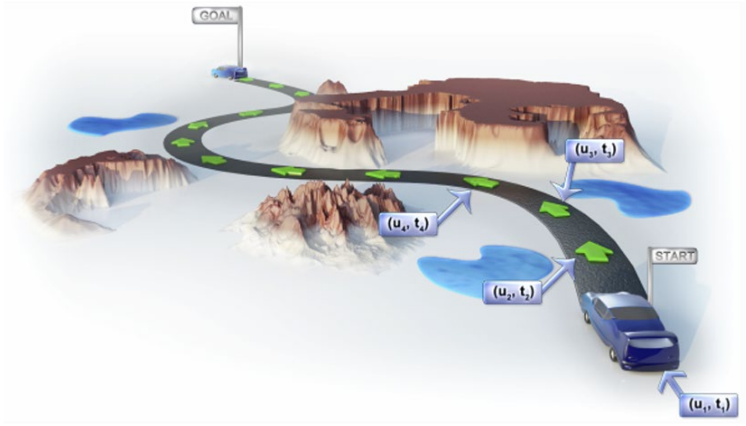
Start & Goal e.g., start and goal states

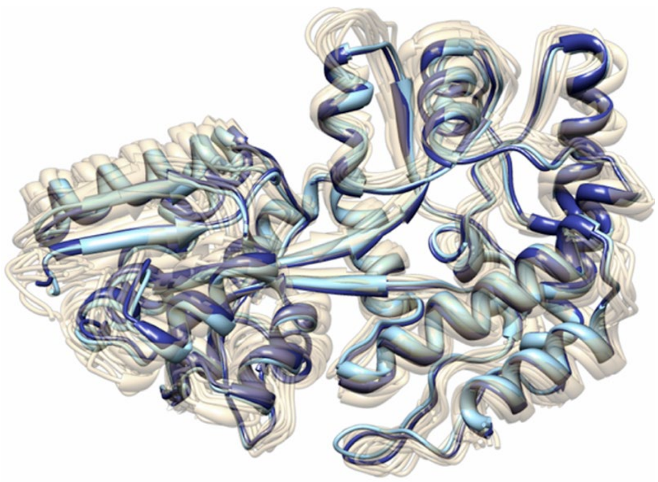
Motion Planner Pick your favorite!

And so on...

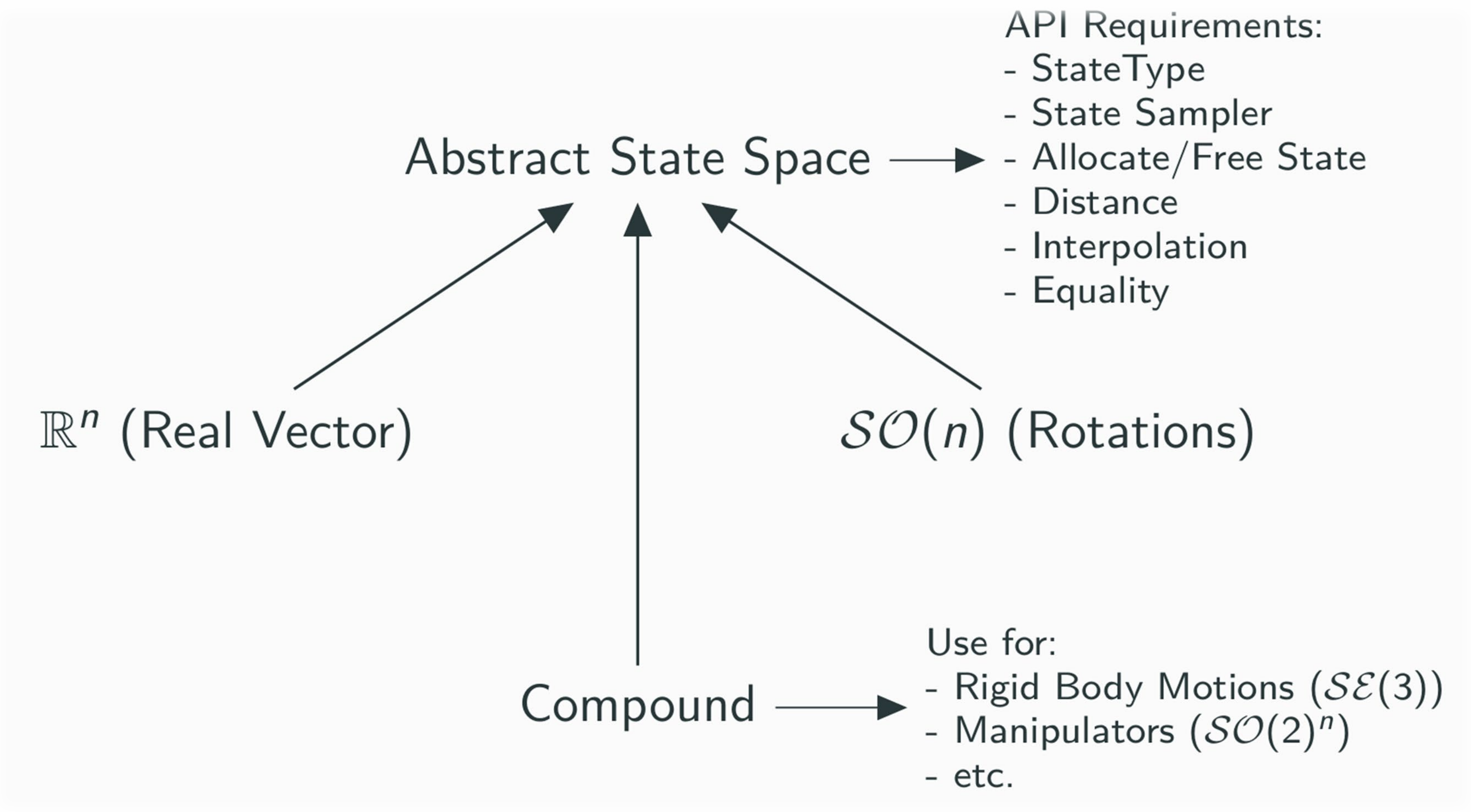
These are specific to your robot and environment, your motion planner, and your problem!

States & State Spaces





States & State Spaces



OMPL is an Abstract Interface

OMPL requires you to define all the components of a motion planning problem:

State Space e.g., the configuration space

State Validator e.g., a collision checker

State Sampler e.g., a uniform sampler

Start & Goal e.g., start and goal states

Motion Planner Pick your favorite!

And so on...

These are specific to your robot and environment, your motion planner, and your problem!

State Validators

The state validator is problem specific, and must be defined by the user / layer on OMPL.

- e.g., OMPL.app, MoveIt!, V-REP, and the list goes on...

For a robot, this could be a collision checker, are joint angles in bounds, etc.

For a molecule, this could be the energy of the state, use the metropolis criterion, etc.

Optionally, state validators can return:

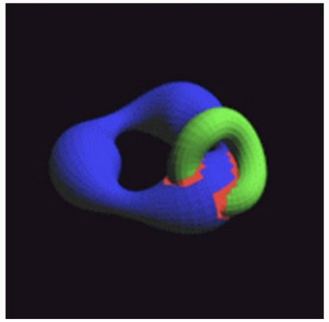
- Distance to nearest invalid state (i.e., obstacle)
- Gradient of distance

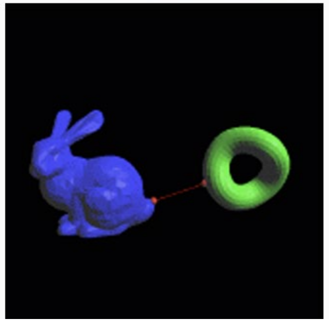
Which planners and other components can use!

Collision Checking

To do collision checking, you must first choose a model of the world (e.g., point clouds, triangle meshes, etc.)

- ROS (robot operating system, common middleware used in robotics) offers a sensor-driven world model with point clouds (represented as octrees) and triangle meshes.
- OMPL.app uses triangle meshes and FCL (flexible collision library) for collision checking.
- Easy to add wrappers to whatever collision checker you want!





Images from pointclouds.org and PQP site

OMPL is an Abstract Interface

OMPL requires you to define all the components of a motion planning problem:

State Space e.g., the configuration space

State Validator e.g., a collision checker

State Sampler e.g., a uniform sampler

Start & Goal e.g., start and goal states

Motion Planner Pick your favorite!

And so on...

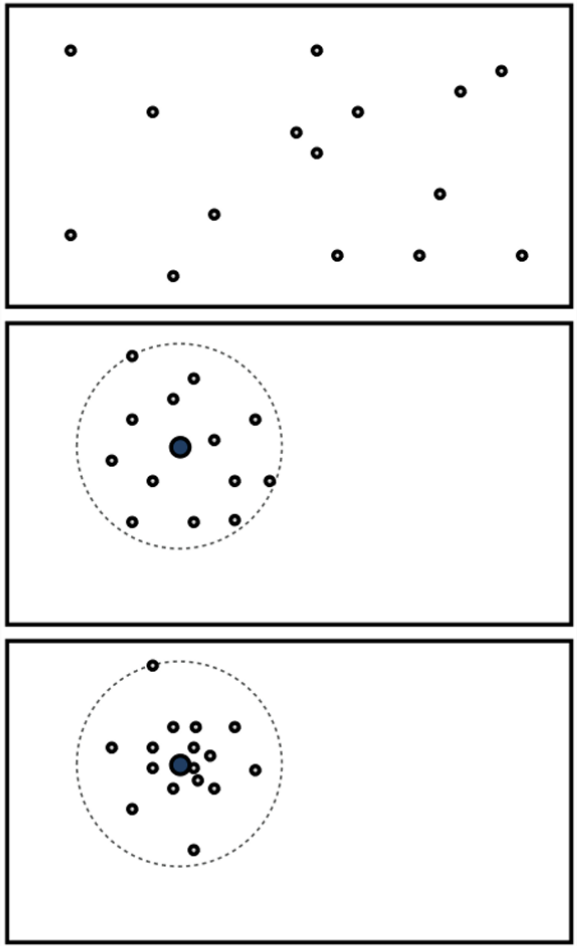
These are specific to your robot and environment, your motion planner, and your problem!

State Samplers

Every State Space needs a State Sampler

- Sample Uniformly
- Sample Nearby
- Sample from a Gaussian

These can be specialized (i.e., biased), e.g., bridge test, experience-based, medial axis, etc.

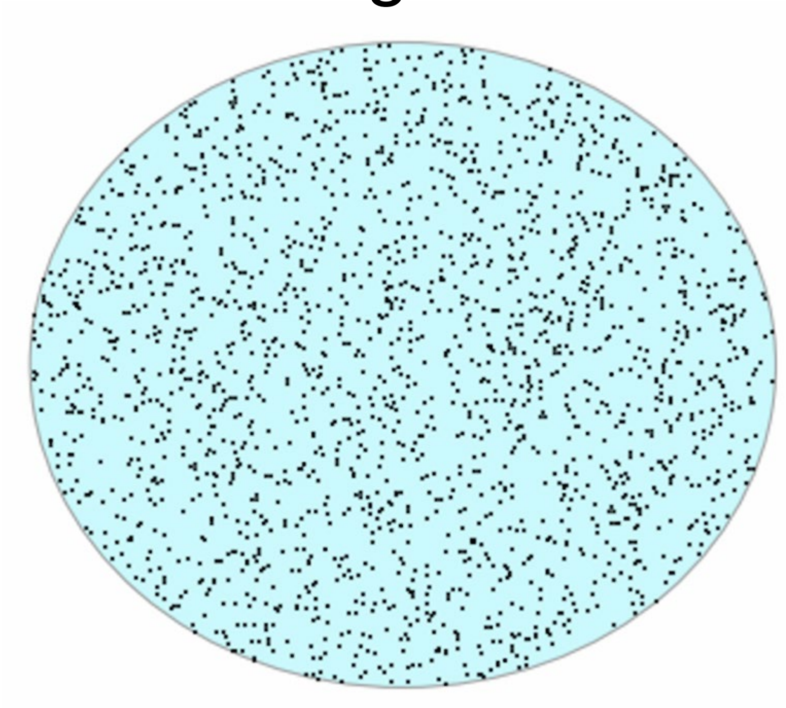


How to Sample Uniformly in Different Spaces?

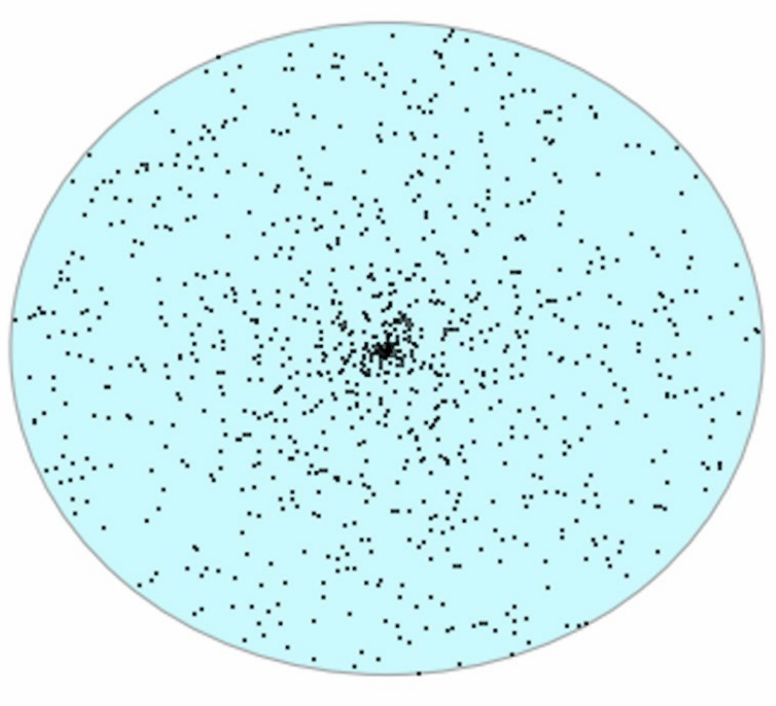
How do you uniformly sample inside the area of a circle, and more generally a ball in $\mathbb{R}^n$?

How to Sample Uniformly in Different Spaces?

Right:



Wrong:



Marsaglia n-Ball Sampling

Algorithm 2.5.4: Generating Uniform Random Vectors inside the n-Ball

output: Random vector $\mathbf{Z}$ uniformly distributed within the n-ball.

1 Generate a random vector $\mathbf{X} = (X_1, \ldots, X_n)$ with iid $\mathsf{N}(0, 1)$ components.

2 Generate $R = U^{1/n}$, with $U \sim \mathsf{U}(0, 1)$.

3 $\mathbf{Z} \leftarrow R\,\mathbf{X}/||\mathbf{X}||$

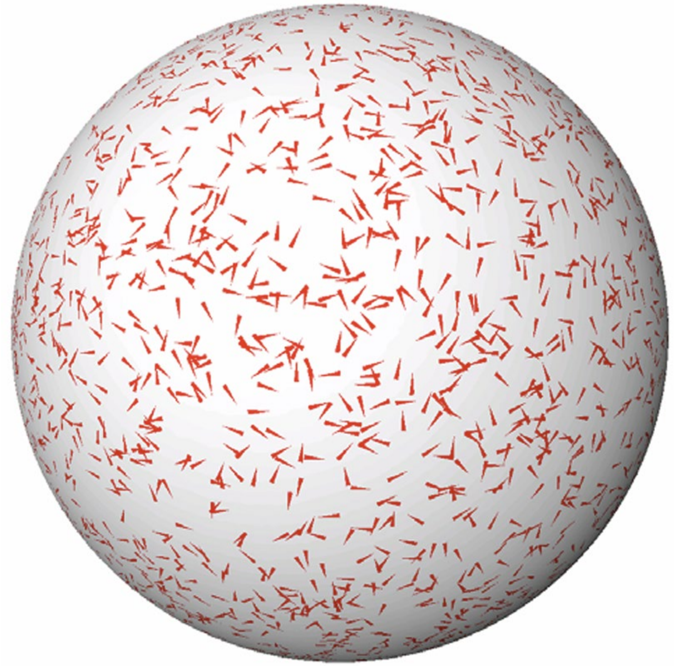
4 **return** $\mathbf{Z}$

Marsaglia, G. "Choosing a Point from the Surface of a Sphere." Ann. Math. Stat. 43, 645-646, 1972.

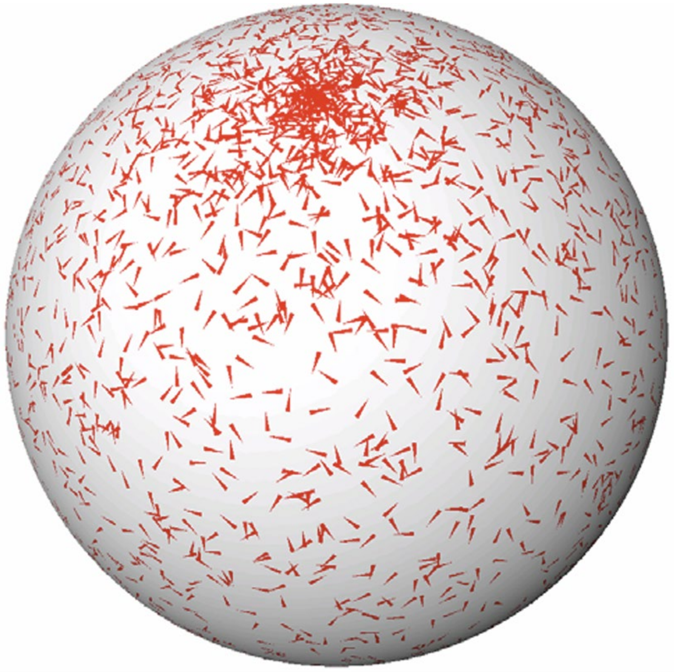
So Many Ways to Get Sampling Wrong

e.g., 3D Rotations:

Right:



Wrong:



Nearest Neighbors

A nearest neighbor datastructure (NN) is essential for most sampling-based planners. NNs allow for efficient lookup of spatial similar points.

k-NN (the k nearest neighbors to a point) can be computed efficiently with kd-trees in low-dimensional, Euclidean (Rn) spaces.

In high-dimensional spaces, approximate NN works much better.

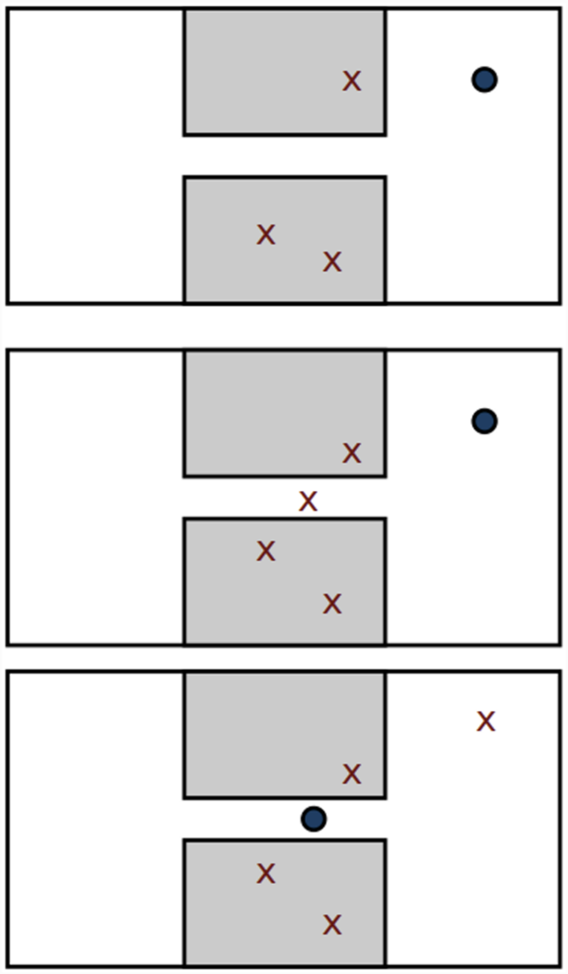
In non-Euclidean spaces (e.g., any space with rotations), data structures other than kd-trees are necessary. OMPL provides a few implementations (GNATs for metric spaces, etc.).

Valid State Samplers

Valid State Samplers combine State Samplers with Validity Checkers.

In the simplest form: sample n times until the state is valid.

Can try other strategies, like finding states with high clearance or in narrow regions.



OMPL is an Abstract Interface

OMPL requires you to define all the components of a motion planning problem:

State Space e.g., the configuration space

State Validator e.g., a collision checker

State Sampler e.g., a uniform sampler

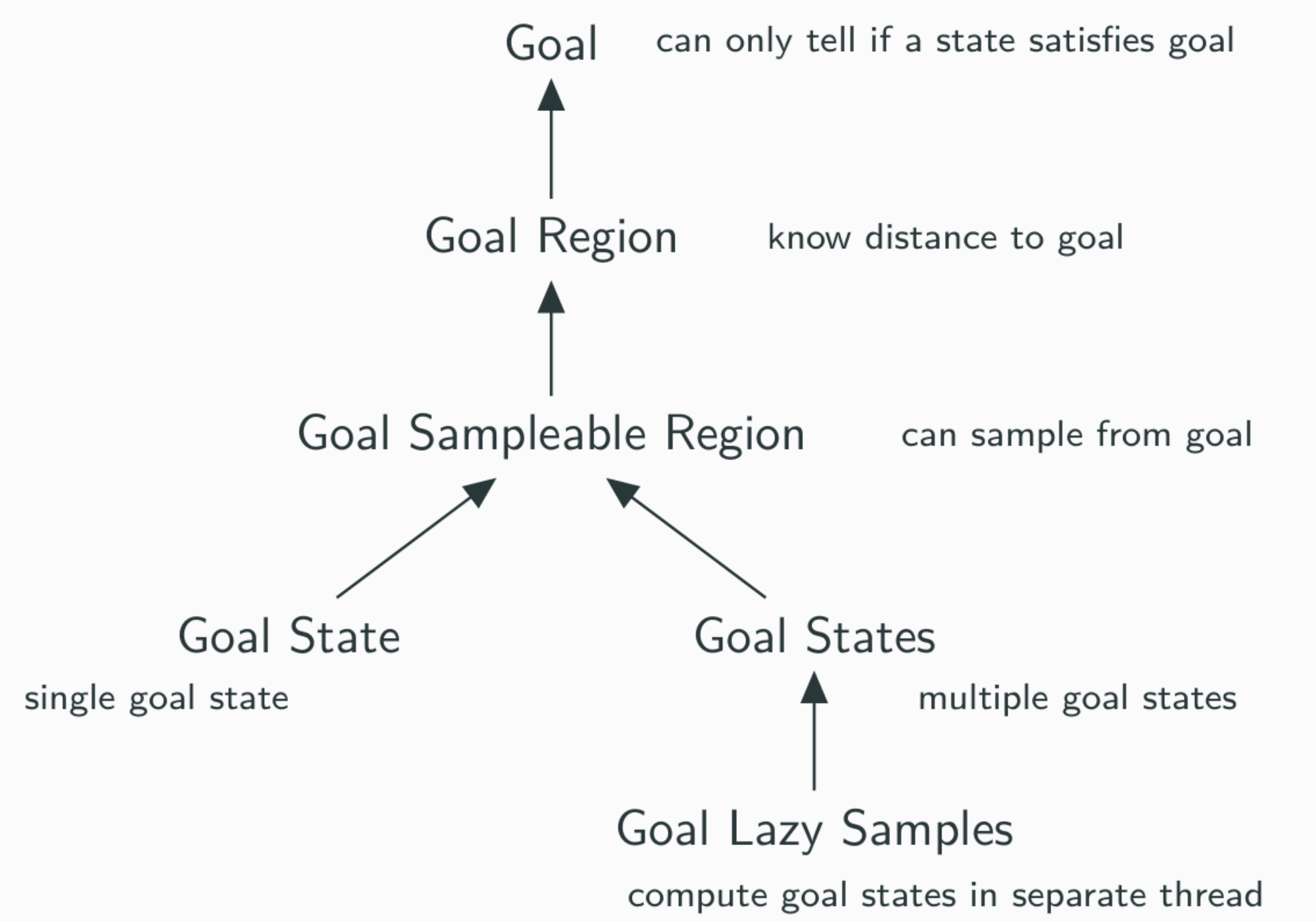
Start & Goal e.g., start and goal states

Motion Planner Pick your favorite!

And so on...

These are specific to your robot and environment, your motion planner, and your problem!

Goals


Goal
can only tell if a state satisfies goal
Goal Region
know distance to goal
Goal Sampleable Region
can sample from goal
Goal State
single goal state
Goal States
multiple goal states
Goal Lazy Samples
compute goal states in separate thread

OMPL is an Abstract Interface

OMPL requires you to define all the components of a motion planning problem:

State Space e.g., the configuration space

State Validator e.g., a collision checker

State Sampler e.g., a uniform sampler

Start & Goal e.g., start and goal states

Motion Planner Pick your favorite!

And so on...

These are specific to your robot and environment, your motion planner, and your problem!

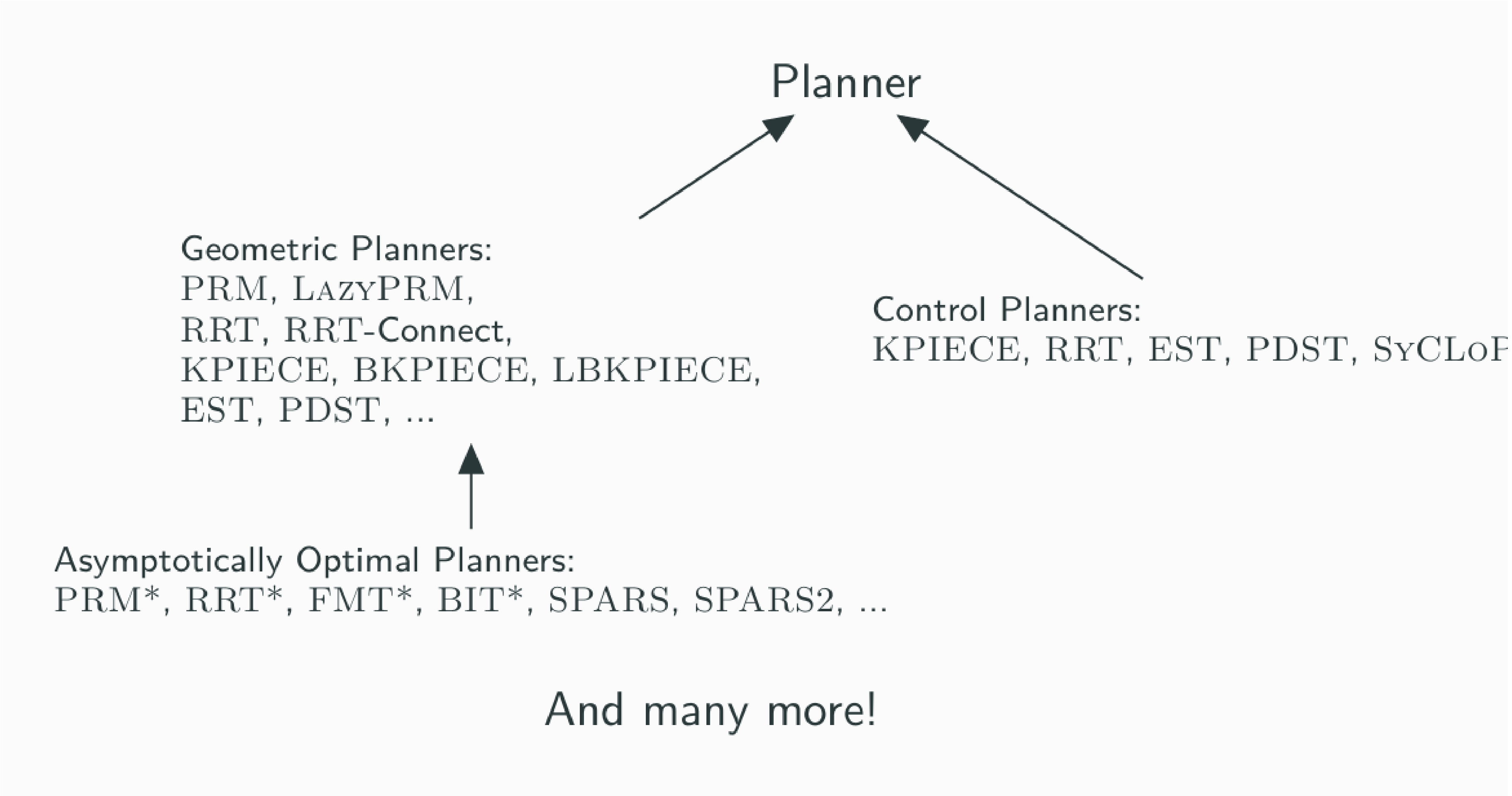
Planners

Planners take in as input Problem Definitions. Problem definitions have at least one start state and a goal object.

Planners only need to implement two methods:

- `solve`: Takes a Planner Termination Condition as an argument. This can be based on time, external events, etc.
- `clear`: Clear internal memory, get ready to solve another problem

Many Planners Available in OMPL



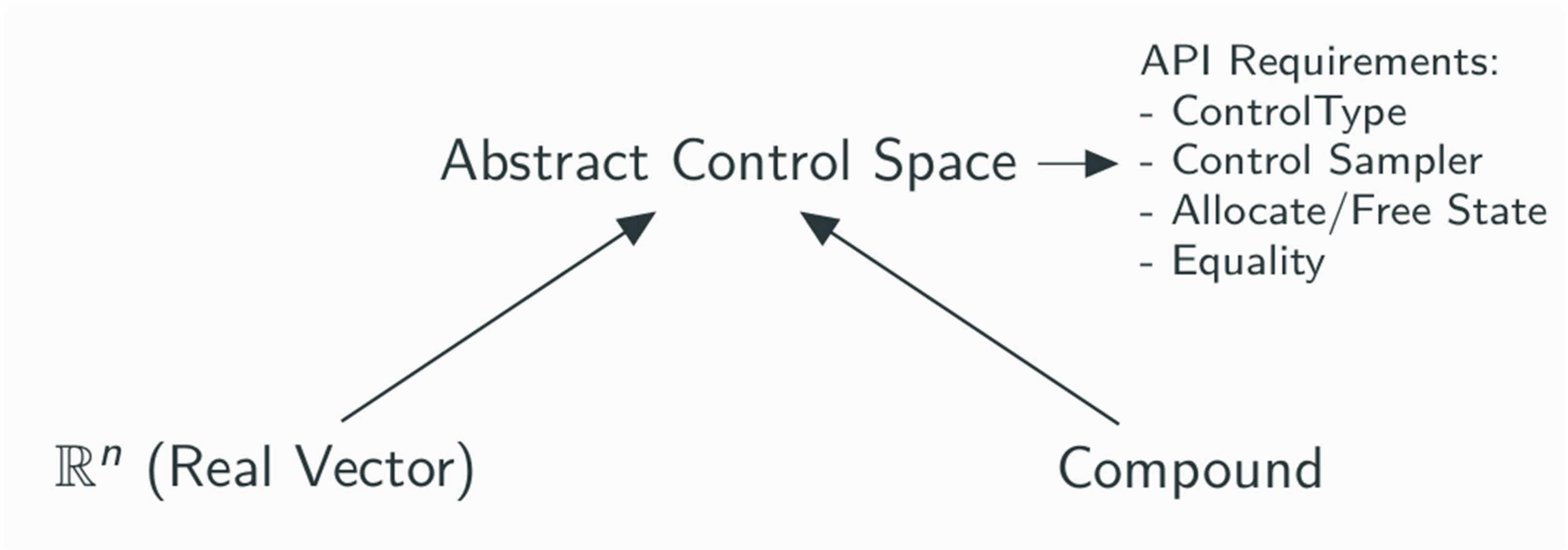
The OMPL Bifurcation

There are two main namespaces in OMPL:

- `geometric`: Planning for rigid bodies with geometric constraints (e.g., manipulators, free-flying bodies, in general holonomic robots). This will be Project 3.

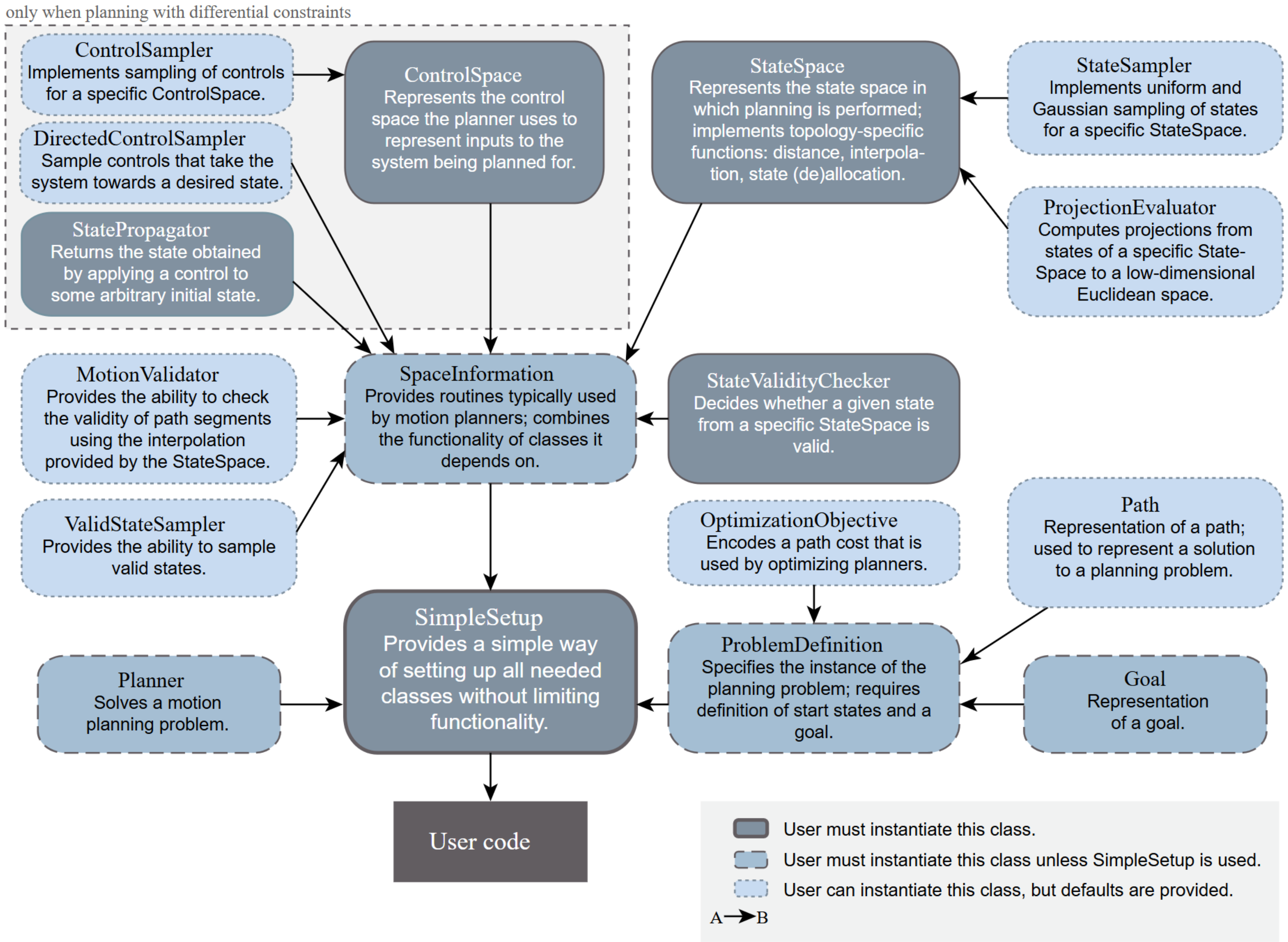
- `control`: Planning for systems with dynamical constraints (e.g., cars, torque controlled manipulators, generally non-holonomic robots). This will be Project 4.

Control Spaces



Summing Up…

API Overview



Planner Arena (`http://plannerarena.org`)

Motion planning problem

Abstract

Benchmark attribute

time

- [x] Show advanced options
  - [ ] Show as cumulative distribution function
  - [ ] Include results after simplification
  - [ ] Show box plots for parametrized benchmarks
  - [x] Hide outliers in box plots
  - [ ] Use log scale for Y-axis

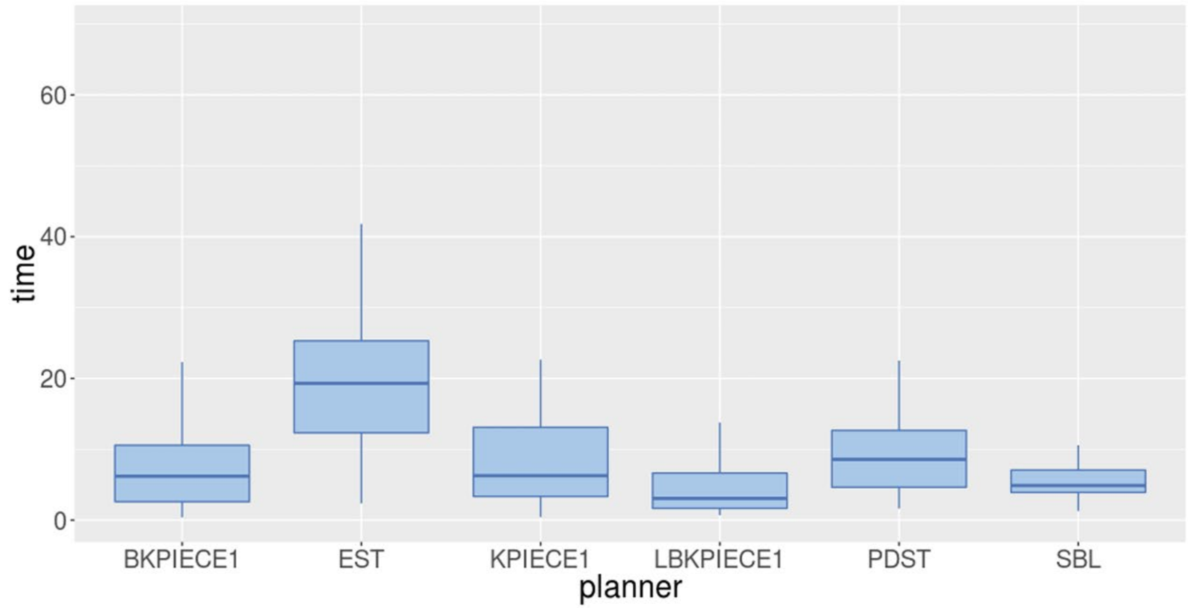
Version

0.15.0

Selected planners

- [x] BKPIECE1
- [x] EST
- [x] KPIECE1
- [x] LBKPIECE1
- [x] PDST
- [ ] PRM
- [ ] RRT
- [ ] RRTConnect
- [x] SBL
- [ ] STRIDE
- [ ] TRRT

Download plot as PDF | Download plot as RData



Number of missing data points out of the total number of runs per planner

planner	missing	total
BKPIECE1	0	50
EST	0	50
KPIECE1	0	50
LBKPIECE1	0	50
PDST	0	50
SBL	0	50

OMPL Online

Website at `http://ompl.kavrakilab.org/`, with:

- Searchable documentation
- Tutorials
- Demos
- and More!

Live Demo!