# Nearest-neighbor search in robot motion planning

## CG Course, Lecture 10

Michal Kleinbort

Tel Aviv University, May 2020

# A robot

- A mechanical device, equipped with actuators and sensors, that is controlled by a computing system
- Operates in a real-world workspace, populated by physical objects
- Performs tasks by executing motions in the workspace
- An autonomous robot is required to plan its own motions automatically in order to achieve a given task

# Some examples

Robotic vacuum cleaners

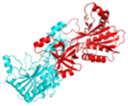
Self-driving cars

Robotic arms

Drones

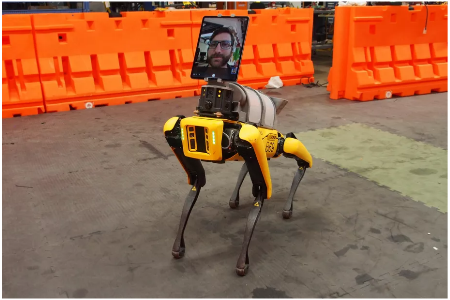
Robotic arms for medical use

Multi-robot settings



Proteins can be considered as robots that execute motion in order to fold

# Some examples


In the context of COVID-19

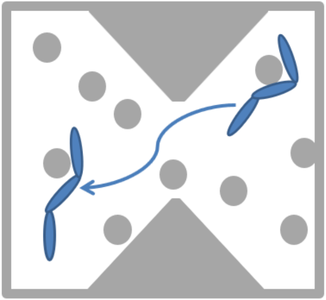
# The motion-planning problem

Given:

- A robot $R$
- A workspace $\mathcal{W}$ (with obstacles)
- Initial and final positions

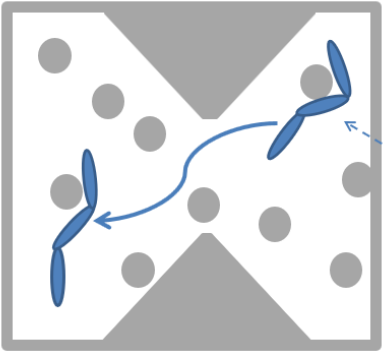
Goal:

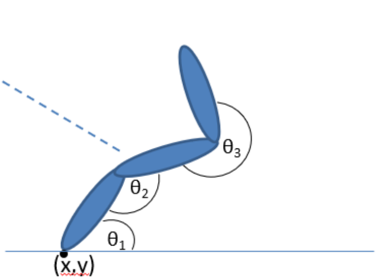
- Plan a collision-free continuous path for the robot from the initial position to the final position



# A configuration of the robot

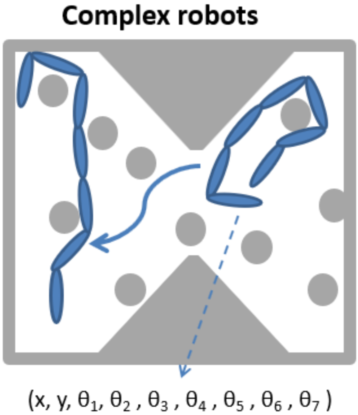
A configuration of the robot is represented by a set of parameters, e.g., $(x, y, \Theta_1, \Theta_2, \Theta_3)$

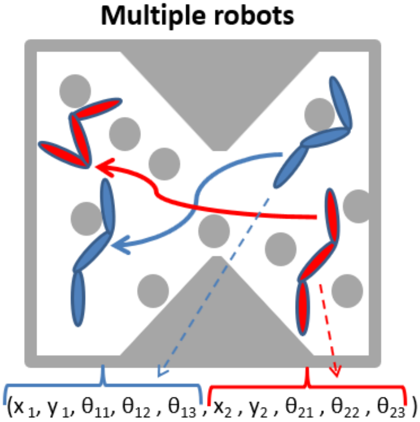




# The dimension

The dimension of the motion-planning problem is defined by the length of each configuration



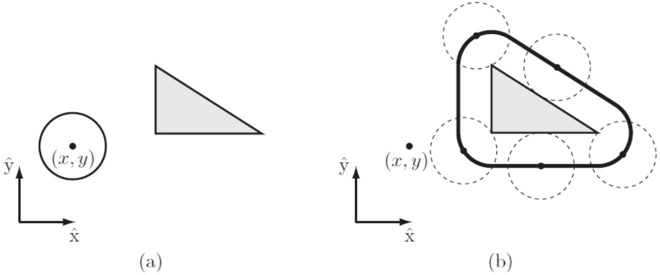



# The configuration space

The $d$-dimensional space $\mathcal{C}$ containing all possible configurations of the robot is called the configuration space (C-space).

A subset $\mathcal{F} \subset \mathcal{C}$ of all the collision-free configurations is called the free space.

The C-obstacles, defined as $\mathcal{C}_{\text{forb}} = \mathcal{C} \setminus \mathcal{F}$, are rarely represented exactly (may have a complex mathematical representation).




Figures from [Lynch and Park, 16]

# An alternative formulation of the MP problem

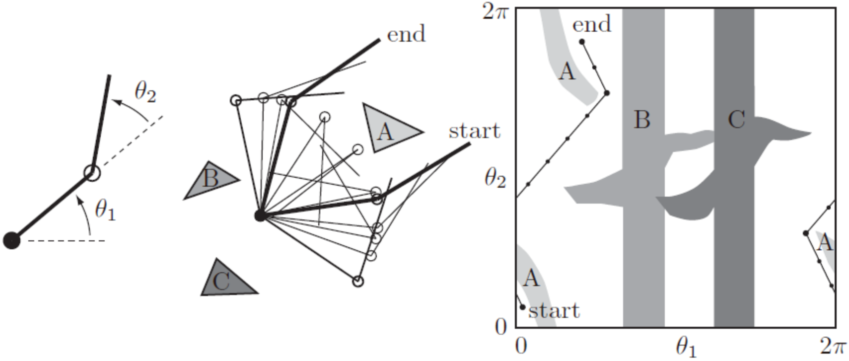
Given:

- A point robot
- A $d$-dimensional configuration space $\mathcal{C}$ (C-space)
- C-obstacles (often not explicitly given) $\mathcal{C}_{\text{forb}}$
- Free space $\mathcal{F} = \mathcal{C} \setminus \mathcal{C}_{\text{forb}}$
- Initial and final configurations

Goal:

- Plan a continuous path in the free space from the initial configuration to the final configuration

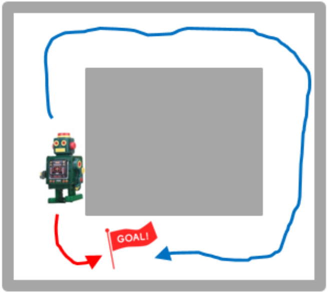
# An alternative formulation of the MP problem




Figures from [Lynch and Park, 16]

# Challenges

- High-dimensional problems are “hard” to solve
- Finding an optimal path is harder than finding a path
  - minimal path length
  - maximal distance from obstacles
  - smoothness



# Sampling-based methods for solving the problem

- Attempt to capture the structure of the C-space by constructing a graph (called a roadmap)
  - The nodes are collision-free configurations sampled at random
  - Two nearby nodes are connected by an edge if the path between them (usually a straight line) is collision-free
- Are often probabilistically complete
- Novel methods also ensure asymptotic optimality

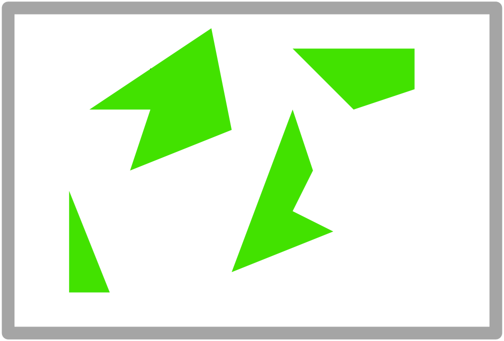
# Primitive operations in sampling-based methods

- Collision detection (CD)
  - Determines whether a configuration or a C-space path between two configurations is collision-free. The latter is termed local planning (LP)
  - Complexity usually depends on both the complexity of the workspace obstacles and the complexity of the robot
- Nearest-neighbor search (NN)
  - Returns the nearest neighbor (or neighbors) of a given configuration
  - Complexity depends on the number $n$ of nodes and the dimension $d$

The main practical computational bottleneck is typically considered to be CD (including LP)

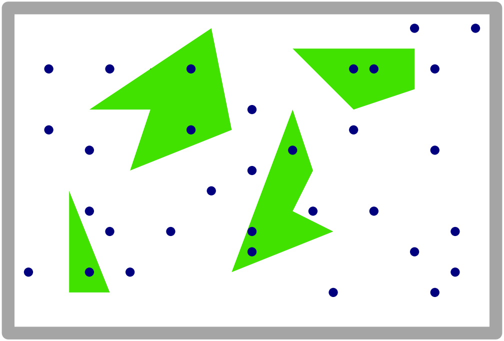
# An example: s-PRM* [Karaman and Frazzoli, 11]

The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



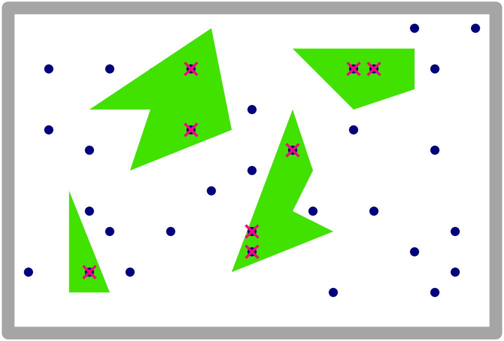
# An example: s-PRM* [Karaman and Frazzoli, 11]

The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



# An example: s-PRM* [Karaman and Frazzoli, 11]

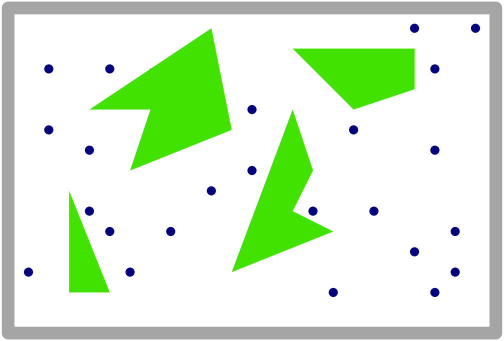
The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



Involves CD operation

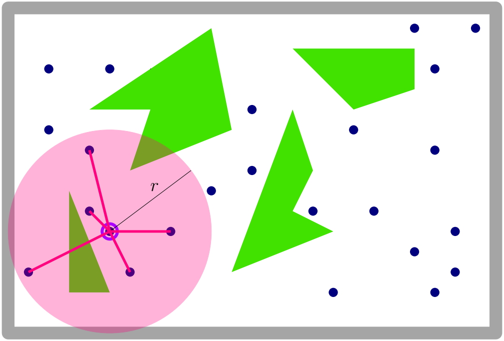
# An example: s-PRM* [Karaman and Frazzoli, 11]

The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



# An example: s-PRM* [Karaman and Frazzoli, 11]

The Probabilistic Roadmap Method (PRM) - Multi-query algorithm

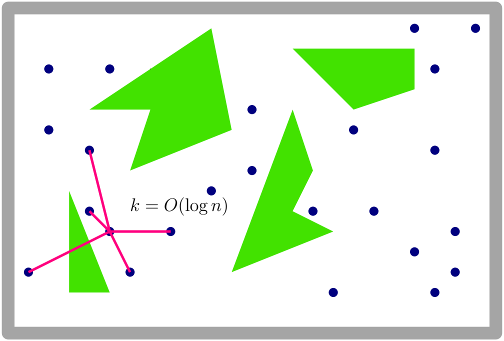



Involves NN operation ($r$-nearest neighbors or $k$-nearest neighbors)

$$r_{\mathrm{PRM}^*}(n) = 2\left[\left(1+\frac{1}{d}\right)\cdot\left(\frac{\mu(\mathcal{C}_{\mathrm{free}})}{\zeta_d}\right)\cdot\left(\frac{\log n}{n}\right)\right]^{1/d}$$

# An example: s-PRM* [Karaman and Frazzoli, 11]

The Probabilistic Roadmap Method (PRM) - Multi-query algorithm

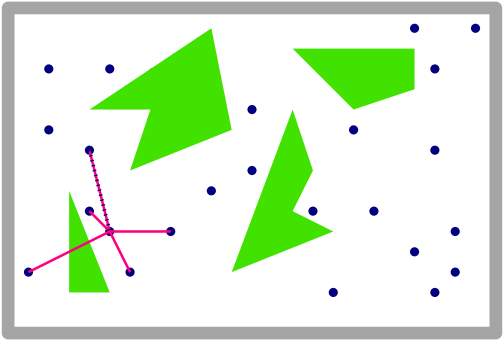



Involves NN operation ($r$-nearest neighbors or $k$-nearest neighbors)

$$k_{\mathrm{PRM}^*}(n) = 2e \log n$$

# An example: s-PRM* [Karaman and Frazzoli, 11]

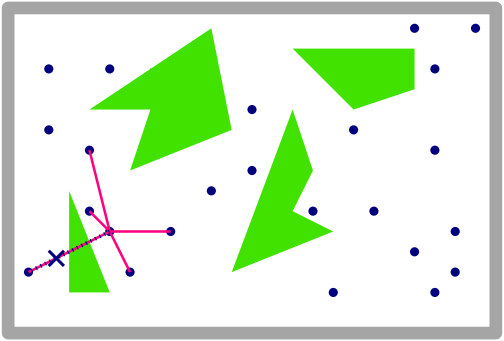
The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



Involves CD operation (as an LP sub-procedure)

# An example: s-PRM* [Karaman and Frazzoli, 11]

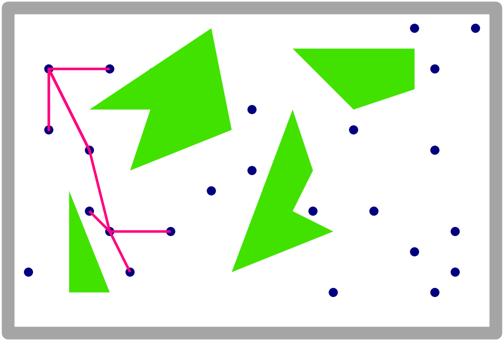
## The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



Involves CD operation (as an LP sub-procedure)

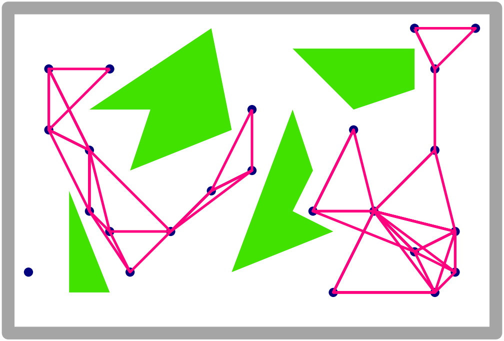
# An example: s-PRM* [Karaman and Frazzoli, 11]

The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



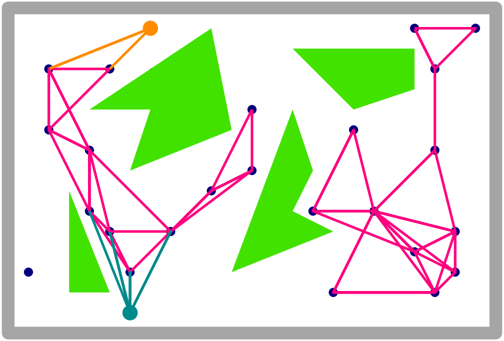
# An example: s-PRM* [Karaman and Frazzoli, 11]

The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



# An example: s-PRM* [Karaman and Frazzoli, 11]

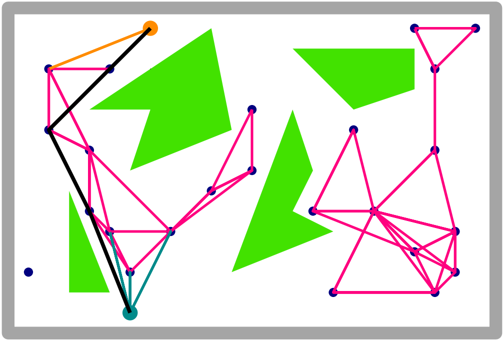
The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



Query: Given start and goal configurations, find the shortest path between the two configurations

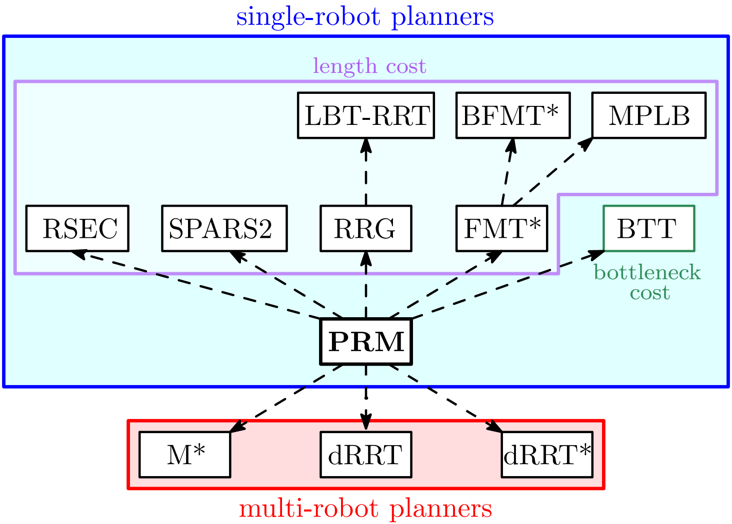
# An example: s-PRM* [Karaman and Frazzoli, 11]

## The Probabilistic Roadmap Method (PRM) - Multi-query algorithm



Query: Given start and goal configurations, find the shortest path between the two configurations

# Sampling-based planners

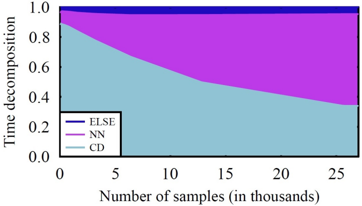


# Our results [K., Salzman and Halperin '15, '16]

- We formally prove that the complexity of NN search dominates the asymptotic running time of several AO algorithms

# Our results [K., Salzman and Halperin '15, '16]

- We formally prove that the complexity of NN search dominates the asymptotic running time of several AO algorithms
- We characterize settings in which the role of NN is far from negligible and show experimentally that NN may dominate CD after finite time




- We use an efficient, specifically-tailored NN data structure in such settings to reduce the overall time of motion-planning algorithms

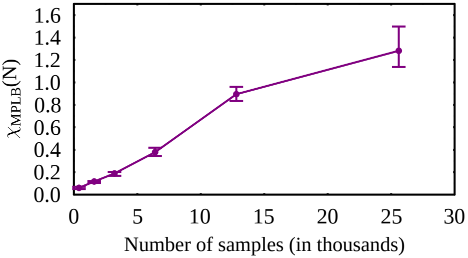
# NN-sensitive settings

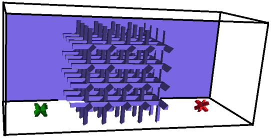
Can be of the following types:

- Algorithms: Planners that algorithmically shift some of the computational weight from CD to NN
- Scenarios: Scenarios in which the computational cost of certain planners is mostly due to NN search
- Parameters: Parameters' values for which the computational cost of certain planners is mostly due to NN search

# NN-sensitive algorithms

We measure the ratio $\chi_{\text{ALG}}(N)$ between the overall time spent on NN and the time spent on CD, after $N$ configurations were sampled, when algorithm ALG is used



Using NN-sensitive algorithms (e.g., Lazy-PRM* [Hauser '15], MPLB [Salzman and Halperin '15] etc.), the ratio $\chi_{\text{ALG}}(N)$ significantly increases as a function of $N$, obtaining values greater than 1, namely, NN takes more time than CD

# NN-sensitive scenarios

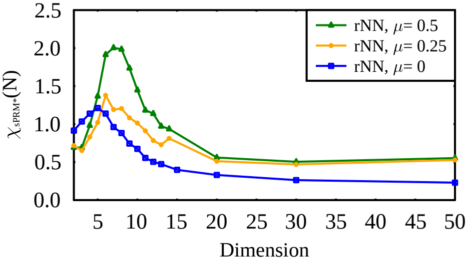
- Let $\mathcal{S} = (\mathcal{W}, \mathcal{R})$ denote a scenario, where:
  - $\mathcal{W}$ denotes the workspace
  - $\mathcal{R}$ denotes the robot system, which is a set of $\ell$ single constant-description complexity robots operating simultaneously
- Let $d$ denote the dimension of $\mathcal{S}$, $d = \Theta(\ell)$
- Let $m$ denote the complexity of the workspace obstacles
- Robot-robot collisions should be taken into account as well

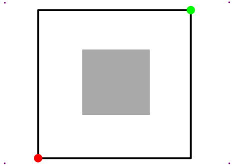
# NN-sensitive scenarios - The effect of $d$

## sPRM* example:

- When $d$ is gradually increased, the ratio $\chi_{\text{sPRM*}}(N)$:
  - Shows an initial increase
  - Then shows a possible decrease
  - Finally, shows a very slow increase or perhaps even tends to some constant value

R-NN



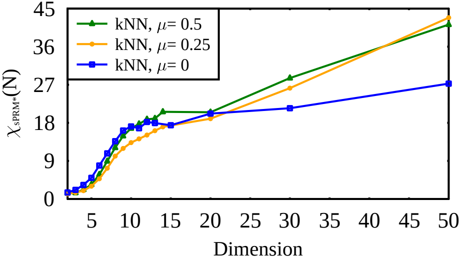
$d$-dimensional workspace
$\mu$ is the obstacle volume
start end

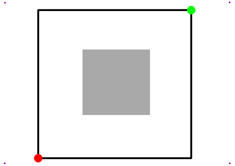
# NN-sensitive scenarios - The effect of $d$

## sPRM* example:

- When $d$ is gradually increased, the ratio $\chi_{\text{sPRM*}}(N)$:
  - Shows an initial increase
  - Then shows a possible decrease
  - Finally, shows a very slow increase or perhaps even tends to some constant value

K-NN: shows a different trend

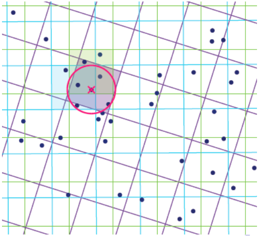


$d$-dimensional workspace
$\mu$ is the obstacle volume
start end

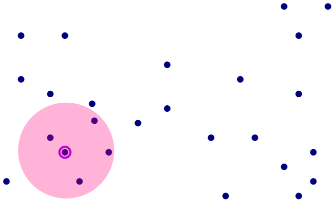
# Adapting all-pairs $r$NN algorithms for sampling-based motion planning

- In several planning algorithms "all-pairs" $r$-NN are used with a predefined value $r(n) = O((\frac{\log n}{n})^{1/d})$
- Randomly transformed grids (RTG) [Aiger et al., 14] is a novel method for approximate all-pairs $r$-NN
- We implemented RTG and used it for certain sampling-based algorithms
- We obtain significant speedups improving: the construction time, the time to find an initial solution, and the time to converge to high-quality solutions



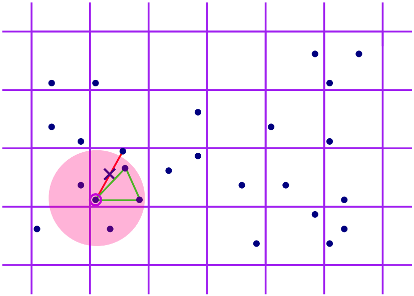
# Randomly transformed grids (RTG) [Aiger et al., 14]

- Given a set $P$ of $n$ points in $\mathbb{R}^d$ and a radius $r$, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability
- The algorithm:
  1. Place a $d$-dimensional axis-parallel grid of cell size $c$ with a random shift (chosen uniformly)
  2. The points of $P$ are partitioned into the grid cells
  3. The distance $||p - q||_2$ between every pair of points $p, q \in P$ within the same cell is computed, and the pair is reported if $||p - q||_2 \leq r$
  4. Repeat steps (1)-(3) $m$ times, producing $m$ distinct grids



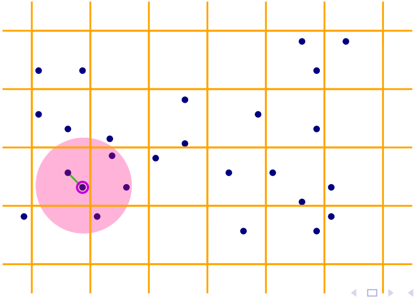
# Randomly transformed grids (RTG) [Aiger et al., 14]

- Given a set $P$ of $n$ points in $\mathbb{R}^d$ and a radius $r$, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability
- The algorithm:
  1. Place a $d$-dimensional axis-parallel grid of cell size $c$ with a random shift (chosen uniformly)
  2. The points of $P$ are partitioned into the grid cells
  3. The distance $||p - q||_2$ between every pair of points $p, q \in P$ within the same cell is computed, and the pair is reported if $||p - q||_2 \leq r$
  4. Repeat steps (1)-(3) $m$ times, producing $m$ distinct grids



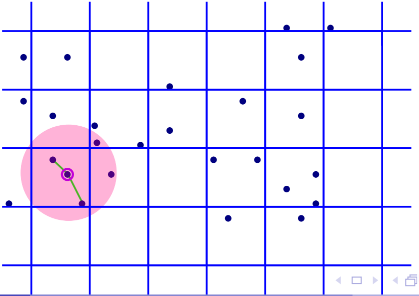
# Randomly transformed grids (RTG) [Aiger et al., 14]

- Given a set $P$ of $n$ points in $\mathbb{R}^d$ and a radius $r$, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability
- The algorithm:
  1. Place a $d$-dimensional axis-parallel grid of cell size $c$ with a random shift (chosen uniformly)
  2. The points of $P$ are partitioned into the grid cells
  3. The distance $||p - q||_2$ between every pair of points $p, q \in P$ within the same cell is computed, and the pair is reported if $||p - q||_2 \leq r$
  4. Repeat steps (1)-(3) $m$ times, producing $m$ distinct grids



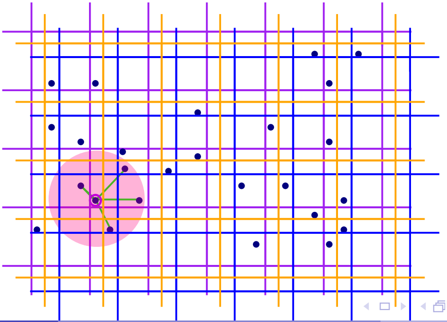
# Randomly transformed grids (RTG) [Aiger et al., 14]

- Given a set $P$ of $n$ points in $\mathbb{R}^d$ and a radius $r$, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability
- The algorithm:
  1. Place a $d$-dimensional axis-parallel grid of cell size $c$ with a random shift (chosen uniformly)
  2. The points of $P$ are partitioned into the grid cells
  3. The distance $||p - q||_2$ between every pair of points $p, q \in P$ within the same cell is computed, and the pair is reported if $||p - q||_2 \leq r$
  4. Repeat steps (1)-(3) $m$ times, producing $m$ distinct grids



# Randomly transformed grids (RTG) [Aiger et al., 14]

- Given a set $P$ of $n$ points in $\mathbb{R}^d$ and a radius $r$, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability
- The algorithm:
  1. Place a $d$-dimensional axis-parallel grid of cell size $c$ with a random shift (chosen uniformly)
  2. The points of $P$ are partitioned into the grid cells
  3. The distance $||p - q||_2$ between every pair of points $p, q \in P$ within the same cell is computed, and the pair is reported if $||p - q||_2 \leq r$
  4. Repeat steps (1)-(3) $m$ times, producing $m$ distinct grids

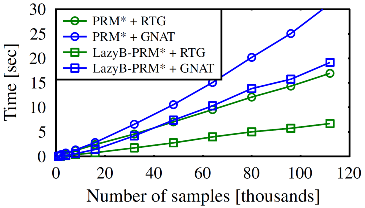


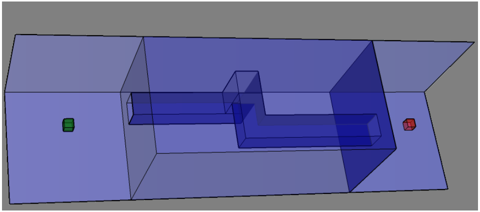
# Factors determining the efficiency

1. The number $m$ of grids:
   - Increasing $m$ increases the probability of capturing true near neighbors in the same cell
   - Increasing $m$ increases the overall running time
2. The cell size $c$:
   - Large $c$ increases the probability of capturing true near neighbors in the same cell
   - Small $c$ causes the ratio between the overall number of inspected pairs and the output size to be close to 1 (reduces the portion of irrelevant checks)

$c$ should be greater than (but very close to) $r$ when a random shift is used

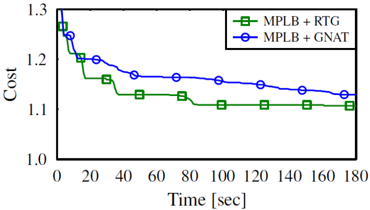
# Experimental results

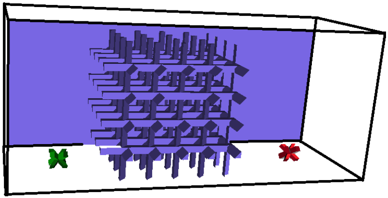


Improved construction time (3D Euclidean C-space)

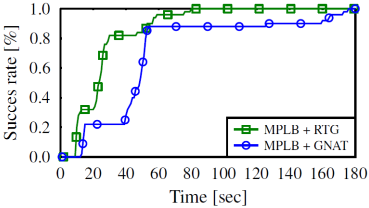
# Experimental results

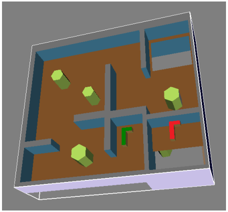


Faster convergence to high-quality solutions (6D non-Euclidean C-space)

# Experimental results



Shorter times for finding an initial solution (6D Euclidean C-space)

In the above scenario for $n = 50K$ we managed to always find a solution. The number of reported NN pairs was $\sim 400K$.

# The End