THE UNIVERSITY of EDINBURGH
informatics

Advanced Robotics

Motion Planning II

Steve Tonneau
School of Informatics
University of Edinburgh

Outline

- Motion planning: goals and challenges
- An intuitive approach: Potential Fields
- **Sampling based motion planning**
  - **Rapidly exploring Random Tree (RRT): principle and code demo (in tutorial)**

    Variants of RRT algorithms
  - **Multi-Query Planner: Probabilistic Roadmap (PRM)**

Sampling based motion planning

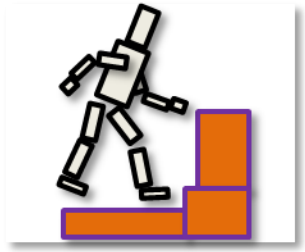
Working in the configuration space

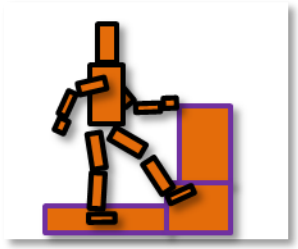
- Today, we'll still only talk about geometry
- Still stuck with the duality:
  - Need to plan in C to control robot
  - How to represent task in operational space (eg. Collision avoidance)

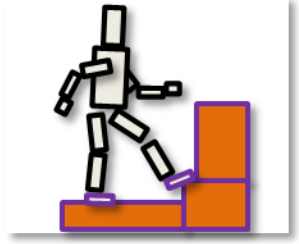
    How is an obstacle represented in C ? We can't really in high dimension…

    Let's see what we can do

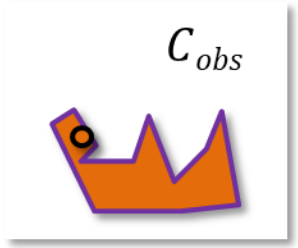
2 (3) subspaces for C







3D environment W



Configuration space C

2 manifolds (subsets) for C

- Given a point q in C, using Forward Kinematics we can determine whether:
  - q is in collision (in C_{free}) => p(q) = true
  - q is not in collision (in C_{obs}) => p(q) = false
- Given:
  - a current configuration q_c
  - a goal configuration q_g
- Design an algorithm to compute a collision free path from q_c to q_g

Let’s discuss

Sampling based motion planning summary

- We have (hopefully) come up with the principles for a global planning algorithm
- A sampling based motion planning algorithm generates a graph were:
  - Nodes are points in the feasible space (in our case C_{free})
  - Edges are feasible paths between Nodes computing with a local steering method:
    - In geometric case, often obtained by interpolation
    - Can be as complex as required by the considered problem
- The formulation is very generic and can be used to represent any robotics planning problem (RRTs were developed for vehicle control, ie differential constraints)

Sampling based motion planning summary

- There are **many** variants in the state of the art
- Three important features:
  - How is the sampling done (Uniformly, close to obstacles etc)?
  - How is the graph extended?
  - What is the **steering method** used?
    - How is feasibility defined?
    - How expensive is the local method?

We'll briefly see two fundamental algorithms

- Rapidly exploring random trees (RRT)
  - Graph extension biased to expand to explore large unexplored areas of the space (more in tutorials)
  - Many variants are used for single-query search:
    computes as fast as possible one path from known configuration to the goal
- Probabilistic Roadmaps (PRMs)
  - Graph extension tries to cover as much as possible the configuration space
  - Multiple-query algorithm: Generate offline, query online

Probablilistic completeness / optimality

Sampling based planners are probabilistically complete: if a solution exists, then the planner will eventually find it (after a possibly infinite time).

Sampling-based approach is not meant to search for the optimal solution, rather it is sub-optimal (variants try to mitigate this).

A big plus:

Sampling-based algorithms are super easy to write

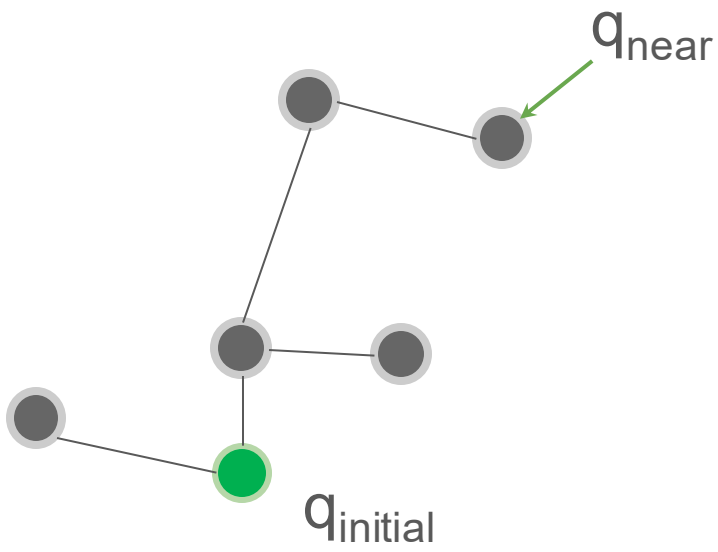
Rapidly-growing Random Trees (RRT)

Basic RRT algorithm – RRT grows from one point

Pseudo code

Algorithm 1 BUILD_RRT(q_{init})

$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
  $q_{rand} \leftarrow$ RANDOM_CONFIG();
  $q_{near} \leftarrow$ NEAREST_NEIGHBOR($q_{rand}, \mathcal{T}$);
  if edge_valid(q_{rand}, q_{near}) **then**
    $\mathcal{T}$.add_vertex(q_{rand});
    $\mathcal{T}$.add_edge(q_{near}, q_{rand});
return $\mathcal{T}$

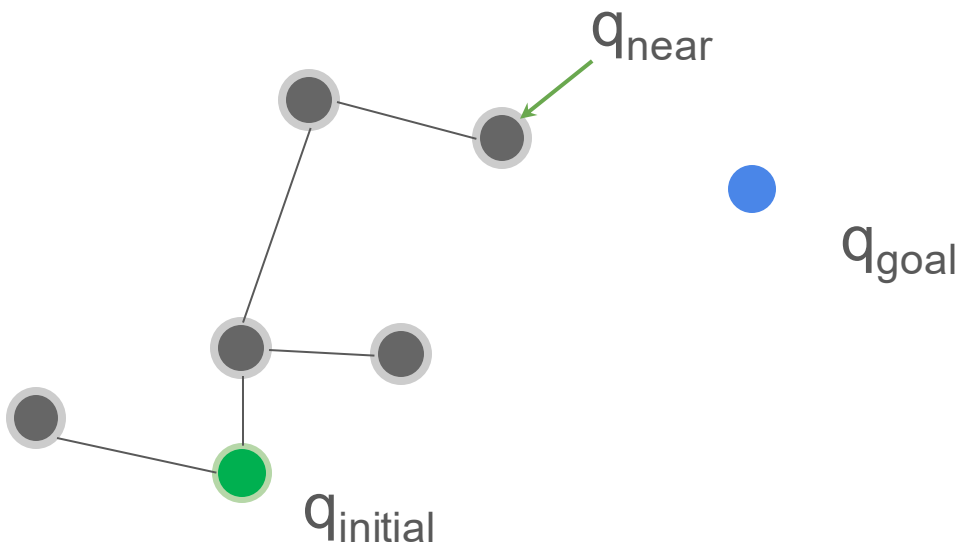


Basic RRT algorithm – single query variant

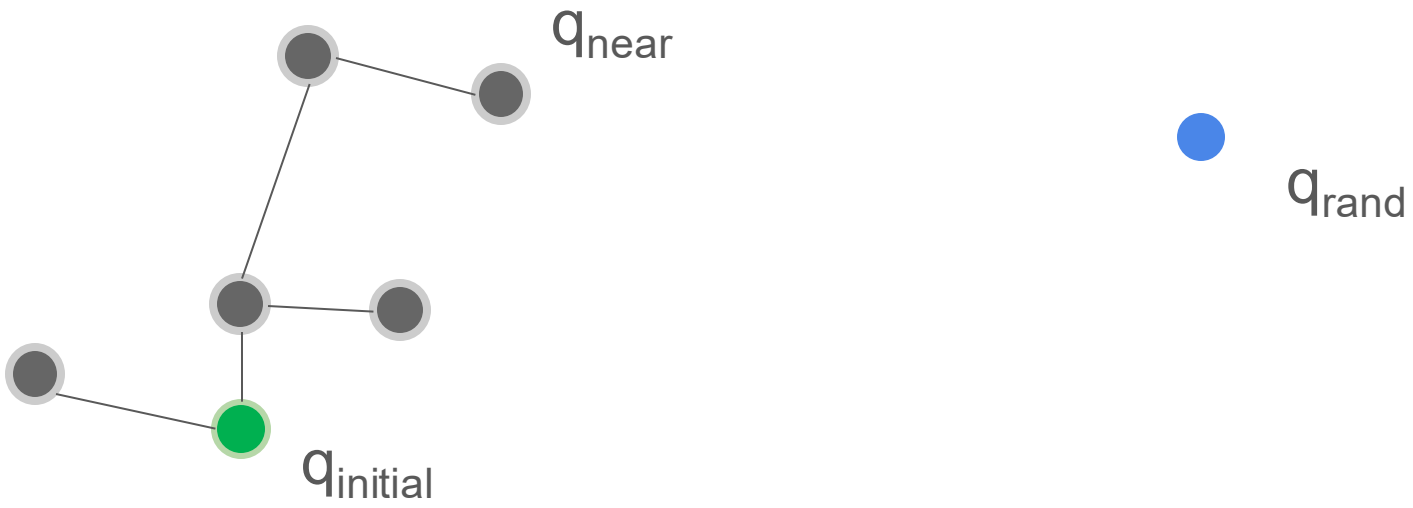
Pseudo code

Algorithm 1 BUILD_RRT(q_{init})

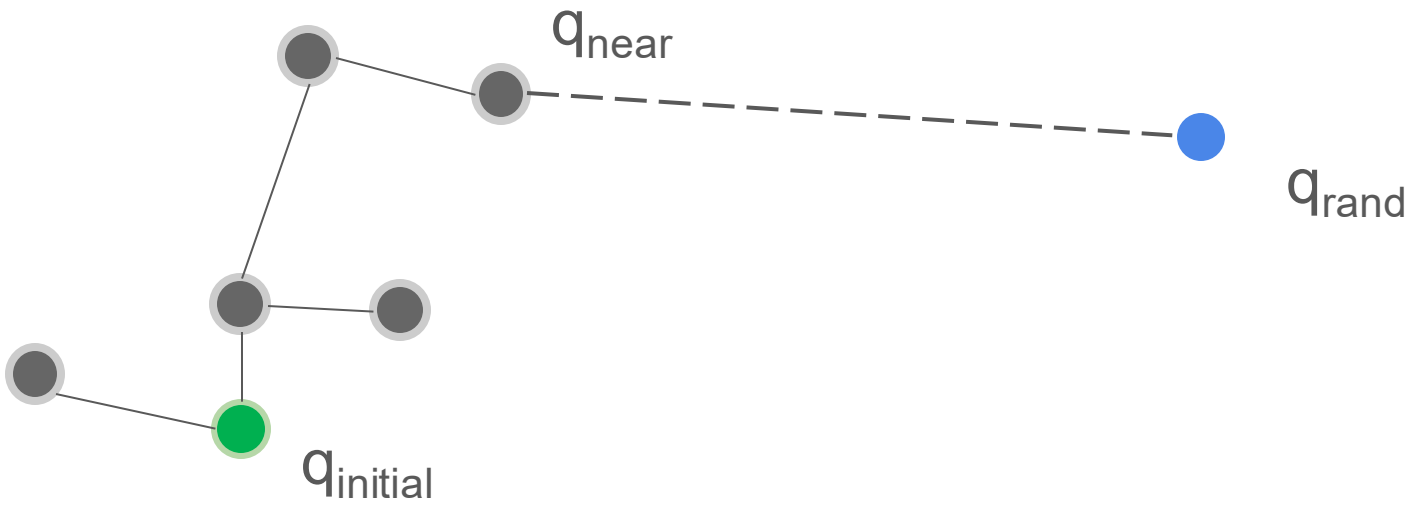
$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
  $q_{rand} \leftarrow$ RANDOM_CONFIG();
  $q_{near} \leftarrow$ NEAREST_NEIGHBOR($q_{rand}, \mathcal{T}$);
  if edge_valid(q_{rand}, q_{near}) **then**
    $\mathcal{T}$.add_vertex(q_{rand});
    $\mathcal{T}$.add_edge(q_{near}, q_{rand});
    if close_to_goal(q_{rand}) **then**
      return SUCCESS
return FAILURE



Basic RRT algorithm

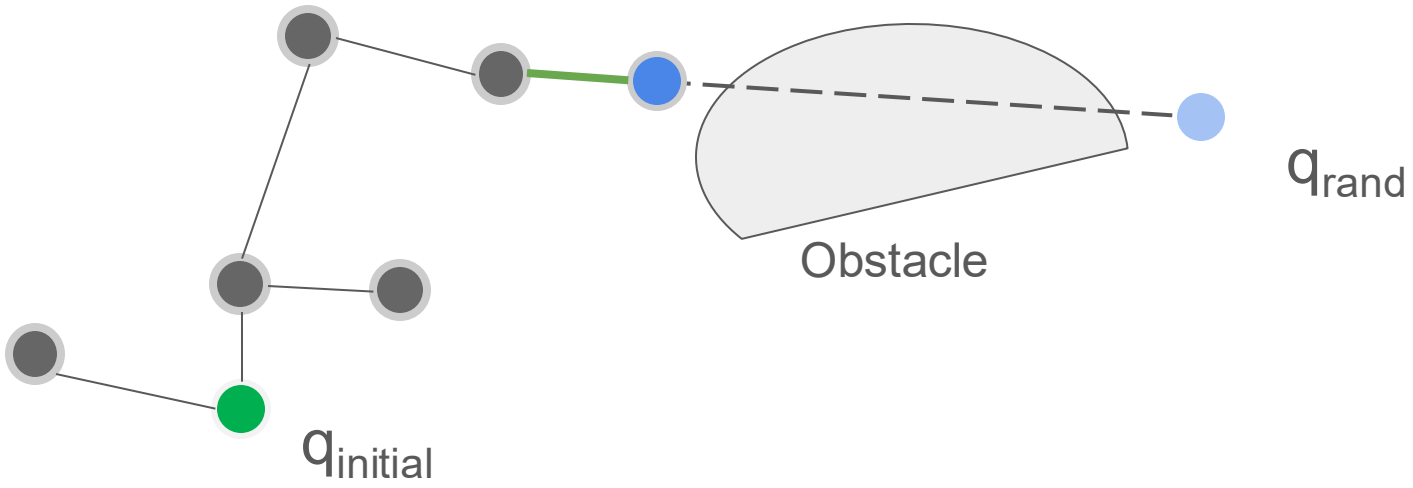


Basic RRT algorithm

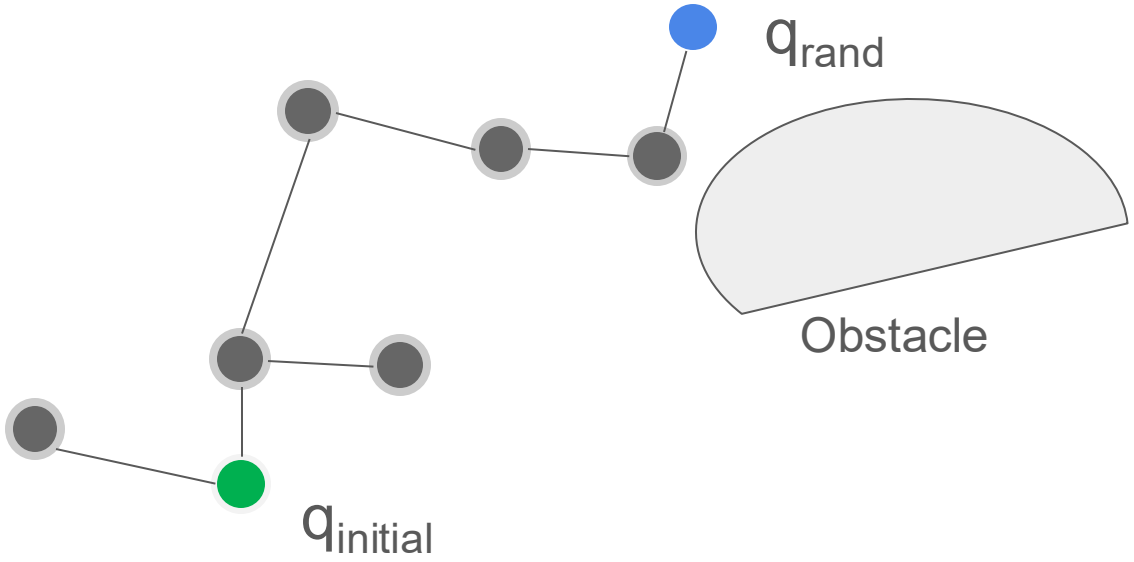


Basic RRT algorithm

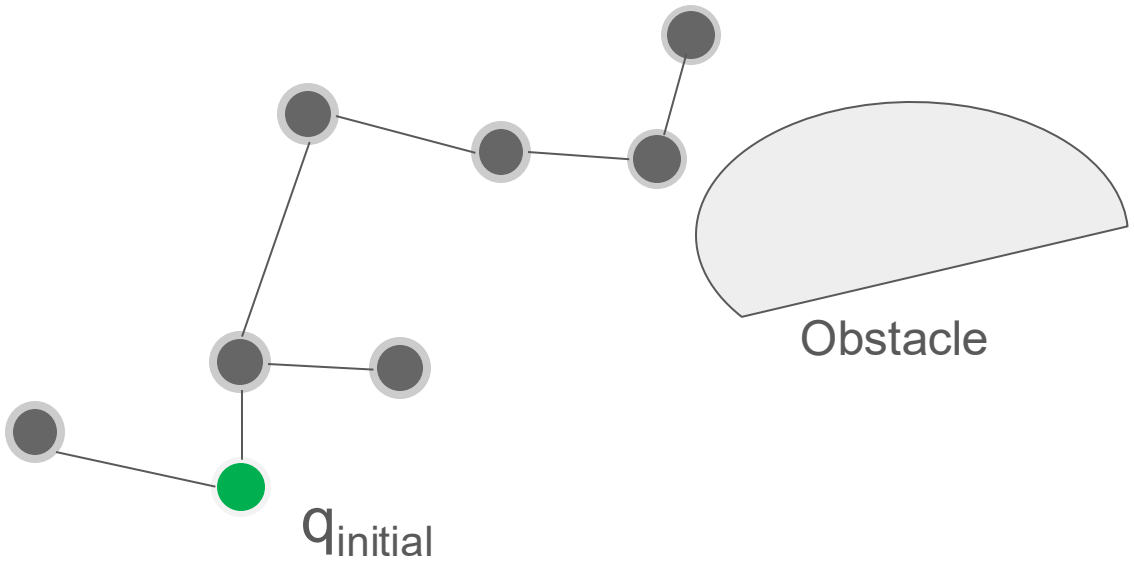
If there is an obstacle lying between q_{near} and q_{rand}, the edge travels up to the obstacle boundary, as far as allowed by the collision check algorithm.



Basic RRT algorithm



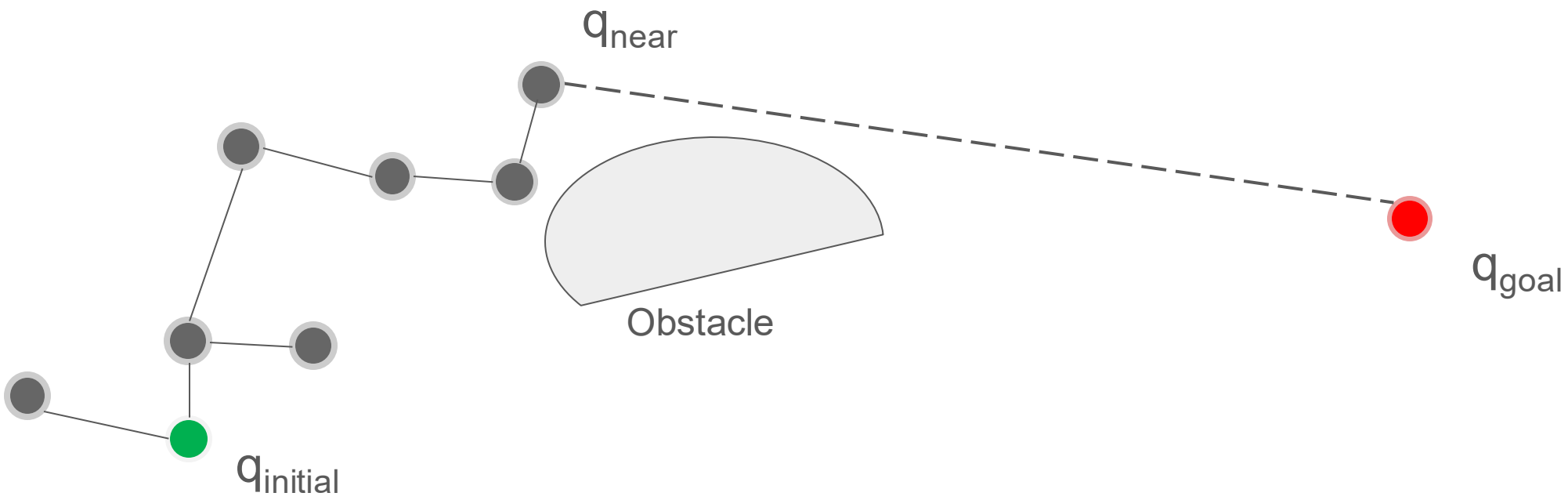
Basic RRT algorithm



Single-query variant:

Check q_{near}=> q_{goal}

Basic RRT algorithm



Single-query variant:

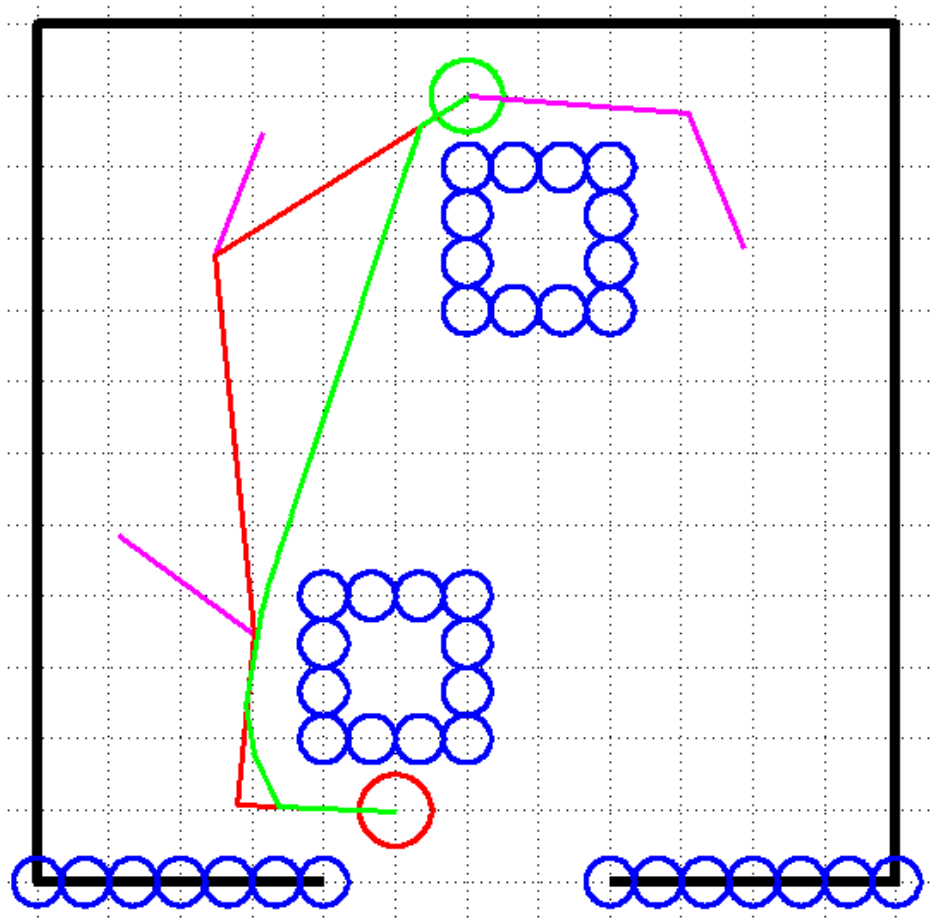
Check q_{near}=> q_{goal}

RRT Example

- In this example, number of iterations is 21.

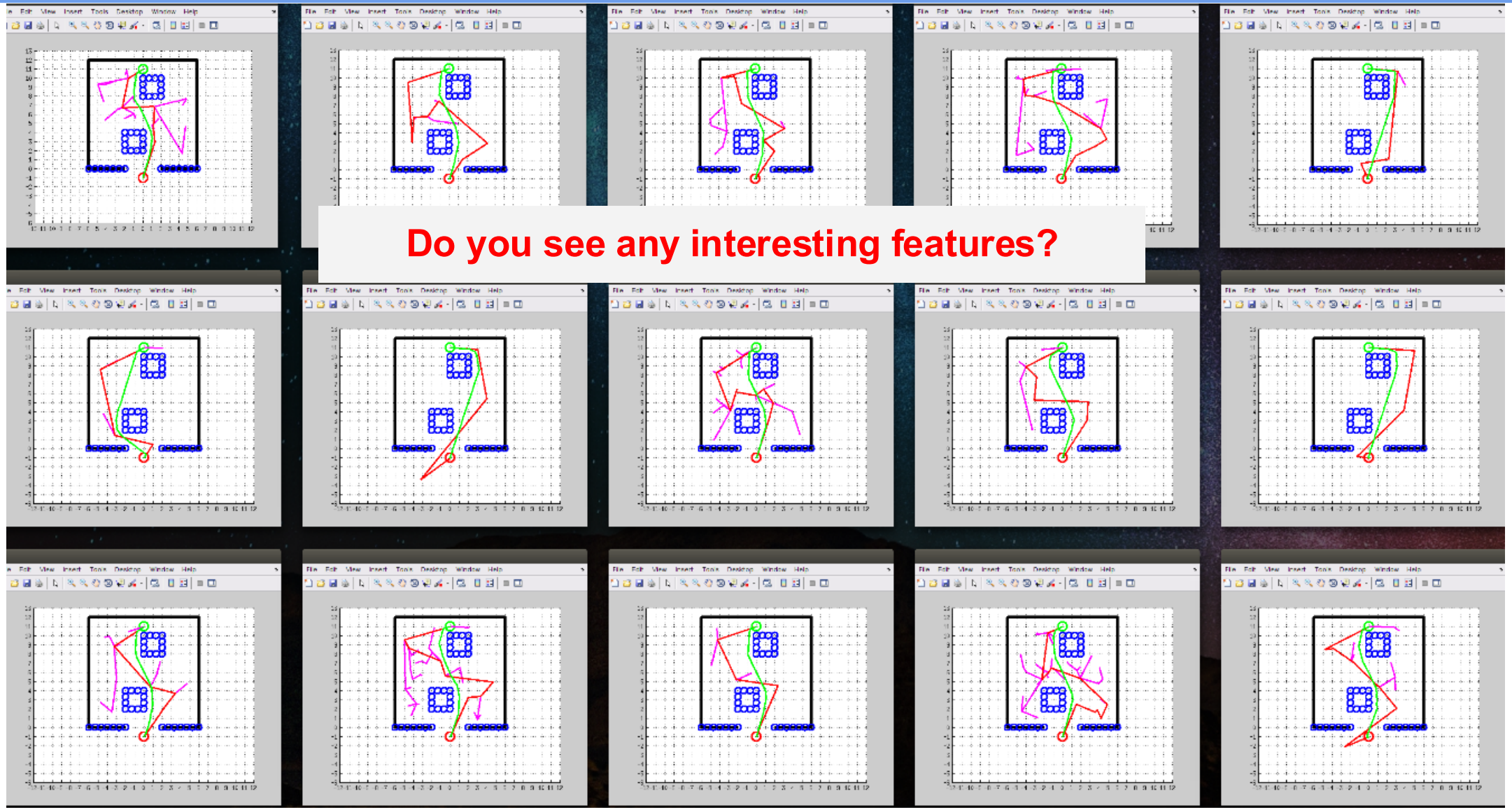
● : initial

● : goal



Code source: University of Illinois at Urbana-Champaign,Control Systems Laboratory for use in GE423 - Mechatronics and research projects: http://coecsl.ece.illinois.edu/ge423/spring13/RickRekoskeAvoid/rrt.html

Same initial & final positions for 15 runs

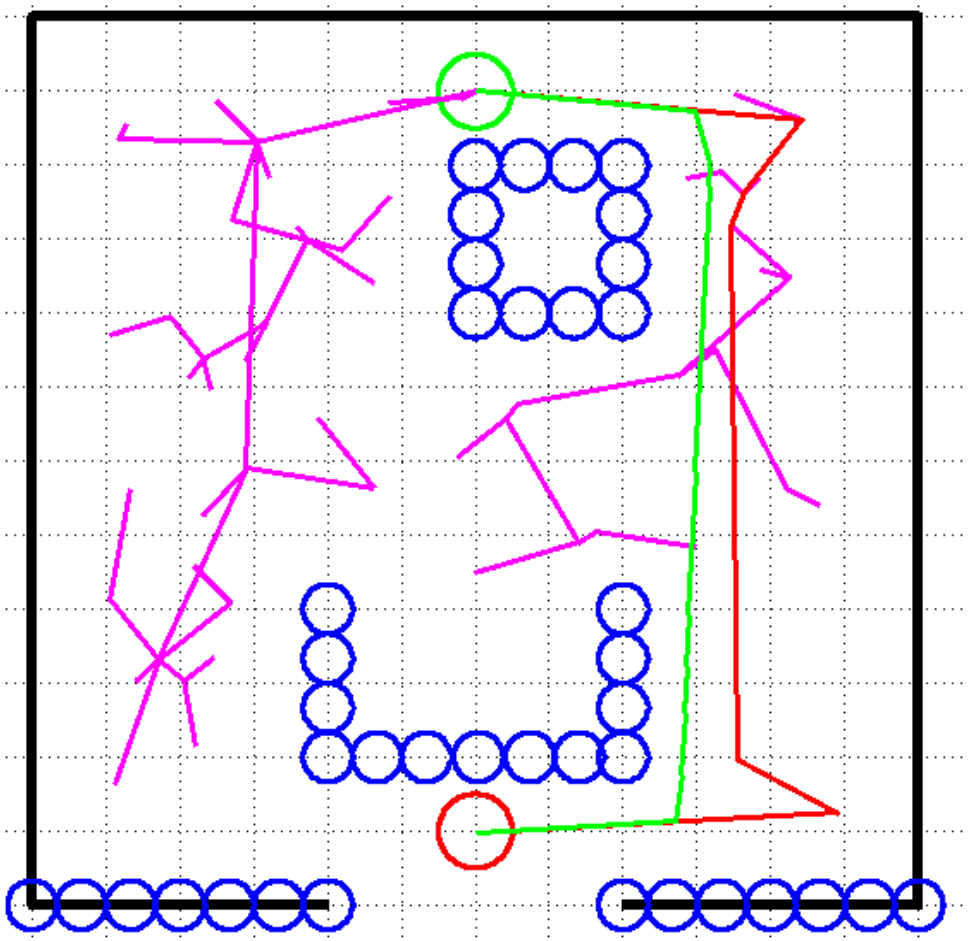


Do you see any interesting features?

RRT Example

- We create a local minima problem encountered in the potential field approach.

- In this example, number of iterations is 114.

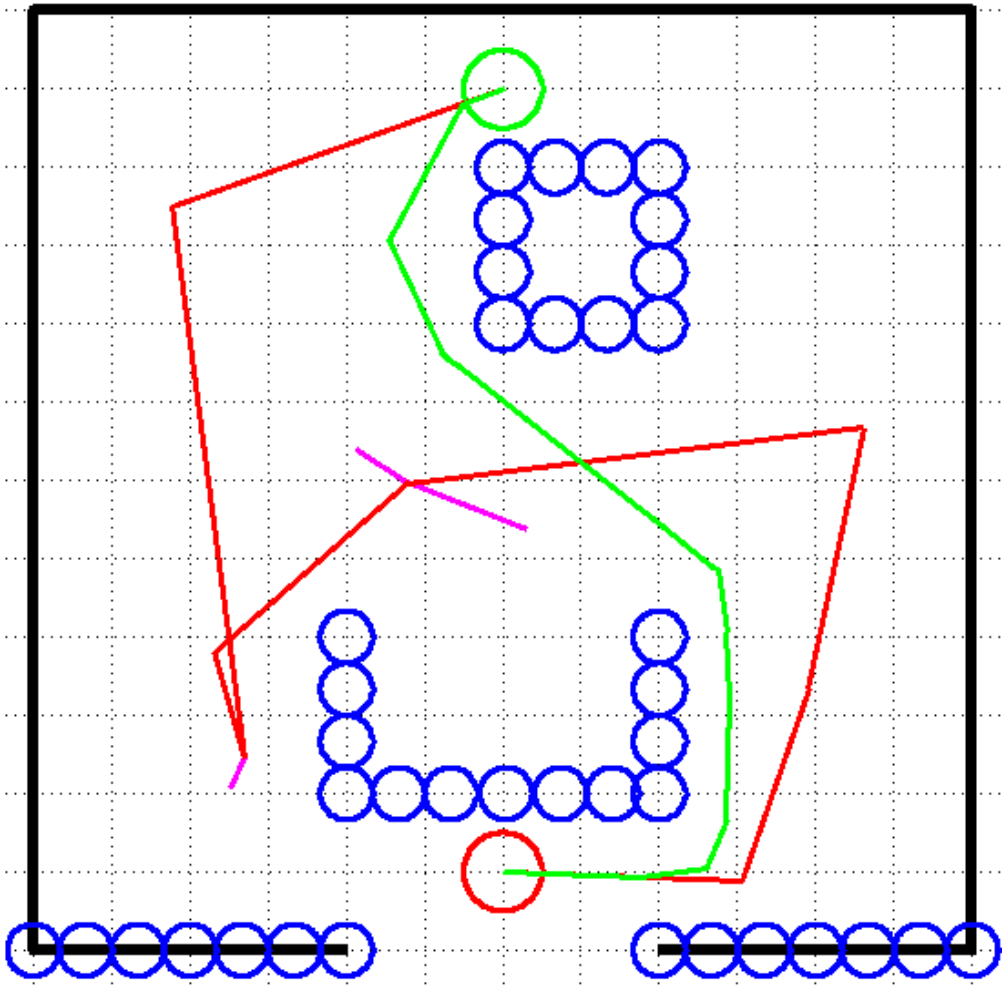


RRT Example

- We create a local minima problem encountered in the potential field approach.

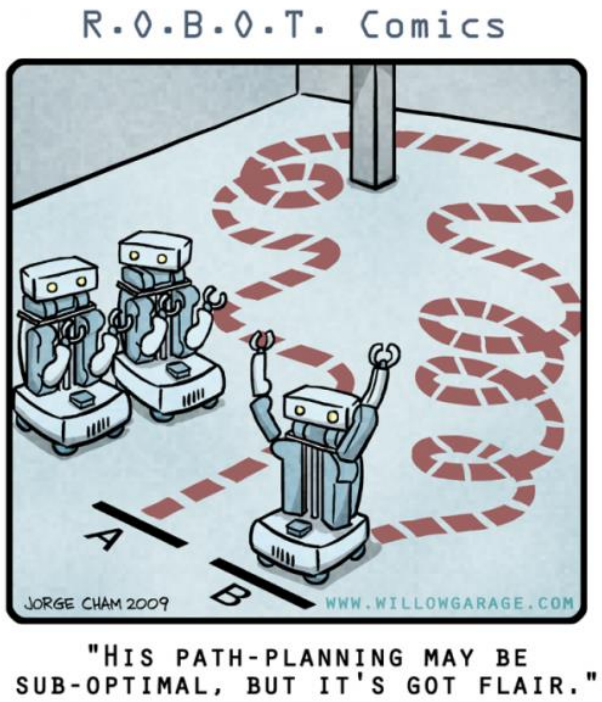
- In this example, number of iterations is 29.

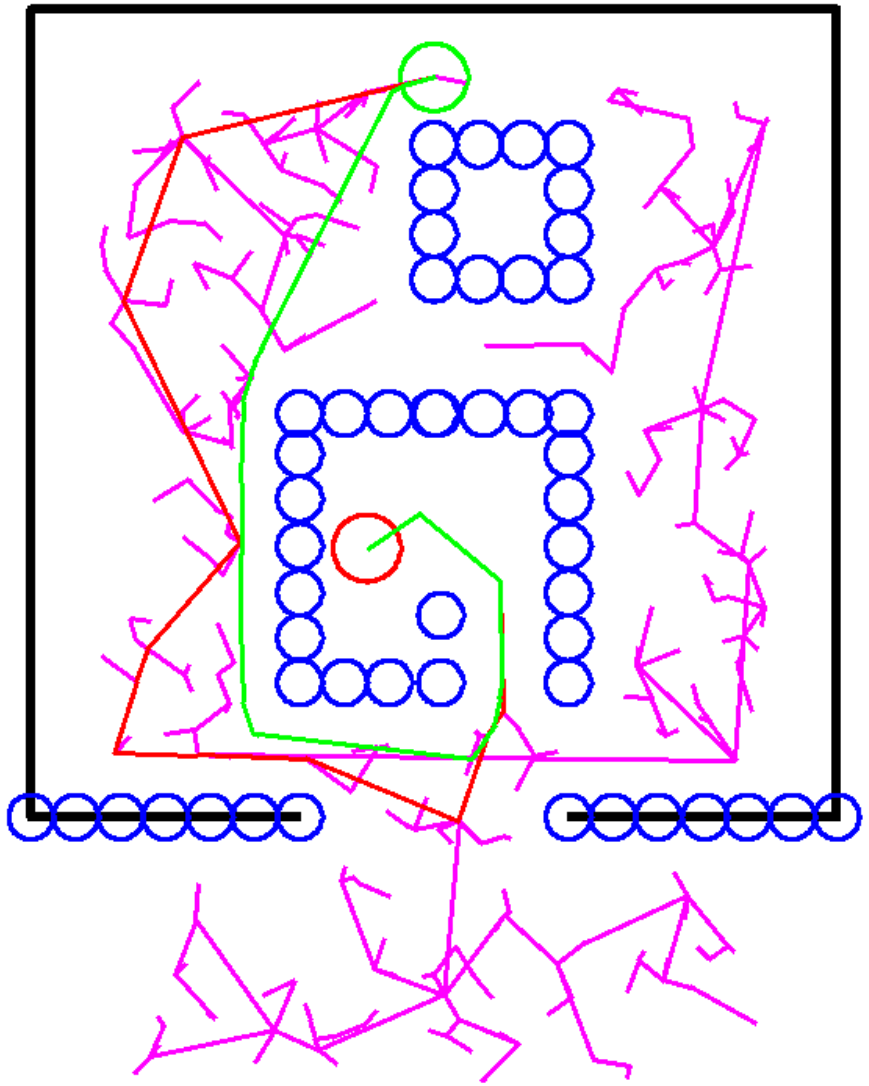
Number of iterations varies!



RRT Example

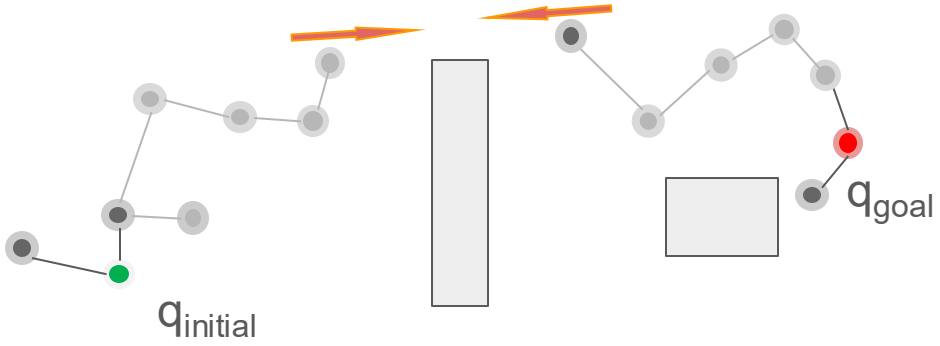
- A trap, a local minima problem encountered in the potential field.
- Number of iterations: 569.





RRT-Connect: grow and connect two RRTs

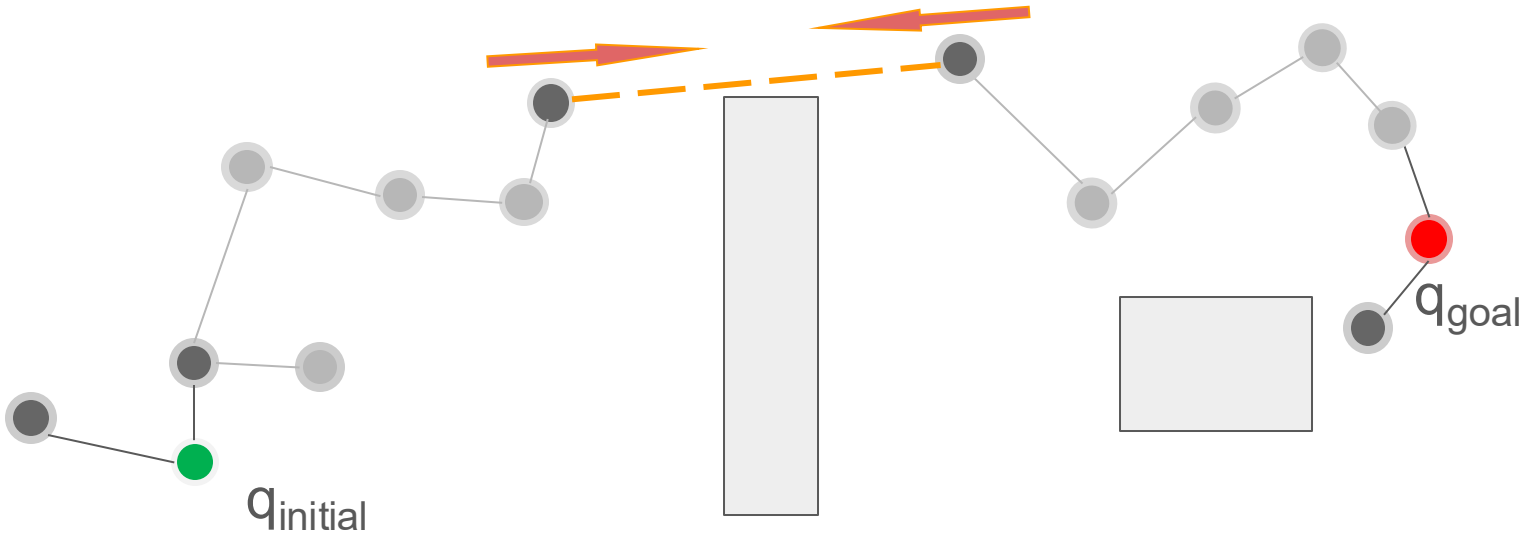
- A simple greedy heuristic that aggressively tries to connect *two trees*, one from the initial configuration and the other from the goal.



- The idea of constructing search trees from the initial and goal configurations comes from classical AI bi-directional search.

RRT-Connect

- This approach finds a graph that covers the space nicely that is independent of the query, ie RRT-Connect is still single query planning.



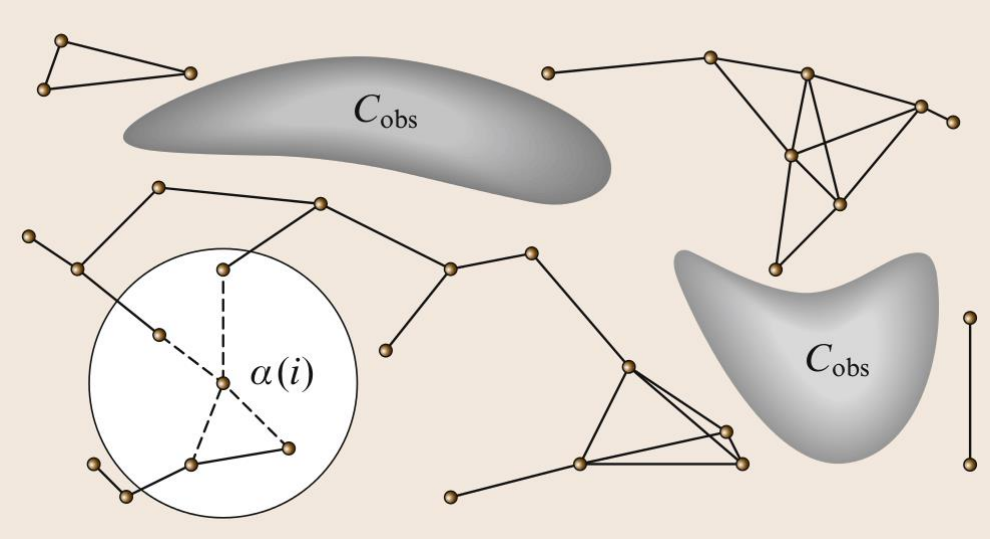
James Kuffner,, Steven LaValle, “RRT-connect: An efficient approach to single-query path planning”, IEEE International Conference on Robotics and Automation (ICRA) 2000.

Probabilistic RoadMaps (PRM)

PRM tries to encode the environment in a graph

N: number of nodes to include in the roadmap

1:G.init(); i=0;
2:while i < N do
3: if α(i) ∈ C_{free} then // is random config. collision-free
4: G.add_ vertex(α(i)); i= i + 1;
5: for q ∈ NEIGHBORHOOD(α(i),G) do
6: if CONNECT (α(i),q) then
7: G.add_ edge (α(i),q);
8: end if
9: end for
10: end if
11:end while

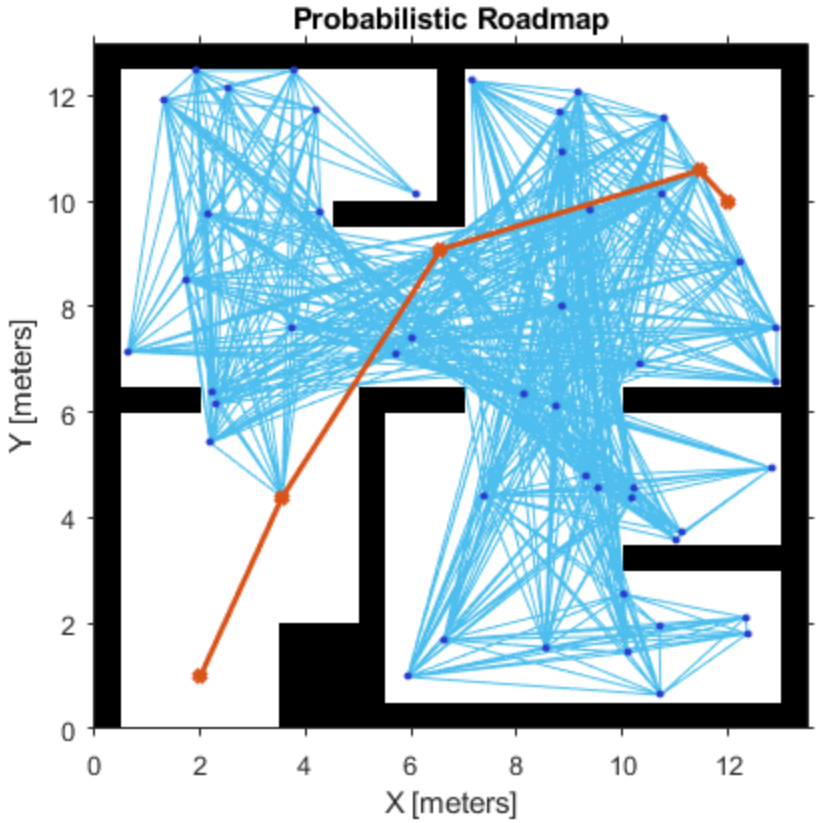


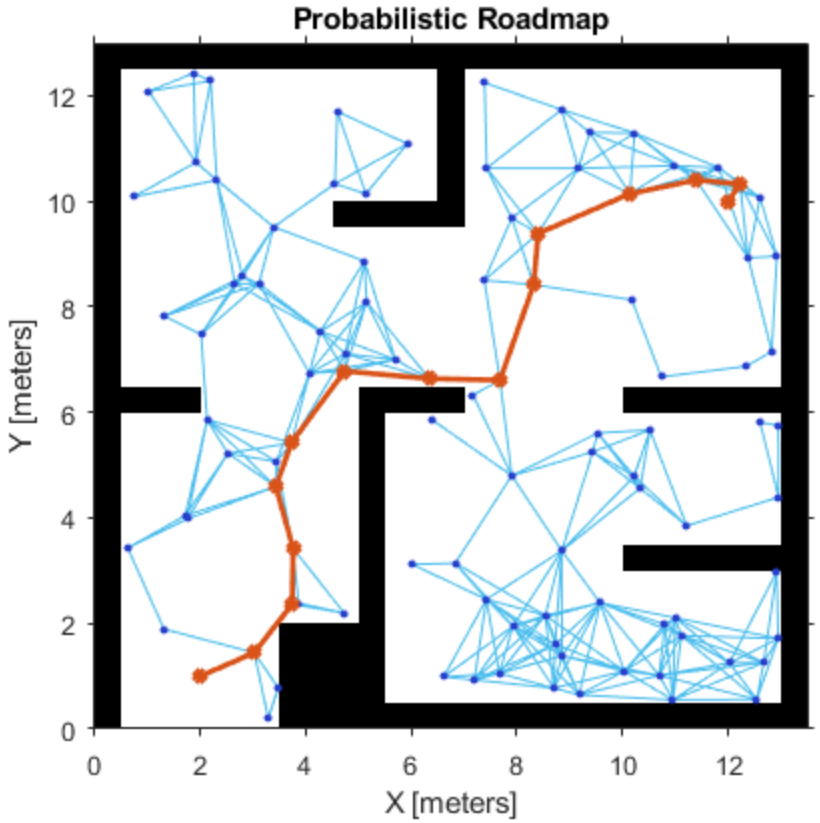
α(i) is connected to all its neighbors.

(Source: Chapter 7, Springer Handbook of Robotics ISBN 978-3-319-32552-1)

Probabilistic Roadmaps

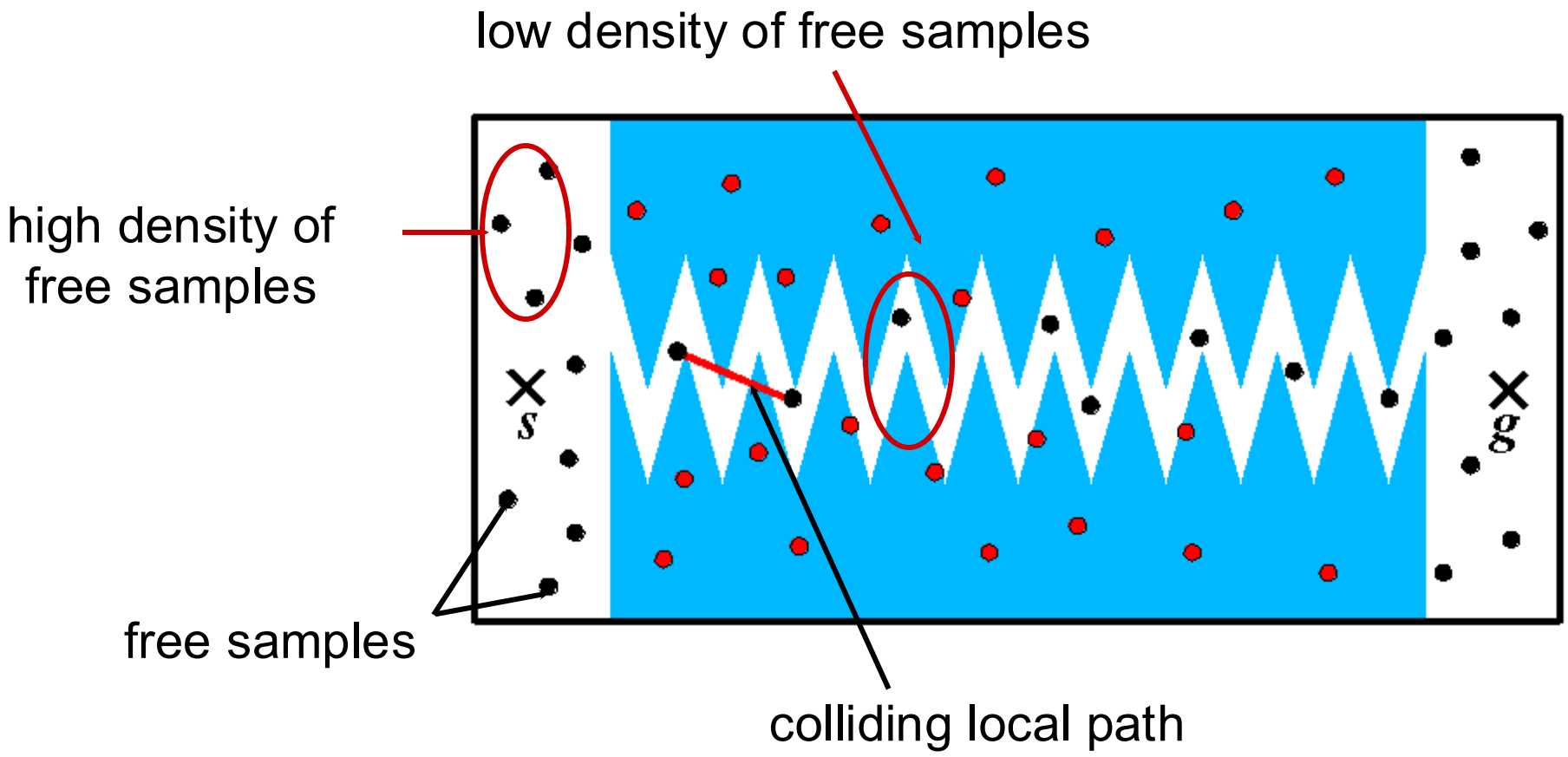
The neighborhood distance for connecting each collision-free node is a tunable parameter. A smaller connection distance reduces the number of connections, resulting in a simpler map (see more here)





More discussion

What can Go Wrong? Narrow Passages



It is difficult to capture the free space associated with the narrow passages, because in the narrow passage, volume associated with the free space is relatively low.

What Can Go Wrong? Lack of Connectivity

A combination of issues:

➢Collection of narrow passages can impede roadmap growth

➢Non-uniformity of a finite set of samples may lead to clustering – so k-nearest neighbor graph (k-NNG) connections may not cover the whole space

➢Similarly, boundary points of obstacles are sparsely sampled