

Sample-based path finding

Lecture 3



主讲人 Chaoqun Wang

Ph.D.

Postdoc Fellow

The Chinese University of Hong Kong

Email: zychaoqun@gmail.com

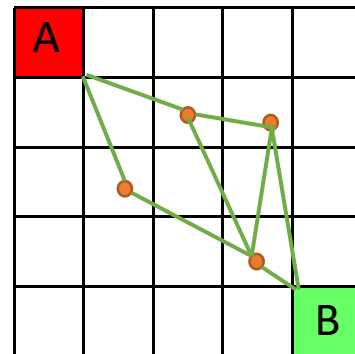
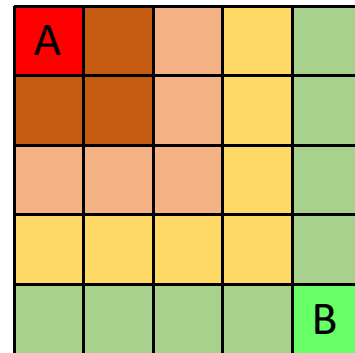




Preliminaries

Sampling Based-Planners

- Do not attempt to explicitly construct the C-Space and its boundaries
- Simply need to know if a single robot configuration is in collision
- Exploits simple tests for collision with full knowledge of the space
- Collision detection is a separate module- can be tailored to the application
- As collision detection improves, so do these algorithms
- Different approaches for single-query and multi-query requests





Preliminaries

Notion of Completeness in Planning

- Complete Planner: always answers a path planning query correctly in bounded time
- Probabilistic Complete Planner: if a solution exists, planner will eventually find it, using random sampling (e.g. Monte Carlo sampling)
- Resolution Complete Planner: same as above but based on a deterministic sampling (e.g. sampling on a fixed grid).



Content



1. Probabilistic Road Map



2. Rapidly-exploring Random Tree



3. Optimal sampling-based path planning methods



4. Advanced path planning methods



5. Implementation



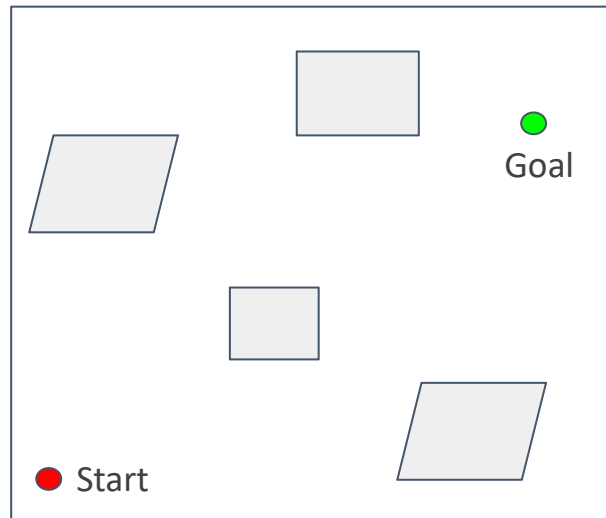
Probabilistic Road Map



Probabilistic Road Map

What is PRM?

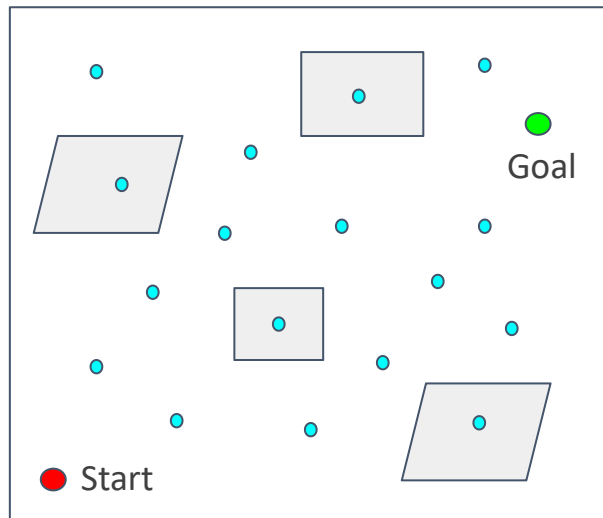
- A graph structure
- Divide planning into two phases:
 - Learning phase:
 - Query phase:
- Checking sampled configurations and connections between samples for collision can be done efficiently.
- A relatively small number of milestones and local paths are sufficient to capture the connectivity of the free space





Probabilistic Road Map

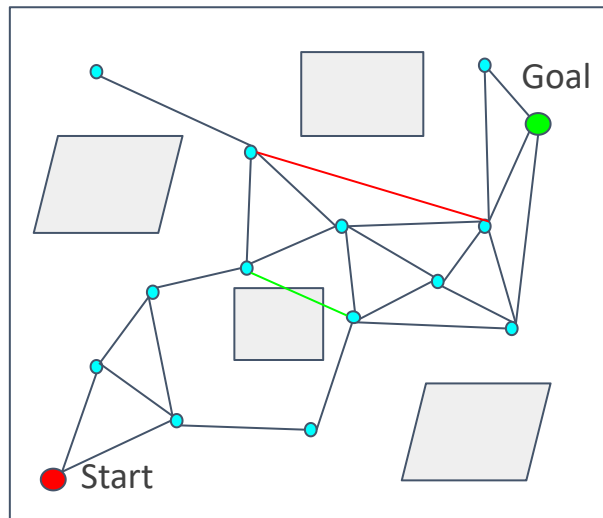
- Learning phase:
 - Sample N points in C-space
 - Delete points that are not collision-free
- Detect the c-space with random points





Probabilistic Road Map

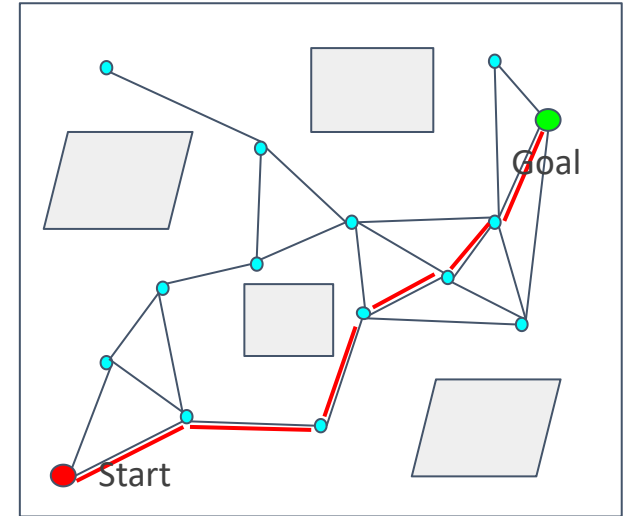
- Learning phase:
 - Connect to nearest points and get collision-free segments.
 - Delete segments that are not collision free.





Probabilistic Road Map

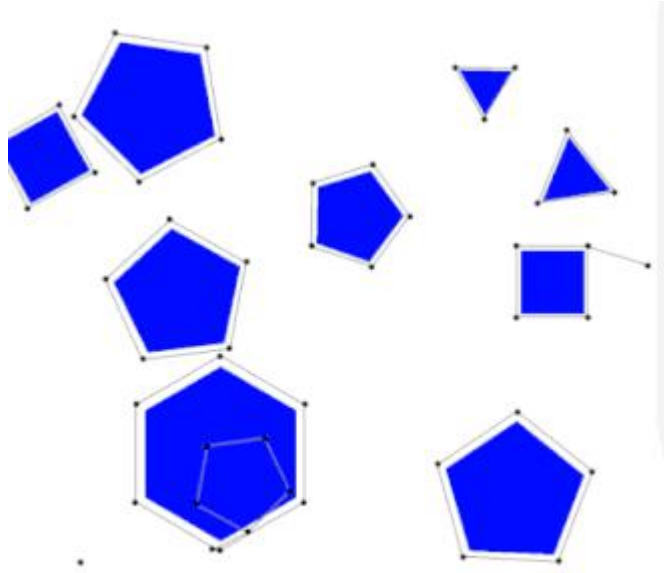
- Query phase:
 - Search on the road map to find a path from the start to the goal (using Dijkstra's algorithm or the A* algorithm).
 - Road map is now similar with the grid map (or a simplified grid map)



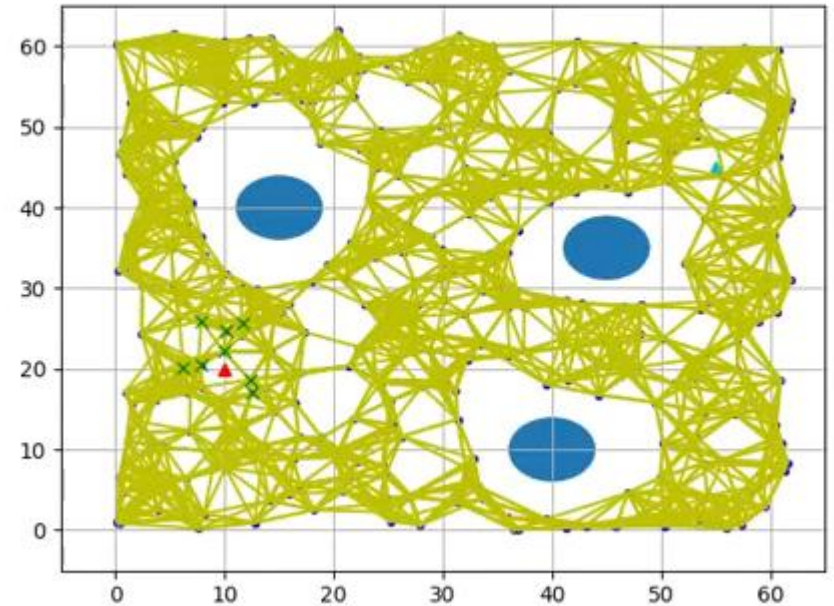


Probabilistic Road Map

Learning phase[1]



Query phase[2]



[1] https://en.wikipedia.org/wiki/Probabilistic_roadmap

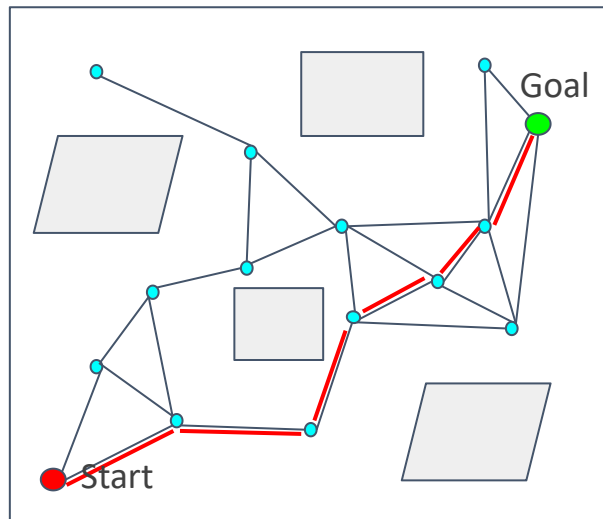
[2] https://www.youtube.com/watch?v=8Dln3sS_p4Q



Probabilistic Road Map

Pros and Cons

- Pros
 - Probabilistically complete
- Cons
 - Required to solve 2 point boundary value problem
 - Build graph over state space but no particular focus on generating a path
 - Not efficient

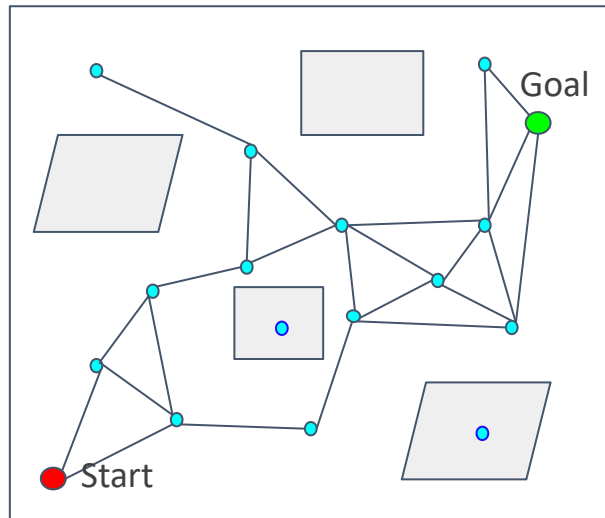




Probabilistic Road Map

Note: towards improving efficiency

- Lazy collision-checking
 - Collision-checking process is time-consuming, especially in complex or high-dimensional environments.
 - Sample points and generate segments without considering the collision (**Lazy**).

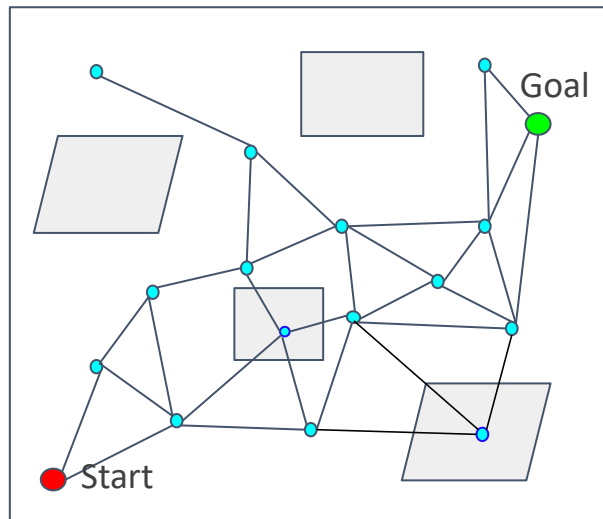




Probabilistic Road Map

Lazy collision-checking

Sample points and generate segments without considering the collision (**Lazy**).



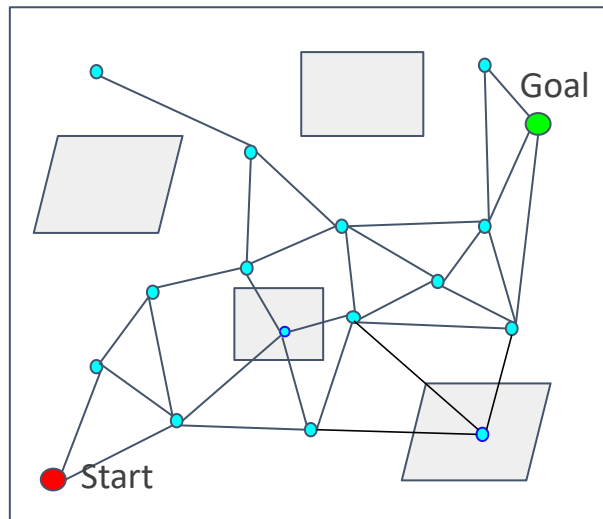


Probabilistic Road Map

Lazy collision-checking

Collision-checking if necessary:

Find a path on the road map generated without collision-checking



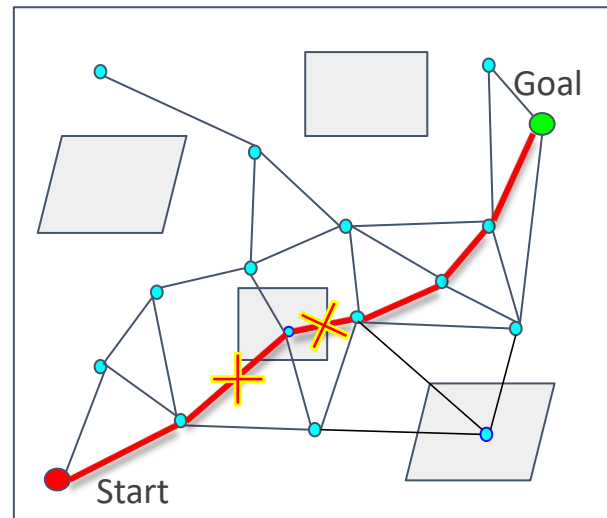


Probabilistic Road Map

Lazy collision-checking

Collision-checking if necessary:

Delete the corresponding edges and nodes if the path is not collision free.



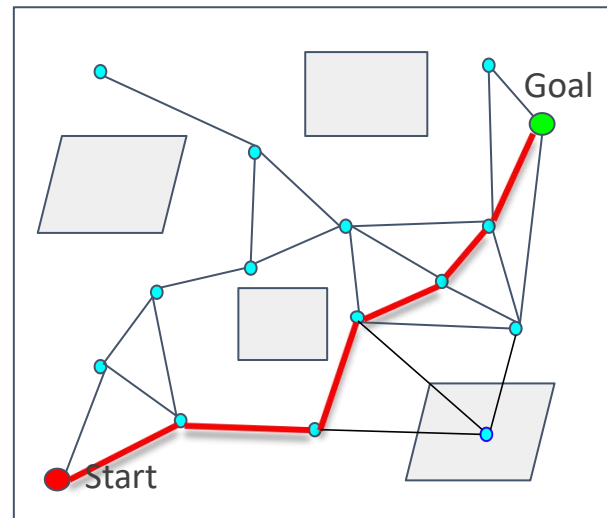


Probabilistic Road Map

Lazy collision-checking

Collision-checking if necessary:

- Delete the corresponding edges and nodes if the path is not collision free.
- Restart path finding.





Probabilistic Road Map

Note:

- Learning phase
- Query phase
- Lazy collision-checking

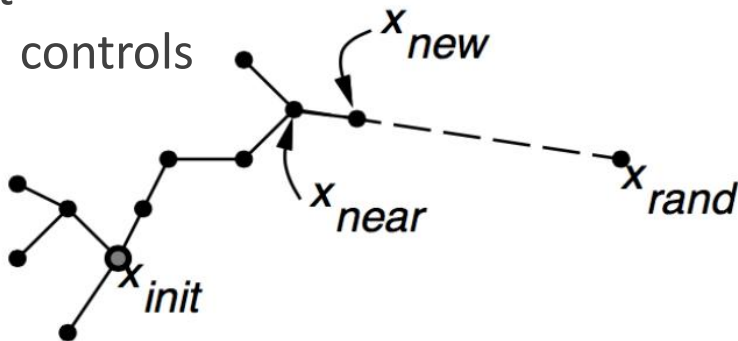


Rapidly-exploring Random Tree



Rapidly-exploring Random Trees

- Build up a tree through generating “next states” in the tree by executing random controls





Rapidly-exploring Random Trees

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init();$

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow Sample(\mathcal{M});$

$x_{near} \leftarrow Near(x_{rand}, \mathcal{T});$

$x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize);$

$E_i \leftarrow Edge(x_{new}, x_{near});$

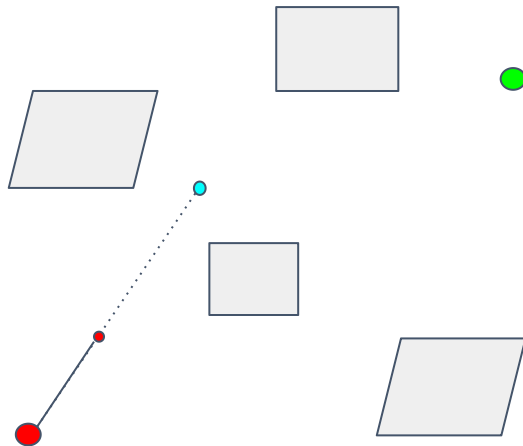
if $CollisionFree(\mathcal{M}, E_i)$ **then**

$\mathcal{T}.addNode(x_{new});$

$\mathcal{T}.addEdge(E_i);$

if $x_{new} = x_{goal}$ **then**

Success();





Rapidly-exploring Random Trees

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init();$

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow Sample(\mathcal{M});$

$x_{near} \leftarrow Near(x_{rand}, \mathcal{T});$

$x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize);$

$E_i \leftarrow Edge(x_{new}, x_{near});$

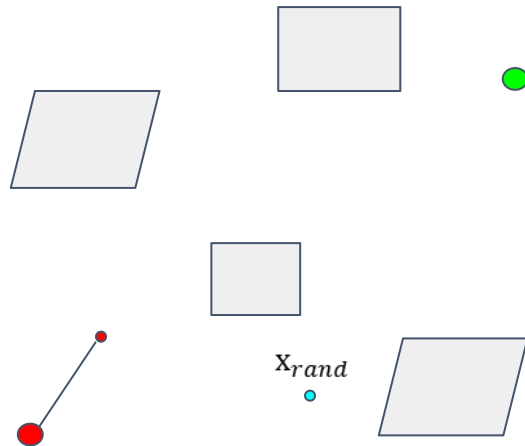
if $CollisionFree(\mathcal{M}, E_i)$ **then**

$\mathcal{T}.addNode(x_{new});$

$\mathcal{T}.addEdge(E_i);$

if $x_{new} = x_{goal}$ **then**

Success();





Rapidly-exploring Random Trees

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init();$

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow Sample(\mathcal{M});$

$x_{near} \leftarrow Near(x_{rand}, \mathcal{T});$

$x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize);$

$E_i \leftarrow Edge(x_{new}, x_{near});$

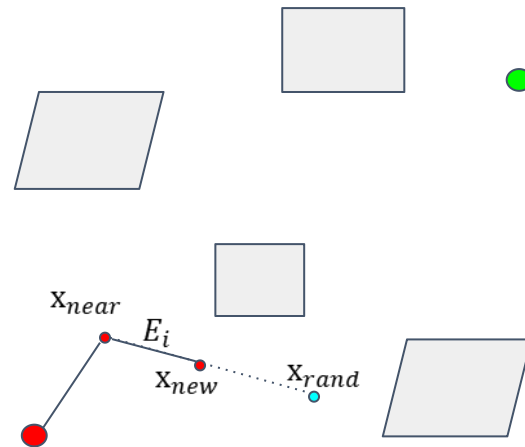
if $CollisionFree(\mathcal{M}, E_i)$ **then**

$\mathcal{T}.addNode(x_{new});$

$\mathcal{T}.addEdge(E_i);$

if $x_{new} = x_{goal}$ **then**

Success $()$;





Rapidly-exploring Random Trees

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init();$

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow Sample(\mathcal{M});$

$x_{near} \leftarrow Near(x_{rand}, \mathcal{T});$

$x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize);$

$E_i \leftarrow Edge(x_{new}, x_{near});$

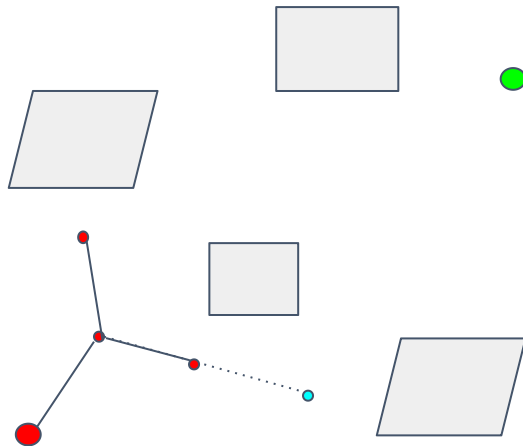
if $CollisionFree(\mathcal{M}, E_i)$ **then**

$\mathcal{T}.addNode(x_{new});$

$\mathcal{T}.addEdge(E_i);$

if $x_{new} = x_{goal}$ **then**

Success();





Rapidly-exploring Random Trees

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init();$

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow Sample(\mathcal{M});$

$x_{near} \leftarrow Near(x_{rand}, \mathcal{T});$

$x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize);$

$E_i \leftarrow Edge(x_{new}, x_{near});$

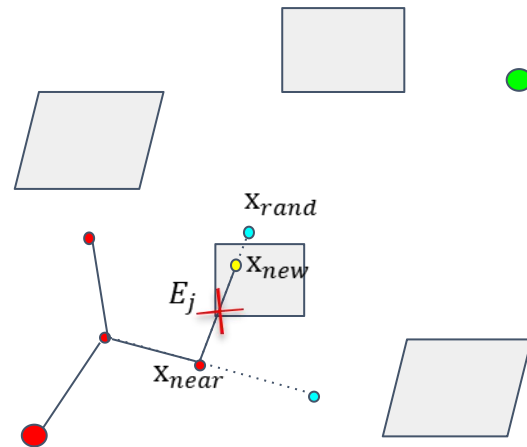
if $CollisionFree(\mathcal{M}, E_i)$ **then**

$\mathcal{T}.addNode(x_{new});$

$\mathcal{T}.addEdge(E_i);$

if $x_{new} = x_{goal}$ **then**

Success $();$





Rapidly-exploring Random Trees

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init();$

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow Sample(\mathcal{M});$

$x_{near} \leftarrow Near(x_{rand}, \mathcal{T});$

$x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize);$

$E_i \leftarrow Edge(x_{new}, x_{near});$

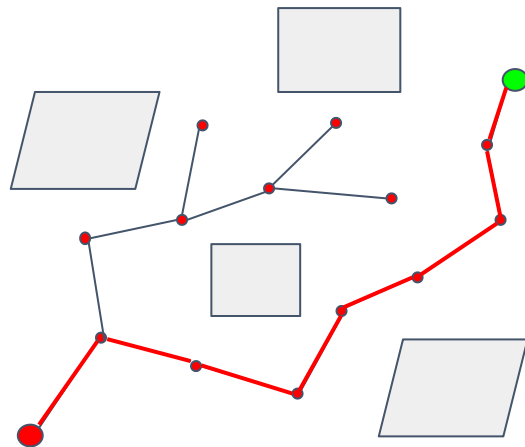
if $CollisionFree(\mathcal{M}, E_i)$ **then**

$\mathcal{T}.addNode(x_{new});$

$\mathcal{T}.addEdge(E_i);$

if $x_{new} = x_{goal}$ **then**

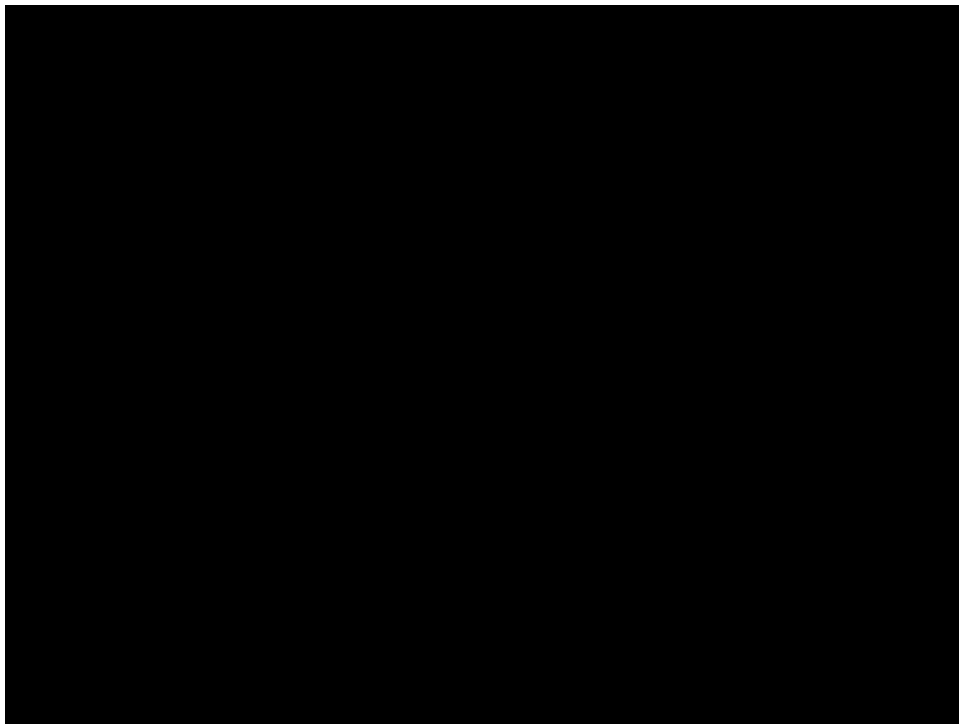
Success();





Rapidly-exploring Random Trees

Demonstration of RRT[1]

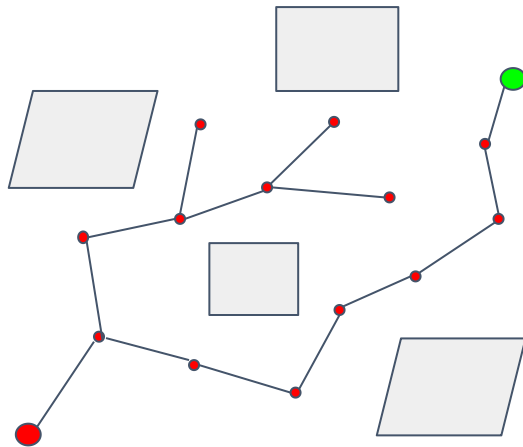


[1] https://www.youtube.com/watch?v=pOFtvZ_qVsA



Rapidly-exploring Random Trees

- **Pros:**
 - Aims to find a path from the start to the goal
 - More target-oriented than PRM
- **Cons:**
 - Not optimal solution
 - Not efficient(leave room for improvement)
 - Sample in the whole space

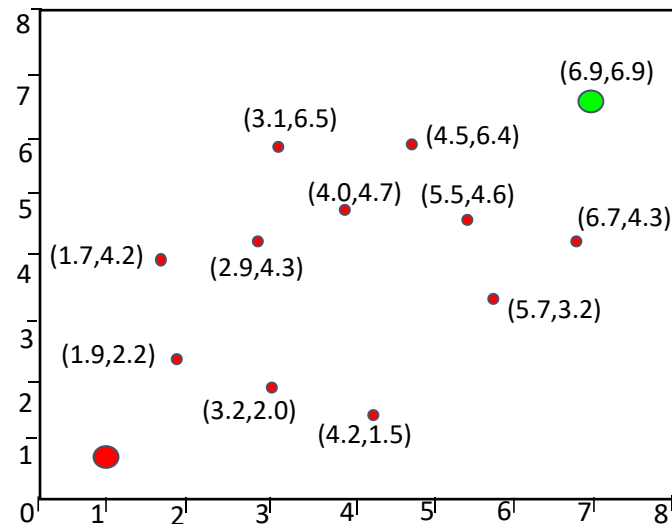
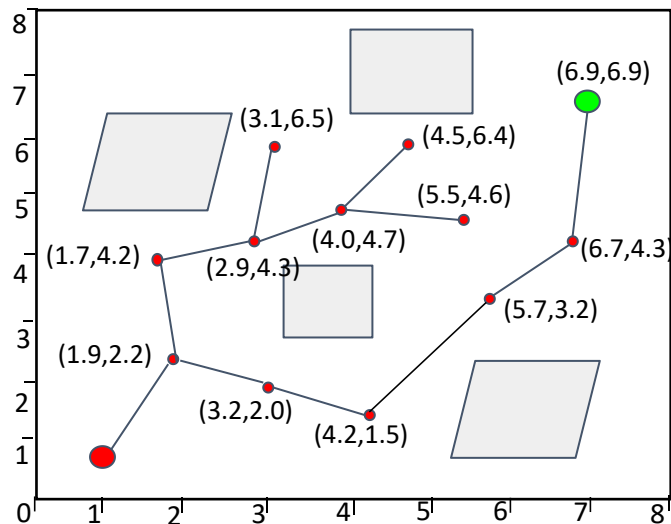




Rapidly-exploring Random Trees

Note: towards improving efficiency

Kd-tree

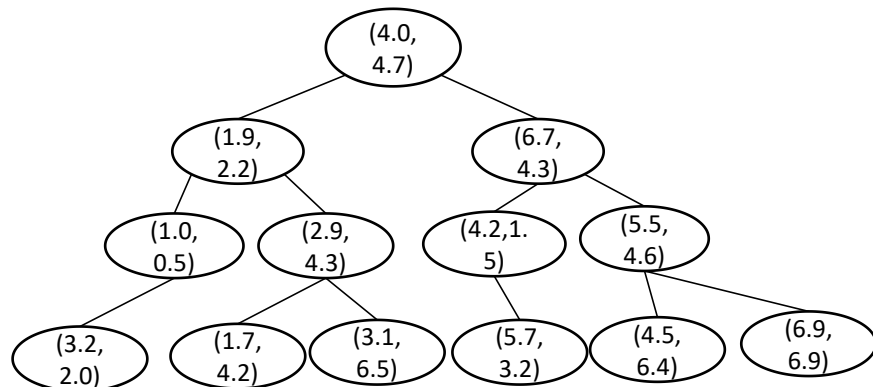
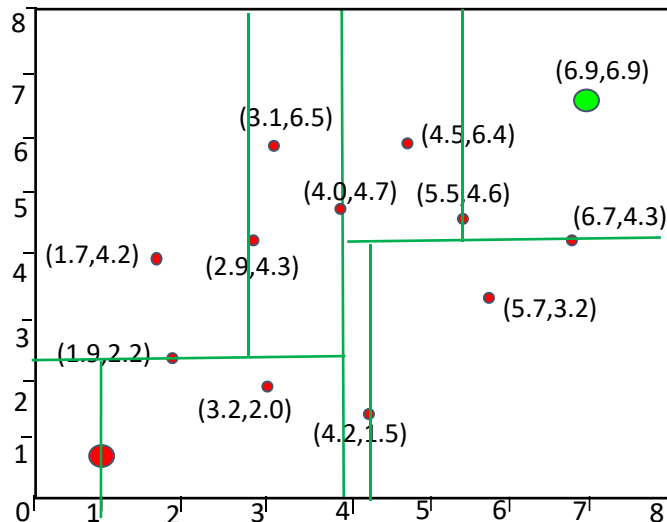




Rapidly-exploring Random Trees

Note: towards path planning efficiency

- Kd-tree



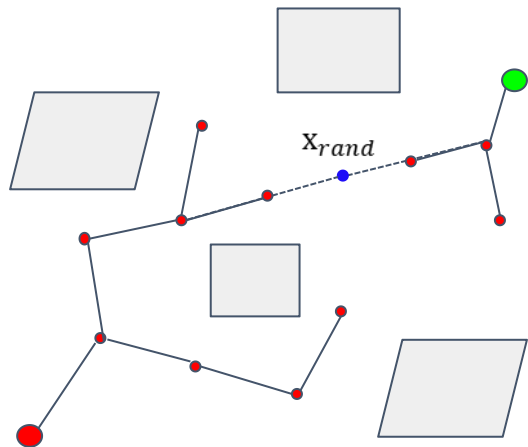
- Other variants: Spatial grid, hill climbing, etc
- 参考: <https://blog.csdn.net/junshen1314/article/details/51121582>



Rapidly-exploring Random Trees

Note: towards improving efficiency

Bidirectional RRT / RRT Connect



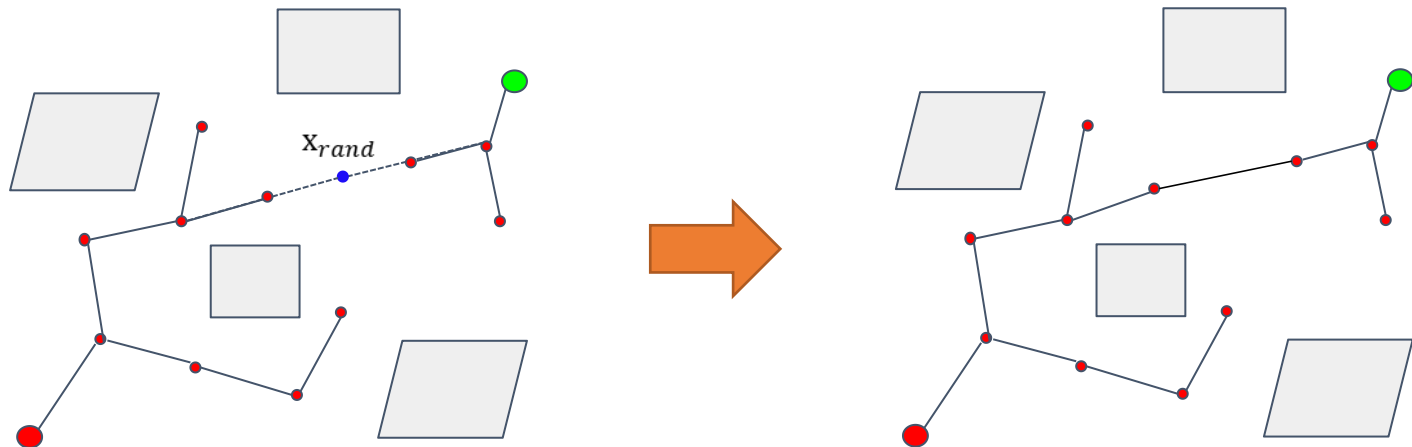
- Grow a tree from both the start point and the goal point
- Path finding when two trees are connected



Rapidly-exploring Random Trees

Note: towards improving efficiency

Bidirectional RRT / RRT Connect

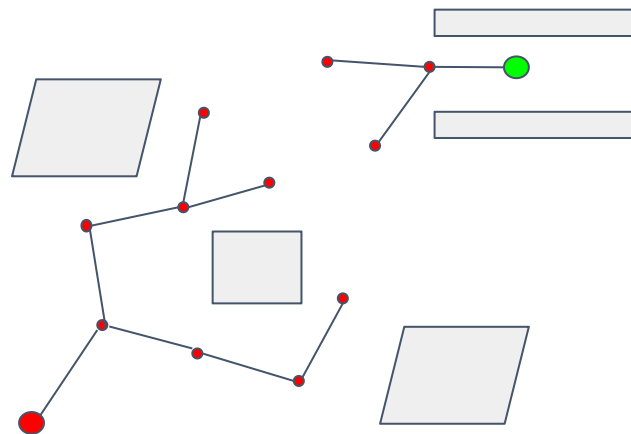
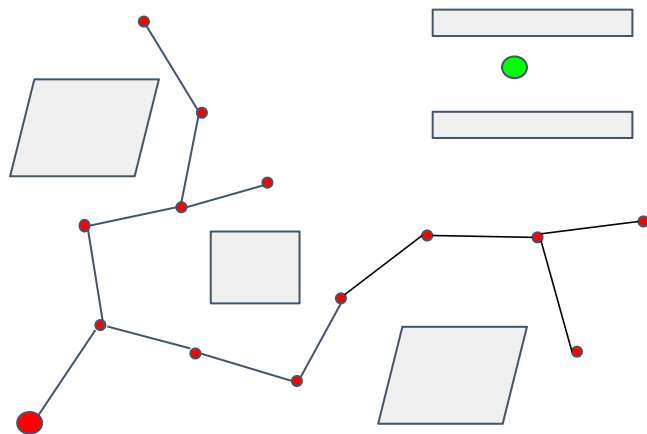




Rapidly-exploring Random Trees

Note: towards path planning efficiency

Bidirectional RRT / RRT Connect





Rapidly-exploring Random Trees

- Incrementally build
- Rapidly searching
- Key functions: Sampling, Nearest, Collision-checking



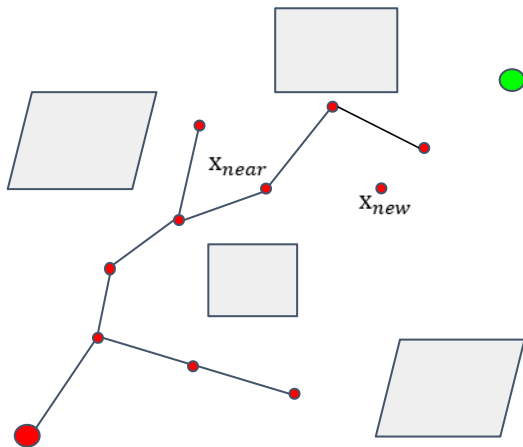


Optimal sampling-based path planning methods



Optimal sampling-based path planning methods

Rapidly-exploring Random Tree*



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init()$;

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow \text{Sample}(\mathcal{M})$;

$x_{near} \leftarrow \text{Near}(x_{rand}, \mathcal{T})$;

$x_{new} \leftarrow \text{Steer}(x_{rand}, x_{near}, \text{StepSize})$;

if $\text{CollisionFree}(x_{new})$ **then**

$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new})$;

$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new})$;

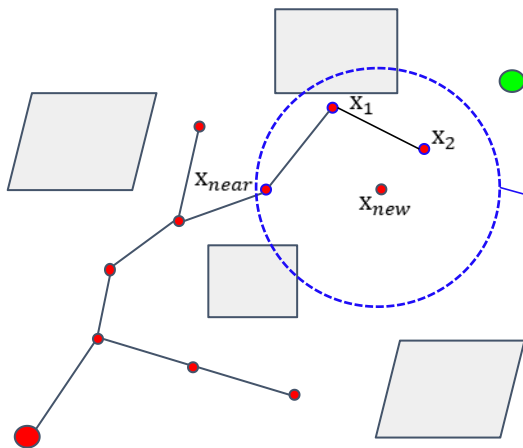
$\mathcal{T}.addNodeEdge(x_{min}, x_{new})$;

$\mathcal{T}.rewire()$;



Optimal sampling-based path planning methods

Rapidly-exploring Random Tree*



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init()$;

for $i = 1$ to n **do**

$x_{rand} \leftarrow \text{Sample}(\mathcal{M})$;

$x_{near} \leftarrow \text{Near}(x_{rand}, \mathcal{T})$;

$x_{new} \leftarrow \text{Steer}(x_{rand}, x_{near}, \text{StepSize})$;

if $\text{CollisionFree}(x_{new})$ **then**

$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new})$;

$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new})$;

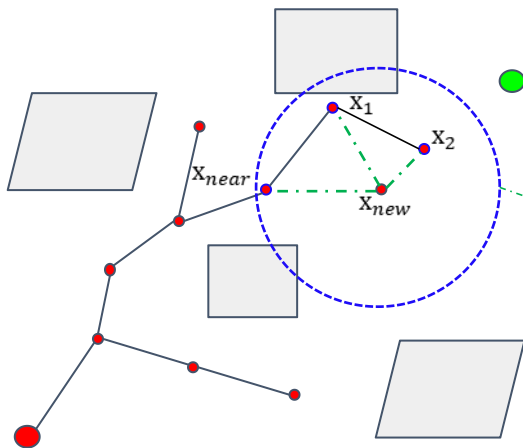
$\mathcal{T}.addNodeEdge(x_{min}, x_{new})$;

$\mathcal{T}.rewire()$;



Optimal sampling-based path planning methods

Rapidly-exploring Random Tree*



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init()$;

for $i = 1$ to n **do**

$x_{rand} \leftarrow \text{Sample}(\mathcal{M})$;

$x_{near} \leftarrow \text{Near}(x_{rand}, \mathcal{T})$;

$x_{new} \leftarrow \text{Steer}(x_{rand}, x_{near}, \text{StepSize})$;

if $\text{CollisionFree}(x_{new})$ **then**

$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new})$;

$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new})$;

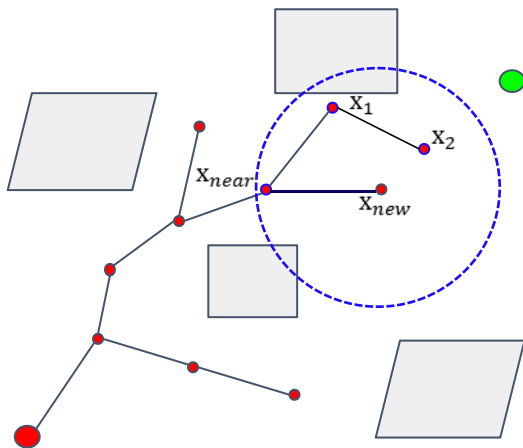
$\mathcal{T}.addNodeEdge(x_{min}, x_{new})$;

$\mathcal{T}.rewire()$;



Optimal sampling-based path planning methods

Rapidly-exploring Random Tree*



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init();$

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow \text{Sample}(\mathcal{M});$

$x_{near} \leftarrow \text{Near}(x_{rand}, \mathcal{T});$

$x_{new} \leftarrow \text{Steer}(x_{rand}, x_{near}, \text{StepSize});$

if $\text{CollisionFree}(x_{new})$ **then**

$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new});$

$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new});$

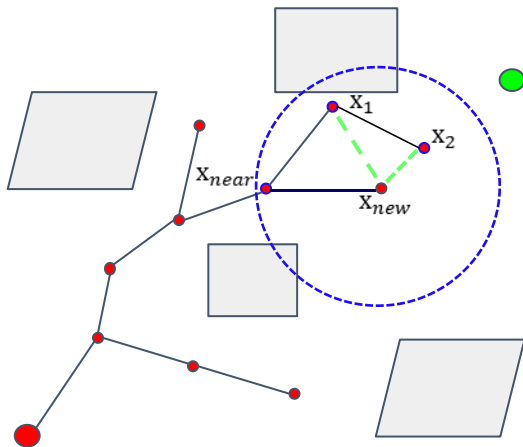
$\mathcal{T}.addNodeEdge(x_{min}, x_{new});$

$\mathcal{T}.rewire();$



Optimal sampling-based path planning methods

Rapidly-exploring Random Tree*



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init()$;

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow \text{Sample}(\mathcal{M})$;

$x_{near} \leftarrow \text{Near}(x_{rand}, \mathcal{T})$;

$x_{new} \leftarrow \text{Steer}(x_{rand}, x_{near}, \text{StepSize})$;

if $\text{CollisionFree}(x_{new})$ **then**

$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new})$;

$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new})$;

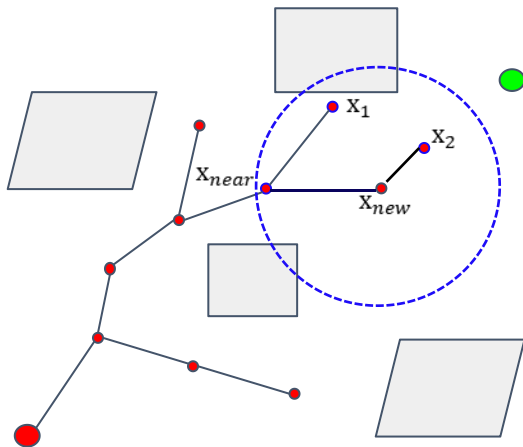
$\mathcal{T}.addNodeEdge(x_{min}, x_{new})$;

$\mathcal{T}.rewire()$;



Optimal sampling-based path planning methods

Rapidly-exploring Random Tree*



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}.init()$;

for $i = 1$ **to** n **do**

$x_{rand} \leftarrow \text{Sample}(\mathcal{M})$;

$x_{near} \leftarrow \text{Near}(x_{rand}, \mathcal{T})$;

$x_{new} \leftarrow \text{Steer}(x_{rand}, x_{near}, \text{StepSize})$;

if $\text{CollisionFree}(x_{new})$ **then**

$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new})$;

$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new})$;

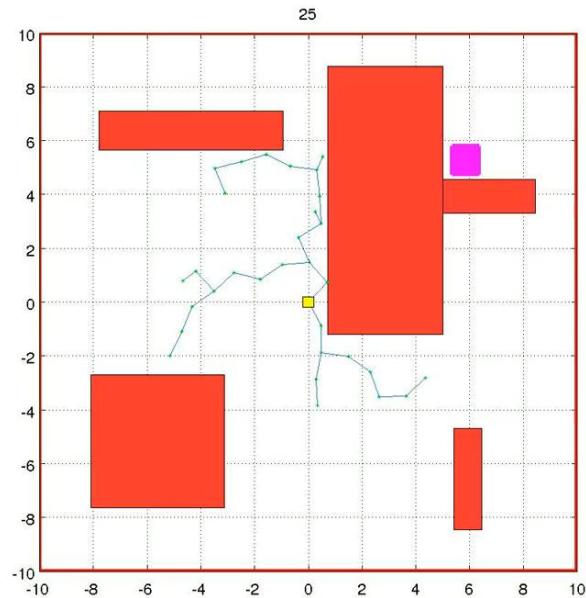
$\mathcal{T}.addNodeEdge(x_{min}, x_{new})$;

$\mathcal{T}.rewire()$;



Optimal sampling-based path planning methods

Rapidly-exploring Random Tree*

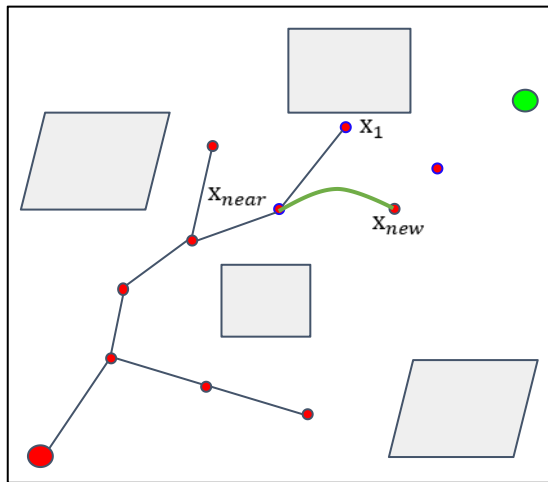
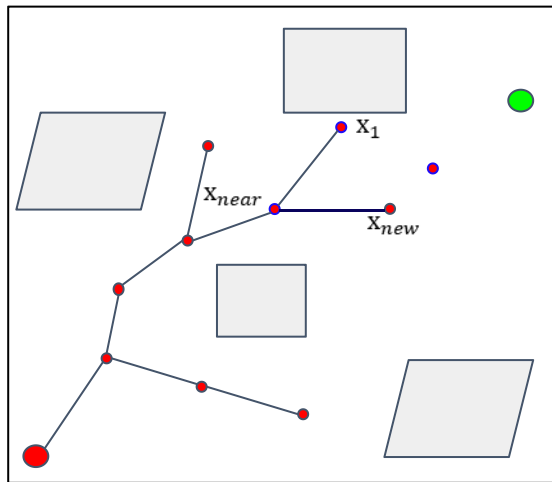


<https://www.youtube.com/watch?v=YKiQTJpPFkA>



Optimal sampling-based path planning methods

Kinodynamic-RRT*

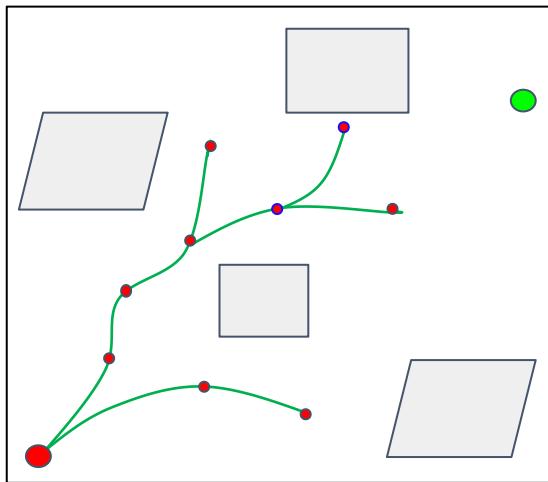
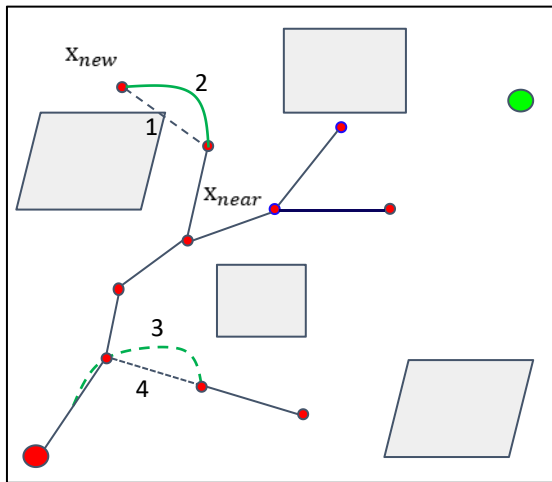


Change Steer() function to fit with motion or other constraints in robot navigation.



Optimal sampling-based path planning methods

Kinodynamic-RRT*

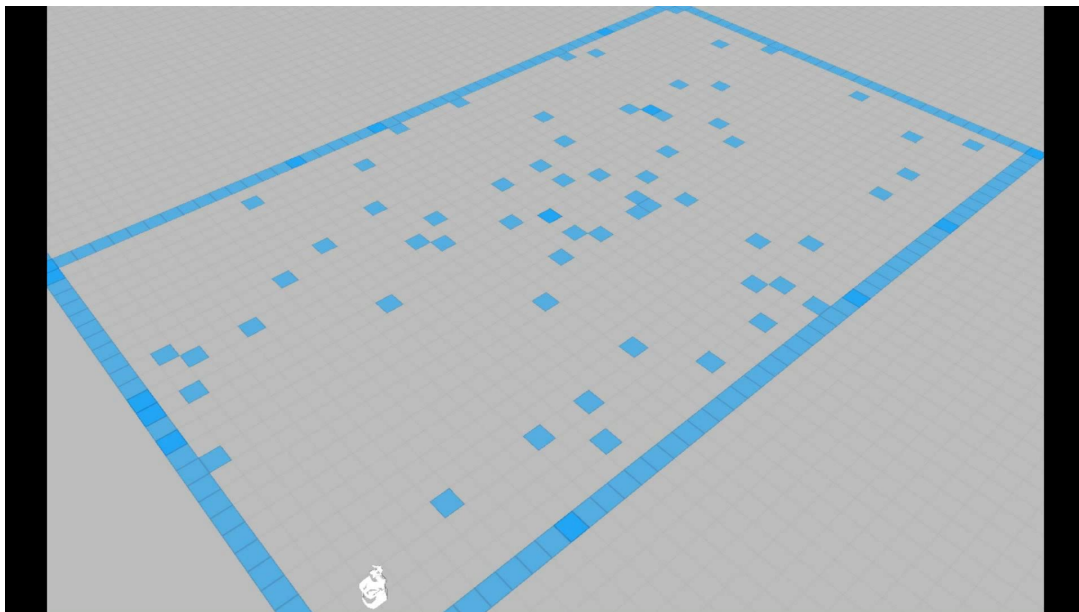


Change **Steer()** function to fit with motion or other constraints in robot navigation.



Optimal sampling-based path planning methods

Kinodynamic-RRT*

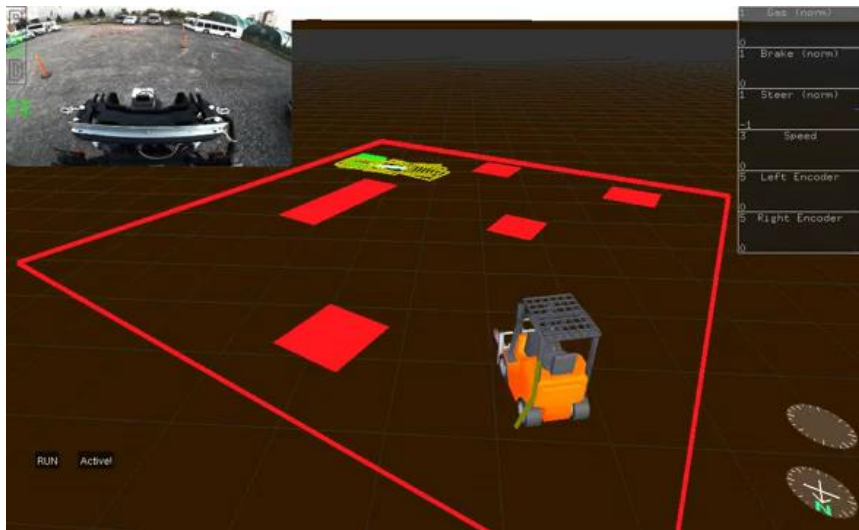


Change Steer() function to fit with motion or other constraints in robot navigation.



Optimal sampling-based path planning methods

Anytime-RRT*



Keep optimizing the leaf RRT tree when the robot executes the current trajectory Anytime Fashion

[Anytime Motion Planning using the RRT*](#)



Optimal sampling-based path planning methods

- Rewire function
- RRT*
- Kino-dynamic RRT*
- Anytime RRT*



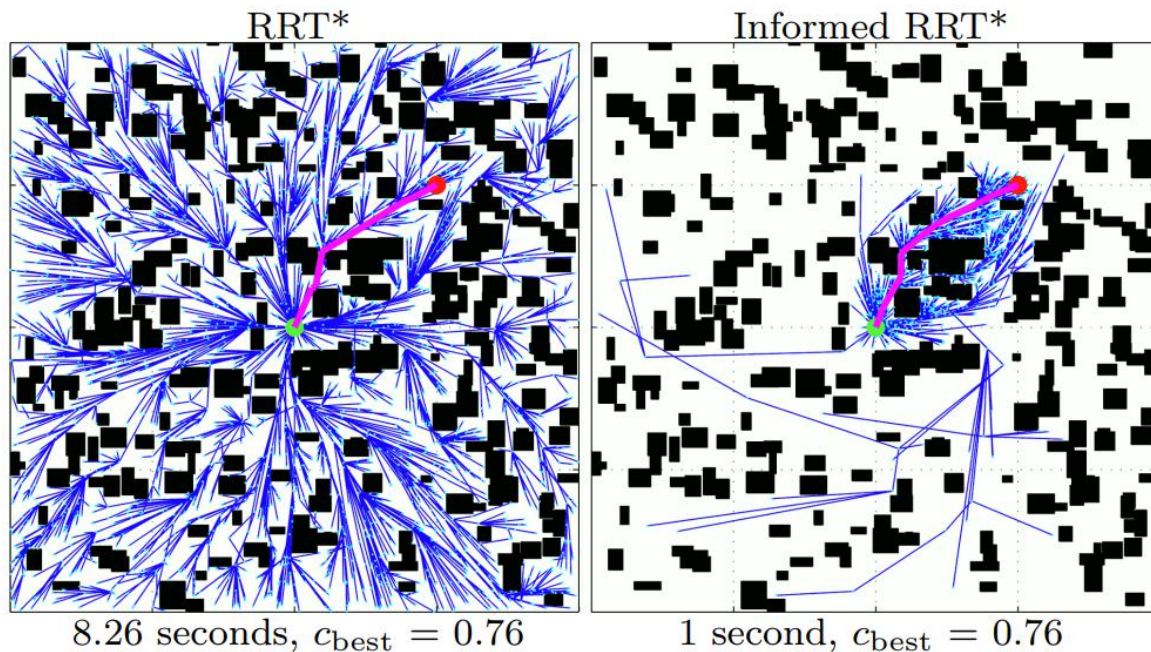


Advanced Sampling-based Methods



Advanced Sampling-based Methods

Informed RRT*

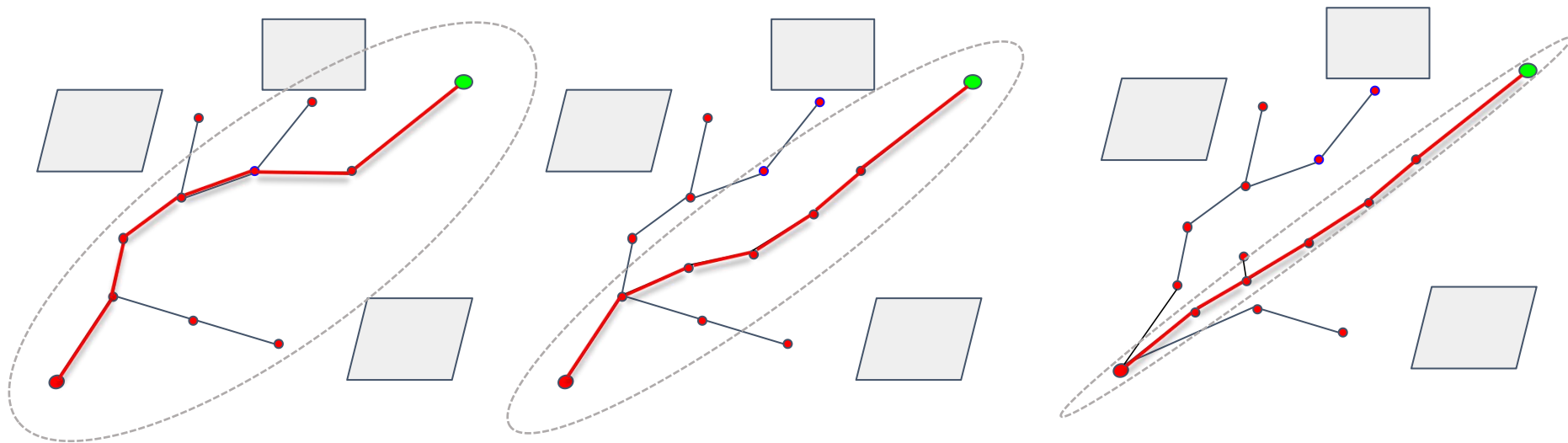


[Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic](#)



Advanced Sampling-based Methods

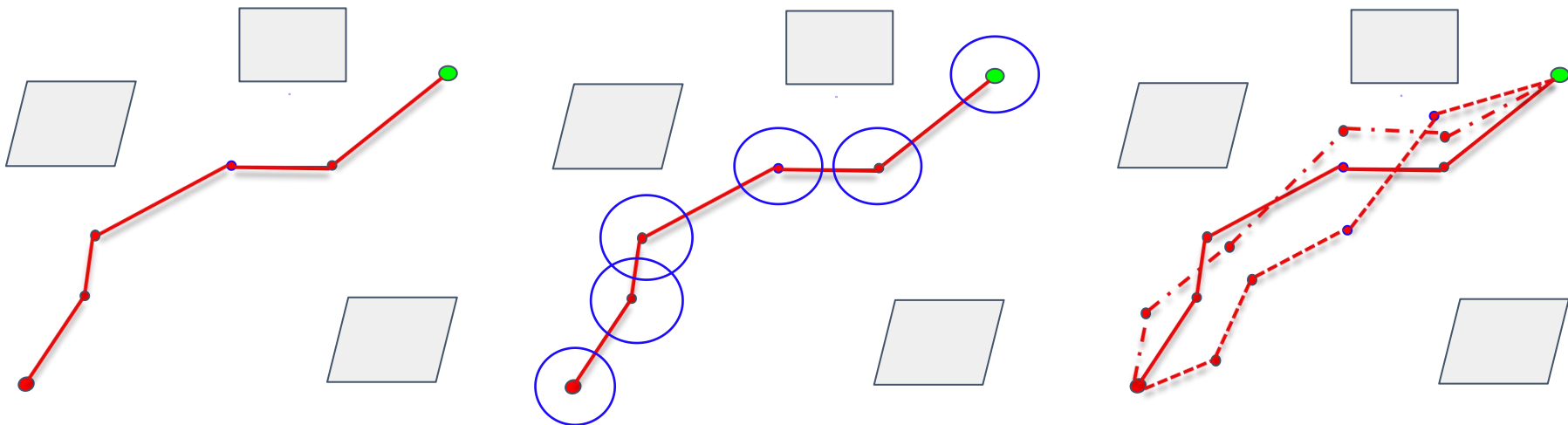
Informed RRT*





Advanced Sampling-based Methods

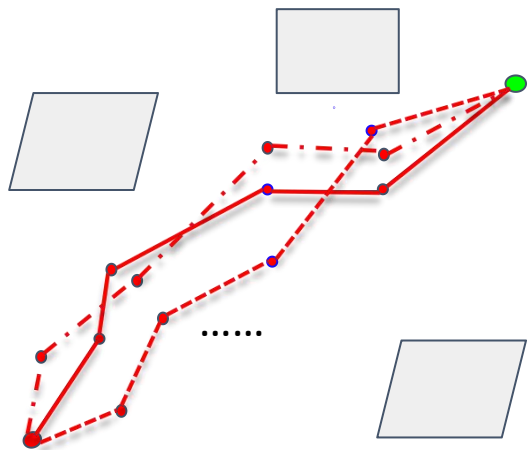
Cross-entropy motion planning



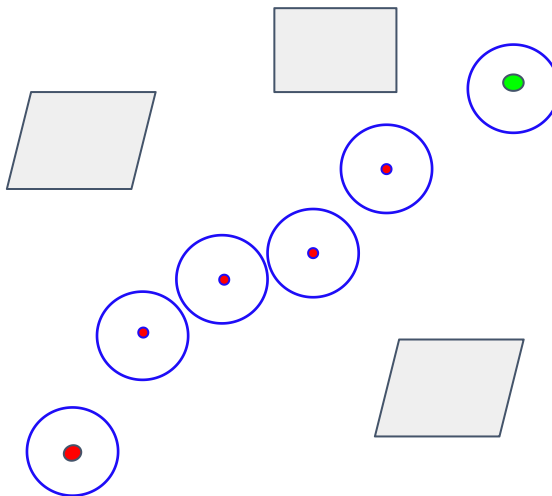


Advanced Sampling-based Methods

Cross-entropy motion planning



Select elite paths



Update sample region

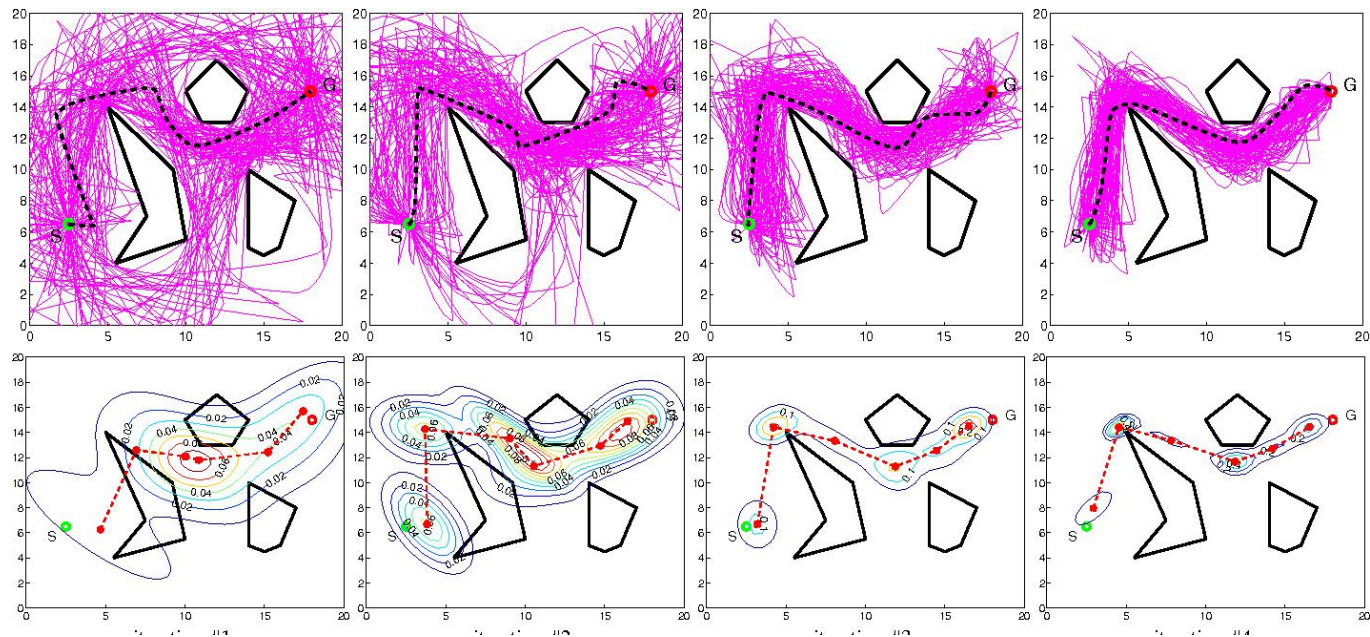


Next sample round



Advanced Sampling-based Methods

Cross-entropy motion planning



Link of implementation on github:



Advanced Sampling-based Methods

Other variants

- Lower Bound Tree RRT (LBTRRT)[a]
- Sparse Stable RRT[b]
- Transition-based RRT (T-RRT)[c]
- Vector Field RRT[d]
- Parallel RRT (pRRT)[e]
- Etc.[f]

[1] An Overview of the Class of Rapidly-Exploring Random Trees

[2] <http://msl.cs.uiuc.edu/rrt/>

[a] <https://arxiv.org/pdf/1308.0189.pdf>

[b] http://pracsyslab.org/sst_software

[c] http://homepages.laas.fr/jcortes/Papers/jaillet_aaaiWS08.pdf

[d] <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6606360>

[e] https://robotics.cs.unc.edu/publications/lchnowski2012_IROS.pdf

[f] <https://github.com/zychaoqun>

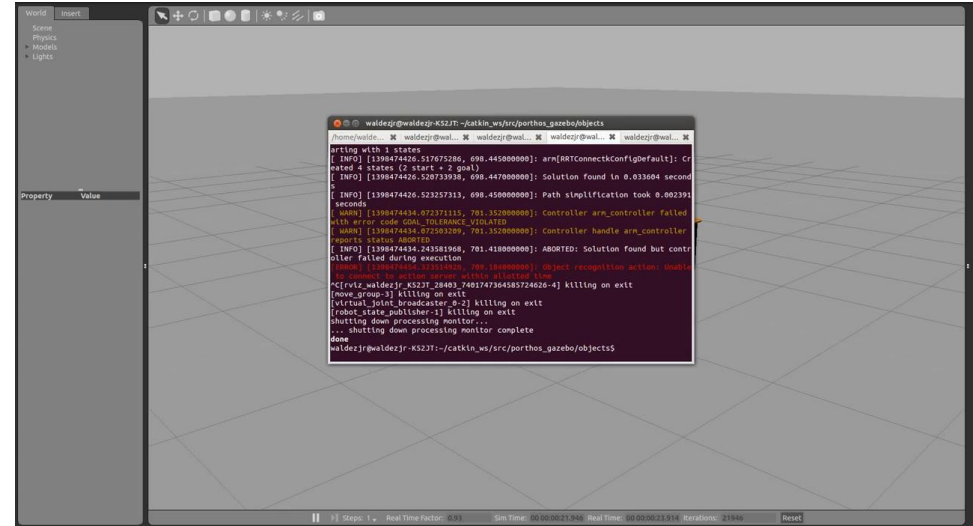


Implementation



Implementation

- Open Motion Planning Library [1]
- Moveit with ROS [2]
- Tutorials[3]



[1] <https://ompl.kavrakilab.org/>

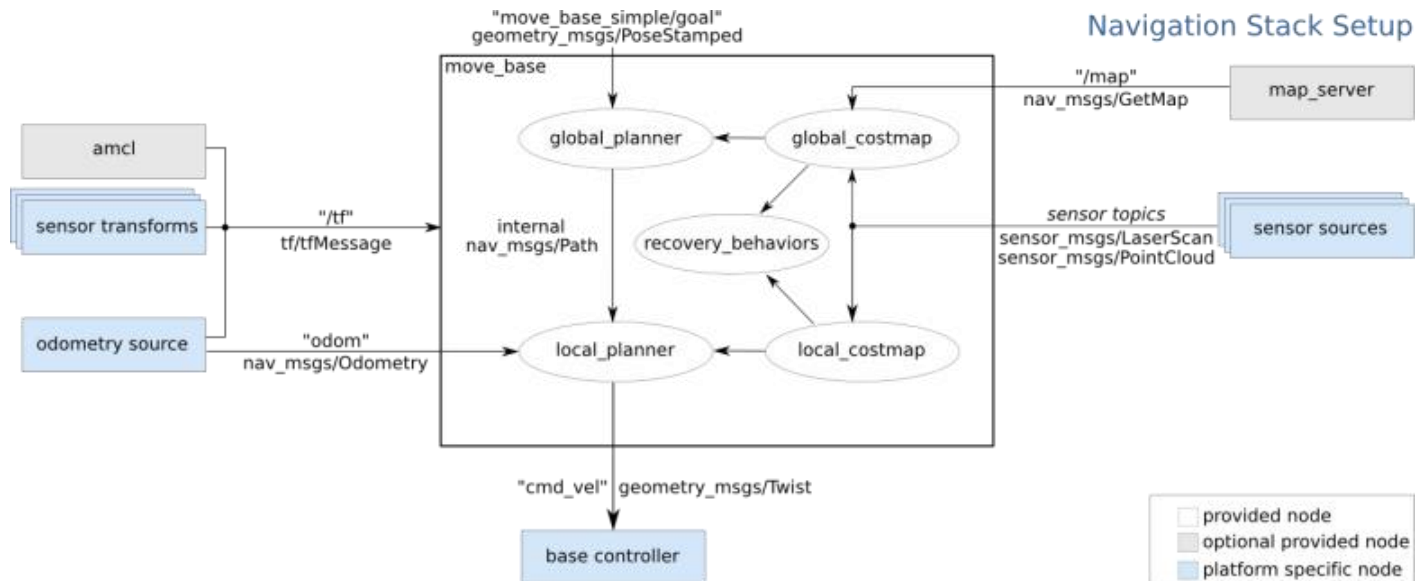
[2] <https://moveit.ros.org/>

[3] https://industrial-training-master.readthedocs.io/en/melodic/_source/session4/Motion-Planning-CPP.html



Implementation

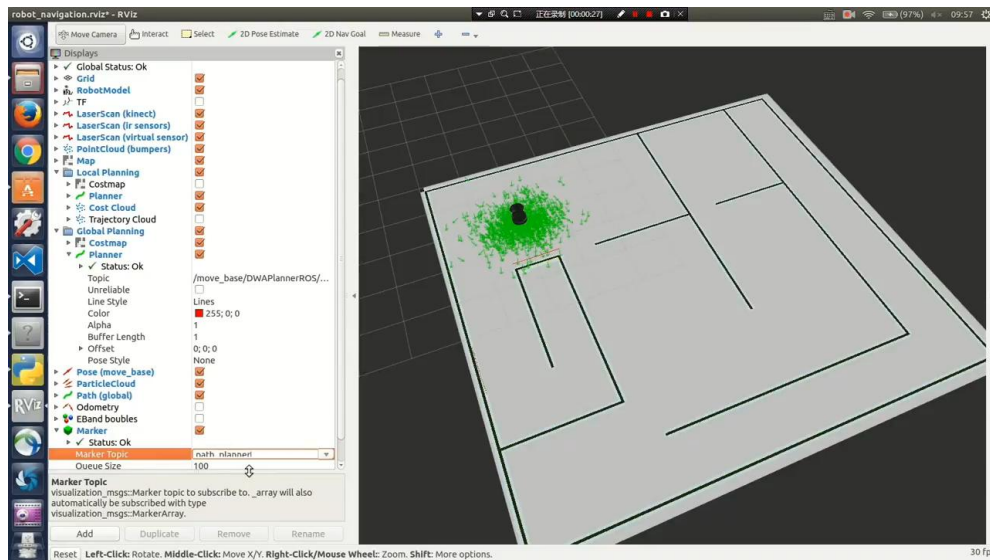
- Navigation stick - ROS





Implementation

- Navigation stack - ROS
 - Global planner
A*, D*, RRTs, etc
 - Local planner
Dwa, eband, Teb, etc



Video demonstration of RRT implemented on ROS [1]

[1] <https://youtu.be/FsZ9b6fsQUg>



Homework

- Implementation of RRT
 - You can either use MATLAB or C++
 - Hints: write RRT as a global planner in ROS
- Bonus: Implementation of Informed-RRT*



在线问答

Q&A



结语

感谢各位聆听!

Thanks for Listening ●

