Introduction to Multi-Agent Programming

4. Search Algorithms and Path-finding

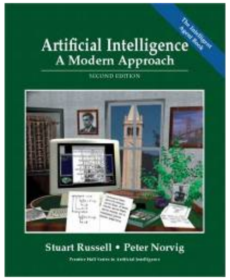
Robot Motion Planning & Multi-Robot Planning

Alexander Kleiner, Bernhard Nebel

Contents

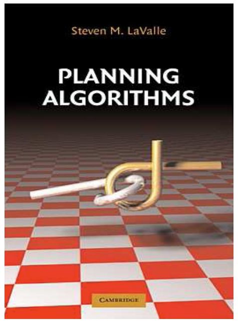
Literature

Illustrations and content presented in this lecture where taken from:



Artificial Intelligence – A Modern Approach, 2nd Edition

by Stuart Russell - Peter Norvig

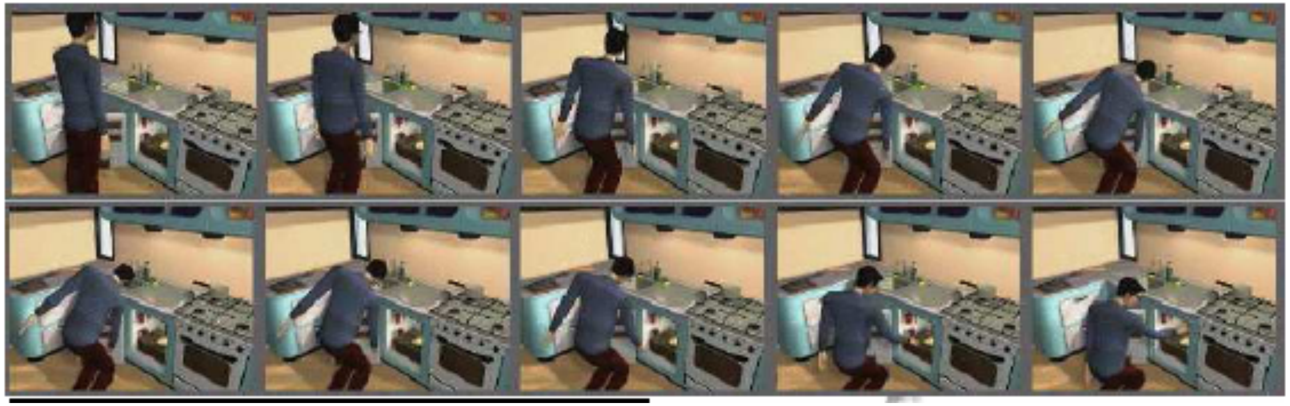


Planning Algorithms

By Steven M. LaValle

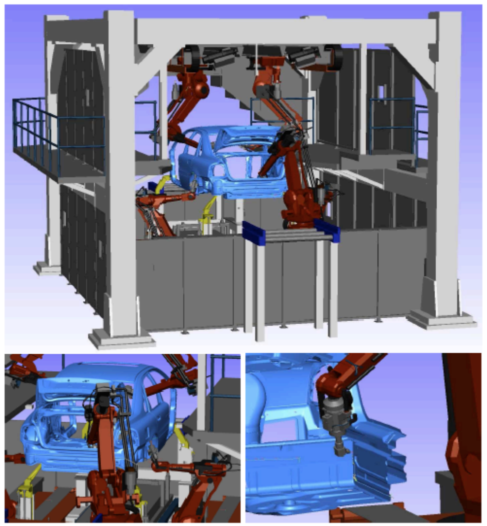
Available for downloading at: http://planning.cs.uiuc.edu/

Robot Motion Planning
Introduction



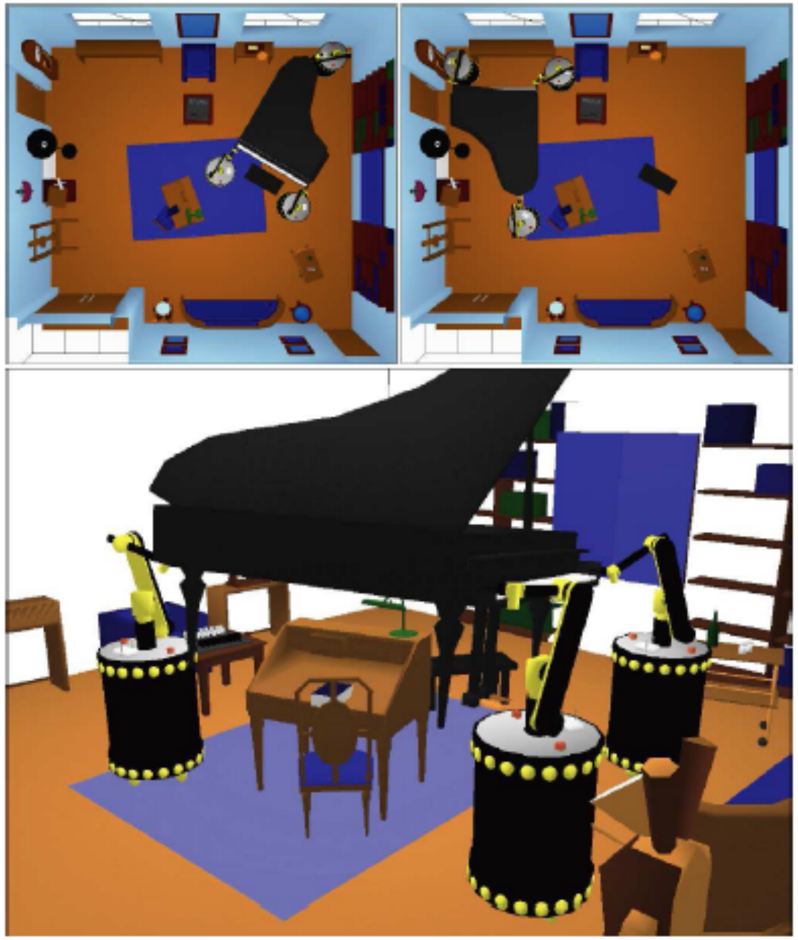
A motion computed by a planning algorithm, for a digital actor to reach into a refrigerator

A planning algorithm computes the motions of 100 digital actors moving across terrain with obstacles

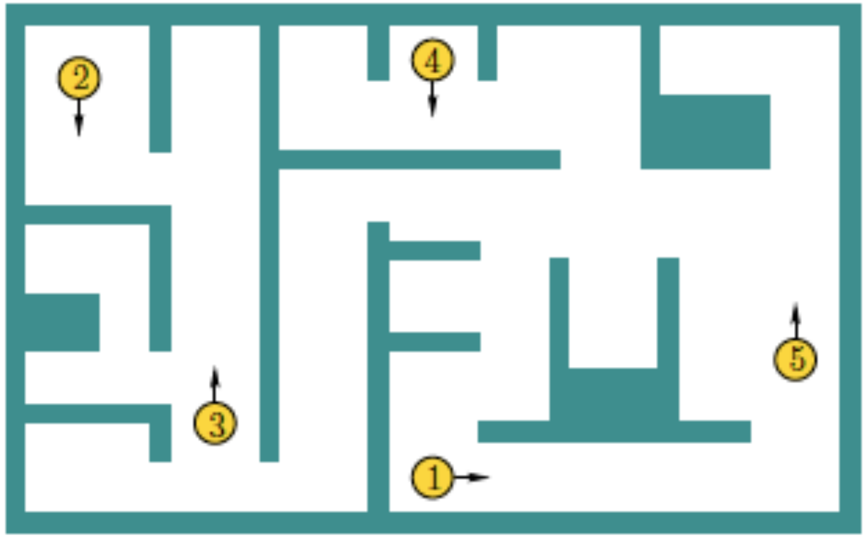


An application of motion planning to the sealing process in automotive manufacturing

Robot Motion Planning
Introduction



Using mobile robots to move a piano



Several mobile robots attempt to successfully navigate in an indoor environment while avoiding collisions with the walls and each other

Robot Motion Planning

Problem Formulation

The configuration space $\mathcal{C}$
is the space containing all **possible** configurations of the robot

Suppose world $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$

Obstacle region $\mathcal{O} \subset \mathcal{W}$

Rigid robot $\mathcal{A} \subset \mathcal{W}$

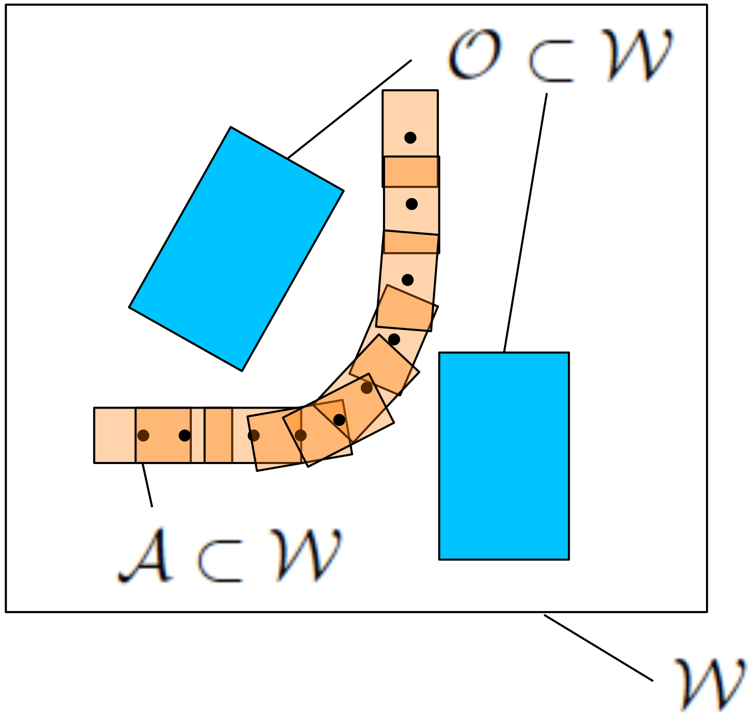
Robot configuration $q \in \mathcal{C}$

$$q = (x_t, y_t, \theta) \text{ for } \mathcal{W} = \mathbb{R}^2$$

Obstacle region $\mathcal{C}_{obs} \subseteq \mathcal{C}$ is defined by:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

Which is the set of all configurations q at which A(q), the transformed robot, intersects $\mathcal{O}$



The *free space* is defined by:

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$$

Robot Motion Planning

Problem / Solution Concepts

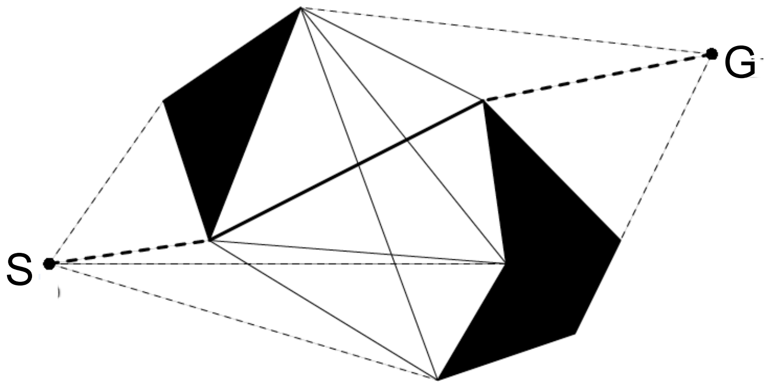
Problem: Find continuous path $\tau : [0,1] \longrightarrow C_{free}$

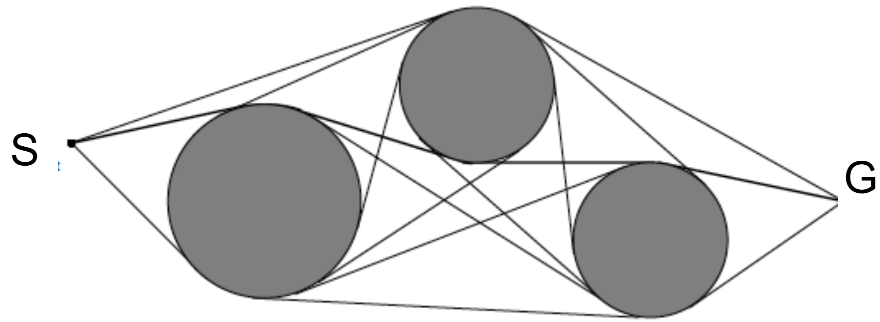
With $\tau(0) = c_{start}$ and $\tau(1) = c_{goal}$

- Requirements
  - Shortest path
  - Minimal execution time (requiring a good fit with the motion model, least amount of rotations, etc.)
  - Maximal distance to obstacles (needed in dynamic environments, and when sensors are unreliable)
- Many solution concepts:
  1. Potential Fields (more details in a later lecture)
  2. Visibility Graphs
  3. Grid-based Planning
  4. Sampling-based Planning

Robot Motion Planning
Visibility Graphs

- Approximation of obstacles as polygons
- Visibility Graph S: Build graph S=(V,G),
  - where V is the set of all vertices from the corners of polygon obstacles
  - and E the set of all visible connections between them
- Planning with discrete methods (e.g. A*)
- Simplification at RoboCup Soccer: Every obstacle is considered as a circle!
  → Edges are constructed from circle tangents
- Advantage: Depends only on number of obstacles
- Disadvantages: (1) Paths very close to obstacles (2) How to get good polygons?



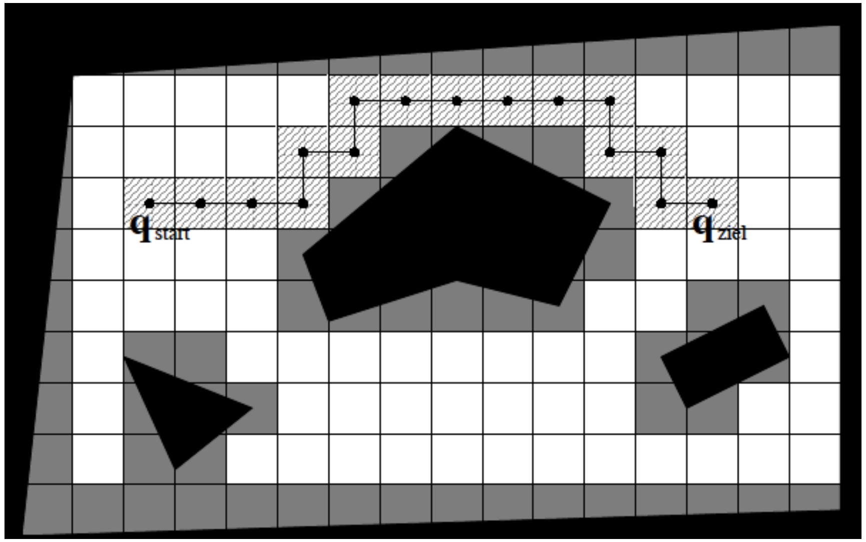


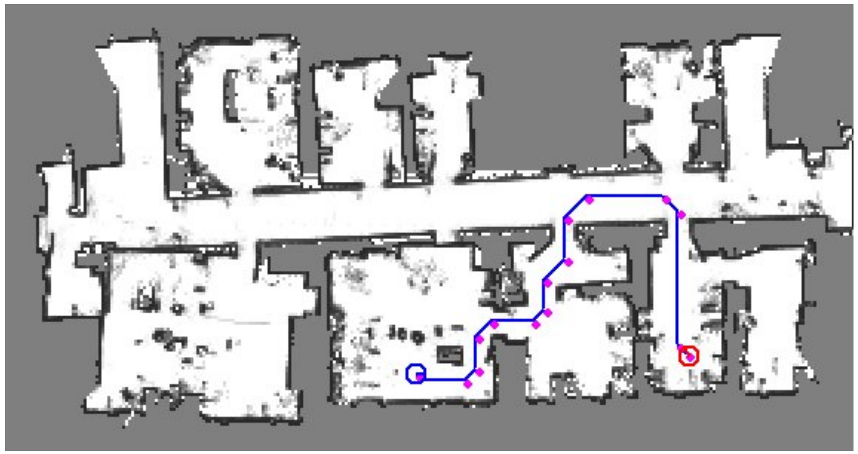
Visibility Graphs
Example

Path very close to other robots

Robot Motion Planning
Grid-based Planning

- Planning on a subdivision of C_{free} into smaller cells
- Simplification: grow borders of obstacles up to the diameter of the robot, e.g., by Gaussian blur
- Construction of graph G=(V,E), where V is the set of cells and E represents their neighbor-relations
- Planning with discrete methods (e.g. A*)
  - Resulting path is a sequence of cells
- Hierarchical planning: find path on coarse resolution and re-plan on more fine grained resolutions
- Disadvantage:
  - Memory usage grows with the size of the environment
  - Fails in narrow passages of C_{free}
- Advantage: No polygons!





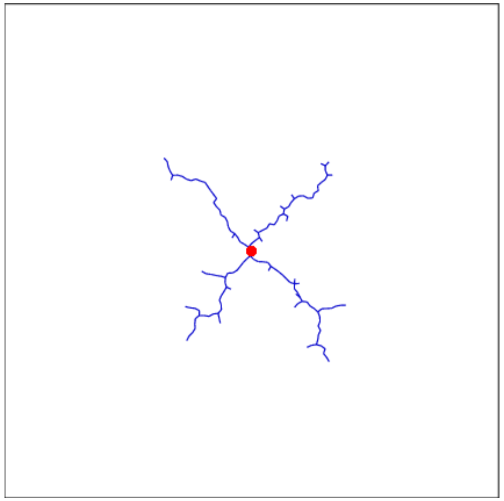
Grid-based Planning
Example

Better: path sufficiently far from other !

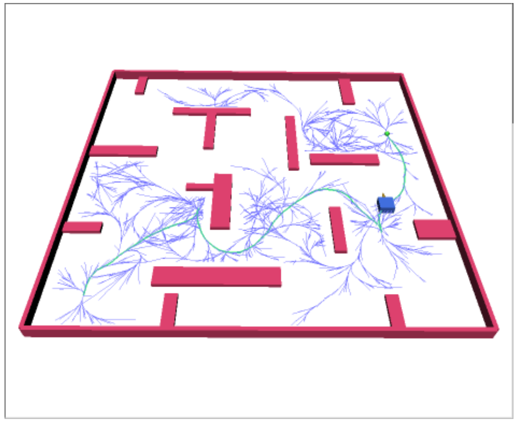
Robot Motion Planning

Sampling-based Motion Planning

- Basic Idea: To avoid explicit construction of C_{obs}
- Instead: probe C_{free} with a sampling scheme
- Builds a graph G=(V,E) by connecting sampled locations
  - each e ∈ E has to be collision free!
  - on G a solution can be found by discrete search methods (e.g. A*)
- Critical part: Random Sampling
- Time consuming part: Collision Checks



Sampling without obstacles



Sampling with obstacles

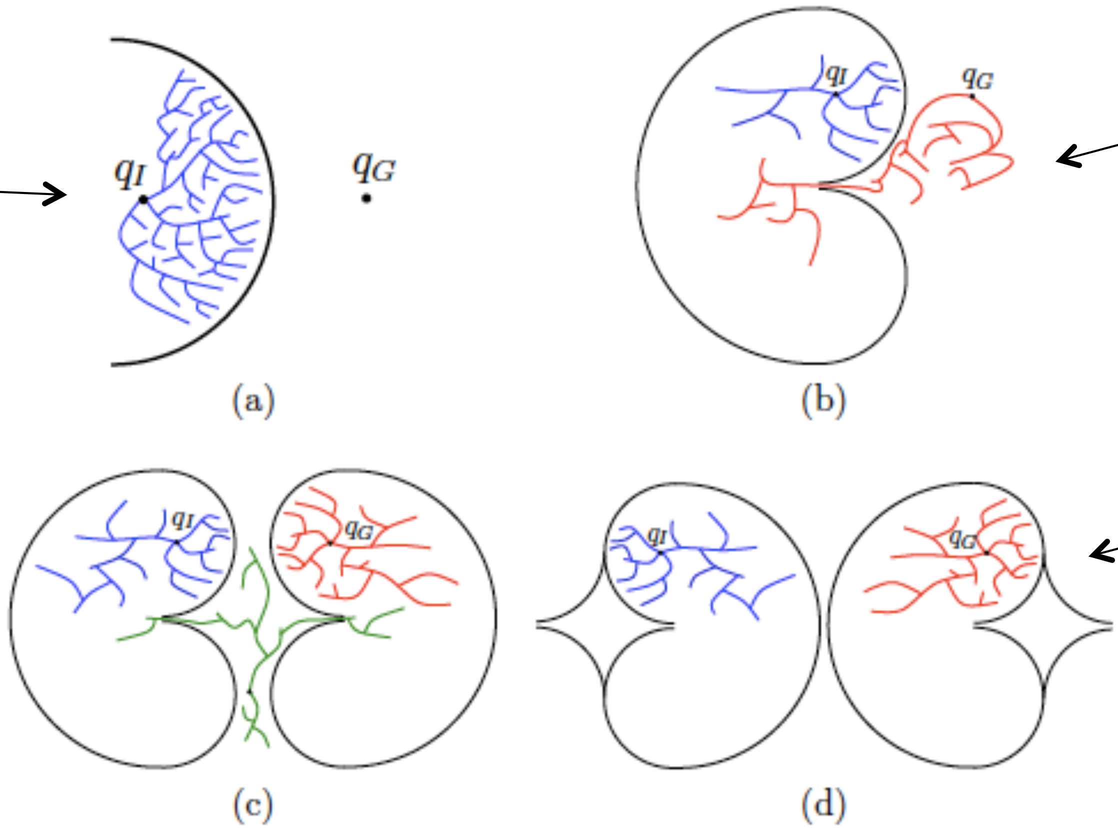
Sampling-based Motion Planning
General Procedure

1. Initialization:
   - Let G=(V,E) be an undirected search graph with $(q_{start}, q_{goal}) \in V$, $E = \varnothing$
2. Vertex Selection Method (VSM):
   - Select a vertex $q_{curr} \in V$ for expansion
3. Local Planning Method (LPM):
   - Select any $q_{new} \in C_{free}$ by sampling
   - Find a path $\tau_s : [0:1] \rightarrow C_{free}$ such that $\tau(0) = q_{curr}$ and $\tau(1) = q_{new}$
   - τ_s must be collision free, if not, go to 2)
4. Insert new Vertex & Edge in the Graph:
   - Insert q_{new} to V
   - Insert edge between q_{curr} and q_{new}
5. Check for a Solution:
   - Check if there is a valid path on G from q_{start} to q_{goal}, if yes: terminate
6. Return to step 2) until any termination criterion is met

Sampling-based Motion Planning
Difficulties

Multi-resolution search required to quickly overcome cavities

Bidirectional search needed in some cases



Sometimes even multi-dimensional search needed

Hard to solve even with multi-dimensional search

Sampling-based Motion Planning

Random Sampling / Deterministic Sampling

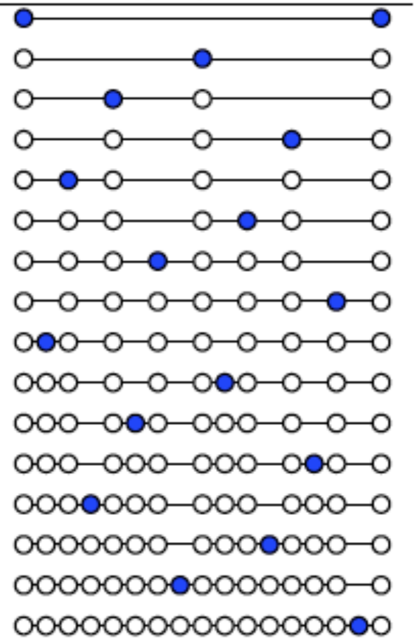
- A Sampling sequence should reach every point in C! However, C is uncountably infinite …
- In practice, sampling has to terminate early. Hence the sequence of sampling matters!
- Dense Sequence: A sequence getting with increasing size arbitrarily close to every element in C
- Random sampling:
  - Suppose C=[0,1] and I ⊂ C is an interval of length *e*. If *k* samples are chosen independently at random, the probability that none of them falls into I is $(1-e)^k$. As k approaches infinity, this probability converges to zero. This means random sampling is probably dense.
- Deterministic sampling:
  - Suppose C=[0,1] and we want to place 16 samples
  - Simple approach:
    - Select the set S={i/16 | 0<i<16} so that all samples are evenly distributed
  - What if we want to make S into a sequence? What is the best ordering? What if 16 points are not enough, i.e., are not reaching every interesting point in C?
  - Problem with “sorting by increasing value”: after i=8 half of C has been neglected! It would be preferable to have a nice covering of C for every i

Sampling-based Motion Planning

The Van der Corput sequence

- Idea: to reverse the order of the bits, when the sequence is represented with binary decimals
- By reversing the bits, the most significant bit toggles in every step, which means that the sequence alternates between the lower and upper halves of C

i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	



Sequence for i<=16

Note: Both method can also be applied for $C \subseteq \Re^m$ by sampling each dimension independently

Sampling-based Motion Planning

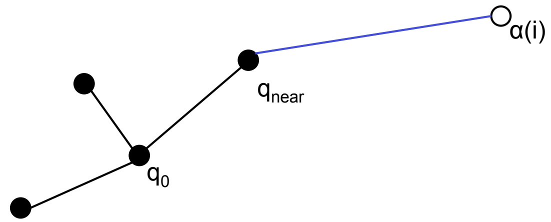
Rapidly Exploring Dense Trees (RDTs)

Basic algorithm for RDTs
(without obstacles):

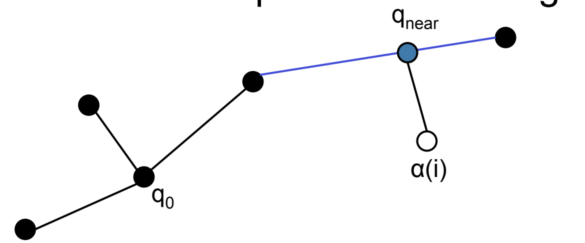
```
SIMPLE_RDT(q_0)
  G.init(q_0);
  for i = 1 to k do
      G.add_vertex(α(i));
      q_n ← NEAREST(S(G), α(i));
      G.add_edge(q_n, α(i));
```

- Requires a dense sequence α(i)
- Let S(G) be the set of all points reached by G (either vertices or edges)
- Connects iteratively edges from α(i) to those nearest in G

Case 1: Nearest point is a vertex

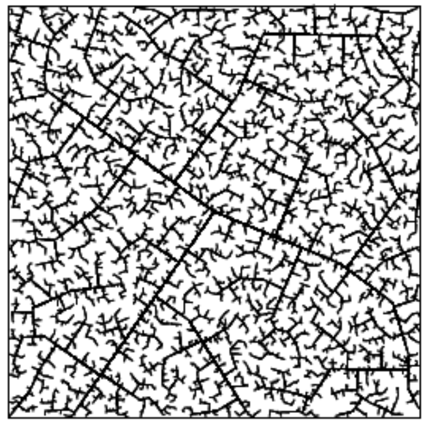


Case 2: Nearest point is on an edge



Result:


45 iterations


2345 iterations

Sampling-based Motion Planning

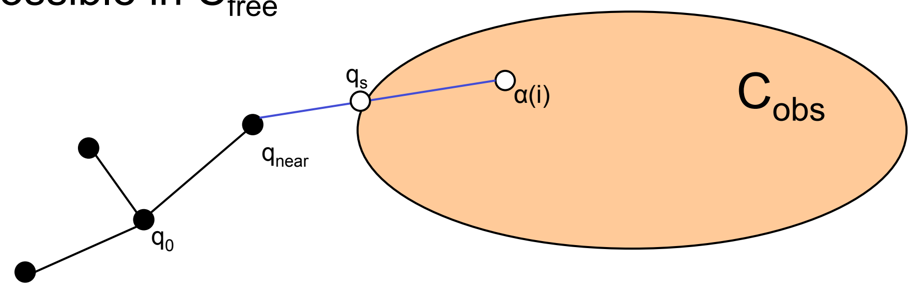
Rapidly Exploring Dense Trees (RDTs)

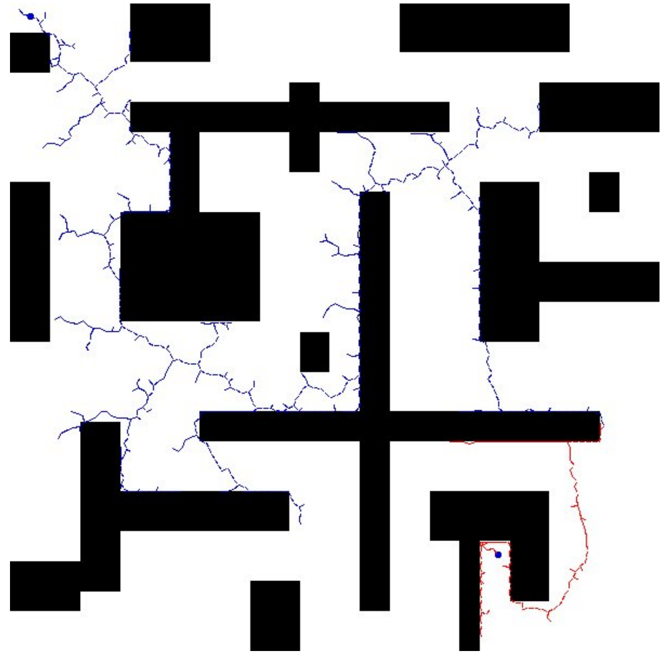
Basic algorithm for RDTs (with obstacles):

RDT(q_0)

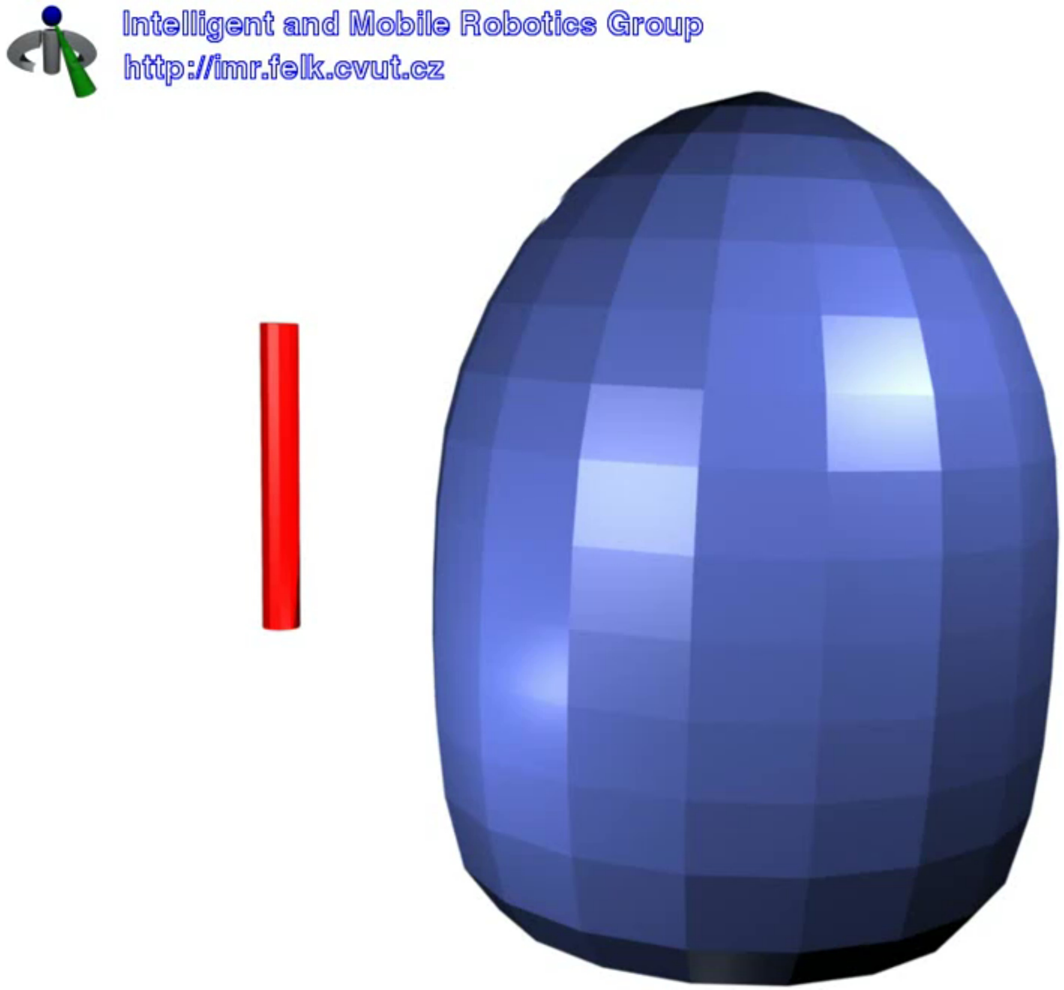
 $\mathcal{G}$.init(q_0);
 for $i = 1$ **to** k **do**
  $q_n \leftarrow$ NEAREST($S, \alpha(i)$);
  $q_s \leftarrow$ STOPPING-CONFIGURATION(q_n,$\alpha(i)$);
  if $q_s \neq q_n$ **then**
   $\mathcal{G}$.add_vertex(q_s);
   $\mathcal{G}$.add_edge(q_n, q_s);

- STOPPING-CONFIGURATION() returns the nearest configuration possible in C_{free}





Bug trap video on YouTube



http://www.youtube.com/watch?v=qci_AktcrD4

Multi Robot Planning

Problem

- So far, we considered problems with single agents in static environments
- When multiple robots plan and navigate at the same time, robot-robot collisions might occur

- Obvious solution: To use a central planner that plans trajectories for all robots simultaneously

Multi Robot Planning
Problem Formulation

1. World $\mathcal{W}$ and obstacle region $\mathcal{O}$

2. There are *m* robots: $\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^m$

3. Each robot A_i has both initial and goal configuration q^i_{init} q^i_{goal}

4. The state space considers configuration of all robots simultaneously:

$$X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m$$

C is he configuration space

A state $x \in X$ specifies a combination of all robot configurations and my be expressed as:

$$x = (q^1, q^2, \ldots, q^m)$$

The dimension of X is N, which is: $N = \sum_{i=1}^{m} \dim(\mathcal{C}^i)$

Multi Robot Planning
Problem Formulation

Obstacle region 1: robot-obstacle (walls, etc.);

$$X^i_{obs} = \{x \in X \mid \mathcal{A}^i(q^i) \cap \mathcal{O} \neq \emptyset\}$$

Obstacle region 2: robot-robot:

$$X^{ij}_{obs} = \{x \in X \mid \mathcal{A}^i(q^i) \cap \mathcal{A}^j(q^j) \neq \emptyset\}$$

Entire obstacle region:

$$X_{obs} = \left(\bigcup_{i=1}^{m} X^i_{obs} \right) \bigcup \left(\bigcup_{ij,\ i \neq j} X^{ij}_{obs} \right)$$

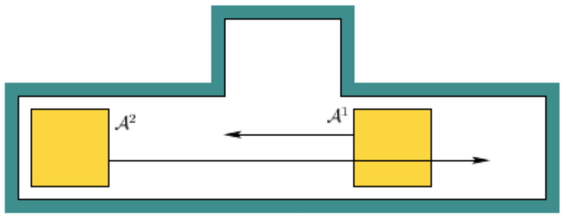
5. Initial state: $x_I \in X_{free}$ with $x_I = (q^1_I, \ldots, q^m_I)$
6. Goal state: $x_G \in X_{free}$ with $x_G = (q^1_G, \ldots, q^m_G)$
7. Compute $\tau : [0,1] \rightarrow X_{free}$ such that $\tau(0) = x_{init}$, $\tau(1) \in x_{goal}$

Multi Robot Planning
Complexity

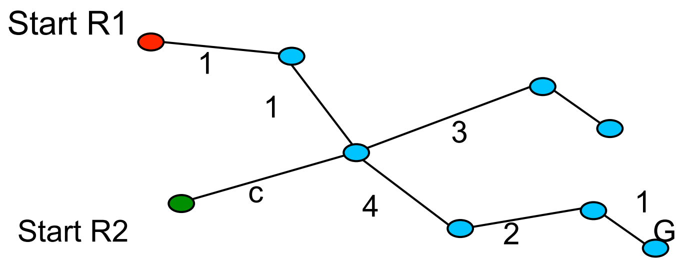
- X can be considered as an ordinary C space (and all the methods we learned can be applied)
  - However, the dimension of X grows linearly with the number of robots!
  - Complete planning algorithms require time that is at least exponential in the dimension of X!
  - Sample-based methods are more likely to scale well in practice when there many robots, but the dimension of X might still be too high

Decoupled planning

- Decoupled approaches
  - search first for motion plans for each single robot while ignoring plans of other robots
  - Solve then occurring conflicts by different strategies, e.g., stop, drive back, driver slower, …
- Prioritized Planning:
  - Straightforward approach that sorts robots by priority and plans for higher priority robots first
  - Lower priority robots plan by viewing the higher priority robots as obstacles (but in time-space!)
- Incomplete: Planning can fail to find a valid multi-robot path although there exists one!



If A^1 ignores the plan of A^2, then completeness is lost when using the prioritized planning approach.



Planning in time-space Only when c=2 conflict

Prioritized Planning

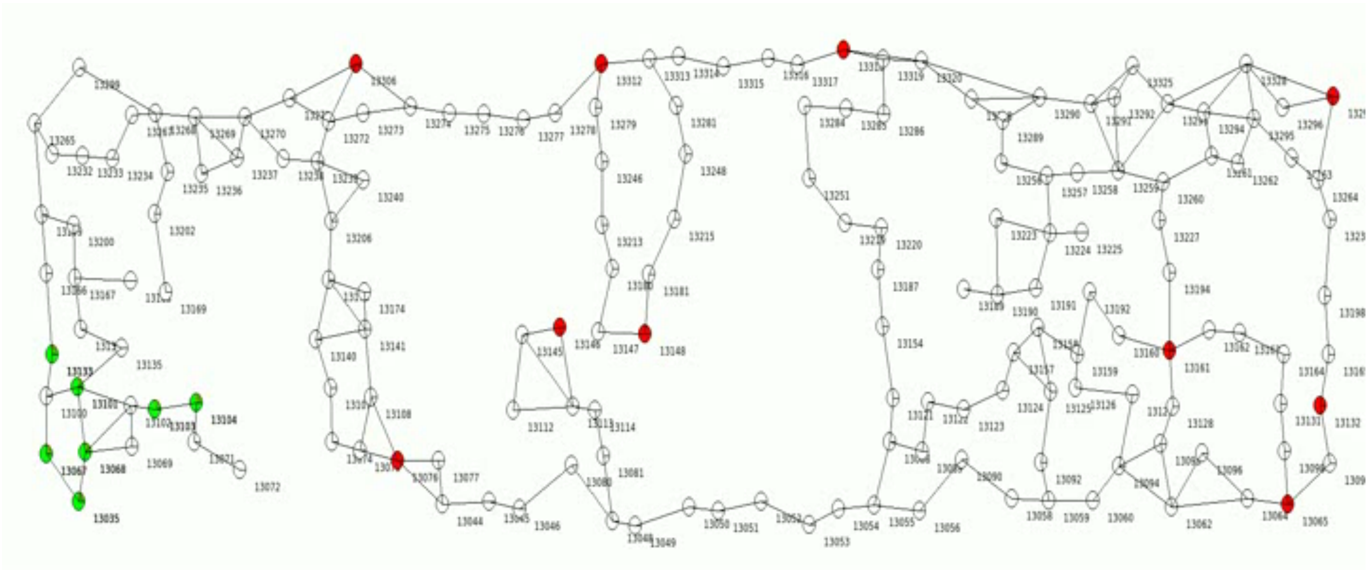
Optimizing priority schemes

- Idea: to interleave the search for collision-free paths with the search for a solvable priority scheme
- Randomized hill-climbing search
- Randomly flips priorities of two robots (max_flips)
- Performs random restarts to avoid local minima (max_tries)
- Any-time algorithm!
- Can also be implemented as a Genetic Algorithm

```
for tries := 1 to MAX_TRIES do
    select random order Π
    if (tries = 1) then
        Π*:=Π
    for flips := 1 to MAX_FLIPS do
        chose random i,j with i<j
        Π':= swap(i,j,Π)
        If cost(Π') < cost(Π) then
            Π:=Π'
    end for;
    If cost(Π) < cost(Π*) then
            Π*:=Π
end for;
return Π*
```

M. Benewitz et al.2001

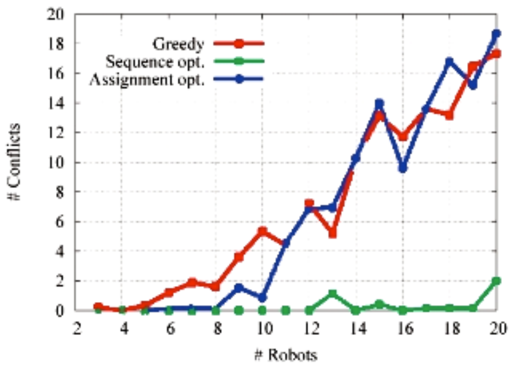
Prioritized Planning
Performance of prioritized schemes



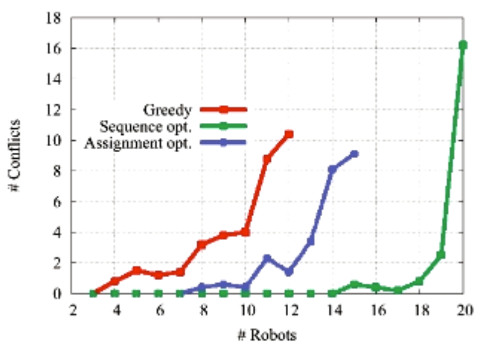
Goal nodes
Robot start nodes
Multi-robot plan

Conflicts vs. # of robots: **Greedy** (red), **optimizing priorities** (green)

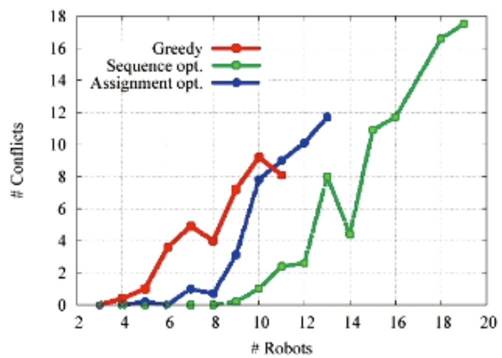
Greedy Priorities: The shorter the path the higher the priority



Map 1



Map 2



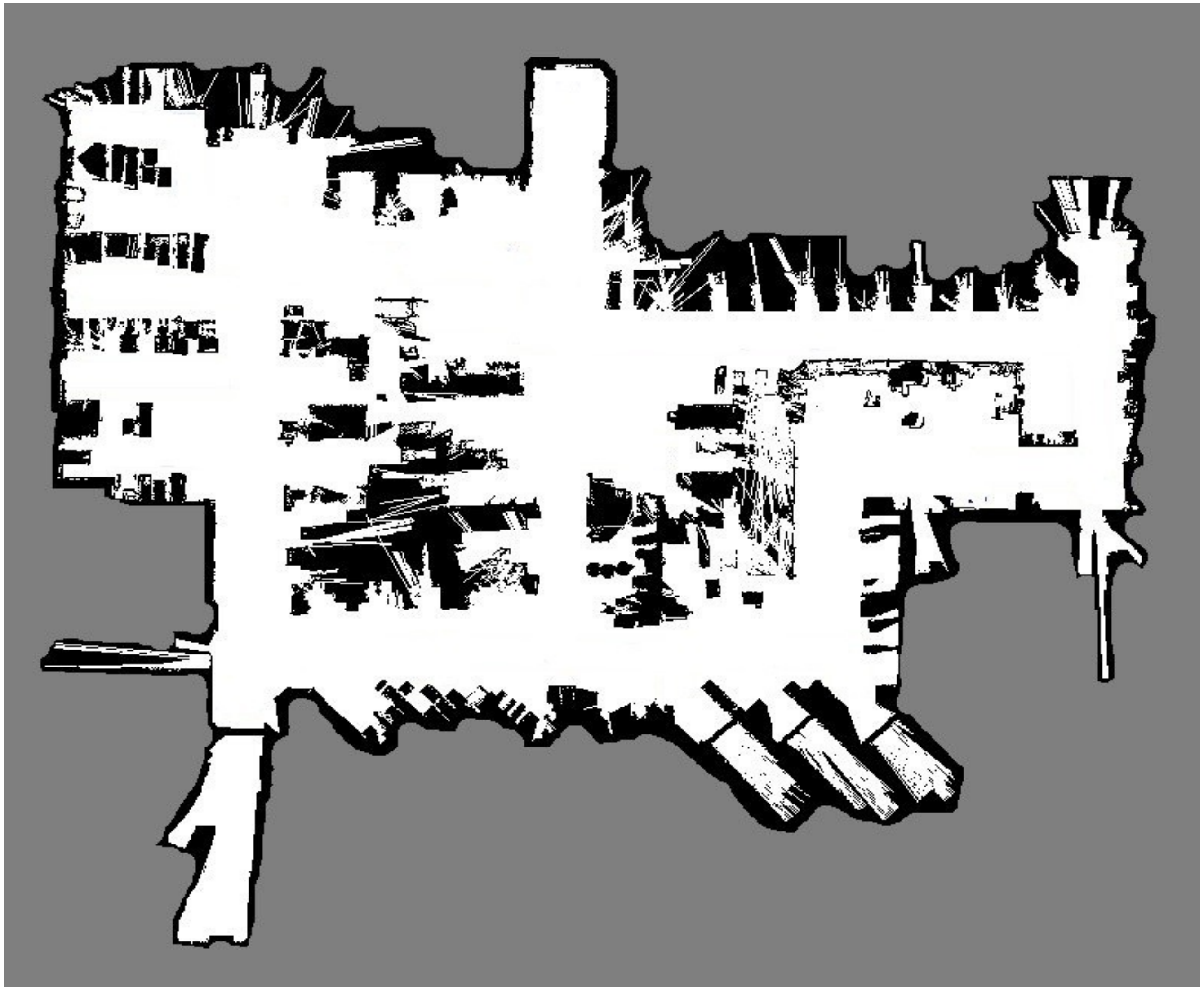
Map 3

Multi-Robot Path Planning
Road-Map based approaches

- Automatic generation of road maps (offline)
- A road map consists of streets and street crossings
- On lanes vehicles drive in convoys
- Robots are coordinated at street crossings, otherwise convoy driving
  - Right before left
  - Robot priorities according to cargo
  - Etc.
- Automatic generation by computing Voronoi diagrams
- Street merging by Linear Programming (LP)
- Complete solution (but suboptimal in terms of distance)

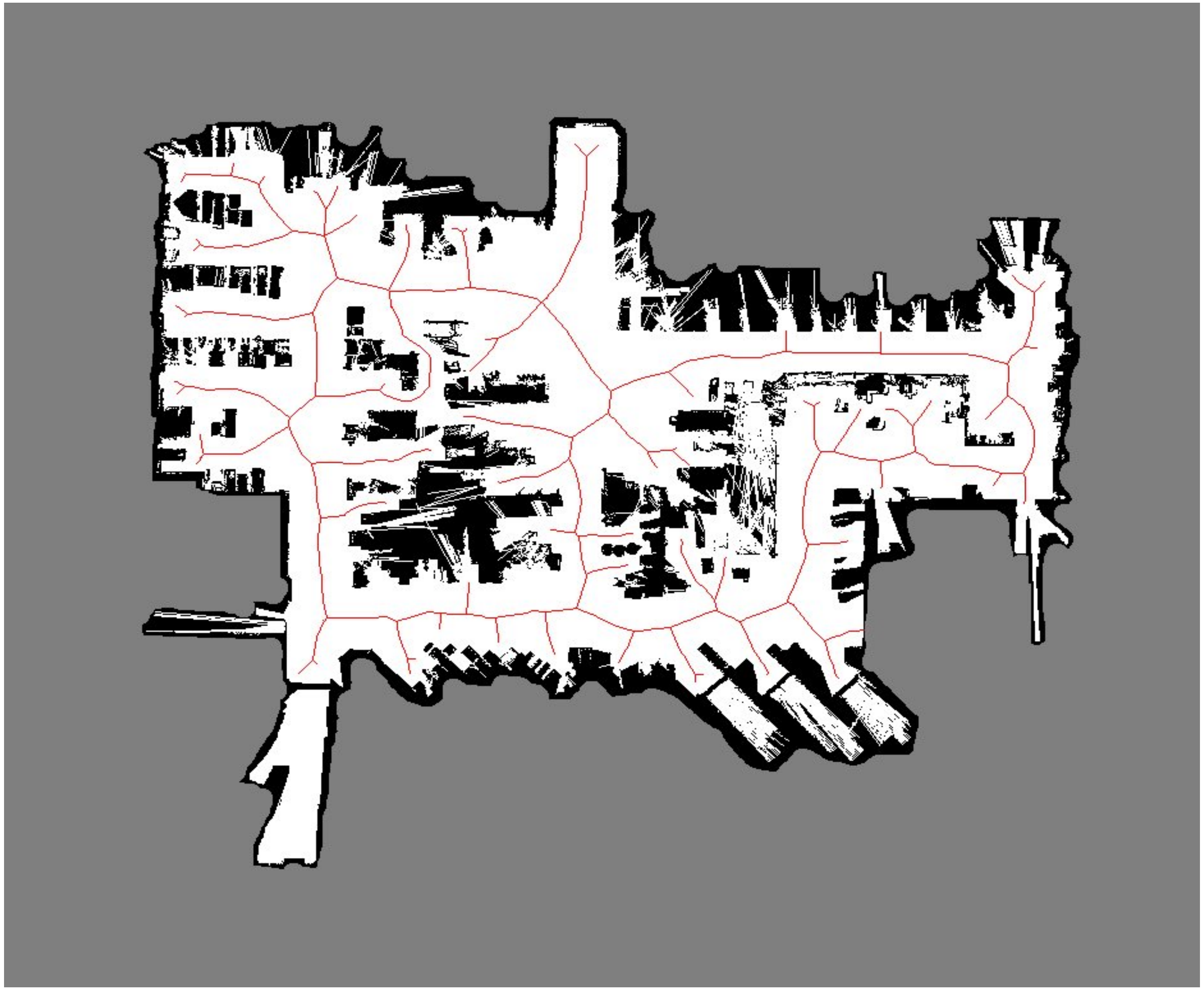
Automatic Road Map Generation

Input: Grid Map



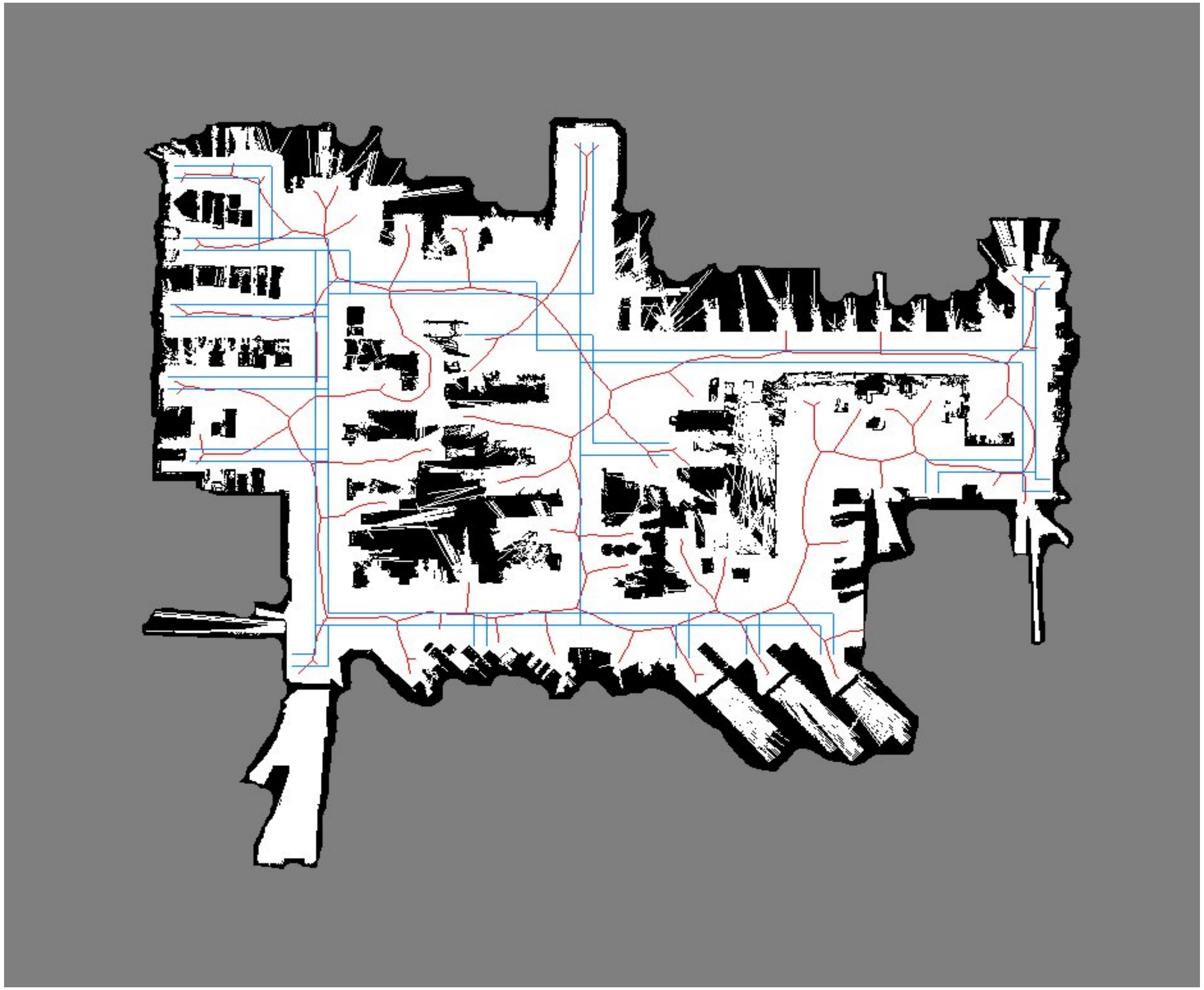
Automatic Road Map Generation

Voronoi Graph: Automatic Detection of drivable areas



Automatic Road Map Generation

Constraint solver: Generating the road map



Automatic Road Map Generation

Final Result: Road map with lanes & crossings

Summary

- Sampling-based Planning methods are well suited for Robot Motion Planning
- To find complete solutions in multi-Robot Planning is generally intractable
- However, solutions sufficient for many problem domains can be found by decoupled techniques
- Complete solutions can be found by Road-Map planners