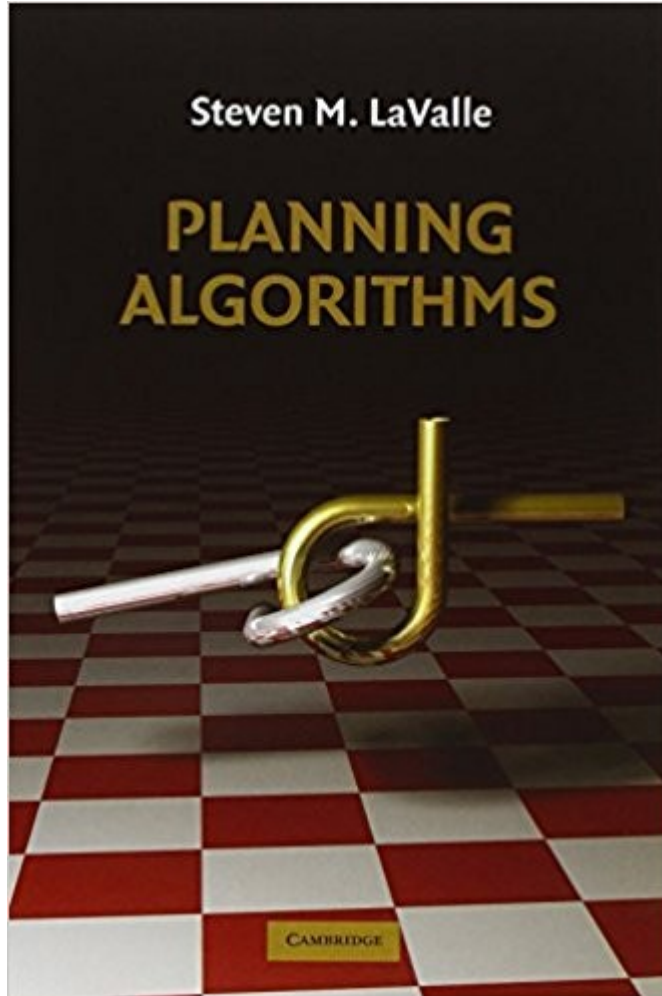


Motion Planning: Rapidly-Exploring Random Trees

Adrian Sieler, adrian.sielier@tu-berlin.de

Book recommendation

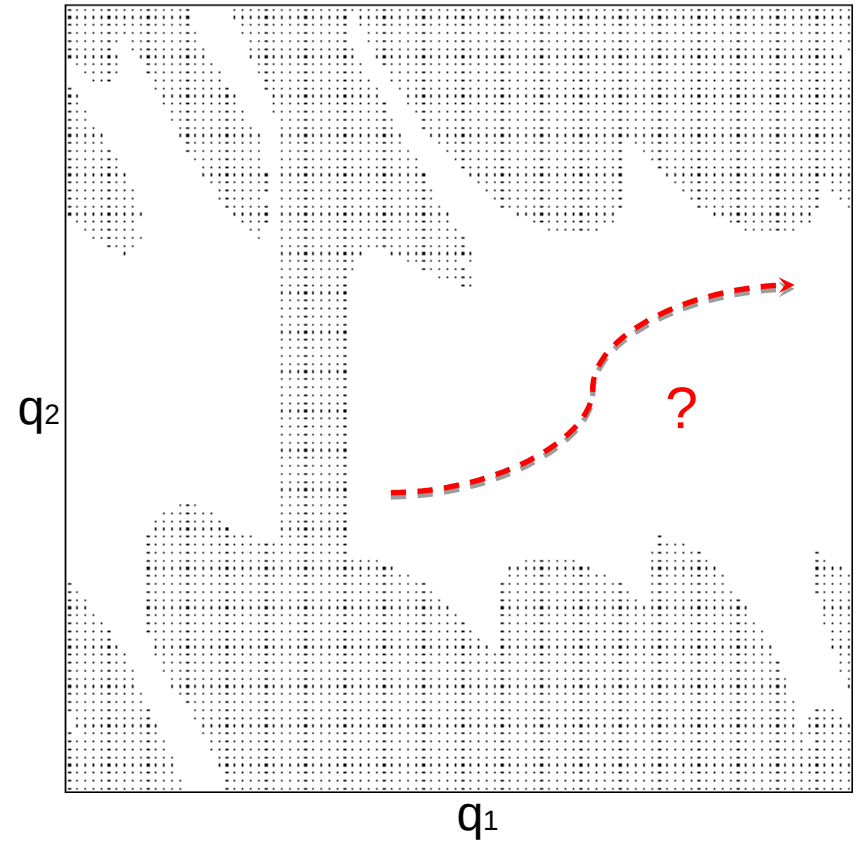
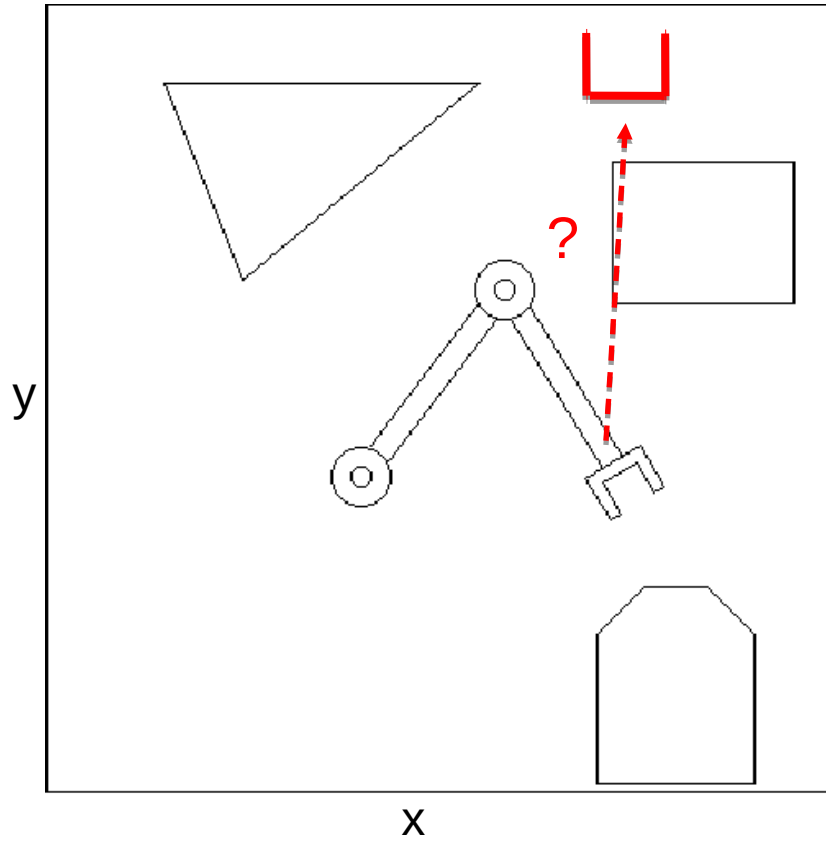


Available for free - online!

<http://planning.cs.uiuc.edu/>

Chapter:
**Sampling-based
Motion Planning**

Motion planning



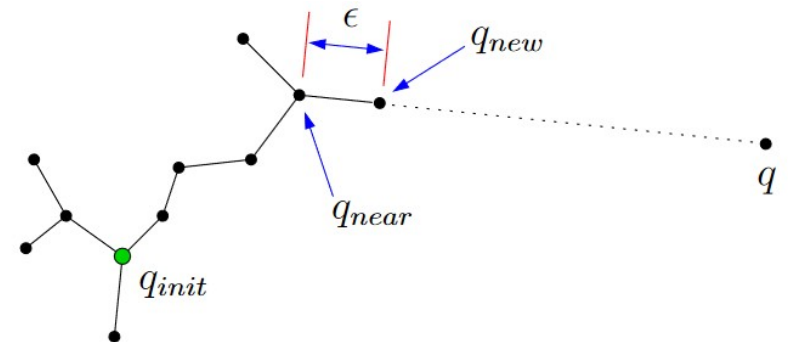
Rapidly-Exploring Random Trees

BUILD_RRT(q_{init})

```
1   $\mathcal{T}.$ init( $q_{init}$ );  
2  for  $k = 1$  to  $K$  do  
3       $q_{rand} \leftarrow \text{RANDOM\_CONFIG}()$ ;  
4      EXTEND( $\mathcal{T}, q_{rand}$ );  
5  Return  $\mathcal{T}$ 
```

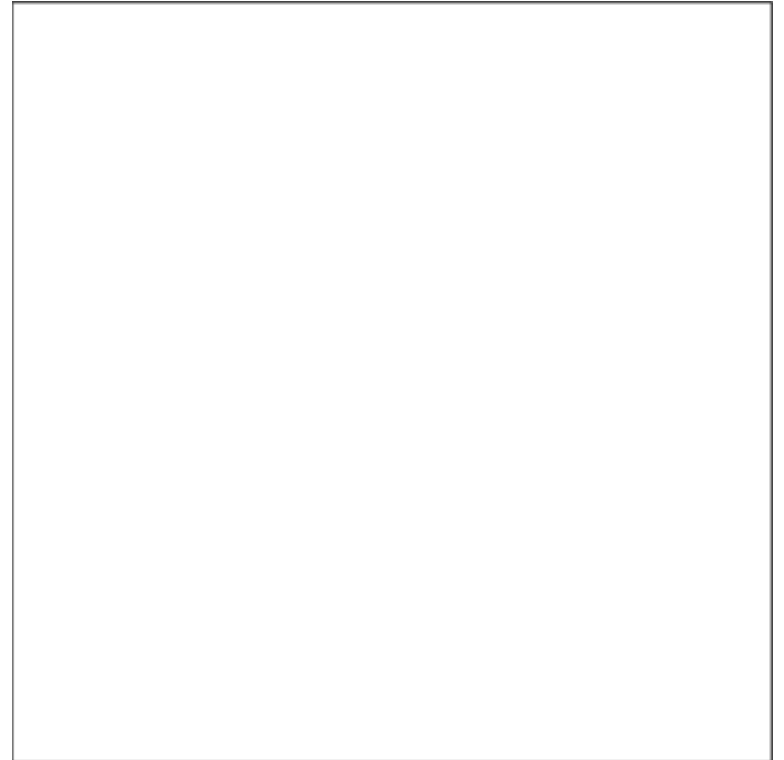
EXTEND(\mathcal{T}, q)

```
1   $q_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(q, \mathcal{T})$ ;  
2  if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then  
3       $\mathcal{T}.$ add_vertex( $q_{new}$ );  
4       $\mathcal{T}.$ add_edge( $q_{near}, q_{new}$ );  
5      if  $q_{new} = q$  then  
6          Return Reached;  
7      else  
8          Return Advanced;  
9  Return Trapped;
```

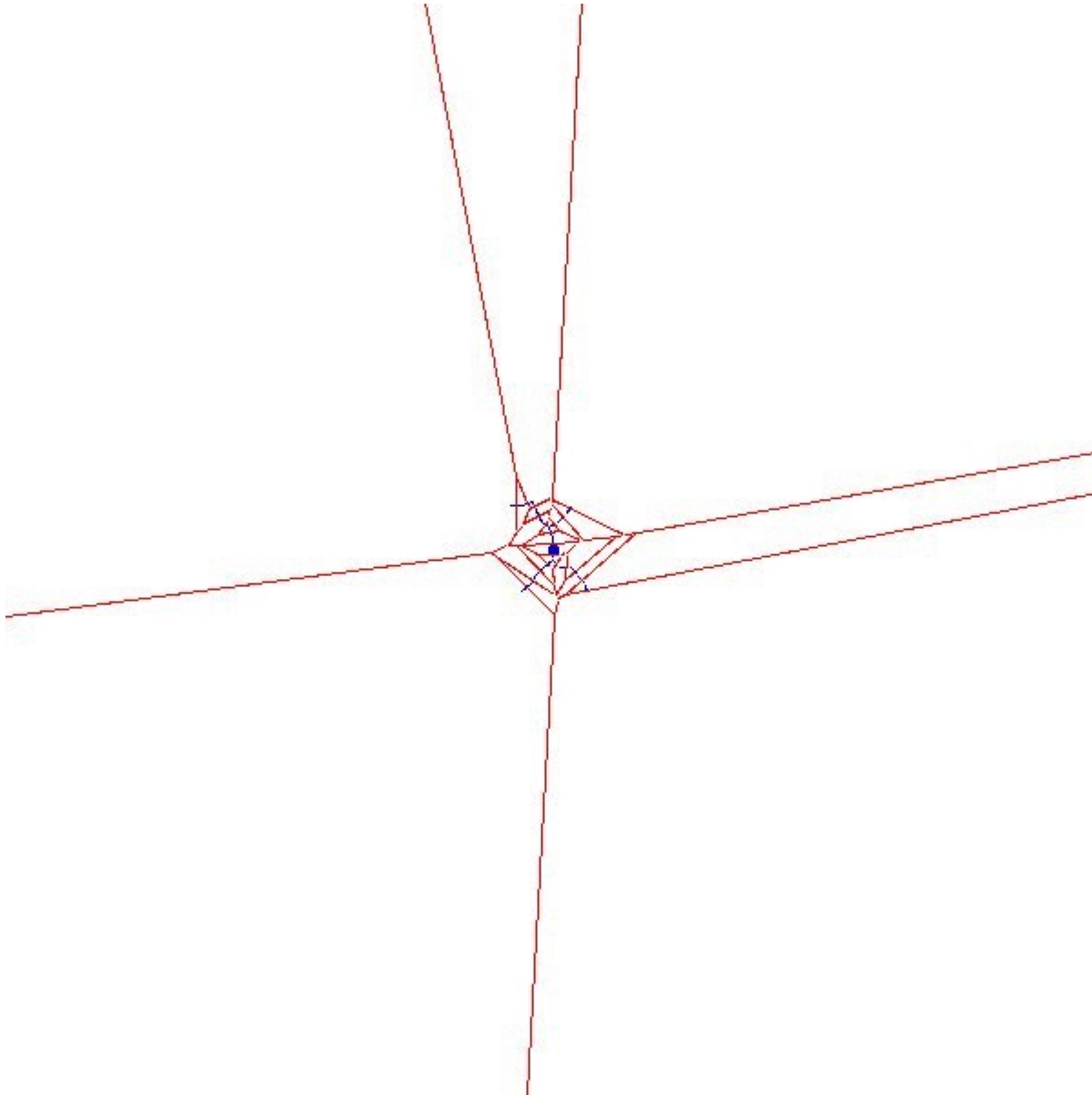


Rapidly-Exploring Random Trees

- ▶ Sampling based - no fixed grid needed
- ▶ Explores free space rapidly (Voronoi bias)
- ▶ Can handle differential constraints
- ▶ Probabilistic complete



Voronoi Bias



Exploration versus Exploitation

Exploration seeks **understanding of the state space**, irrespective of a particular task.

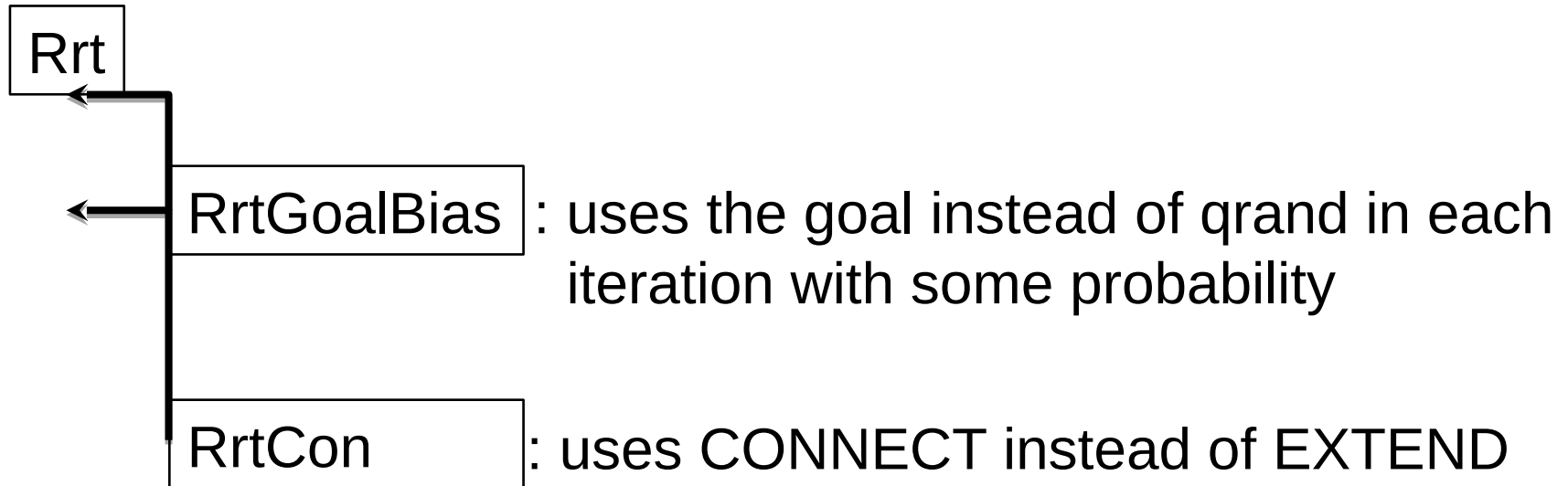
In motion planning, the process **exploration** seeks to understand the connectivity of the configuration space, irrespective of solving a particular motion planning problem.

Guided exploration seeks **efficient** understanding of the state space, irrespective a particular task, by **leveraging available information**.

Exploitation strives to **accomplish a particular task** as **efficiently as possible** by **leveraging available information**.

In motion planning, **exploitation** seeks a valid path for a **particular task**, based on available information.

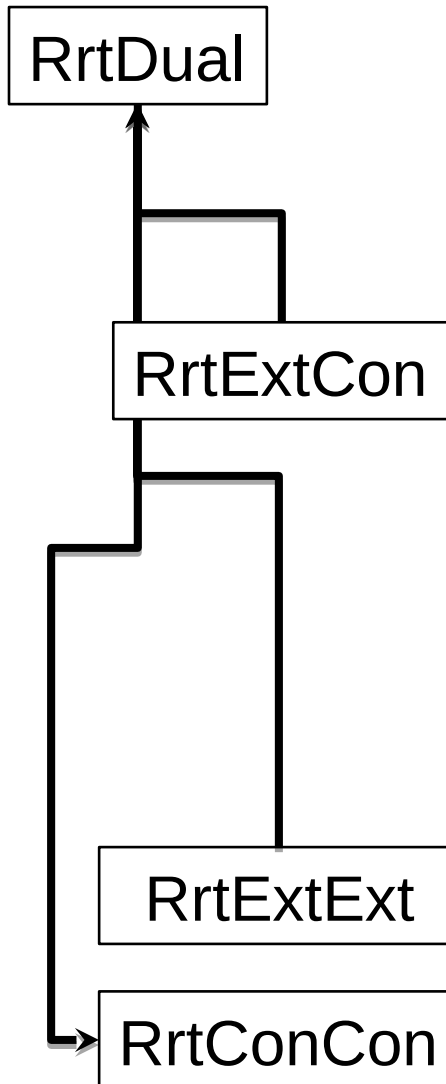
RrtGoalBias/RrtCon



CONNECT(\mathcal{T}, q)

```
1  repeat
2       $S \leftarrow \text{EXTEND}(\mathcal{T}, q)$ ;
3  until not ( $S = \text{Advanced}$ )
4  Return  $S$ ;
```

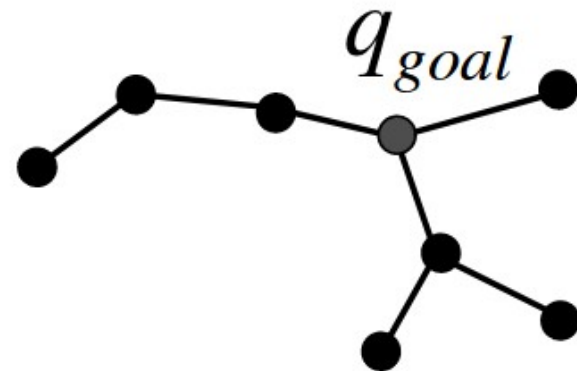
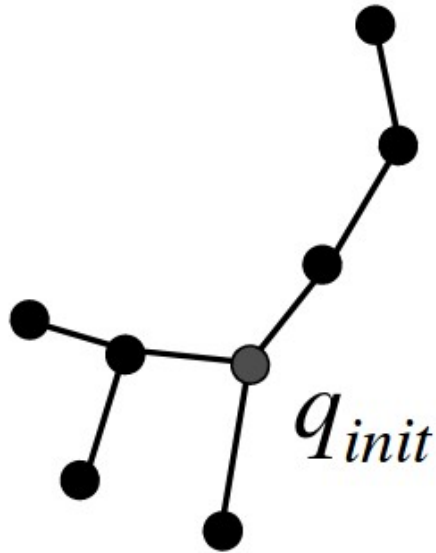
Using two trees



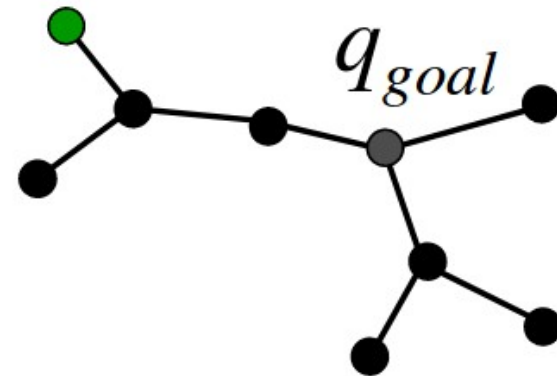
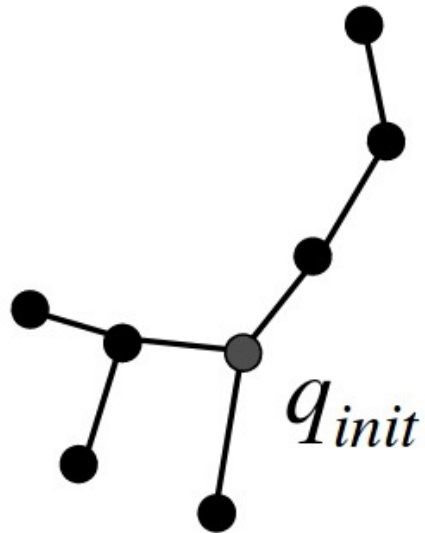
RRT_BIDIRECTIONAL(x_{init}, x_{goal})

```
1   $\mathcal{T}_a.\text{init}(x_{init}); \mathcal{T}_b.\text{init}(x_{goal});$   
2  for  $k = 1$  to  $K$  do  
3       $x_{rand} \leftarrow \text{RANDOM\_STATE}();$   
4      if not ( $\text{EXTEND}(\mathcal{T}_a, x_{rand}) = \text{Trapped}$ ) then  
5          if ( $\text{EXTEND}(\mathcal{T}_b, x_{new}) = \text{Reached}$ ) then  
6              Return  $\text{PATH}(\mathcal{T}_a, \mathcal{T}_b);$   
7       $\text{SWAP}(\mathcal{T}_a, \mathcal{T}_b);$   
8  Return Failure
```

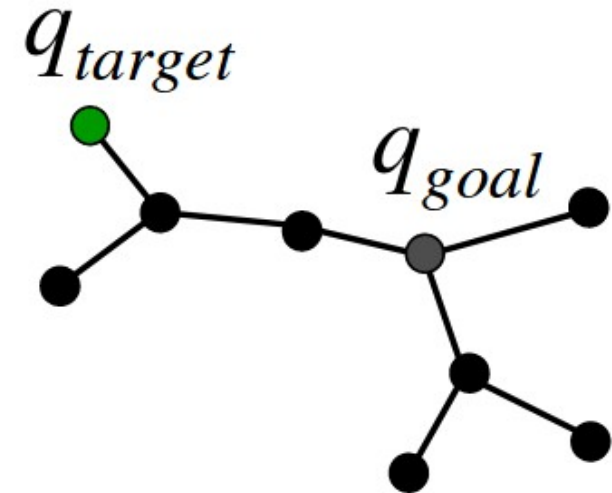
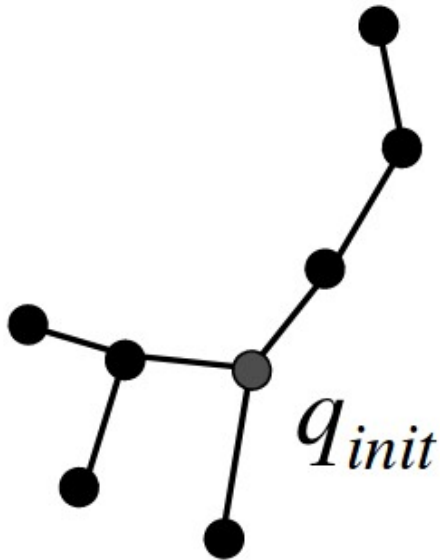
Bi-directional RRT Connect (RRTConCon)



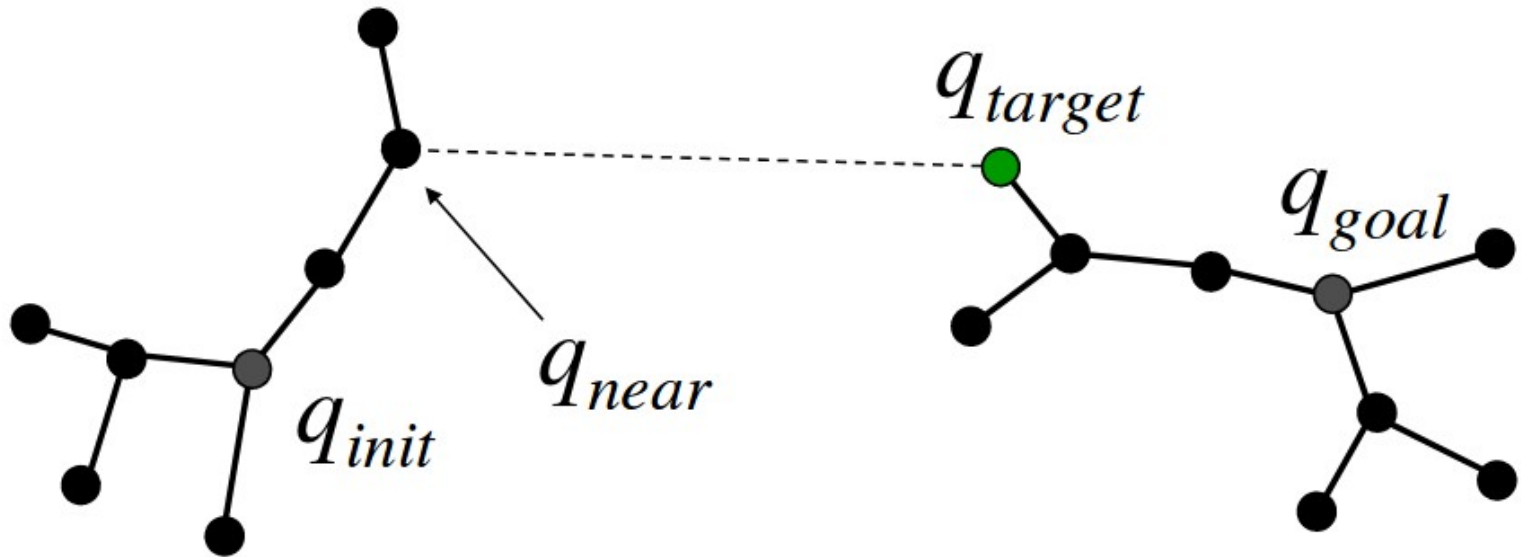
Bi-directional RRT Connect



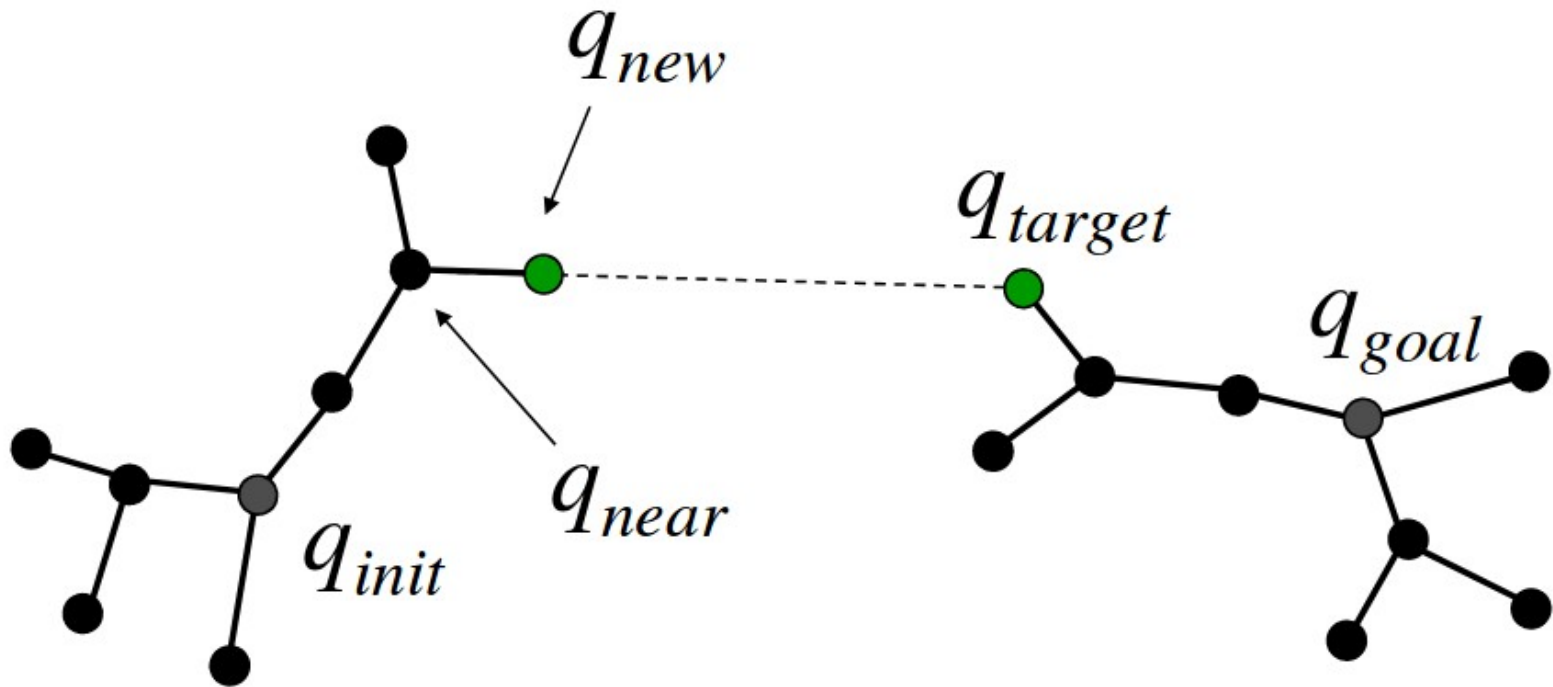
Bi-directional RRT Connect



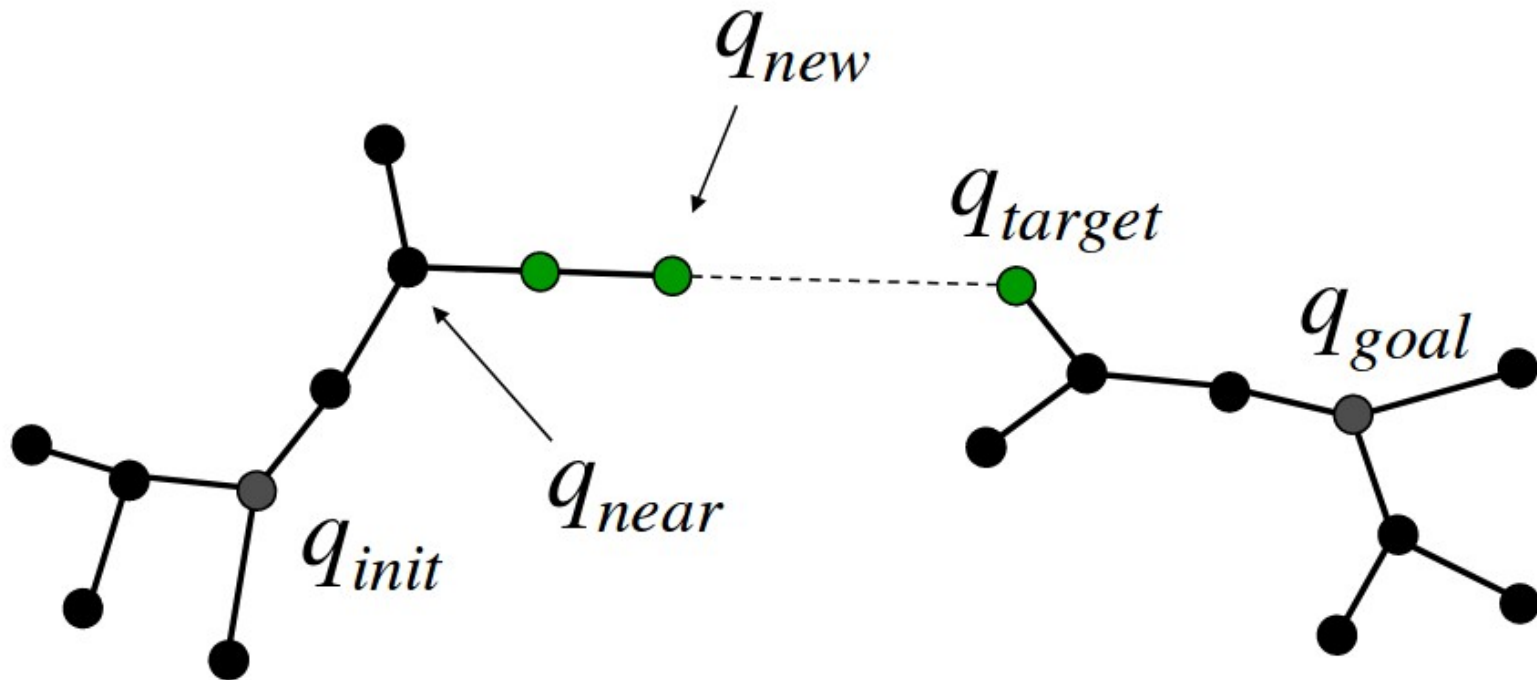
Bi-directional RRT Connect



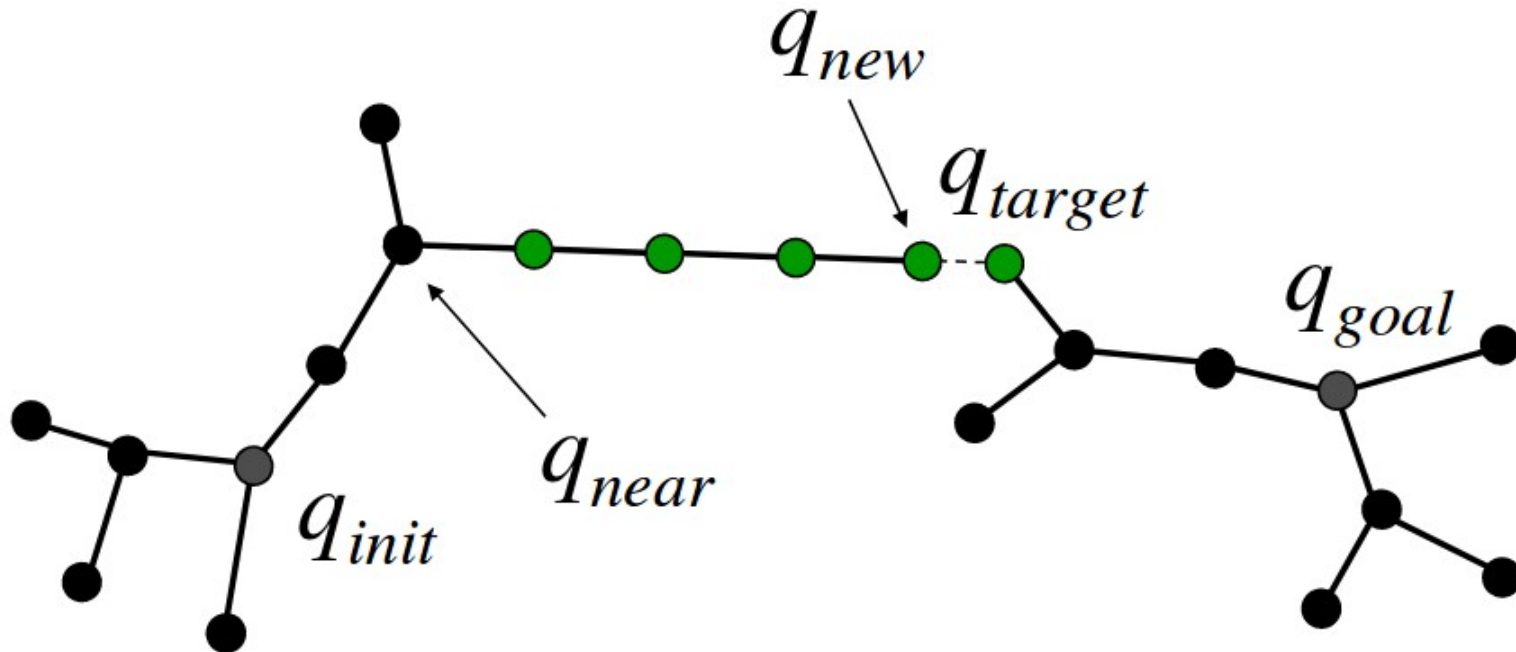
Bi-directional RRT Connect



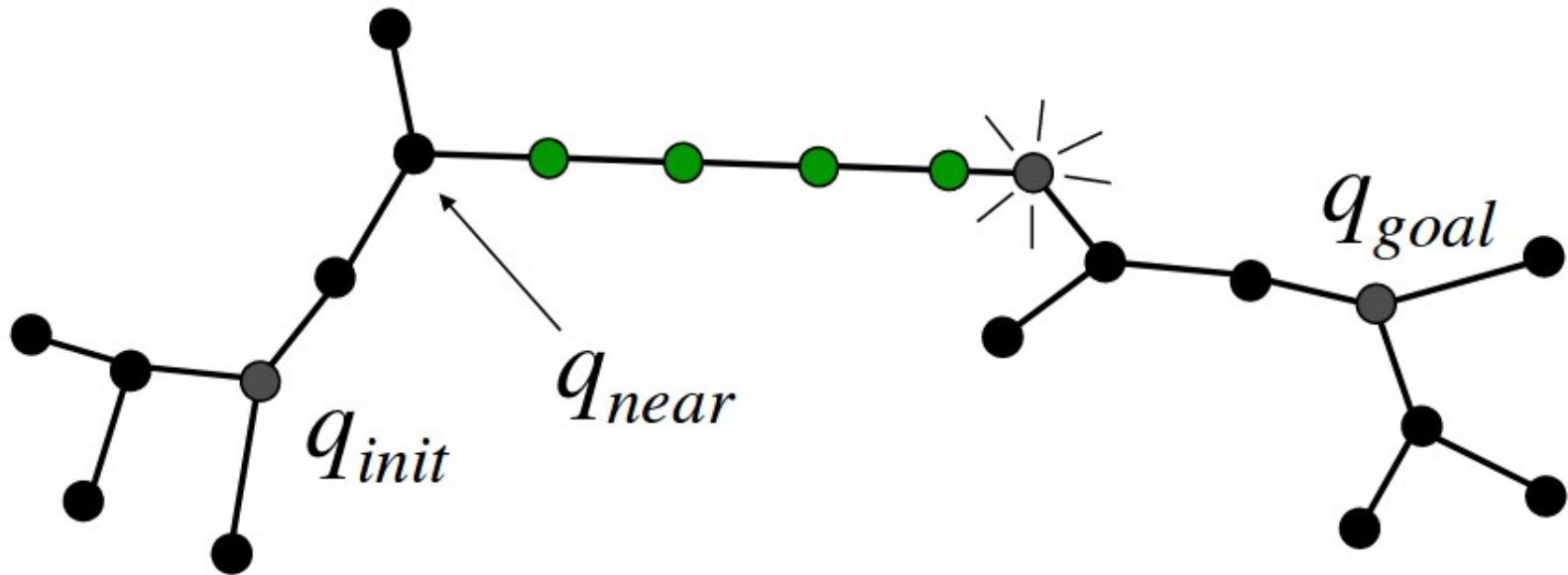
Bi-directional RRT Connect



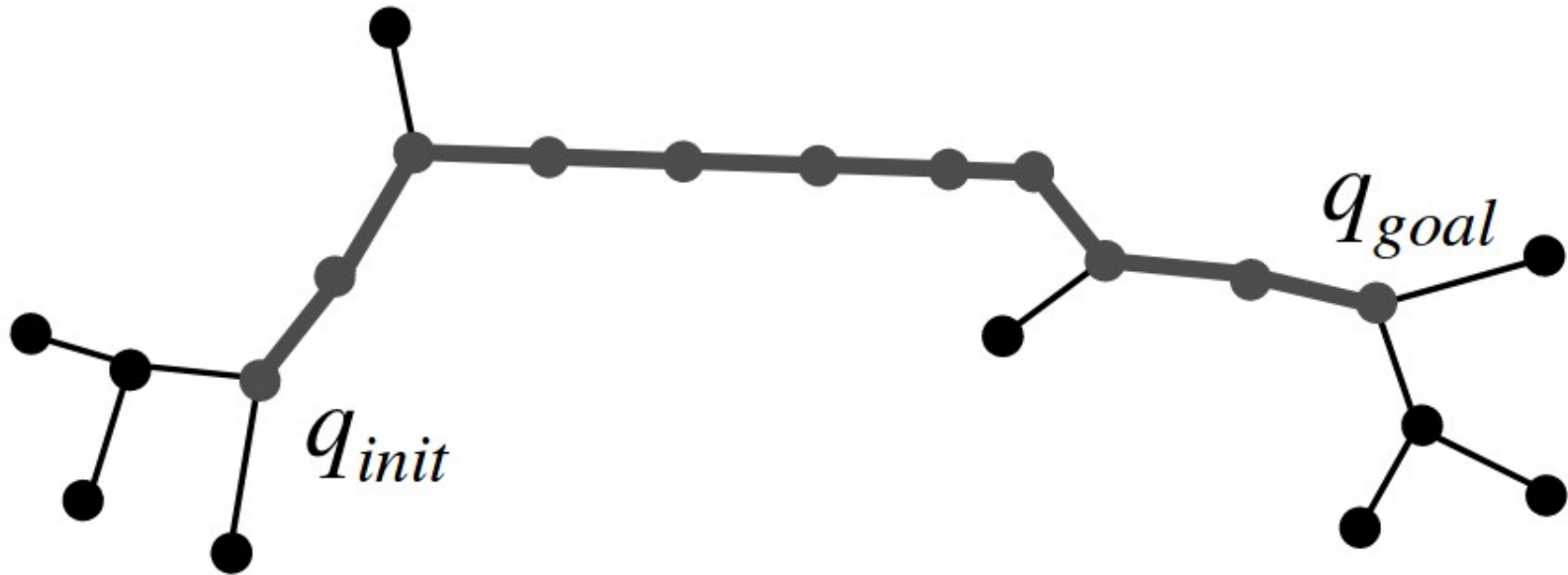
Bi-directional RRT Connect



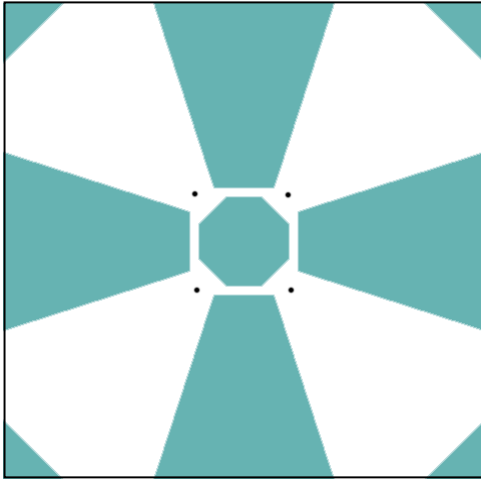
Bi-directional RRT Connect



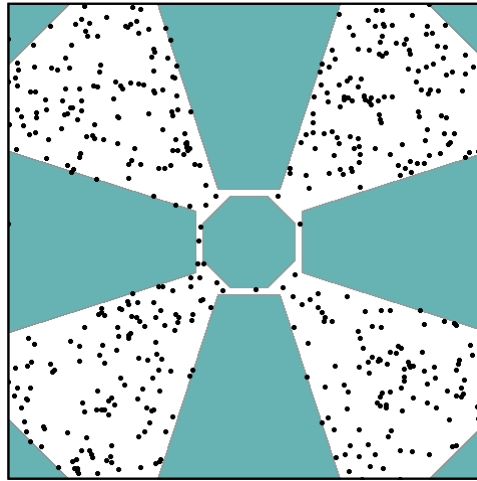
Bi-directional RRT Connect



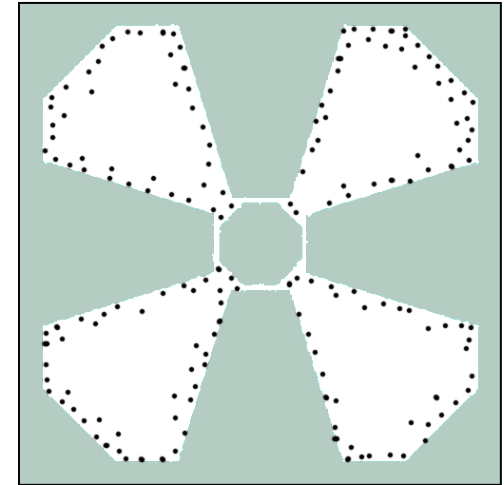
Different Sampling Strategies



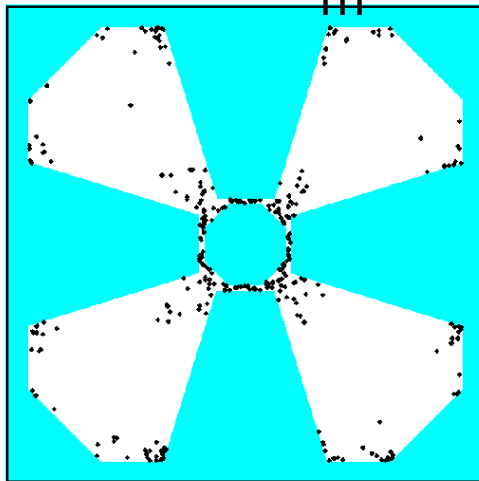
ideal



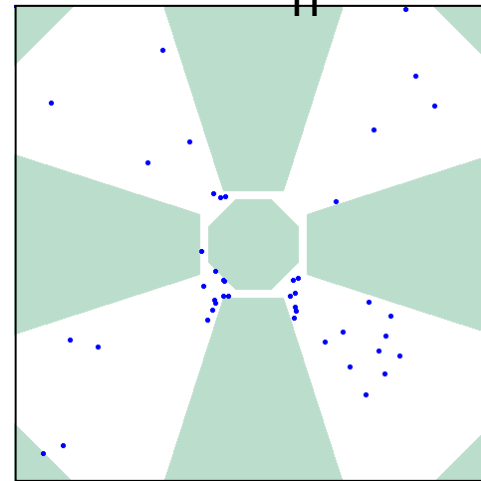
uniform



Gaussian



Bridge



Utilitarian

Robotics Library: www.roboticslibrary.org

- ▶ A framework for developing robotics applications

math Mathematics	util Timers, Threads, Mutexes, ...	xml XML Abstraction
hal Hardware Abstraction	kin DH-Kinematics	mdl Rigid Body Kinematics/Dynamics
sg Scene Graph Abstraction	plan Motion Planning	ctrl Operational Space Control



- ▶ Pro: All models we need are already included
- ▶ Con: only sparse documentation
(there are doxygen docs)

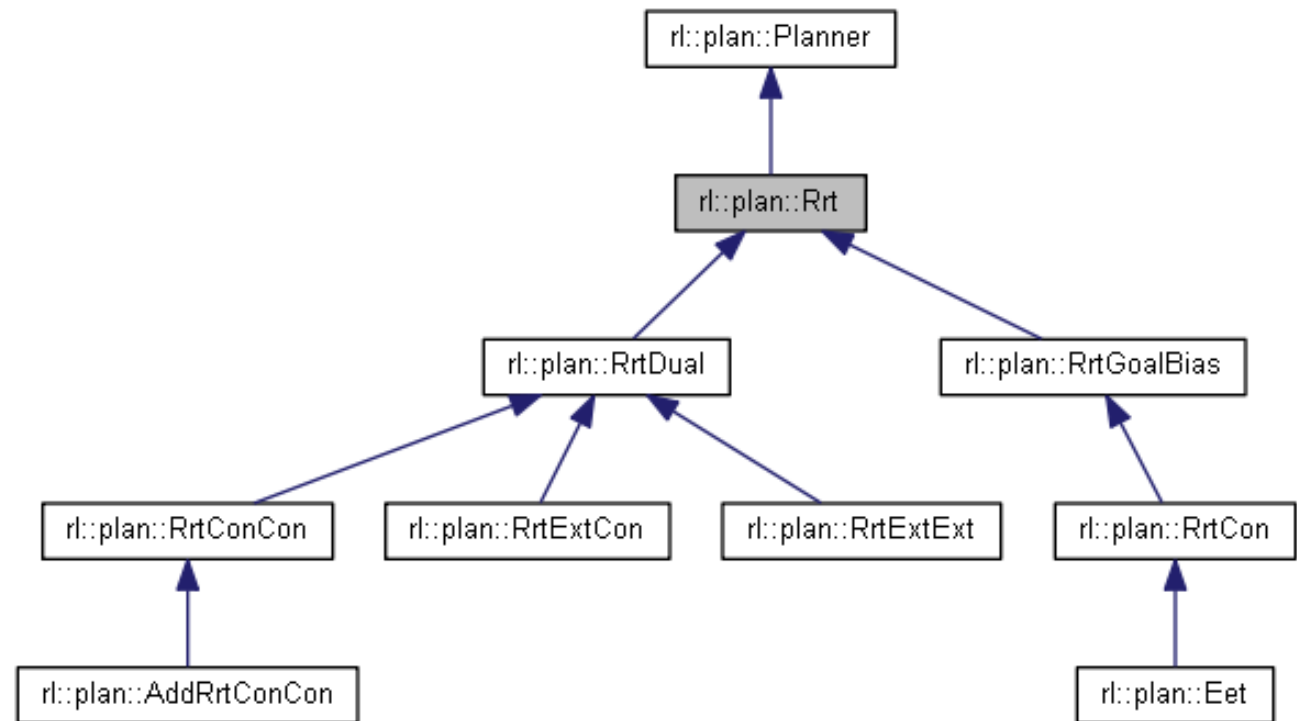
rl::plan::Rrt Class Reference

[Classes](#) | [Public Member Functions](#) | [Public Attributes](#) | [Protected Member Functions](#) | [Static Protected Member Functions](#) | [Protected Attributes](#) | [List of all members](#)

Rapidly-Exploring Random Trees. [More...](#)

#include [<Rrt.h>](#)

Inheritance diagram for rl::plan::Rrt:



You can easily install RobLib on Ubuntu

► Install prerequisites

- `sudo add-apt-repository ppa:roblib/ppa`
- `sudo apt-get update`
- `sudo apt-get install librl-dev`

• Download and extract tutorialPlan.zip:

- `tar xzf tutorialPlan.zip`

► Build

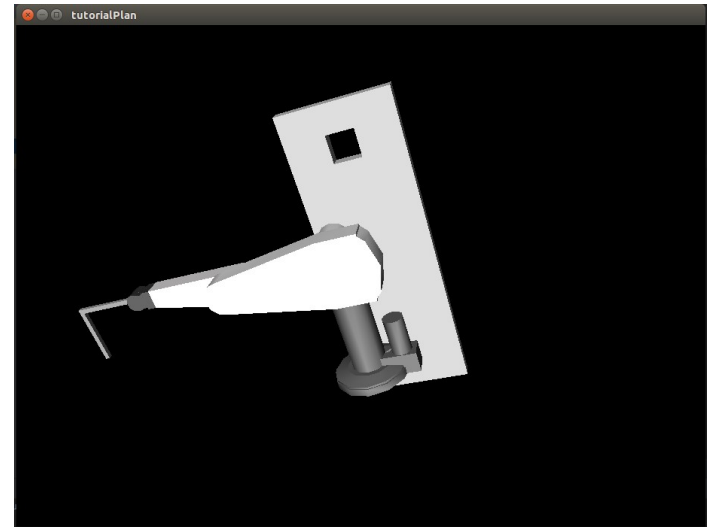
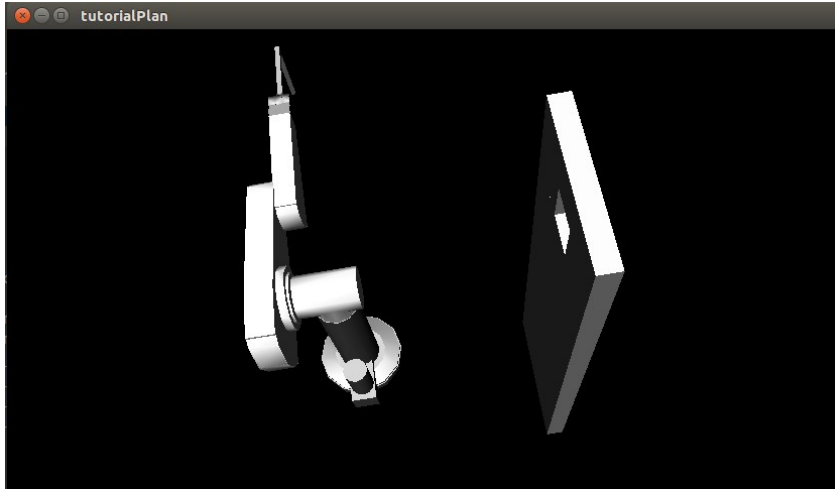
- `cd tutorialPlan/build`
- `cmake ..`
- `make`

► Test:

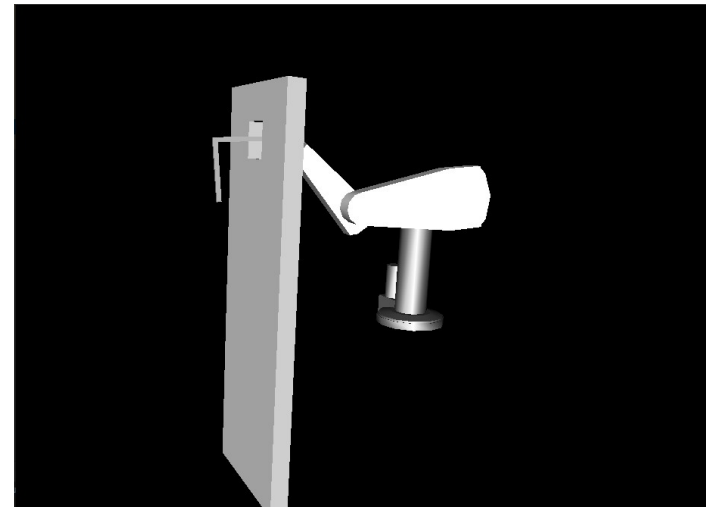
- `cd tutorialPlan/build`
- `./tutorialPlan` - Press space to start planning, F12 to reset

Puma Environment

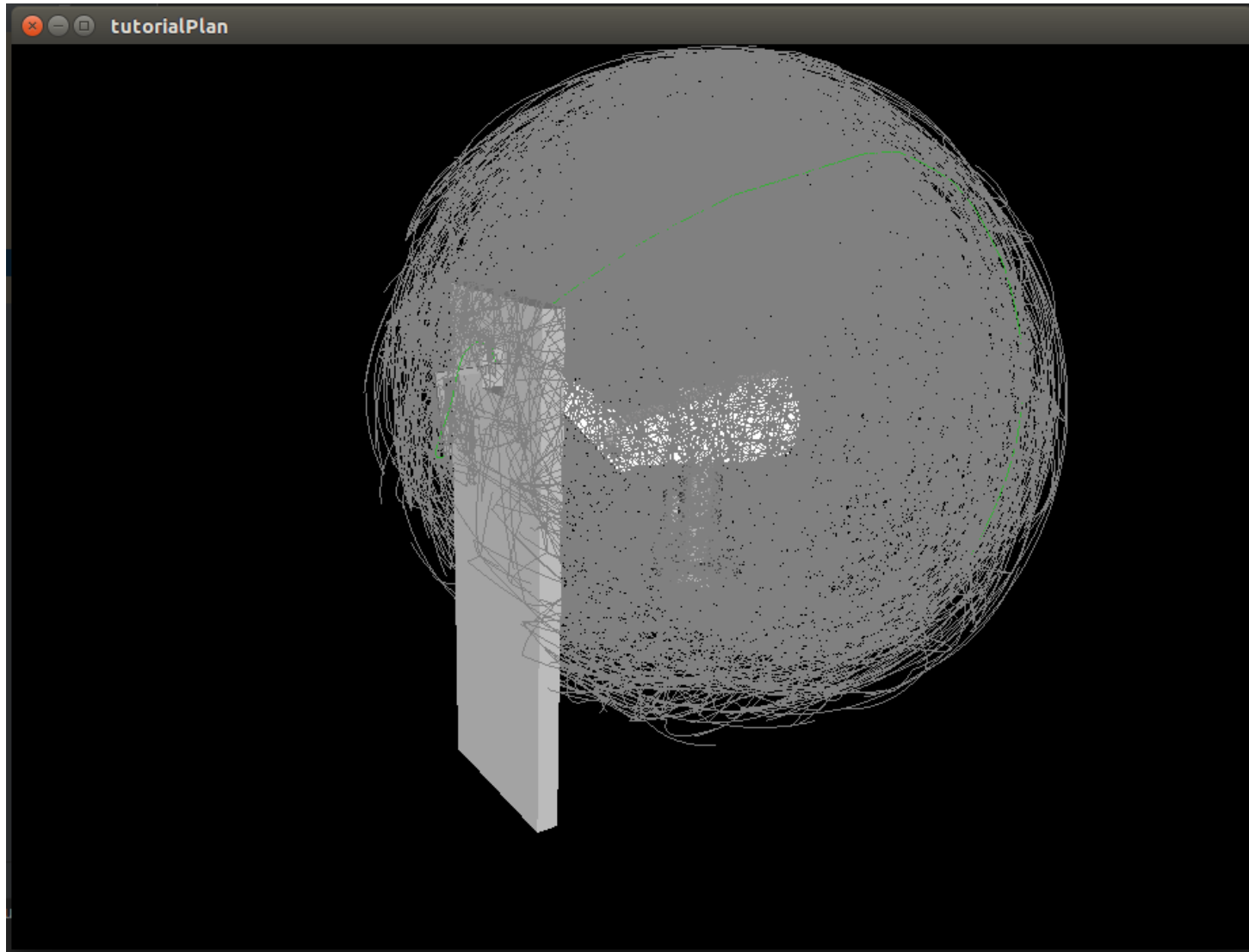
- Start Configuration



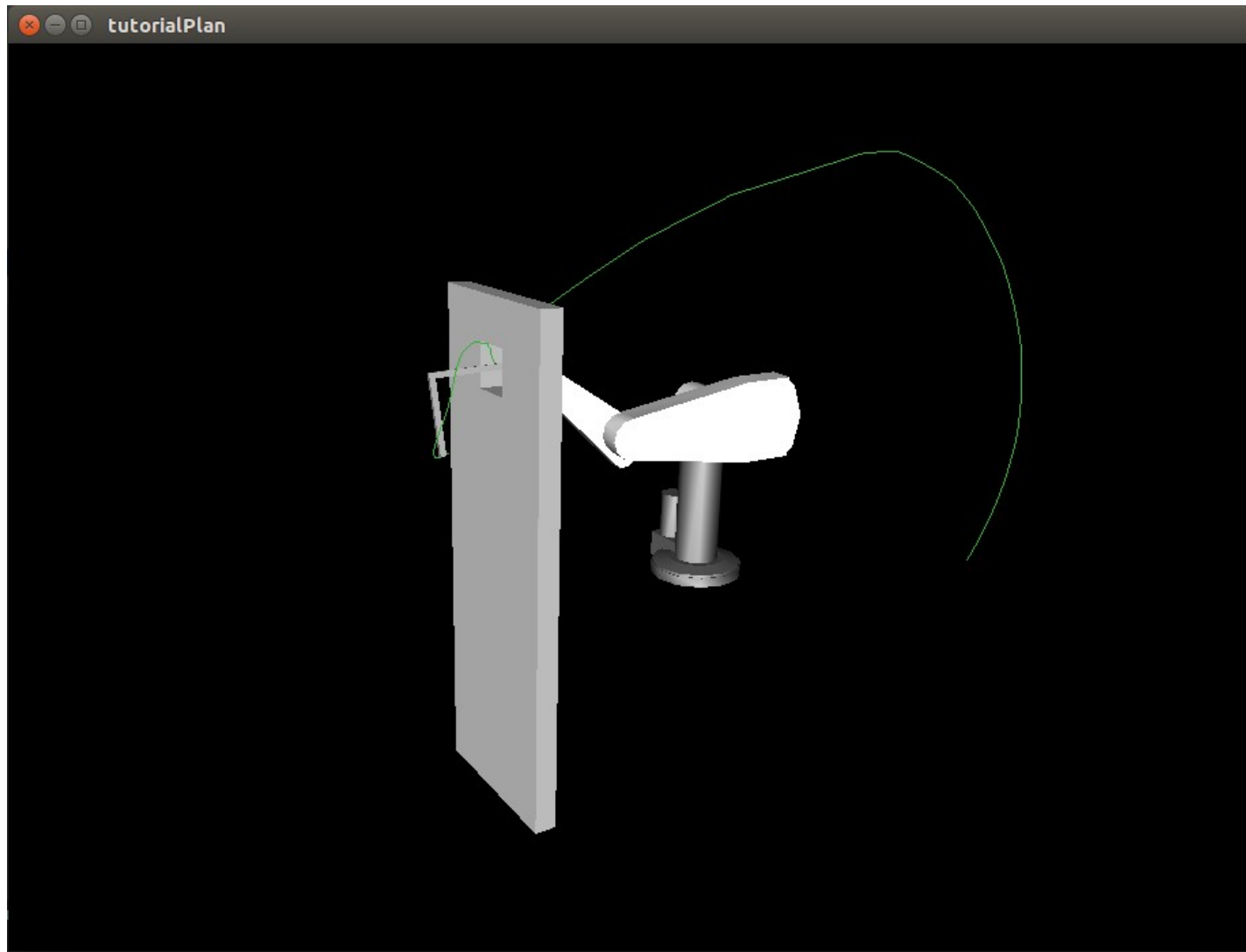
- Goal Configuration



Puma - Motion Planning Execution (1)



Puma - Motion Planning Execution (2)



Windows (VS2010) installation is also possible

► Install prerequisites

- Install Visual Studio 2010 (Express is fine)
- Install Cmake
- Install Qt <http://download.qt-project.org/archive/qt/4.8/4.8.5/qt-win-opensource-4.8.5-vs2010.exe>
- Install roblib <http://www.roboticslibrary.org/tutorials/install-windows> follow all instructions on the page!
- Download and extract tutorialPlan.zip:

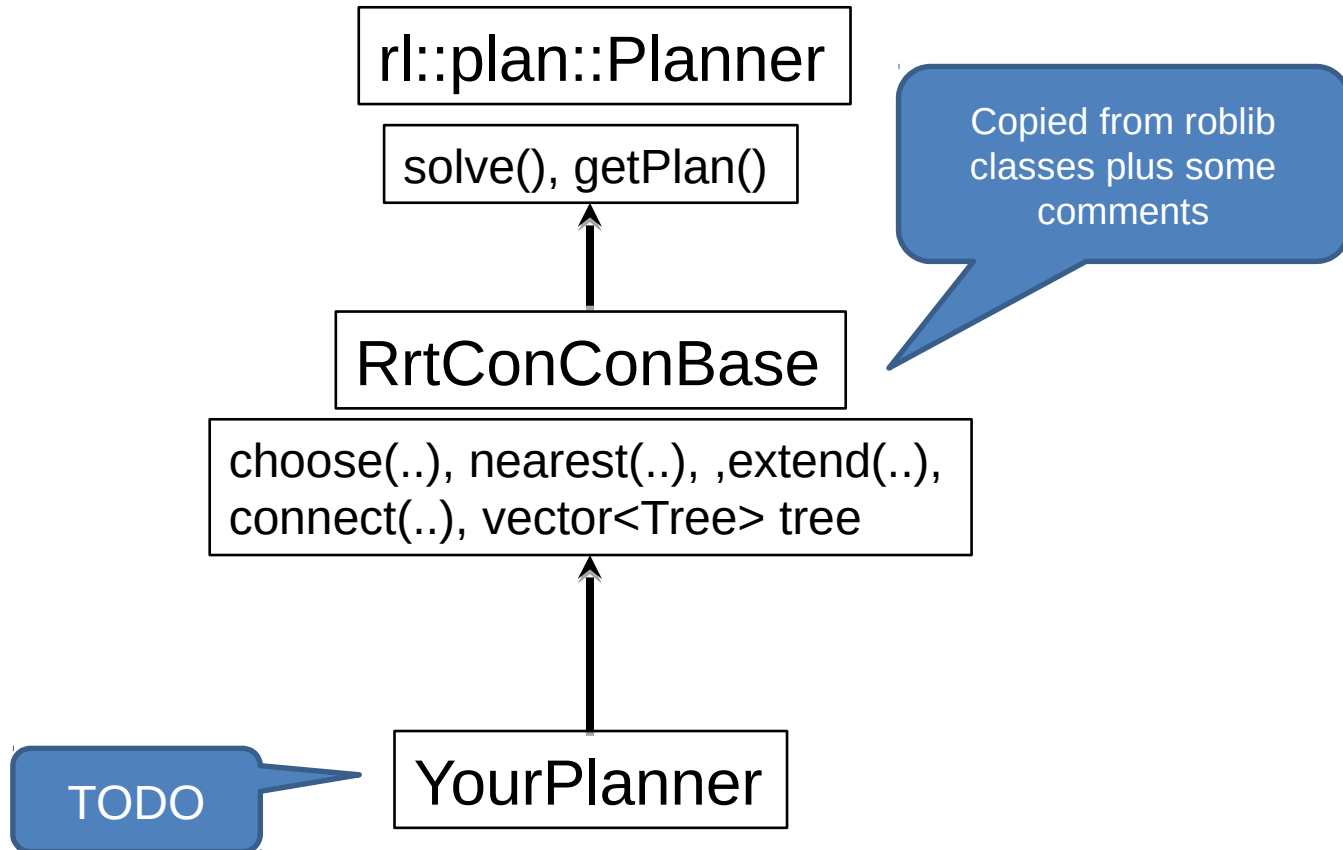
► Build:

- Open tutorialPlan/CMakeLists.txt in CMake - generate Visual Studio 10 project files in the build directory
- Open build/tutorialPlan.sln in Visual Studio 2010
- Compile RelWithDebInfo (Debug is not supported - sorry)

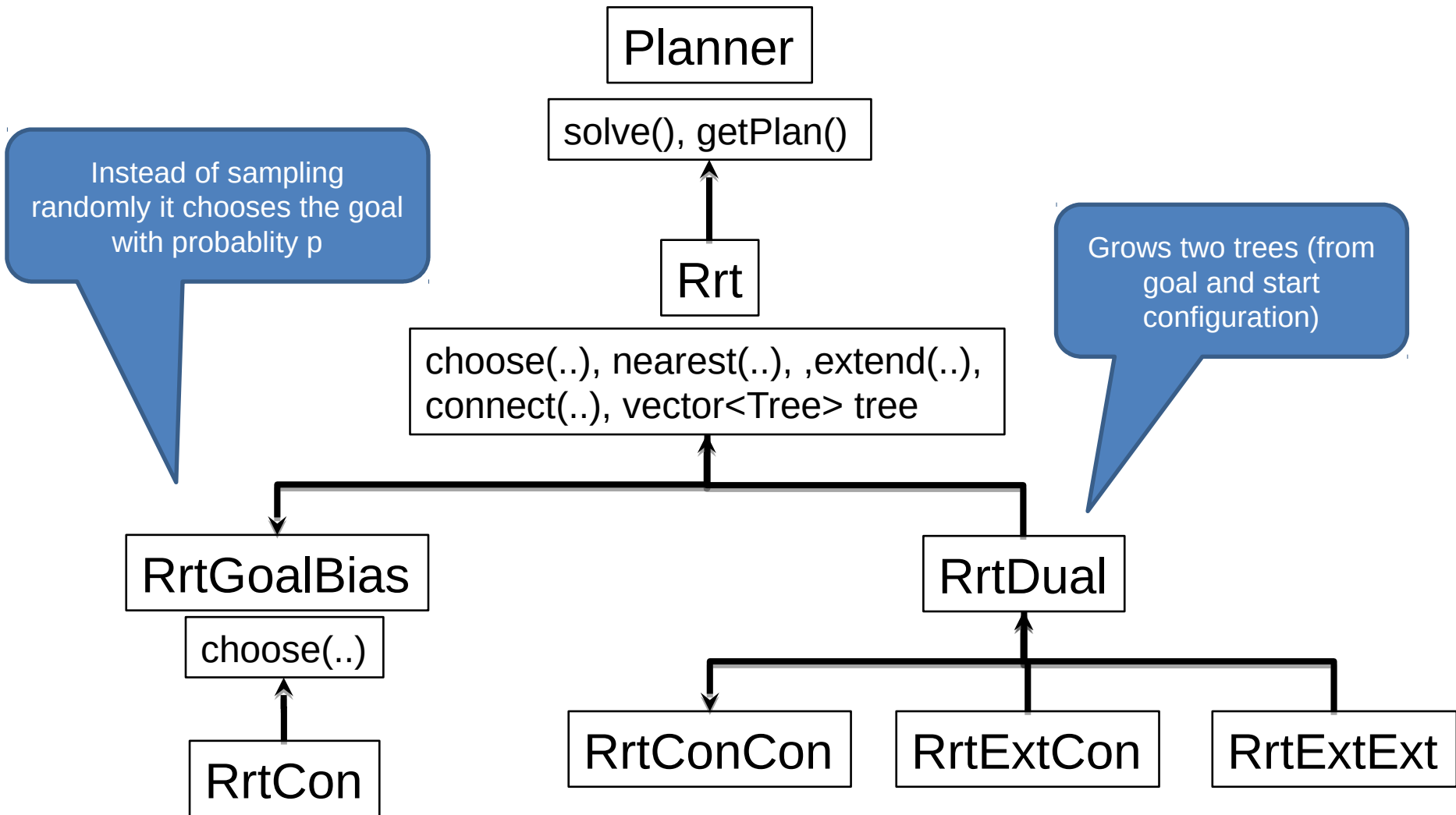
► Test:

- press F5 or open build/RelWithDebugInfo/tutorialPlan.exe - Press space to start planning, F12 to reset

Hierarchy of RRT Classes for assignment




Hierarchy of RRT Classes in rl::plan



Writing Your Planner

- ▶ Modify YourPlanner.cpp
- ▶ Inherit from RrtConConBase (or any other RRT based planner in rl::plan)
- ▶ Set parameters of your algorithm in TutorialPlanSystem.cpp
- ▶ Implement better choose(), connect(), extend(), solve()
- ▶ You can add parameters for vertices in RrtConConBase::VertexBundle

Evaluating your algorithm

- ▶ Every time you run `./tutorialPlan` an entry is added to `benchmark.csv`
- ▶ This entry shows: computation time needed to solve the problem, number of nodes and edges created, number of queries needed to construct them, etc.
- ▶ Take a look into  `~/roblib/examples/tutorialPlan/tutorialPlanSystem.cpp` to see the details and/or to edit them
- ▶ Run your algorithm at least ten times! – Change start and goal!

Writing the Documentation

- ▶ Explanation of the proposed extensions with a code snippet
- ▶ Reasoning about extensions that have not improved the runtime
- ▶ Performance evaluation of your final algorithm
- ▶ (Optional) Extra points can be gained by performing an ablation study of your extensions with regard to the runtime



References

- ▶ <http://roboticslibrary.org>
- ▶ Robotics library examples:
<https://github.com/roboticslibrary/rl-examples>
- ▶ Introduction to RRTs: msl.cs.uiuc.edu/rrt/
- ▶ Planning Book: Sections in 5.5 and 14.4 in
planning.cs.uiuc.edu
- ▶ RRTExtCon:
<http://msl.cs.uiuc.edu/~lavalle/papers/KufLav00.pdf>
- ▶ Avoid exhausted Nodes:
<http://homepages.laas.fr/jcortes/Papers/icra07paper.pdf>
- ▶ Avoid Voronoi bias:
<http://ieeexplore.ieee.org/iel5/10495/33250/01570709.pdf>



**WRITE PLANNING &
CONTROL SOFTWARE**



**CONDUCT
EXPERIMENTS**



**DEVELOP HARDWARE
DRIVERS**

**CREATE
SOFT
HANDS**

STUDENT ASSISTANTS WANTED

robotics.tu-berlin.de/menue/open_positions



**ROBOTICS AND BIOLOGY
LABORATORY**

Skills:

- C++ / Python
- Linux
- English