Technology Transfer and Outreach to the Robotics Community

Lydia Kavraki
Rice University

• How to have maximum impact on the robotics community?

• How to increase the users of the technology and potential applications
Connection to Existing Tools

ROS + MoveIt!

ExCAPE robotics related activities
Robot Operating System (ROS)

• Provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

• Actively developed by the Open Source Robotics Foundation (ORSF) for the DAPRA latest challenge

• Since 2009. Installed on more than 30 robotic platforms

• Who is using ROS: academia and industry
Gazebo

• Gazebo is a 3D robot simulator with dynamics. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics).

• Model Database: an online repository of models, materials, meshes, and plugins provides Gazebo with access to user created content through a simple drag-and-drop interface.

• Very actively developed by OSRF....

• Available: now?
Development of OMPL

ROS
+ MoveIt!

GAZEBO

ExCAPE

Open Motion Planning Library
OMPL
OMPL: Abstract interface to core sampling-based motion planning concepts

- state space / control space
- state validator (e.g., collision checker)
- sampler
- goal (problem definition)
- planner

... except robot & workspace...

Roadmaps:
- PRM [Kavraki, Svestka, Latombe, Overmars '96]
- Obstacle based PRM [Amato, Bayazit, Dale '98]
- Medial Axis PRM [Wilmarth, Amato, Stiller '98]
- Gaussian PRM [Boor, Overmars, van der Stappen '01]
- Bridge Building Planner [Hsu, Jiang, Reif, Sun '03]
- Hierarchical PRM [Collins, Agarwal, Harer '03]
- Improving PRM Roadmaps [Morales, Rodriguez, Amato '03]
- Entropy guided Path-planning [Burns, Brendan, Brock '04]
- RESAMPL [Rodriguez, Thomas, Pearce, Amato '06]
- Probab. foundations of PRM [Hsu, Latombe, Kurniawati '06]
- Adaptive PRM [Kurniawati et al. '08]
- Multi-model planning [Hauser et al. '10]
- Small-tree PRM [Lanteigne et al. '11]
- Rapidly-exploring Random Roadmap [Alterovitz et al. '11]

Trees:
- EST [Hsu et al.'97, '00]
- RRT [Kuffner, LaValle '98]
- RRT-Connect [Kuffner, LaValle '00]
- 3BL [Sanchez, Latombe '01]
- RRF [Li, Shie '02]
- Guided EST [Phillips et al. '03]
- PDRRT [Ranganathan, Koenig '04]
- SRT [Plaku et al. '05]
- DDRRT [Yershova et al. '05]
- ADDRRT [Jaillet et al. '05]
- RRT-Blossom [Kalisiak, van Panne '06]
- PDST [Ladd, Kavraki '06]
- Utility RRT [Burns, Brock '07]
- GRIP [Bekris, Kavraki '07]
- Multiparticle RRT [Zucker et al. '07]
- TC-RRT [Stillman et al. '07]
- RRT-JT [Vande Wege et al.'07]
- DSLX [Plaku, Kavraki, Vardi '08]
- KPIECE [Şucan, Kavraki '08]
- RPDST [Tsianos, Kavraki '08]
- BiSpace [Diankov et al. '08]
- GRRT [Chakravorty, Kumar '09]
- IKBiRRT [Berenson et al. '09]
- CBiRRT [Berenson et al. '09]
- J+RRT [Vahrenkamp '09]
- RG-RRT [Shkolnik et al. '09]
- J2A-BiRRT [Plaku et al. '10]
OMPL API overview

- ControlSampler
- StatePropagator
- ValidStateSampler
- MotionValidator
- ControlSpace
- SpaceInformation
- User code

- StateSpace
- StateSampler
- ProjectionEvaluator
- StateValidityChecker
- SimpleSetup
- ProblemDefinition
- Goal
- Path

only when planning with differential constraints

A is owned by B

User code

must instantiate
must instantiate, unless using SimpleSetup
can instantiate, but defaults available

Wednesday, June 19, 13
Motion planning (in simulation) with a complex robot and a complex goal: specification is given with a co-safe LTL formula
LTLMoP and ROS

Gazebo

LTLMoP
Benefits

- Research
- Education
- Industry
Thank you!
States & state spaces

abstract state space
States & state spaces

API requirements:
- StateType
- alloc/free state
- distance
- interpolation
- state equality
States & state spaces

abstract state space

API requirements:
- StateType
- alloc/free state
- distance
- interpolation
- state equality

rotation (2D,3D)

translation ($\mathbb{R}^n$)
States & state spaces

abstract state space

- rotation (2D, 3D)
- translation ($\mathbb{R}^n$)

compound

used for:
- rigid body motions
- manipulators
- ...

API requirements:
- StateType
- alloc/free state
- distance
- interpolation
- state equality
Control spaces & controls

- Needed only for control-based planning

- Analogous to state spaces and states:

  - Abstract control space
  - $\mathbb{R}^n$
  - Compound

  API requirements:
  - ControlType
  - alloc/free control
  - equality
State validators

- Problem-specific; **must** be defined by user or defined by layer on top of OMPL core \(\rightarrow\) **MoveIt!**

- Checks whether state is collision-free, joint angles and velocities are within bounds, etc.

- **Optionally,** specific state validator implementations can return
  - distance to nearest invalid state (i.e., nearest obstacle)
  - gradient of distance

*Can be exploited by planners / samplers!*
Samplers

• For every **state space** there needs to be a **state sampler**

• State samplers need to support the following:
Samplers

• For every **state space** there needs to be a **state sampler**

• State samplers need to support the following:
  
  • sample uniform
Samplers

- For every **state space** there needs to be a **state sampler**
- State samplers need to support the following:
  - sample uniform
  - sample uniform near given state
Samplers

• For every **state space** there needs to be a **state sampler**

• State samplers need to support the following:
  
  • sample uniform

  • sample uniform near given state

  • sample from Gaussian centered at given state
Many ways to get sampling wrong

Example: uniformly sampling 3D orientations

naïve & wrong:  
correct:

Images from Kuffner, ICRA '04
Similar issues occur for nearest neighbors

• $k$ nearest neighbors can be computed efficiently with $kd$-trees in **low-dimensional, Euclidean** spaces.

• In high-dimensional spaces **approximate** nearest neighbors much better.

• In **non-Euclidean** spaces (e.g., any space that includes **rotations**), other data structures are necessary.
**Valid state samplers**

- **Valid state samplers** combine low-level **state samplers** with the **validity checker**

- Simplest form: sample at most $n$ times to get valid state or else return failure
Valid state samplers

- **Valid state samplers** combine low-level **state samplers** with the **validity checker**

  - Simplest form: sample at most $n$ times to get valid state or else return failure

  - Other sampling strategies:
**Valid state samplers**

- **Valid state samplers** combine low-level state samplers with the **validity checker**

- Simplest form: sample at most \( n \) times to get valid state or else return failure

- Other sampling strategies:
  - Try to find samples with a large clearance
Valid state samplers

• **Valid state samplers** combine low-level state samplers with the validity checker

• Simplest form: sample at most $n$ times to get valid state or else return failure

• Other sampling strategies:
  
  • Try to find samples with a large clearance
  
  • Try to find samples near obstacles (more dense sampling in/near narrow passages)
Goals

- **Goal**: can only tell whether state satisfies Goal condition
- **GoalRegion**: provides distance to goal region
- **GoalSampleableRegion**: can sample from goal region
- **GoalState**: single goal state
- **GoalStates**: multiple goal states
- **GoalLazySamples**: multiple goal states, computed in separate thread
OMPL planning algorithms

• Take as input a problem definition: object with one or more start states and a goal object

• Planners need to implement two methods:

  • solve:
    – takes PlannerTerminationCondition object as argument
    – termination can be based on timer, external events, ...

  • clear:
    clear internal data structures, free memory, ready to run solve again
Many planners available in OMPL

**planning with controls**

KPIECE
RRT
EST
Syclop
PDST

**geometric planning**

KPIECE, BKPIECE, LBKPIECE
PRM, LazyPRM
RRT, RRTConnect, LazyRRT
EST, SBL
PDST
STRIDE

Optimizing planners:

PRM*
RRT*, BallTreeRRT*
T-RRT
SPARS, SPARS-2

= available soon!
API overview

only when planning with differential constraints

- ControlSampler → ControlSpace
- StatePropagator
- ValidStateSampler → SpaceInformation
- StateSampler
- StateSpace
- StateValidityChecker
API overview

Only when planning with differential constraints

- ControlSampler
- ControlSpace
- StateSpace
- StateSampler
- StatePropagator
- SpaceInformation
- StateValidityChecker
- ValidStateSampler
- Planner
- SimpleSetup
- ProblemDefinition
- Goal

User code

- Must instantiate
- Must instantiate, unless using SimpleSetup
- Can instantiate, but defaults available
- A → B
  A is owned by B
API overview

only when planning with differential constraints

- **ControlSampler** → **ControlSpace**
- **StatePropagator**
- **ValidStateSampler** → **SpaceInformation**
- **StateSampler** ← **StateSpace**
- **StateValidityChecker** ← **StateSpace**
- **Planner** → **SimpleSetup**
- **ProblemDefinition** ← **SimpleSetup**
- **Goal**
- **Path**

User code

- **must instantiate**
- **must instantiate, unless using SimpleSetup**
- **can instantiate, but defaults available**
- **A→B** → **A is owned by B**

Wednesday, June 19, 13
**API overview**

only when planning with differential constraints

- ControlSampler → ControlSpace
- StatePropagator
- ValidStateSampler → SpaceInformation
- MotionValidator
- Planner → SimpleSetup
- StateSampler
- ProjectionEvaluator
- StateSpace
- StateValidityChecker
- Path
- ProblemDefinition
- Goal

User code

- must instantiate
- must instantiate, unless using SimpleSetup
- can instantiate, but defaults available
- A → B
  - A is owned by B

Wednesday, June 19, 13
Minimal code example

```python
space = SE3StateSpace()
# set the bounds (code omitted)

ss = SimpleSetup(space)
# "isStateValid" is a user-supplied function
ss.setStateValidityChecker(isStateValid)

start = State(space)
goal = State(space)
# set the start & goal states to some values
# (code omitted)

ss.setStartAndGoalStates(start, goal)
solved = ss.solve(1.0)
if solved:
    print setup.getSolutionPath()
```
StateSpacePtr space(new SE3StateSpace());

// set the bounds (code omitted)

SimpleSetup ss(space);

// "isValidState" is a user-supplied function
ss.setStateValidityChecker(isStateValid);

ScopedState<SE3StateSpace> start(space);
ScopedState<SE3StateSpace> goal(space);

// set the start & goal states to some values
// (code omitted)

ss.setStartAndGoalStates(start, goal);

bool solved = ss.solve(1.0);

if (solved)
    setup.getSolutionPath().print(std::cout);
Benchmarking

![Graph showing benchmarking results for different algorithms]

- RRTConnect
- RRT
- LBKPIECE1
- KPIECE1
- SBL
- EST
- BasicPRM

![Bar chart showing performance comparison]

- % of Tests

Wednesday, June 19, 13
Benchmarks

SimpleSetup setup;
// motion planning problem setup code omitted
Benchmark b(setup, "My First Benchmark");

b.addPlanner(base::PlannerPtr(new geometric::RRT(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::KPIECE1(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::SBL(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::EST(setup.getSpaceInformation())));
b.addPlanner(base::PlannerPtr(new geometric::PRM(setup.getSpaceInformation())));

b.benchmark(runtime_limit, memory_limit, run_count, true);
b.saveResultsToFile();

Script post-processes benchmark log files to create/update SQLite database and plots
OMPL.app
OMPL.app
OMPL.app
Sample OMPL.app problems
Resources to get started with OMPL
The Open Motion Planning Library

OMPL, the Open Motion Planning Library, consists of many state-of-the-art sampling-based motion planning algorithms. OMPL itself does not contain any code related to, e.g., collision checking or visualization. This is a deliberate design choice, so that OMPL is not tied to a particular collision checker or visualization front end.

OMPL.app, the front-end for OMPL, contains a lightweight wrapper for the FCL and PQP collision checkers and a simple GUI based on PyQt / PySide. The graphical front-end can be used for planning motions for rigid bodies and a few vehicle types (first-order and second-order cars, a blimp, and a quadrotor). It relies on the Assimp library to import a large variety of mesh formats that can be used to represent the robot and its environment.

Contents of This Library

- OMPL contains implementations of many sampling-based algorithms such as PRM, RRT, EST, SBL, KPIECE, SyCLOP, and several variants of these planners. See available planners for a complete list.
- All these planners operate on very abstractly defined state spaces. Many commonly used state spaces are already implemented (e.g., SE2, SE3, \( \mathbb{R}^n \), etc.).
- For any state space, different state samplers can be used (e.g., uniform, Gaussian, obstacle based, etc.).
- API overview
- Documentation for just the OMPL core library (i.e., without the "app" layer).

Getting Started

- The OMPL primer provides a brief background on sampling-based motion planning, and an overview of OMPL.
- Download and install OMPL.
- Learn how to use the OMPL.app GUI.
- Demos and tutorials
- Frequently Asked Questions
- Familiarize yourself with the Boost structures used throughout OMPL.
- Learn how to integrate your own code with OMPL's build system.
- If interested in using Python, make sure to read the documentation for the Python bindings.

Other Resources

- OMPL for education
- Gallery of example uses of OMPL.
- If you are interested in the ROS interface to OMPL, please read the tutorial on using OMPL within ROS.
- Third-party contributions. (Contribute your own extensions!)

News & Events

- OMPL has been accepted as a mentoring organization for the 2013 Google Summer of Code!
- OMPL has won the 2012 Open Source Software World Grand Challenge!
- An article about OMPL has been accepted for publication in IEEE's Robotics & Automation Magazine! It will appear in the December 2012 issue.
- At ROSCON, Sachin Chitta and Ioan \( \text{\c{S}} \text{\c{c}} \text{\c{a}} \text{n} \) gave a talk about MoveIt!, the new motion planning stack in ROS. It provides a common interface to motion planning libraries in ROS (including OMPL). It will eventually replace the arm navigation stack.
- IROS 2011 Tutorial on Motion Planning for Real Robots. This hands-on tutorial describes how to use the ROS and OMPL, but it also gives some background on sampling-based motion planning.
The Open Motion Planning Library

OMPL, the Open Motion Planning Library, consists of many state-of-the-art sampling-based motion planning algorithms. OMPL itself does not contain any code related to, e.g., collision checking or visualization. This is a deliberate design choice, so that OMPL is not tied to a particular collision checker or visualization front end.

OMPLApp, the front-end for OMPL, contains a lightweight wrapper for the FCL and PQP collision checkers and a simple GUI based on PyQt / PySide. The graphical front-end can be used for planning motions for rigid bodies and a few vehicle types (first-order and second-order cars, a blimp, and a quadrotor). It relies on the Assimp library to import a large variety of mesh formats that can be used to represent the robot and its environment.

Current version: 0.12.2
Released: Jan 22, 2013
Click for citation, if you use OMPL in your work

Online at:
http://ompl.kavrakilab.org

Contact us at:
ompl-devel@lists.sourceforge.net
ompl-users@lists.sourceforge.net

Public repositories at:
https://bitbucket.org/ompl
OMPL for education

• Programming assignments centered around OMPL, available upon request.

• Ongoing educational assessment.

• Already in use in several robotics / motion planning classes.

Happy OMPL users: students in the Algorithmic Robotics class at Rice, Fall 2010
Discussion

• OMPL actively developed, but ready for general use

• Can easily implement new algorithms from many reusable components

• Simple high-level interface:
  • Can treat motion planner almost as a black box
  • Easy enough that non-experts can use it

• Interface generic enough to be extensible in many ways

*We want your contributions!*
Acknowledgements

Rice University:
Lydia Kavraki
Ryan Luna
Matt Maly
Bryant Gipson
Devin Grady
Amit Bhatia

Willow Garage:
Ioan Şucan
Sachin Chitta

Funding from:
NSF CCLI grant #0920721
Willow Garage