



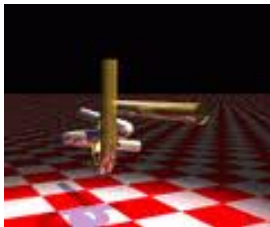
# Robotics

## Path Planning

*Path finding vs. trajectory optimization, local vs. global, Dijkstra, Probabilistic Roadmaps, Rapidly Exploring Random Trees, non-holonomic systems, car system equation, path-finding for non-holonomic systems, control-based sampling, Dubins curves*

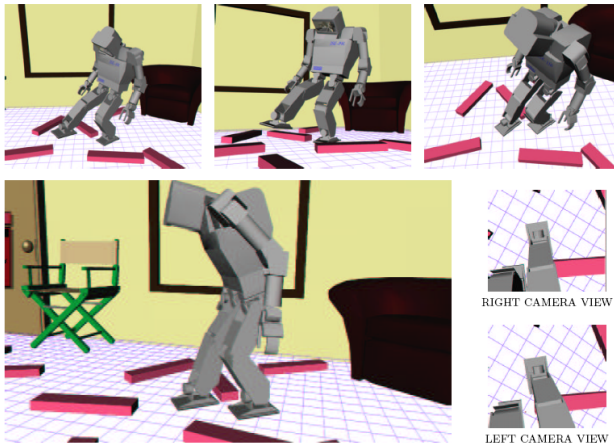
Marc Toussaint  
U Stuttgart

## Path finding examples



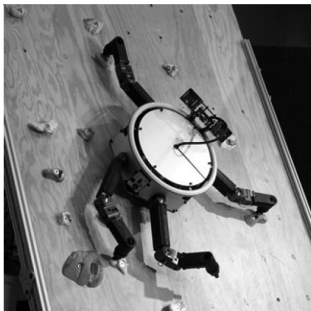
Alpha-Puzzle, solved with James Kuffner's RRTs

# Path finding examples



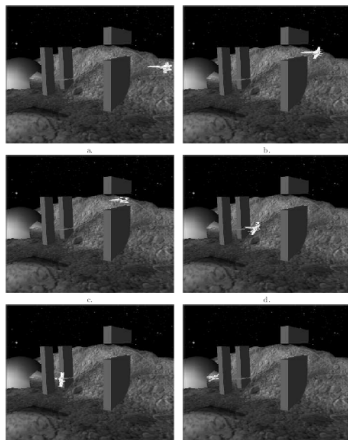
J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep Planning Among Obstacles for Biped Robots. Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2001.

## Path finding examples



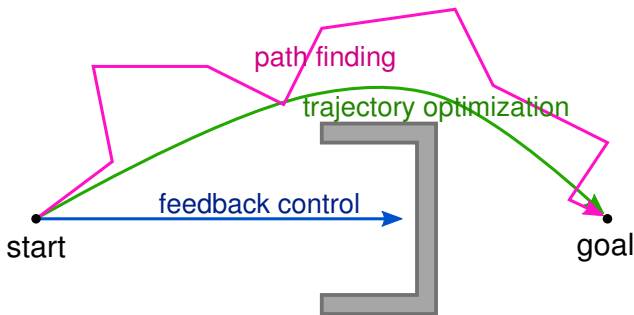
T. Bretl. Motion Planning of Multi-Limbed Robots Subject to Equilibrium Constraints: The Free-Climbing Robot Problem. International Journal of Robotics Research, 25(4):317-342, Apr 2006.

# Path finding examples



S. M. LaValle and J. J. Kuffner. Randomized Kinodynamic Planning.  
International Journal of Robotics Research, 20(5):378–400, May 2001.

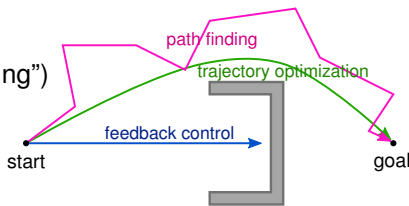
# Feedback control, path finding, trajectory optim.



- Feedback Control: E.g.,  $q_{t+1} = q_t + J^\sharp(y^* - \phi(q_t))$
- Trajectory Optimization:  $\operatorname{argmin}_{q_{0:T}} f(q_{0:T})$
- Path Finding: Find some  $q_{0:T}$  with only valid configurations

# Control, path finding, trajectory optimization

- Combining methods:
  - 1) Path Finding
  - 2) Trajectory Optimization (“smoothing”)
  - 3) Feedback Control



- Many problems can be solved with only feedback control (though not optimally)
- Many more problems can be solved *locally* optimal with only trajectory optimization
- Tricky problems need path finding: *global* search for valid paths

# Outline

- Heuristics & Discretization (slides from Howie CHoset's CMU lectures)
  - Bugs algorithm
  - Potentials to guide feedback control
  - Dijkstra
- Sample-based Path Finding
  - Probabilistic Roadmaps
  - Rapidly Exploring Random Trees

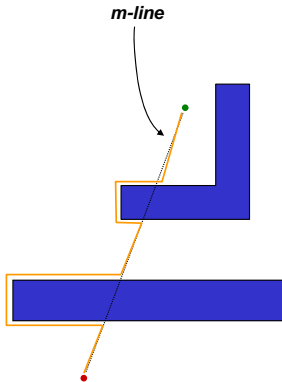


# A better bug?



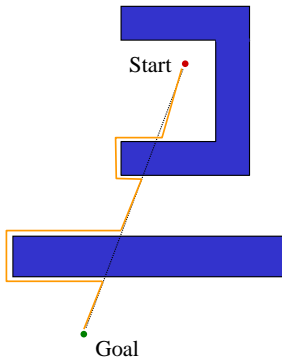
## "Bug 2" Algorithm

- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal



# A better bug?

## "Bug 2" Algorithm



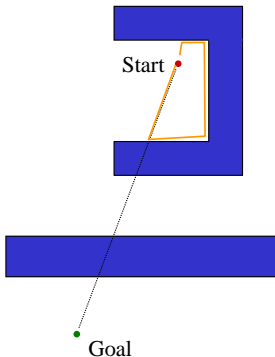
- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal

Better or worse than Bug1?



# A better bug?

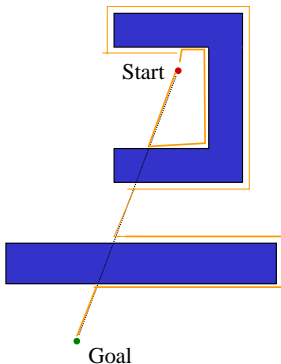
## "Bug 2" Algorithm



- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the *m-line* again.
- 3) Leave the obstacle and continue toward the goal

# A better bug?

## "Bug 2" Algorithm



- 1) head toward goal on the *m-line*
- 2) if an obstacle is in the way, follow it until you encounter the m-line again *closer to the goal*.
- 3) Leave the obstacle and continue toward the goal

Better or worse than Bug1?

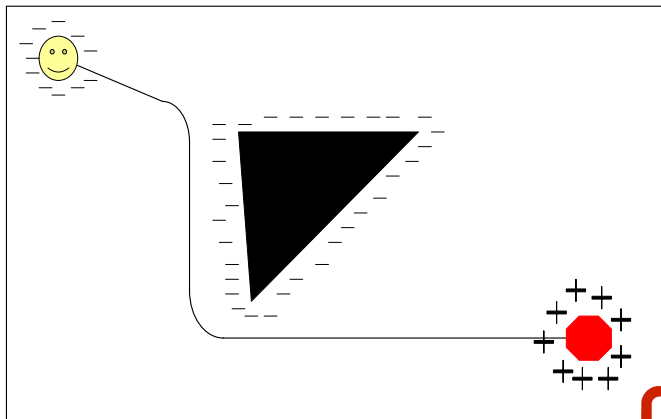


## BUG algorithms – conclusions

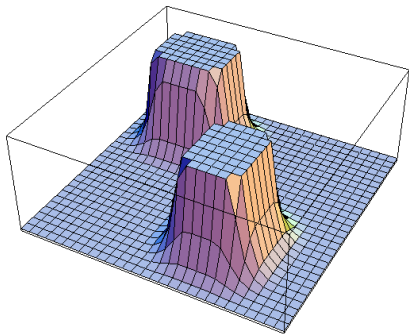
- Other variants: TangentBug, VisBug, RoverBug, WedgeBug, ...
- only 2D! (TangentBug has extension to 3D)
- Guaranteed convergence
- Still active research:  
K. Taylor and S.M. LaValle: *I-Bug: An Intensity-Based Bug Algorithm*

⇒ Useful for minimalistic, robust 2D goal reaching  
– not useful for finding paths in joint space

# Start-Goal Algorithm: Potential Functions



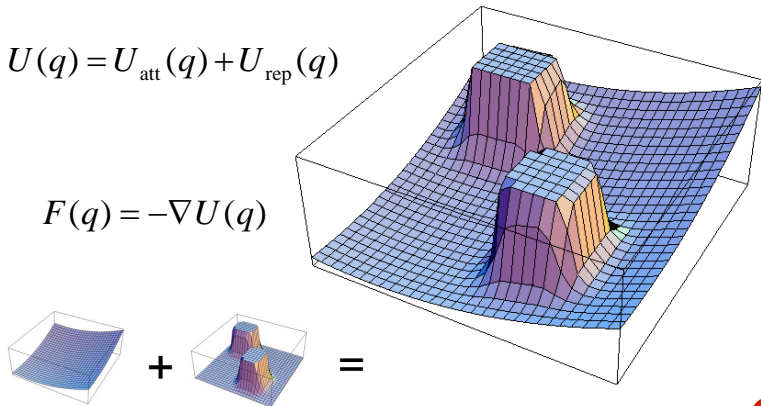
# Repulsive Potential



# Total Potential Function

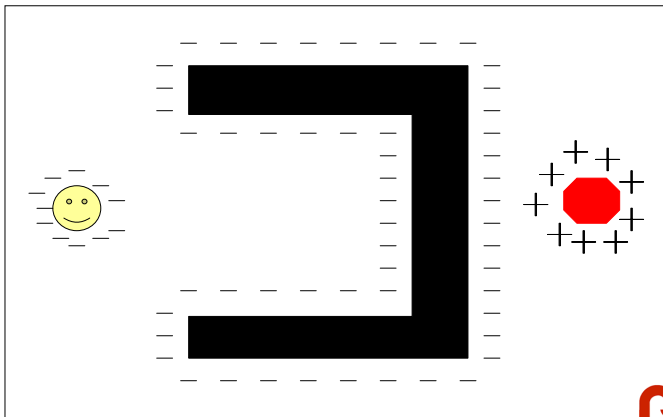
$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

$$F(q) = -\nabla U(q)$$





## Local Minimum Problem with the Charge Analogy



## Potential fields – conclusions

- Very simple, therefore popular
- In our framework: Combining a goal (endeffector) task variable, with a constraint (collision avoidance) task variable; then using inv. kinematics is *exactly* the same as “Potential Fields”

⇒ Does not solve locality problem of feedback control.

## The Wavefront in Action (Part 2)

- Now repeat with the modified cells
  - This will be repeated until no 0's are adjacent to cells with values  $\geq 2$
  - 0's will only remain when regions are unreachable

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

# The Wavefront in Action (Part 1)

- Starting with the goal, set all adjacent cells with “0” to the current cell + 1
  - 4-Point Connectivity or 8-Point Connectivity?
  - Your Choice We'll use 8-Point Connectivity in our example

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

## The Wavefront in Action (Part 2)

- Now repeat with the modified cells
  - This will be repeated until no 0's are adjacent to cells with values  $\geq 2$
  - 0's will only remain when regions are unreachable

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

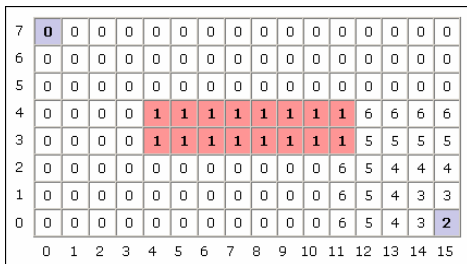
# The Wavefront in Action (Part 3)

- Repeat again...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

# The Wavefront in Action (Part 4)

- And again...



# The Wavefront in Action (Part 5)

- And again until...

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	7	7	7	7
4	0	0	0	0	1	1	1	1	1	1	1	1	6	6	6
3	0	0	0	0	1	1	1	1	1	1	1	1	5	5	5
2	0	0	0	0	0	0	0	0	0	0	7	6	5	4	4
1	0	0	0	0	0	0	0	0	0	0	7	6	5	4	3
0	0	0	0	0	0	0	0	0	0	0	7	6	5	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



# The Wavefront in Action (Done)

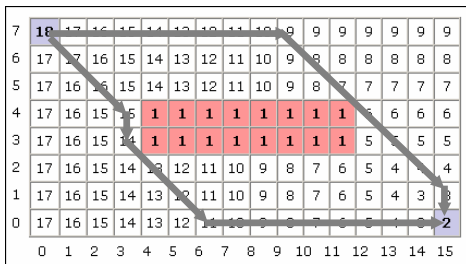
- You're done
  - Remember, 0's should only remain if unreachable regions exist

7	18	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	1	1	1	1	1	1	1	1	6	6	6	6
3	17	16	15	14	1	1	1	1	1	1	1	1	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

# The Wavefront, Now What?

- To find the shortest path, according to your metric, simply always move toward a cell with a lower number
  - The numbers generated by the Wavefront planner are roughly proportional to their distance from the goal

Two  
possible  
shortest  
paths  
shown

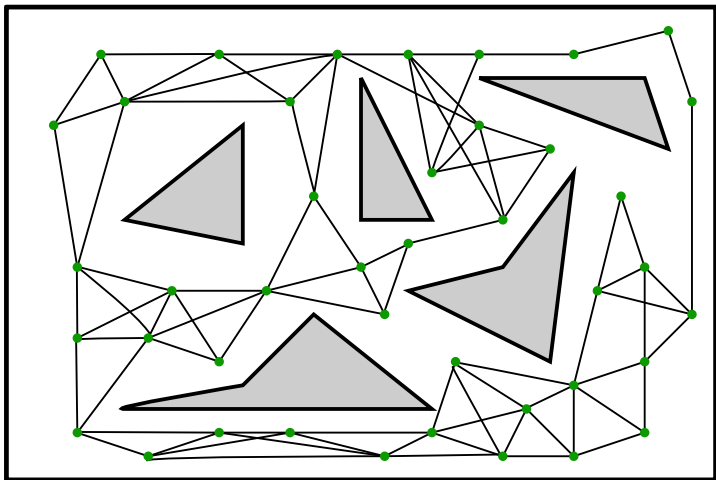


# Dijkstra Algorithm

- Is efficient in **discrete domains**
  - Given start and goal node in an arbitrary graph
  - Incrementally label nodes with their distance-from-start
- Produces optimal (shortest) paths
- Applying this to continuous domains requires discretization
  - Grid-like discretization in high-dimensions is daunting! (*curse of dimensionality*)
  - What are other ways to “discretize” space more efficiently?

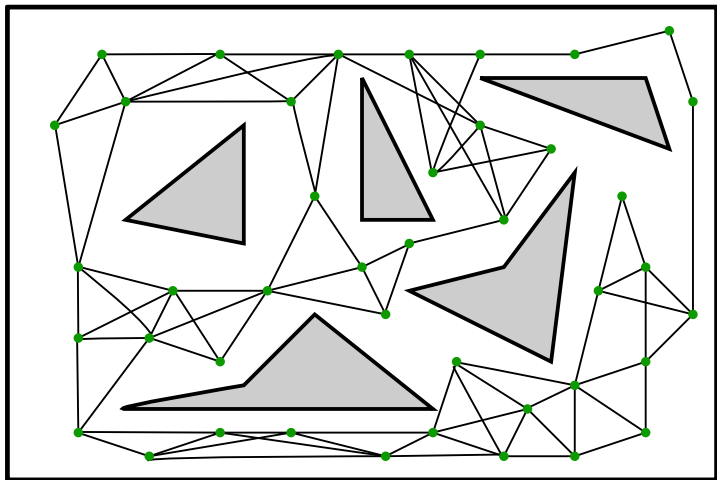
# Sample-based Path Finding

# Probabilistic Road Maps



[Kavraki, Svetska, Latombe, Overmars, 95]

# Probabilistic Road Maps

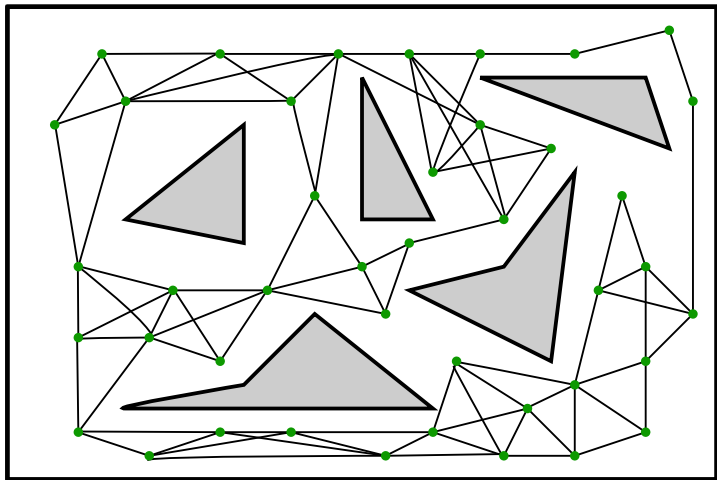


[Kavraki, Svetska, Latombe, Overmars, 95]

$q \in \mathbb{R}^n$  describes configuration

$Q_{\text{free}}$  is the set of configurations without collision

# Probabilistic Road Maps



[Kavraki, Svetska, Latombe, Overmars, 95]

Probabilistic Road Map

- generates a graph  $G = (V, E)$  of configurations
- such that configurations along each edges are  $\in Q_{\text{free}}$

Given the graph, use (e.g.) Dijkstra to find path from  $q_{\text{start}}$  to  $q_{\text{goal}}$ .



# Probabilistic Road Maps – generation

---

**Input:** number  $n$  of samples, number  $k$  number of nearest neighbors

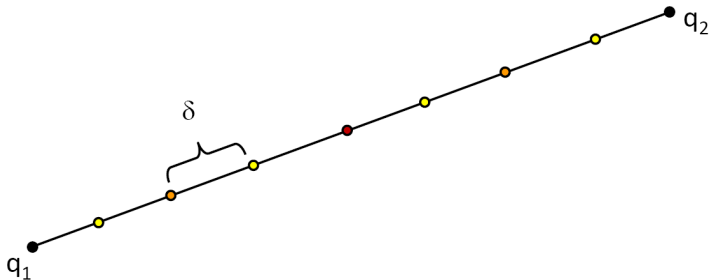
**Output:** PRM  $G = (V, E)$

```
1: initialize  $V = \emptyset, E = \emptyset$ 
2: while  $|V| < n$  do                                     // find  $n$  collision free points  $q_i$ 
3:    $q \leftarrow$  random sample from  $Q$ 
4:   if  $q \in Q_{\text{free}}$  then  $V \leftarrow V \cup \{q\}$ 
5: end while
6: for all  $q \in V$  do                                       // check if near points can be connected
7:    $N_q \leftarrow k$  nearest neighbors of  $q$  in  $V$ 
8:   for all  $q' \in N_q$  do
9:     if  $\text{path}(q, q') \in Q_{\text{free}}$  then  $E \leftarrow E \cup \{(q, q')\}$ 
10:  end for
11: end for
```

---

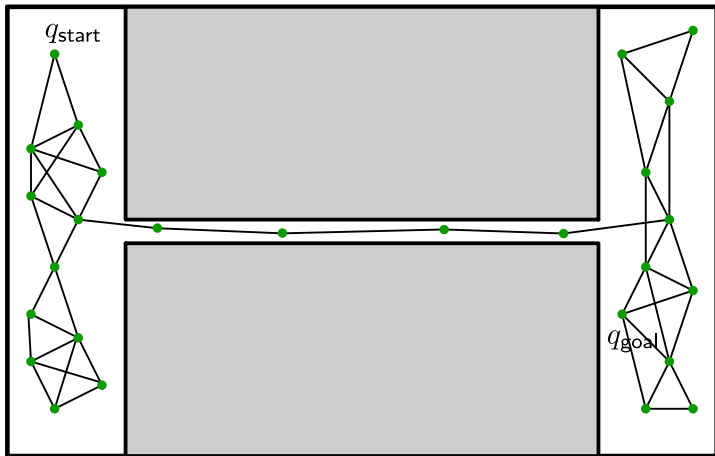
where  $\text{path}(q, q')$  is a local planner (easiest: straight line)

# Local Planner



tests collisions up to a specified resolution  $\delta$

# Problem: Narrow Passages



The smaller the gap (clearance  $\varrho$ ) the more unlikely to sample such points.

# PRM theory

(for uniform sampling in  $d$ -dim space)

- Let  $a, b \in Q_{\text{free}}$  and  $\gamma$  a path in  $Q_{\text{free}}$  connecting  $a$  and  $b$ .

Then the probability that *PRM* found the path after  $n$  samples is

$$P(\text{PRM-success} \mid n) \geq 1 - \frac{2|\gamma|}{\varrho} e^{-\sigma \varrho^d n}$$

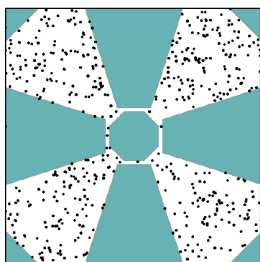
$$\sigma = \frac{|B_1|}{2^d |Q_{\text{free}}|}$$

$\varrho$  = clearance of  $\gamma$  (distance to obstacles)

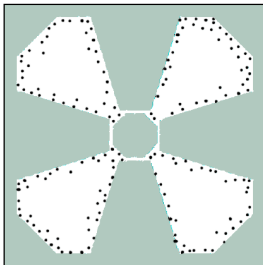
(roughly: the exponential term are “volume ratios”)

- This result is called *probabilistic complete* (one can achieve any probability with high enough  $n$ )
- For a given success probability,  $n$  needs to be exponential in  $d$

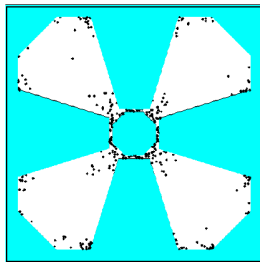
# Other PRM sampling strategies



uniform



Gaussian



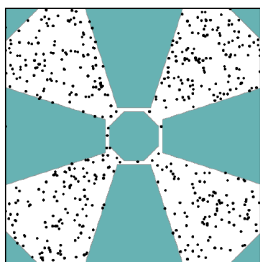
Bridge

illustration from O. Brock's lecture

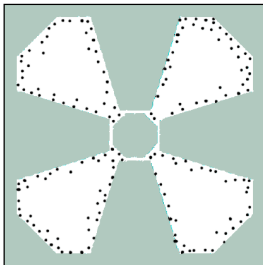
Gaussian:  $q_1 \sim \mathcal{U}$ ;  $q_2 \sim \mathcal{N}(q_1, \sigma)$ ; if  $q_1 \in Q_{\text{free}}$  and  $q_2 \notin Q_{\text{free}}$ , add  $q_1$  (or vice versa).

Bridge:  $q_1 \sim \mathcal{U}$ ;  $q_2 \sim \mathcal{N}(q_1, \sigma)$ ;  $q_3 = (q_1 + q_2)/2$ ; if  $q_1, q_2 \notin Q_{\text{free}}$  and  $q_3 \in Q_{\text{free}}$ , add  $q_3$ .

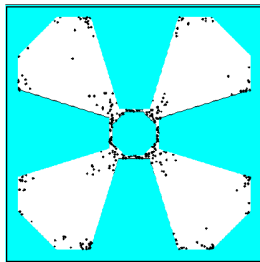
# Other PRM sampling strategies



uniform



Gaussian



Bridge

illustration from O. Brock's lecture

Gaussian:  $q_1 \sim \mathcal{U}$ ;  $q_2 \sim \mathcal{N}(q_1, \sigma)$ ; if  $q_1 \in Q_{\text{free}}$  and  $q_2 \notin Q_{\text{free}}$ , add  $q_1$  (or vice versa).

Bridge:  $q_1 \sim \mathcal{U}$ ;  $q_2 \sim \mathcal{N}(q_1, \sigma)$ ;  $q_3 = (q_1 + q_2)/2$ ; if  $q_1, q_2 \notin Q_{\text{free}}$  and  $q_3 \in Q_{\text{free}}$ , add  $q_3$ .

- Sampling strategy can be made more intelligence: “utility-based sampling”
- Connection sampling  
(once earlier sampling has produced connected components)

## Probabilistic Roadmaps – conclusions

- Pros:
  - Algorithmically very simple
  - Highly explorative
  - Allows probabilistic performance guarantees
  - Good to answer many queries in an *unchanged* environment
- Cons:
  - Precomputation of exhaustive roadmap takes a long time (but not necessary for “Lazy PRMs”)

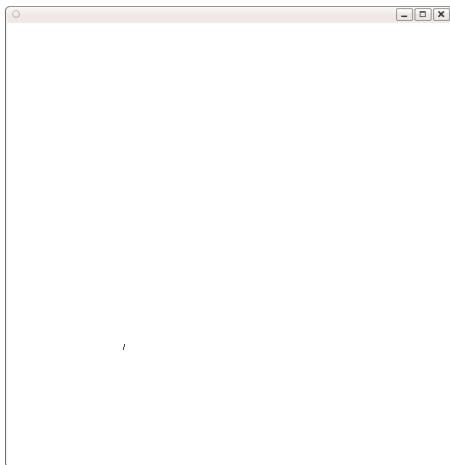
# Rapidly Exploring Random Trees

2 motivations:

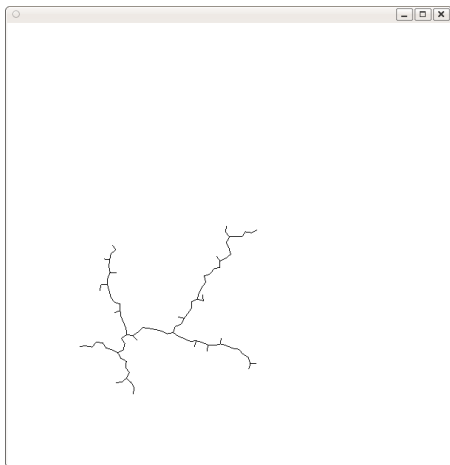
- Single Query path finding: Focus computational efforts on paths for specific ( $q_{\text{start}}$ ,  $q_{\text{goal}}$ )
- Use actually controllable DoFs to incrementally explore the search space: *control-based* path finding.

(Ensures that RRTs can be extended to handling differential constraints.)

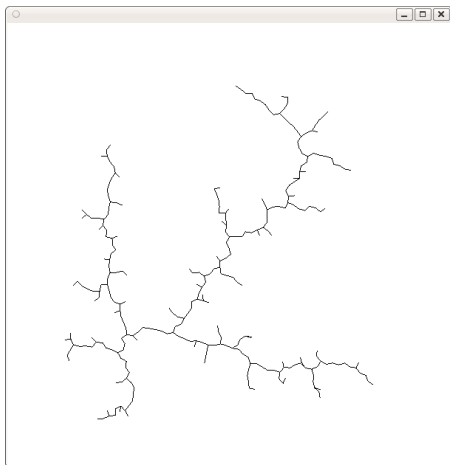




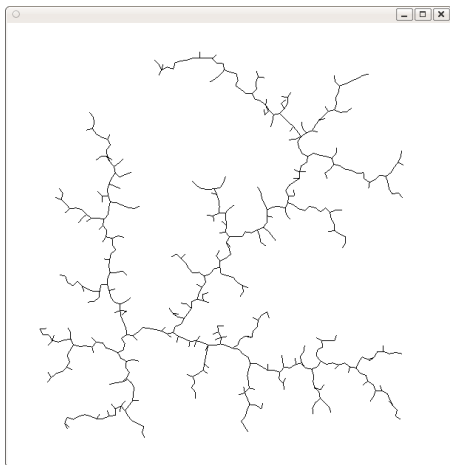
$$n = 1$$



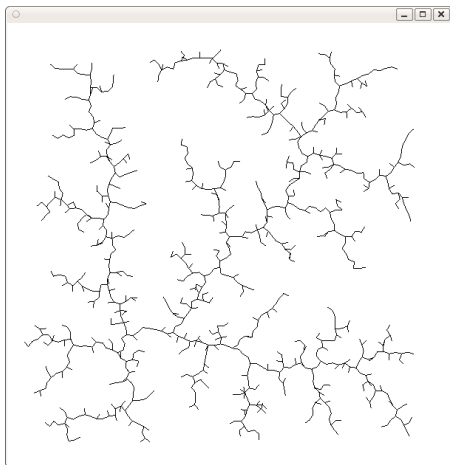
$n = 100$



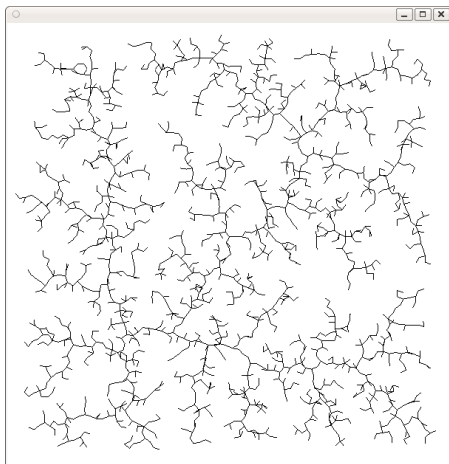
$n = 300$



$n = 600$



$n = 1000$



$n = 2000$

# Rapidly Exploring Random Trees

Simplest RRT with straight line local planner and step size  $\alpha$

---

**Input:**  $q_{\text{start}}$ , number  $n$  of nodes, stepsize  $\alpha$

**Output:** tree  $T = (V, E)$

1: initialize  $V = \{q_{\text{start}}\}$ ,  $E = \emptyset$

2: **for**  $i = 0 : n$  **do**

3:    $q_{\text{target}} \leftarrow$  random sample from  $Q$

4:    $q_{\text{near}} \leftarrow$  nearest neighbor of  $q_{\text{target}}$  in  $V$

5:    $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|} (q_{\text{target}} - q_{\text{near}})$

6:   **if**  $q_{\text{new}} \in Q_{\text{free}}$  **then**  $V \leftarrow V \cup \{q_{\text{new}}\}$ ,  $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$

7: **end for**

---

# Rapidly Exploring Random Trees

RRT growing directedly towards the goal

---

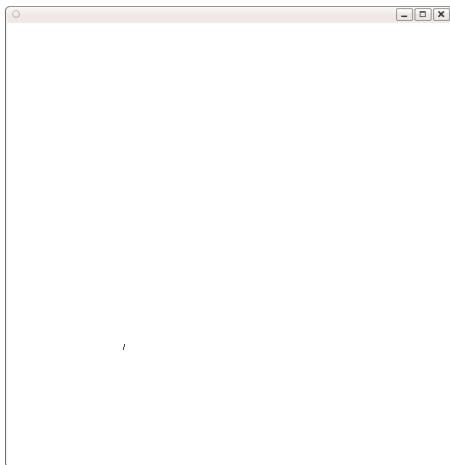
**Input:**  $q_{\text{start}}$ ,  $q_{\text{goal}}$ , number  $n$  of nodes, stepsize  $\alpha$ ,  $\beta$

**Output:** tree  $T = (V, E)$

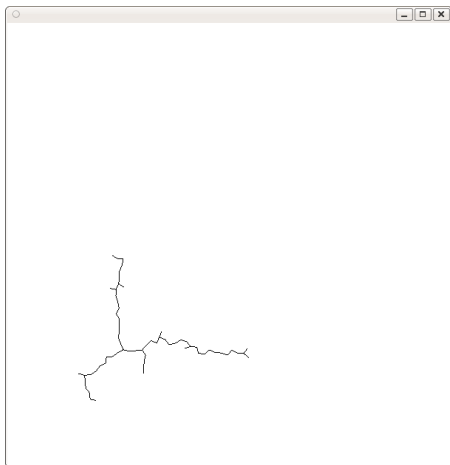
```
1: initialize  $V = \{q_{\text{start}}\}$ ,  $E = \emptyset$ 
2: for  $i = 0 : n$  do
3:   if  $\text{rand}(0, 1) < \beta$  then  $q_{\text{target}} \leftarrow q_{\text{goal}}$ 
4:   else  $q_{\text{target}} \leftarrow$  random sample from  $Q$ 
5:    $q_{\text{near}} \leftarrow$  nearest neighbor of  $q_{\text{target}}$  in  $V$ 
6:    $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|} (q_{\text{target}} - q_{\text{near}})$ 
7:   if  $q_{\text{new}} \in Q_{\text{free}}$  then  $V \leftarrow V \cup \{q_{\text{new}}\}$ ,  $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$ 
8: end for
```

---

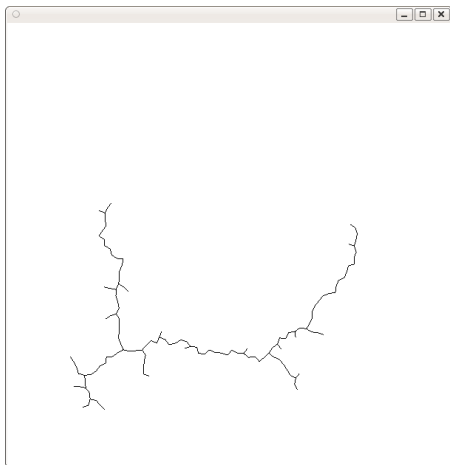




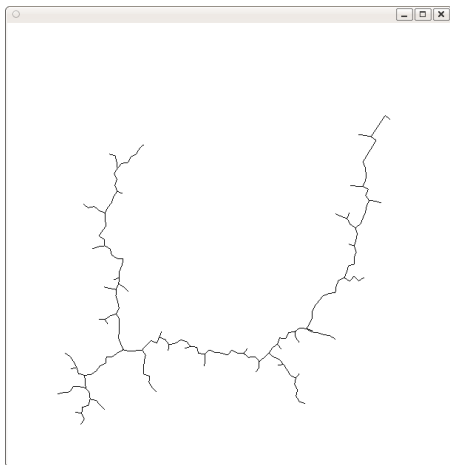
$$n = 1$$



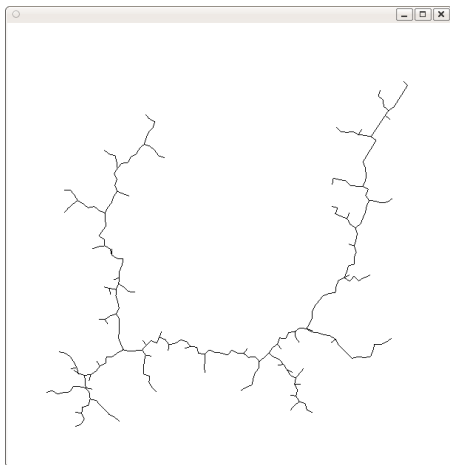
$$n = 100$$



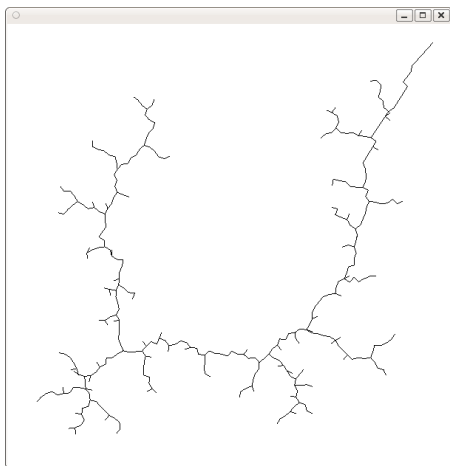
$$n = 200$$



$n = 300$



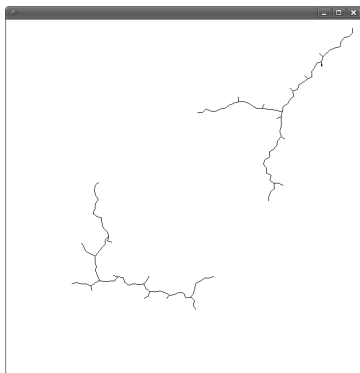
$n = 400$



$n = 500$

# Bi-directional search

- grow two trees starting from  $q_{\text{start}}$  and  $q_{\text{goal}}$



let one tree grow towards the other  
(e.g., “choose  $q_{\text{new}}$  of  $T_1$  as  $q_{\text{target}}$  of  $T_2$ ”)

# Summary: RRTs

- Pros (shared with PRMs):
  - Algorithmically very simple
  - Highly explorative
  - Allows probabilistic performance guarantees
- Pros (beyond PRMs):
  - Focus computation on single query ( $q_{\text{start}}, q_{\text{goal}}$ ) problem
  - Trees from multiple queries can be merged to a roadmap
  - Can be extended to differential constraints (nonholonomic systems)
- To keep in mind (shared with PRMs):
  - The metric (for nearest neighbor selection) is sometimes critical
  - The local planner may be non-trivial



## References

Steven M. LaValle: *Planning Algorithms*,  
<http://planning.cs.uiuc.edu/>.

Choset et. al.: *Principles of Motion Planning*, MIT Press.

Latombe's "motion planning" lecture, <http://robotics.stanford.edu/~latombe/cs326/2007/schedule.htm>

# Non-holonomic systems

# Non-holonomic systems

- We define a **nonholonomic system** as one with **differential constraints**:

$$\dim(u_t) < \dim(x_t)$$

$\Rightarrow$  *Not all degrees of freedom are directly controllable*

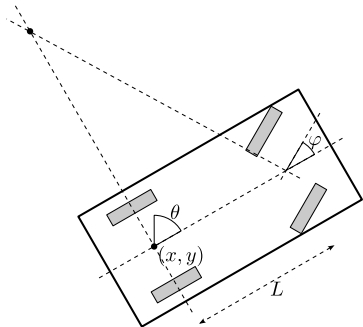
- Dynamic systems are an example!
- General definition of a differential constraint:  
For any given state  $x$ , let  $U_x$  be the tangent space that is generated by controls:

$$U_x = \{\dot{x} : \dot{x} = f(x, u), u \in U\}$$

$$(\text{non-holonomic} \iff \dim(U_x) < \dim(x))$$

The elements of  $U_x$  are elements of  $T_x$  subject to additional *differential constraints*.

# Car example



$$\dot{x} = v \cos \theta$$

$$\dot{y} = v \sin \theta$$

$$\dot{\theta} = (v/L) \tan \varphi$$

$$|\varphi| < \Phi$$

**State**  $q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$

**System equation**

**Controls**  $u = \begin{pmatrix} v \\ \varphi \end{pmatrix}$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \cos \theta \\ v \sin \theta \\ (v/L) \tan \varphi \end{pmatrix}$$

## Car example

- The car is a *non-holonomic* system: not all DoFs are controlled,  $\dim(u) < \dim(q)$   
We have the *differential constraint*  $\dot{q}$ :

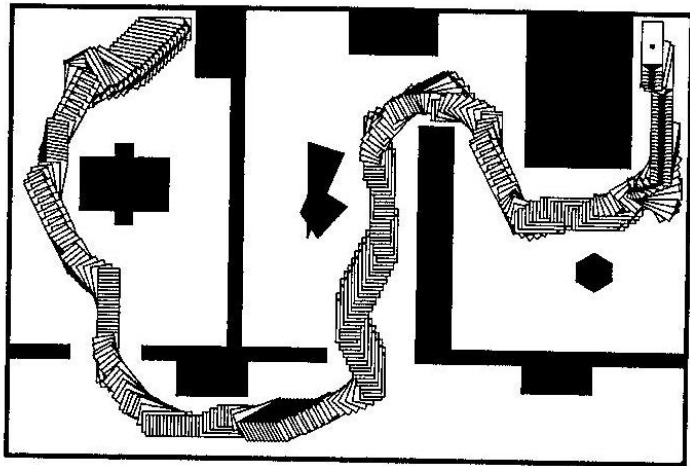
$$\dot{x} \sin \theta - \dot{y} \cos \theta = 0$$

“A car cannot move directly lateral.”

- Analogy to dynamic systems: Just like a car cannot instantly move sideways, a dynamic system cannot instantly change its position  $q$ : the current change in position is *constrained* by the current velocity  $\dot{q}$ .

# Path finding with a non-holonomic system

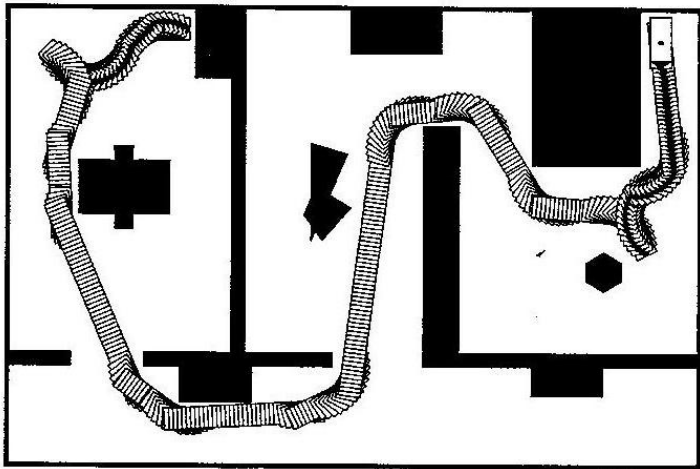
Could a car follow this trajectory?



This is generated with a PRM in the state space  $q = (x, y, \theta)$  *ignoring the differential constraint*.

## Path finding with a non-holonomic system

This is a solution we would like to have:



The path respects **differential constraints**.

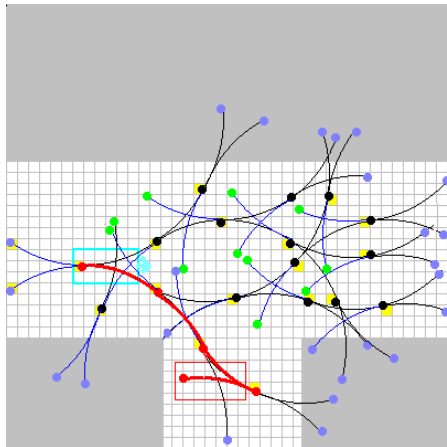
Each step in the path corresponds to setting certain controls.

# Control-based sampling to grow a tree

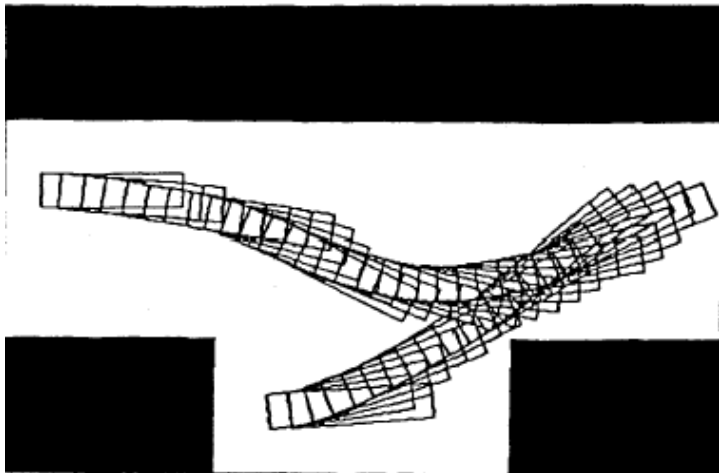
- Control-based sampling: fulfils none of the nice exploration properties of RRTs, but fulfils the differential constraints:
  - 1) Select a  $q \in T$  from tree of current configurations
  - 2) Pick control vector  $u$  at random
  - 3) Integrate equation of motion over short duration (picked at random or not)
  - 4) If the motion is collision-free, add the endpoint to the tree



## Control-based sampling for the car

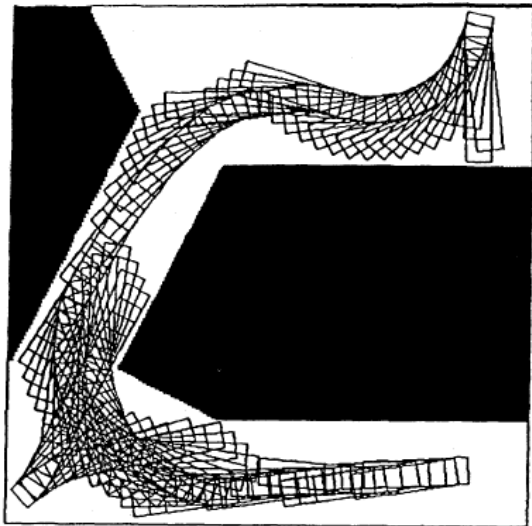


- 1) Select a  $q \in T$
- 2) Pick  $v$ ,  $\phi$ , and  $\tau$
- 3) Integrate motion from  $q$
- 4) Add result if collision-free

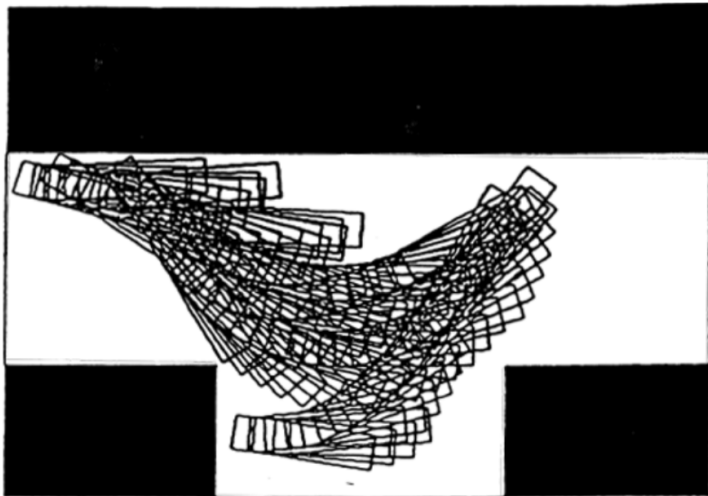


J. Barraquand and J.C. Latombe. Nonholonomic Multibody Robots:  
Controllability and Motion Planning in the Presence of Obstacles. *Algorithmica*,  
10:121-155, 1993.

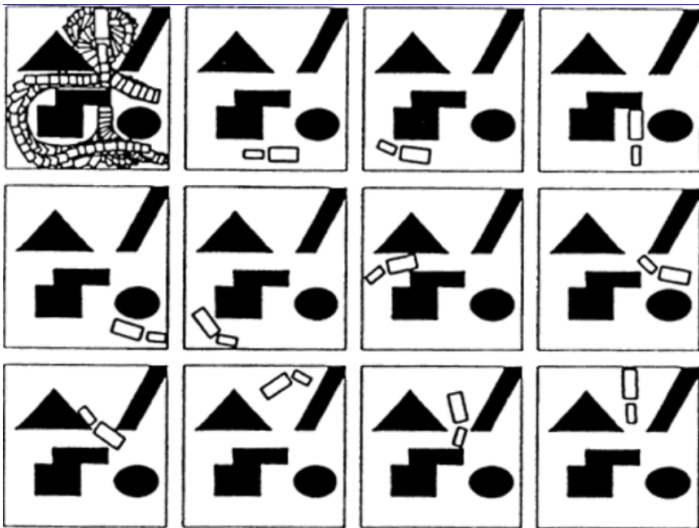
car parking



car parking



parking with only left-steering



with a trailer

# Better control-based exploration: RRTs revisited

- RRTs with differential constraints:

---

**Input:**  $q_{\text{start}}$ , number  $k$  of nodes, **time interval**  $\tau$

**Output:** tree  $T = (V, E)$

- 1: initialize  $V = \{q_{\text{start}}\}$ ,  $E = \emptyset$
  - 2: **for**  $i = 0 : k$  **do**
  - 3:    $q_{\text{target}} \leftarrow$  random sample from  $Q$
  - 4:    $q_{\text{near}} \leftarrow$  **nearest** neighbor of  $q_{\text{target}}$  in  $V$
  - 5:   **use local planner to compute controls**  $u$  **that steer**  $q_{\text{near}}$  **towards**  $q_{\text{target}}$
  - 6:    $q_{\text{new}} \leftarrow q_{\text{near}} + \int_{t=0}^{\tau} \dot{q}(q, u) dt$
  - 7:   **if**  $q_{\text{new}} \in Q_{\text{free}}$  **then**  $V \leftarrow V \cup \{q_{\text{new}}\}$ ,  $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
  - 8: **end for**
- 

- Crucial questions:
  - How measure *near* in nonholonomic systems?
  - How find controls  $u$  to steer towards target?

# Metrics

Standard/Naive metrics:

- Comparing two 2D rotations/orientations  $\theta_1, \theta_2 \in SO(2)$ :
  - a) Euclidean metric between  $e^{i\theta_1}$  and  $e^{i\theta_2}$
  - b)  $d(\theta_1, \theta_2) = \min\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\}$
- Comparing two configurations  $(x, y, \theta)_{1,2}$  in  $\mathbb{R}^2$ :  
Euclidean metric on  $(x, y, e^{i\theta})$
- Comparing two 3D rotations/orientations  $r_1, r_2 \in SO(3)$ :  
Represent both orientations as unit-length quaternions  $r_1, r_2 \in \mathbb{R}^4$ :
$$d(r_1, r_2) = \min\{|r_1 - r_2|, |r_1 + r_2|\}$$
where  $|\cdot|$  is the Euclidean metric.  
(Recall that  $r_1$  and  $-r_1$  represent exactly the same rotation.)

# Metrics

Standard/Naive metrics:

- Comparing two 2D rotations/orientations  $\theta_1, \theta_2 \in SO(2)$ :
  - a) Euclidean metric between  $e^{i\theta_1}$  and  $e^{i\theta_2}$
  - b)  $d(\theta_1, \theta_2) = \min\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\}$
- Comparing two configurations  $(x, y, \theta)_{1,2}$  in  $\mathbb{R}^2$ :  
Euclidean metric on  $(x, y, e^{i\theta})$
- Comparing two 3D rotations/orientations  $r_1, r_2 \in SO(3)$ :  
Represent both orientations as unit-length quaternions  $r_1, r_2 \in \mathbb{R}^4$ :
$$d(r_1, r_2) = \min\{|r_1 - r_2|, |r_1 + r_2|\}$$
where  $|\cdot|$  is the Euclidean metric.  
(Recall that  $r_1$  and  $-r_1$  represent exactly the same rotation.)

- **Ideal metric:**

Optimal cost-to-go between two states  $x_1$  and  $x_2$ :

- Use optimal trajectory cost as metric
- This is as hard to compute as the original problem, of course!!
- Approximate, e.g., by neglecting obstacles.



# Dubins curves

- Dubins car: constant velocity, and steer  $\varphi \in [-\Phi, \Phi]$
- Neglecting obstacles, there are only **six** types of trajectories that connect any configuration  $\in \mathbb{R}^2 \times \mathbb{S}^1$ :

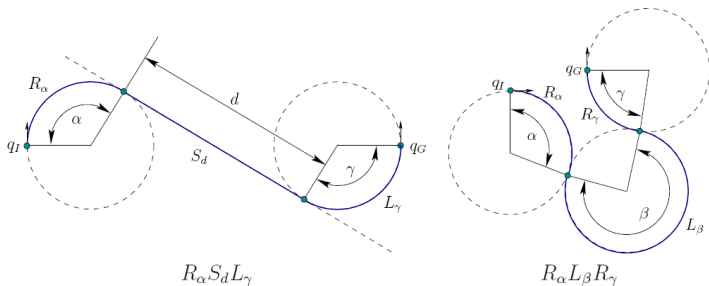
$$\{LRL, RLR, LSL, LSR, RSL, RSR\}$$

- annotating durations of each phase:

$$\{L_\alpha R_\beta L_\gamma, R_\alpha L_\beta R_\gamma, L_\alpha S_d L_\gamma, L_\alpha S_d R_\gamma, R_\alpha S_d L_\gamma, R_\alpha S_d R_\gamma\}$$

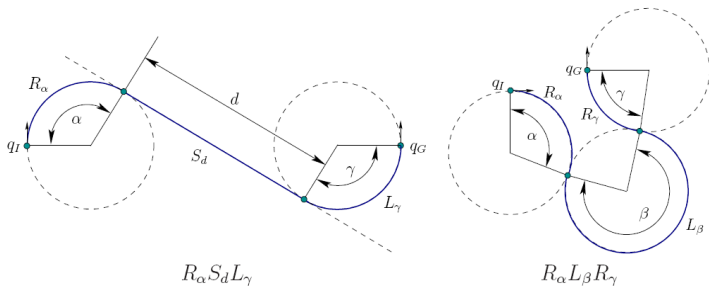
with  $\alpha \in [0, 2\pi), \beta \in (\pi, 2\pi), d \geq 0$

# Dubins curves



→ By testing all six types of trajectories for  $(q_1, q_2)$  we can define a Dubins metric for the RRT – and use the Dubins curves as controls!

# Dubins curves



→ By testing all six types of trajectories for  $(q_1, q_2)$  we can define a Dubins metric for the RRT – and use the Dubins curves as controls!

- **Reeds-Shepp curves** are an extension for cars which can drive back. (includes 46 types of trajectories, good metric for use in RRTs for cars)