Control of Mobile Robots

Motion planning

Prof. Luca Bascetta (luca.bascetta@polimi.it)

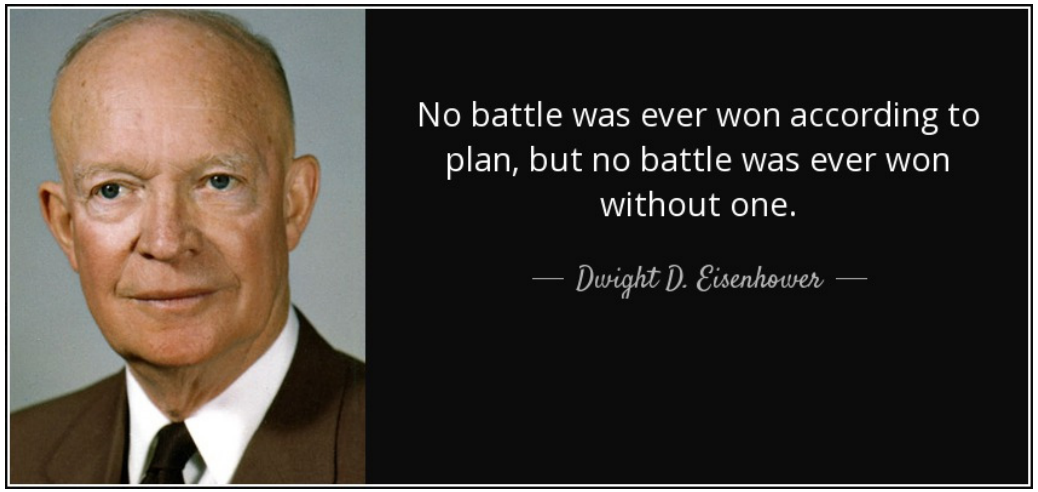
Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria

Organization and motivations

Navigation is the ability that transforms a mobile robot into an autonomous mobile robot.
Two of the most important navigation functionalities are planning and control.
We start introducing the motion planning problem and planning techniques.

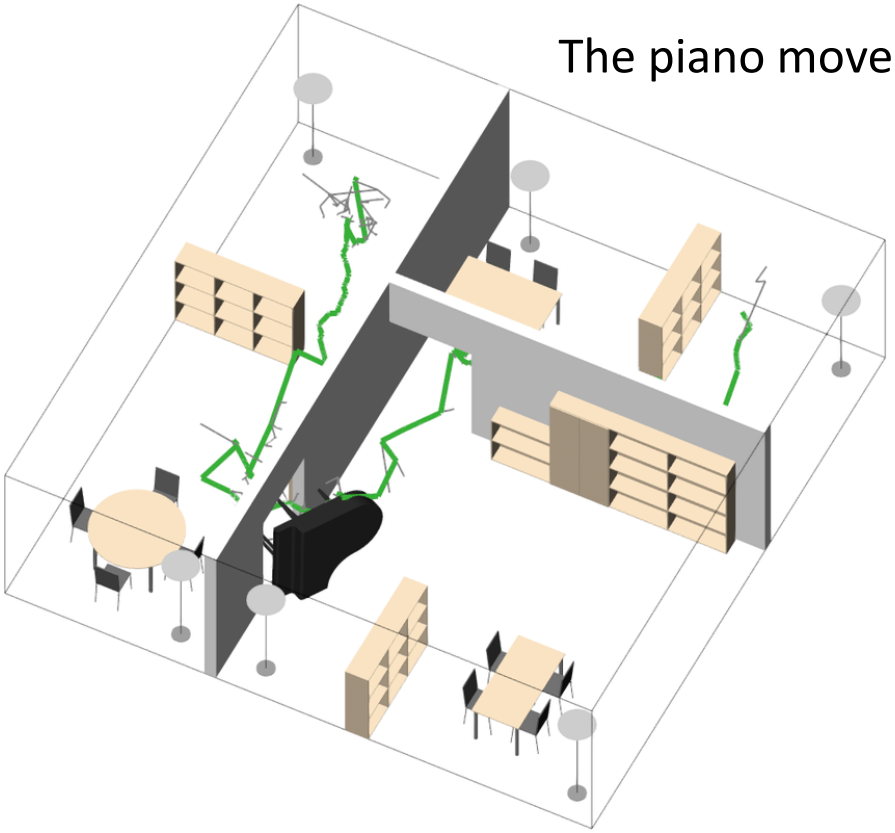
The main topics on motion planning are

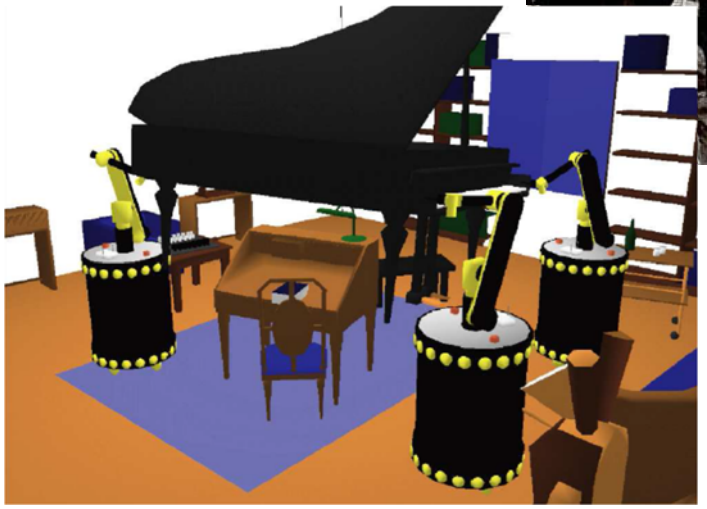
- introducing the motion planning problem
- flatness-based planning
- planning vs. control
- an overview of planning algorithms
- sampling-based planning
- optimal sampling-based planning
- introduction to kinodynamic optimal planning



What is planning?

The piano mover's problem

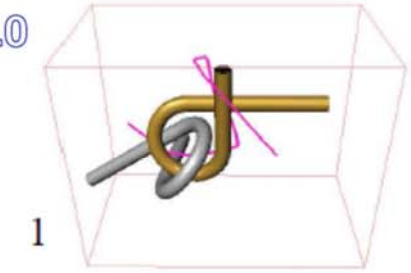


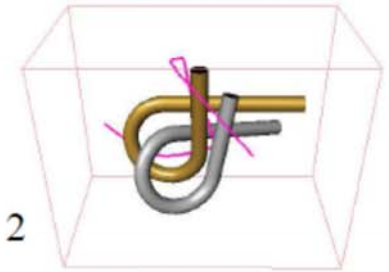


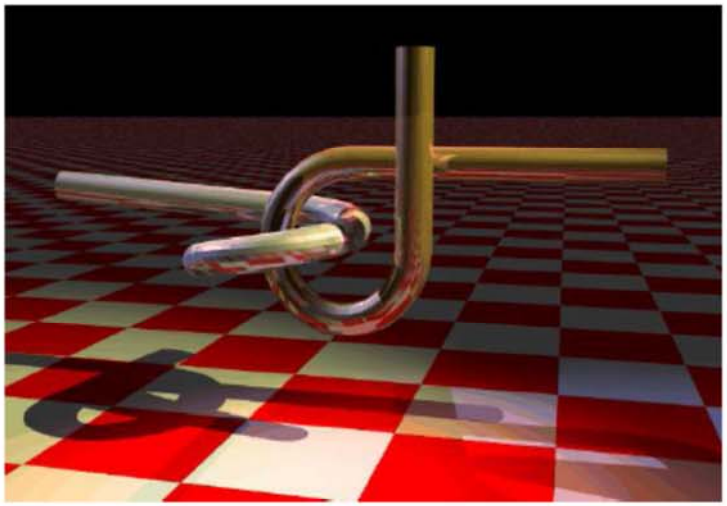
What is planning?

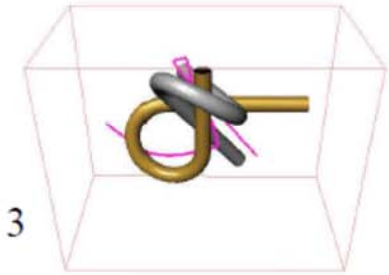
Intelligent and Mobile Robotics Group
http://imr.felk.cvut.cz

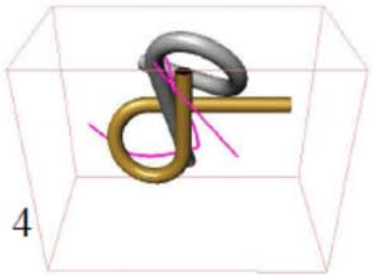
Alpha Puzzle 1.0

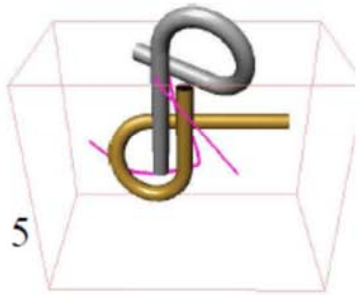




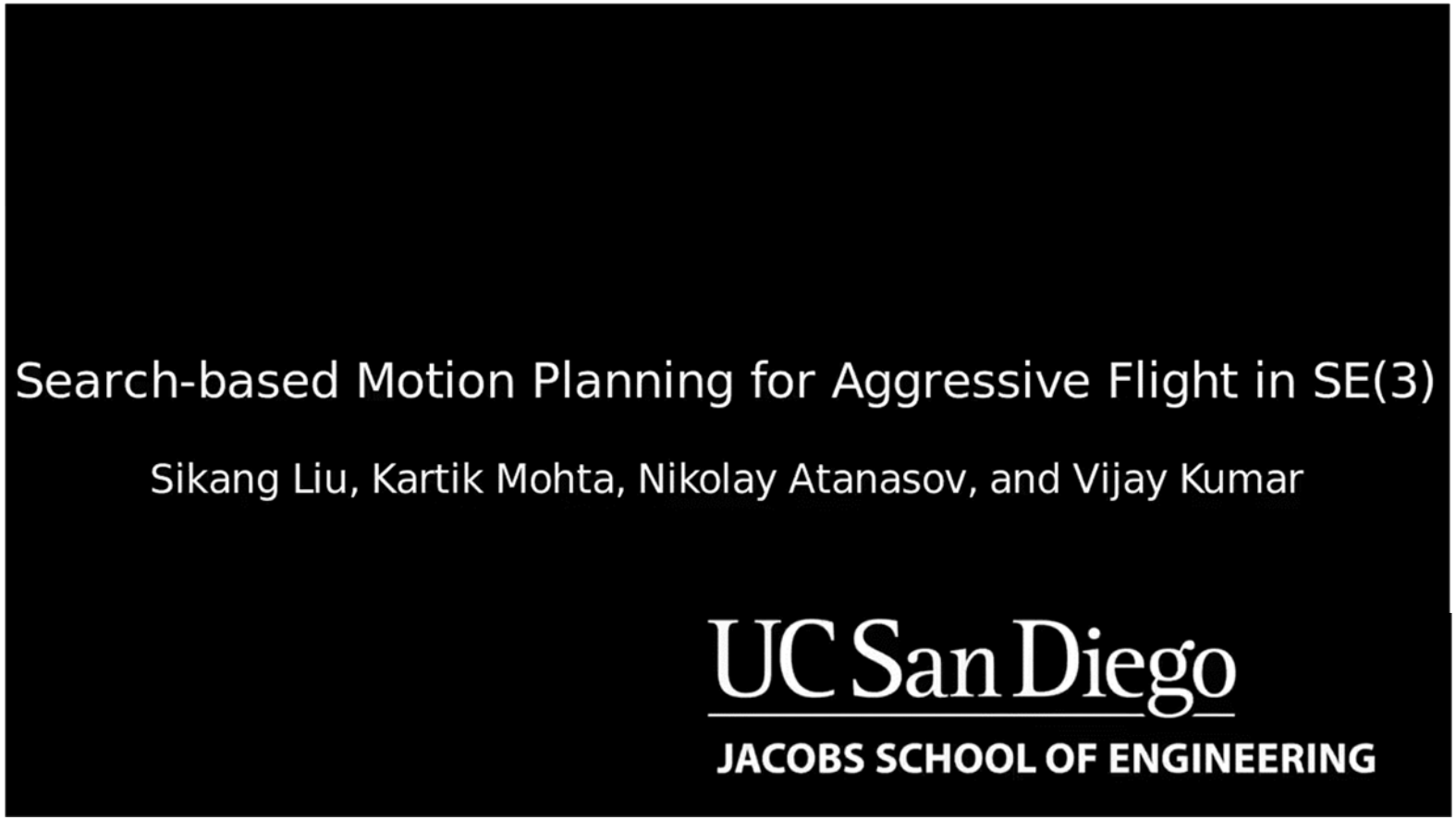








What is planning?



What is planning?

The motion planning problem: path and timing law

We will concentrate on the problem of planning a trajectory for a mobile robot.

Trajectory planning can be broken down into:

- finding a *geometric path*
- determining a *timing law*

If the mobile robot is subject to nonholonomic constraints, path planning becomes more difficult

Consider a mobile robot that moves from an initial configuration $\boldsymbol{q}(t_i) = \boldsymbol{q}_i$ to a final configuration $\boldsymbol{q}(t_f) = \boldsymbol{q}_f$ in the absence of obstacles.

The *trajectory planning problem* is the problem of finding $\boldsymbol{q}(t)$ for $t \in [t_i, t_f]$ subject to the initial and final configuration constraints, and can be broken down into:

- finding a geometric path $\boldsymbol{q}(s)$ with $\mathrm{d}\boldsymbol{q}(s)/\mathrm{d}s \neq \boldsymbol{0}$
- determining a timing law $s = s(t)$ with $s \in [s(t_i), s(t_f)]$, $\dot{s}(t) \geq 0$ for $t \in [t_i, t_f]$

A possible choice for s is the *natural coordinate* (arc length along the path), in this case

- $s_i = s(t_i) = 0$
- $s_f = s(t_f) = L$ (the path length)

Sometimes it is useful to introduce a *normalized natural coordinate* $s \in [0,1]$.

Thanks to the space/time separation we have

$$\dot{\mathbf{q}} = \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}s}\dot{s} = \mathbf{q}'\dot{s}$$

- $\boldsymbol{q}'$ represents the change in the robot configuration with respect to space and is a vector directed as the tangent to the path in the configuration space
- $\dot{s}$ is a scalar quantity representing the modulus of the velocity

The space/time separation can be applied to nonholonomic constraints

$$A^T(\mathbf{q})\,\dot{\mathbf{q}} = A^T(\mathbf{q})\,\mathbf{q}'\dot{s} = \mathbf{0}$$

and if $\dot{s}(t) > 0$ for $t \in [t_i, t_f]$

$$A^T(\mathbf{q})\,\mathbf{q}' = \mathbf{0}$$

This is a condition on the *geometric path admissibility* induced by kinematic constraints.

Geometrically admissible paths are thus the solutions of the nonlinear system

$$\mathbf{q}' = G(\mathbf{q})\,\tilde{\mathbf{u}}$$

where $\tilde{\boldsymbol{u}}$ is a vector of geometric inputs

$$\mathbf{u}(t) = \tilde{\mathbf{u}}(s)\,\dot{s}(t)$$

We consider the example of the unicycle model.

The pure rolling constraint can be written as

$$\begin{bmatrix}\sin(\theta) & -\cos(\theta) & 0\end{bmatrix}\mathbf{q}' = x'\sin(\theta) - y'\cos(\theta) = 0$$

and geometrical admissible paths are solutions of the system

$$\begin{aligned} x' &= \tilde{v}\cos(\theta) \\ y' &= \tilde{v}\sin(\theta) \\ \theta' &= \tilde{\omega} \end{aligned}$$

where

$$\begin{aligned} v(t) &= \tilde{v}(s)\dot{s}(t) \\ \omega(t) &= \tilde{\omega}(s)\dot{s}(t) \end{aligned}$$

For example we can select

$$\tilde{v}(s) = 1 \qquad \tilde{\omega}(s) = \frac{\omega(t)}{v(t)}$$

in this way

$$\begin{aligned} x' &= \cos(\theta) \\ y' &= \sin(\theta) \\ \theta' &= \tilde{\omega} \end{aligned}$$

where the new input $\tilde{\omega}$ is the curvature.

We start introducing properties and tools that allow to transform a nonlinear dynamic system in a simpler form.

The first property we introduce is called *differential flatness*.

A nonlinear system

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

is *differentially flat* if there exists a set of outputs $\boldsymbol{y}$, called *flat outputs*, such that the state $\boldsymbol{x}$ and the control inputs $\boldsymbol{u}$ can be expressed algebraically as a function of $\boldsymbol{y}$ and its time derivatives up to a certain order

$$\mathbf{x} = \mathbf{x}\left(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, \ldots, \mathbf{y}^{(r)}\right)$$

$$\mathbf{u} = \mathbf{u}\left(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, \ldots, \mathbf{y}^{(r)}\right)$$

Differentially flat systems: the unicycle example

A unicycle kinematic model

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \omega$$

is a differentially flat system, when flat outputs $z_1 = x$ and $z_2 = y$ are selected.

The state vector of the kinematic model can be expressed as a function of the flat outputs and their first derivatives

$$x = z_1 \qquad y = z_2 \qquad \theta = \begin{cases} \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 > 0 \\ \pi + \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 < 0 \\ \frac{\pi}{2}\,\mathrm{sign}\,(\dot{z}_2) & \dot{z}_1 = 0 \end{cases}$$

Differentially flat systems: the unicycle example

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \omega$$

The input vector of the kinematic model can be expressed as a function of the flat outputs and their first and second order derivatives.

Using the first two equations of the model we can derive a relation for the linear velocity (assuming $v > 0$)

$$\sqrt{\dot{z}_1^2 + \dot{z}_2^2} = \sqrt{\dot{x}^2 + \dot{y}^2} = \sqrt{v^2\cos^2(\theta) + v^2\sin^2(\theta)} = v$$

From the third equation of the model and the flat relation on θ we derive a relation for the angular velocity

$$\omega = \dot{\theta} = \frac{1}{1 + \left(\frac{\dot{z}_2}{\dot{z}_1}\right)^2} \frac{\dot{z}_1\ddot{z}_2 - \dot{z}_2\ddot{z}_1}{\dot{z}_1^2} = \frac{\dot{z}_1\ddot{z}_2 - \dot{z}_2\ddot{z}_1}{\dot{z}_1^2 + \dot{z}_2^2}$$

Differentially flat systems: the unicycle example

Summarizing, the mappings obtained by the flat transformation are

$$x = z_1 \qquad y = z_2 \qquad \theta = \begin{cases} \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 > 0 \\ \pi + \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 < 0 \\ \frac{\pi}{2}\operatorname{sign}(\dot{z}_2) & \dot{z}_1 = 0 \end{cases}$$

and

$$v = \sqrt{\dot{z}_1^2 + \dot{z}_2^2} \qquad \omega = \frac{\dot{z}_1 \ddot{z}_2 - \dot{z}_2 \ddot{z}_1}{\dot{z}_1^2 + \dot{z}_2^2}$$

We must observe that this mapping is singular when

$$\dot{z}_1^2 + \dot{z}_2^2 = 0$$

In this case the linear velocity is zero, and θ and ω are not well-defined.

Differentially flat systems: the bicycle example

A bicycle kinematic model

$$\begin{aligned}\dot{x} &= v\cos(\theta)\\ \dot{y} &= v\sin(\theta)\\ \dot{\theta} &= v\tan(\phi)/\ell\\ \dot{\phi} &= \omega\end{aligned}$$

is a differentially flat system, when flat outputs $z_1 = x$ and $z_2 = y$ are selected.

The state vector of the kinematic model can be expressed as a function of the flat outputs and their first derivatives, as for the unicycle model

$$x = z_1 \qquad y = z_2 \qquad \theta = \begin{cases} \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 > 0 \\ \pi + \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 < 0 \\ \frac{\pi}{2}\,\mathrm{sign}\,(\dot{z}_2) & \dot{z}_1 = 0 \end{cases}$$

Differentially flat systems: the bicycle example

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = v\tan(\phi)/\ell$$
$$\dot{\phi} = \omega$$

From the first two equations we get also

$$\tan(\theta) = \frac{\dot{y}}{\dot{x}} \qquad \Rightarrow \qquad 1+\tan^2(\theta) = \frac{\dot{x}^2+\dot{y}^2}{\dot{x}^2}$$

and differentiating the first relation with respect to time

$$\left(1+\tan^2(\theta)\right)\dot{\theta} = \frac{\ddot{y}\dot{x}-\dot{y}\ddot{x}}{\dot{x}^2} \qquad \Rightarrow \qquad \dot{\theta} = \frac{\ddot{y}\dot{x}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}$$

From the last equation we now obtain

$$\tan(\phi) = \frac{\dot{\theta}\ell}{v} = \frac{\ddot{y}\dot{x}-\dot{y}\ddot{x}}{(\dot{x}^2+\dot{y}^2)^{3/2}}\ell \qquad \Rightarrow \qquad \phi = \arctan\left(\frac{\ddot{y}\dot{x}-\dot{y}\ddot{x}}{(\dot{x}^2+\dot{y}^2)^{3/2}}\ell\right)$$

Differentially flat systems: the bicycle example

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = v\tan(\phi)/\ell$$
$$\dot{\phi} = \omega$$

As for the input vector, the linear velocity can be obtained from the first two equations, as in the unicycle case (again assuming $v > 0$)

$$\sqrt{\dot{z}_1^2 + \dot{z}_2^2} = \sqrt{\dot{x}^2 + \dot{y}^2} = \sqrt{v^2\cos^2(\theta) + v^2\sin^2(\theta)} = v$$

The second input can be instead derived from the last equation

$$\omega = \dot{\phi} = \frac{\mathrm{d}}{\mathrm{d}t}\left\{\arctan\left(\frac{\ddot{y}\dot{x} - \dot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}}\ell\right)\right\} = \frac{\mathrm{d}}{\mathrm{d}t}\left\{\arctan\left(\frac{\ddot{z}_2\dot{z}_1 - \dot{z}_2\ddot{z}_1}{(\dot{z}_1^2 + \dot{z}_2^2)^{3/2}}\ell\right)\right\}$$

Once the states and inputs have been expressed in terms of the flat outputs, given an output trajectory $\boldsymbol{y}(t)$, the associated trajectory of the state $\boldsymbol{x}$ and of the control inputs $\boldsymbol{u}$ are uniquely determined.

In a differential flat system the state $\boldsymbol{x}$ and the control inputs $\boldsymbol{u}$ can be expressed algebraically as a function of the flat outputs $\boldsymbol{y}$ and their time derivatives up to a certain order.

For a unicycle robot, the kinematic model is a differentially flat system, when flat outputs $z_1 = x$ and $z_2 = y$ are selected.

Consider the following planning problem:

"*find a trajectory* $q(t) = [x(t) \quad y(t)]^T$ *to move a unicycle robot in an obstacle free environment, from an initial configuration* $q_i = [x_i \quad y_i \quad \theta_i \quad v_i]^T$ *at* $t_i = 0$ *to a final configuration* $q_f = [x_f \quad y_f \quad \theta_f \quad v_f]^T$ at $t_f = \bar{t}_f$"

How can we exploit flatness to solve this problem?

We assume that the two flat outputs can be expressed as polynomials with respect to the time variable

$$z_1(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$
$$z_2(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3$$

We have selected the orders of the polynomials in such a way that we have 8 coefficients, that can be determined applying the 8 initial and final constraints

$$z_1(0) = a_0 = x_i \qquad \dot{z}_1(0) = a_1 = v_i \cos(\theta_i)$$
$$z_2(0) = b_0 = y_i \qquad \dot{z}_2(0) = b_1 = v_i \sin(\theta_i)$$
$$z_1(\bar{t}_f) = a_0 + a_1 \bar{t}_f + a_2 \bar{t}_f^2 + a_3 \bar{t}_f^3 = x_f \qquad \dot{z}_1(\bar{t}_f) = a_1 + 2a_2 \bar{t}_f + 3a_3 \bar{t}_f^2 = v_f \cos(\theta_f)$$
$$z_2(\bar{t}_f) = b_0 + b_1 \bar{t}_f + b_2 \bar{t}_f^2 + b_3 \bar{t}_f^3 = y_f \qquad \dot{z}_2(\bar{t}_f) = b_1 + 2b_2 \bar{t}_f + 3b_3 \bar{t}_f^2 = v_f \sin(\theta_f)$$

From the first four equations we can directly compute the values of a_0, a_1, b_0, b_1, the other coefficients can be computed from the following linear system

$$\underbrace{\begin{bmatrix} \bar{t}_f^2 & \bar{t}_f^3 & 0 & 0 \\ 0 & 0 & \bar{t}_f^2 & \bar{t}_f^3 \\ 2\bar{t}_f & 3\bar{t}_f^2 & 0 & 0 \\ 0 & 0 & 2\bar{t}_f & 3\bar{t}_f^2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_2 \\ a_3 \\ b_2 \\ b_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} x_f - a_0 - a_1\bar{t}_f \\ y_f - b_0 - b_1\bar{t}_f \\ v_f \cos(\theta_f) - a_1 \\ v_f \sin(\theta_f) - b_1 \end{bmatrix}}_{\mathbf{b}}$$

As A is structurally not singular for $\bar{t}_f \neq 0$ ($\det(A) = -\bar{t}_f^8$), we can compute the other coefficients as $\boldsymbol{x} = A^{-1}\boldsymbol{b}$.

In conclusion, we have a closed-form solution for the planning problem.

In case the heading θ or the control variables v and ω are required, we can apply the relation derived by the flatness model

$$\theta = \begin{cases} \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 > 0 \\ \pi + \arctan\left(\frac{\dot{z}_2}{\dot{z}_1}\right) & \dot{z}_1 < 0 \\ \frac{\pi}{2}\operatorname{sign}(\dot{z}_2) & \dot{z}_1 = 0 \end{cases}$$

$$\sqrt{\dot{z}_1^2 + \dot{z}_2^2} = \sqrt{\dot{x}^2 + \dot{y}^2} = \sqrt{v^2\cos^2(\theta) + v^2\sin^2(\theta)} = v$$

$$\omega = \dot{\theta} = \frac{1}{1 + \left(\frac{\dot{z}_2}{\dot{z}_1}\right)^2} \frac{\dot{z}_1\ddot{z}_2 - \dot{z}_2\ddot{z}_1}{\dot{z}_1^2} = \frac{\dot{z}_1\ddot{z}_2 - \dot{z}_2\ddot{z}_1}{\dot{z}_1^2 + \dot{z}_2^2}$$

The procedure can be slightly modified in order to introduce the optimization of a cost function.

Let's consider, for example, a planning problem with actuation cost, where the function to be minimized is

$$J(v,\omega) = \int_0^{T_f} \begin{bmatrix} v \\ \omega \end{bmatrix}^T R \begin{bmatrix} v \\ \omega \end{bmatrix} \mathrm{d}t$$

To include the minimization of this cost function we should increase the number of coefficients in the time polynomials describing the flat outputs, so that we have more parameters than constraints (initial and final conditions)

$$z_1(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4$$

$$z_2(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3$$

In this way we can determine $a_0 \dots a_3$ and $b_0 \dots b_3$ from the initial and final conditions, using the previous relations, and a_4 and T_f enforcing the minimization of the cost function.

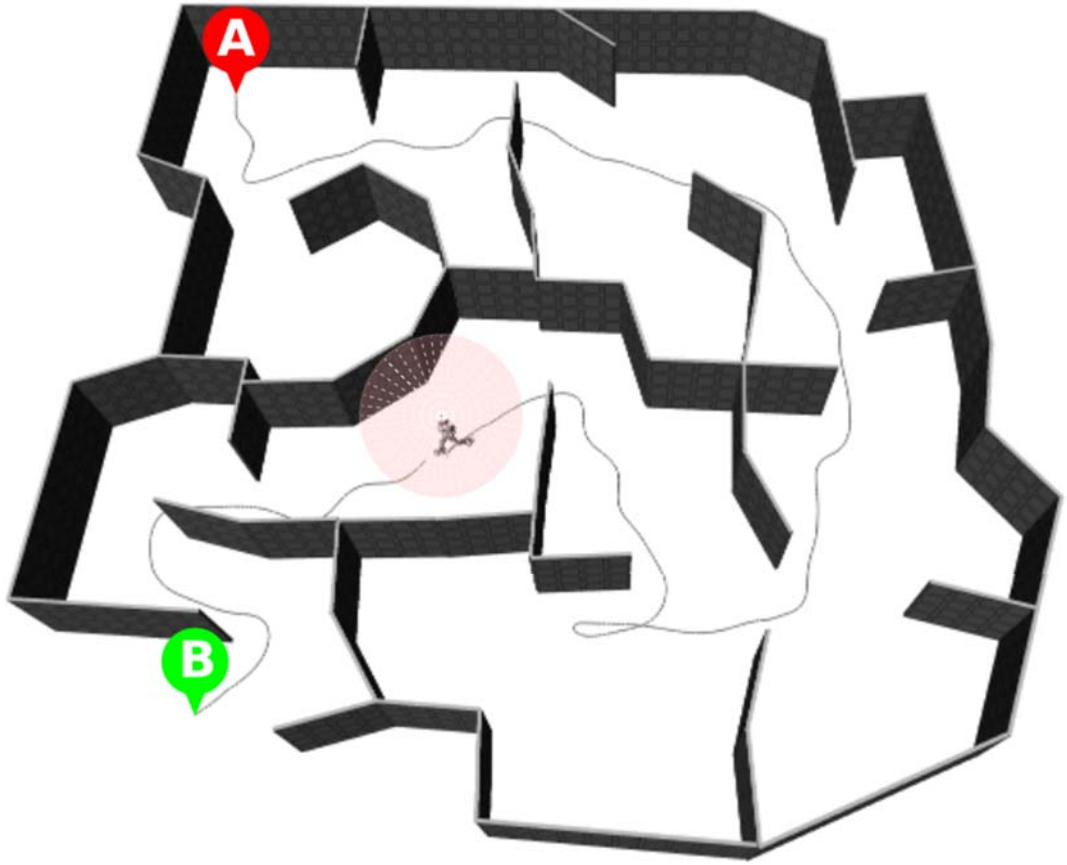
Other properties of dynamical systems can be exploited to set up similar planning procedures.

We will now introduce more general and efficient planning techniques, that allow to consider an environment possibly characterized by obstacles, but we would like first to discuss the relation between planning and control.

Planning and control: two sides of the same coin

- The planner (*global planner*) knows the map of the environment
- It has a global knowledge of the whole environment

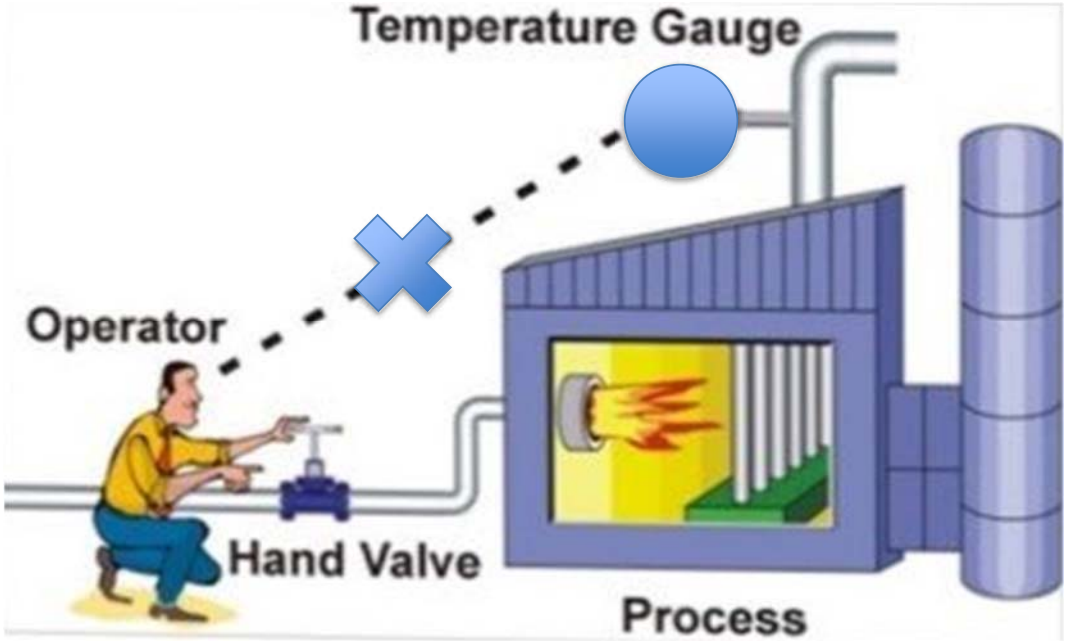
- The controller (*local planner*) knows only the environment around the vehicle, within the sensor visibility range
- It has a local knowledge of the environment, but perceives unknown and moving obstacles within the sensor visibility range



Planning and control: two sides of the same coin

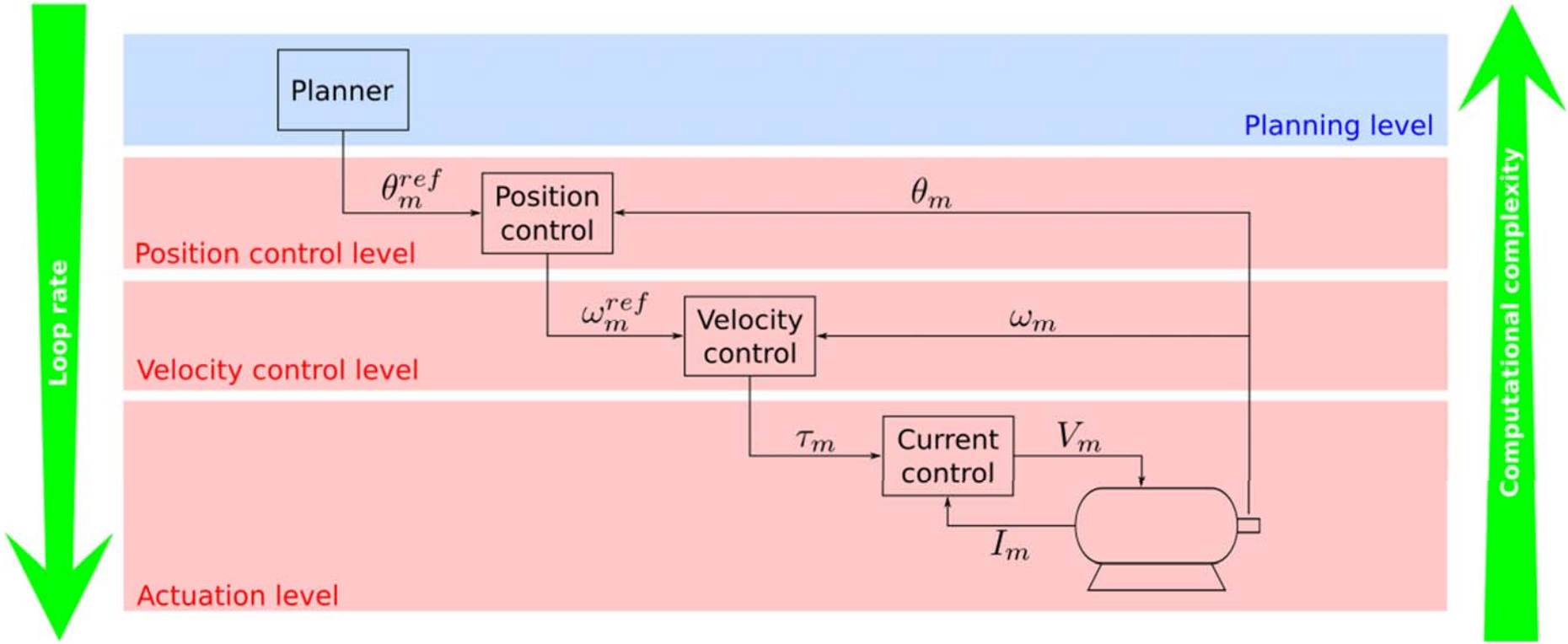
Can we use planning alone to compute the control actions?

Planning is equivalent to open-loop control...



...you cannot expect to have a high performance without measurements!

Can we use planning alone to compute the control actions?

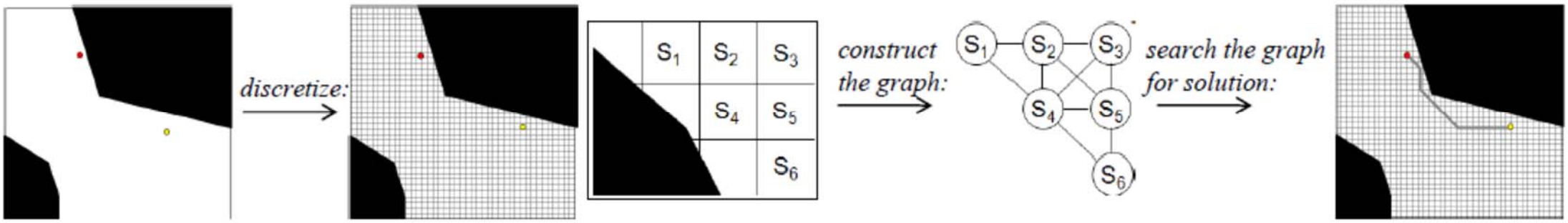


Exploiting frequency separation is the key to set up a system with high performance!

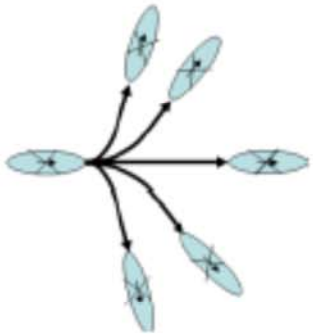
Motion planning: one problem, many different solutions… search-based planning

- Generate a graph representation of the planning problem
- Search the graph for a solution
- Graph construction and graph search can be interleaved (i.e., construct only what is necessary)

2D grid-based graph representation for 2D (x, y) search-based planning

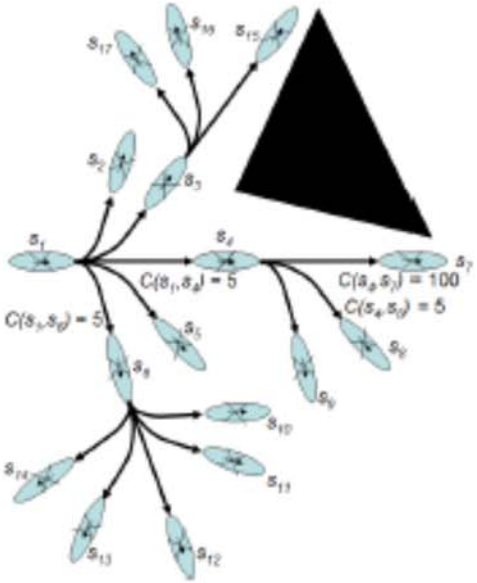


Lattice-based graph representation for 3D (x, y, θ) planning

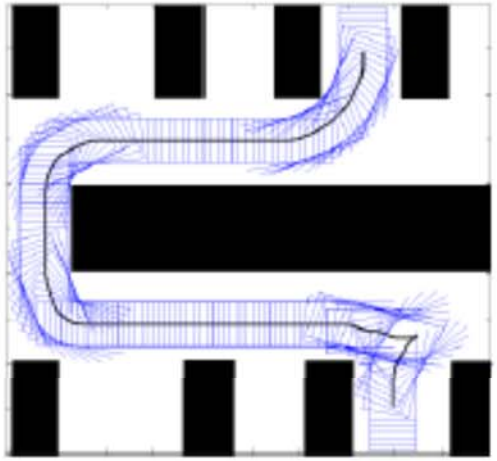


motion primitives

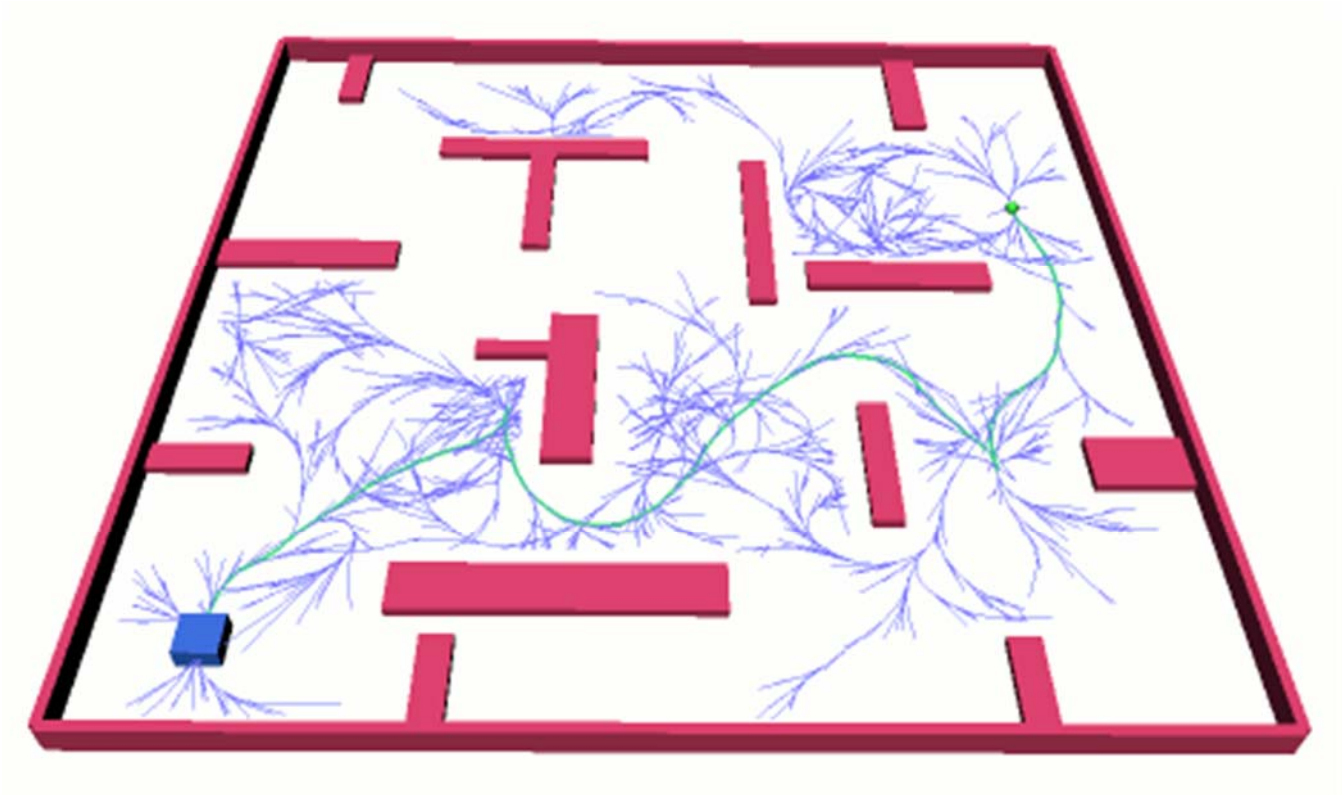
construct the graph:



search the graph for solution:



Motion planning: one problem, many different solutions… sampling-based planning



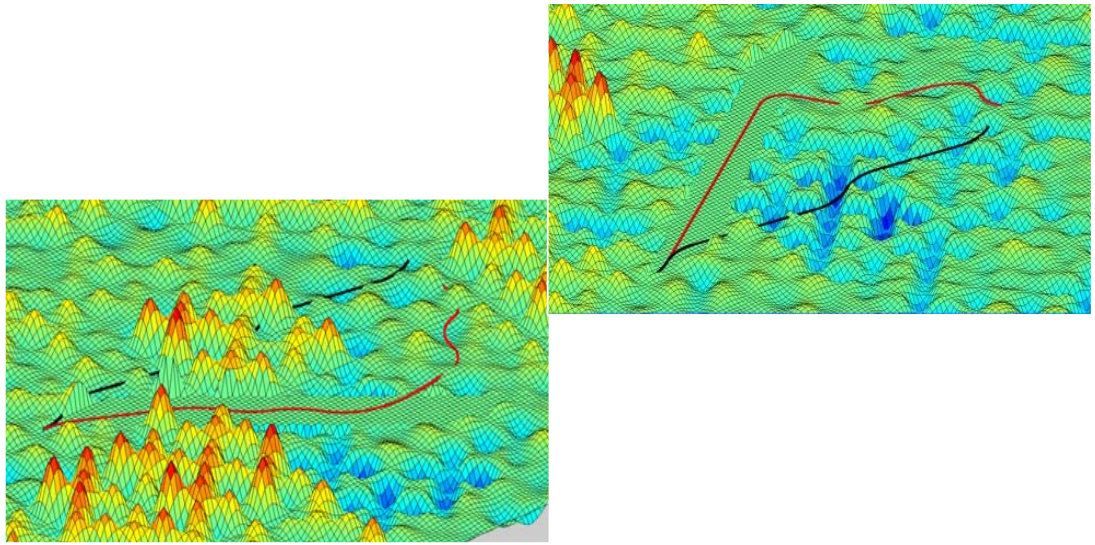
Motion planning: one problem, many different solutions… model-based planning

At each sample time the planner finds the best local trajectory within the sensor range minimizing the cost function

$$J(\mathbf{u}) = \int_{t_0}^{t_0+T} \gamma(\mathbf{x}, \mathbf{u})\, \mathrm{d}t + \Gamma(t_0 + T)$$

under the constraints

- vehicle kinematic model
- control constraints
- safe stopping constraint



Sampling-based planning algorithms: preliminary definitions

We now focus on sampling-based planning algorithms, and we start introducing some definitions and formulating the problem.

Discussing kinematics of mobile robots we introduce the concepts of system configuration and Lagrange or generalized coordinates.

We call:

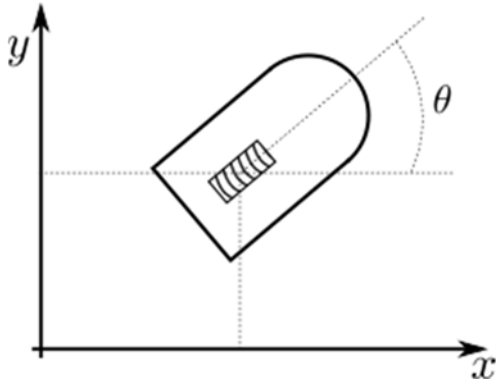
- *configuration space* $\mathcal{Q} \subset \mathbb{R}^d$, the set representing all the possible system configurations;
- *obstacle region* $\mathcal{Q}_{obs} \subset \mathcal{Q}$, the set including the configurations that lead to collisions;
- *free space*, the set $\mathcal{Q}_{free} := \mathcal{Q} \backslash \mathcal{Q}_{obs}$;
- *initial condition* $\boldsymbol{q}_{init} \in \mathcal{Q}_{free}$, is the initial robot configuration;
- *goal region* $\mathcal{Q}_{goal} \subset \mathcal{Q}_{free}$, the set of configurations the robot should reach.

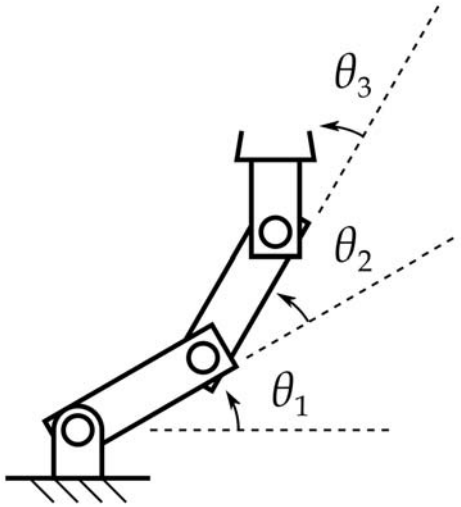
Configuration space: a mobile robot and a manipulator

In the case of a unicycle mobile robot the configuration is described by

$$\mathbf{q} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$$

and the configuration space is $\mathcal{Q} \subset \mathbb{R}^3$.





In the case of a serial manipulator the configuration space is given by the joint coordinates

$$\mathbf{q} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \end{bmatrix}$$

and the configuration space is $\mathcal{Q} \subset \mathbb{R}^3$.

Configuration space: topological structure

More in general, the generalized coordinates describing the motion of a robot can be:

- *Cartesian coordinates* (e.g., (x, y) position of a mobile robot), taking value in Euclidean spaces
- *angular coordinates* (e.g., orientation θ of a mobile robot), taking value in $SO(m)$

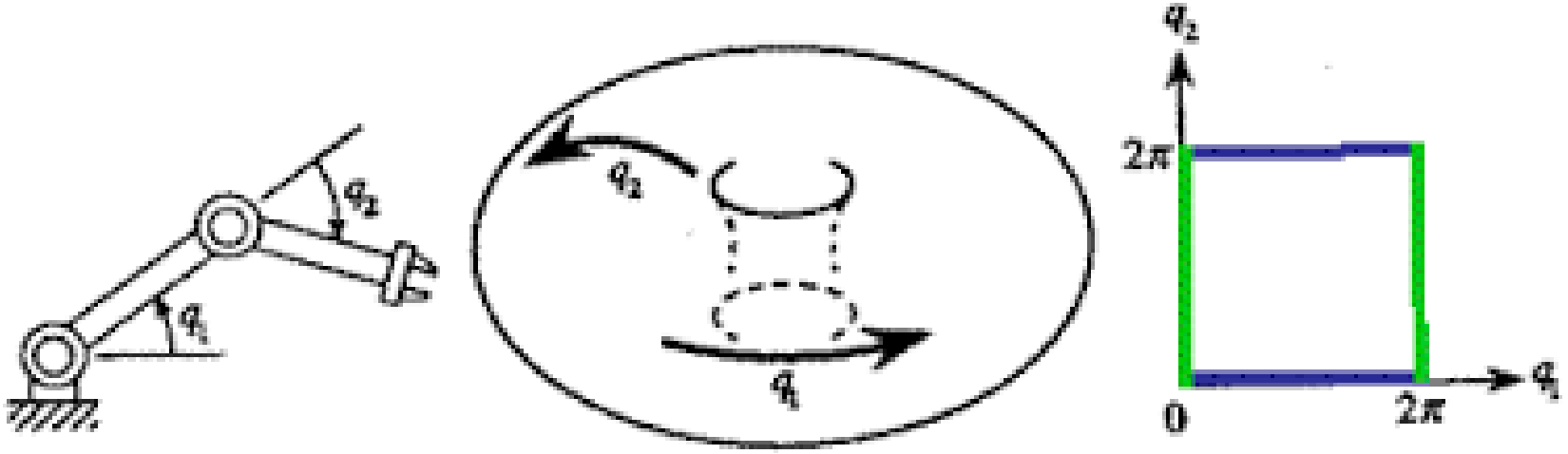
$SO(m)$ is the *special orthonormal group* of real $m \times m$ matrices with

- orthonormal columns
- unitary determinant

The configuration space is obtained as a Cartesian product of these spaces.

Configuration space: examples of topological structures

For a fixed-base planar manipulator with 2 revolute joints the configuration space is a torus, a subset of $\left(\mathbb{R}^2 \times SO(2)\right)^2$, and has dimension 2.



This configuration space can be locally represented as a subset of $\mathbb{R}^2$.

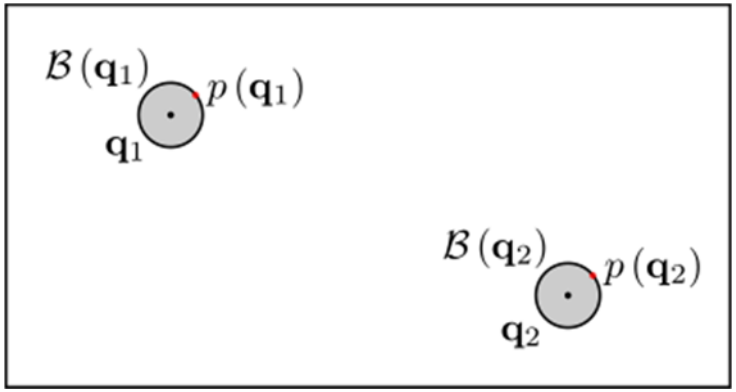
Configuration space: distance

As the configuration space is not an Euclidean space, how to compute a distance in $\mathcal{Q}$?

Consider two configurations $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$

- $\mathcal{B}(\boldsymbol{q}_1)$, $\mathcal{B}(\boldsymbol{q}_2)$ are the subsets of the workspace occupied by the robot in the two configurations
- $p(\boldsymbol{q}_1)$, $p(\boldsymbol{q}_2)$ the positions of a point p on the robot in the two configurations

The distance between two configurations $\boldsymbol{q}_1$, $\boldsymbol{q}_2$ tends to zero if the regions occupied by the robot $\mathcal{B}(\boldsymbol{q}_1)$, $\mathcal{B}(\boldsymbol{q}_2)$ tend to coincide.



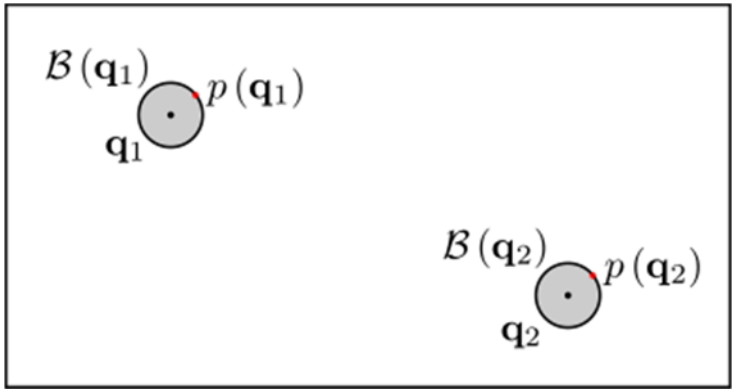
A distance that satisfy this property is

$$d(\mathbf{q}_1, \mathbf{q}_2) = \max_{p \in \mathcal{B}} \|p(\mathbf{q}_1) - p(\mathbf{q}_2)\|$$

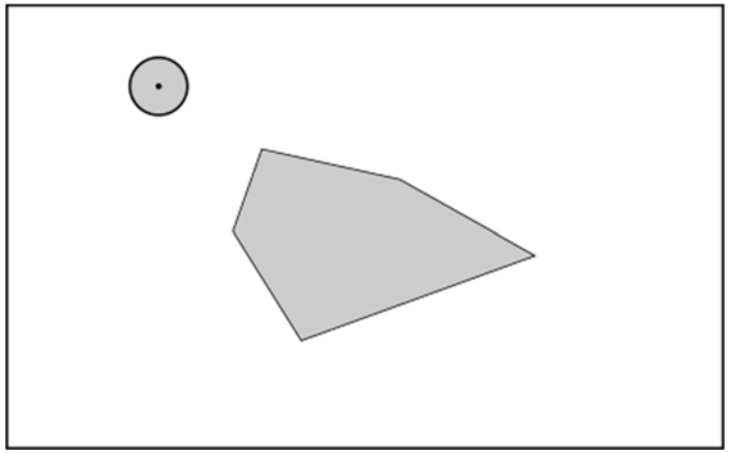
Using this distance is not convenient as it requires to characterize the volume occupied by the robot in the two configurations.

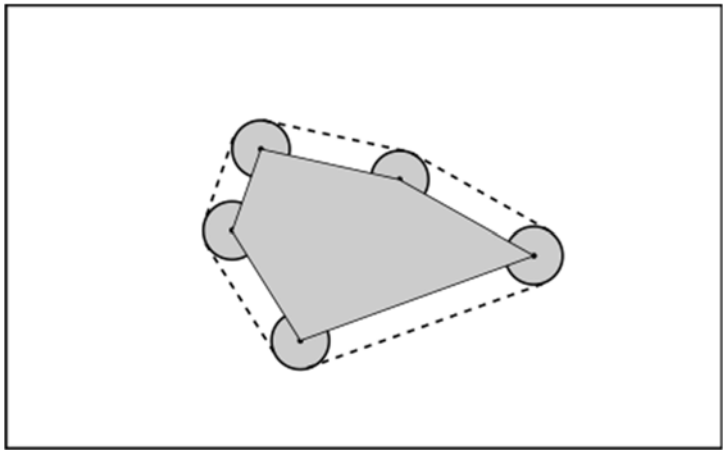
It is usually simplified adopting an Euclidean norm

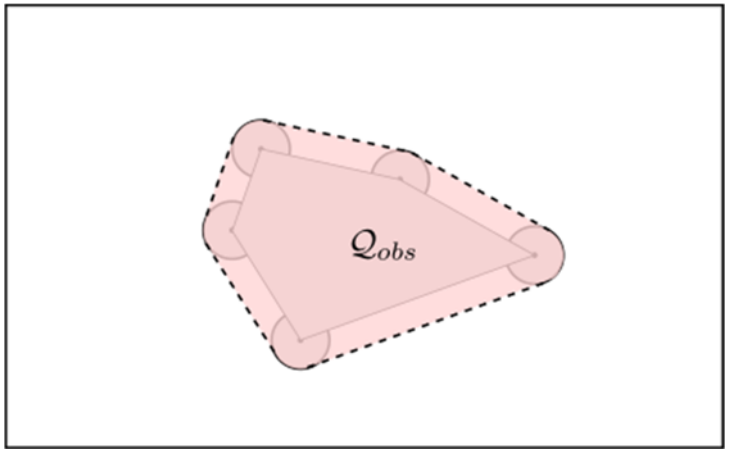
$$d(\mathbf{q}_1, \mathbf{q}_2) = \|\mathbf{q}_1 - \mathbf{q}_2\|$$



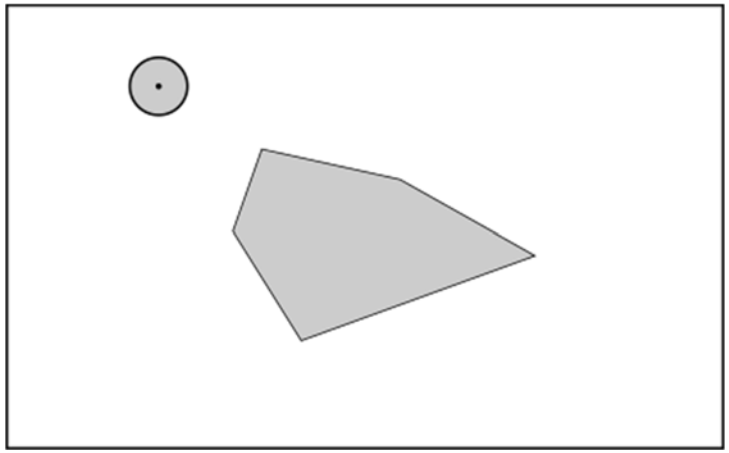
In order to consider obstacles we must derive their image in the configuration space, and we also need to determine the configuration space starting from the workspace.

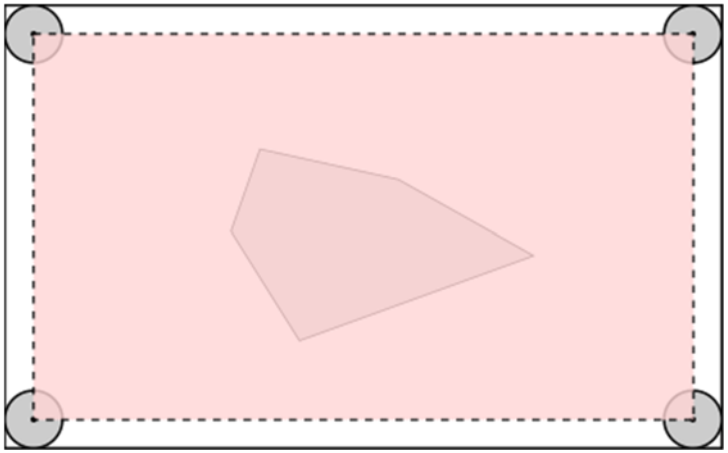


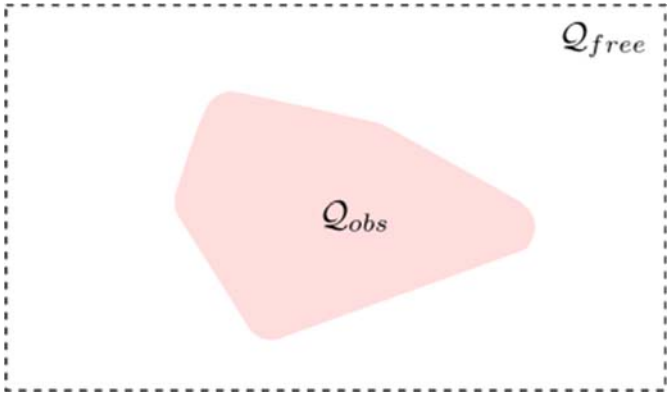




In order to consider obstacles we must derive their image in the configuration space, and we also need to determine the configuration space starting from the workspace.







Sampling-based planning algorithms: preliminary definitions

A function $\sigma: [0,1] \rightarrow \mathbb{R}^d$ is a

- *path*, if it is continuous;
- *collision-free path*, if it is a path and $\sigma(\tau) \in \mathcal{Q}_{free}$ for all $\tau \in [0,1]$;
- *feasible path*, if it is a collision-free path, $\sigma(0) \in \boldsymbol{q}_{init}$ and $\sigma(1) \in \mathcal{Q}_{goal}$.

We first introduce the *feasible path planning* problem:

Given a path planning problem $(\mathcal{Q}_{free}, \boldsymbol{q}_{init}, \mathcal{Q}_{goal})$, *find a feasible path* $\sigma: [0,1] \rightarrow \mathcal{Q}_{free}$ *such that* $\sigma(0) = \boldsymbol{q}_{init}$ *and* $\sigma(1) \in \mathcal{Q}_{goal}$, *if one exists. If no such path exists, report failure.*

All the algorithms we introduce are based on some common primitive procedures.

Sampling

- $Sample: \omega \rightarrow \{Sample_i(\omega)\}_{i \in \mathbb{N}} \subset \mathcal{Q}$, is a function that returns a random configuration sampled from $\mathcal{Q}$. Samples are independent and identically distributed, and drawn from a uniform distribution.
- $SampleFree: \omega \rightarrow \{SampleFree_i(\omega)\}_{i \in \mathbb{N}} \subset \mathcal{Q}_{free}$, is a function that returns a random free configuration sampled from $\mathcal{Q}_{free}$.

Nearest neighbor

Given a graph $G = (V, E)$, where $V \subset \mathcal{Q}$, and a point $\boldsymbol{q} \in \mathcal{Q}$, the function

$$Nearest : (G, \mathbf{q}) \rightarrow \mathbf{v} \in V$$

returns the vertex in V that is the closest to $\boldsymbol{q}$ in terms of a given distance function.

We start considering the Euclidean distance

$$Nearest\,(G, \mathbf{q}) := \underset{\mathbf{v} \in V}{\operatorname{argmin}} \|\mathbf{q} - \mathbf{v}\|$$

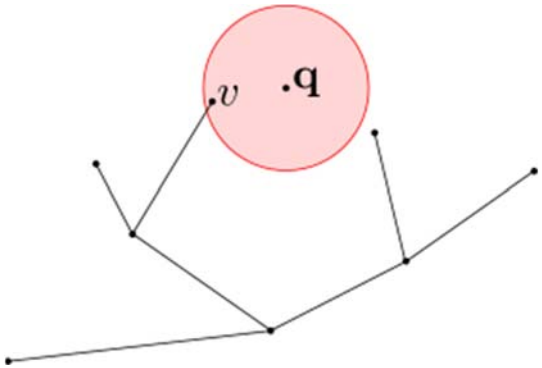
We can also consider some variants, like

$$kNearest : (G, \mathbf{q}, k) \rightarrow \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$$

returning the k vertices in V that are the nearest to $\boldsymbol{q}$, or

$$Near : (G, \mathbf{q}, r) \rightarrow V' \subset V$$

returning the vertices in V that are contained in a ball or radius r centered at $\boldsymbol{q}$.



Steering

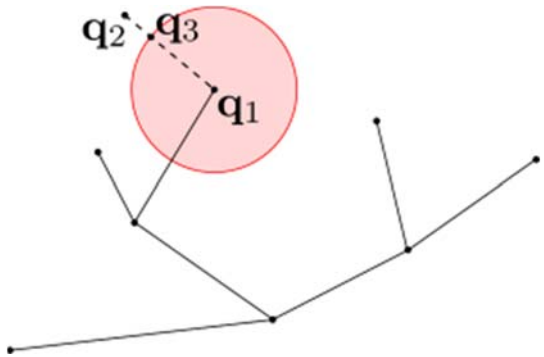
Given two configurations $\boldsymbol{q}_1, \boldsymbol{q}_2 \in \mathcal{Q}$, the function $Steer: (\boldsymbol{q}_1, \boldsymbol{q}_2) \rightarrow \boldsymbol{q}_3$ returns a configuration $\boldsymbol{q}_3 \in \mathcal{Q}$ that is "closer" to $\boldsymbol{q}_2$ than $\boldsymbol{q}_1$ is (*non-exact steering*).

Considering again the Euclidean distance

$$Steer(\mathbf{q}_1, \mathbf{q}_2) := \underset{\mathbf{q}_3 \in \mathbf{B}_{\mathbf{q}_1, \eta}}{\operatorname{argmin}} \|\mathbf{q}_3 - \mathbf{q}_2\|$$

Collision test

Given two configurations $\boldsymbol{q}_1, \boldsymbol{q}_2 \in \mathcal{Q}$, the Boolean function $CollisionFree(\boldsymbol{q}_1, \boldsymbol{q}_2)$ return *true* if the line segment between $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ lies in $\mathcal{Q}_{free}$, *false* otherwise.



Probabilistic RoadMaps (PRM): the algorithm

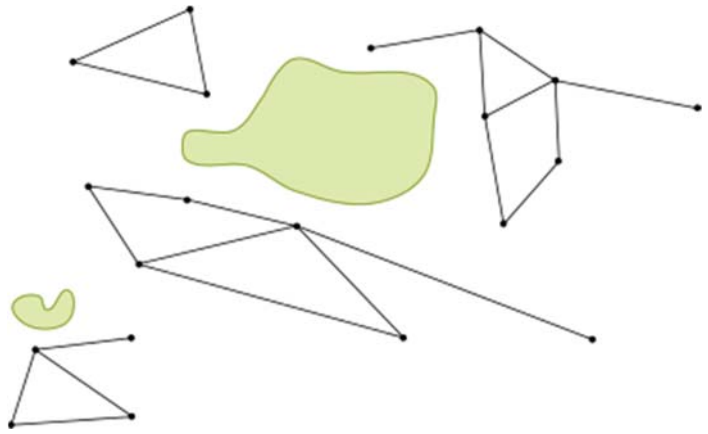
Given a path planning problem $(\mathcal{Q}_{free}, \boldsymbol{q}_{init}, \mathcal{Q}_{goal})$ and an integer N, PRM consists of two phases:

- *roadmap construction*, this is a pre-processing phase in which a roadmap is constructed by attempting connections among N randomly sampled configurations in $\mathcal{Q}_{free}$;
- *query phase*, paths connecting initial and final configurations are extracted from the roadmap.

We now analyze the algorithm describing the first phase, the query phase is a simple search of a path in the graph.

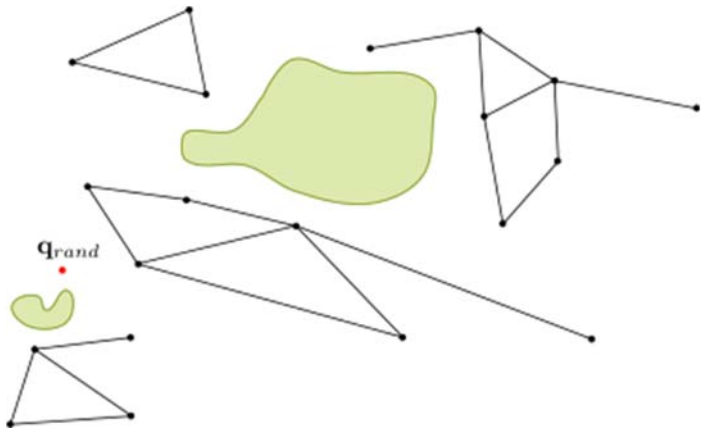
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



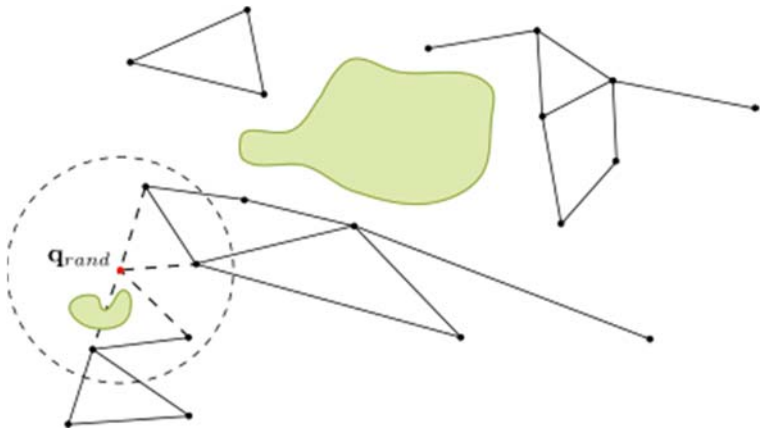
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



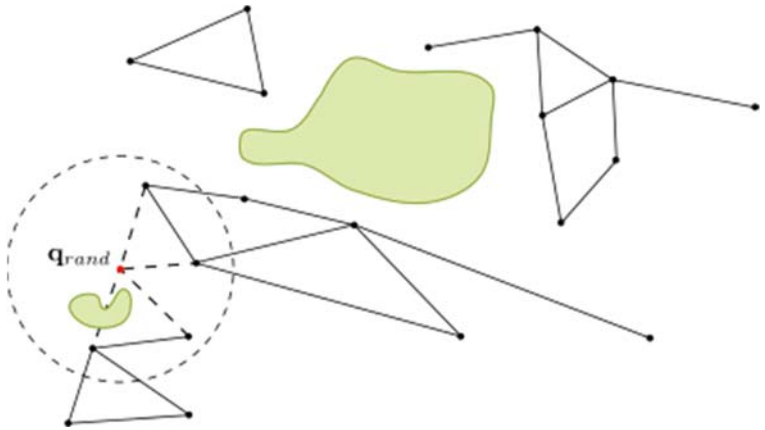
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



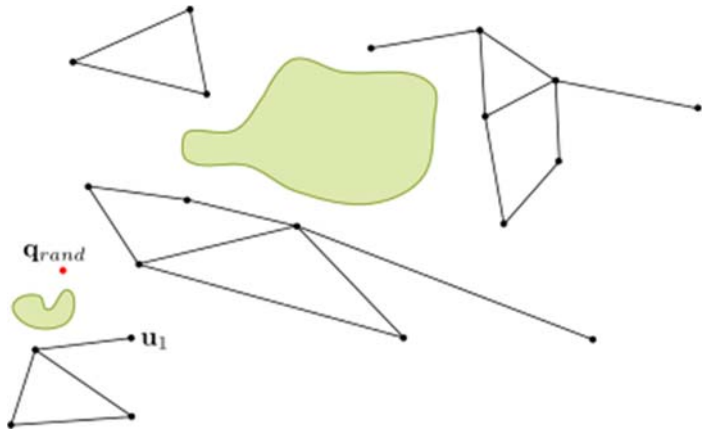
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



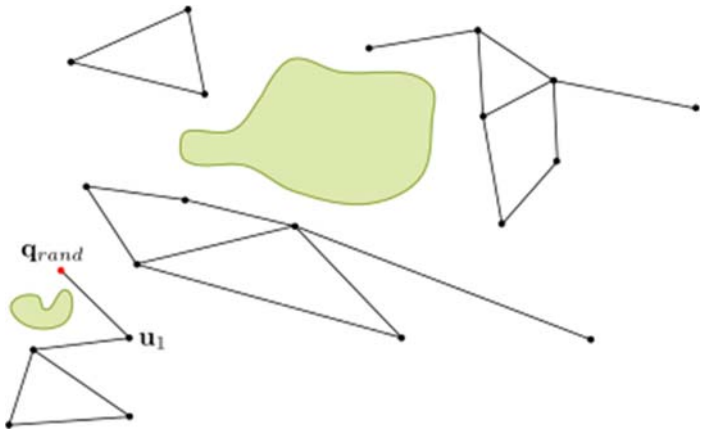
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



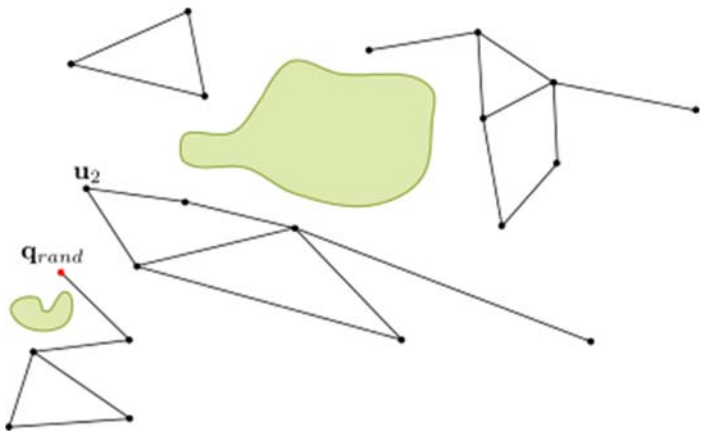
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



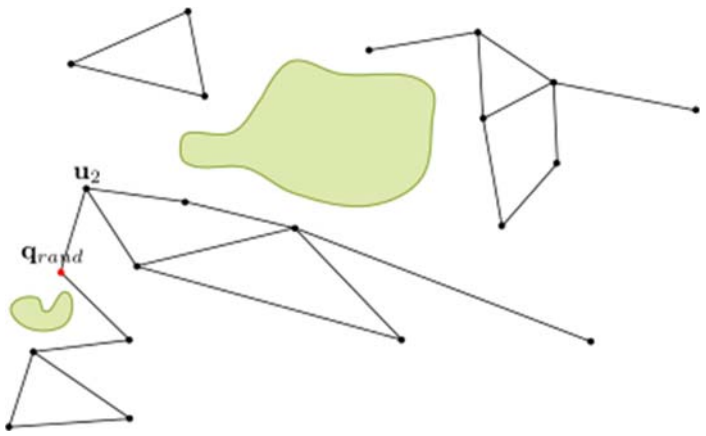
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



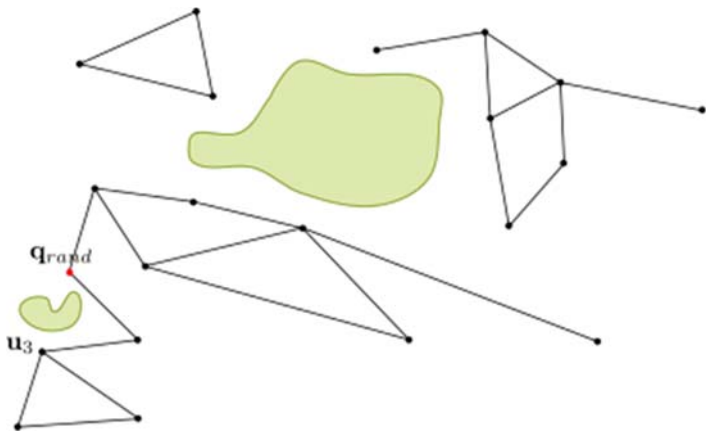
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



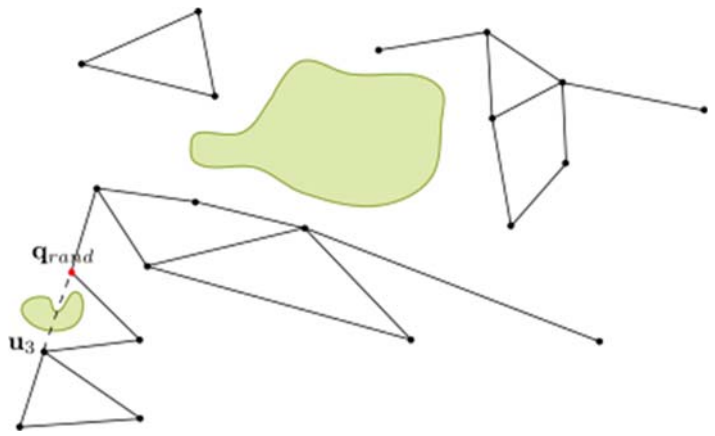
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



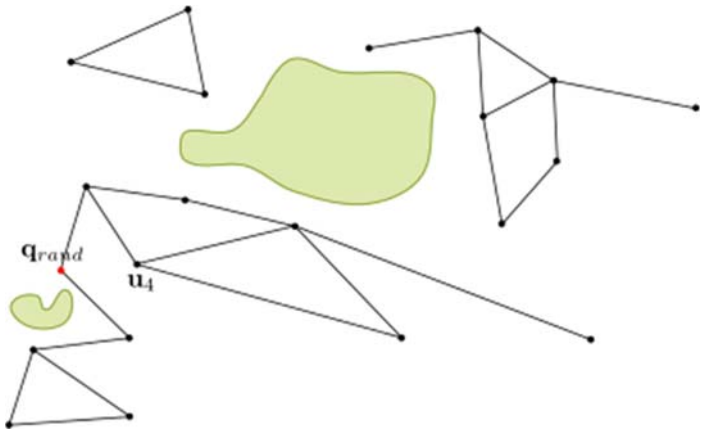
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



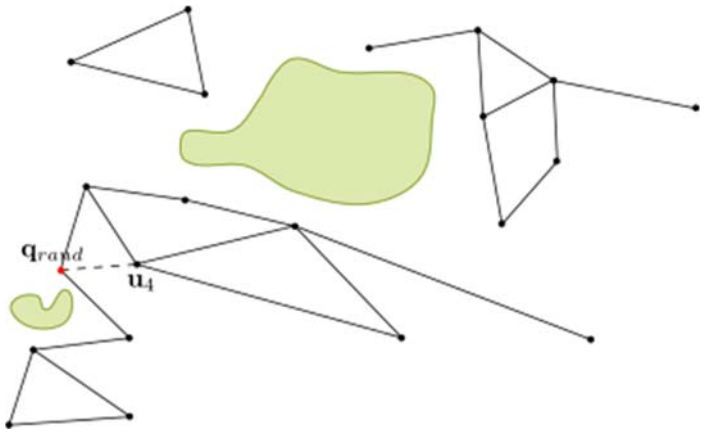
Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



Probabilistic RoadMaps (PRM): roadmap construction

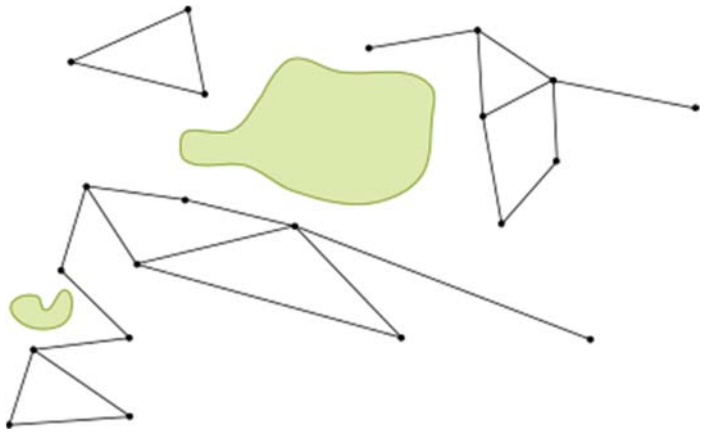
$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$



Probabilistic RoadMaps (PRM): roadmap construction

$V \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $i = 0, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $U \leftarrow Near(G, \mathbf{q}_{rand}, r)$;
  $V \leftarrow V \cup \{\mathbf{q}_{rand}\}$;
  foreach $\mathbf{u} \in U$ *in order of increasing* $\|\mathbf{u} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ *and* $\mathbf{u}$ *are not in the same connected component of* G **then**
      if $CollisionFree(\mathbf{q}_{rand}, \mathbf{u})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{rand}, \mathbf{u})\}$;
      end
    end
  end
end
return $G = (V, E)$

$N = 21$



Probabilistic RoadMaps (PRM): variants

The result of the execution of this part of the algorithm is a roadmap composed of N nodes.

This roadmap is a *forest*, collection of trees.

The algorithm can be simplified (*sPRM*) allowing connections between vertices in the same connected component.

PRM and sPRM can be implemented considering different choices for the set U of vertices to which connection are attempted:

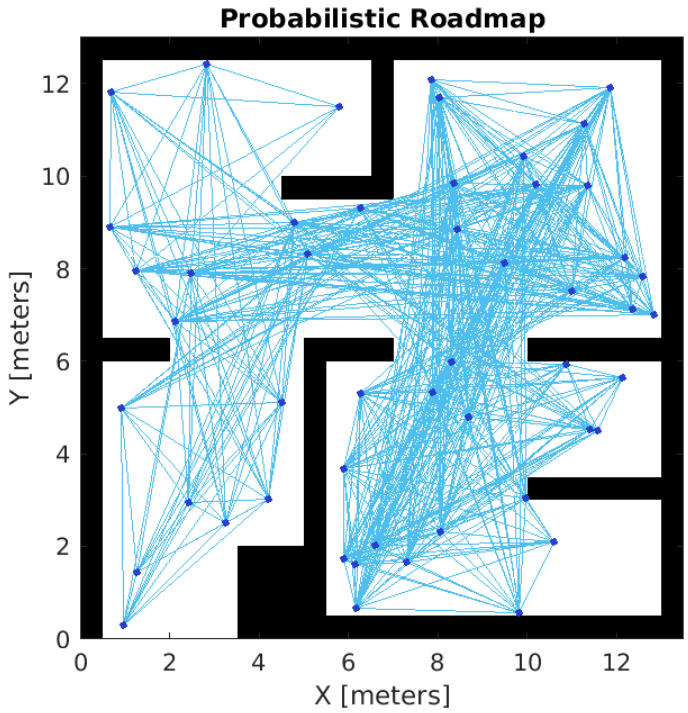
- $Near$, the vertices in V that are contained in a ball of radius r are considered;
- $kNearest$, the nearest k neighbors are considered;
- $Near$ with variable radius, the vertices in V that are contained in a ball of radius $r(N)$ are considered.

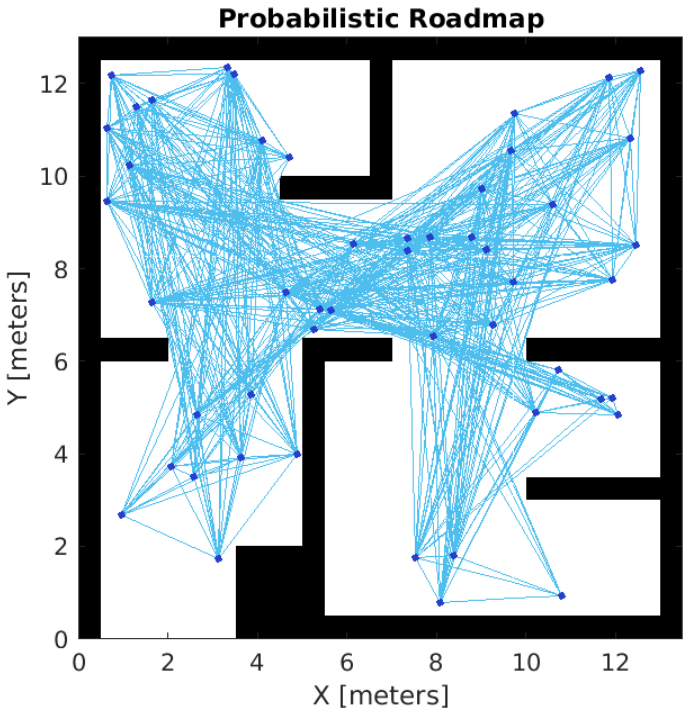
Probabilistic RoadMaps (PRM): “simplified” roadmap construction

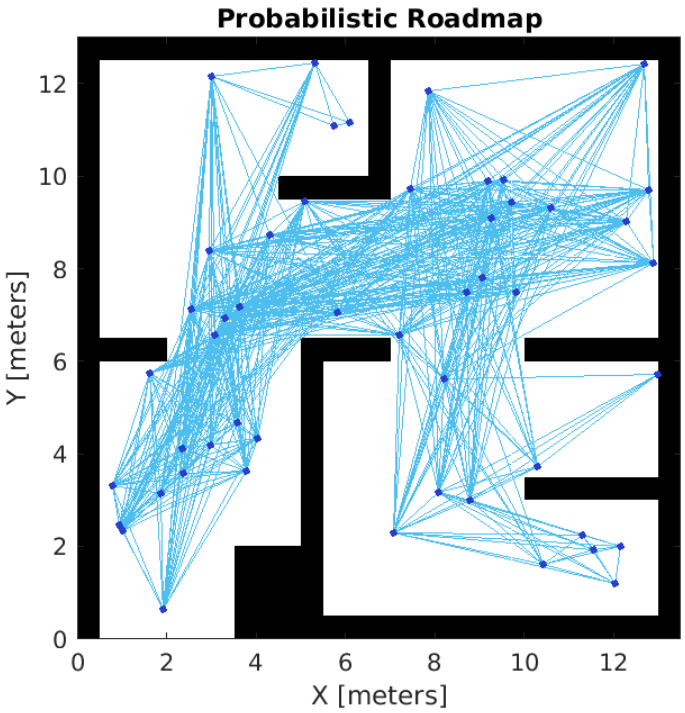
$V \leftarrow \{\mathbf{q}_{init}\} \cup \{SampleFree_i, i = 1, \ldots, N\}$;
$E \leftarrow \emptyset$;
foreach $\mathbf{v} \in V$ **do**
  $U \leftarrow Near(G, \mathbf{v}, r) \setminus \{\mathbf{v}\}$;
  foreach $\mathbf{u} \in U$ **do**
    if $CollisionFree(\mathbf{v}, \mathbf{u})$ **then**
      $E \leftarrow E \cup \{(\mathbf{v}, \mathbf{u})\}$;
    end
  end
end
return $G = (V, E)$

Probabilistic RoadMaps (PRM): an example

Three executions of sPRM with $N = 50$

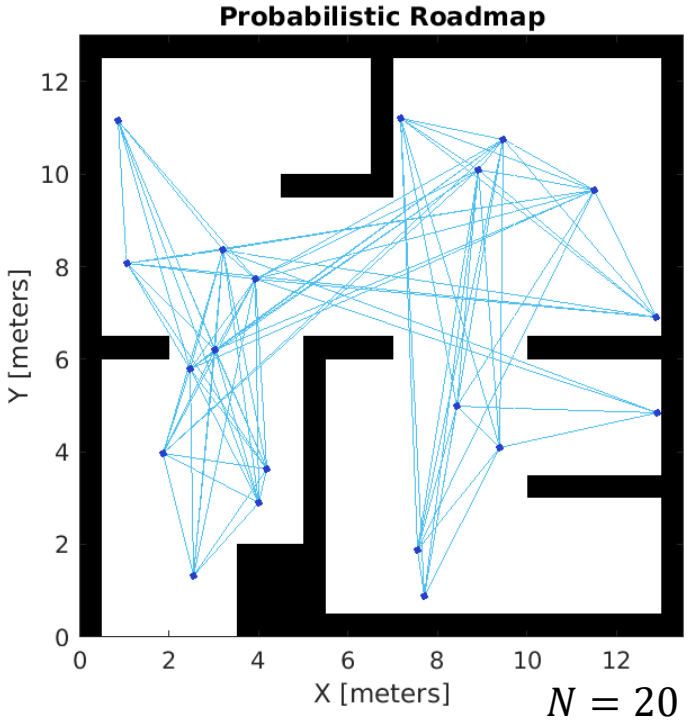




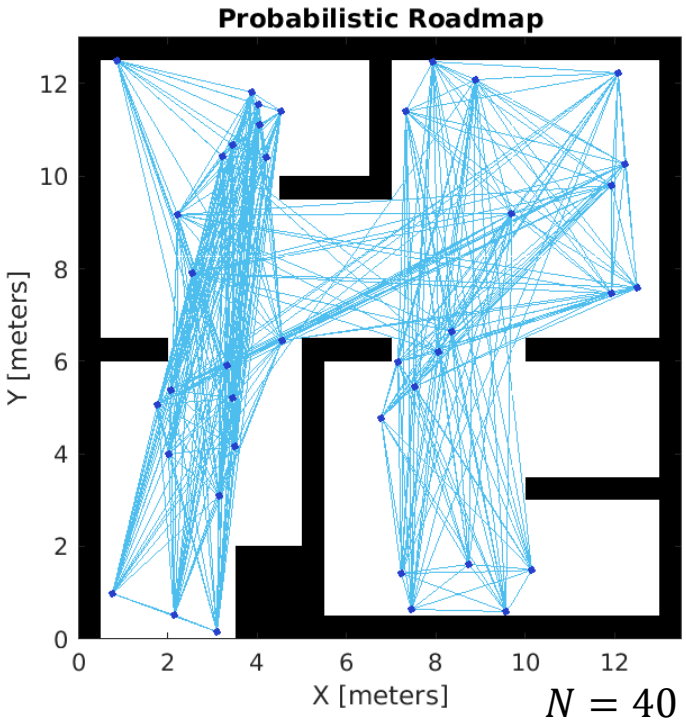


Probabilistic RoadMaps (PRM): an example

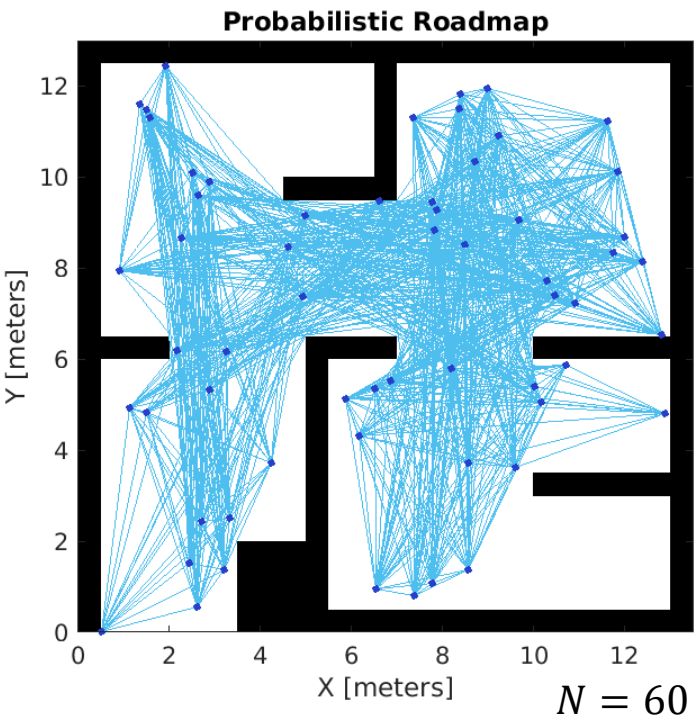
Five executions of sPRM with an increasing number of nodes



$N = 20$



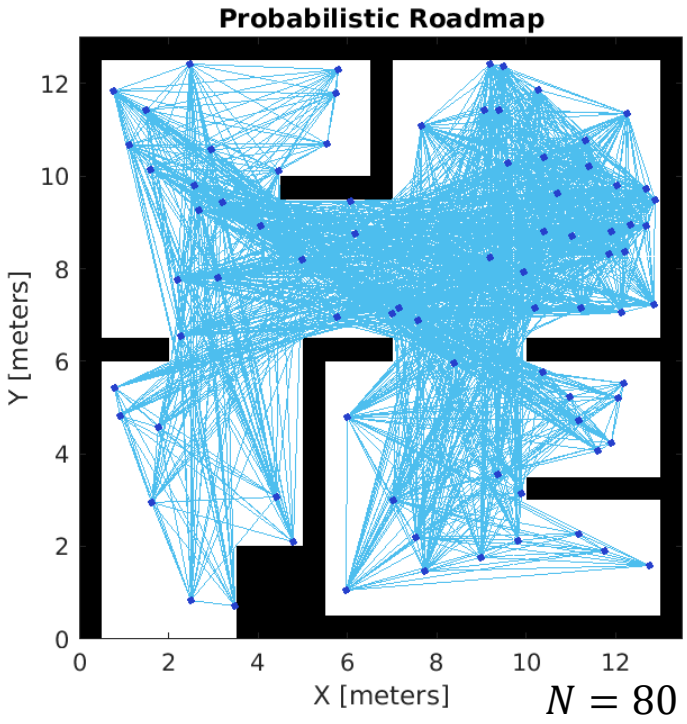
$N = 40$



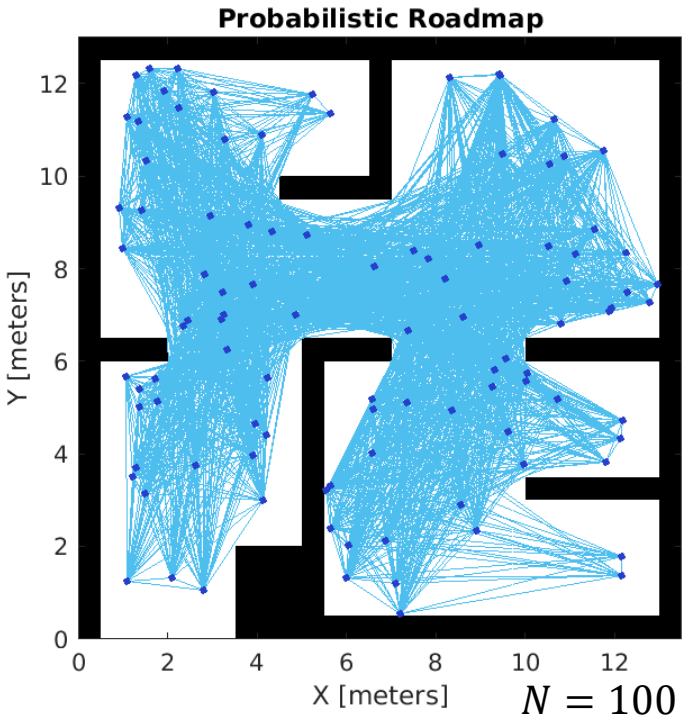
$N = 60$

Probabilistic RoadMaps (PRM): an example

Five executions of sPRM with an increasing number of nodes



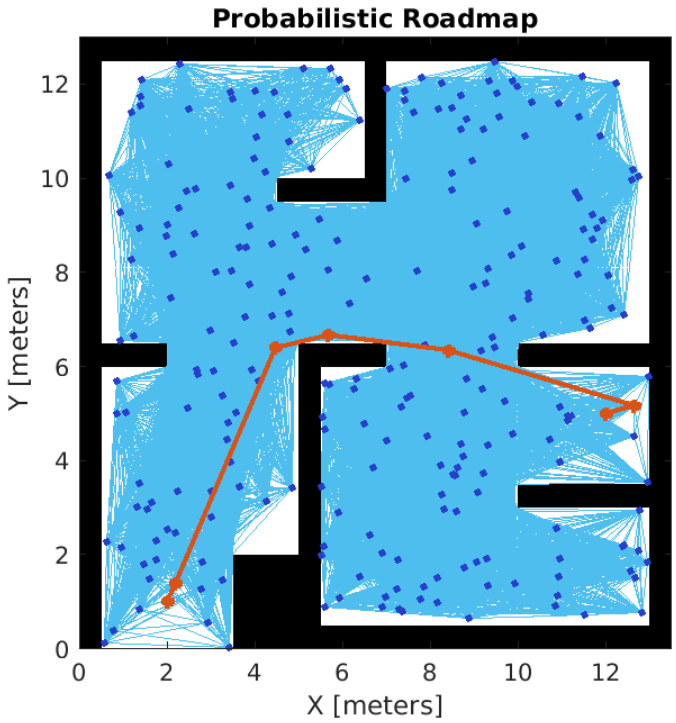
$N = 80$

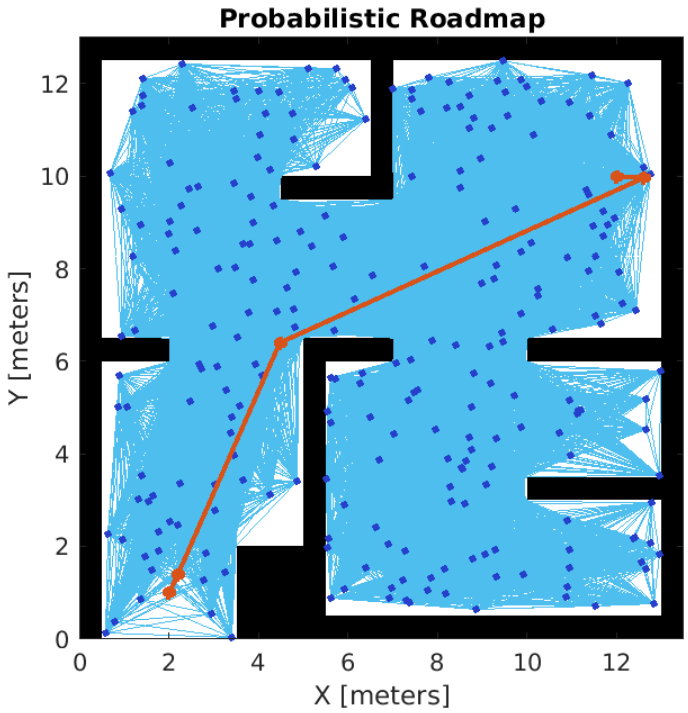


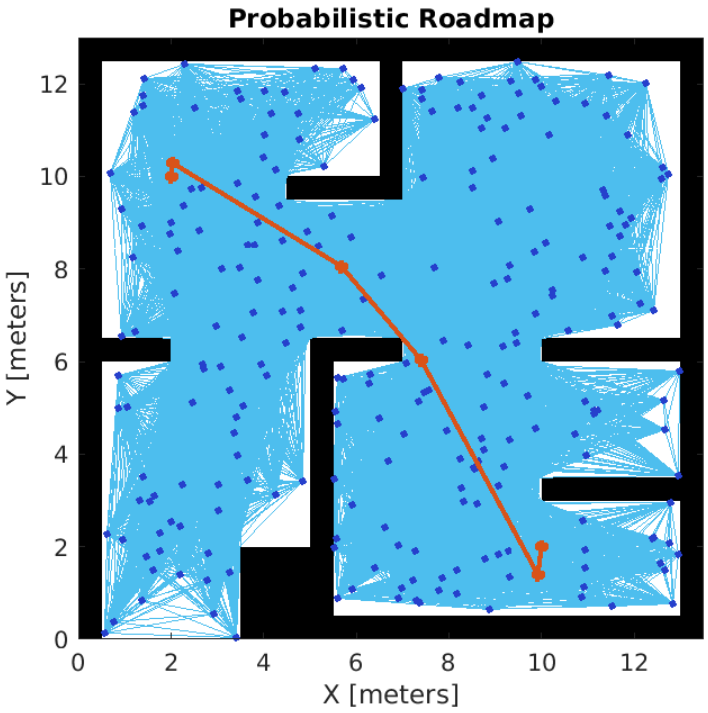
$N = 100$

Probabilistic RoadMaps (PRM): an example

Querying different paths using the same roadmap ($N = 250$)







Rapidly-exploring Random Trees (RRT): the algorithm

Given a path planning problem $(\mathcal{Q}_{free}, \boldsymbol{q}_{init}, \mathcal{Q}_{goal})$ and an integer N, RRT aims at a single query application.

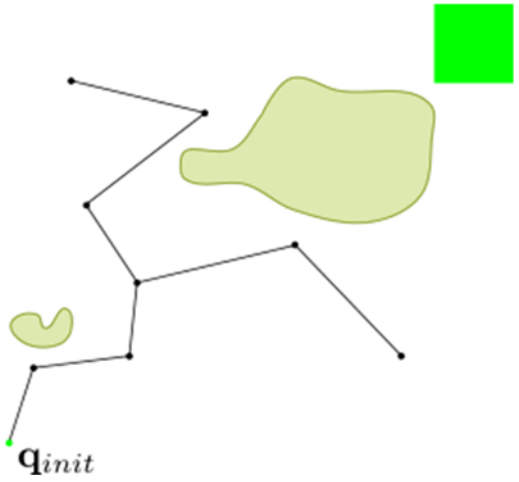
There is thus no distinction between graph construction and query, the algorithm incrementally builds a tree of feasible trajectories, rooted at the initial configuration.

We now analyze the basic version of the algorithm.

Rapidly-exploring Random Trees (RRT): the algorithm

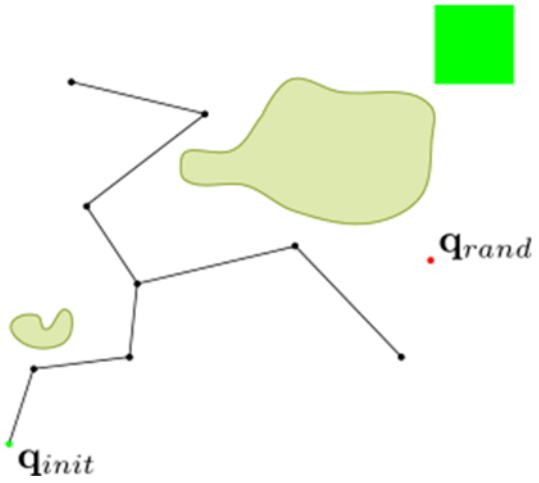
$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
  end
end
return $G = (V, E)$

Or it can be stopped when the goal region is reached



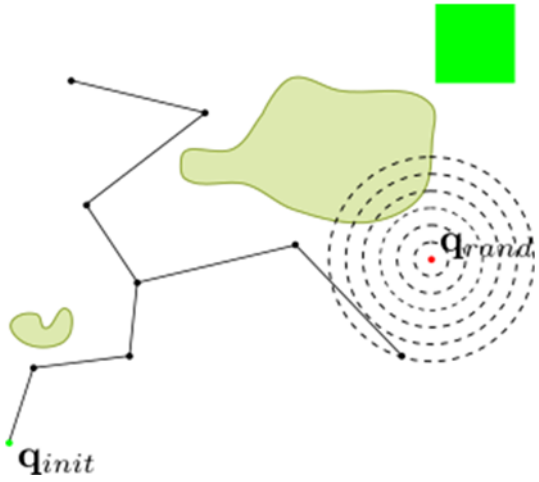
Rapidly-exploring Random Trees (RRT): the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
  end
end
return $G = (V, E)$



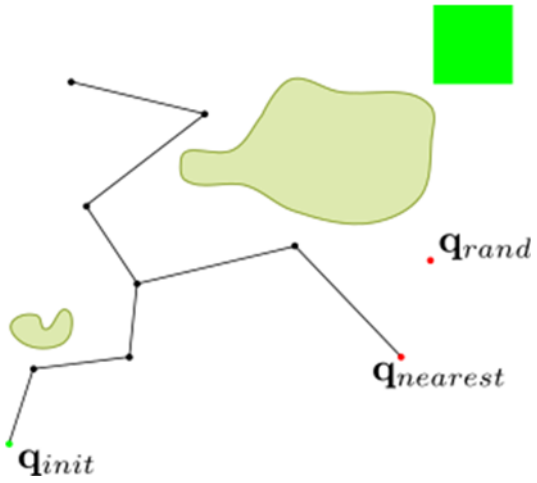
Rapidly-exploring Random Trees (RRT): the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
  end
end
return $G = (V, E)$



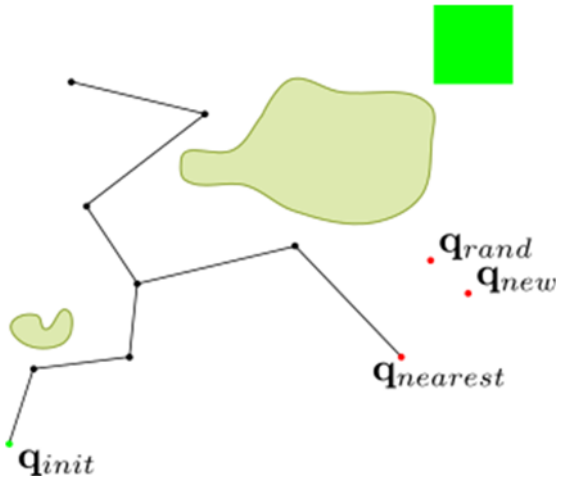
Rapidly-exploring Random Trees (RRT): the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
  end
end
return $G = (V, E)$



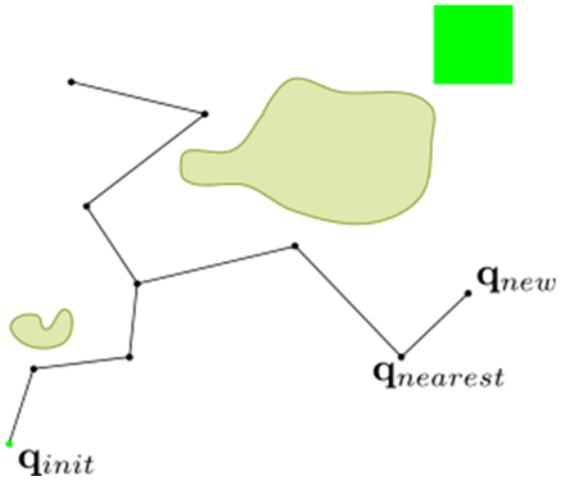
Rapidly-exploring Random Trees (RRT): the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
  end
end
return $G = (V, E)$

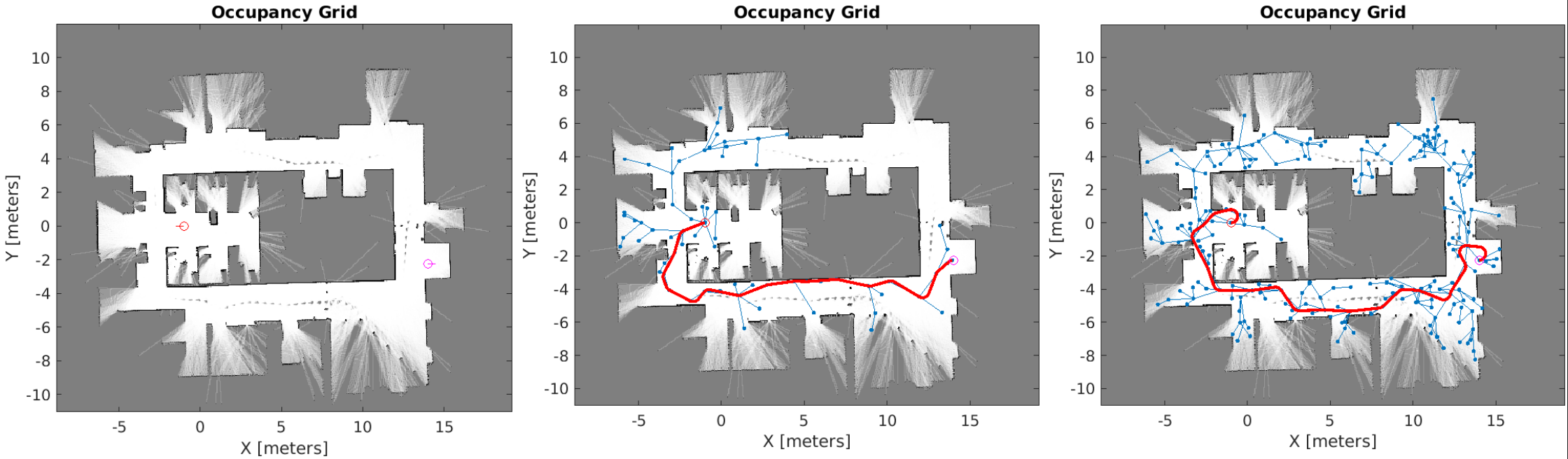


Rapidly-exploring Random Trees (RRT): the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
  end
end
return $G = (V, E)$



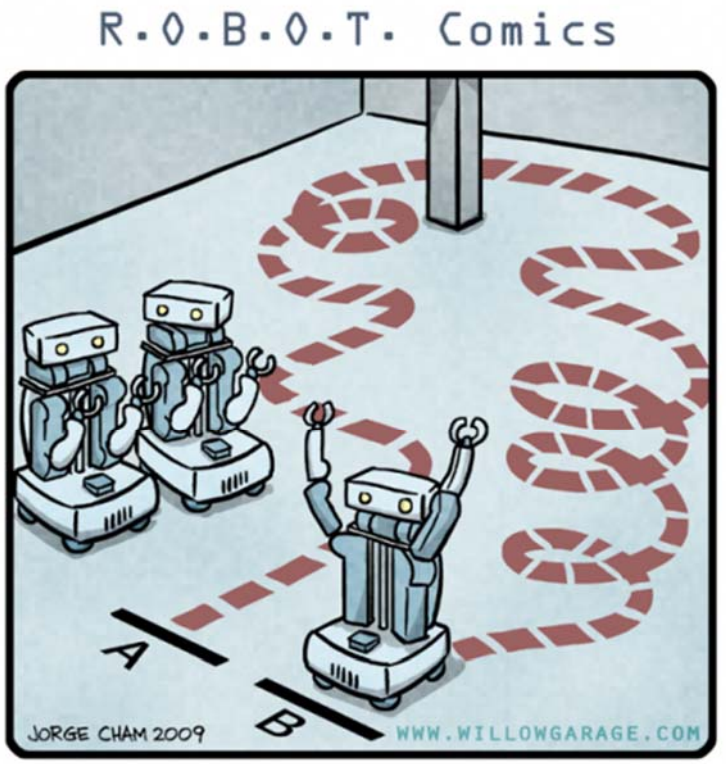
Rapidly-exploring Random Trees (RRT): an example



Some comments on the main characteristics of the two planners:

- both consider a non-exact steering function, neglecting robot kinematic and dynamic constraints
- both compute a generic path from the initial to the desired configuration, without considering any optimality criteria (time, length, actuation effort, etc.)

In robotic applications optimality is often an important issue to deal with…



The problem addressed by PRM and RRT is the *feasible path planning* problem:

Given a path planning problem $(\mathcal{Q}_{free}, \boldsymbol{q}_{init}, \mathcal{Q}_{goal})$, *find a feasible path* $\sigma: [0,1] \rightarrow \mathcal{Q}_{free}$ *such that* $\sigma(0) = \boldsymbol{q}_{init}$ *and* $\sigma(1) \in \mathcal{Q}_{goal}$, *if one exists. If no such path exists, report failure.*

Let Σ denote the set of all paths, and Σ_{free} the set of all collision-free paths.

We introduce a function $c: \Sigma \rightarrow \mathbb{R}_{\geq 0}$, called *cost function*, that is assumed to be:

- *monotonic*, i.e., for all $\sigma_1, \sigma_2 \in \Sigma$, $c(\sigma_1) \leq c(\sigma_1 | \sigma_2)$
- *bounded*

We can now introduce the *optimal path planning* problem…

We can now introduce the *optimal path planning* problem:

Given a path planning problem $(\mathcal{Q}_{free}, \boldsymbol{q}_{init}, \mathcal{Q}_{goal})$, *and a cost function* $c: \Sigma \to \mathbb{R}_{\geq 0}$, *find a feasible path* $\sigma^\star$ *such that* $c(\sigma^\star) = \min\{c(\sigma) : \sigma \text{ is feasible}\}$. *If no such path exists, report failure.*

From PRM and RRT we can now derive two *asymptotically optimal* planners, PRM* and RRT*.

Optimal sampling-based planning: from PRM to PRM*

$V \leftarrow \{\mathbf{q}_{init}\} \cup \{SampleFree_i, i = 1, \ldots, N\};$
$E \leftarrow \emptyset;$
foreach $\mathbf{v} \in V$ **do**
  $U \leftarrow Near(G, \mathbf{v}, r) \setminus \{\mathbf{v}\};$
  foreach $\mathbf{u} \in U$ **do**
    if $CollisionFree(\mathbf{v}, \mathbf{u})$ **then**
      $E \leftarrow E \cup \{(\mathbf{v}, \mathbf{u})\};$
    end
  end
end
return $G = (V, E)$

$V \leftarrow \{\mathbf{q}_{init}\} \cup \{SampleFree_i, i = 1, \ldots, N\};$
$E \leftarrow \emptyset;$
foreach $\mathbf{v} \in V$ **do**
  $U \leftarrow Near\left(G, \mathbf{v}, \gamma_{PRM}\left(\log(N)/N\right)^{1/d}\right) \setminus \{\mathbf{v}\};$
  foreach $\mathbf{u} \in U$ **do**
    if $CollisionFree(\mathbf{v}, \mathbf{u})$ **then**
      $E \leftarrow E \cup \{(\mathbf{v}, \mathbf{u})\};$
    end
  end
end
return $G = (V, E)$

Optimal sampling-based planning: PRM*

$V \leftarrow \{\mathbf{q}_{init}\} \cup \{SampleFree_i, i = 1, \ldots, N\};$
$E \leftarrow \emptyset;$
foreach $\mathbf{v} \in V$ **do**
  $U \leftarrow Near\left(G, \mathbf{v}, \gamma_{PRM} \left(\log(N)/N\right)^{1/d}\right) \setminus \{\mathbf{v}\};$
  foreach $\mathbf{u} \in U$ **do**
    if $CollisionFree(\mathbf{v}, \mathbf{u})$ **then**
      $E \leftarrow E \cup \{(\mathbf{v}, \mathbf{u})\};$
    end
  end
end
return $G = (V, E)$

$$\gamma_{PRM} > \gamma^{\star}_{PRM} = 2\left(1 + 1/d\right)^{1/d} \left(\mu\left(\mathcal{Q}_{free}\right)/\zeta_d\right)^{1/d}$$

- d is the dimension of the configuration space
- ζ_d is the volume of the unit ball in the d-dimensional Euclidean space
- $\mu(\mathcal{Q}_{free})$ is the Lebesgue measure of the obstacle-free space

The rate of decay of the radius is such that the average number of connections attempted from a vertex is proportional to $log(N)$.

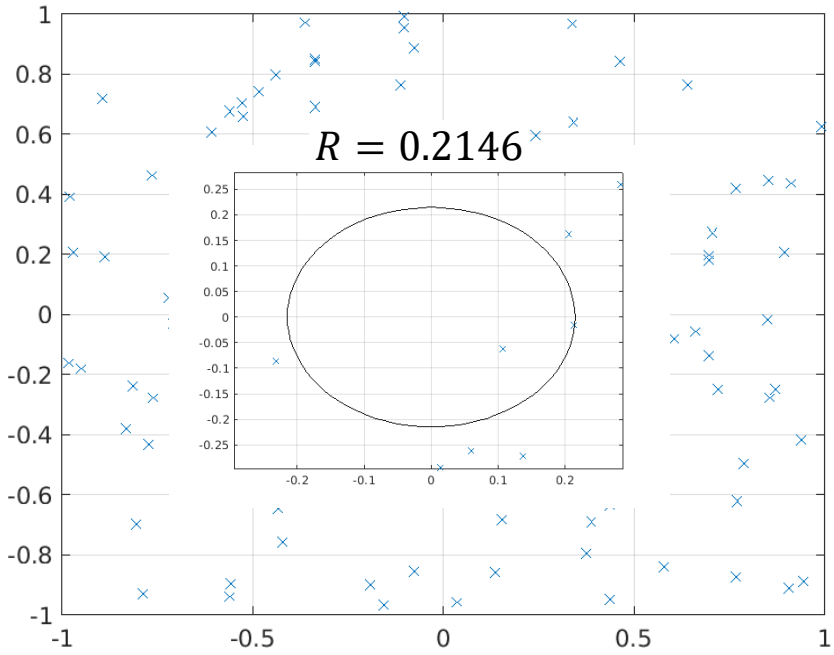
A variant of PRM, characterized by a variable radius exists.

The radius is suggested to be selected as a function of *sample dispersion*.

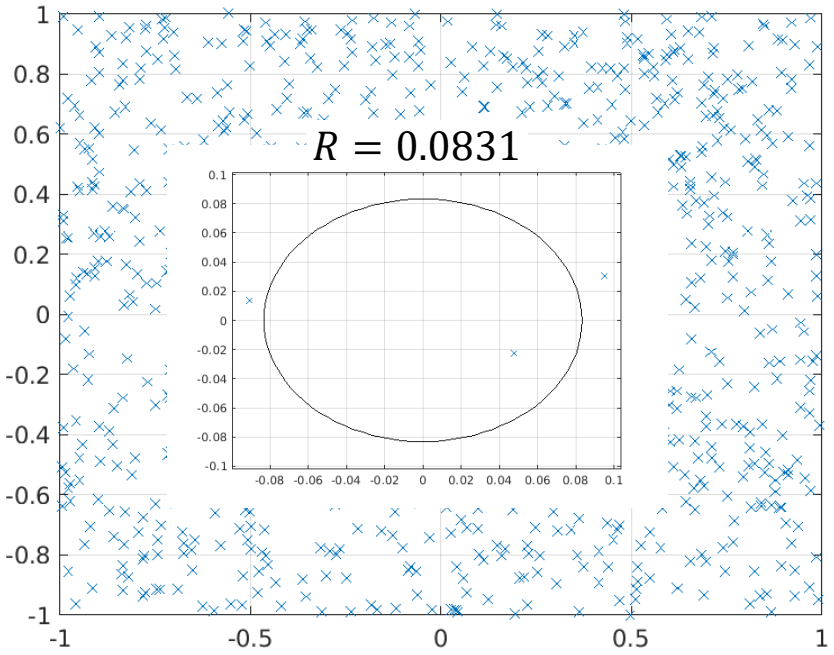
Dispersion of a point set contained in a bounded set $\mathcal{S} \subset \mathbb{R}^d$ is the radius of the largest empty ball centered in $\mathcal{S}$.

It can be shown that the dispersion of a set of n random points sampled uniformly and independently in a bounded set is $O\left((\log(N)/N)^{1/d}\right)$.

Optimal sampling-based planning: PRM*

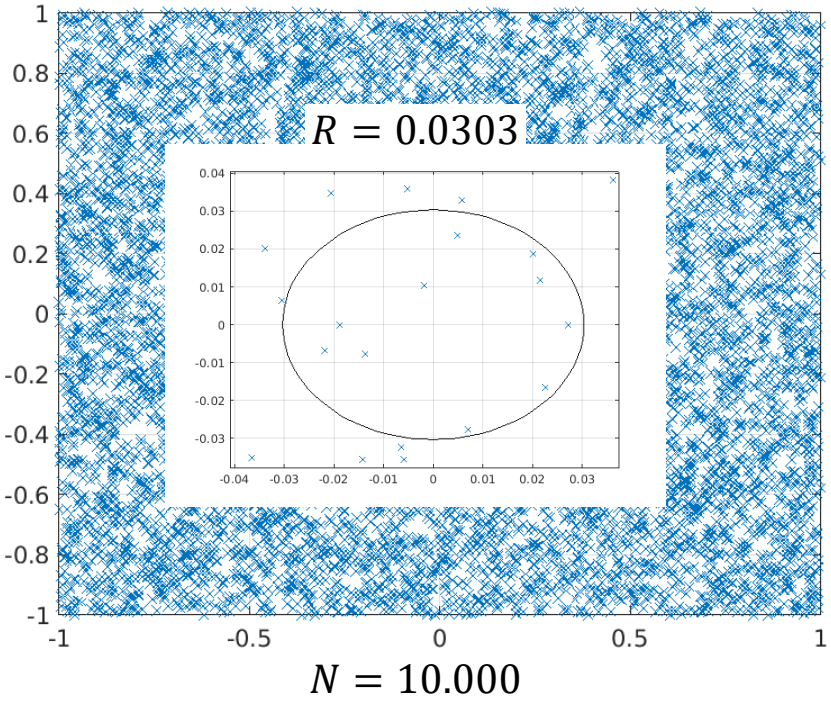


$N = 100$

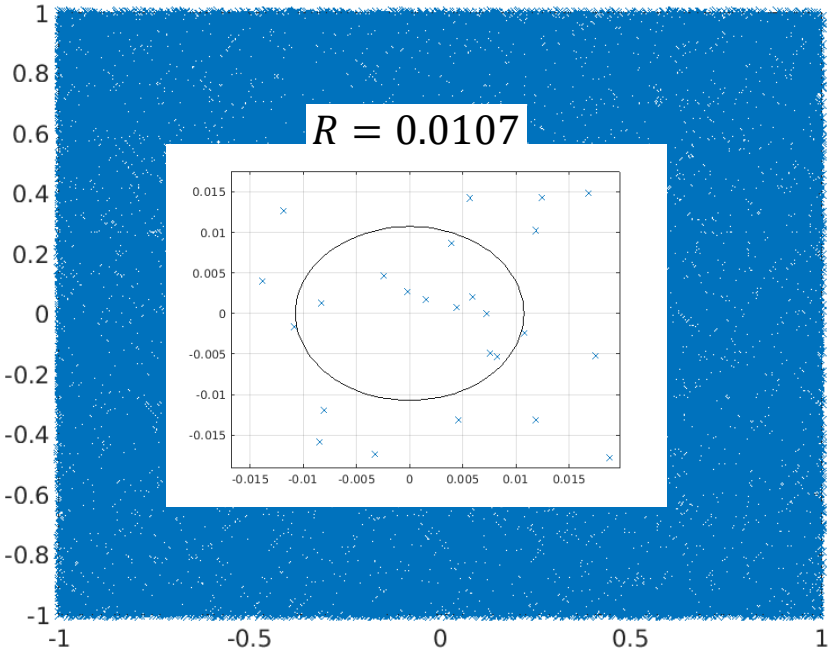


$N = 1.000$

Optimal sampling-based planning: PRM*



$N = 10.000$



$N = 100.000$

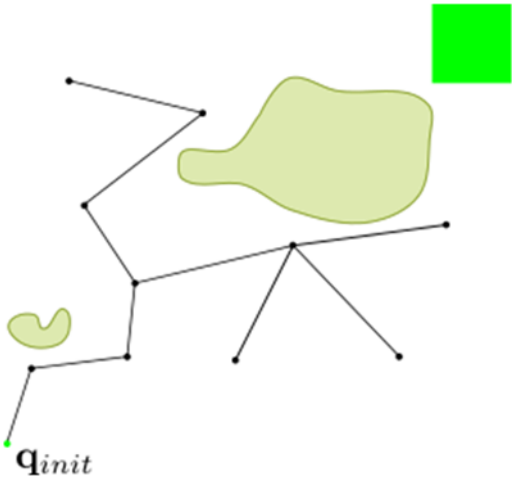
Optimal sampling-based planning: RRG a step towards RRT*

Rapidly exploring Random Graph is an algorithm that builds a connected roadmap, possibly including cycles, in an incremental way (as opposed to the batch way used by PRM).

RRG is similar to RRT as it first attempts to connect the nearest node to the new sample.

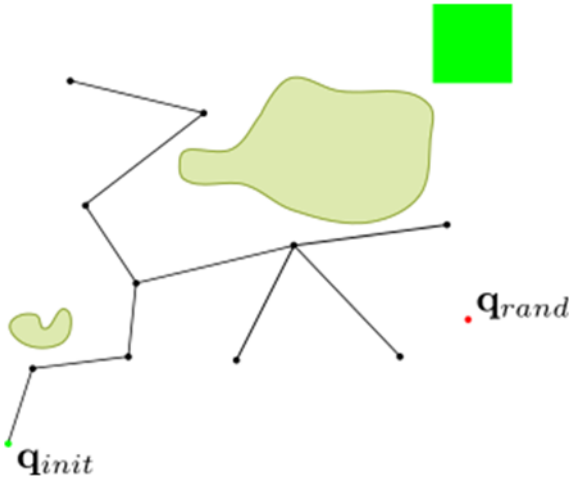
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



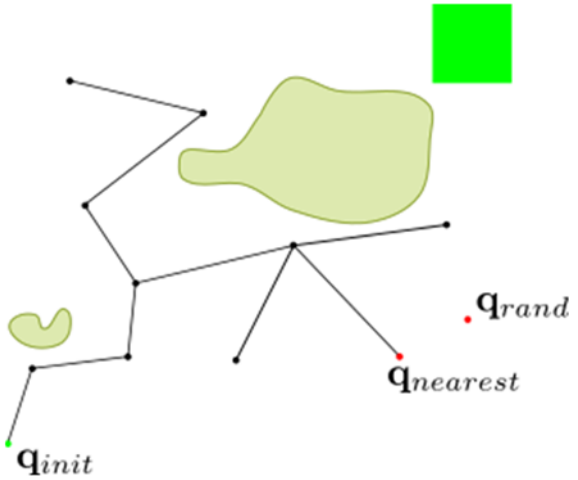
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



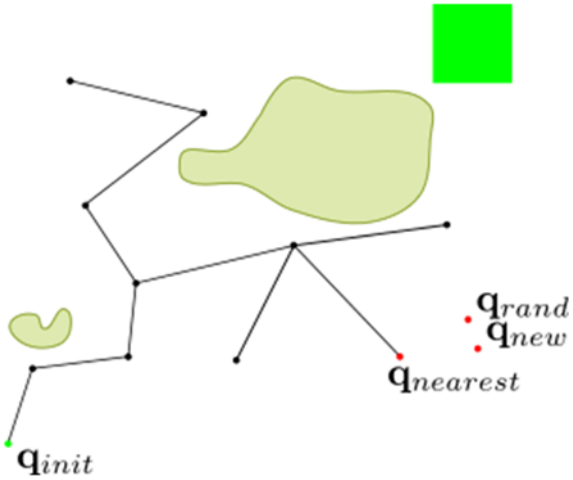
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



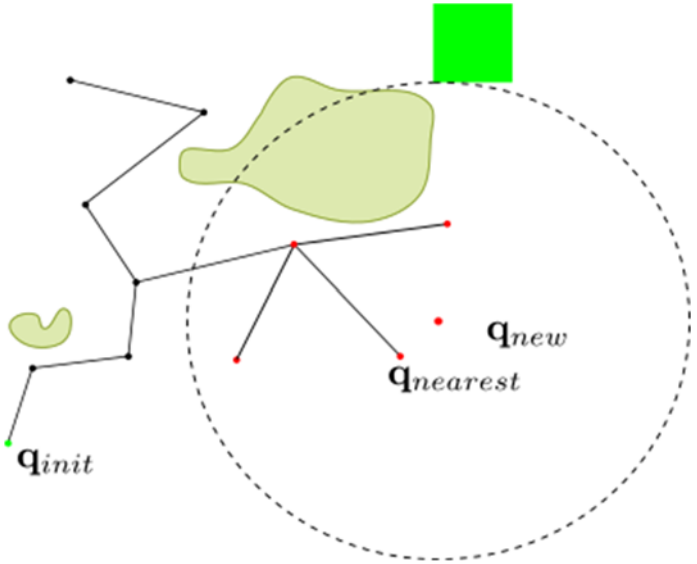
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



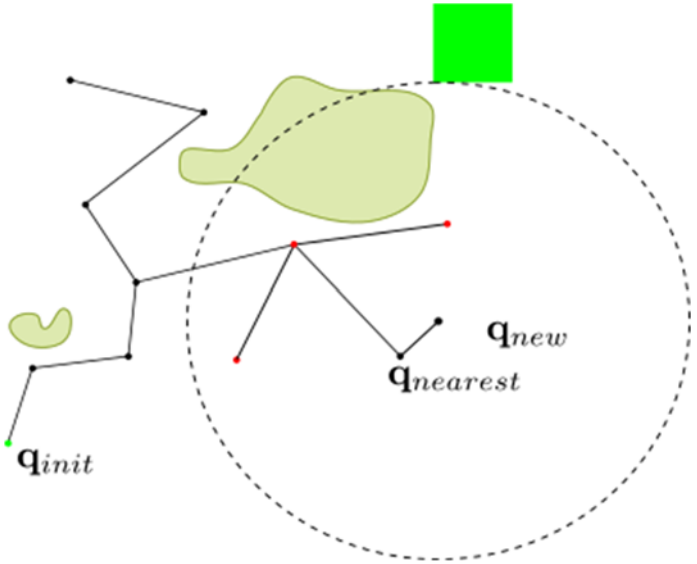
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



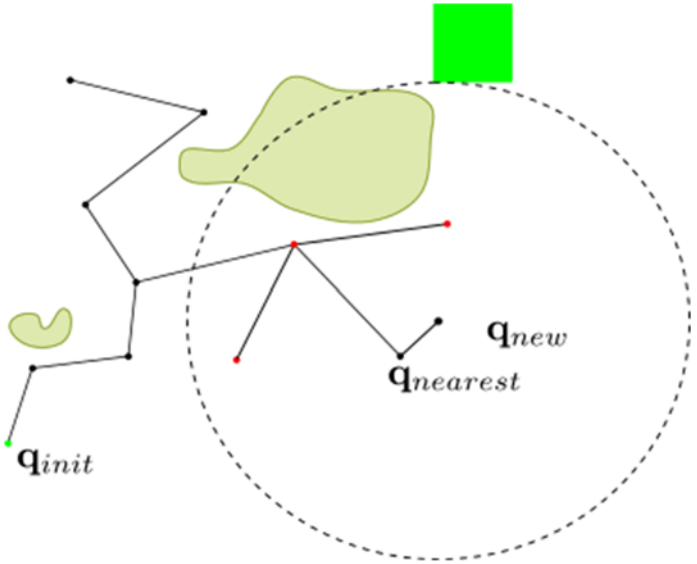
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



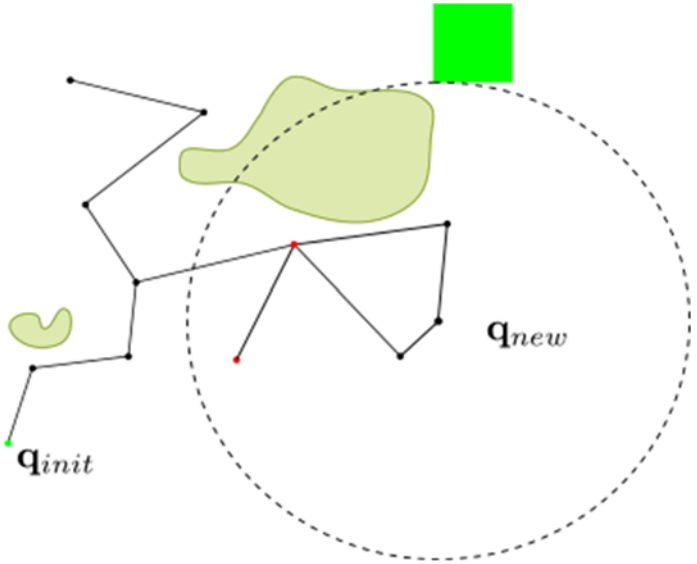
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



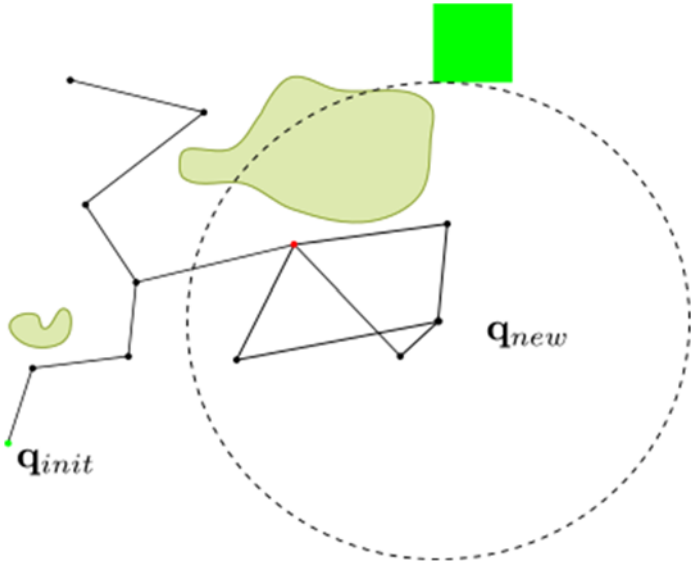
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



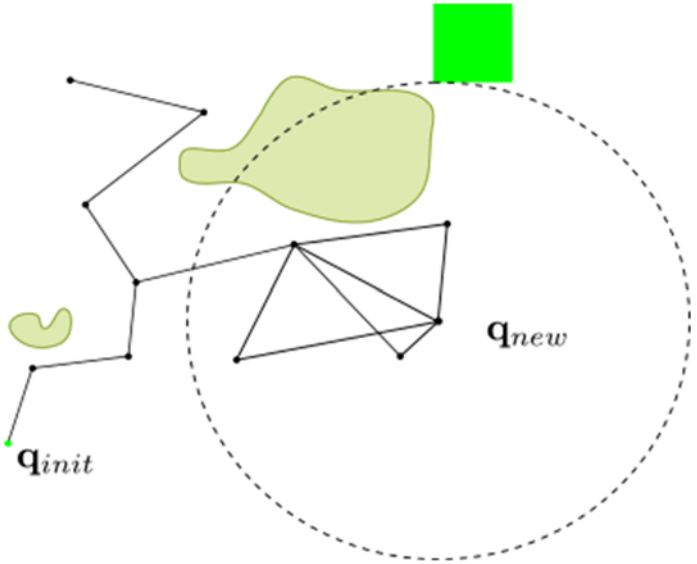
Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



Optimal sampling-based planning: RRG a step towards RRT*

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$



Optimal sampling-based planning: from RRT to RRG

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRG}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      if $CollisionFree\,(\mathbf{q}_{near}, \mathbf{q}_{new})$ **then**
        $E \leftarrow E \cup \{(\mathbf{q}_{near}, \mathbf{q}_{new})\}$;
      end
    end
  end
end
return $G = (V, E)$

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $E \leftarrow E \cup \{(\mathbf{q}_{nearest}, \mathbf{q}_{new})\}$;
  end
end
return $G = (V, E)$

Summarizing RRG has the following peculiarities:

- a new vertex is added to the vertex set V, then connections are attempted from all other vertices in V that are within a ball of radius

$$r(\mathrm{card}(V)) = \min\left\{\gamma_{RRG}\left(\log(\mathrm{card}(V))/\mathrm{card}(V)\right)^{1/d}, \eta\right\}$$

  where

$$\gamma_{RRG} > \gamma^{\star}_{RRG} = 2(1+1/d)^{1/d}\left(\mu(\mathcal{Q}_{free})/\zeta_d\right)^{1/d}$$

- for each successful connection a new edge is added
- for the same sampling sequence, the RRT graph (a directed tree) is a subgraph of the RRG graph (an undirected graph, possibly containing cycles)
- the two graphs share the same vertex set, and the edge set of the RRT graph is a subset of that of the RRG graph

Rapidly exploring Random Tree*

Maintaining a tree structure rather than a graph is not only economical in terms of memory requirements, but may also be advantageous in some applications…

RRT* is obtained by modifying RRG in such a way that formation of cycles is avoided, by removing “redundant” edges.

Since RRT and RRT* graphs are directed trees with the same root and vertex set, and edge sets that are subsets of that of RRG, this amounts to a “rewiring” of the RRT tree, ensuring that vertices are reached through a minimum-cost path.

Before describing the algorithm, we introduce a couple of useful functions...

Parent

Given a tree $G = (V, E)$, the function $Parent: V \to V$ maps a vertex $v \in V$ to the unique vertex $u \in V$ such that $(u, v) \in E$.

If $v_0 \in V$ is the root vertex of G, by definition $Parent(v_0) = v_0$.

Cost

$Cost: V \to R_{\geq 0}$ is a function that maps a vertex $v \in V$ to the cost of the unique path from the root of the tree to v.

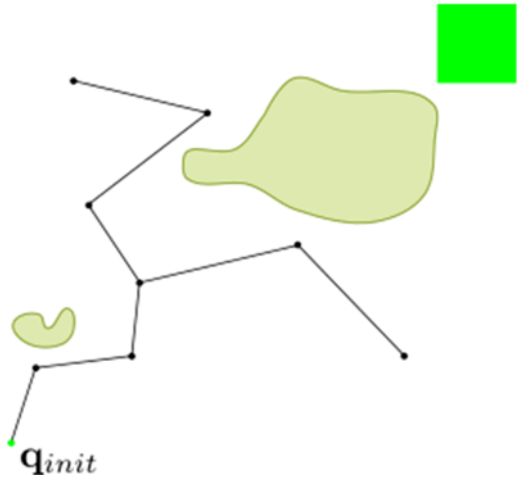
For simplicity, we assume an additive cost function

$$Cost(v) = Cost(Parent(v)) + Cost((Parent(v), v))$$

Rapidly exploring Random Tree*: the algorithm

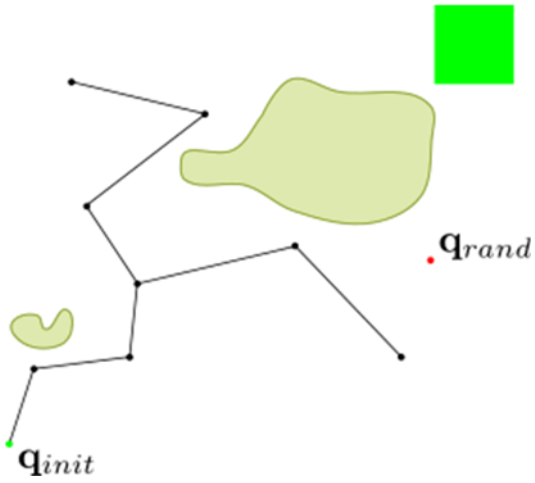
$V \leftarrow \{\mathbf{q}_{init}\};$
$E \leftarrow \emptyset;$
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i;$
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand});$
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand});$
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right);$
    $V \leftarrow V \cup \{\mathbf{q}_{new}\};$
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest};$
    $c_{min} \leftarrow Cost\,(\mathbf{q}_{nearest}) + Cost\,((\mathbf{q}_{nearest}, \mathbf{q}_{new}));$
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      ⋮

Or it can be stopped when the goal region is reached



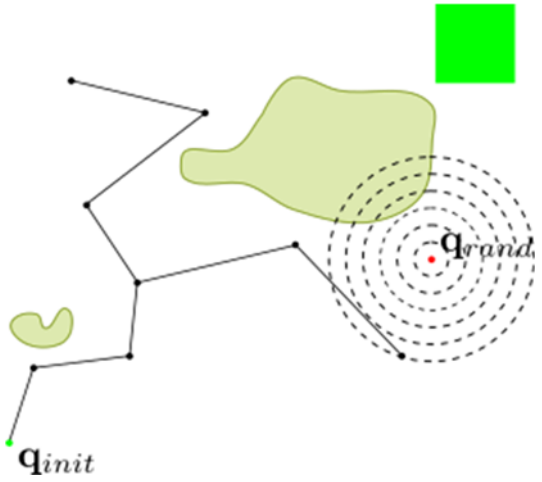
Rapidly exploring Random Tree*: the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;
    $c_{min} \leftarrow Cost\,(\mathbf{q}_{nearest}) + Cost\,((\mathbf{q}_{nearest}, \mathbf{q}_{new}))$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      ⋮



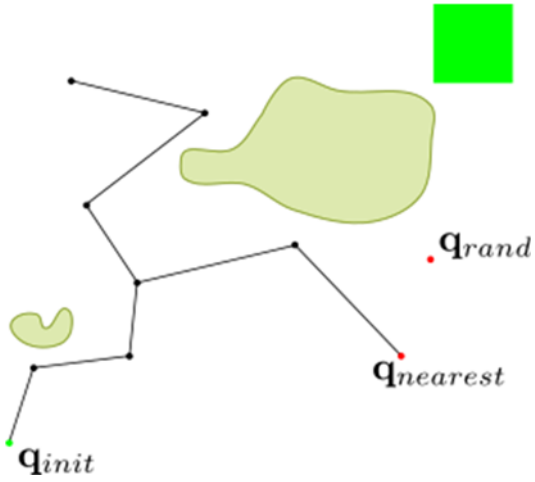
Rapidly exploring Random Tree*: the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;
    $c_{min} \leftarrow Cost\,(\mathbf{q}_{nearest}) + Cost\,((\mathbf{q}_{nearest}, \mathbf{q}_{new}))$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      ⋮



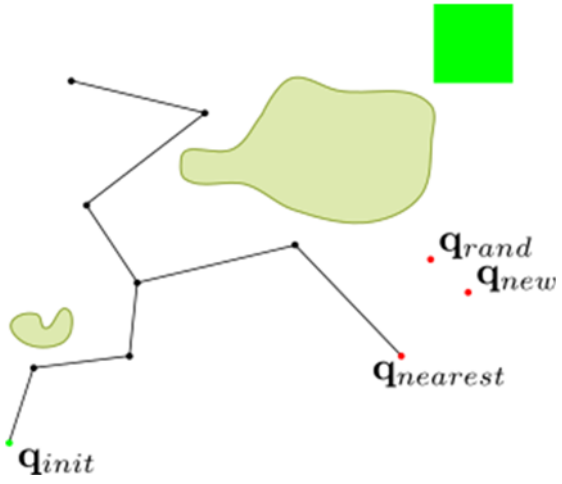
Rapidly exploring Random Tree*: the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;
    $c_{min} \leftarrow Cost\,(\mathbf{q}_{nearest}) + Cost\,((\mathbf{q}_{nearest}, \mathbf{q}_{new}))$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      ⋮



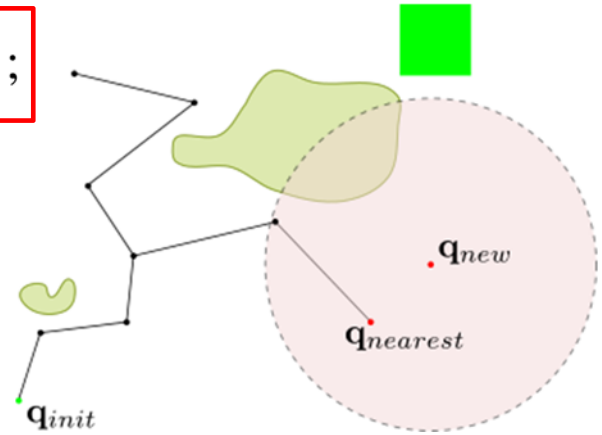
Rapidly exploring Random Tree*: the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;
    $c_{min} \leftarrow Cost\,(\mathbf{q}_{nearest}) + Cost\,((\mathbf{q}_{nearest}, \mathbf{q}_{new}))$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      ⋮



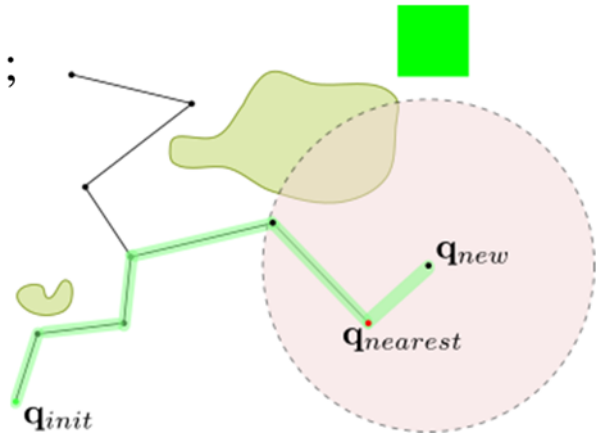
Rapidly exploring Random Tree*: the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;
    $c_{min} \leftarrow Cost\,(\mathbf{q}_{nearest}) + Cost\,((\mathbf{q}_{nearest}, \mathbf{q}_{new}))$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      ⋮



Rapidly exploring Random Tree*: the algorithm

$V \leftarrow \{\mathbf{q}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{q}_{rand} \leftarrow SampleFree_i$;
  $\mathbf{q}_{nearest} \leftarrow Nearest\,(G, \mathbf{q}_{rand})$;
  $\mathbf{q}_{new} \leftarrow Steer\,(\mathbf{q}_{nearest}, \mathbf{q}_{rand})$;
  if $CollisionFree\,(\mathbf{q}_{nearest}, \mathbf{q}_{new})$ **then**
    $Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\,(V)\right)/\mathrm{card}\,(V)\right)^{1/d}, \eta\right\}\right)$;
    $V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;
    $c_{min} \leftarrow Cost\,(\mathbf{q}_{nearest}) + Cost\,((\mathbf{q}_{nearest}, \mathbf{q}_{new}))$;
    foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
      ⋮



Rapidly exploring Random Tree*: the algorithm

$Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\left(V\right)\right)/\mathrm{card}\left(V\right)\right)^{1/d}, \eta\right\}\right)$;

$V \leftarrow V \cup \{\mathbf{q}_{new}\}$;

$\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;

$c_{min} \leftarrow Cost\left(\mathbf{q}_{nearest}\right) + Cost\left(\left(\mathbf{q}_{nearest}, \mathbf{q}_{new}\right)\right)$;

foreach $\mathbf{q}_{near} \in Q_{near}$ **do**

  if $CollisionFree\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right) \wedge$

    $\left(Cost\left(\mathbf{q}_{near}\right) + Cost\left(\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right)\right) < c_{min}\right)$ **then**

      $\mathbf{q}_{min} \leftarrow \mathbf{q}_{near}$;

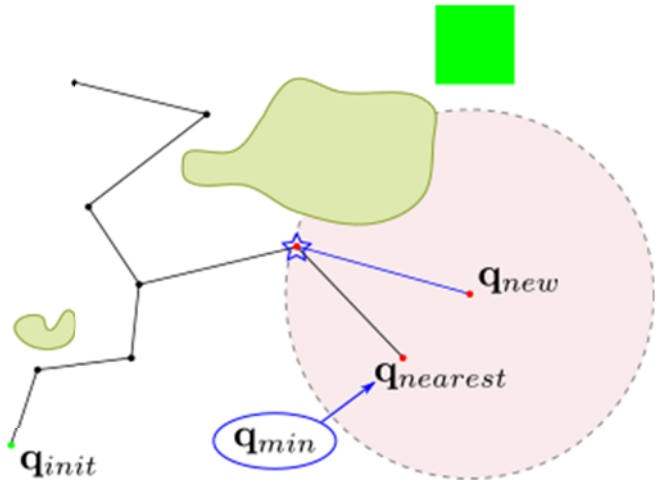
      $c_{min} \leftarrow Cost\left(\mathbf{q}_{near}\right) + Cost\left(\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right)\right)$;

  end

end

$E \leftarrow E \cup \{(\mathbf{q}_{min}, \mathbf{q}_{new})\}$;

Connect along a minimum cost path



Rapidly exploring Random Tree*: the algorithm

$Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\left(V\right)\right)/\mathrm{card}\left(V\right)\right)^{1/d}, \eta\right\}\right)$;

$V \leftarrow V \cup \{\mathbf{q}_{new}\}$;

$\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;

$c_{min} \leftarrow Cost\left(\mathbf{q}_{nearest}\right) + Cost\left(\left(\mathbf{q}_{nearest}, \mathbf{q}_{new}\right)\right)$;

foreach $\mathbf{q}_{near} \in Q_{near}$ **do**

  if $CollisionFree\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right) \wedge$

    $\left(Cost\left(\mathbf{q}_{near}\right) + Cost\left(\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right)\right) < c_{min}\right)$ **then**

      $\mathbf{q}_{min} \leftarrow \mathbf{q}_{near}$;

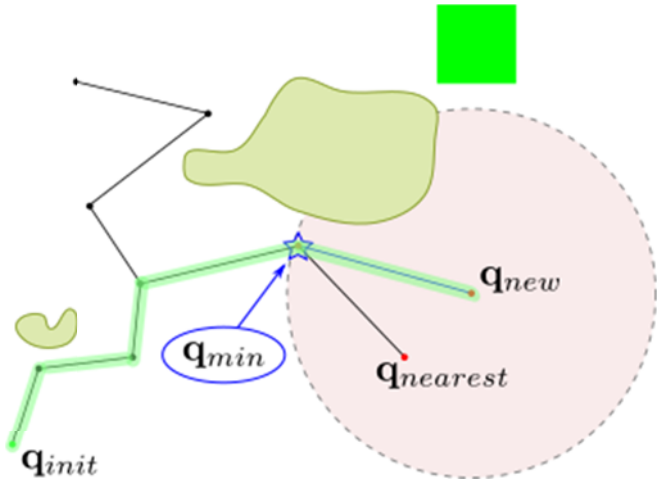
      $c_{min} \leftarrow Cost\left(\mathbf{q}_{near}\right) + Cost\left(\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right)\right)$;

  end

end

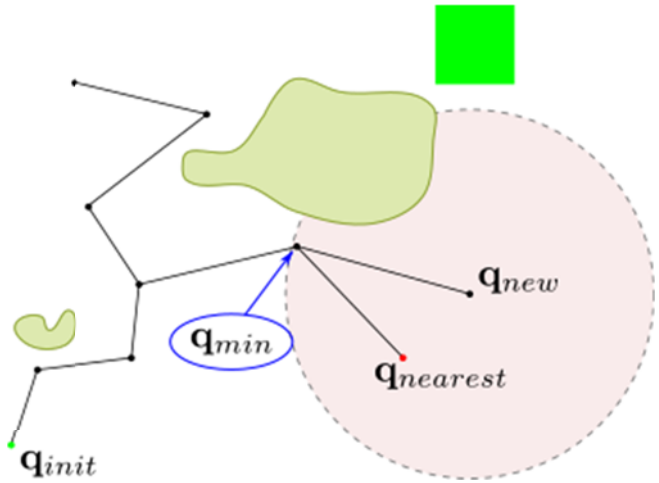
$E \leftarrow E \cup \{(\mathbf{q}_{min}, \mathbf{q}_{new})\}$;

Connect along a minimum cost path



Rapidly exploring Random Tree*: the algorithm

$Q_{near} \leftarrow Near\left(G, \mathbf{q}_{new}, \min\left\{\gamma_{RRT^\star}\left(\log\left(\mathrm{card}\left(V\right)\right)/\mathrm{card}\left(V\right)\right)^{1/d}, \eta\right\}\right)$;
$V \leftarrow V \cup \{\mathbf{q}_{new}\}$;
$\mathbf{q}_{min} \leftarrow \mathbf{q}_{nearest}$;
$c_{min} \leftarrow Cost\left(\mathbf{q}_{nearest}\right) + Cost\left(\left(\mathbf{q}_{nearest}, \mathbf{q}_{new}\right)\right)$;
foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
  if $CollisionFree\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right) \wedge$
   $\left(Cost\left(\mathbf{q}_{near}\right) + Cost\left(\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right)\right) < c_{min}\right)$ **then**
    $\mathbf{q}_{min} \leftarrow \mathbf{q}_{near}$;
    $c_{min} \leftarrow Cost\left(\mathbf{q}_{near}\right) + Cost\left(\left(\mathbf{q}_{near}, \mathbf{q}_{new}\right)\right)$;
  end
end
$E \leftarrow E \cup \{(\mathbf{q}_{min}, \mathbf{q}_{new})\}$;



Rapidly exploring Random Tree*: the algorithm

$\vdots$

$E \leftarrow E \cup \{(\mathbf{q}_{min}, \mathbf{q}_{new})\}$;

foreach $\mathbf{q}_{near} \in Q_{near}$ **do**

  if $CollisionFree(\mathbf{q}_{new}, \mathbf{q}_{near}) \wedge$

    $(Cost(\mathbf{q}_{new}) + Cost((\mathbf{q}_{new}, \mathbf{q}_{near})) < Cost(\mathbf{q}_{near}))$ **then**

      $\mathbf{q}_{parent} \leftarrow Parent(\mathbf{q}_{near})$;

      $E \leftarrow (E \setminus \{(\mathbf{q}_{parent}, \mathbf{q}_{near})\}) \cup \{(\mathbf{q}_{new}, \mathbf{q}_{near})\}$;

  end

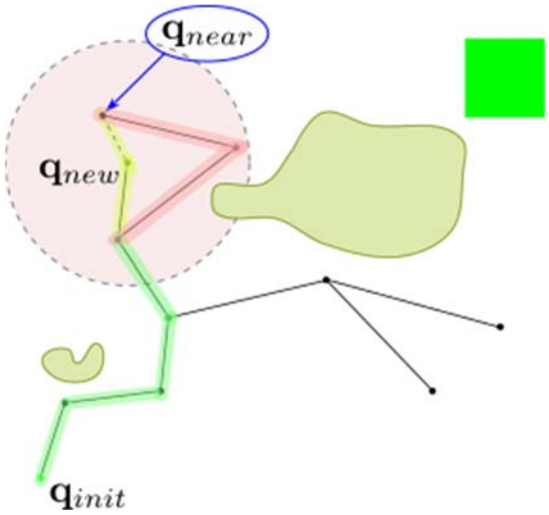
end

end

end

return $G = (V, E)$

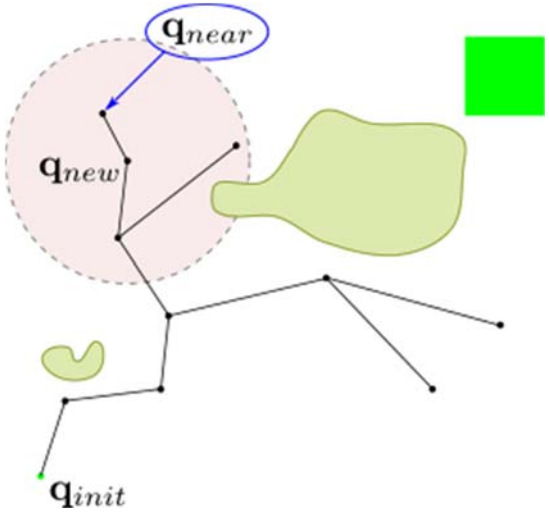
Rewire the tree



Rapidly exploring Random Tree*: the algorithm

$E \leftarrow E \cup \{(\mathbf{q}_{min}, \mathbf{q}_{new})\}$;
foreach $\mathbf{q}_{near} \in Q_{near}$ **do**
  if $CollisionFree(\mathbf{q}_{new}, \mathbf{q}_{near}) \wedge$
   $(Cost(\mathbf{q}_{new}) + Cost((\mathbf{q}_{new}, \mathbf{q}_{near})) < Cost(\mathbf{q}_{near}))$ **then**
    $\mathbf{q}_{parent} \leftarrow Parent(\mathbf{q}_{near})$;
    $E \leftarrow (E \setminus \{(\mathbf{q}_{parent}, \mathbf{q}_{near})\}) \cup \{(\mathbf{q}_{new}, \mathbf{q}_{near})\}$;
  end
end
end
end
return $G = (V, E)$

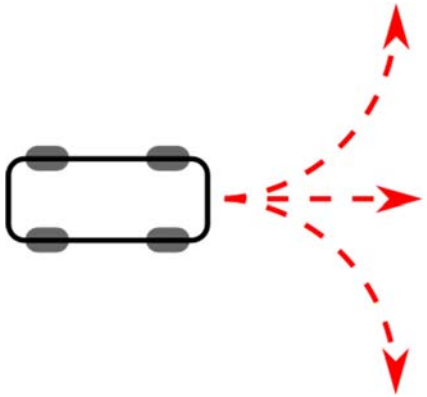
Rewire the tree

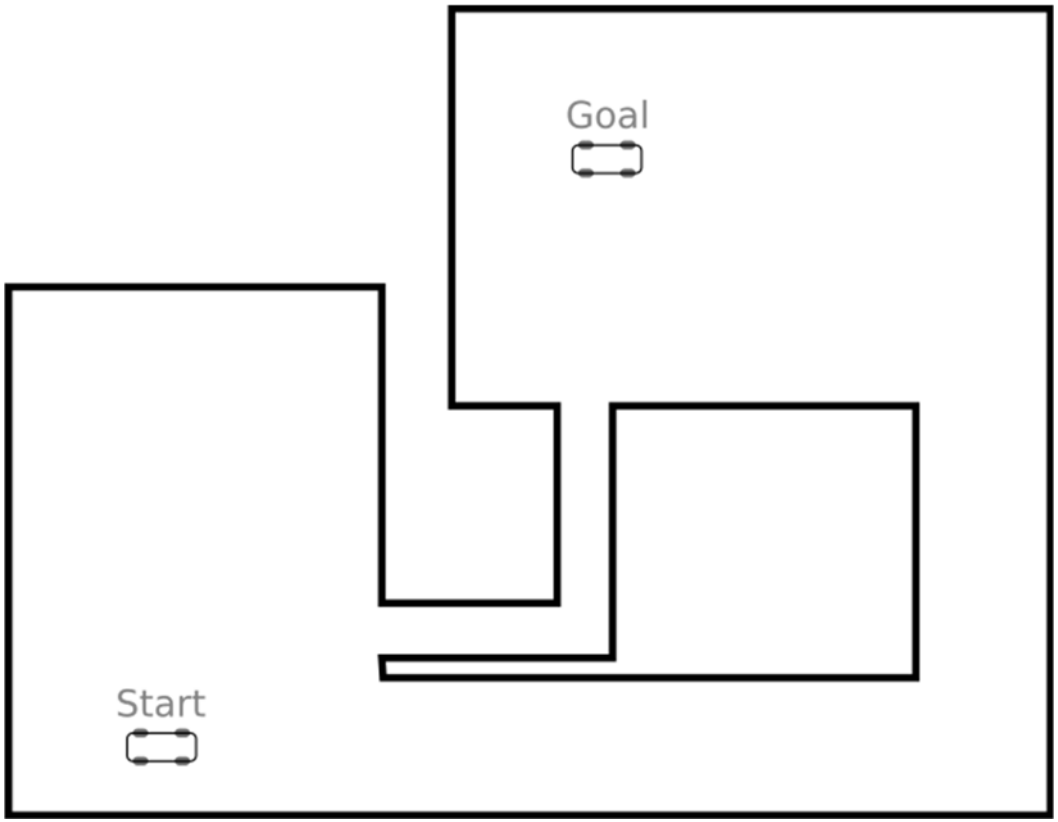


What is kinodynamic planning?

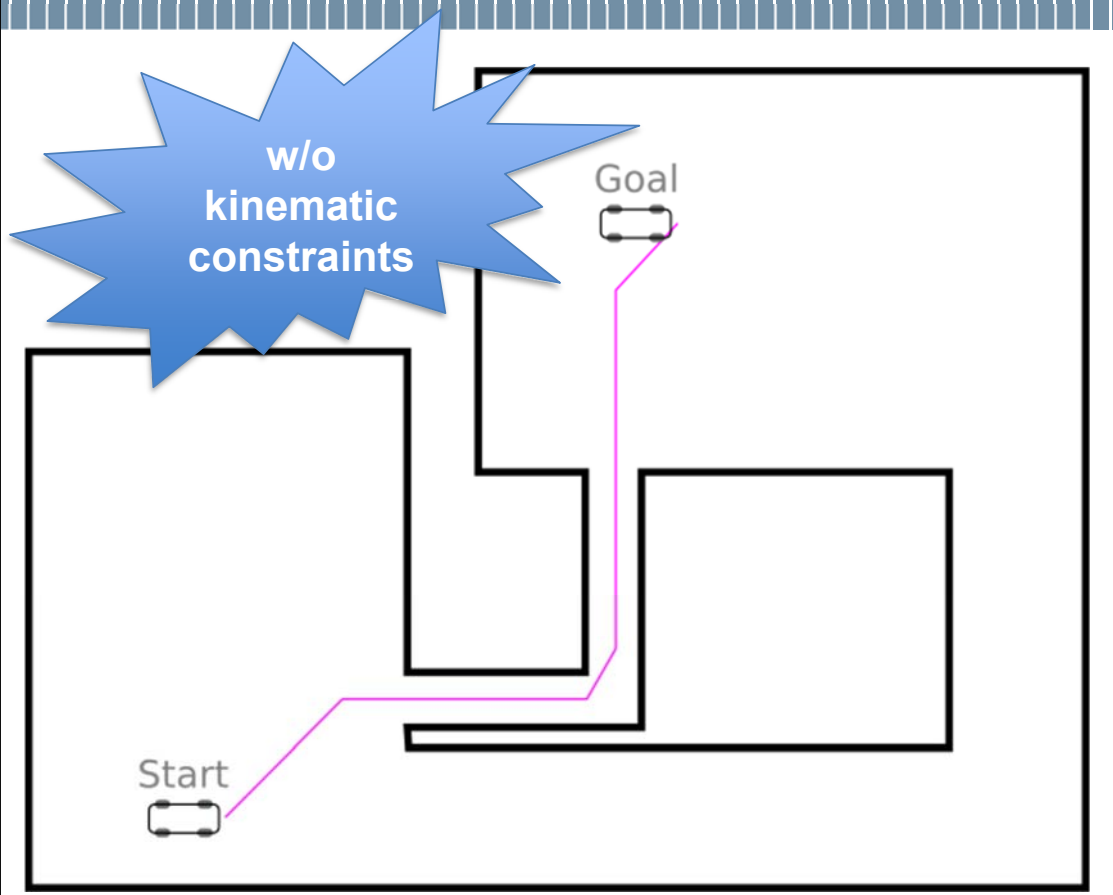
Consider the following example:

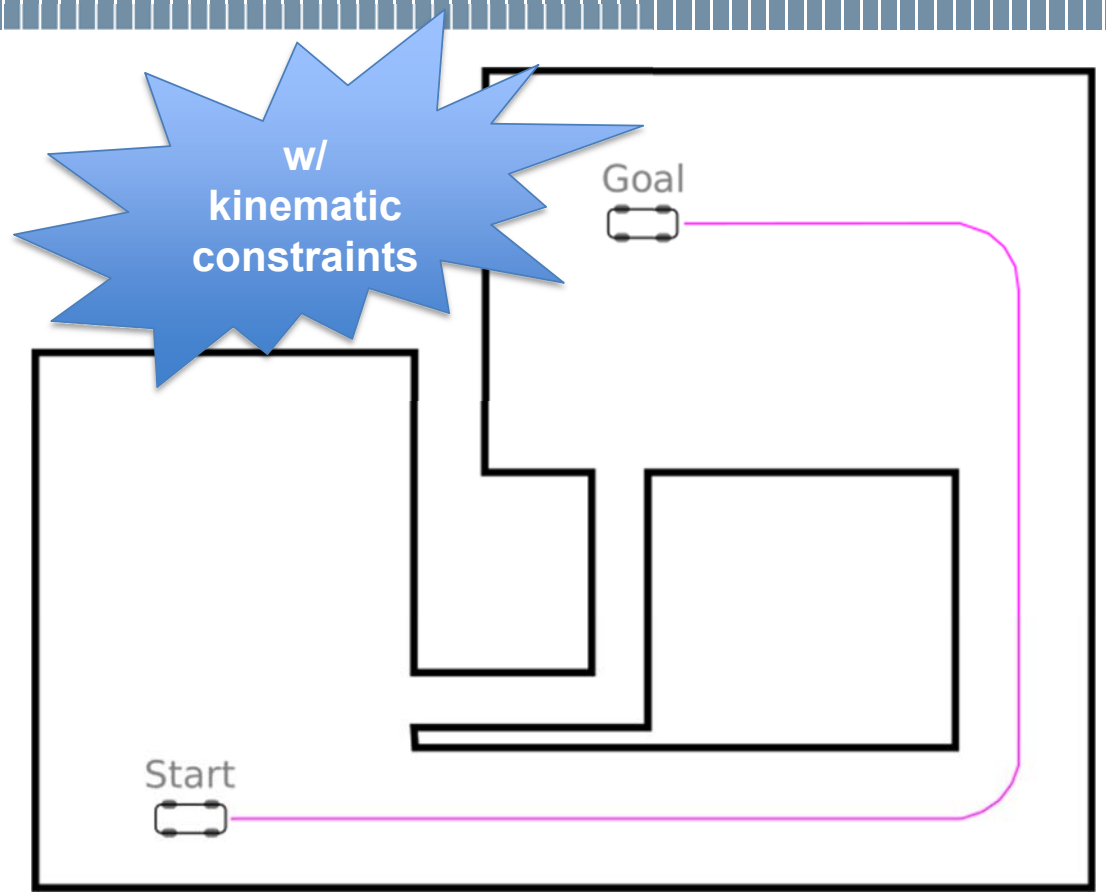
"a vehicle characterized by a limited turning radius and without reverse must plan a trajectory in the following environment"



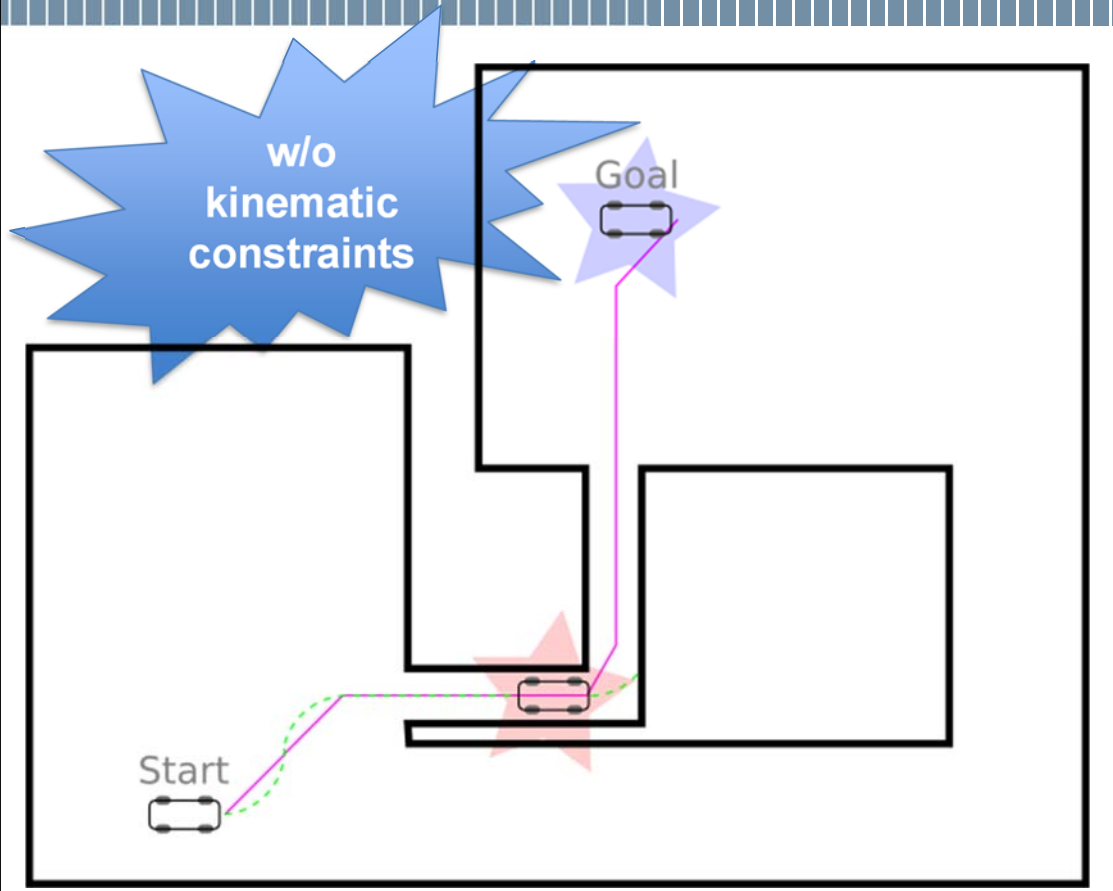


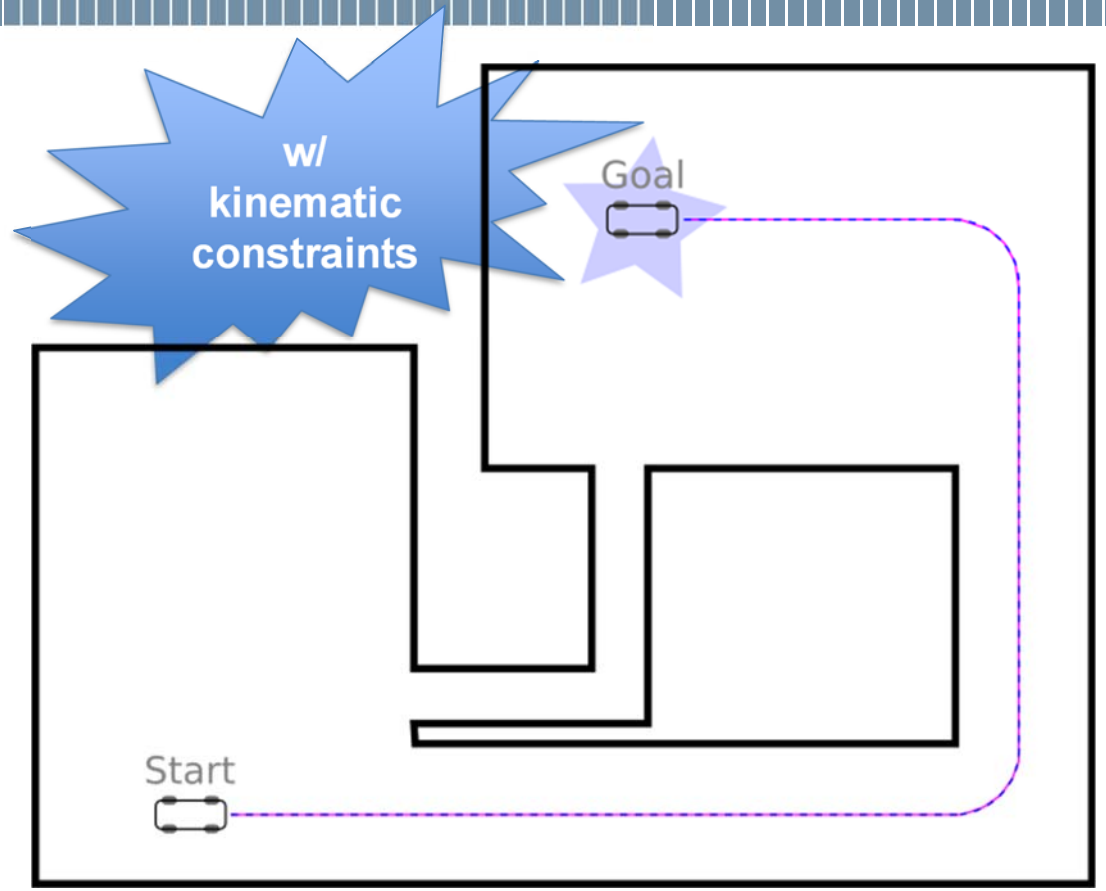
What is kinodynamic planning?





What is kinodynamic planning?





What is kinodynamic planning?

Kinodynamic planning combines:

- the search for a collision-free path;
- the kinematic and/or dynamic constraints that characterize the robot so that the resulting trajectory is feasible.

Kinodynamic constraints can be categorized into three main groups:

- *differential constraints*, representing the robot motion model;
- *nonholonomic constraints*, representing only the kinematic constraints to which the robot is subjected;
- *hard bounds*, any bound on the derivatives of the configuration (velocity, acceleration, jerk, ...).

They are the most common constraints, representing the motion model of the robot.

Let the compact sets $\mathcal{S} \subset \mathbb{R}^d$ and $\mathcal{U} \subset \mathbb{R}^m$, with $d \geq 2$ and $m \geq 1$, represent the state and the control spaces. The dynamic system

$$\dot{\mathbf{s}}(t) = \mathbf{f}(\mathbf{s}(t), \mathbf{u}(t))$$

with $\boldsymbol{s}(t) \in \mathcal{S}$ and $\boldsymbol{u}(t) \in \mathcal{U}$, represents a differential constraint.

The state $\boldsymbol{s}$ usually includes the configuration vector and its first derivative

$$\mathbf{s}(t) = \begin{bmatrix} \mathbf{q}(t) \\ \dot{\mathbf{q}}(t) \end{bmatrix}$$

We can now define a mapping $\kappa : \mathcal{S} \to \mathcal{Q}$ that maps a state to the corresponding configuration.

Using this mapping we can define the *free state space* $\mathcal{S}_{free} = \{\boldsymbol{s} \in \mathcal{S} \mid \kappa(\boldsymbol{s}) \in \mathcal{Q}_{free}\}$.

If there are also hard bounds on the state variables

$$h(\mathbf{s}) \leq 0$$

they can be included in the definition of the free state space $\mathcal{S}_{free} = \{\boldsymbol{s} \in \mathcal{S} \mid \kappa(\boldsymbol{s}) \in \mathcal{Q}_{free} \wedge h(\boldsymbol{s}) \leq 0\}$.

On the other side, in case of actuation bounds the free control space can be defined as $\mathcal{U}_{free} = \{\boldsymbol{u}(t) \in \Omega \subset \mathcal{U}\}$ where Ω is the set of admissible controls.

Finally, a *trajectory* $\sigma(t): [0,T] \rightarrow \mathcal{S}$ of duration T is a function generated by integrating the differential constraints with input $u: [0,T] \rightarrow \mathcal{U}$ and initial state $\sigma(0) = \boldsymbol{s}_0$, $\boldsymbol{s}_0 \in \mathcal{S}_{free}$.

A path planning problem is thus defined by the tuple $(\mathcal{S}_{free}, \mathcal{U}_{free}, \boldsymbol{s}_0, \mathcal{S}_{goal})$.

$\mathcal{S}_{goal} \subset \mathcal{S}_{free}$ is the set of states defining the goal region.

Feasible kinodynamic planning

We can now introduce the *feasible kinodynamic trajectory planning* problem:

Finding a trajectory $\sigma(t) : [0, T] \rightarrow \mathcal{S}$ *from a set of dynamic states and controls such that the trajectory*

- *starts from an initial state* $\sigma(0) = \boldsymbol{s}_0$;
- *reaches the goal region,* $\sigma(T) \in \mathcal{S}_{goal}$;
- *avoids collisions with obstacles and satisfies bounds on the state variables,* $\sigma(t) \in \mathcal{S}_{free}$;
- *fulfills the actuation bounds,* $\boldsymbol{u}(t) \in \mathcal{U}_{free}$.

In most applications, the solution to the trajectory planning problem has to satisfy desired properties, i.e., minimizing energy consumption, time, length, maximizing safety,..

We can now introduce the *optimal kinodynamic trajectory planning* problem:

Finding a feasible trajectory $\sigma^{\star}(t) : [0, T^{\star}] \rightarrow \mathcal{S}$ *from a set of dynamic states and controls that minimizes an objective function in the form*

$$\min_{\mathbf{u}(t), T} \int_0^T g(\mathbf{s}(t), \mathbf{u}(t)) \, \mathrm{d}t$$

where g *is an arbitrary function, depending on the specific motion planning problem.*

What should we change to our planning algorithms to make them suitable to solve the optimal kinodynamic planning problem?

Steering is the function that computes a trajectory starting from an initial state $\boldsymbol{s}_1$ and ending at a final state $\boldsymbol{s}_2$ (*exact steering*).

The *exact optimal steering* problem can be formulated as:

Given a cost function assigning a non-zero cost to each non-trivial feasible trajectory, the state based optimal steering is the problem of finding a feasible trajectory $\sigma^\star(t) : [0, T^\star] \rightarrow \mathcal{S}$, *such that the trajectory*

- *starts from an initial state* $\sigma(0) = \boldsymbol{s}_1$;
- *reaches the final state* $\sigma(T^\star) = \boldsymbol{s}_2$;
- *fulfills the actuation bounds,* $\boldsymbol{u}(t) \in \mathcal{U}_{free}$;
- *minimizes the cost (objective) function.*

Let's consider an example.

The steering problem can be formulated as an optimal control problem, it is usually called *two point boundary value problem*.

Let's see how we have to modify RRT* in order to take into account the exact steering function.

$$\min_{a(t),\omega(t),\tau} \int_0^\tau \left(1 + \begin{bmatrix}\omega(t) & a(t)\end{bmatrix} R \begin{bmatrix}\omega(t) & a(t)\end{bmatrix}^T\right) \mathrm{d}t$$

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \omega$$
$$\dot{v} = a$$

$$\omega_{min} \leq \omega(t) \leq \omega_{max} \quad t \in [0,\tau]$$
$$a_{min} \leq a(t) \leq a_{max} \quad t \in [0,\tau]$$

$$x(0) = x_0,\ y(0) = y_0,\ \theta(0) = \theta_0,\ v(0) = v_0$$
$$x(\tau) = x_f,\ y(\tau) = y_f,\ \theta(\tau) = \theta_f,\ v(\tau) = v_f$$

Let's consider a synthetic version of the planner.

$V \leftarrow \{\mathbf{s}_{init}\}$;
$E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $\mathbf{s}_{rand} \leftarrow SampleFree_i$;
  $V_{near} \leftarrow NearNodes\,(V, \mathbf{s}_{rand})$;
  $(\mathbf{s}_{best}, e_{best}) \leftarrow FindBestParent\,(V_{near}, \mathbf{s}_{rand})$;
  if $\mathbf{s}_{best} \neq \mathbf{0}$ **then**
    $(V_{i+1}, E_{i+1}) \leftarrow UpdateConnections\,(V_{near}, \mathbf{s}_{rand}, \mathbf{s}_{best}, e_{best})$;
  end
end
return $G = (V, E)$

These two procedures have to be modified to consider the exact steering

$$V_{near} = \{\boldsymbol{s} \in \mathcal{S} : Cost(\sigma_{s_{rand},s}) \leq d_{th} \vee Cost(\sigma_{s,s_{rand}}) \leq d_{th}\}$$

FindBestParent is defined as

$c_{best} \leftarrow \infty$;
$E \leftarrow \emptyset$;
for $\mathbf{s}_{near} \in V_{near}$ **do**
  $\sigma_{near} = Steer(\mathbf{s}_{near}, \mathbf{s}_{rand})$;
  if $CollisionFree(\mathbf{s}_{near}, \mathbf{s}_{rand})$ **then**
    if $Cost(\mathbf{s}_{near}) + Cost(\sigma_{near}) < c_{best}$ **then**
      $c_{best} = Cost(\mathbf{s}_{near}) + Cost(\sigma_{near})$;
      $\mathbf{s}_{best} = \mathbf{s}_{near}$;
      $e_{best} = (\mathbf{s}_{rand}, \mathbf{s}_{best})$;
    end
  end
end
return $\mathbf{s}_{best}, e_{best}$

UpdateConnections is defined as

$V \leftarrow V \cup \mathbf{s}_{rand}$;
$E \leftarrow E \cup (\mathbf{s}_{best}, \mathbf{s}_{rand})$;
for $\mathbf{s}_{near} \in V_{near} \setminus \mathbf{s}_{best}$ **do**
  $\sigma_{near} = Steer(\mathbf{s}_{rand}, \mathbf{s}_{near})$;
  if $CollisionFree(\mathbf{s}_{rand}, \mathbf{s}_{near})$ **then**
    if $Cost(\mathbf{s}_{rand}) + Cost(\sigma_{near}) < Cost(\mathbf{s}_{near})$ **then**
      $\mathbf{s}_{parent} \leftarrow Parent(\mathbf{s}_{near})$;
      $E \leftarrow E \setminus (\mathbf{s}_{parent}, \mathbf{s}_{near})$;
      $E \leftarrow E \cup (\mathbf{s}_{rand}, \mathbf{s}_{near})$;
    end
  end
end
return V, E

Kinodynamic RRT*: an example

Consider a 4D state space (x, y, θ, v) planning problem

$$\dot{x}(t) = v(t)\cos\theta(t)$$
$$\dot{y}(t) = v(t)\sin\theta(t)$$
$$\dot{\theta}(t) = \omega(t)$$
$$\dot{v}(t) = a(t)$$

$$J(\mathbf{u}, \tau) = \int_0^{\tau} \left[1 + \mathbf{u}(t)^T R\mathbf{u}(t)\right] dt$$