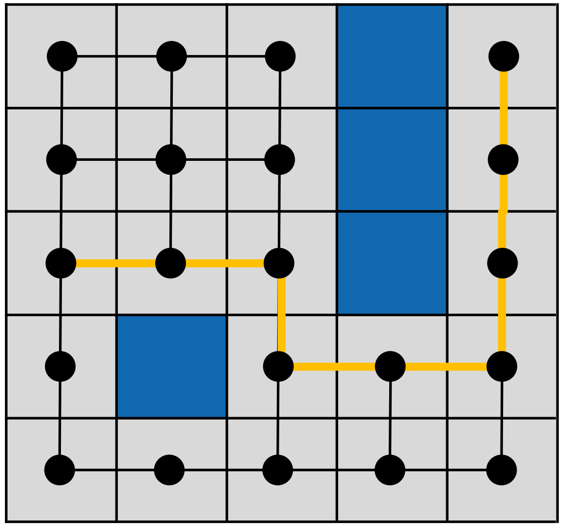
ETH zürich

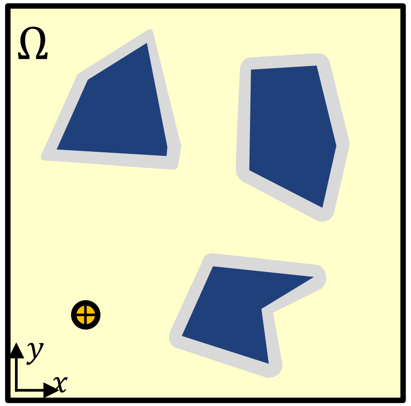
Planning II
- sampling-based motion planning

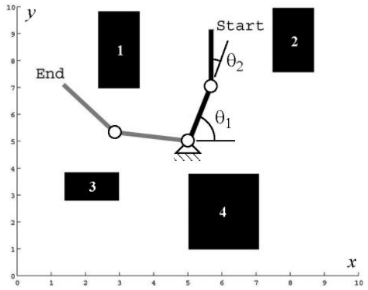
Roland Siegwart, Margarita Chli, Juan Nieto, **Nick Lawrance**

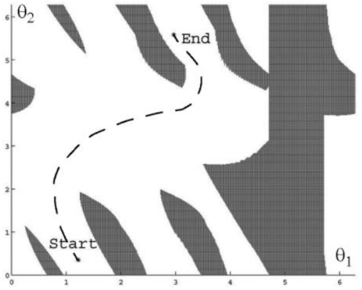
Last time

- Motion planning
  - Aim - Design a feasible open-loop trajectory that satisfies global obstacles constraints
  - Representation – how to define the robot's understanding of the world, and ensure that it is sufficient to complete the task
  - Configuration space – the space defined by the robot's state description + environment





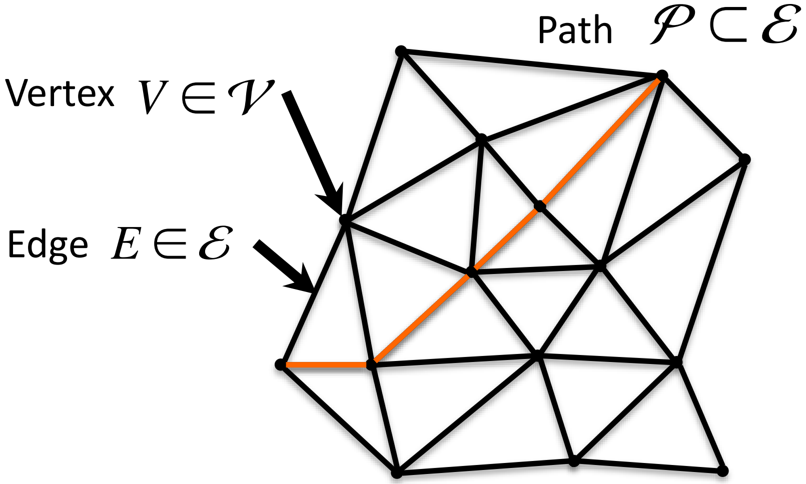




Last time

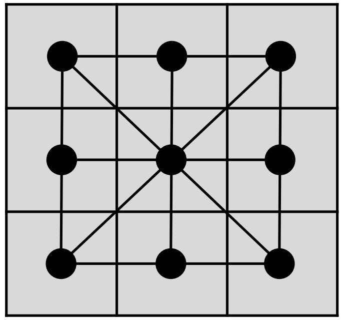
- Graph search methods
  - Graphs are constructions of vertices and connecting edges
  - Graph search techniques are used to find low-cost paths through graphs
  - Breadth-first and depth-first search – complete searches from start (**unweighted graphs**)
  - Djikstra – search outwards in order of cost from start (**weighted graphs**)
  - A* – focused search that **prioritises searching towards the goal** using an admissible heuristic

Graph $G = (\mathcal{V}, \mathcal{E})$



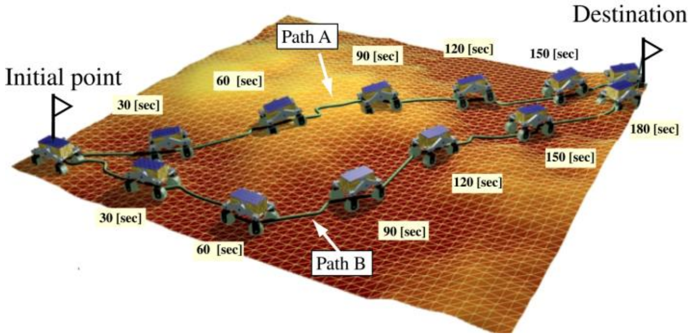
Why anything else?

- For grid-based graph plans:
  - Can be difficult to select a grid resolution to compromise between being sufficiently fine to retain valid solutions (obstacles inflated by grid size), and sufficiently coarse to find a solution in a reasonable time
  - Connectivity being defined by the grid shape can be limiting



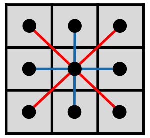
- Especially challenging in higher dimensional spaces – grids suffer badly from the *curse of dimensionality* (exponential growth in number of cells)

Dimensionality problem in grids

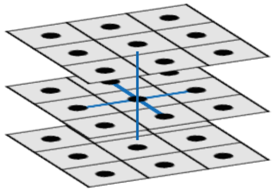


- Time complexity of breadth-first algorithm in a uniform grid as a function of the number of dimensions $O(|V| + |E|)$

- Number of nodes in a 2D grid 100x100 = 10^4, cells have 4 direct face-neighbours, 8 diagonal



- Number of nodes in a 3D grid 100x100x100 = 10^6, cells have 6 direct face-neighbours, 26 diagonal



- Number of nodes in a 6D grid 100 cells per dimension = 10^{12}, …

Note: in practice there are more efficient multi-resolution representations that can be used

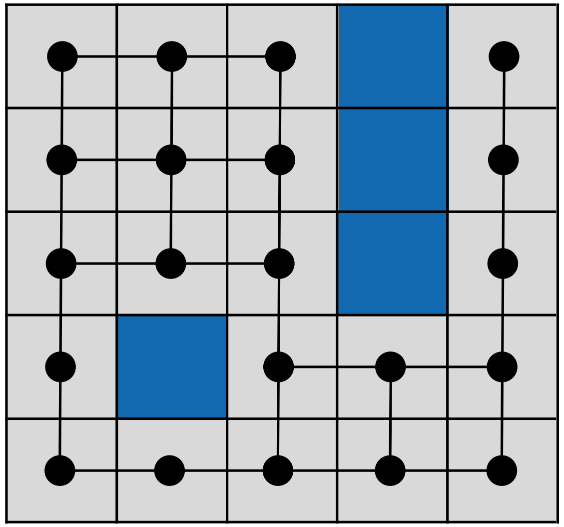
Sampling-Based Planning

- To get around the problem of trying to divide the region **uniformly** into small cells, one approach is to **randomly sample** locations in the space and try and connect the samples

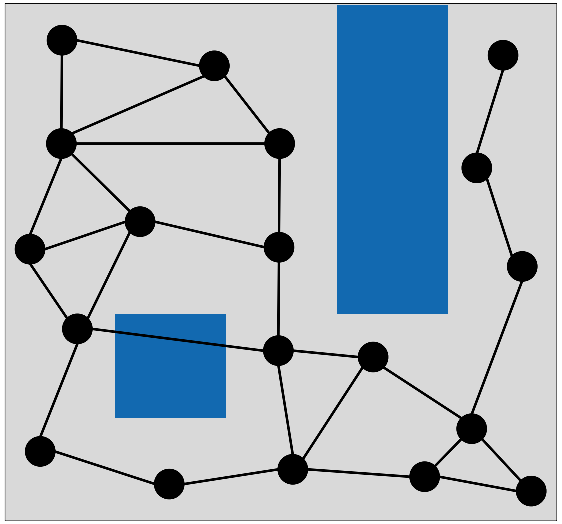
- Often, a larger proportion of the working volume is free space, so if two points are ‘near’ each other, it is often the case that they can be connected by a simple path (e.g. straight line)

- Basic idea of sampling-based planning
  - Abandon the concept of explicitly characterizing C_{free} and C_{obs}
  - A collision detection algorithm probes C to see whether some configuration lies in C_{free}

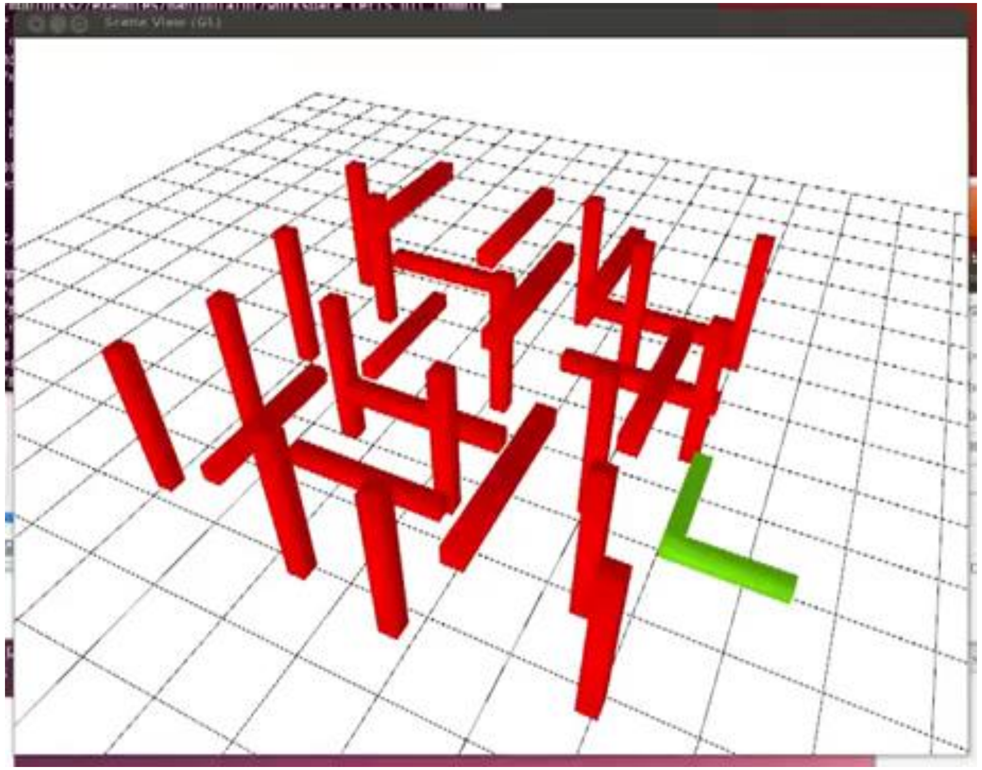
- Explicitly use obstacles and grid to build valid graph



- Use collision checks to build valid graph



Sampling-based planning example: Piano Mover’s Problem



https://www.youtube.com/watch?v=3_S3GPxAMYA

https://www.youtube.com/watch?v=HdfAzUXvmOQ

Today

1. Sampling-based methods
   - PRM
   - RRT

2. Planning under differential constraints (using RRT)

3. Review of related work and research papers

Main reference: LaValle, Steven M. *Planning algorithms*. Cambridge university press, 2006 (Ch 5 and Ch 14)

SAMPLING-BASED MOTION PLANNING

Motivation for sampling-based

- The previous methods rely on an explicit representation of the obstacles in configuration space

- This may result in an excessive computational burden in:
  a. high-dimensions, and
  b. environments with a large number of obstacles

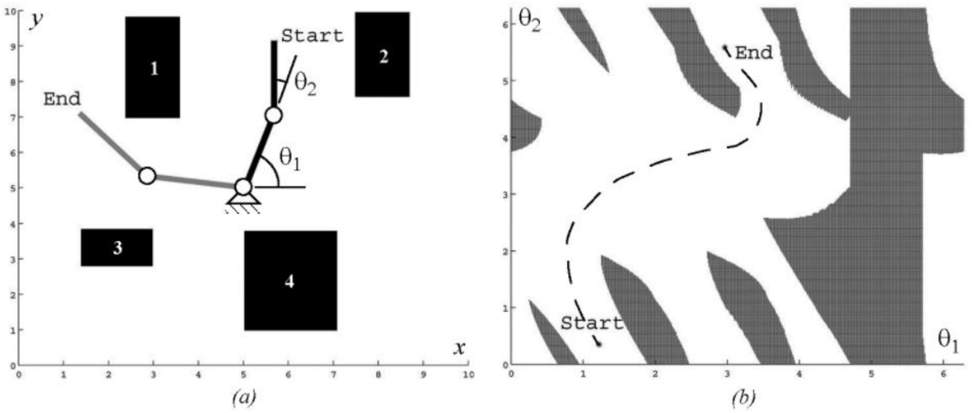


Figure 6.1
Physical space (a) and configuration space (b): (a) A two-link planar robot arm has to move from the configuration *start* to *end*. The motion is thereby constraint by the obstacles 1 to 4. (b) The corresponding configuration space shows the free space in joint coordinates (angle θ_1 and θ_2) and a path that achieves the goal.

Motivation for sampling-based

- Avoiding such a representation is the main underlying idea of sampling-based methods

- Instead of using an explicit representation of the environment, sampling-based methods rely on a collision-checking module

Motivation for sampling-based

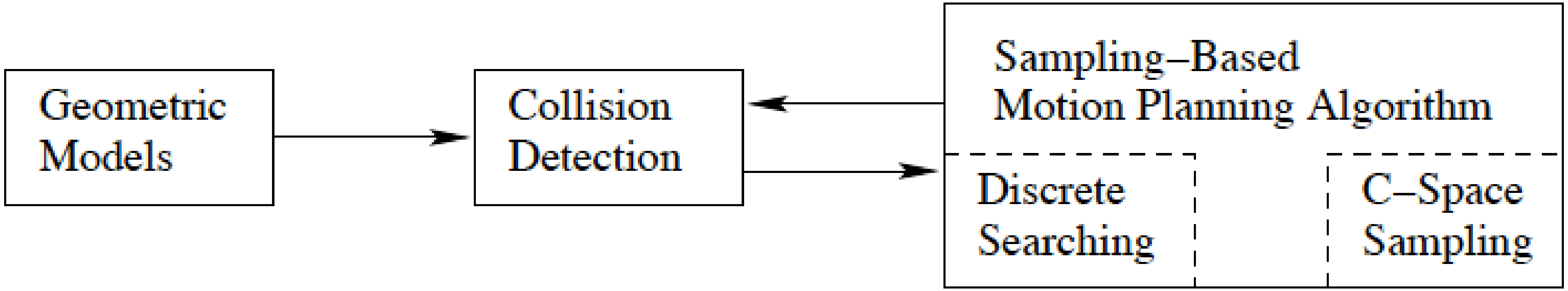


Figure 5.1: The sampling-based planning philosophy uses collision detection as a "black box" that separates the motion planning from the particular geometric and kinematic models. C-space sampling and discrete planning (i.e., searching) are performed.

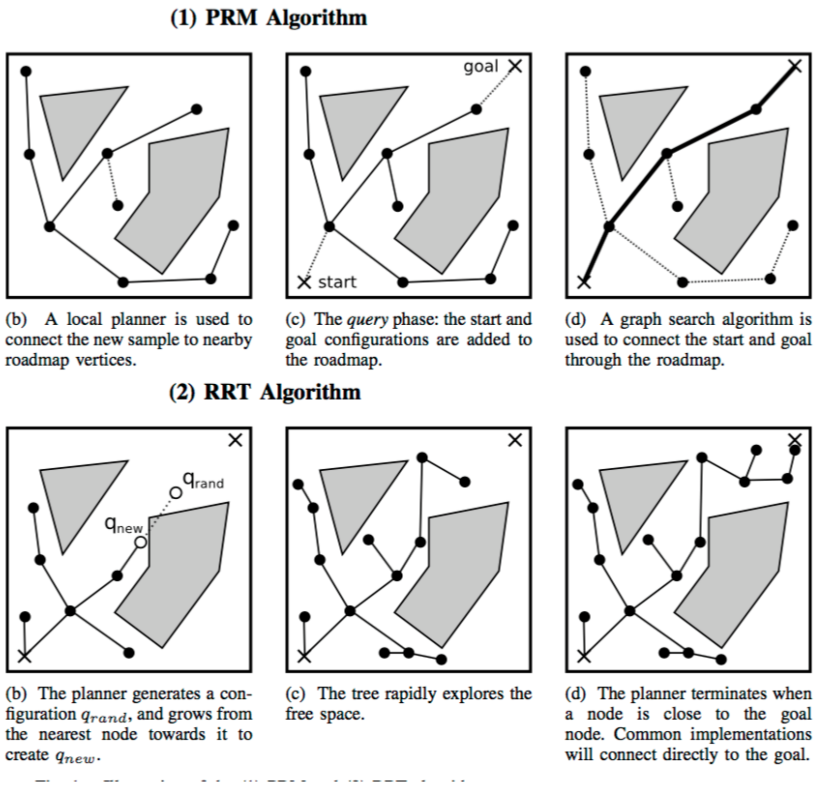
LaValle, Steven M. *Planning algorithms*. Cambridge university press, 2006

Sampling-based Motion Planning

- The two most influential sampling-based motion planning algorithms to date are:
  - Probabilistic Roadmaps (PRM, 1996)
  - Rapidly-exploring random trees (RRT, 1998)

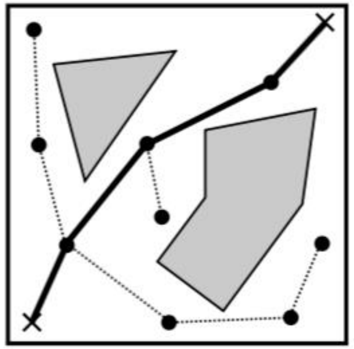
PRM:
1. Roadmap construction
2. Search

RRT:
1. Roadmap construction and search online



Recent Progress on Sampling Based Dynamic Motion Planning Algorithms, Short et al, AIM 2016

The general framework



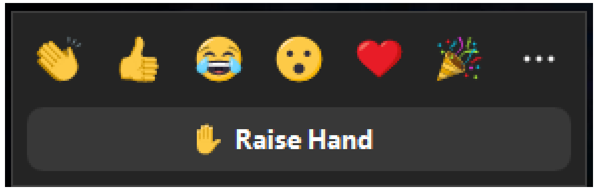
(d) A graph search algorithm is used to connect the start and goal through the roadmap.

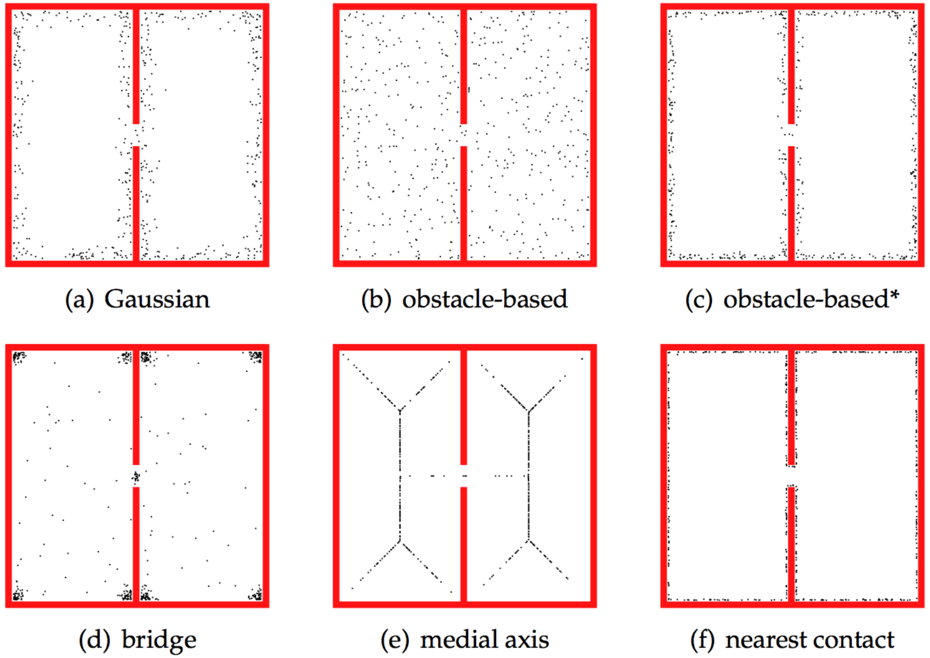
- Incremental Sampling and Searching:

1. Initially, the graph is empty $G = (V, E)$
2. Then, a **random configuration** is generated and added to V
   - <u>Need a Sampling algorithm</u>
3. For every new vertex c, select a number of nodes from V ‘near’ c and try to connect c to each of them using a **local planner**
   - <u>Need to define a distance function</u>
4. If local path is **collision free**, a new edge is added to E
   - <u>Need collision detection approach</u>
5. Check for a solution (connection between origin and target)
6. Return to step 2

Sampling

- Random uniform sampling is the easiest of all sampling methods in C-spaces
  - Combining samples from different dimensions can combined to provide samples of the whole space
- Uniform sampling is typically used, however there are others based on different heuristics:





Sampling-based Motion Planning: Analysis and Path Quality, R. Geraerts PhD Thesis, 2006

Metrics

- Sampling-based algorithms need to get a ‘sense of proximity’ among samples in configuration space
  - What does it mean to be the closest? This depends on your C-space

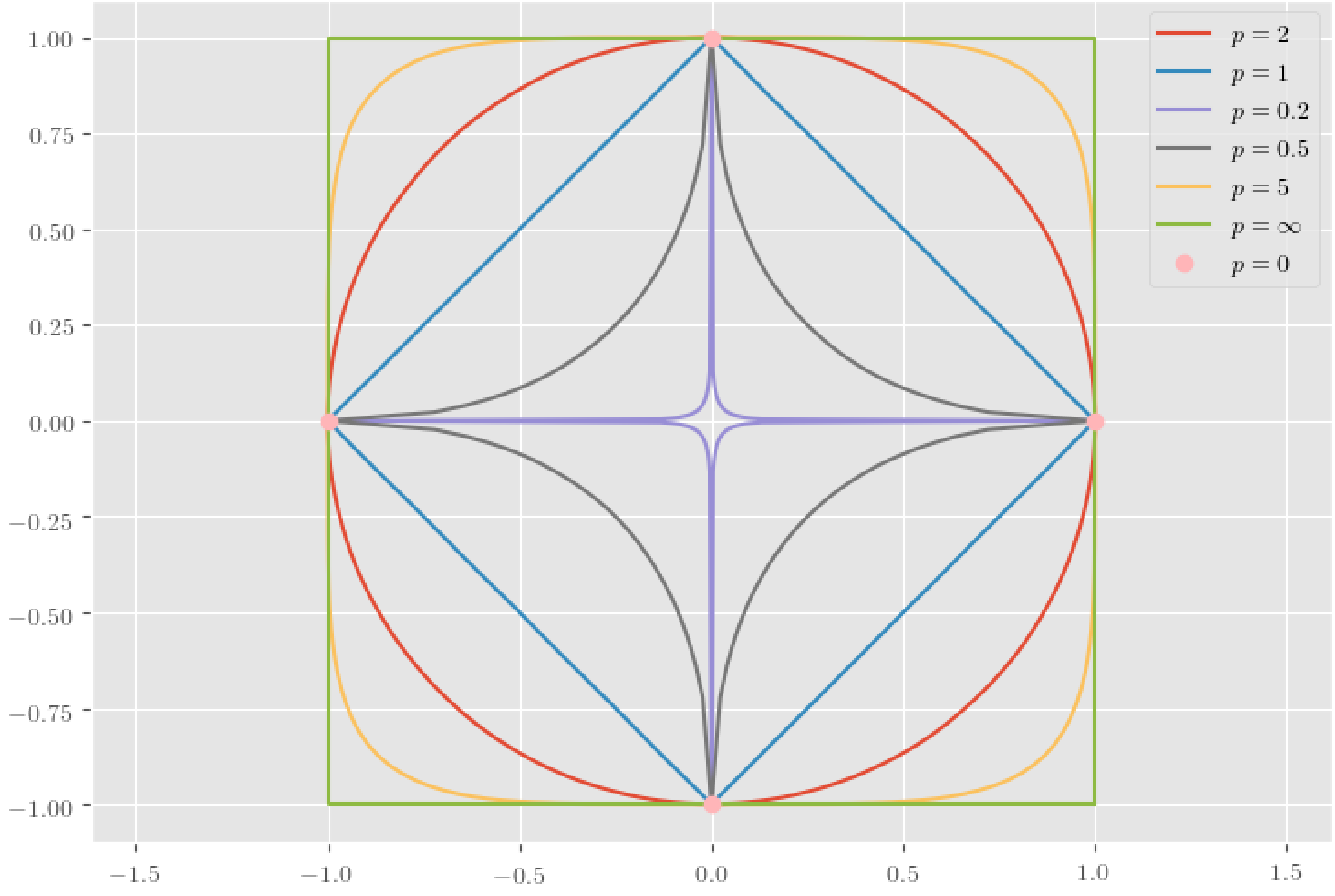
- In virtually all sampling-based algorithms performance depends on the choice of metric

L_p **metrics** The most important family of metrics over $\mathbb{R}^n$ is given for any $p \geq 1$ as

$$\rho(x, x') = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}. \tag{5.1}$$

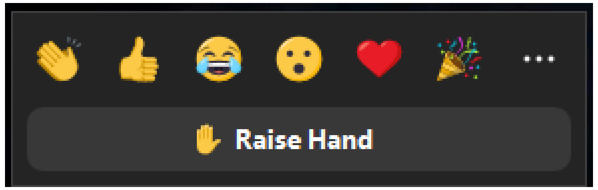
l_p-norm

Unit circle for l_p-norms, $||x||_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$



Metrics

- In most robotic problems, the configuration space will include Euclidean points plus angles $(x, y, z, \phi, \theta, \psi)$, how should you combine these distances?
  - The simplest approach is to perform a weighted combination between rotational and translational distances



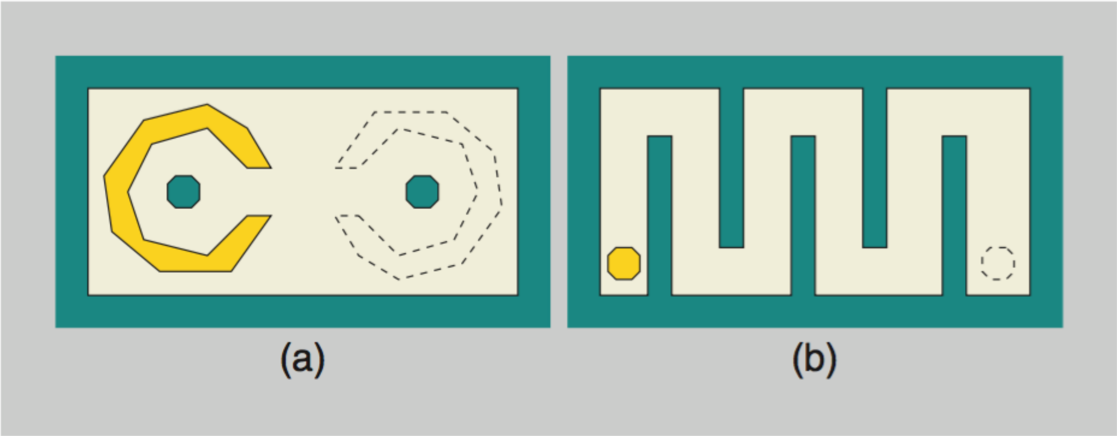


Figure 11. Rotation versus translation domination: (a) The task is to move the C shape to the right. Rotation dominates. Performance should improve if rotation is weighted heavily in the metric. (b) In this case, the translation dominates and should therefore be weighted more heavily if this fact is known in advance.

Motion Planning Tutorial, Part 1, LaValle, 2011

Collision Detection

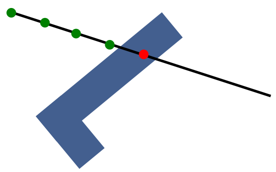
- Once it has been decided where the samples will be placed, the next problem is to determine **collisions**

- In most planning applications, the majority of computation time is spent in collision checking

- A variety of collision detection algorithms exist,
  - ranging from theoretical algorithms that have excellent computational complexity
  - to heuristic, practical algorithms whose performance is tailored to a particular application

Collision Detection

- Examples:
  - Two-phases methods: broad phase, narrow-phase
  - Hierarchical methods: decompose each body intro a tree, each vertex represents a bounding region
  - Checking a path segment: by sampling the interval



- If objects and robot can be modeled as convex, we can do linear-time collision detection

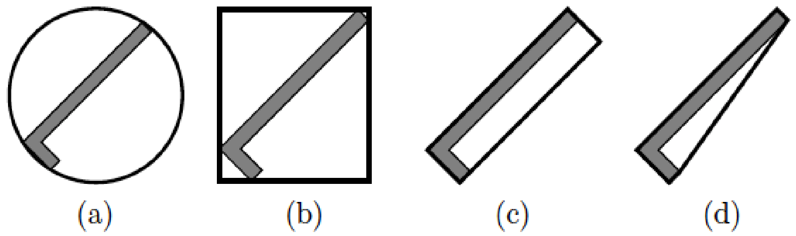
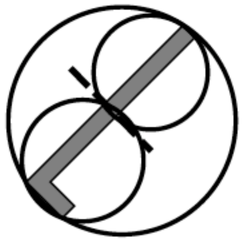


Figure 5.9: Four different kinds of bounding regions: (a) sphere, (b) axis-aligned bounding box (AABB), (c) oriented bounding box (OBB), and (d) convex hull. Each usually provides a tighter approximation than the previous one but is more expensive to test for overlapping pairs.



Collision Detection

- New representations have been adopted in recent years that facilitate collision checking
  - At the cost of spending more time in building the representation

- One example is the Euclidean Signed Distance Function (ESDF)
  - https://www.youtube.com/watch?v=PlqT5zNsvwM

PROBABILISTIC ROAD MAPS (PRM)

Probabilistic Road Maps

Kavraki, Lydia E., et al. "Probabilistic roadmaps for path planning in **high-dimensional configuration** spaces." *IEEE Transactions on Robotics and Automation* 12.4 (1996): 566-580.

Probabilistic Road(M)aps (PRMs):

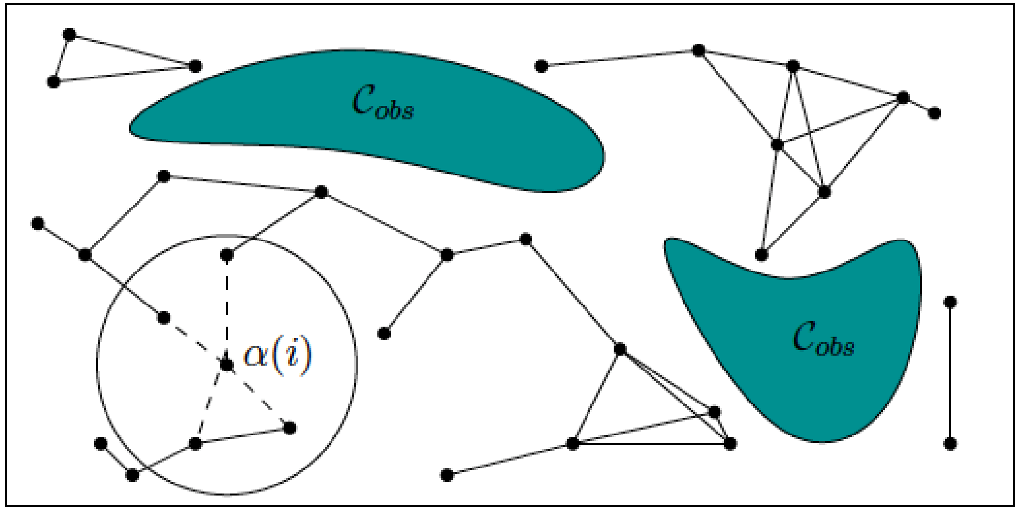
- ***Step 1:*** Build a roadmap by connecting nearby (sampled, free-space) configurations using simple planners to construct a graph of valid path segments

- ***Step 2:*** Query; Search the graph using a graph search technique (A*)

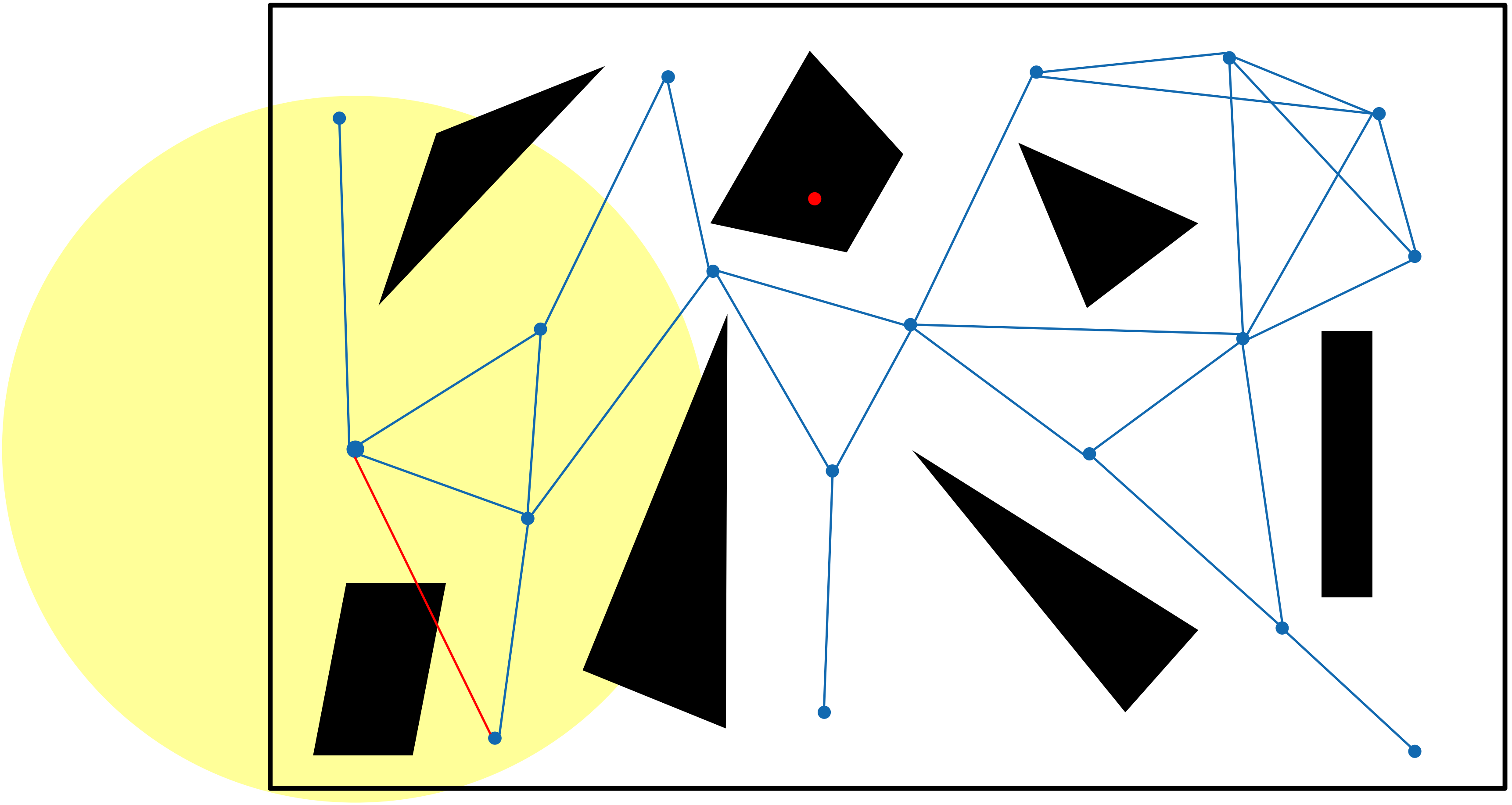
PRM: Step 1, building the roadmap

BUILD_ROADMAP

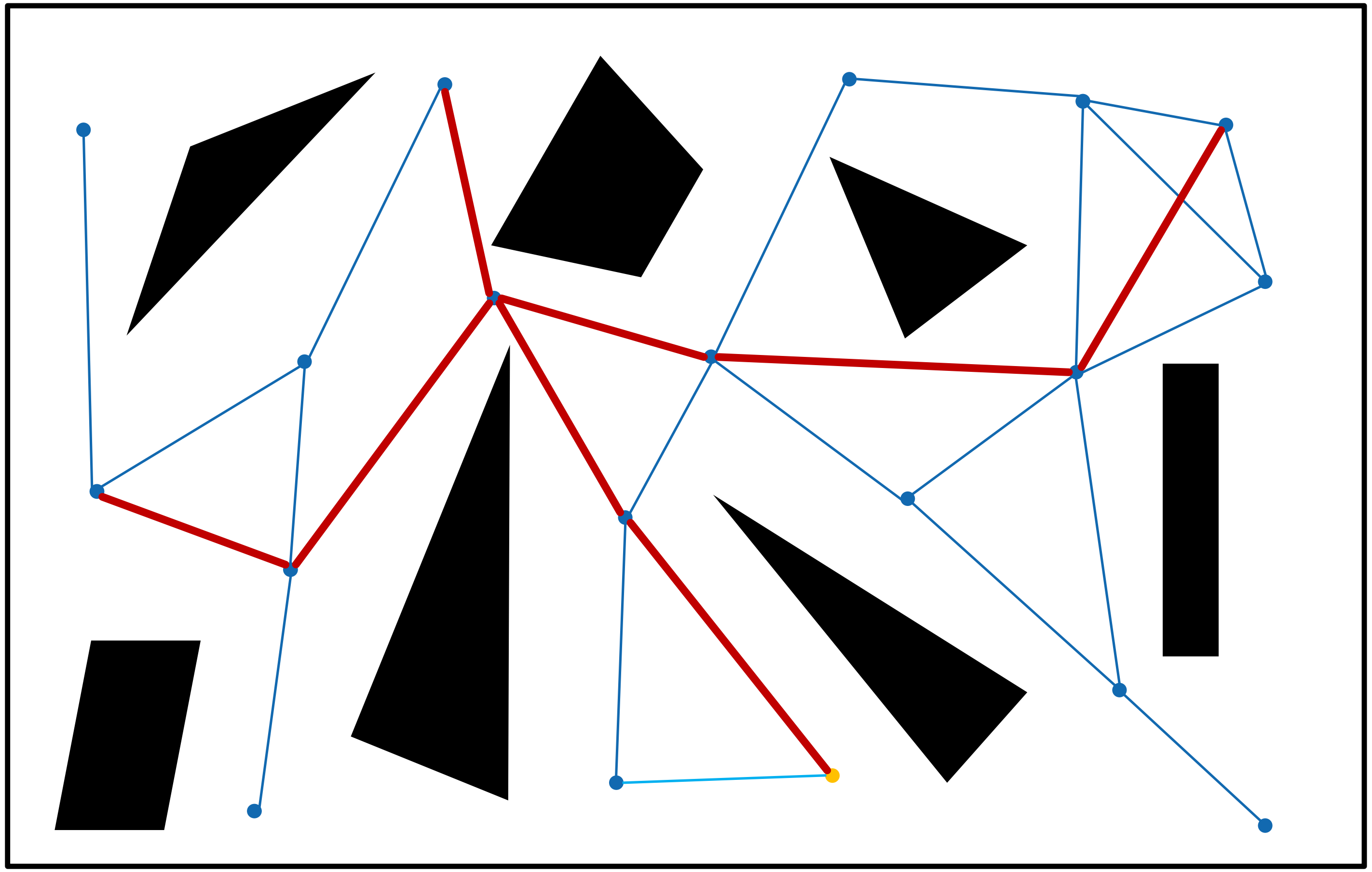
$\mathcal{G}$.init(); $i \leftarrow 0$;
while $i < N$
  if $\alpha(i) \in \mathcal{C}_{free}$ **then**
    $\mathcal{G}$.add_vertex($\alpha(i)$); $i \leftarrow i + 1$;
    for each $q \in$ NEIGHBORHOOD($\alpha(i)$,$\mathcal{G}$)
      if ((**not** $\mathcal{G}$.same_component($\alpha(i), q$)) **and** CONNECT($\alpha(i), q$)) **then**
        $\mathcal{G}$.add_edge($\alpha(i), q$);



PRM



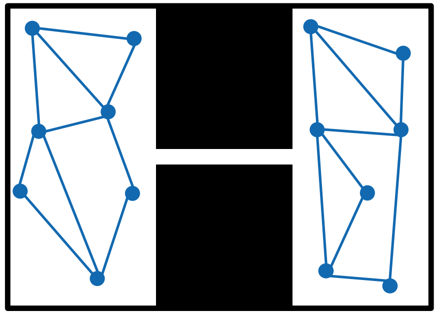
PRM



PRM

- Conceptually very simple, and became rapidly popular due to PRM’s ability to solve previously challenging problems (high-dimension planning, snake robots, piano-mover)

- A few limitations with the original method, most of which were largely subsequently solved in derivative variations
  - Requires holonomic motion
  - Can suffer from ‘narrow passage problem’
  - Not suited for dynamic environments



PRM Summary

- A graph is constructed in the configuration space by connecting random configurations
- PRM is designed for **multiple-queries**
- The underlying assumption is that it is worth performing **substantial pre-computation** on a given environment to enable multiple planning queries

- However, often we are interested only in a **efficient single-query** without pre-computations
  - E.g. in dynamic environments, different robots model
- In this case we want to **combine exploration and search** into a single method
  - This motivates tree-based planning approaches

TREE-BASED PLANNERS

Rapidly-exploring Random Trees (RRT)

- Problem – How do we connect all these disjoint paths?
- Solution – Instead of sampling all our target states at once, we start from a start configuration, and **incrementally** grow outwards from there – forming a tree of connected, dynamically feasible local paths
- Terminate when we reach a configuration in a goal **region**
- For **single-query searches**

S. M. LaValle. *Rapidly-exploring random trees: A new tool for path planning*. TR 98-11, Computer Science Dept., Iowa State Univ., Oct. 1998.

Tree-based Planners

- There exist many types of sampling-based planners that create tree structures of the free-space
  - E.g. RRT (LaValle 1998), EST (Hsu, 2001), SBL (Sanchez, 2005), KPIECE, etc
- In general
  - Start with a root node
  - Employs an expansion heuristic (typically gives the name to the method)
  - Connect node to tree if collision free path
- Note: the methods often bias the expansion of the tree toward the goal state

Tree-based vs Roadmap-base planners

- The tree-based is for single-query planning
- In roadmaps, when planning with differential constraints, it's not easy to encode control information (undirected edges)
- Tree-based, on the other hand, can include complex dynamics in their directed graphs
  - Control information can be encoded in each edge

Rapidly Exploring Dense Trees (RDT)

- RDTs are an incremental sampling and searching approach that gives good performance “without any parameter tuning”
- The idea is to incrementally construct a search tree that gradually improves the resolution
- A dense sequence of samples is used to incrementally build the tree
  - If the sequence is random, is called rapidly exploring random tree (RRT) (LaValle 2001)

```
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));
```

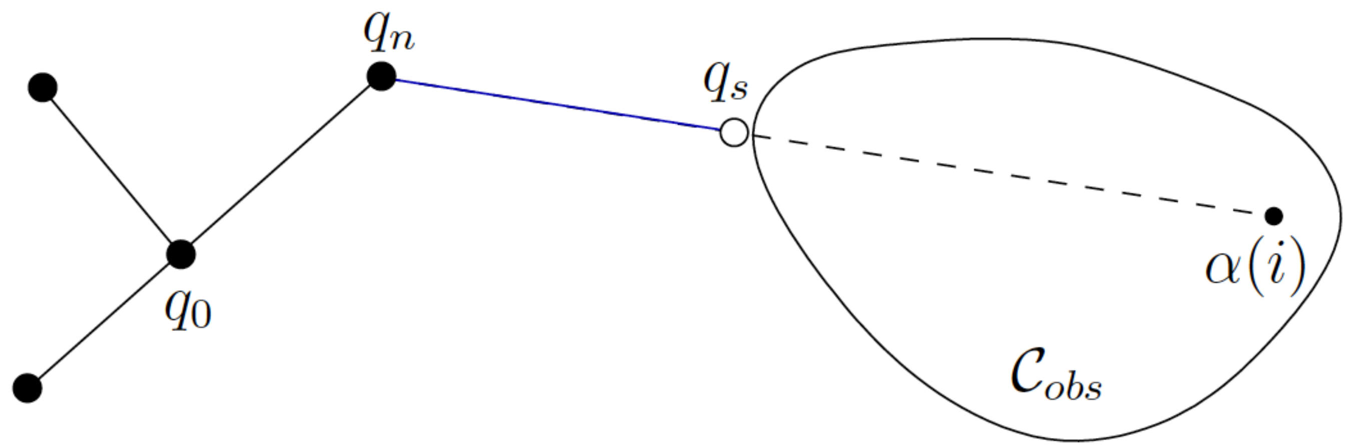
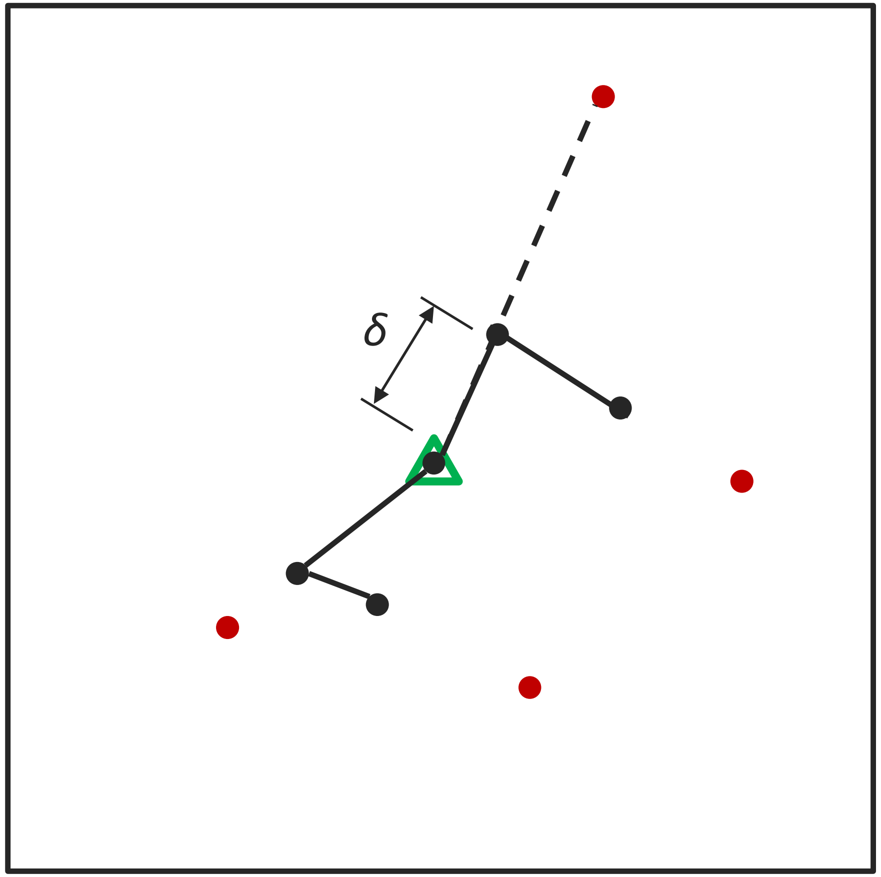


Figure 5.20: If there is an obstacle, the edge travels up to the obstacle boundary, as far as allowed by the collision detection algorithm.

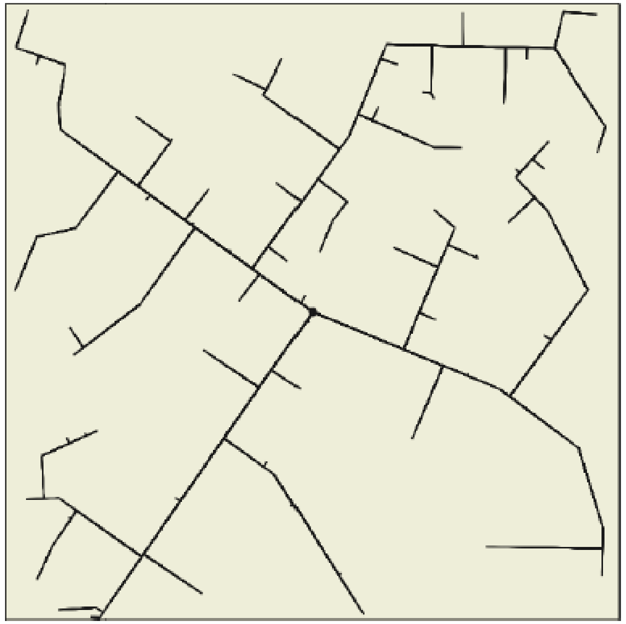
RRT – Simple case

- Let's consider a simple case, where the motion is unconstrained (holonomic)



RRT

- The main idea is to incrementally probe an explore the C-space
  - In the early iterations the RRT quickly reaches the unexplored parts
  - However, RRT is dense in the limit, which means that it will eventually get to any point in the space


45 iterations


2345 iterations

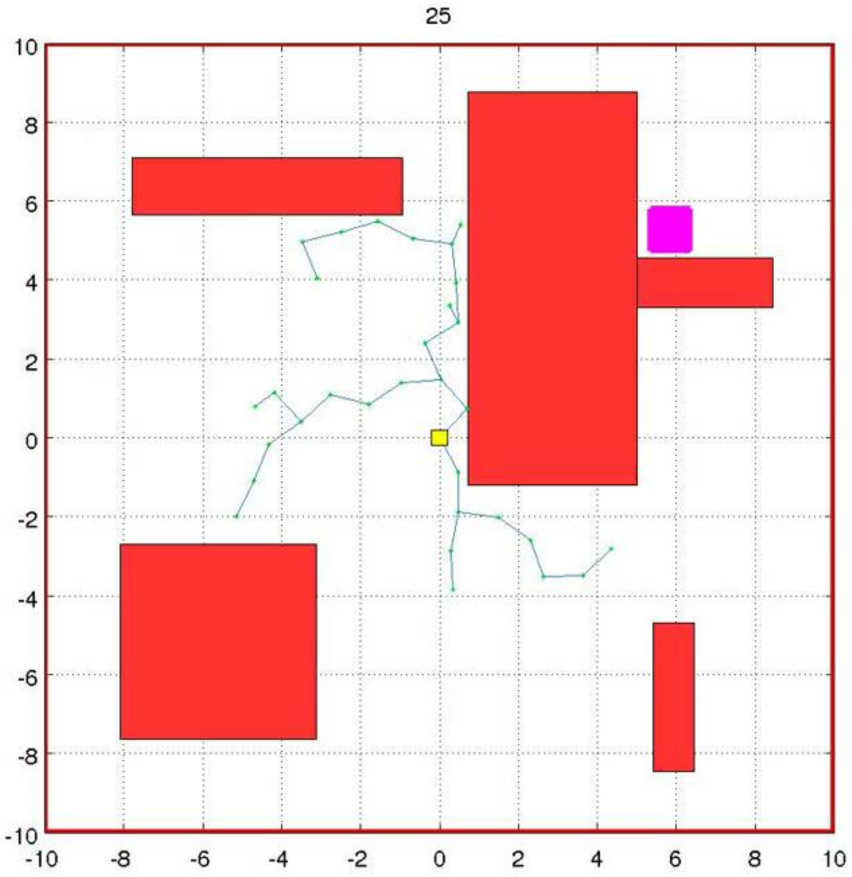
RRT

RRT growing towards the goal

Input: q_{start}, q_{goal}, number n of nodes, stepsize α, β
Output: tree $T = (V, E)$

1: initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
2: **for** $i = 0 : n$ **do**
3: **if** 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_{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**

RRT



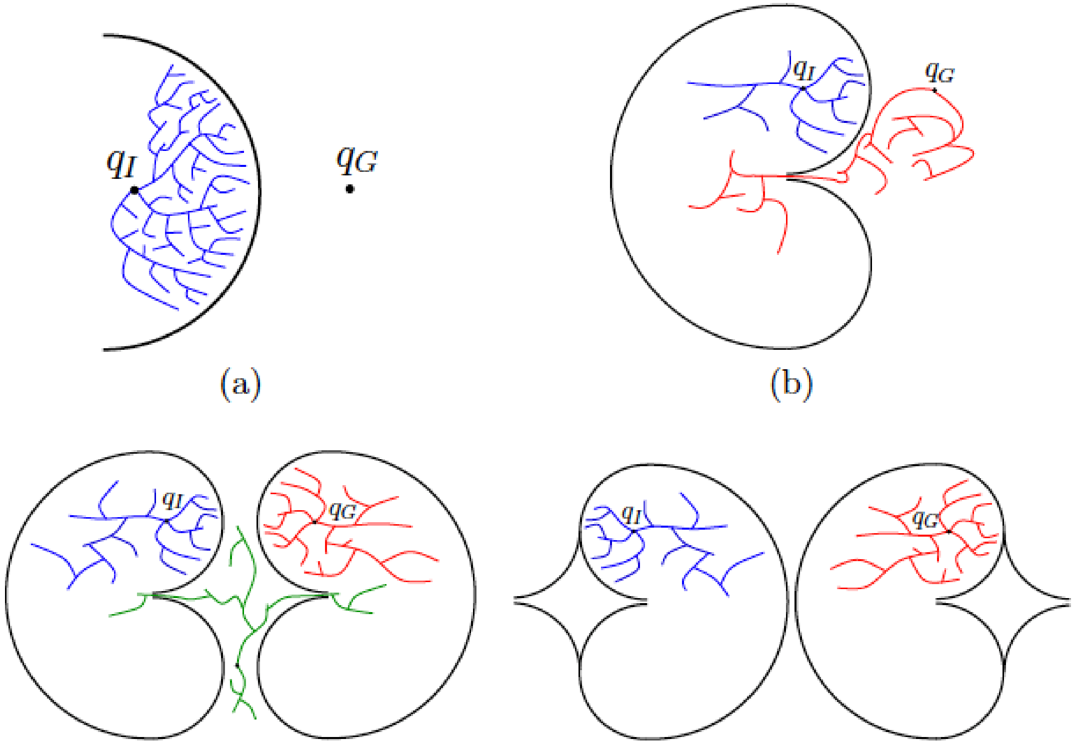
https://www.youtube.com/watch?v=FAFw8DoKvik

Number of search trees

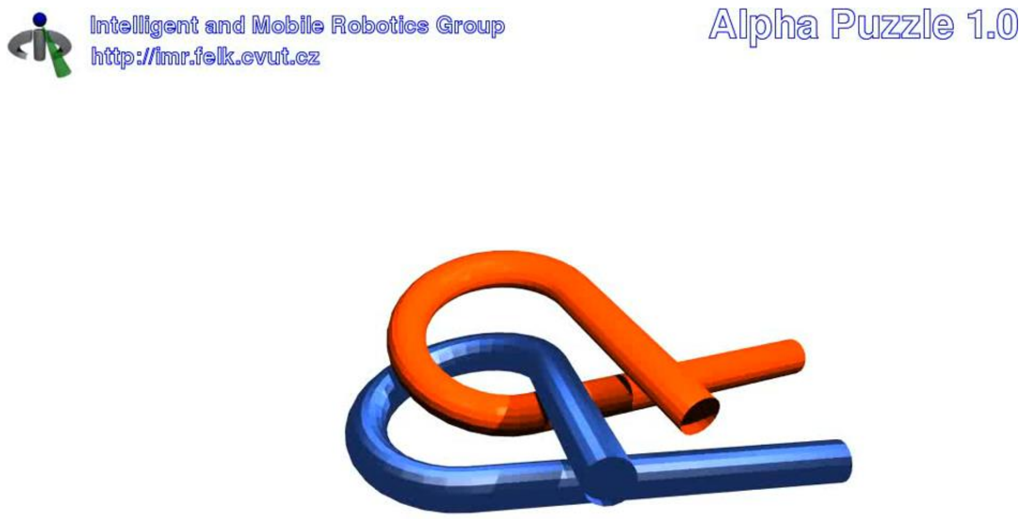
A. Unidirectional: single tree

B. Bidirectional: two-trees, one center in q_I and one in q_G

C. Multi-directional: for certain challenging problems, it makes sense to grow trees from other places (requires additional problem information)



(a) (b) (c) (d)



https://youtu.be/rPgZyq15Z-Q

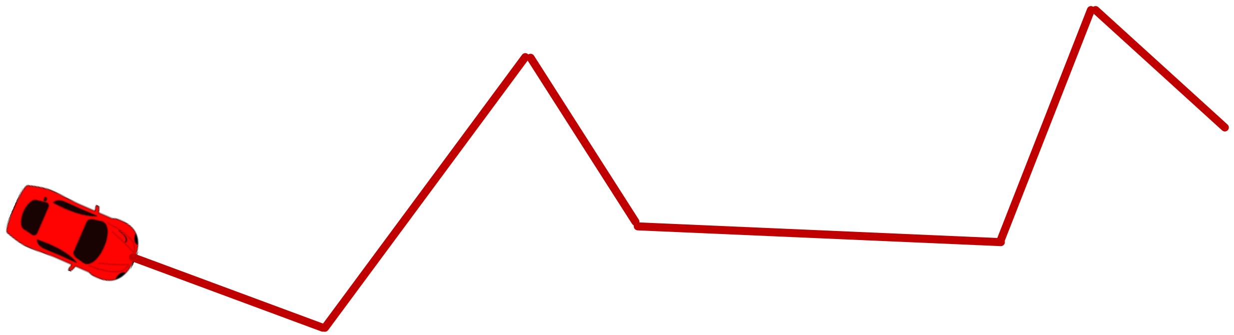
https://www.youtube.com/watch?v=MhTSIdQvy3I

PLANNING UNDER DIFFERENTIAL CONSTRAINTS

Local Planner: Non-holonomic planning

- So far we have connect vertices (samples) with a linear Local Planner
- The only constraints we considered are the obstacles
- In practice, many (most) robots will be non-holonomic, i.e. they have constraints (kinematics, dynamics) that cannot be integrated into positional constraints
  - The kind of different constraints that appear in planning depend not only on the system but also on how the task is decomposed
  - Non-holonomic planning is typically used for problems with kinematics constraints only

$$\dot{x} = u_s \cos\theta$$
$$\dot{y} = u_s \sin\theta$$
$$\dot{\theta} = \frac{u_s}{L} \tan u_\phi .$$



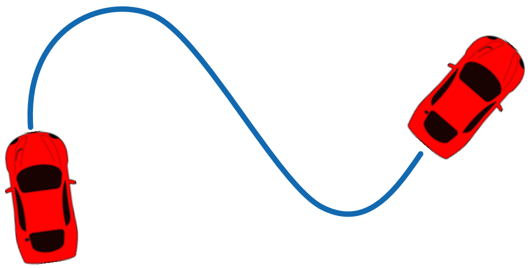
Local Planner: Non-holonomic motion

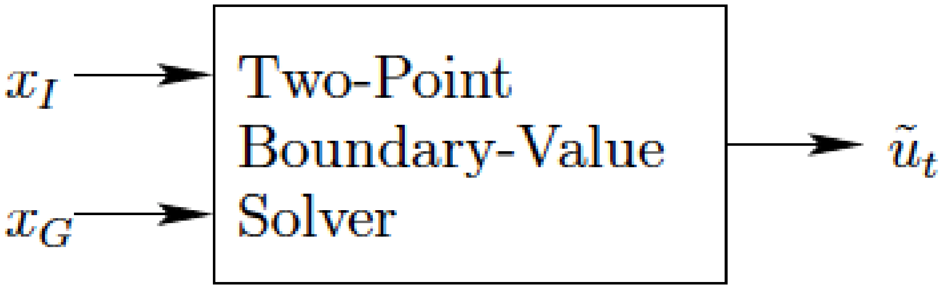
- Possible solutions:
  - **PRM**
    - A classic approach in robotics is to **decouple the problem** by solving a basic path planning and then finding a trajectory and controller that satisfies the dynamics and tracks of the path
    - PRM may require the connection of thousands of configurations to find a solution
    - Resolving a non-linear control problem for each connection seems impractical
  - **Randomized Potential Fields**
    - They depend heavily on the **choice of a good heuristic potential function**, which is very difficult when considering: obstacles, kinematic differential, and dynamic constraints
  - **RRT**
    - They can be **directly applied to non-holonomic planning**
    - This is because RRT does not require any connection between pairs of configurations

RRT

- RRTs are specifically designed to handle nonholonomic constraints and also dynamic constraints (Kinodynamic planning)

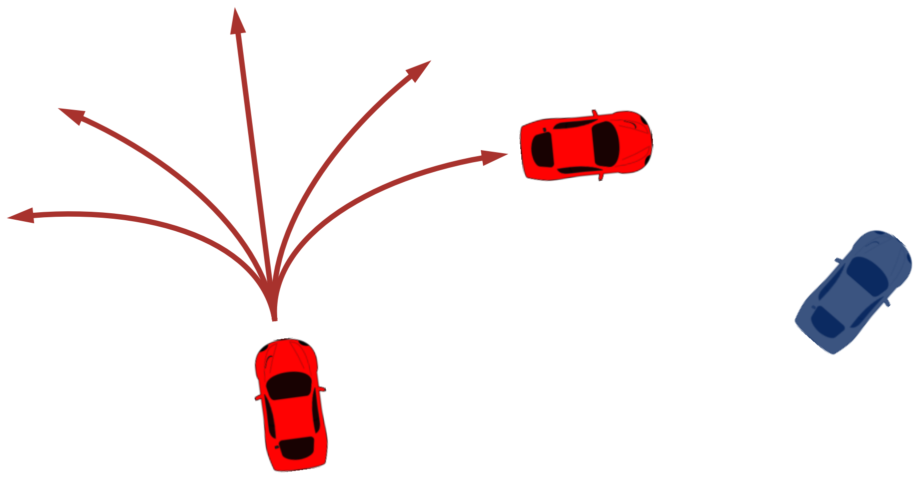
- Solution 1
  - Use a solver to find the control inputs that connect the two-points (two point boundary value problem)
  - Problem: it can be expensive, and provide solutions that are quite long in comparison with the shortest path



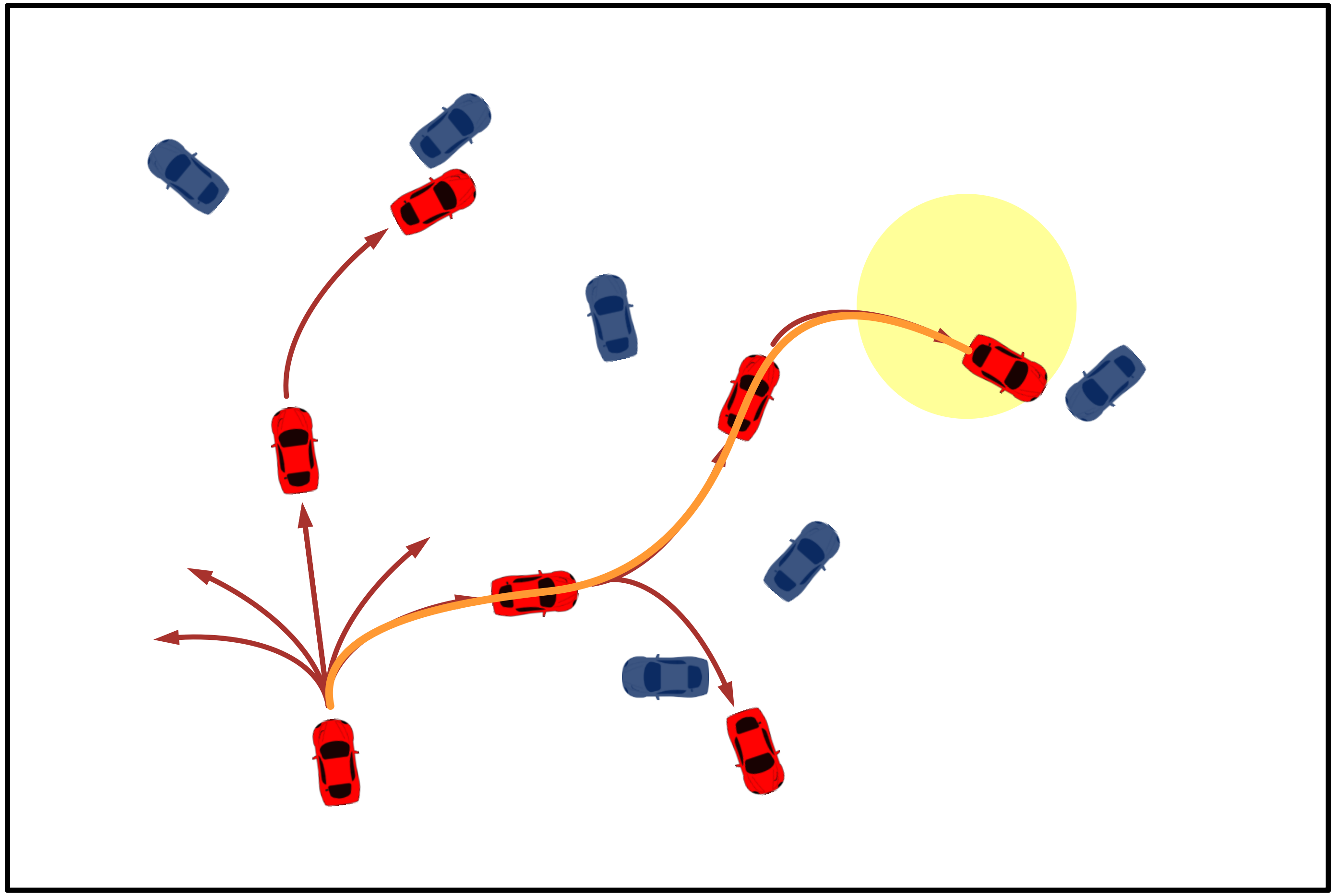


Rapidly-exploring Random Trees (RRT)

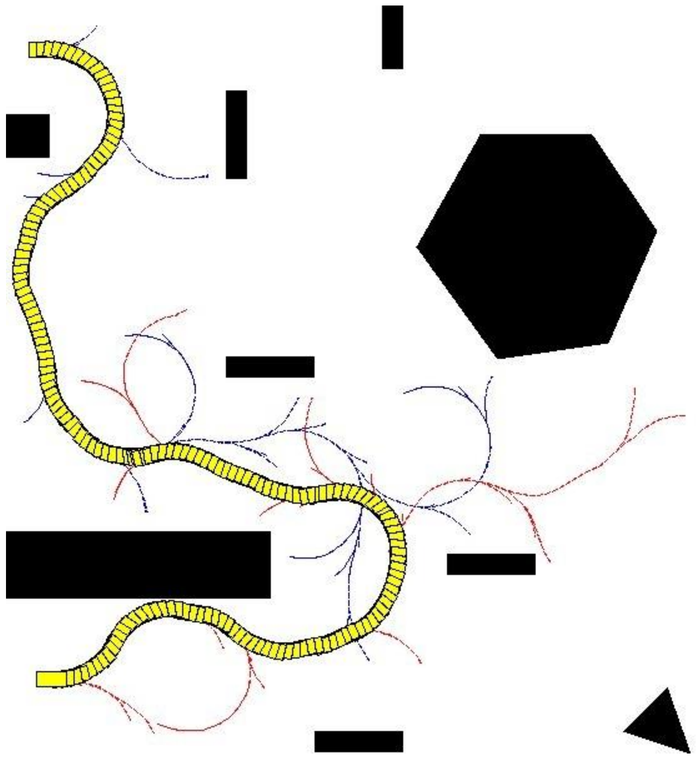
- Solution 2
  - We only require ***approximately*** reaching the intermediate goal states
  - How about instead of trying to join to specific samples, we just sample our controls to generate *feasible* local paths and choose the configuration that gets closest to the target configuration?

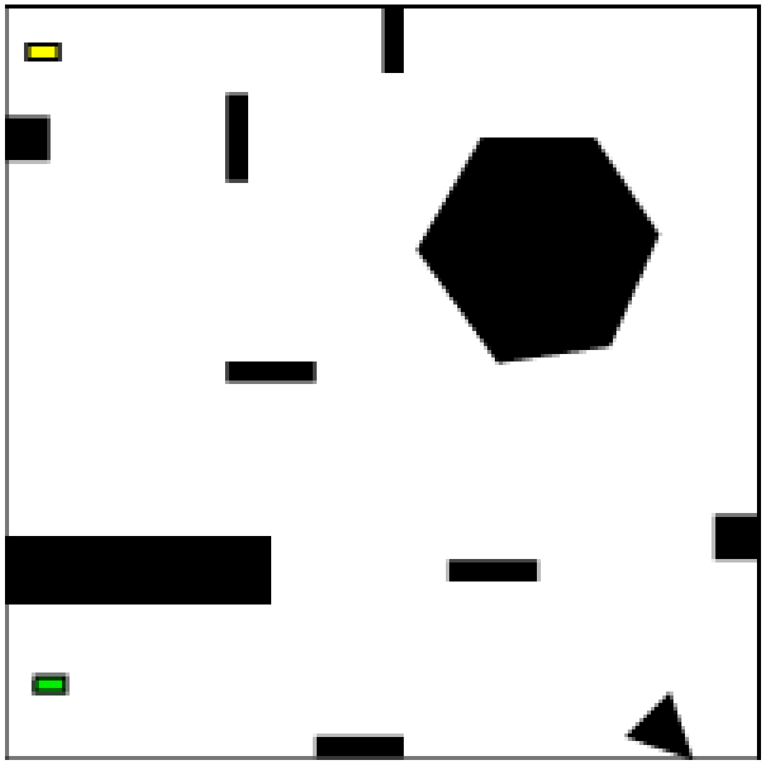


RRT



RRT Example





Steve LaValle (http://msl.cs.uiuc.edu/rrt/gallery.html)

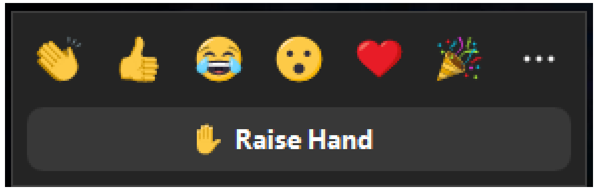
OPTIMALITY AND GUARANTEES

Theoretical guarantees

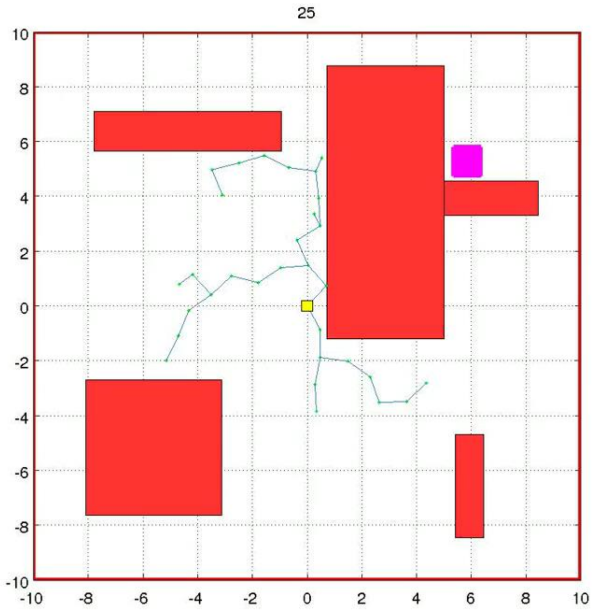
- PRM (and RRT) were known to be **probabilistically complete** fairly early on, namely that:

"If there exists a solution, it will be found with probability → 1 as the number of samples → ∞"

Will the RRT find the shortest path?



RRT



https://www.youtube.com/watch?v=FAFw8DoKvik

Theoretical guarantees

- PRM was known to be **probabilistically complete** fairly early on, namely that:

> "If there exists a solution, it will be found with probability $\rightarrow$ 1 as the number of samples $\rightarrow \infty$"

- However, it was observed that in practice, the resulting paths were obviously sub-optimal (tended to contain jagged bits). In fact, it was shown that they almost surely converge to a **non**-optimal solution

- They also lacked the ability to easily improve on an already generated path. You could restart the sampling from scratch, but not easily 'fix' an existing path (within the same framework)

A star rises

- Sertac Karaman and Emilio Frazzoli worked on incremental versions of PRM and RRT, and developed PRM* and RRT*, **asymptotically optimal** versions of PRM and RRT

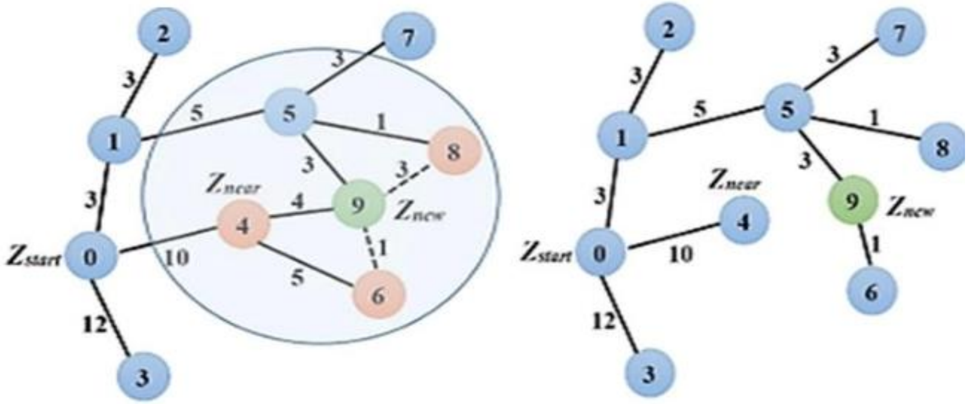
"The cost of the returned solution will approach the optimal as the number of samples $\rightarrow \infty$"

- The 'star' versions use a rewiring technique and a formal proof to select a connection radius as a function of sample density to provide these guarantees

Karaman, S. and Frazzoli, E. "Sampling-based algorithms for optimal motion planning." *The International Journal of Robotics Research* 30.7 (2011): 846-894.

RRT*

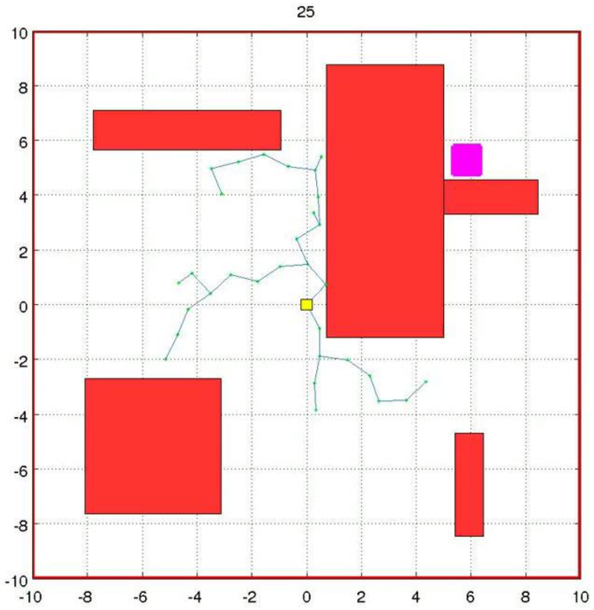
- RRT* inherits all the properties of RRT and works similarly to RRT. However, it introduced two promising features called near neighbor search and rewiring tree operations
  - Near neighbor operations finds the best parent node for the new node before its insertion in tree
  - Rewiring operation rebuilds the tree within this radius of area k to maintain the tree with minimal cost between tree connections



Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions, Noreen, 2016

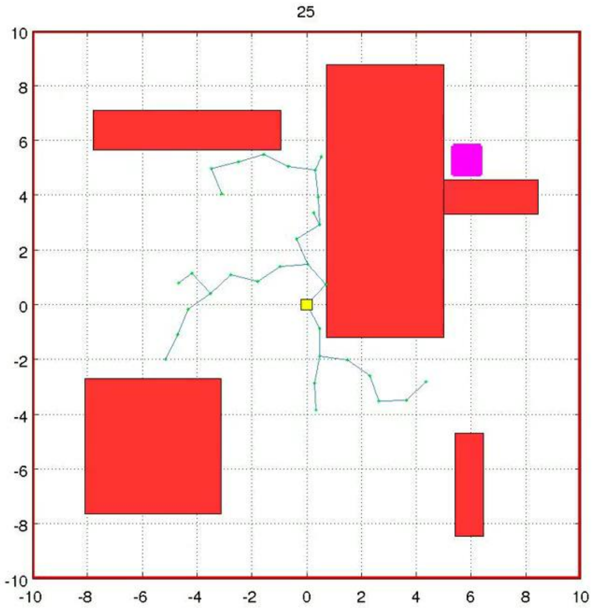
RRT* Comparison

RRT



https://www.youtube.com/watch?v=FAFw8DoKvik

RRT*



https://www.youtube.com/watch?v=YKiQTJpPFkA

The many (many) version of PRM and RRT

- The popularity of sampling-based planners has lead to a huge variety of probabilistic sampling-based planning methods

- Most focus of finding more optimal solutions faster, sometimes at the cost of formal guarantees

- Examples include:
  - Searching from start and goal and meshing the two trees
  - Limiting the search space once a plan has been found
  - Varying sampling density based on domain-knowledge
  - Computational improvements (collision checks, bootstrapping search, efficient data structures)

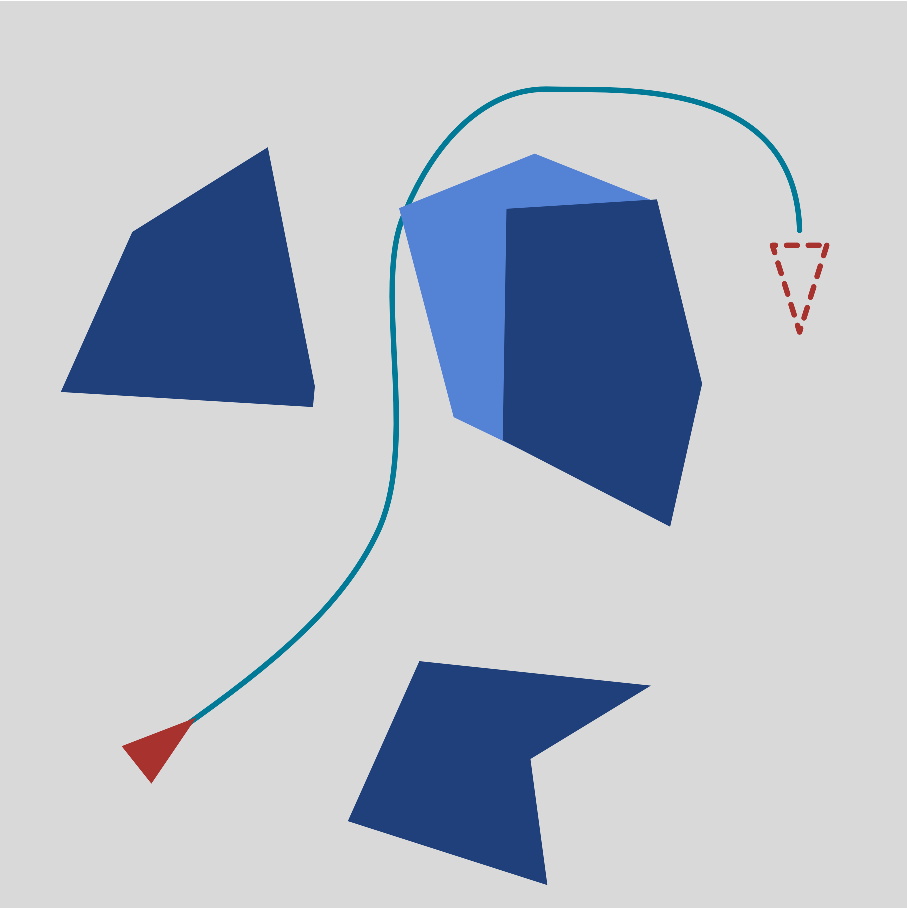
Summary of PRM* and RRT* results

Table 1. Summary of results. Time and space complexity are expressed as a function of the number of samples n, for a fixed environment.

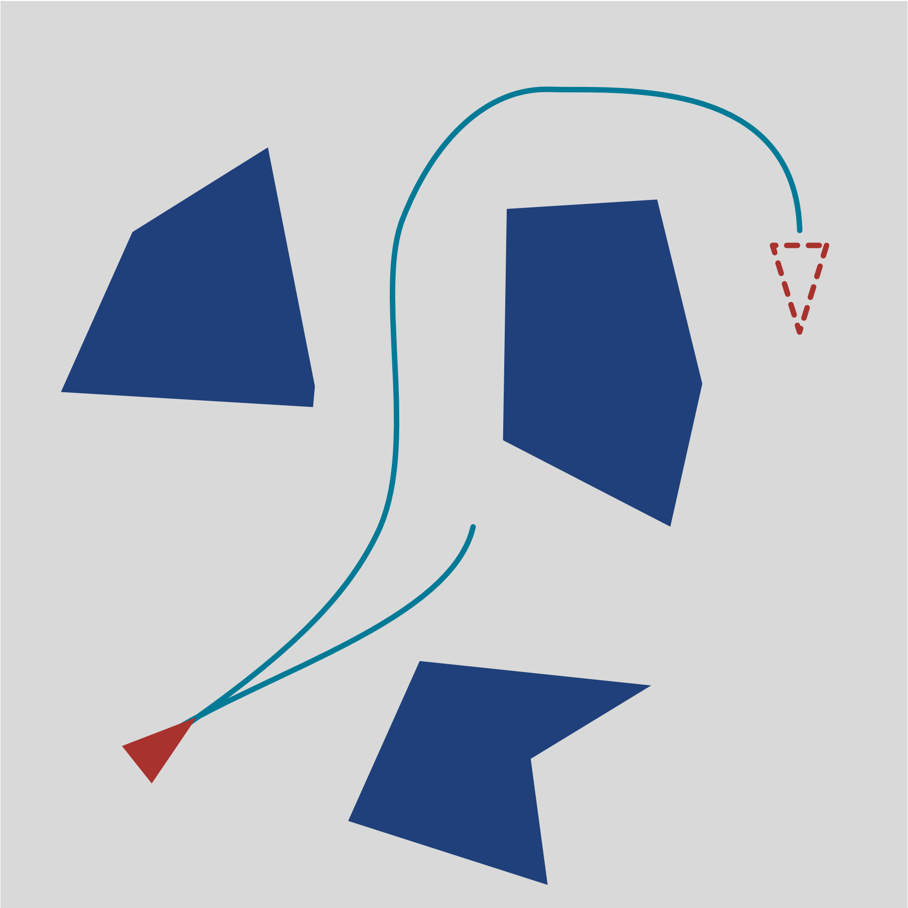
	Algorithm	Probabilistic completeness	Asymptotic optimality	Monotone convergence	Time complexity: Processing	Time complexity: Query	Space complexity
Existing algorithms	PRM	Yes	no	Yes	$O(n \log n)$	$O(n \log n)$	$O(n)$
	sPRM	Yes	Yes	Yes	$O(n^2)$	$O(n^2)$	$O(n^2)$
	k-sPRM	Conditional	No	No	$O(n \log n)$	$O(n \log n)$	$O(n)$
	RRT	Yes	No	Yes	$O(n \log n)$	$O(n)$	$O(n)$
Proposed algorithms	PRM*, k-PRM*	Yes	Yes	No	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	RRG, k-RRG	Yes	Yes	Yes	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	RRT*, k-RRT*	Yes	Yes	Yes	$O(n \log n)$	$O(n)$	$O(n)$

PLANNING UNDER UNCERTAINTY
(NOT IN EXAM)

- Environment uncertainty



- Motion uncertainty



Planning under uncertainty

- So far, we've discussed planning with known models, and then dealing with variation in environments and models at the execution stage using a lower-level controller or local planner
- However, in many cases we know something about the uncertainty in our models **at the planning stage** and we should take this into account when planning
- There are many approaches, but it almost always helps to understand one of the more fundamental representations, the **Markov Decision Process (MDP)**

Markov Property → Markov Process → Markov Decision Process

- Firstly, a system is **Markovian** if, for each state, we can fully describe the likelihood of following states from **only** the current state

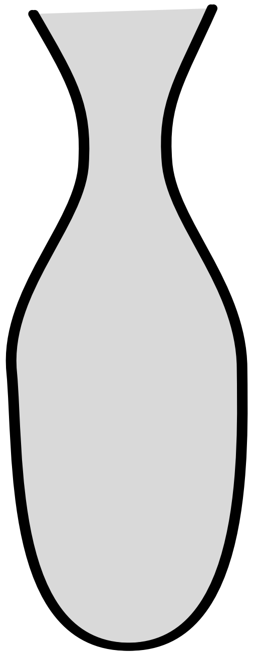
$$f(s) = Pr(s'|s)$$

- Basically, we can determine what will happen next without knowing the history up to that point

- Alternatively, what do we need in our state representation in order to define the likelihood of the next state

Markov Property

$P(B_3 = Red) = 1.0$



- Consider an urn containing two red balls and a blue ball
- We will draw balls from the urn sequentially **without** replacement. Our state will be the colour of the last draw
- If two balls have been drawn, and the second was red, what is the probability of the next draw being red? $P(B_3 = Red|B_2 = Red)$?
- Without knowing the first draw, the probability is 0.5
- If we **did** know the first draw, the probability would be 0 or 1

$$P(B_3 = Red \mid B_2 = Red) \neq P(B_3 = Red|B_2 = Red, B_1)$$

– knowledge of the second ball only is **non Markovian**

Markov Property

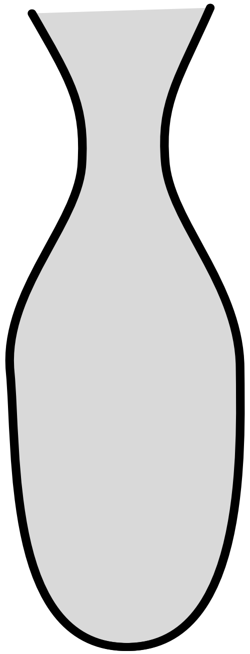
$P(B_3 = Red|B_2 = Blue) = 0.5$

- Now, consider that after each draw, we put the previous draw back in the urn (**with** replacement)

- Now, knowledge of the last draw **does not** change the probabilities of the next draw

$$P(B_3|B_2) = P(B_3|B_2, B_1)$$
$$\Rightarrow P(B_3|B_2) \perp B_1$$

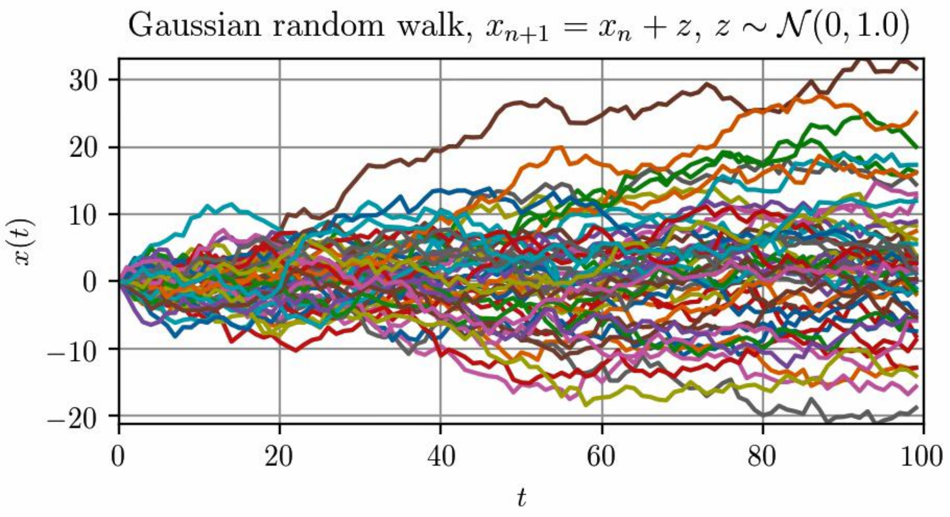
– knowledge of the current ball fully defines the likelihood of the next draw, so it **is Markovian**



Markov Process

- A **Markov Process** or **Markov Chain** is a 2-tuple representing a sequence of Markovian events

$$(S, \mathcal{P})$$



Where:

- S is a set of states
- $\mathcal{P}: S \rightarrow S$ is a state transition function ($\mathcal{P}(s) = s'$)

Consider a simple Gaussian random walk

- $S = \mathbb{R}$

$$x_{n+1} = x_n + z \qquad z \sim \mathcal{N}(0,1)$$

$$\mathcal{P}: \quad p(x_{n+1} | \, x_n) = \mathcal{N}(x_n, 1)$$

Markov Decision Process

- A **Markov Decision Process** adds decisions (actions) and rewards:

$$(S, A, \mathcal{P}, \mathcal{R})$$

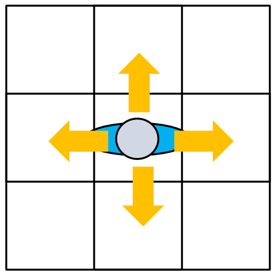
Where:

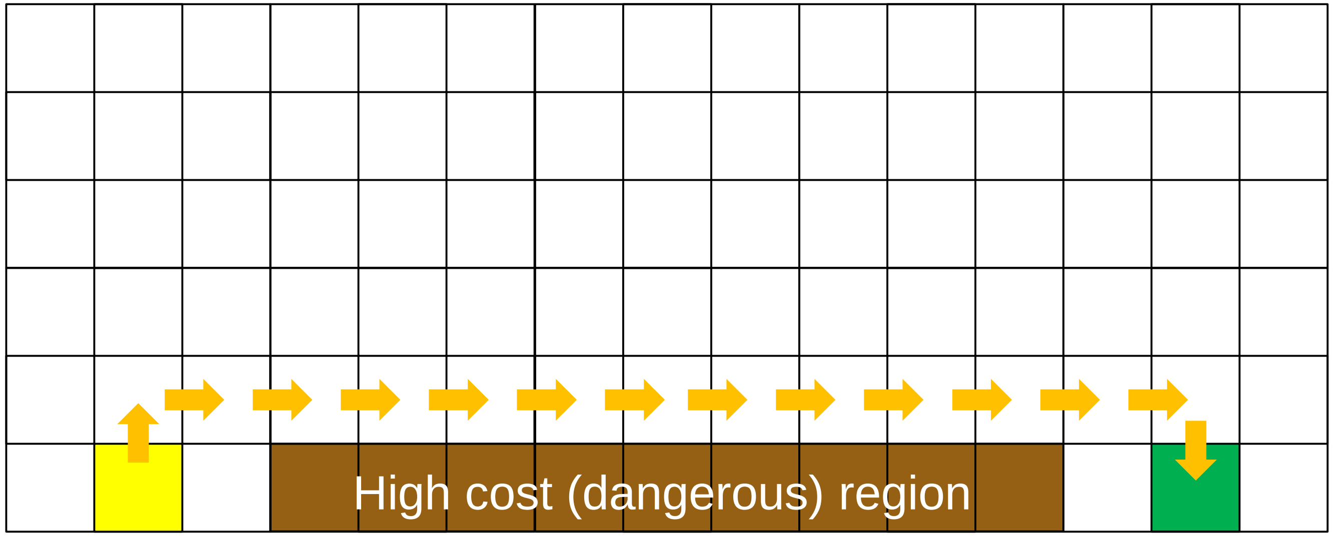
- A is a set of actions (defined $\forall\, s \in S$)
- $\mathcal{P}: S \times A \to S$ is the transition function (now with actions, $\Pr(s'|s, a)$)
- $\mathcal{R}: S \times S \times A \to R$ is a reward function ($\mathcal{R}(s', s, a)$ is the reward for being in state s, taking action a and transitioning to state s')

Markov Decision Process

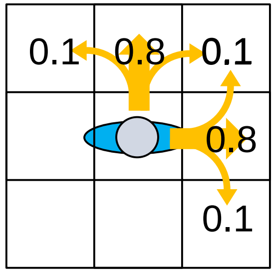
- We can use MDPs to generate plans that take probabilities into account (dynamic programming, reinforcement learning, etc.)

- First, consider a deterministic model, and traditional planning

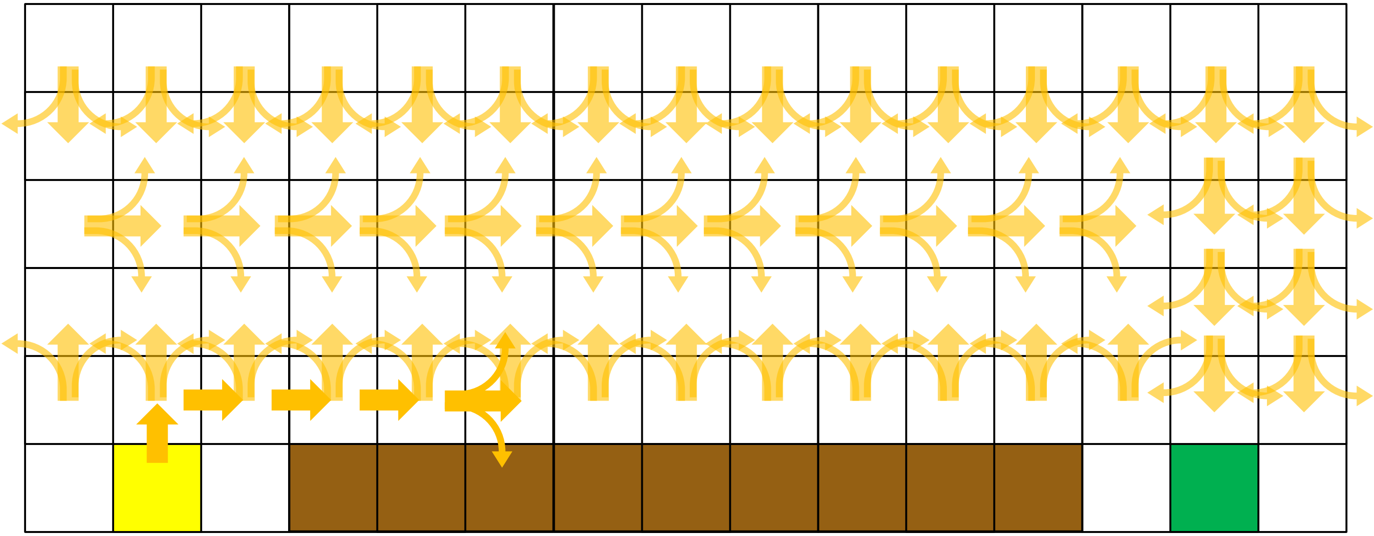




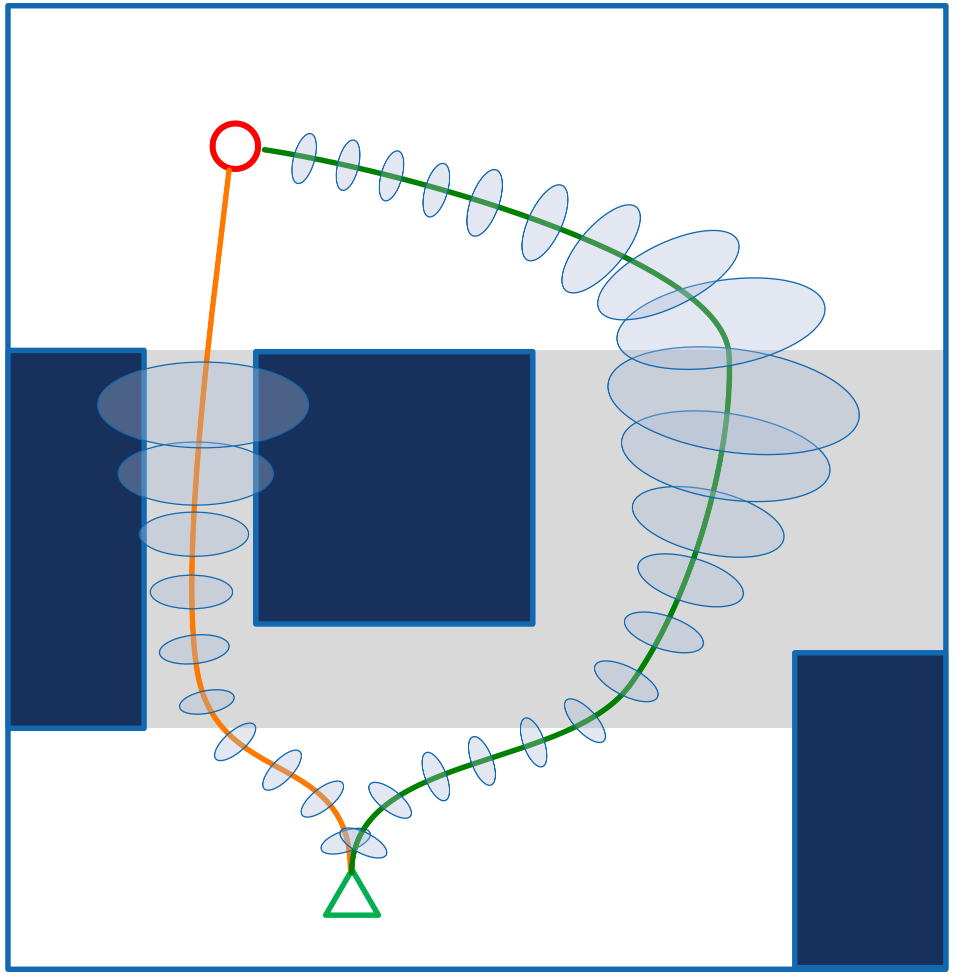
- After the pub… → probabilistic motion, $p(s'|s,a)$



- We would like to choose the correct **action** for each possible state, given a reward function ($r(s',s,a)$) – i.e. a **policy** rather than a **path**



Incorporating navigation and/or motion planning uncertainty

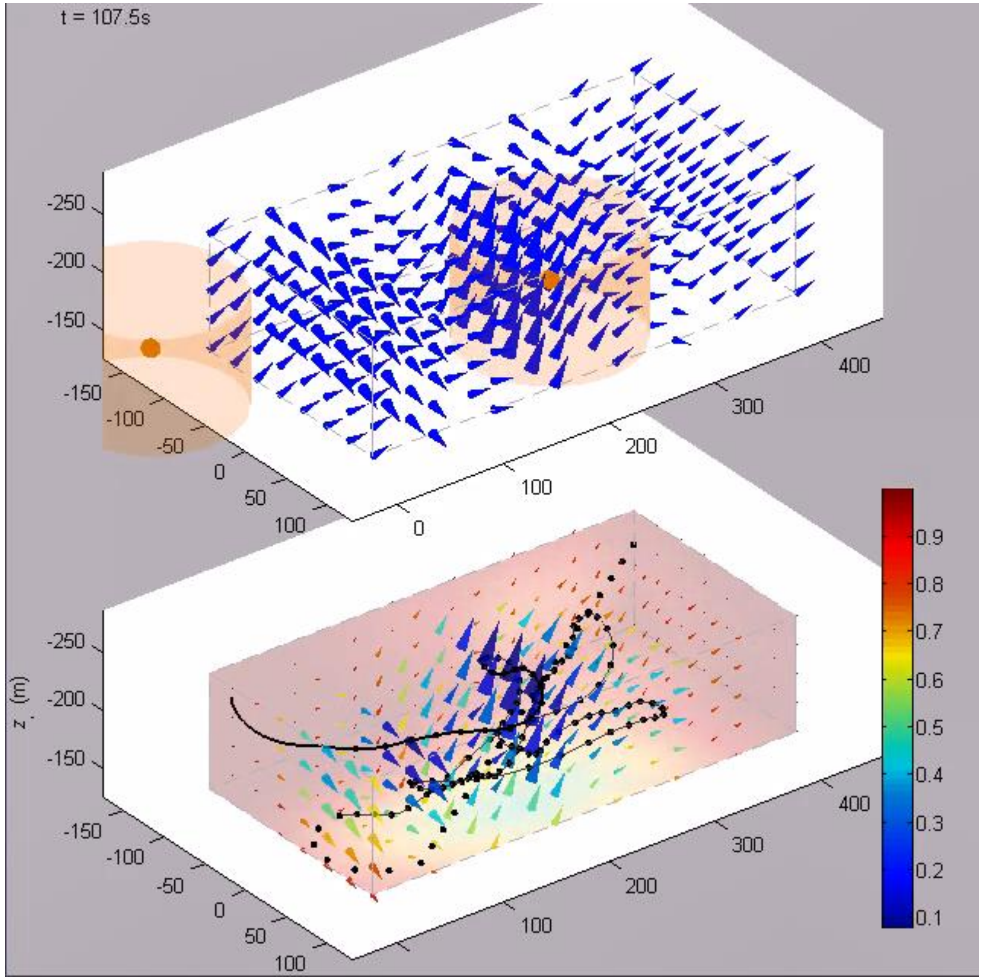


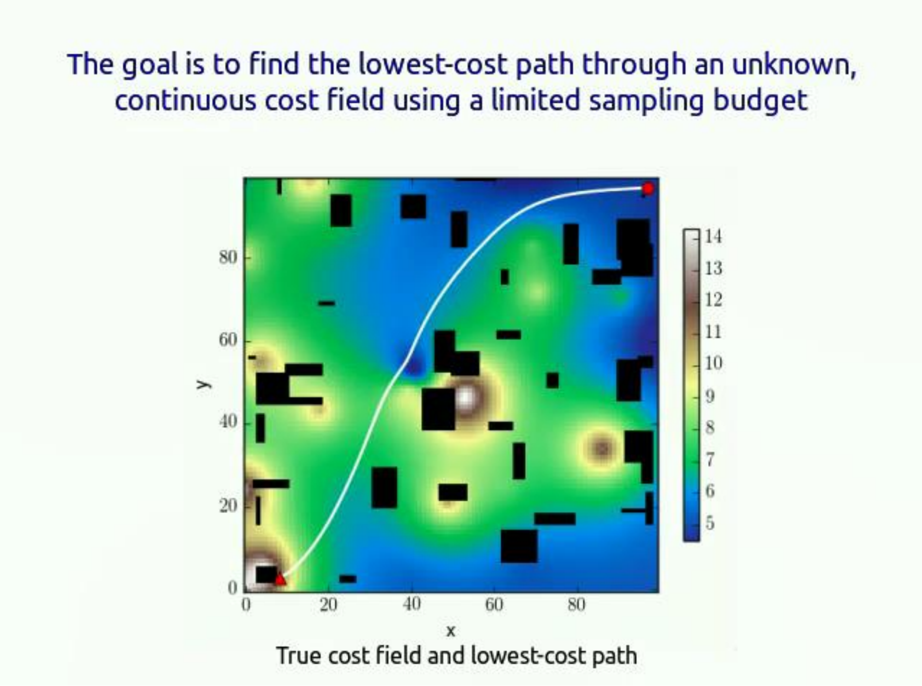
Bry, A. and Roy, N., 2011, May. Rapidly-exploring random belief trees for motion planning under uncertainty. In *2011 IEEE international conference on robotics and automation* (pp. 723-730).

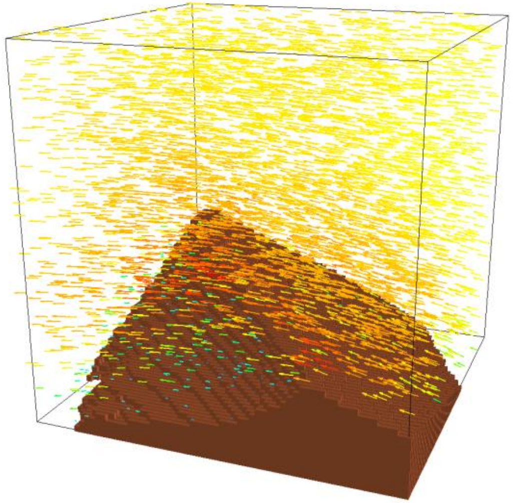
Uncertain Environments

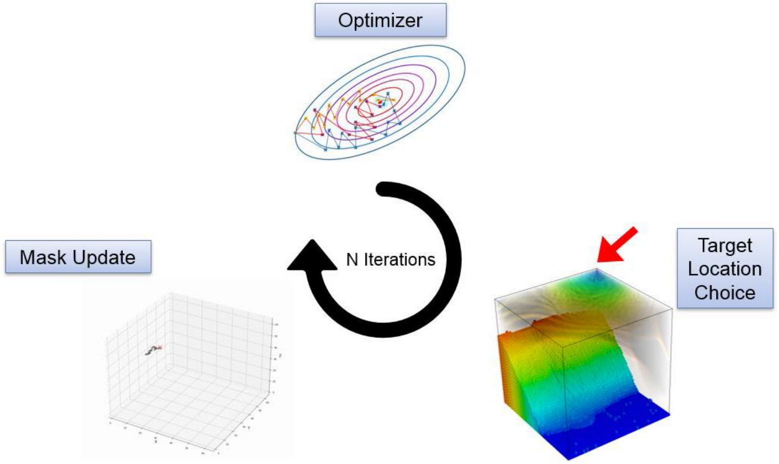
Pfeiffer, Mark, et al. "A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments." *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

Informative planning







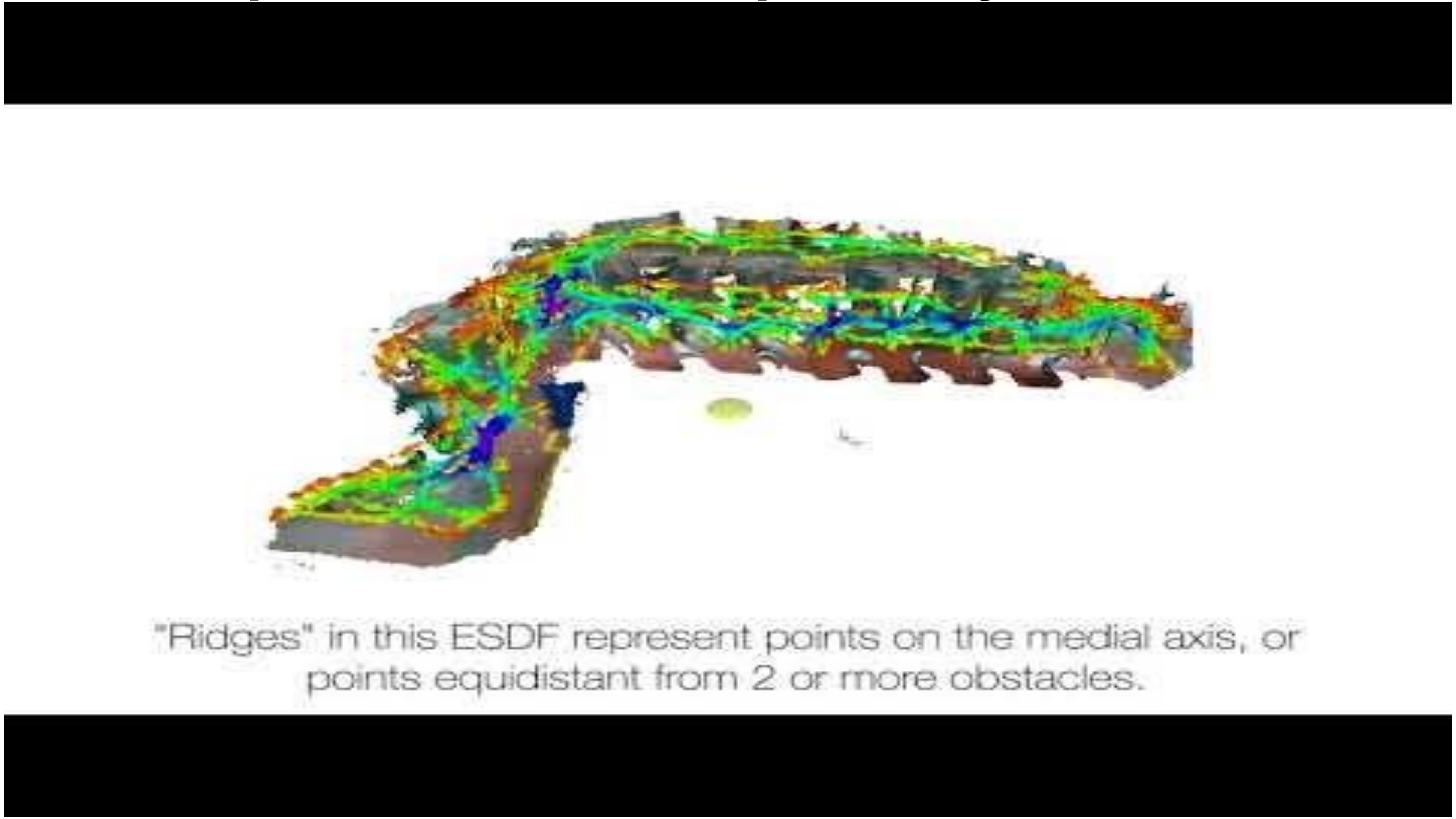


Florian Achermann and Stefan Parapid

Information-gathering planning

Schmid, Lukas, et al. "An Efficient Sampling-based Method for Online Informative Path Planning in Unknown Environments." *IEEE Robotics and Automation Letters* 5.2 (2020): 1500-1507.

Distance-field representations for planning



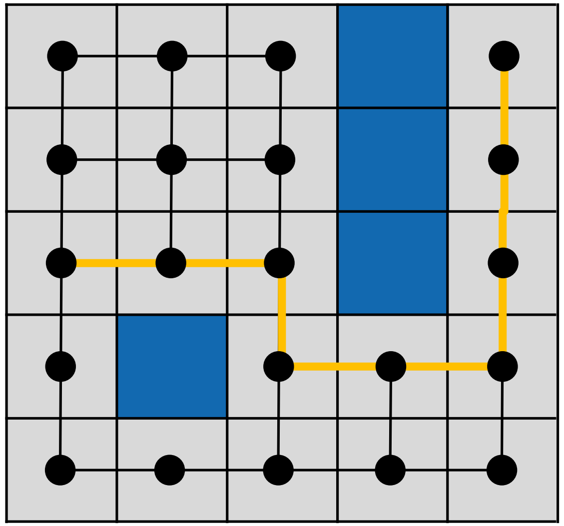
Oleynikova, Helen, et al. "Sparse 3d topological graphs for micro-aerial vehicle planning." *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

MOTION PLANNING: SUMMARY

Summary

- We introduced the notion of a Graph to model the environment and solve for the planning problem

- For this Grid-based problems we discussed about
  - **Breadth-first search** (BFS)
  - **Depth-first search** (DFS)
  - **Dijkstra**’s Algorithm (for weighted graphs)
  - **A*** (introduces a heuristic to go faster towards the goal)



Summary

- We also discussed the concept of configuration space
  - To represent the robot and environment in a common space
  - It allows us to think about the robot as point
  - We can then employ the same algorithms to plan in different robots: articulated, car-like, flying, etc.

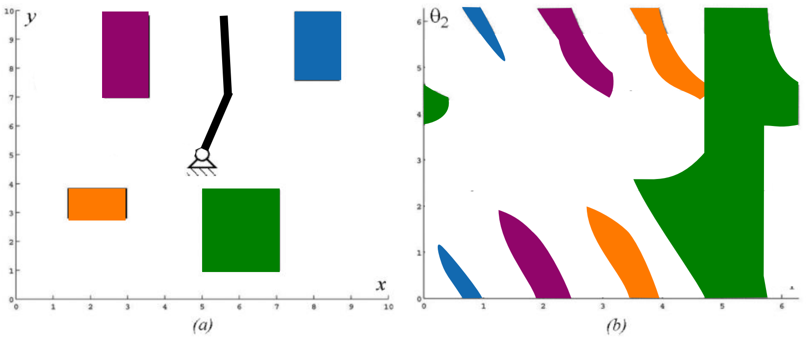


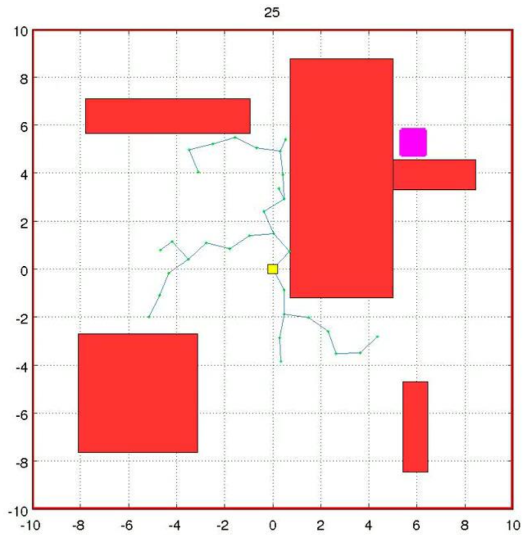
Figure 6.1
Physical space (a) and configuration space (b): (a) A two-link planar robot arm has to move from the configuration *start* to *end*. The motion is thereby constraint by the obstacles 1 to 4. (b) The corresponding configuration space shows the free space in joint coordinates (angle θ_1 and θ_2) and a path that achieves the goal.

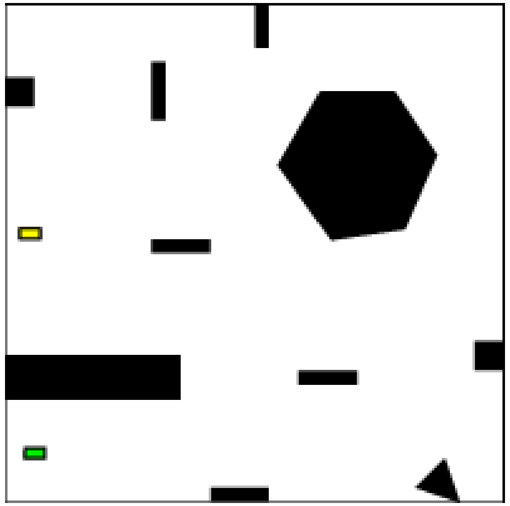
Summary

- Finally, we introduced the idea of sampling-based planning approaches
  - More efficient than grid-based
  - We don't need an explicit representation of the obstacles

- We saw two different approaches
  - Probabilistic roadmaps
  - Rapidly-exploring Random Trees

- Showed how to use RRT with non-holonomic systems

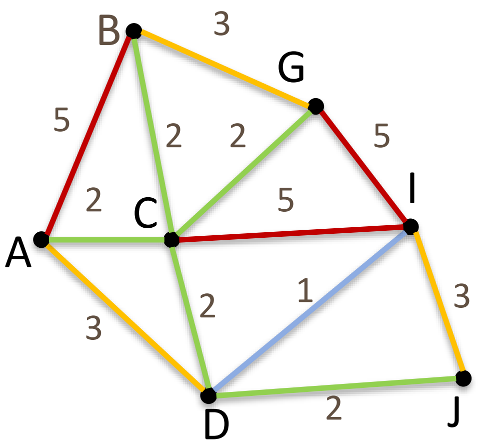




Exam

- What is in the exam (planning)?
  - Exam contains a mixture of short-form (T/F) and long-form questions
  - Multiple choice questions can cover main ideas in planning – mostly covered in textbook:
    - Hierarchies, planner structure
    - Properties and features of planners
      - Completeness and optimality
      - Heuristics
      - Applicability (why use this planner, would it be applicable to … ?)
      - etc…

Exam



- Long form questions:
  - Basic (hand-worked) implementation of major planning algorithms
  - Graph search
  - Potential fields
  - Collision avoidance

ACCEPTED

Node	Arrival cost	Best parent

FRONTIER

Node	Current lowest arrival cost	Best parent

- Not covered:
  - Implementation of sampling-based methods (but properties might be!)
  - Probabilistic planning (MDPs, explicit uncertainty reasoning)

Questions ?