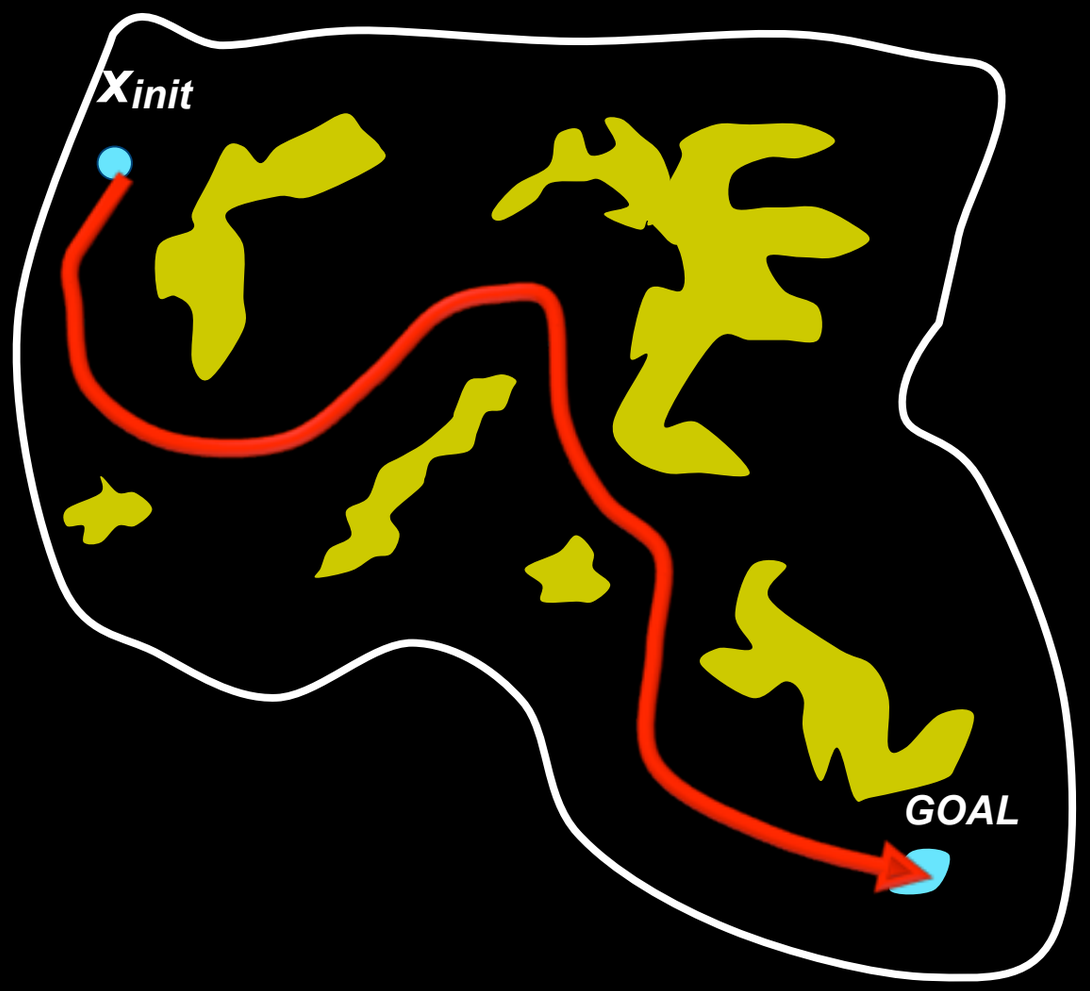
OOPSMP:
An Object-Oriented Programming System for Motion Planning

Mark Moll & Lydia E. Kavraki
Physical & Biological Computing Group
Rice University
Houston, TX
USA

Introduction

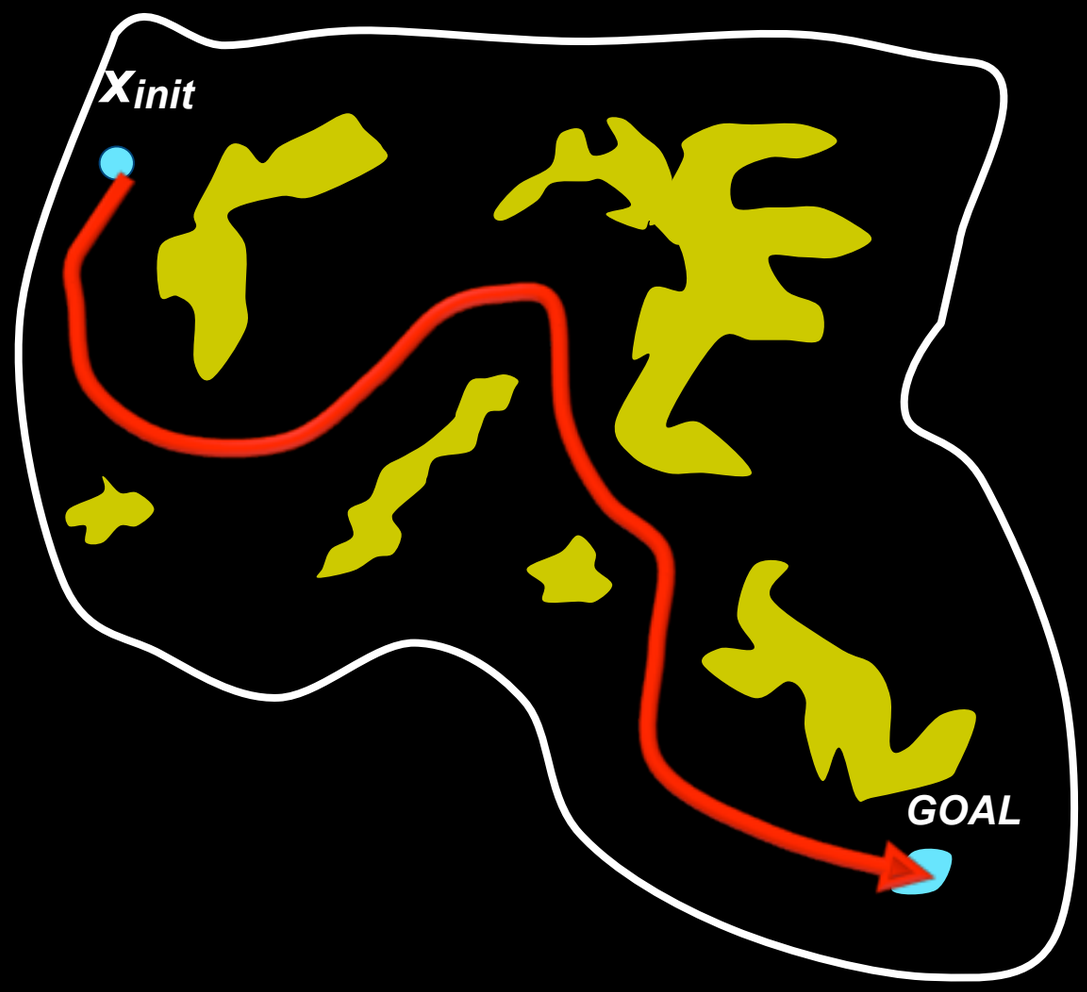
Motion planning: given two configurations of a robot, find a collision-free path that connects them.



Introduction

Motion planning: given two configurations of a robot, find a collision-free path that connects them.

More generally: find paths for one or more robots that satisfy all constraints on geometry, physics, safety, etc.



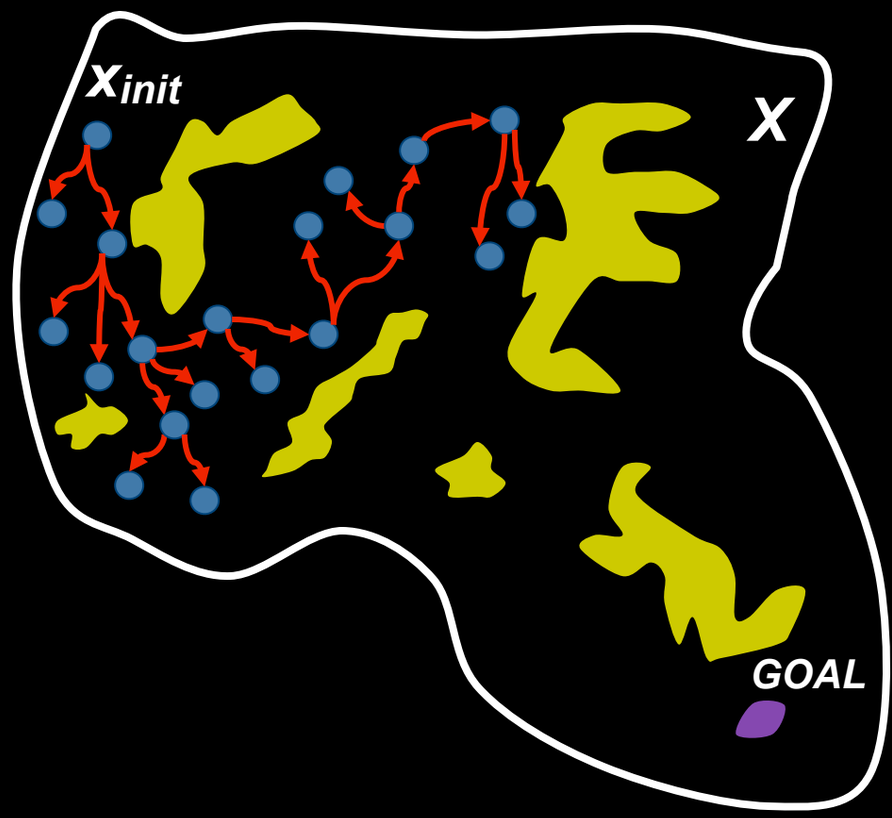
Motion planning problems are hard

PROBLEM	COMPLEXITY
Geometric Constraints:	
Sofa Mover (3DOF)	$O(n^{2+\varepsilon})$ - not implemented [HS96]
Piano Mover (6DOF)	Polynomial – no practical algorithm [SS83]
n Disks in the Plane	NP-Hard [SS83]
n Link Chain in 3D	PSPACE-Complete [HSS87]
Generalized Mover	PSPACE-Complete [Canny88]
Dynamics Constraints:	
Point with Newtonian Dynamics	NP-Hard [DXCR93]
Polygon Dubin's Car (Linear)	Decidable [CPK08]
Nonlinear	Unknown, probably undecidable
Discrete Transitions and Dynamics Constraints:	
Hybrid Systems	Undecidable [Alur et. al 95]

Exact, Approximate, and Probabilistically Complete Algorithms

Method	Advantage	Disadvantage
exact	theoretically insightful	impractical
cell decomposition	easy	does not scale
control-based	online, very robust	requires good trajectory
potential fields	online, easy	slow or fail
sampling-based	fast and effective	cannot recognize impossible query

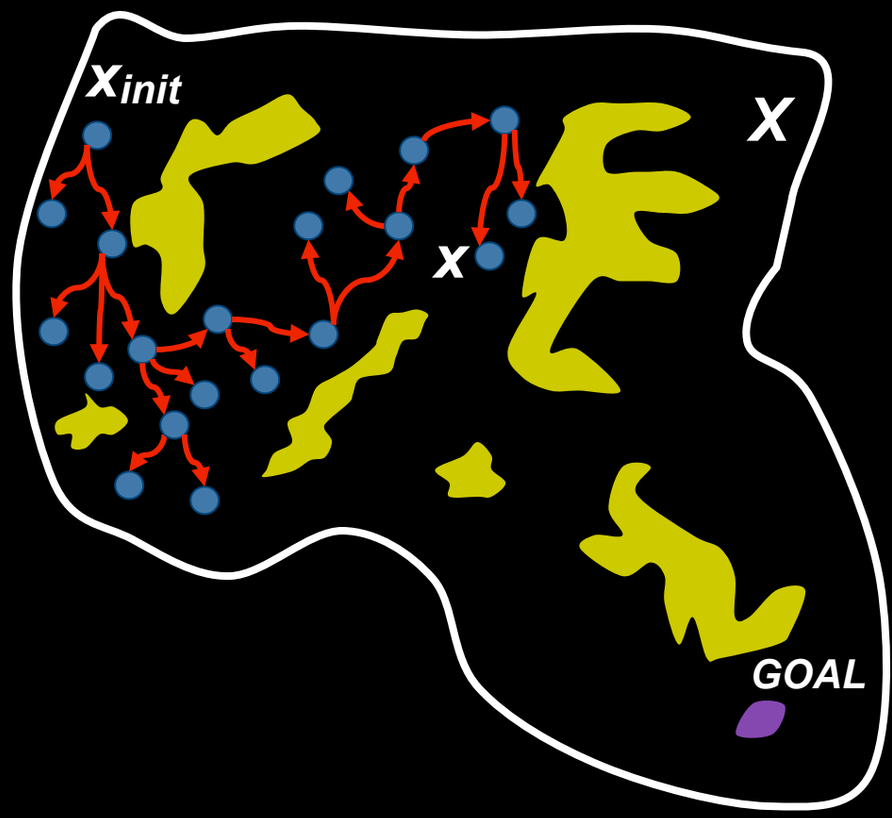
Sampling-based algorithms



Sampling-based algorithms

Compute a trajectory from an initial state to a goal region

Trajectory should satisfy all state constraints (e.g., no collisions) and dynamics constraints

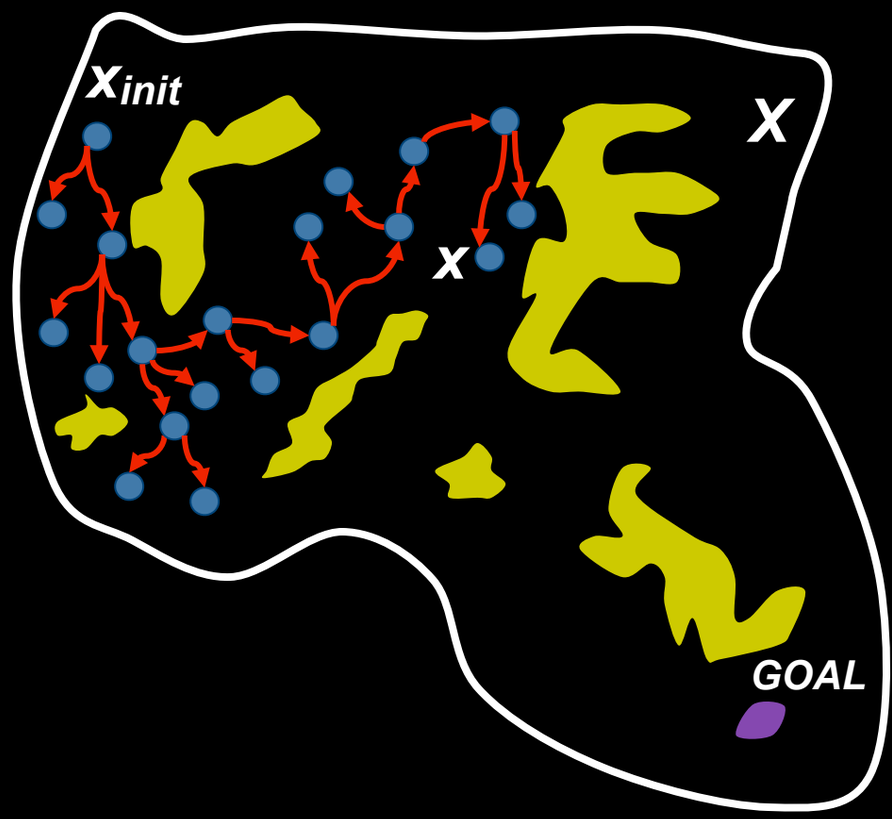


Sampling-based algorithms

Compute a trajectory from an initial state to a goal region

Trajectory should satisfy all state constraints (e.g., no collisions) and dynamics constraints

Basic tree-search framework



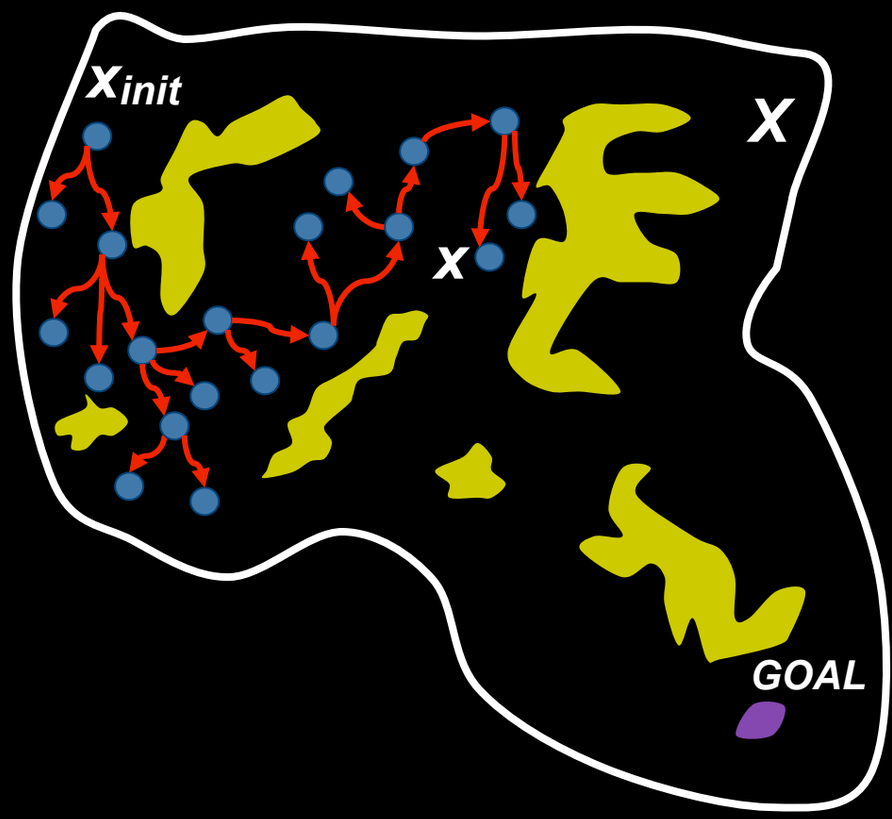
Sampling-based algorithms

Compute a trajectory from an initial state to a goal region

Trajectory should satisfy all state constraints (e.g., no collisions) and dynamics constraints

<u>Basic tree-search framework</u>

repeat until solution



Sampling-based algorithms

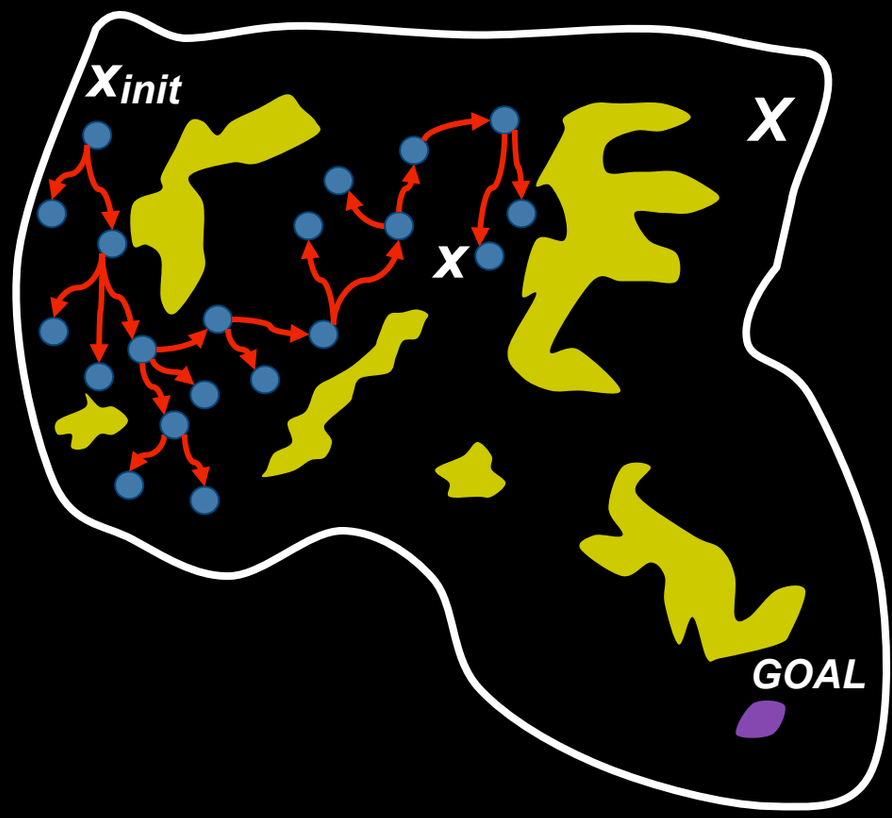
Compute a trajectory from an initial state to a goal region

Trajectory should satisfy all state constraints (e.g., no collisions) and dynamics constraints

<u>Basic tree-search framework</u>

repeat until solution

- select state ***x*** from tree ***T***



Sampling-based algorithms

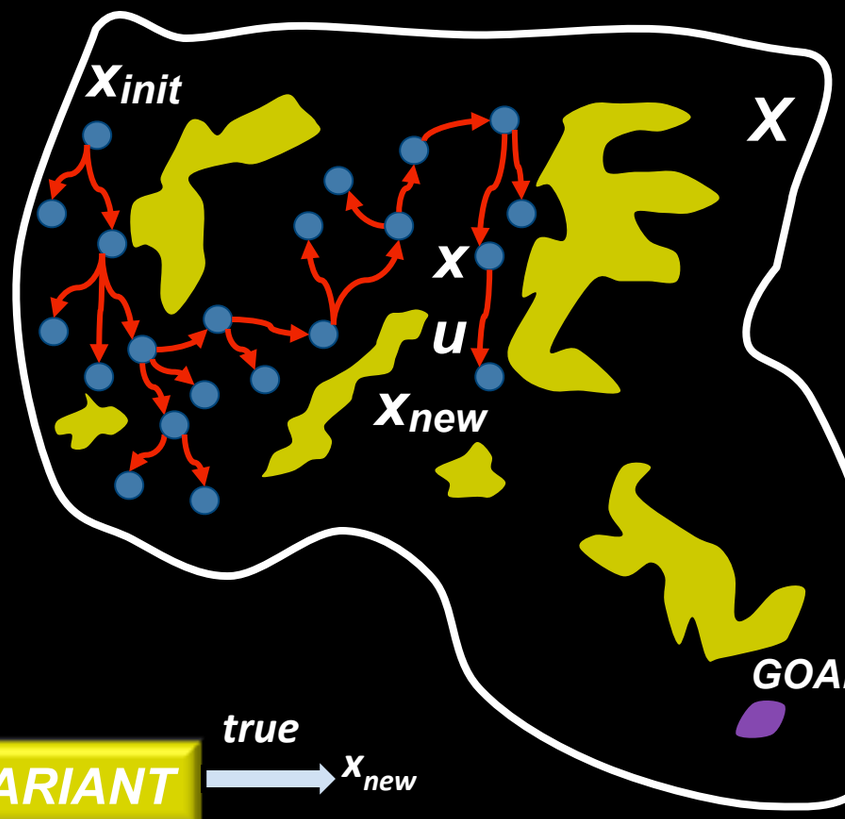
Compute a trajectory from an initial state to a goal region

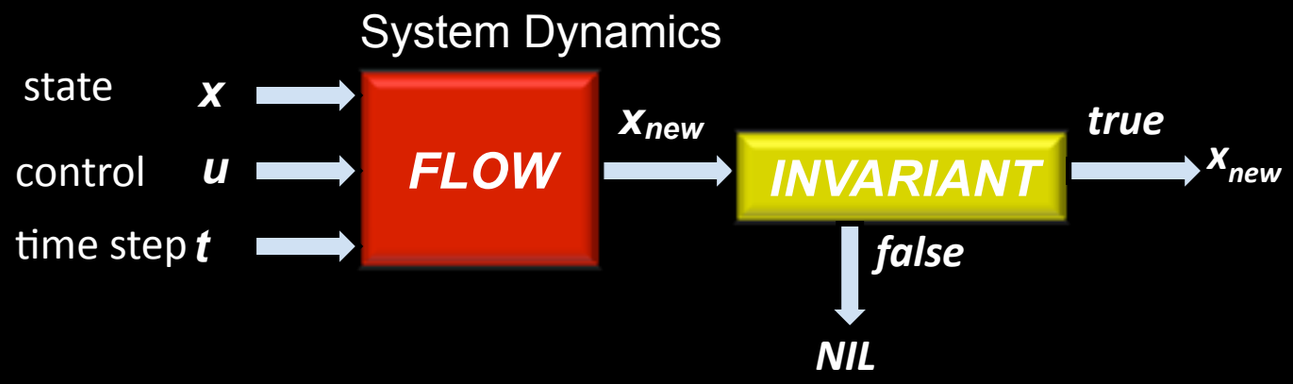
Trajectory should satisfy all state constraints (e.g., no collisions) and dynamics constraints

<u>Basic tree-search framework</u>

repeat until solution

- select state ***x*** from tree ***T***
- create a new tree branch from ***x***





Sampling-based algorithms

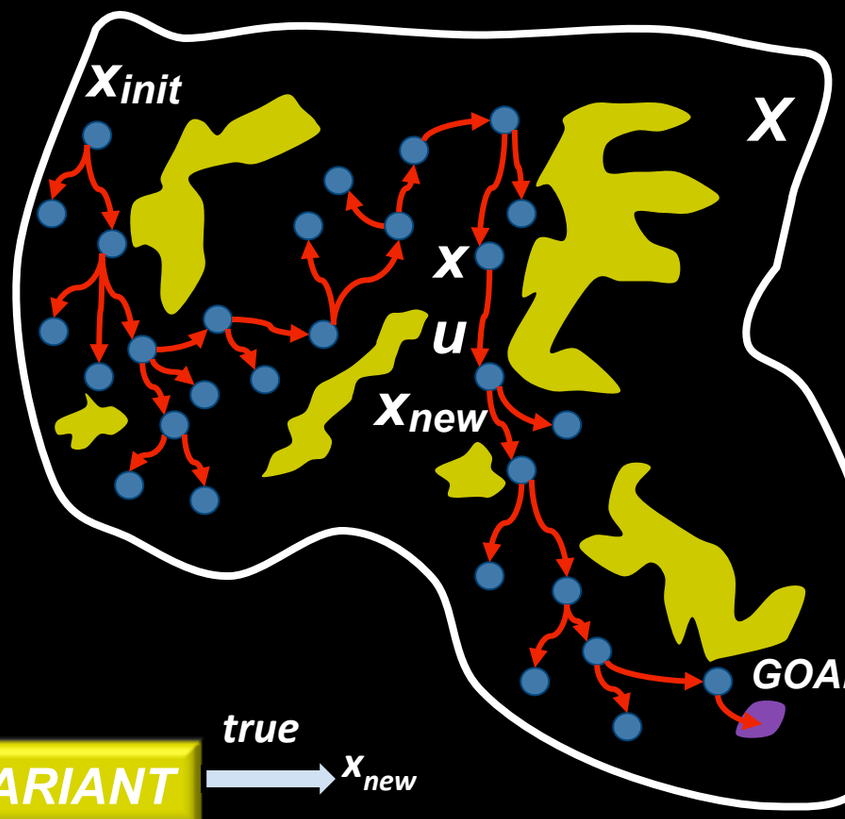
Compute a trajectory from an initial state to a goal region

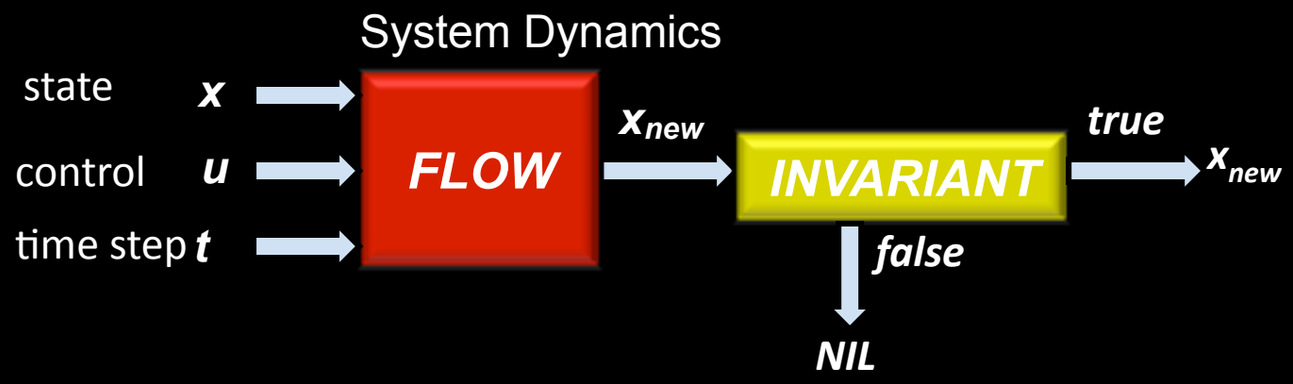
Trajectory should satisfy all state constraints (e.g., no collisions) and dynamics constraints

<u>Basic tree-search framework</u>

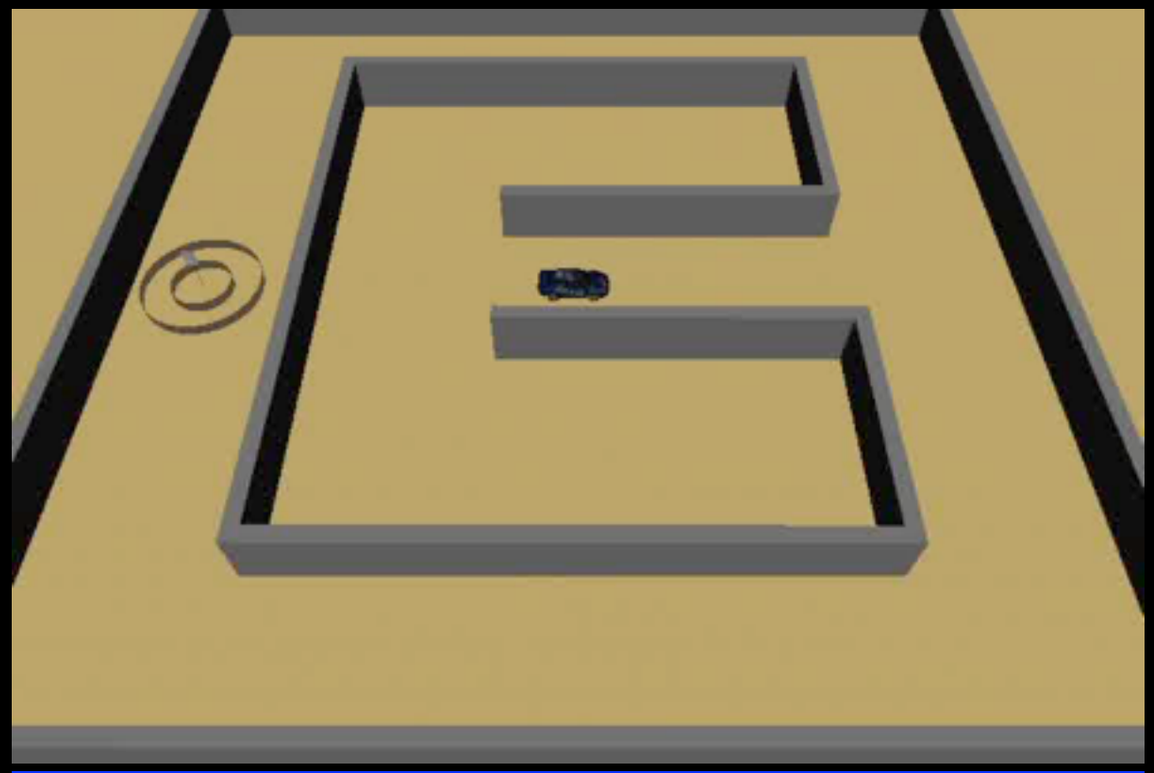
repeat until solution

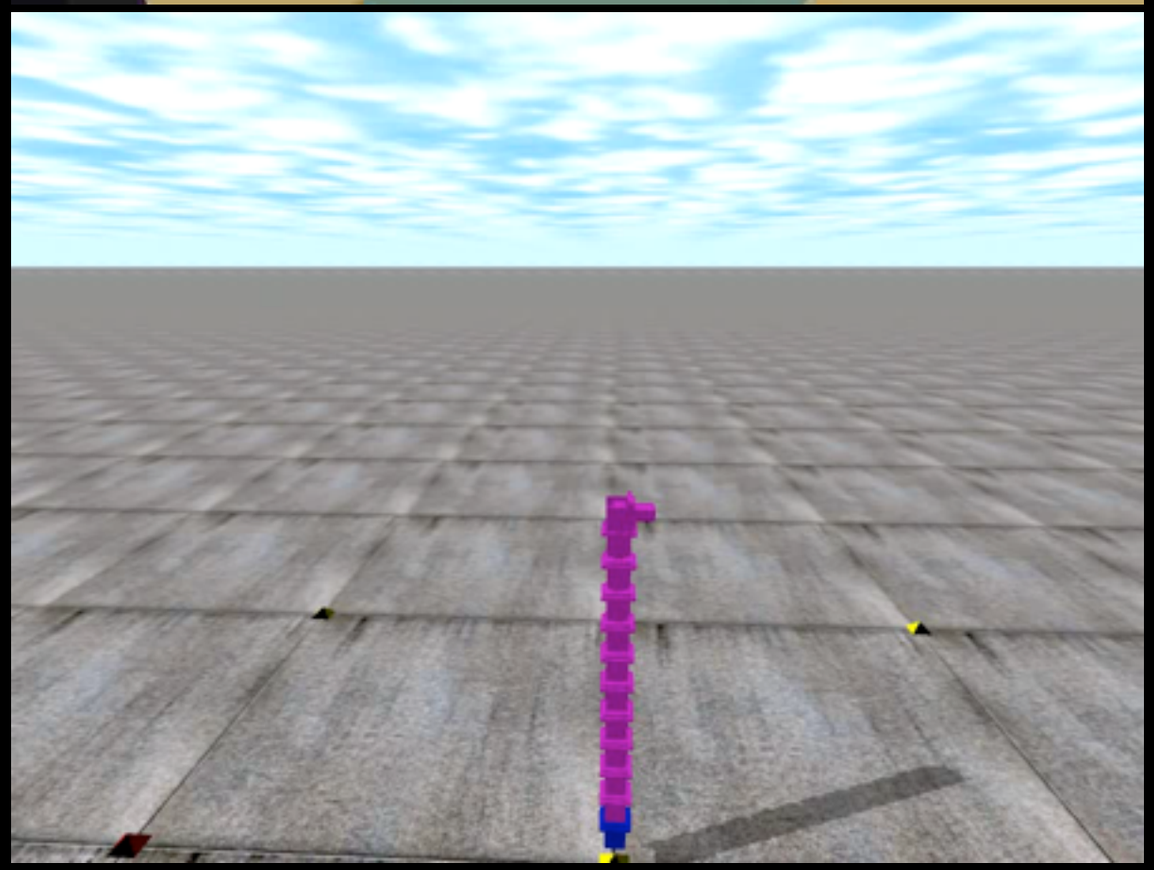
- select state ***x*** from tree ***T***
- create a new tree branch from ***x***

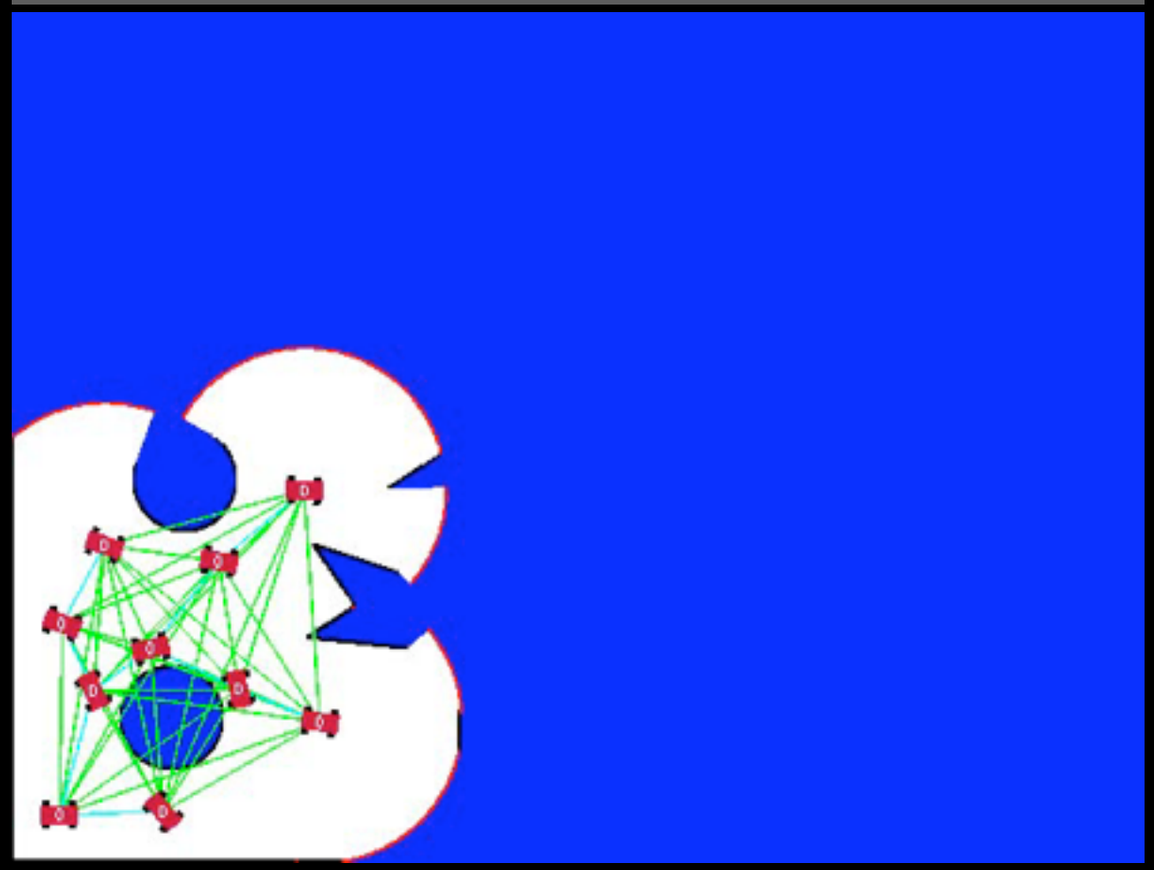




Examples







Outline

- Motivation
- Main features of OOPSMP
- Visual walk-through of using OOPSMP
- OOPSMP plug & play
- Concluding remarks

Motivation

The need for motion planning software

- Teaching tool:
  - Play with different algorithms
  - Quickly create interesting motion planning problems
  - Minimal programming overhead

The need for motion planning software

- Research tool:
  - Easily compare new algorithms with “standard” algorithms
  - Run algorithms on benchmark problems
  - Reuse common data structures

Existing motion planning software

- **Motion Strategies Library**, LaValle et al.
- **Motion Planning Kit**, Schwarzer, Saha, Latombe
- **KineoWorks**, Laumond et al.
- **OpenRAVE**, Diankov & Kuffner

OOPSMP:

Object-Oriented Programming System for Motion Planning

OOPSMP:

Object-Oriented Programming System for Motion Planning

- contains several motion planning algorithms
- is easily extensible with new algorithms
- has an easy to use graphical front-end
- runs on a variety of platforms
- already used in several classes and research groups around the world

Main features

- Stand-alone version
  - command line version
  - graphical version
- Can be used as a library
- Google SketchUp plugin
  - easy to create motion planning problems
- Source code available for download
- Runs on Windows, Linux, and Apple OS X

Motion planner components

Motion planner components

- high-level planning algorithms

PRM,
RRT,
EST,
bi-tree planner,
DDRRT*,
VisPRM*

Motion planner components

- high-level planning algorithms
- sampling strategies

uniform, Gaussian, obstacle-based, bridge test

Motion planner components

- high-level planning algorithms
- sampling strategies
- connection strategies

random
neighbors,
approx. n.n.
exact n.n.

Motion planner components

- high-level planning algorithms
- sampling strategies
- connection strategies
- path generation

geodesics, best or random control, collisions

Motion planner components

- high-level planning algorithms
- sampling strategies
- connection strategies
- path generation
- state spaces

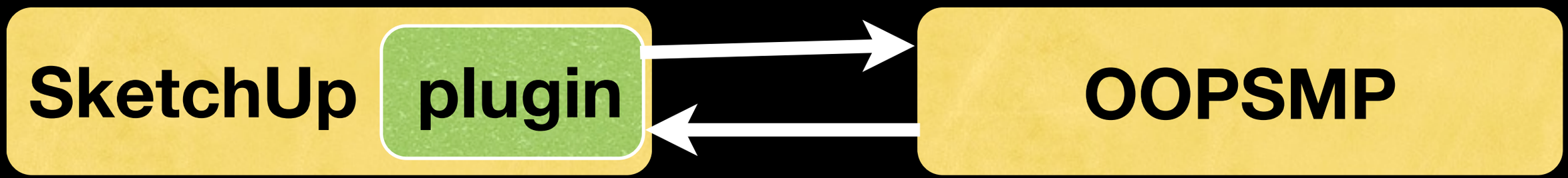
SE(2),
SE(3),
car-like
vehicles,
multiple
robots

OOPSMP workflow

- Define motion planning problem:
  - Define an environment
  - Define a robot
  - Specify queries
  - Specify global planner (PRM, RRT, etc.)
  - Specify local planner
- Run OOPSMP to solve motion planning problem

Google SketchUp:
a graphical front-end to OOPSMP

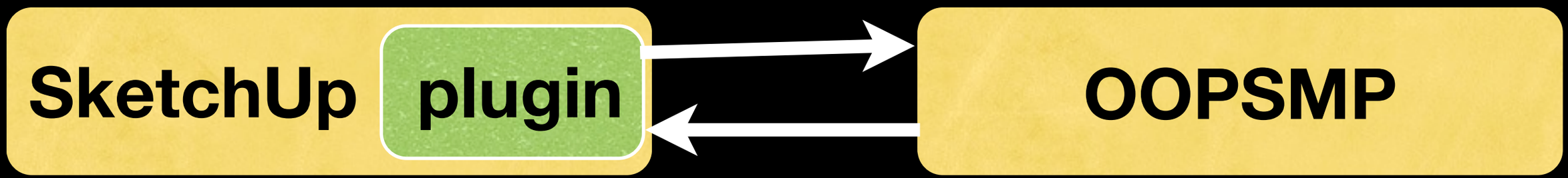
XML spec. of
motion planning problem



solution paths

Google SketchUp:
a graphical front-end to OOPSMP

XML spec. of
motion planning problem

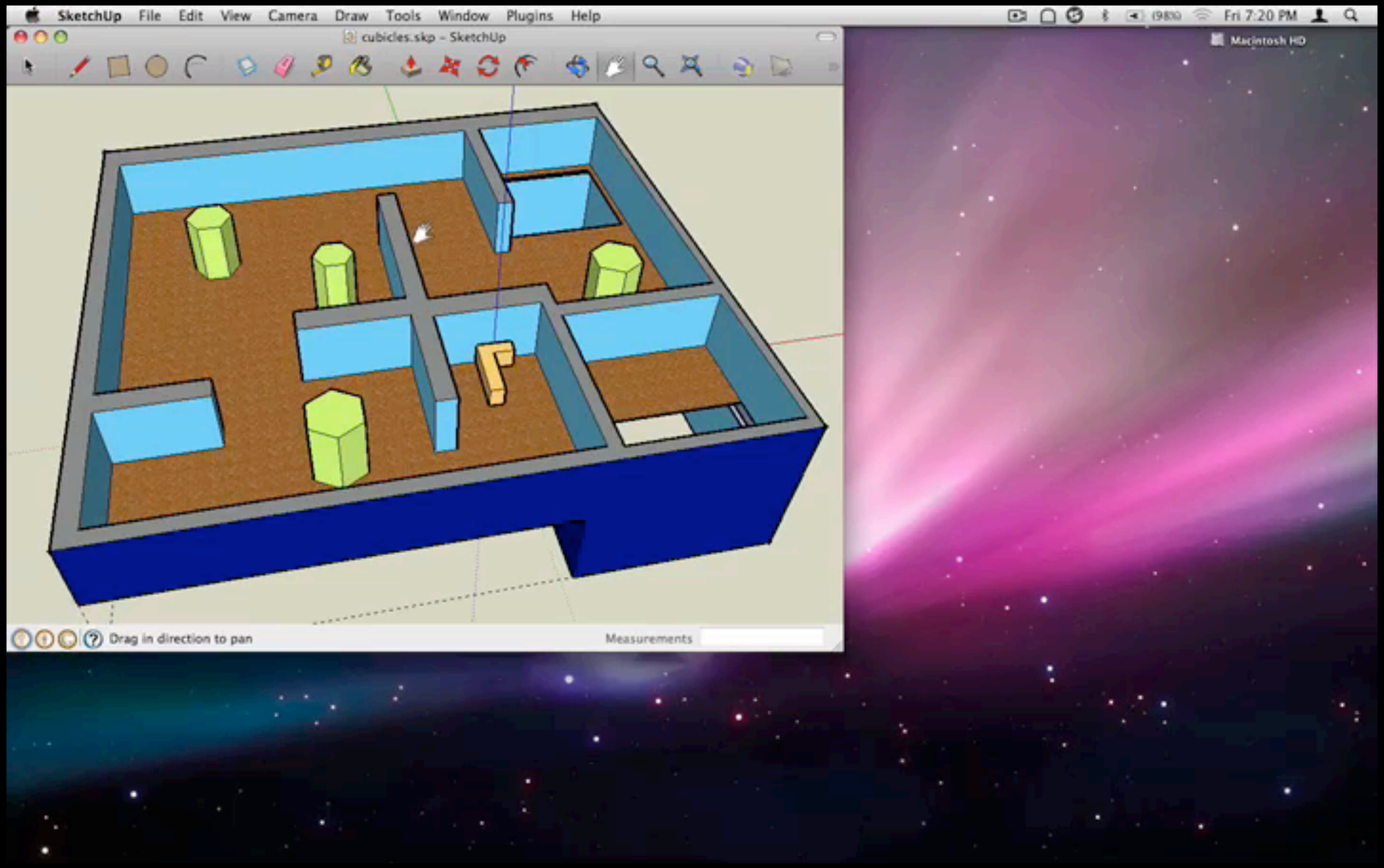


solution paths

Pros: ease of use, access to many 3D models

Cons: does not (yet) support all OOPSMP features,
not available for Linux

Motion planning with OOPSMP & SketchUp



OOPSMP Plug & Play

OOPSMP input

- XML input
  - Pros: self-describing, easy to parse/generate
  - Cons: very verbose
- OOPSMP XML files can include other XML files
  - Can reuse environment/robot for many motion planning problems

XML code for example

PRM.xml

```
1  <factory instance="Manager">
2    <call fn="setProgram">
3      <arg type="pointer">
4        <factory instance="MotionPlannerProgram">
5          <call fn="setFileNames">…
19         <call fn="setQueries">…
51         <call fn="setCoreRobotData">…
220        <call fn="setMotionPlanner">…
247        <call fn="setPreprocessTime">…
250        <call fn="setSolveTime">…
253      </factory>
254    </arg>
255  </call>
256</factory>
```

Line: 257 Column: 1 XML Tab Size: 2

XML code for example

PRM.xml

```
1  <factory instance="Manager">
2    <call fn="setProgram">
3      <arg type="pointer">
4        <factory instance="MotionPlannerProgram">
5          <call fn="setFileNames">…
19         <call fn="setQueries">…
51         <call fn="setCoreRobotData">…
220        <call fn="setMotionPlanner">…
247        <call fn="setPreprocessTime">…
250        <call fn="setSolveTime">…
253      </factory>
254    </arg>
255  </call>
256</factory>
```

create object

Line: 257 Column: 1 XML Tab Size: 2

XML code for example

PRM.xml

```
1  <factory instance="Manager">
2    <call fn="setProgram">
3      <arg type="pointer">
4        <factory instance="MotionPlannerProgram">
5          <call fn="setFileNames">…
19         <call fn="setQueries">…
51         <call fn="setCoreRobotData">…
220        <call fn="setMotionPlanner">…
247        <call fn="setPreprocessTime">…
250        <call fn="setSolveTime">…
253      </factory>
254    </arg>
255  </call>
256</factory>
```

create object

call object's method

Line: 257 Column: 1 XML Tab Size: 2

XML code maps to C++ objects/methods

Main Page | Namespaces | Classes | Files | Directories | Related Pages

Alphabetical List | Class List | Class Hierarchy | Class Members

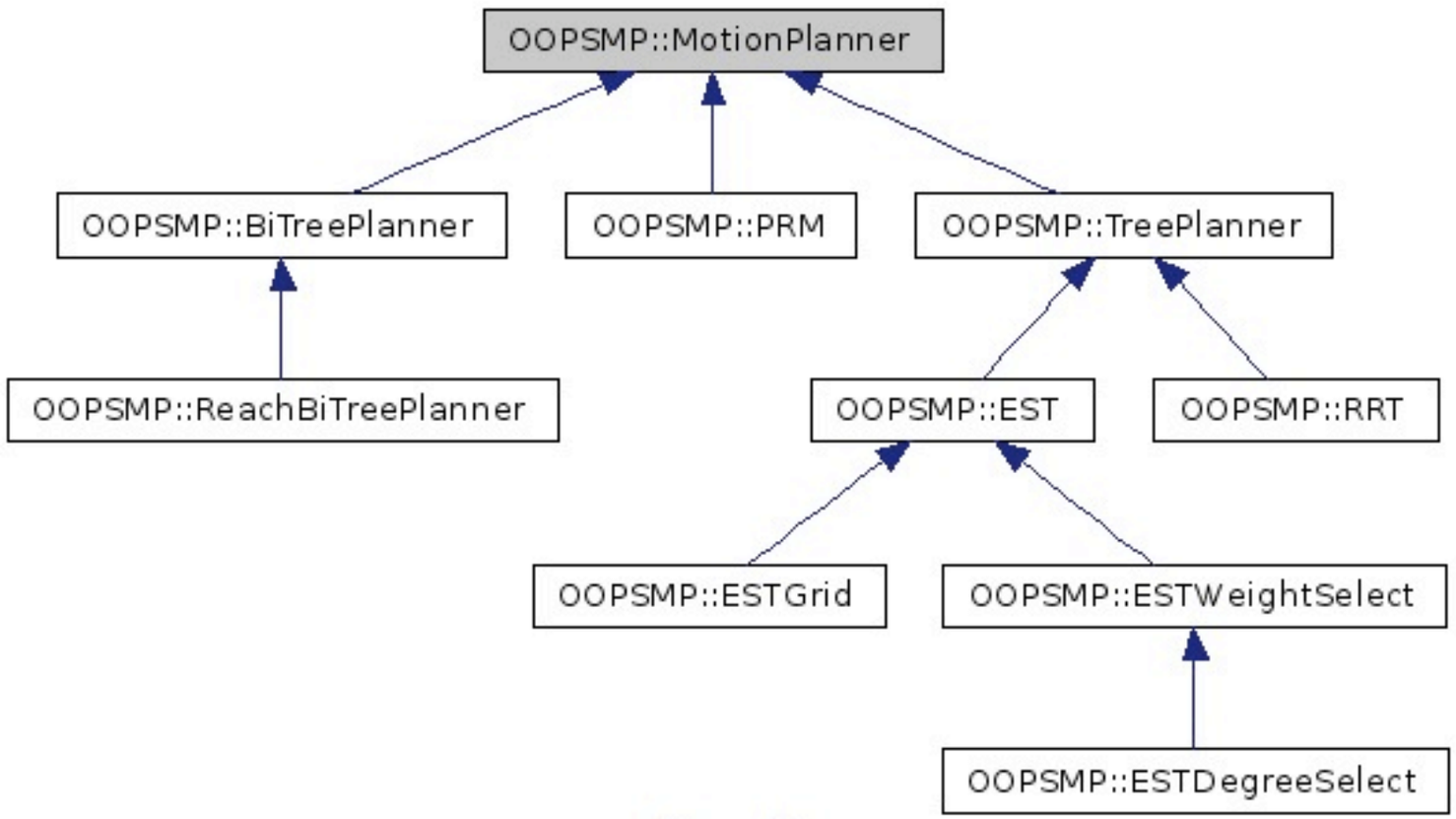
OOPSMP::MotionPlanner

OOPSMP::MotionPlanner Class Reference

Common functionality every motion planner should provide [written by Erion Plaku]. More...

#include <OOPSMPMotionPlanner.H>

Inheritance diagram for OOPSMP::MotionPlanner:



[legend]

XML code maps to C++ objects/methods

Free the memory allocated to different data elements.

Access motion planner data

bool	**areGapsAllowed** (void) const *Indicate whether gaps are allowed in path connections, i.e., path from a to b does not necessarily end at b , but somewhere near b.*
Query_t	**getQuery** (void) const *Access the current query the motion planner is solving.*
virtual bool	**isProblemSolved** (void) const=0 *Indicate whether the current motion planning query has been solved or not.*

Setup motion planner

void	**setCoreRobotData** (CoreRobotData_t crd) *Setup the core robot data, i.e., motion planning components.*
virtual void	**allowGaps** (const bool allowGaps) *Indicate whether or not gaps are allowed, i.e., path from a to b does not necessarily end at b , but somewhere near b.*
virtual void	**setQuery** (Query_t query) ***Set** the motion planning qeury.*

Use motion planner to solve one or more queries

virtual void	**solveMultipleQueries** (const double ptime, const double stime, Queries_t queries, const int *perm=NULL)=0 *Solve multiple motion planning queries.*
virtual void	**solveSingleQuery** (const double ptime, const double stime, Query_t query) *Solve a single motion planning query.*

Solve the current motion planner query

OOPSMP Plug & Play

Can add new global planners, local planners, collision checkers, etc. without having to recompile main OOPSMP code:

- Derive from abstract classes that define API
- Bundle new code in a plugin
- New code automatically accessible from XML

Future directions

Future directions

- Add more planners:
  - Path-Directed Subdivision of Trees (PDST)
  - TemporalHyDICE (planner for hybrid systems + LTL)
  - ...

Future directions

- Add more planners:
  - Path-Directed Subdivision of Trees (PDST)
  - TemporalHyDICE (planner for hybrid systems + LTL)
  - ...
- Add support for physics engines
  - *Get support for articulated mechanisms “for free”!*

Future directions

- Add more planners:
  - Path-Directed Subdivision of Trees (PDST)
  - TemporalHyDICE (planner for hybrid systems + LTL)
  - ...
- Add support for physics engines
  - *Get support for articulated mechanisms “for free”!*
- Add support for COLLADA 1.5
  - file format for geometry, kinematics, physics

Summary

With OOPSMP it is relatively easy to:

Summary

With OOPSMP it is relatively easy to:

- “play” with various state-of-the-art sampling-based motion planning algorithms,

Summary

With OOPSMP it is relatively easy to:

- “play” with various state-of-the-art sampling-based motion planning algorithms,
- add new algorithms,

Summary

With OOPSMP it is relatively easy to:

- “play” with various state-of-the-art sampling-based motion planning algorithms,
- add new algorithms,
- develop new benchmarks,

Summary

With OOPSMP it is relatively easy to:

- “play” with various state-of-the-art sampling-based motion planning algorithms,
- add new algorithms,
- develop new benchmarks,
- perform parameter sweeps,

Summary

With OOPSMP it is relatively easy to:

- “play” with various state-of-the-art sampling-based motion planning algorithms,
- add new algorithms,
- develop new benchmarks,
- perform parameter sweeps,
- compare algorithm performance,

Summary

With OOPSMP it is relatively easy to:

- "play" with various state-of-the-art sampling-based motion planning algorithms,
- add new algorithms,
- develop new benchmarks,
- perform parameter sweeps,
- compare algorithm performance,
- embed motion planning system in larger robotic system.

OOPSMP team

Christopher Alme
Kostas Bekris
Nick Bridle
Drew Bryant
Neal Ehardt
Nicolas Feltman
Nurit Haspel
Allison Heath
Lydia E. Kavraki
Andrew Ladd
Mark Moll
Erion Plaku
Amarda Shehu
Ioan Şucan
Konstantinos Tsianos
You!

lead developer

http://kavrakilab.org/OOPSMP