RRT - Rapidly-Exploring Random Trees

Tim Jagla, André Keuns, Anne Reich

Otto-von-Guericke-Universität Magdeburg

February 2, 2015

Computer Systems in Engineering

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

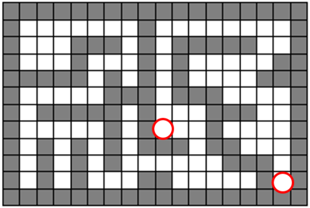

Figure: first example for path planing [2]

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

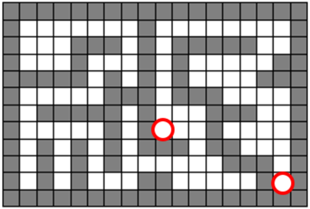

Figure: second example for path planing [2]

Algorithms

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

Computer Systems in Engineering

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

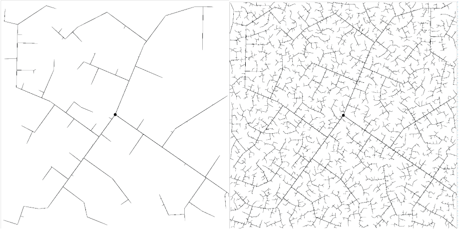

Figure: rrt with 45 and 2345 iterations [2]

Computer Systems in Engineering

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
- The algorithm is relatively simple, which facilitates performance analysis

Computer Systems in Engineering

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

Computer Systems in Engineering

Notations

$$
\begin{aligned}
T &= \text{RRT (Tree of vertices)} \\
C &= \text{configuration space of a rigid body or} \\
&\quad \text{systems of bodies in a world} \\
T(C) &= \text{tanget bundle of the configuration space} \\
C_{goal} &= \text{goal region, } C_{goal} \subset C \\
C_{obs} &= \text{obstacle region, } C_{obs} \subset C \\
C_{free} &= \text{region without obstacles, } C_{free} \subset C \\
q_{init} &= \text{initale state} \\
q_n &= \text{neighbor of a state} \\
alpha &= \text{random state} \\
edges &= \text{correspond to a path that lies entirely in } C_{free}
\end{aligned}
$$

Computer Systems in Engineering

Challanges of our work

- the implementation from 2D to 6D over 3D
  - we are want direct to the configuration space of the robot and so cloud some difficulties go away
- determination of the nearest neighbor of an state/point in space to a state in our tree
  - one way of determination is over the cartesian space, we build us the from our configurations the x-y-z-coordinates in space and calculate the euclidean distance
  - the same way we are going by the splitting of edges
- checking of collision by the trajectory of the robot
  - one way is, we use our available function for the velocity profile and trajectory generation, and test if the path to our goal state is free
  - an another way

Computer Systems in Engineering

Pseudo Code

```
generate_rt(robot, vertex_count, delta_time):
{
  q_init = is the current configuration of the robot

  T(q_init)

  for i to vertex_count do
    alpha = generate_random_state(robot)
    q_n = find_nearest_neighbor(robot, alpha, T)
    q_s = generate_state(robot, q_n, alpha, delta_time)
    T.insert_state(q_s)
    T.insert_edge(q_n, q_s)

  return T;
}
```

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

```
generate_random_state(robot):
{
  min_angle = the minimum angle limits of the robot
  max_angle = the minimum angle limits of the robot

  return min_angle + (max_angle - min_angle) * random value
      between 0 and 1
}
```

Listing 2: pseudocode for random state generation

Computer Systems in Engineering

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 $alpha$

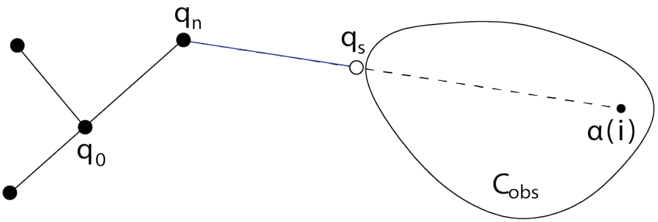

Figure: generate state methode [2]

Computer Systems in Engineering

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)

Computer Systems in Engineering

Thank you for your attention

Computer Systems in Engineering