RRT - Rapidly-Exploring Random Trees

Tim Jagla, André Keuns, Anne Reich

Otto-von-Guericke-Universität Magdeburg

February 1, 2015

Collision Free Path Planning

Motivation

- path planning: find a path from location *A* to *B*
- Example for path planning:
  - mobile robot inside a build
  - shall go to location XY
- Example extension for collision free path planning:
  - avoiding walls and not falling stairs

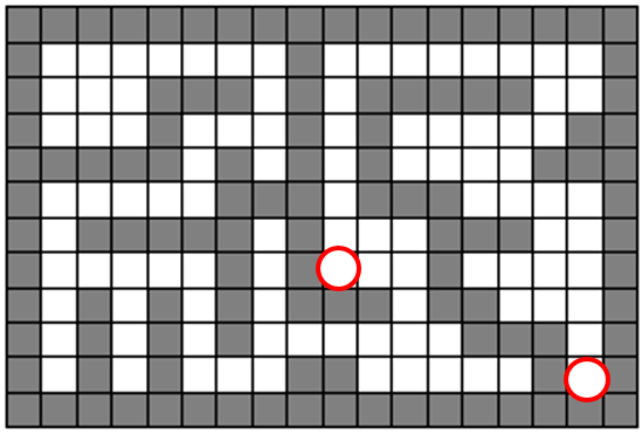

Figure: first example for path planing

Simple Example

- Simple General Forward Search
  - State: Unvisited, Dead, Alive
  - Priority queue, Q, with the set of alive states
  - Start loop over Q
  - In each while iteration check next state
    - It is the goal, is terminate
    - Otherwise, it tries applying every possible action

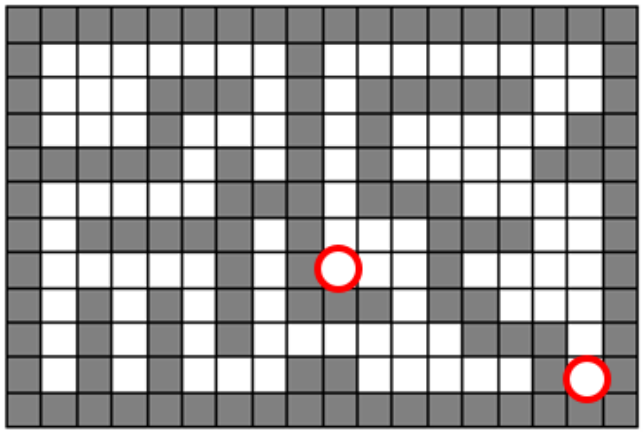

Figure: second example for path planing

Algorithms

- Other known collsion free path planning algorithms
  - Breadth first
  - Deep first
  - Dijikstra‘s algorithm
  - A*
  - Best First
  - Backward search
  - ...

Principles

- Basic Ingredients of Planning
  - State
  - Input
  - Initial and goal states
  - A criterion: Feasibility and/or Optimality
  - a plan

Rapidly-Exploring Random Trees

Principles

- Grows a tree tooted at the starting configuration by using random samples from the search space
- As each sample is drawn, a connection is attempted between it an the nearest state in the tree
- If the connection is feasible, this results in the addition of the new state to the tree
- The probability of expanding an existing state is proportional to the size of ist Voronoi region
    - As the largest Voronoi regions belong to the states on the frontier of the search = the tree preferentially expands towards large unsearched areas

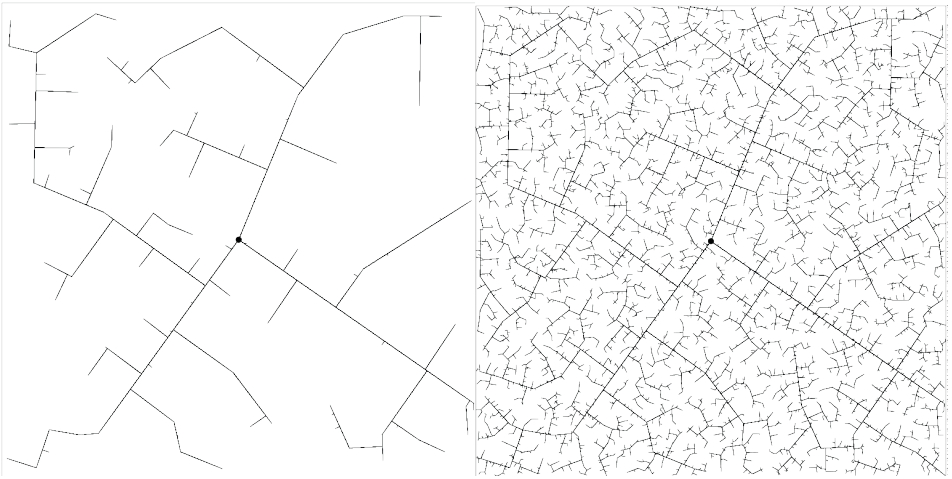

Figure: rrt with 45 and 2345 iterations

Nice Properties

- The expansion is heavily biased toward unexplored portions of the state space
- The distribution of vertices approaches the sampling distribution, leading to consistent behavior
- Is probabilistically complete under very general conditions
- sThe algorithm is relatively simple, which facilitates performance analysis

Nice Properties

- It always remains connected, even though the number of edges is minimal
- Can be considered as a plat planning module, which can be adapted and incorporated into a wide variety of planning systems
- Entire plath planning algorithms can be constructed without requiring the ability to steer the system between two prescibed states, which greatly broadens the applicability of RRTs

Notations

T = RRT (Tree of vertices)

C = configuration space of a rigid body or systems of bodies in a world

$T(C)$ = tanget bundle of the configuration space

C_{goal} = goal region, $C_{goal} \subset C$

C_{obs} = obstacle region, $C_{obs} \subset C$

C_{free} = region without obstacles, $C_{free} \subset C$

q_{init} = initale state

q_n = neighbor of a state

$alpha$ = random state

$edges$ = correspond to a path that lies entirely in C_{free}

Pseudo Code

```
1  generate_rt(robot, vertex_count, delta_time):
2  {
3    q_init = is the current configuration of the robot
4
5    T(q_init)
6
7    for i to vertex_count do
8      alpha = generate_random_state(robot)
9      q_n = find_nearest_neighbor(robot, alpha, T)
10     q_s = generate_state(robot, q_n, alpha, delta_time)
11     T.insert_state(q_s)
12     T.insert_edge(q_n, q_s)
13
14   return T;
15 }
```

Listing 1: pseudocode for rrt algorithm

Function: generate random state

- generate a random state between the minimum and the maximum configuration limits of the robot

```
1 generate_random_state(robot):
2 {
3   min_angle = the minimum angle limits of the robot
4   max_angle = the minimum angle limits of the robot
5
6   return min_angle + (max_angle - min_angle) * random value
        between 0 and 1
7 }
```

Listing 2: pseudocode for random state generation

Function: find nearest neighbor

- bla

Function: generate state

- checked the path between the q_n and the $alpha$ of collisions free
- if it collisions, then give back the last state before the collision
- if the time for trajectory larger as the $delta\_time$, then give back the state at $delta\_time$
- else, then give back the $alpha$

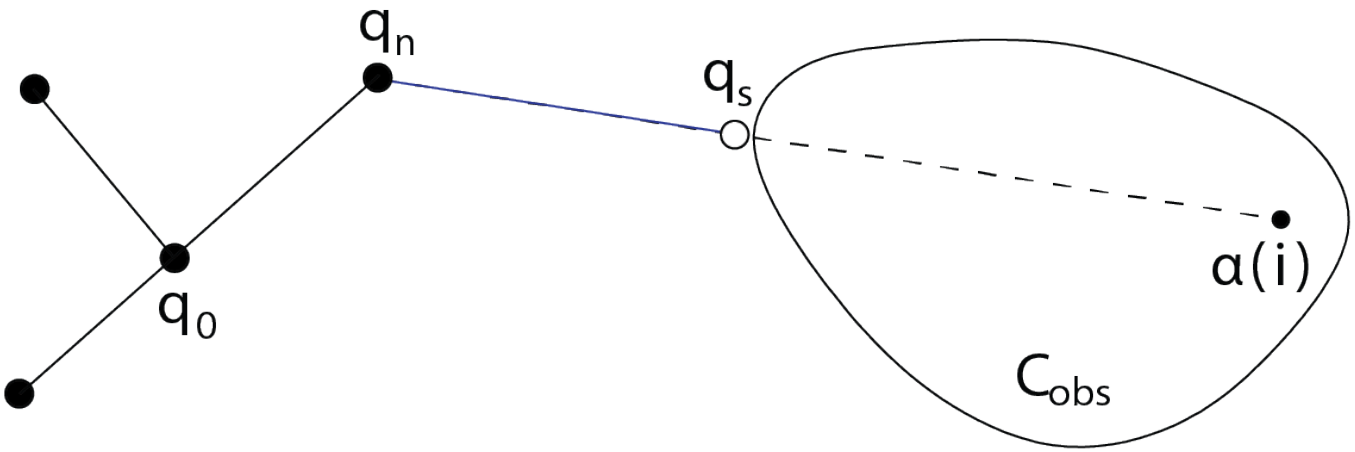

Figure: generate state methode

Challanges of our work

überlegung 2d -> 3d -> 6d / configuration space

Live Demo

graph aufbau vollständigen graphen

Sources

1 Rapidly-Exploring Random Trees: A New Tool for Path Planning - Steven M. LaValle
http://coitweb.uncc.edu/~xiao/itcs6151-8151/RRT.pdf (03.02.2015)

2 Planning Algorithms - Steven M. LaValle
http://planning.cs.uiuc.edu/ (03.02.2015)

3 http://en.wikipedia.org/wiki/Rapidly_exploring_random_tree (03.02.2015)

Thank you for your attention