Probabilistic Robotics: Planning & Control

(Sample-Based Planning/State-Space/LQR)

METR 4202: **Robotics** & Automation

Dr Surya Singh -- Lecture # 11

October 11, 2017

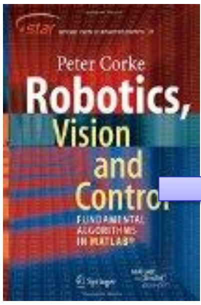
metr4202-staff@itee.uq.edu.au

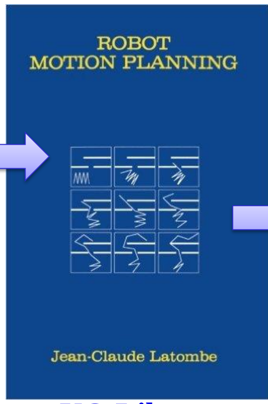
http://robotics.itee.uq.edu.au/~metr4202/ [http://**metr4202.com**]

Lecture Schedule

Week	Date	Lecture (W: 3:05p-4:50, 7-222)
1	26-Jul	Introduction + Representing Position & Orientation & State
2	2-Aug	Robot Forward Kinematics (Frames, Transformation Matrices & Affine Transformations)
3	9-Aug	Robot Inverse Kinematics & Dynamics (Jacobians)
4	16-Aug	*Ekka Day* (Robot Kinematics & Kinetics Review)
5	23-Aug	Jacobians & Robot Sensing Overview
6	30-Aug	Robot Sensing: Single View Geometry & Lines
7	6-Sep	Robot Sensing: Basic Feature Detection
8	13-Sep	Robot Sensing: Scalable Feature Detection
9	20-Sep	Mid-Semester Exam & Multiple View Geometry
	27-Sep	*Study break*
10	4-Oct	Motion Planning
11	11-Oct	**Probabilistic Robotics: Planning & Control** **(Sample-Based Planning/State-Space/LQR)**
12	18-Oct	Probabilistic Robotics: Localization & SLAM
13	25-Oct	The Future of Robotics/Automation + Challenges + Course Review

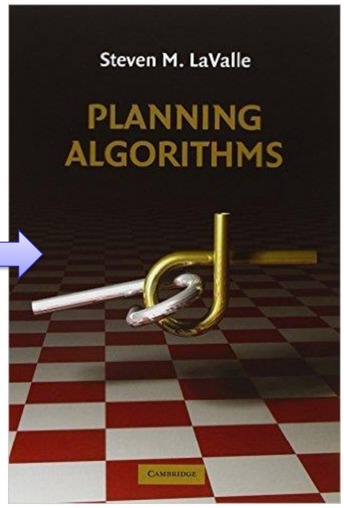
Reference Material





UQ Library
(TJ211.4 .L38 1991)

Chapter 5

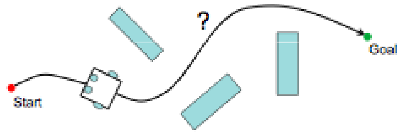


UQ Library / Online (PDF)

Recap: Simple Planners!

Motion Planning Problem in General

- Motion planning problem:



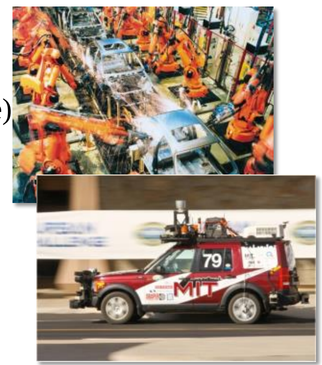
- Last Week:
  - Configuration space
  - Cell decomposition
  - Guided search using A*
  - Potential fields

Artwork based on Teller, RRTs

Motion Planning Problem in General

- Given:
  - Robot's dynamics
  - A map of the environment
  - (perfect information, but discovered online)
  - Robot's pose in the map
  - A goal pose in the map
- Find a sequence of
  - Actuation commands
  - (such as steer, gas/brake, transmission)
  - In real time (requires efficient algorithms)

  … that drive system to the goal pose



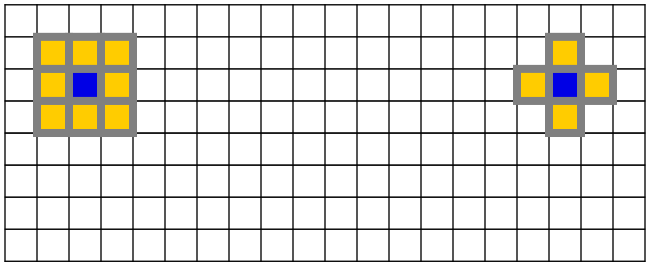
- Problem is essential in almost all robotics applications irrespective of size, type of actuation, sensor suite, task domain, etc.

Artwork based on Teller, RRTs

Connectivity

- 8-Point Connectivity

  - 4-Point Connectivity



Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

The Wavefront Planner: Setup

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

Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

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	0
6	0	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	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	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	0	3	3
0	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	15

Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

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 >= 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	0
6	0	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	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	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	0	4	3	3
0	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	15

Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

The Wavefront in Action (Part 3)

- Repeat again...

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

Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

The Wavefront in Action (Part 4)

- And again...

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

Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

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

Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

The Wavefront in Action (Done)

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

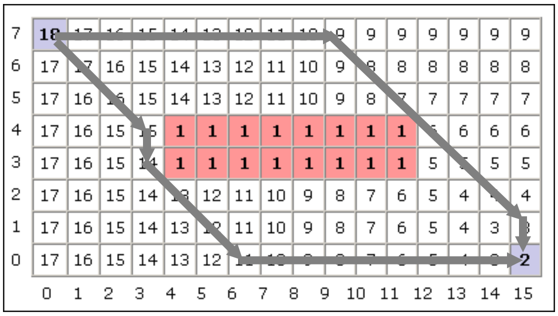
7	**18**	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9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

Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning

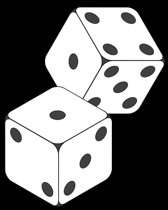
The Wavefront, Now What?

- 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



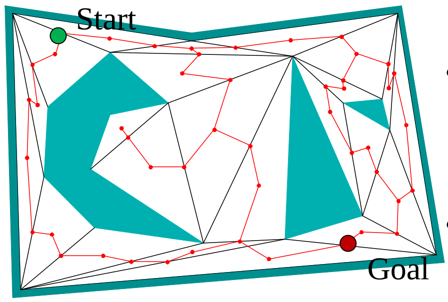
Artwork based on S. Shiry, Fazli, Introduction to Mobile Robots Motion Planning



Sample-Based Motion Planning!

Dice artwork from https://commons.wikimedia.org/wiki/File:2-Dice-Icon.svg

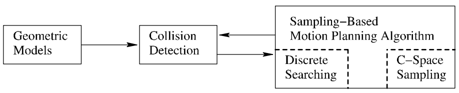
Limits of Geometric Planning Methods



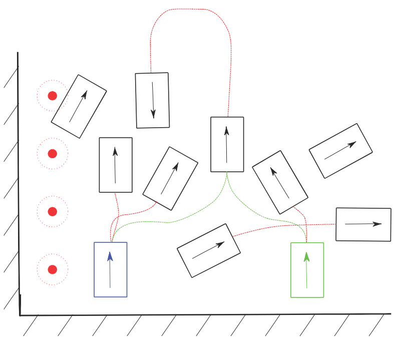
- How does this scale to high degrees of freedom?
- What about “dynamic constraints”?
- What about optimality?
- How to tie this to learning and optimization

Artwork from LaValle, Ch. 6

Sample-Based Motion Planning



- PRMs

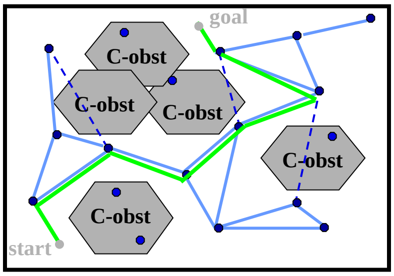


- RRTs

Artwork based on LaValle, Ch. 5

Random Sampling in **C-Space**

C-space



Roadmap Construction (Pre-processing)

1. Randomly generate robot configurations (nodes)
   - Discard invalid nodes (how?)
2. Connect pairs of nodes to form **roadmap edges**
   - Use simple, deterministic *local planner*
   - Discard invalid edges (how?)

Plan Generation (Query processing)

1. Link *start* and *goal* poses into roadmap
2. Find path from *start* to *goal* within roadmap
3. Generate motion plan for each edge used

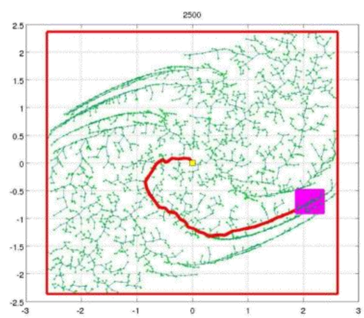
Primitives Required:

1. Method for sampling C-Space points
2. Method for "validating" C-space points and edges

Artwork based on Teller, RRTs

Sampling-Based Methods

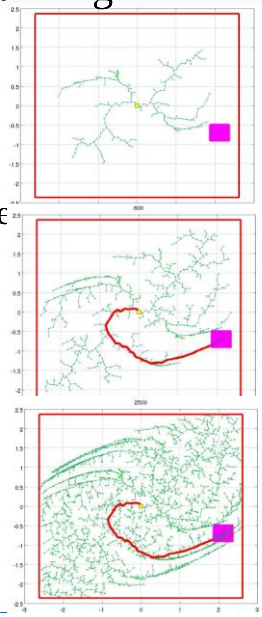
- (Randomly) construct a set of feasible (that is, collision-free) trajectories
- Quite efficient; methods scale well with increasing dimension, # of obstacles
- "Probabilistically complete" (if run long enough, very likely to find a solution)



Artwork based on Teller, RRTs

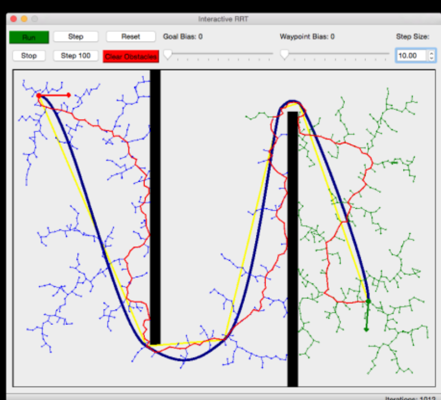
Incremental Sampling-based Motion Planning

- Sometimes building a roadmap a priori might be inefficient (or even impractical)
  - Assumes that all regions of c-space will be utilized during actual motions
- Building a roadmap requires global knowledge
  - But in real settings, obstacles are not known a priori; rather, they are discovered online
- We desire an incremental method:
  - Generate motion plans for a single start, goal pose
  - Expending more CPU yields better motion plans
- The Rapidly-exploring Random Tree (RRT) algorithm meets these requirements



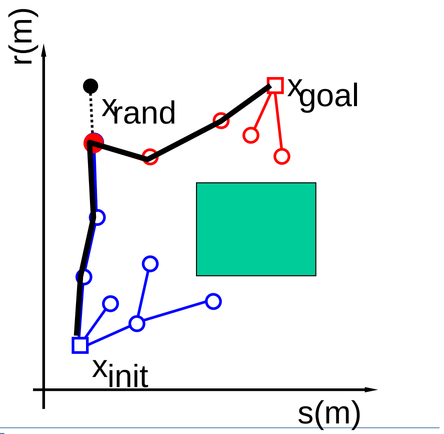
Artwork based on Teller, RRTs

RRT

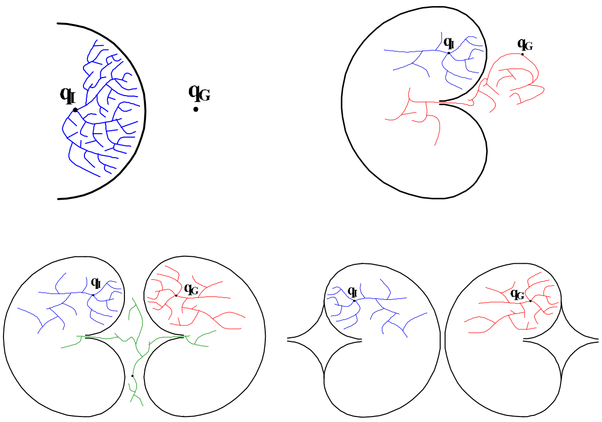


Based on notes from David Johnson (Utah) and Steve Lavalle (UIUC)
Screenshot from: https://github.com/RoboJackets/rrt

Rapidly Exploring Random Trees (RRT)



Sampling and the "Bug Trap" Problem



Artwork based on LaValle, Ch. 5

RRT Components

1. **State Space:** A topological space, X
2. **Boundary Values:** $x_{init} \in X$ and $X_{goal} \subset X$
3. **Collision Detector:** A function, $D : X \to \{true, false\}$, that determines whether global constraints are satisfied from state x. This could be a binary-valued or real-valued function.

- X = Configuration Space or more general
- X = C-space + Velocity + Acceleration + ….

Based on Johnson (Utah)

RRT Components

4. **Inputs:** A set, U, which specifies the complete set of controls or actions that can affect the state.
5. **Incremental Simulator:** Given the current state, $x(t)$, and inputs applied over a time interval, $\{u(t')|t \leq t' \leq t + \Delta t\}$, compute $x(t + \Delta t)$.
6. **Metric:** A real-valued function, $\rho : X \times X \to [0, \infty)$, which specifies the distance between pairs of points in X.

Based on Johnson (Utah)

Basic RRT algorithm

BUILD_RRT(x_{init})

$\mathcal{T}$.init(x_{init});

for $k = 1$ **to** K **do**

  $x_{rand} \leftarrow$ RANDOM_STATE();

  EXTEND($\mathcal{T}, x_{rand}$);

Return $\mathcal{T}$

EXTEND($\mathcal{T}, x$)

$x_{near} \leftarrow$ NEAREST_NEIGHBOR($x, \mathcal{T}$);

if NEW_STATE($x, x_{near}, x_{new}, u_{new}$) **then**

  $\mathcal{T}$.add_vertex(x_{new});

  $\mathcal{T}$.add_edge($x_{near}, x_{new}, u_{new}$);

  if $x_{new} = x$ **then**

    Return *Reached*;

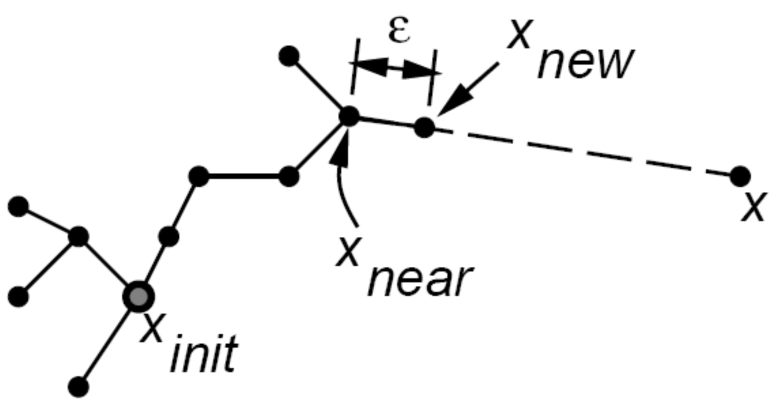
  else

    Return *Advanced*;

Return *Trapped*;

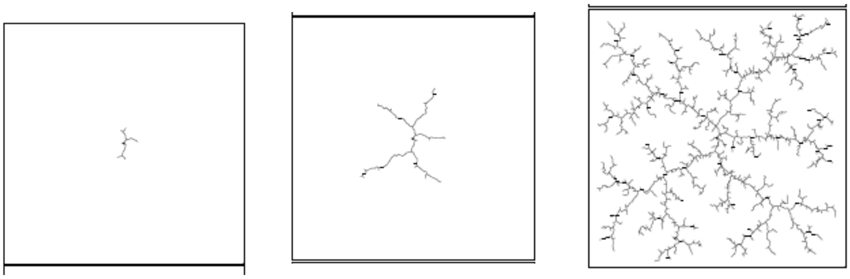
Based on Johnson (Utah)

Basic Extend

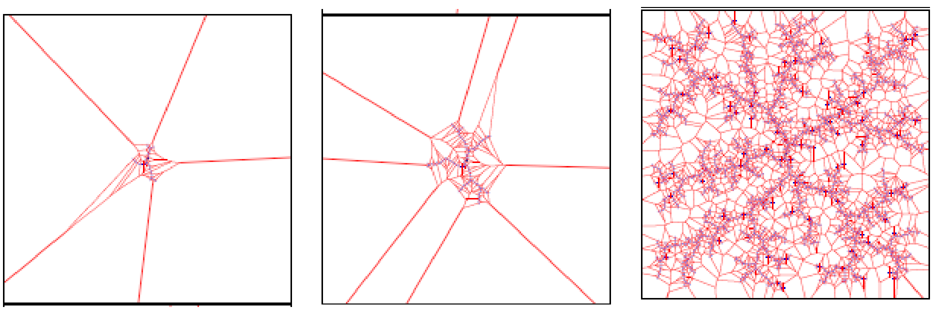


Based on Johnson (Utah)

Example in holonomic empty space

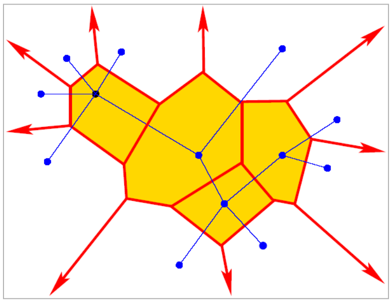


Why “Rapidly Exploring”?

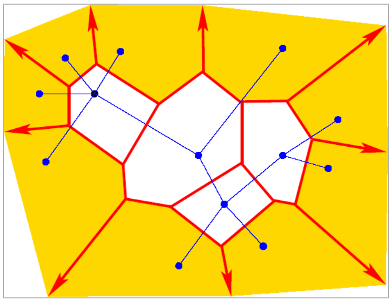


- What is the probability that a vertex will be extended?
  - Proportional to the area of its Voronoi region
- If just choose a vertex at random and extend, then it would act like random walk instead
  - Biased towards start vertex

Refinement vs. Expansion



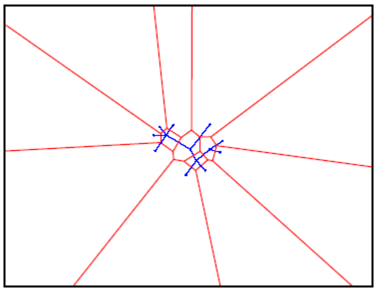
refinement



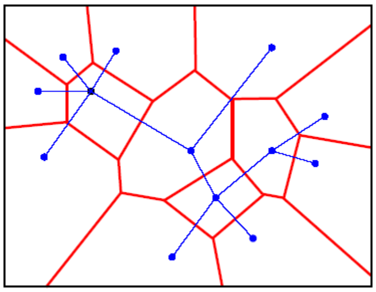
expansion

- Where will the random sample fall?
- How to control the behaviour of RRT?

Determining the Boundary



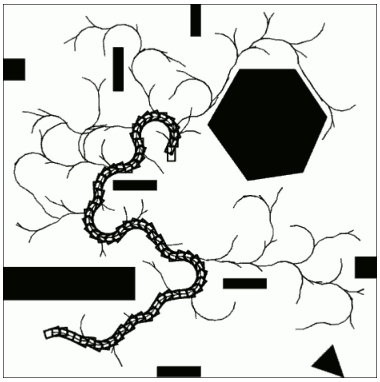
Expansion dominates



Balanced refinement and expansion

- The tradeoff depends on the size of the bounding box

Extension to Non-Holonomic

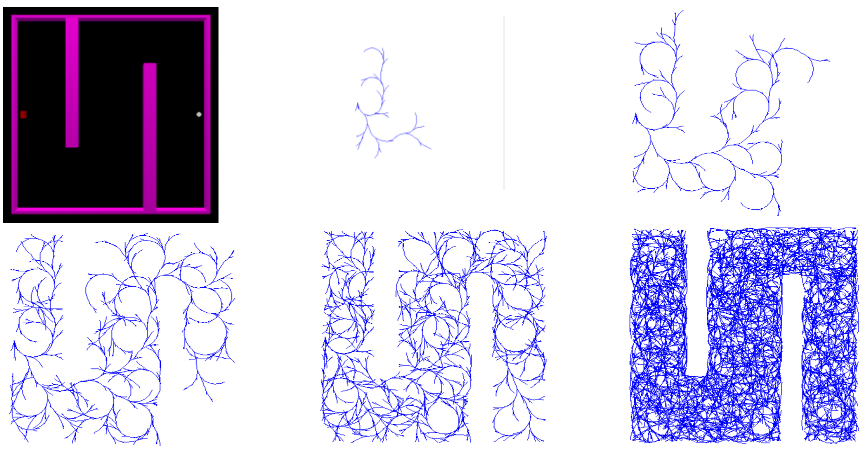


- The `new_state` computation handles all the complicated parts
- Given a state $\boldsymbol{x}$ and inputs $\boldsymbol{u}$

$$\dot{x} = f(x, u)$$

- Integrate numerically to get new position
- What input $\vec{\boldsymbol{u}}$ do we use?
  All | One | Heuristic | ?
- Problems:
  - Non-biased RRT explores in all directions Not aimed at the goal
  - Use a small percentage of targets to be the goal or its neighborhood

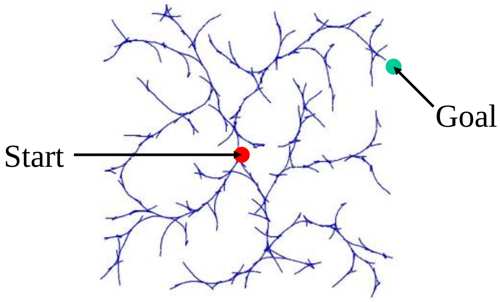
Example: Simple RRT Planner



- Problem: ordinary RRT explores X uniformly
- Solution: bias distribution towards the goal

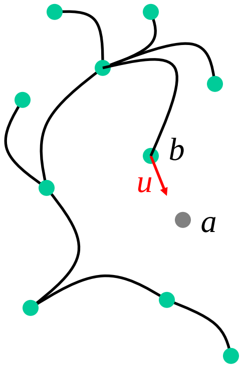
How it Works

- Build a rapidly-exploring random tree in state space (X), starting at sstart
- Stop when tree gets sufficiently close to sgoal



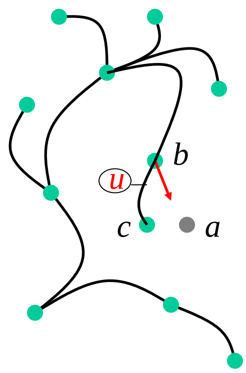
Building an RRT

- To extend an RRT:
  - Pick a random point a in X
  - Find b, the node of the tree closest to a
  - Find control inputs u to steer the robot from b to a



Building an RRT

- To extend an RRT (cont.)
  - Apply control inputs u for time d, so robot reaches c
  - If no collisions occur in getting from a to c, add c to RRT and record u with new edge



Bidirectional Planner Algorithm

RRT_BIDIRECTIONAL(x_{init}, x_{goal})

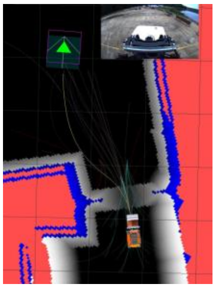
$\mathcal{T}_a$.init(x_{init}); $\mathcal{T}_b$.init(x_{goal});
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    if not (EXTEND($\mathcal{T}_a, x_{rand}$) $=$ *Trapped*) **then**
        if (EXTEND($\mathcal{T}_b, x_{new}$) $=$ *Reached*) **then**
            Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
    SWAP($\mathcal{T}_a, \mathcal{T}_b$);
Return *Failure*

RRT* & Extensions

Based on notes from Teller (MIT)

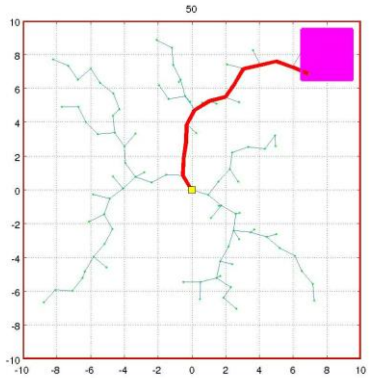
Limitations of RRT

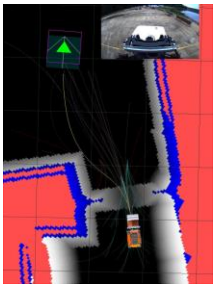
- RRT algorithm is tailored to find **a feasible solution very quickly**.



Limitations of RRT

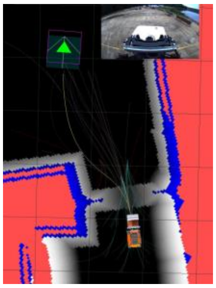
- RRT algorithm is tailored to find **a feasible solution very quickly**.
- However, our simulation results suggest that, **if kept running, the solution is *not* improved in terms of quality.**





Limitations of RRT

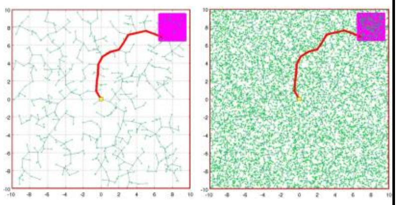
- RRT algorithm is tailored to find **a feasible solution very quickly**.
- However, our simulation results suggest that, **if kept running, the solution is *not* improved in terms of quality.**



- Can we formalize this claim?
- Let s^* be an optimal path and c^* denote its cost.
- Let X_n be a random variable that denotes the cost of the best path in the RRT at the end of iteration n.

$$\mathbb{P}\left(\{\lim_{n\to\infty} X_n = c^*\}\right) = 0$$

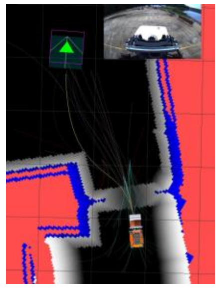
- That is, the probability that RRT will get closer to an optimal solution is zero.
- In other words, the RRT will **get stuck**.



Limitations of RRT

$$\mathbb{P}\left(\{\lim_{n\to\infty} X_n = c^*\}\right) = 0$$

- That is, the probability that RRT will get closer to an optimal solution is zero.
- In other words, the RRT will **get stuck**.



Why?

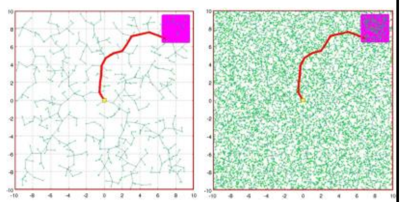
•The RRT **greedily explores** the c-space **exponentially fast**.

•Once *a path* is found, the RRT **traps itself** because of fast greedy exploration.

•*How about:*
- *running the RRT multiple times?*
- *deleting parts of the tree and rebuilding?*
- *new sampling strategies, better nearest neighbor?*

• All these may improve the performance, **but they will not make the RRT optimal.**

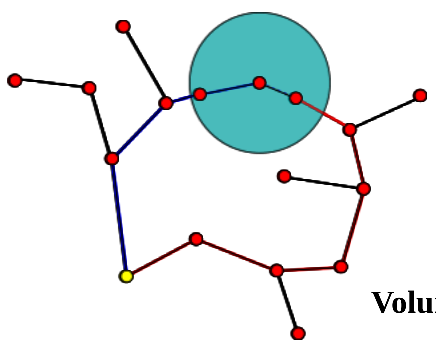
• **For optimality, we need to rethink the RRT algorithm…**



Extension to RRT*

(Karaman, Frazzoli, RSS'10)

- **Intuitively,**
  - **Run an RRT**
  - **Consider rewiring for all the nodes that are inside a ball of radius r_m centered at the new sample**



Volume of the ball:

$$O\left(\frac{\log n}{n}\right)$$

Extension to RRT*

```
V ← {x_init}; E ← ∅; i ← 0;
while i < N do
    G ← (V, E);
    x_rand ← Sample(i); i ← i + 1;
    (V, E) ← Extend(G, x_rand);
```

The body of the algorithm is very similar to the RRT.

```
V' ← V; E' ← E;
x_near ← Nearest(G, x);
x_new ← Steer(G, x);
if ObstacleFree(x_near, x_new) then
    V' := V' ∪ {x_new};
    x_min ← x_near;
    X_close ← CloseNodes(G, x);
    for all x_close ∈ X_close do
        if ObstacleFree(x_close, x_new) then
            c' ← Cost(x_close) + c(Line(x_close, x_new));
            if c' < Cost(x_new) then
                x_min ← x_close;
    E' ← E' ∪ {(x_close, x_min)};
    for all x_close ∈ X_close \ {x_min} do
        if ObstacleFree(x_new, x_close) and
        Cost(x_close) >
        Cost(x_new) + c(Line(x_new, x_close)) then
            x_parent ← Parent(x_close);
            E' ← E' \ {(x_parent, x_close)};
            E' ← E' ∪ {(x_new, x_close)};
return G' = (V', E')
```

Extend the nearest

If obstacle-free segment,
Then add the node to the tree

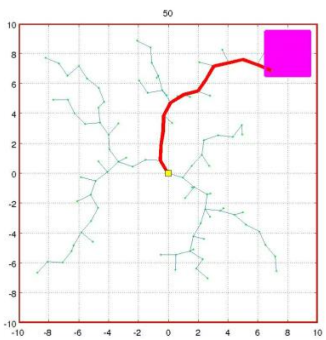
Compute the nodes inside the ball

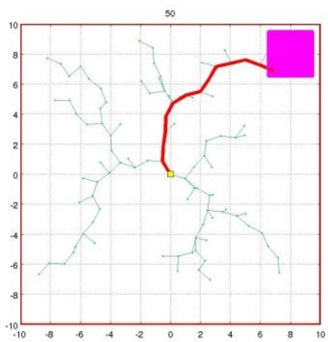
Determine the closest node to the sample in terms of the cost

Extend back to the tree – extend to all the nodes inside the balls

RRT* on simulation examples

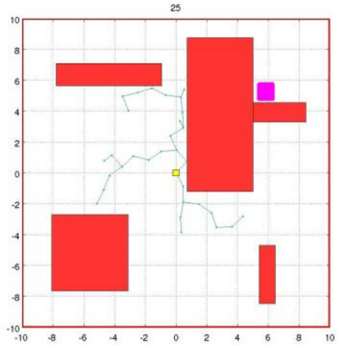
- The obstacle-free case:

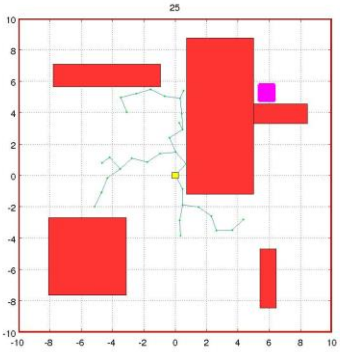




RRT* on simulation examples

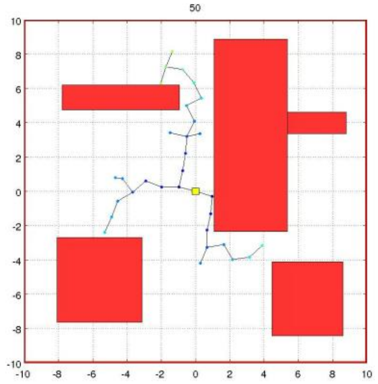
- With some obstacles





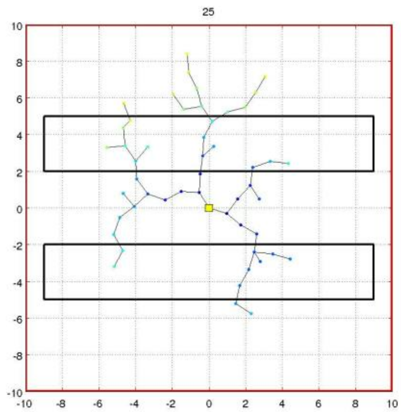
RRT* on simulation examples

- With some obstacles
- Cost color coded:



RRT* on simulation examples

- Upper rectangle is high cost
- Lower rectangle is low cost



Summary

- Studied the Rapidly-exploring Random Tree (RRT) algorithm
- Discussed challenges for motion planning methods in real-world applications
- Discussed magic behind sampling-based methods
- Looked at two applications:
  - Urban Challenge vehicle, Agile Robotics forklift
- Studied the optimality properties of the RRT and the RRT* algorithm.

RRT*: why it works? And how much time does it take to run?

- It can be shown that the RRT* converges to the optimum cost with probability one, that is

$$\mathbb{P}\left(\{\lim_{n\to\infty} X_n = c^*\}\right) = 1$$

- **Then, how about computational complexity?**
  - The number of nodes in the ball is around log(m), where m is the number of nodes.
  - Hence, we need to do log(m) more extensions at each iteration.
  - However, finding the nearest neighbor in a tree with m nodes takes around log(m) time anyway

	RRT	RRT*
Nearest neighbor	O(log m)	O(log m)
Close Nodes	-	O(log m)
Number of extensions	1	O(log m)
Total:	O(log m)	O(log m)

RRT*: why does it work? And how much time does it take to run?

- RRT* converges to the optimum cost with probability one
- RRT* has the same asymptotic computational complexity with the RRT.
  - Some experimental evidence for the theoretical claims:

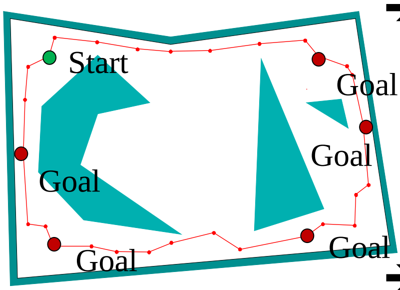
Running time of the RRT* / running time of the RRT vs. number of iterations

Multiple Points & Sequencing

- Sequencing
  - Determining the "best" order to go in

➔ Travelling Salesman Problem

A salesman has to visit each city on a given list exactly once. In doing this, he **starts** from his home city and in the **end he has to return to his home** city. It is plausible for him to select the order in which he visits the cities so that the **total of the distances travelled** in his tour is as small as possible.



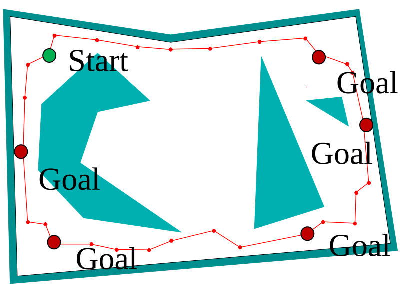
➔ Multi-Goal Problem

A salesman has to visit each city on a given list exactly once. In doing this, he **starts** from his home city and in the **end he has to return to his home** city. It is plausible for him to select the order in which he visits the cities so that the **total of the distances travelled** in his tour is as small as possible.

Artwork based on LaValle, Ch. 6

Travelling Salesman Problem

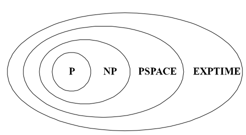
- Given a $n \times n$ distance matrix **C**=(c_{ij})



- Minimize:

$$c(\pi) = \sum_{i=1}^{n} c_{i\pi(i)}$$

- Note that this problem is NP-Hard

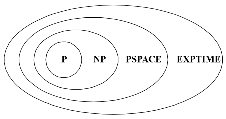


➔ BUT, Special Cases are Well-Solvable!

Artwork based on LaValle, Ch. 6

Travelling Salesman Problem [2]

- This problem is NP-Hard



➔ BUT,
Special Cases are Well-Solvable!

For the Euclidean case
(where the points are on the 2D Euclidean plane) :

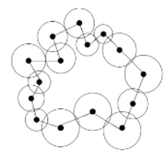
- The shortest TSP tour does not intersect itself, and thus geometry makes the problem somewhat easier.
- If all cities lie on the boundary of a convex polygon, the optimal tour is a cyclic walk along the boundary of the polygon (in clockwise or counterclockwise direction).

***The k*-line TSP**

- The a special case where the cities lie on *k* parallel (or almost parallel) lines in the Euclidean plane.
- EG: Fabrication of printed circuit boards
- Solvable in **O**(n^3) time by Dynamic Programming (Rote's algorithm)

The necklace TSP

- The special Euclidean TSP case where there exist *n* circles around the *n* cities such that every cycle intersects exactly two adjacent circles

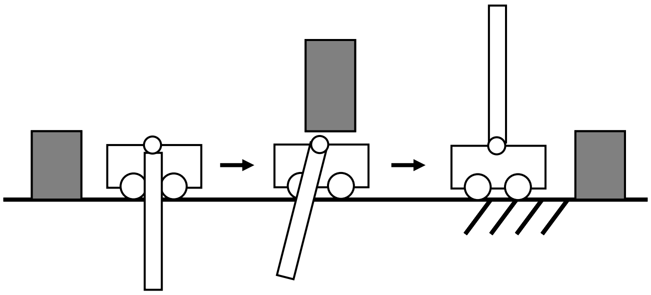


Integrated Planning & Control

(Hooray!!!)

Integrated Planning and Control Methods…

- A motivating problem (for agility)
  - Cart and pole in a cluttered workspace …

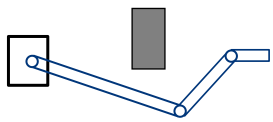


$$\dot{x} = F\left(x, u\right)$$

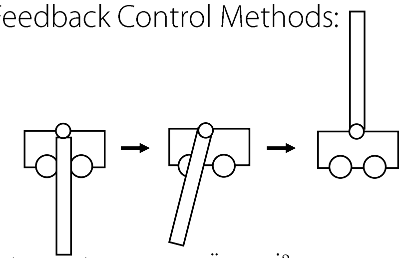
Trajectory Generation with Constraints: Solutions from the Robotics Domain

$$\dot{x} = F(x, u)$$

Motion Planning Methods:



Feedback Control Methods:



$$(M+m)\,\ddot{y} + ml\cos\theta\ddot{\theta} - ml\dot{\theta}^2\sin\theta = f$$

$$m\left(l\cos\theta\ddot{y} + l^2\ddot{\theta} - gl\sin\theta\right) = 0$$

Trajectory Optimization: → Integrated Planning & Feedback:

$$\dot{x} = F(x, u)$$

Planning Methods

Trajectory Optimization

- Direct Collocation (x: pchip poly, u: lin poly)
- Lyapunov *f* (POTools)
- SNOPT
- exploits: LQR can be solved efficiently

LQG Control Functions

Few Questions Before Starting…

- *Can it possibly be this hard?*

  Yes!

  (1) Dynamic systems are nonlinear $m\left(l^2\ddot{\theta} + gl\sin\theta\right) = 0$

  (2) Decision-theoretic planning problems are combinatorial

- ***Underactuated system**?*

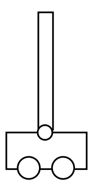
  [≜ control input cannot drive the state to any arbitrary direction]

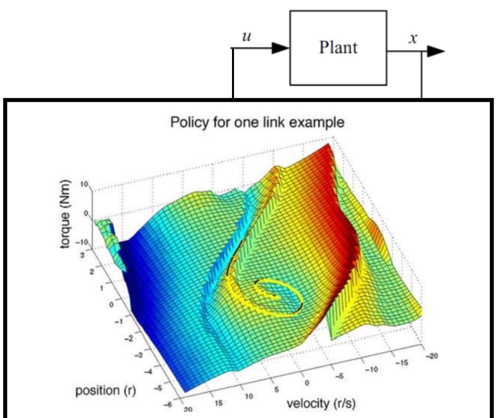
  → DOF>actuators: car-like robot, airplane, cart and pole, etc

  ➔ Actuator saturation!

- *Why Now?*
  1. State-of-the-art (LQR-Trees 2009 RSS best paper, kNitro, SDPARA).
  2. Convex Optimization (c/o relaxation) is ~ "online-able"

Viewing This From a Controls & Policy Perspective





Atkeson, C. & Stephens, B.
"Random sampling of states in dynamic programming" IEEE T. Syst. Man Cybern, 2008

Core Idea:

Feedback motion planning (Assistance function) built from a prior model and updated online

Gain-Scheduled RRT

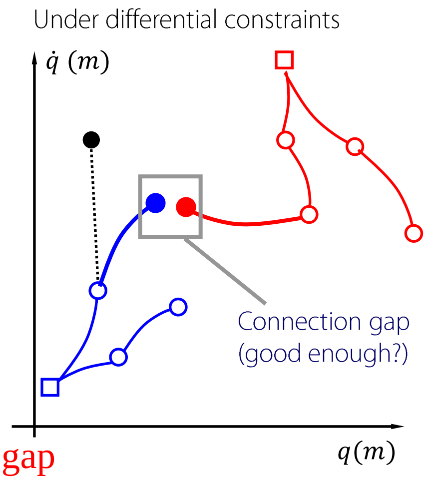
- Rapidly Exploring Random Trees (RRT) (Background):

Features (+):

1. Solve a control problem
2. Scalable
3. Constrained environments

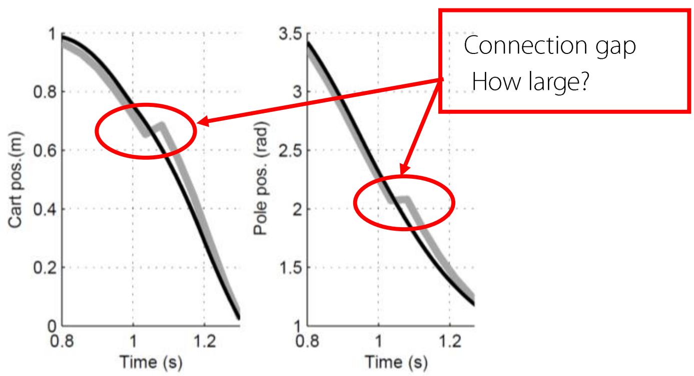
Concerns (--):

4. Works poorly under differential constraints
5. Hard to avoid the connection gap



Gain-Scheduled RRT: RRT Connection Gap

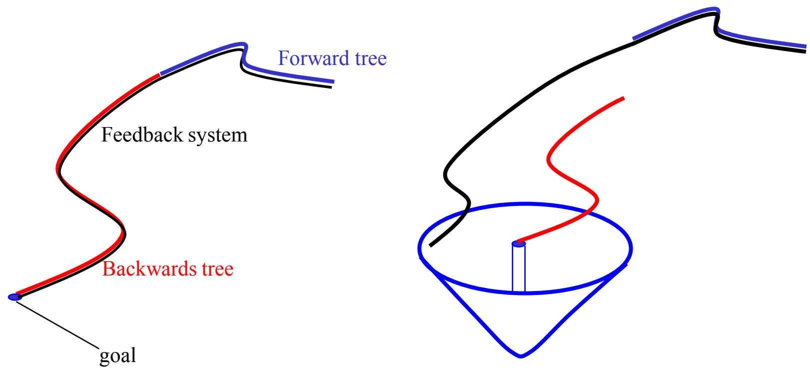
- A RRT solution rarely reaches the goal (or connect the two trees) with zero error



Gain-Scheduled RRT: Relaxing the Search

Single state search:
Extensive exploration

Region search
RRT connection is relaxed



Gain-Scheduled RRT: RoA & Verification

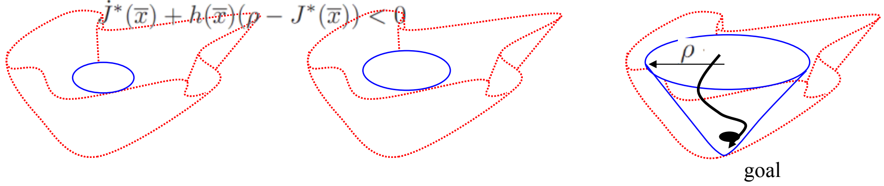
- Find a candidate

  In the LQR case: J: optimal cost-to-go S: Algebraic Ricatti Eq.

  $$V(x) = J^*(x_{goal} - x) = (x_{goal} - x)^T S (x_{goal} - x)$$

- Maximize candidate (ρ)

  $$\dot{J}^*(\bar{x}) + h(\bar{x})(\rho - J^*(\bar{x})) < 0$$



- Verify candidate

  Sum of squares relaxation

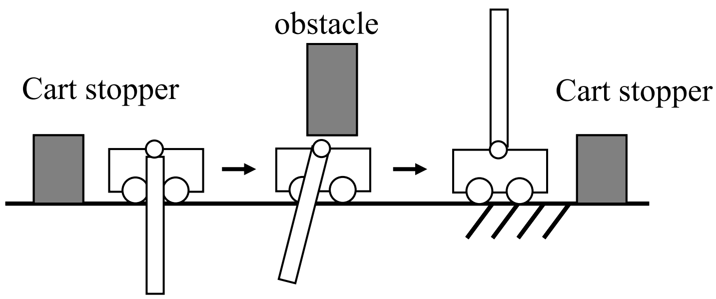
  $V(x)$ locally $\boxed{\text{positive definite}}$ in B_ρ

  $\dot{V}(x)$ locally $\boxed{\text{negative definite}}$ in B_ρ

**R. Tedrake, *"LQR-Trees: Feedback Motion Planning on Sparse Randomized Trees"*, RSS 2009

Gain-Scheduled RRT: Result

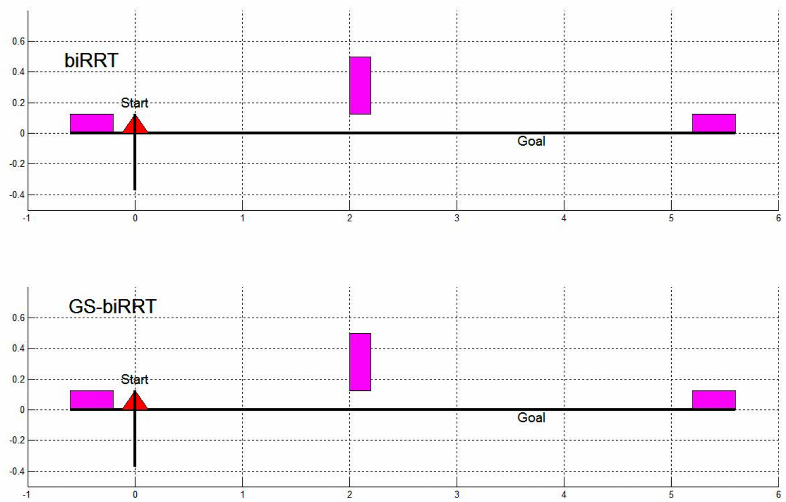
- Cart and pole in a cluttered workspace …



Same initial and final conditions.

Every solution is different due to the random sampling

Gain-Scheduled RRT: Result



Cool Robotics Share