MOTION PLANNING

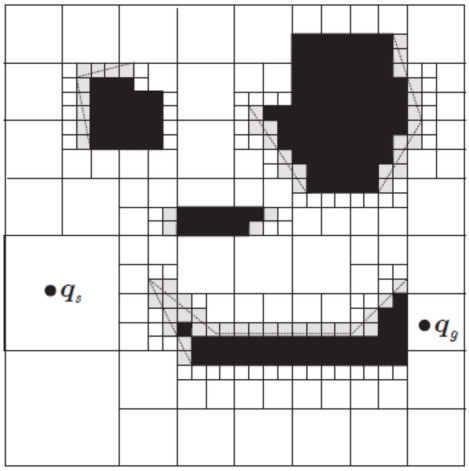
viernes, 15 de noviembre de 2019 11:05 a. m.

- Two main methods:
  - Off-line planning: the workspace geometry is known in advance
  - On-line planning: the workspace geometry is discovered by the robot
- Canonical Problem:
  - Robot B moving a in a workspace
  - B is free-flying in the configuration space, i.e. no kinematic constraints
  - Obstacles $O_1, \dots, O_p$
  - Given a start and final conguration, q_s and q_g, of the robot B in C, plan a **path** that connects q_s and q_g and is **safe**, i.e. it is contained in the free configuration space C_f.
    - $C_f = C - \cup_{i=1}^{p} CO_i$
  - Planning in the workspace is unfeasible because the robot is an infinity of points and the problem becomes untractable, so the planning has to be done in the configuration space where the robot is represented by a point. However we have to accept the added complication that the obstacles are warped in the transformation from the workspace to the configuration space.
  - Extensions:
    - Moving obstacles: contact with the obstacles has to be allowed in order to manipulate them.
- Motion Planning Methods:
  - **All** work in the configuration space
  - **Most** need preliminary computation of the C-obstacles, which is highly expensive because we have to solve the inverse kinematics problem for an infinity of points (the obstacle) and find the region that they represent in C.
  - Computation of C-obstacles:
    - Exact:
      - Uses algebraic models of the obstacles.
      - Rarely used
    - Approximate:
      - Uses a discretization of the configuration space and check for collisions in them.
      - This can be done very efficiently.
  - Classification:
    - **Roadmap Methods**:
      - **Retraction**:
        - It's an **off-line** method, i.e. the geometry is known in advance. Assume $C = R^2$ and C_f a **polygonal** limited subset (its boundary is made of line segments)
        - Clearance of q in C_f
          - $\gamma(q) = \min_{s \in \partial C_f} \|q - s\|$
        - Neighbors of q
          - $N(q) = \{s \in \partial C_f : \|q - s\| = \gamma(q)\}$
        - Generalized Voronoi diagram:
          - $VC_f = \{q \in C_f : card\, N(q) > 1\}$
          - The idea is to create a **roadmap** that maximizes the safety given the closest obstacle points. Depending on the situation we get different kinds of **elementary arcs** in the roadmap
            - Edge-edge: rectilinear arc

        - Vertex-vertex: rectilinear map
        - Vertex-edge: parabolic arc
    - ◇ The result can be seen as a graph.
- ◆ To connect any q to $VG(\gamma)$ we use **retraction**. That is, from q follow $\nabla\gamma$ up to the first intersection $r(q)$ with $VG(C_f)$
    - ◇ $r(\cdot)$ preserves connectivity of C_f so a safe path exists between q_s and q_g iff a path exists on $VG(C_f)$ between $r(q_s)$ and $r(q_g)$
- ◆ **Algorithm**: slides
- ◆ The retraction method is **complete**. Finds a solution if there is one and reports failure if there is not.
- ◆ Complexity: if C_f has v vertices, then $VG(C_f)$ has $O(v)$ arcs, then the overall complexity is $O(v \log v)$

□ **Cell Decompostion**:

- ◆ It's an **off-line** method, i.e. the geometry is known in advance.
- ◆ The idea is to decompose C_f in regions such that it's easy to compute a safe path between two configurations in:
    - ◇ The same cell
    - ◇ Adjacent cells
- ◆ Once the decompostition is done it is only necessary to find a sequence of cells such that q_s is in the initial cell and q_f is in the final one. Different methods are obtained based on the type of decomposition chosen.
- ◆ Exact decomposition:
    - ◇ Assume $C = R^2$ and C_f a **polygonal** limited subset (its boundary is made of line segments)
    - ◇ To do this we decompose C_f into convex polygons. Convexity guarantees both of the previous requirements.
    - ◇ Algorithm: slides
- ◆ Approximate decomposition
    - ◇ Assume $C = R^2$ and C_f a **polygonal** limited subset (its boundary is made of line segments)
    - ◇ We decompose the C_f using the same shape. E.g. squares.
    - ◇ As in exact decomposition, convexity guarantees easy planning within cells and between cells.



- ◇ Algorithm: slides

▪ **Probabilistic Methods**: State of the art motion planners

□ A type of **sample-based** methods. The idea is to build a roadmap of C by repeating:

- ◆ Extract a **sample** q of C.

- Use direct kinematics to compute the **volume** $B(q)$.
- Check collision between $B(q)$ and the workspace obstacles $O_1, \dots, O_p$.
- If $q \in C_f$, **add** q to the roadmap; else, **discard** it.

□ In contrast with the previous methods, we do not assume that the obstacles are given.

□ Preliminary computations of CO_i is completely **avoided**.

□ The sampling can be done according to different criteria. In general, **Randomized sampling outperforms deterministic**.

□ **PRM** (Probabilistic Roadmap)

- Algorithm:
  - Extract a **sample** q of C with **uniform probability distribution**.
  - Coompute $B(q)$ and check for collision
  - If $q \in C_f$, ***a*dd** q to the roadmap; else, **discard** it.
  - Search the PRM for "sufficiently near" configurations q_{near}
  - Connect q to q_{near} with a **free local path**.
- The generation of a free path is delegated to a local planner.
- The metric chosen in C plays a big role in defining which configurations are "near" to others.
- The algorithm is probabilistically complete, i.e. the probability of finding a solution approaches 1 as time goes to infinity. However if there is no solution the algorithm will not say so.
- The main advantage of this algorithms is that they are **extremely fast**.
- It's a **multiple-query** algorithm. New problems enhance the roadmap.

□ **RRT** (Rapidly-exploring Random Tree)

- Algorithm:
  - Root the tree T_s at q_s.
  - Generate q_{rand} in C with **uniform probability distribution**.
  - Search tree for the **nearest** configuration q_{near}.
  - Choose q_{new} at distance δ from q_{near} in the direction of q_{rand}
  - Check for collision q_{new} and the segment from q_{near} to q_{new}.
  - If no collision, add q_{new} to T_s.
- The metric chosen in C plays a big role in defining which configurations are "near" to others.
- The expansion is biased towards unexplored regions because they have higher probability of sampling.
- Problem:
  - There is no bias that drives the expansion towards the goal. Notice that the algorithm does not include the goal at all.
  - To solve it, the most used approach is to make the search bidirectional by growing two trees. One rooted at q_s and the other at q_g.
  - Another approach is to set $q_{rand} = q_g$ so that the expansion step is greedy and aimed towards the goal. In this case we run in to the classic reinforcement learning problem of exploration vs exploitation. One popular choice is ϵ-greedy.
- It is a **single-query** algorithm because the tree is rooted at the starting configuration so every new problem demands the construction of a new tree.
- Extension to Non-holonomic Robots:
  - For this kind of robots, linear paths in C as the ones used by the local planners that connect q_{near} to q_{rand} are **not admissible** in general.

- ◇ One possible solution is to use motion primitives, i.e. a set of admissible local paths produced by a specific choice of velocity inputs.
  - For example: Dubins car
    - $v = \bar{v}$
    - $\omega = [-\bar{\omega}, 0, \bar{\omega}]$

- **Artificial Potential Field Methods**
  - □ **On-line** planning. Planning is done using partial workspace information collected during motion. Wokspace information is integrated using a **sense-plan-move paradigm**.
  - □ The idea is to build a **potential field** in the configuration space C so that the robot is attracted by the goal q_g and is repelled by the C-obstacles. The total potential U is the sum of this attraction and repulsion. The negative gradient of that potential $-\nabla U(q)$ indicates the **most promising local direction of motion**.
  - □ **Attractive potential**:
    - Objective: drive the robot to the goal
    - Possibilities: The condition is that both potential fields are zero when the configuration is the goal configuration.
      - ◇ Paraboloidal
        - Let
          - $e = q_g - q$
          - $k_a > 0$
        - Then the potential is
          - $U_a(q) = \frac{1}{2} k_a e^T(q) e(q) = \frac{1}{2} k_a \|(q)\|^2$
        - The resulting force is **linear**.
          - $f_a = -\nabla U(q) = k_a e(q)$
        - Asymptotically stable. Behaves better close to the goal.
      - ◇ Conical
        - Potential
          - $U_a(q) = k_a \|(q)\|$
        - The resulting force is **constant**
          - $f_a = -\nabla U(q) = k_a \frac{e(q)}{\|(q)\|}$
        - Not asymptotically stable. Behaves better when the distance to the goal is big because attraction is constant.
    - A good solution is to combine the two profiles
      - ◇ $U_a(q) = \begin{cases} \frac{1}{2} k_a \|(q)\|^2 & if \|(q)\| \leq \rho \\ k_b \|(q)\| & if \|(q)\| > \rho \end{cases}$
  - □ Repulsive potential:
    - Objective: drive the robot away from CO
    - We assume that the the CO has been **partitioned** in advance in **convex components**.
    - For each obstacle we define a repulsive field.
      - ◇ $U_{r,i}(q) = \begin{cases} \frac{k_{r,i}}{\gamma} \left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}} \right)^{\gamma} & if\ \eta_i(q) \leq \eta_{0,i} \\ 0 & if \eta_i(q) > \eta_{0,i} \end{cases}$
    - Where
      - ◇ $\eta_{0,i}$ is the **range of influence** of the obstacle CO_i
      - ◇ $\eta_i(q)$ is the **clearance**

- The choice of γ defines the steepness of the field around the obstacle.
- Resulting force:
  - $-\nabla_q U_{r,i} = -\frac{k_{r,i}}{\gamma}\gamma\left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}}\right)^{\gamma-1}\left(-\frac{1}{\eta_i(q)^2}\right)\nabla_q\eta_i$
  - $-\nabla_q U_{r,i} = \frac{k_{r,i}}{\eta_i(q)^2}\left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}}\right)^{\gamma-1}\nabla_q\eta_i$
  - It is **orthogonal to the equipotential contour** passing through q and points away from the obstacle
  - It is **continuous** thanks to the convex decomposition of the CO
- The total repulsive potential of CO
  - $U_r(q) = \sum_{i=1}^{p} U_{r,i}(q)$

□ Superposition:
- $U_t(q) = U_a(q) + U_r(q)$

□ Force fields:
- $f_t(q) = -\nabla U_t(q) = f_a + \sum_{i=1}^{p} f_{r,i}$

□ Planning techniques:

a) Consider f_t as generalized forces:
   - $\tau = f_t(q)$
   - The effect on the robot is filtered by its dynamics
   - Velocities are scaled
   - Full dynamic model of robot
   - For asymptotic stability a damping factor is needed

b) Consider f_t as generalized accelerations:
   - $\ddot{q} = f_t(q)$
   - The effect on the robot is independent on its dynamics
   - Forces are scaled
   - Robot as double integrator
   - For asymptotic stability a damping factor is needed

c) Consider f_t as generalized velocities:
   - $\dot{q} = f_t(q)$
   - The effect on the robot is independent on its dynamics
   - Forces are scaled
   - Robot as single integrator
   - Already asymptotically stable

- Off-line planning:
  - Paths are generated in C by integration of the dynamic model by choosing one of the previous planning techniques.
  - The algorithm is **gradient descent**
- On-line planning:
  - Use the planning techniques to provide the control inputs for the robots.
  - It's a **feedback** scheme.

□ Problem: when combining the attractive and repulsive fields we might get a local minimum in front of the obstacles that might trap the robot.
- All APF will have some local minimum when created. Which means that this method is **not complete**.
- Workarounds exist but they might not be necessary because APF are mainly used for on-line planning and in these situations completeness is

not really necessary.

- ◇ Best-first algorithm:
  i. Build a discretized representation of C_f using a regular grid and associate to each free cell the value of the field U_t there.
  ii. Build a tree T rooted at q_s. At each iteration, select the leaf of T with the minimum value of U_t and add as children its adjacent free cells that are not in T.
  iii. Planning stops when q_g is reached or no further cells can be added to T.
  iv. In case of success, build a solution path by tracing back the path from q_s to q_g.
  - ▸ This does not get stuck in a local minimum because when we move to a child node of the tree we are not demanding that it is APF is lower than the current node so it will keep searching until it reaches the goal.
- ◇ Navigation functions:
  - ▸ Paths generated by the best-first algorithm are not very efficient.
  - ▸ The idea is to build a potential function that has no local minima. These are called navigation functions.
  - ▸ Lets consider what happens to the APF when the C-obstacles are spherical.
    - – In the direction that joins the goal with the center of the CO there will be a local minimum in behind the sphere where the attractive and repulsive fields balance eachother.
    - – However, in the direction tangent to that line the point is not a local minimum but a local maximum in that direction.
    - – This means that a spherical C-obstacle produces behind it a **saddle point**, which is not a problem for the planning because the robot will not be trapped there.
  - ▸ To build the APF we need to map the CO_i to a collection of spheres via a diffeomorphism, build a potential in the transformed space and map it back to the configuration space.
  - ▸ The C-obstacles must be **star-shaped**. This means that there is a central point from which you can reach all other points in the set with a line segment.
  - ▸ A possible generalization of this method is to use **harmonic functions** (solution to Laplace's equation)
  - ▸ For this method you need to have **complete knowledge** of the envirionment and its only suitable for **off-line** planning.
  - ▸ One easy alternative is to build a **numerical navigation function**.
    - – Assign zero potential to the goal and start increasing the potential to the adjacent cells (**wavefront expansion**)
    - – Then simply use gradient descent from the starting cell.
- ◇ Vortex fields:
  - ▸ As an alternative to navigation functions we assign directly a force field (instead of a potential)
  - ▸ The idea is to replace the repulsive potential (that is responsible for local minima) with an action that forces the

robot to go around the C-obstacle.

- **Vortex field**

$$- \quad f_v = \pm \begin{pmatrix} \frac{\partial U_{r,i}}{\partial y} \\ -\frac{\partial U_{r,i}}{\partial x} \end{pmatrix}$$

- We get a vector that is **tangent** to the equipotential contours
- The vortex sense should be chosen depending on the entrance of the robot to the area of influence of the robot.
- Vortex relaxation must be done to avoid infinite orbiting around the obstacle
- Complete knowledge of the configuration space is not required so it is suitable for **on-line** planning.

□ Artificial Potential for wheeled robots.

- Since WMRs are described at the kinematic models, then the APF for this robots are used at the velocity level.
- However WMRs violate one of the assumptions of APF because they are subject to nonholonomic constraits, so they violate the free-flying assumption.
- Then, the artificial force f_t cannot be directly imposed as a generalized velocity $\dot{q}$. A possible approach is to use the f_t to generate a feasible $\dot{q}$ via pseudoinversion.
- Kinematic model
  - $\dot{q} = G(q)u$
- To find the needed inputs we use a least-squares solution.
  - $u = G^{\dagger}(q)\dot{q}_d = G^{\dagger}(q)f_t$
- Example: Unicicle - Slides

Autonomous and Mobile Robotics

Prof. Giuseppe Oriolo

Motion Planning I

Retraction and Cell Decomposition

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Definition

In a common scenario a mobile robot has to move in an environment where there are obstacles

The motion planning problem consists on planning a collision-free motion from an initial configuration q_i to a final configuration q_f

Obviously for this aim an a priori information about the obstacles is requested.

If this information is known the problem can be solved offline, otherwise there is an online problem.

For this latter, the robot needs to plan its motion with a partial or absent information about the workspace.

This information can be obtained by using sensors (deliberative navigation).

There exist also memoryless approaches where the robot acts instinctively to the current situation (reactive navigation).

motivation

- robots are expected to perform tasks in workspaces populated by obstacles

- autonomy requires that the robot is able to plan a collision-free motion from an initial to a final posture on the basis of geometric information

- information about the workspace geometry can be
  - entirely known in advance (off-line planning)
  - gradually discovered by the robot (on-line planning)

the canonical problem

- robot $\mathcal{B}$ (kinematic chain with fixed or mobile base) moving in a workspace $\mathcal{W} = \mathrm{R}^N$, $N = 2$ or 3

  $N=2$ → piano mover's problem $N=3$ → generalized mover's problem

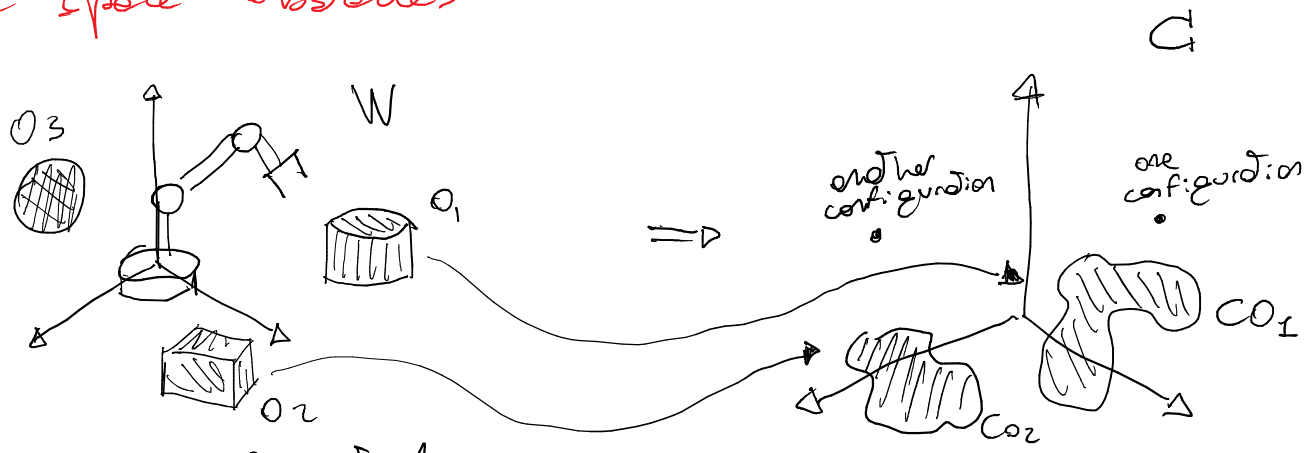
- $\mathcal{B}$ is free-flying in its configuration space $\mathcal{C}$, i.e., it is not subject to kinematic constraints of any kind

  the robot can move in any direction (free to move)

- obstacles $\mathcal{O}_1,...,\mathcal{O}_p$ (fixed rigid objects in $\mathcal{W}$)

given a start configuration $\boldsymbol{q}_s$ and a goal configuration $\boldsymbol{q}_g$ of $\mathcal{B}$ in $\mathcal{C}$, plan a path that connects $\boldsymbol{q}_s$ to $\boldsymbol{q}_g$ and is safe, i.e., it is completely contained in the free configuration space $\mathcal{C}_{\text{free}}$

C-space Obstacles



p = num. of obstacles

the robot in configuration space is represented by a point (a posture)

We want to characterize the configurations that puts the robot in collision with the obstacles $\Rightarrow$

$$CO_i = \{ q \in C : B(q) \cap O_i \neq 0 \}$$

- CO_i: image of the i-th object in the configuration space
- $B(q)$: robot placed at q: volume occupied by the robot at q
- $\cap O_i$: intersection with the obstacle
- i-th C-obstacle

In this way we simply need to avoid the images of the obstacles (C-obstacles) to avoid collisions

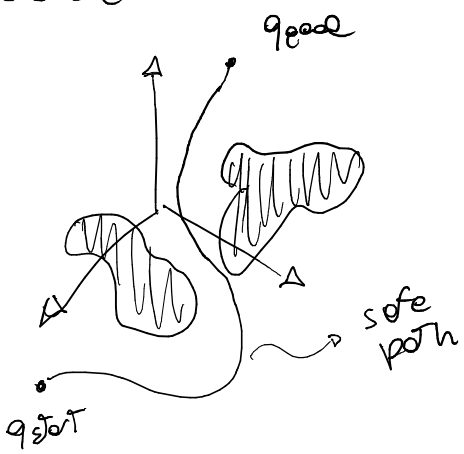
$$CO = \bigcup_{i=1}^{p} CO_i$$ C-Obstacle region $\rightarrow$ forbidden region

$$C_{free} = C - CO$$ free configuration space $\rightarrow$ the collision free region

A path in configuration space is free ("safe" or "collision free") if it lies completely in C_{free}

Therefore:

path: geometric problem

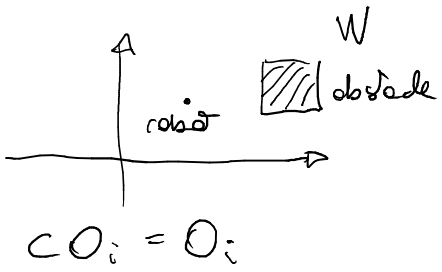


Examples

- Point robot in $W = \mathbb{R}^2$

$$q = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow C = W = \mathbb{R}^2$$

C-obstacles are copies of the original obstacles

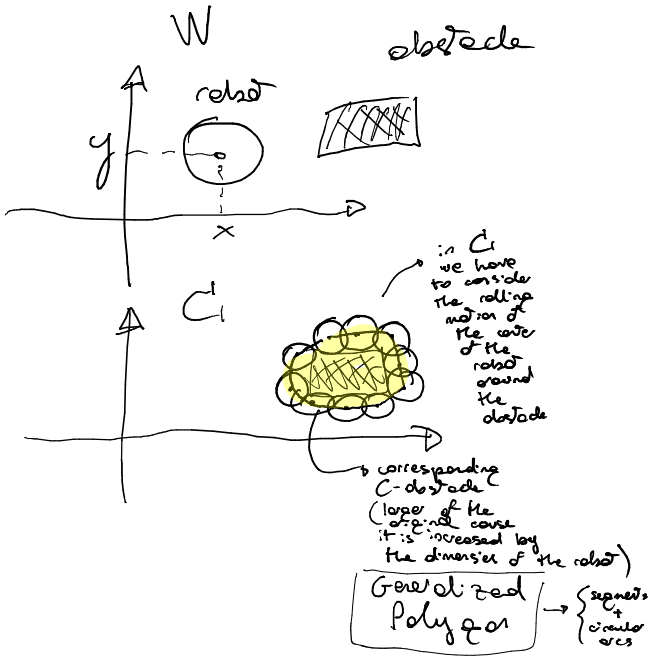


$CO_i = O_i$

- Circular robot in $W = \mathbb{R}^2$

$$q = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow C = W = \mathbb{R}^2$$

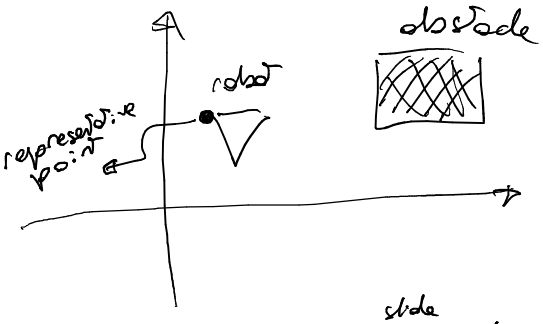
C-obstacles are not copies of the original obstacles



- A polygonal robot in $W = \mathbb{R}^2$ free to translate only (no rotations)

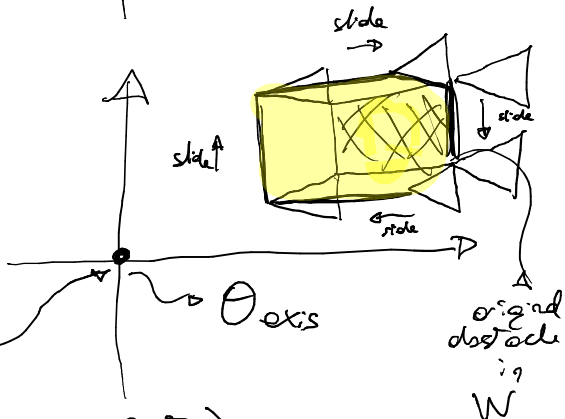
$$q = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow C = W = \mathbb{R}^2$$

fixed orientation



The shape of the C-obstacle depends on where we place the representative point.

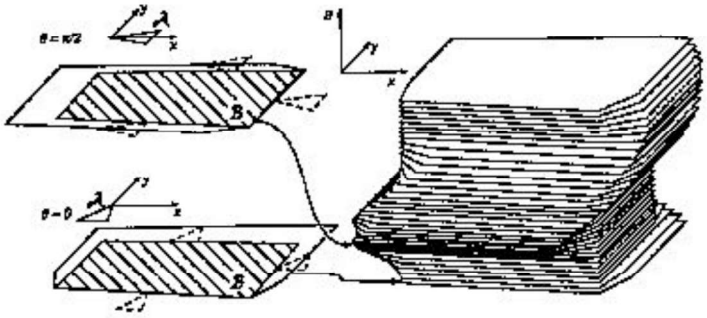
C-obstacles grows, not isotropically



If we add θ variable (robot can now rotate) the C space becomes 3-dimensional → θ axis coming out the whiteboard

C-obstacles when rotations are involved

for a polygonal robot free to translate and rotate on the plane



"grow and stack"

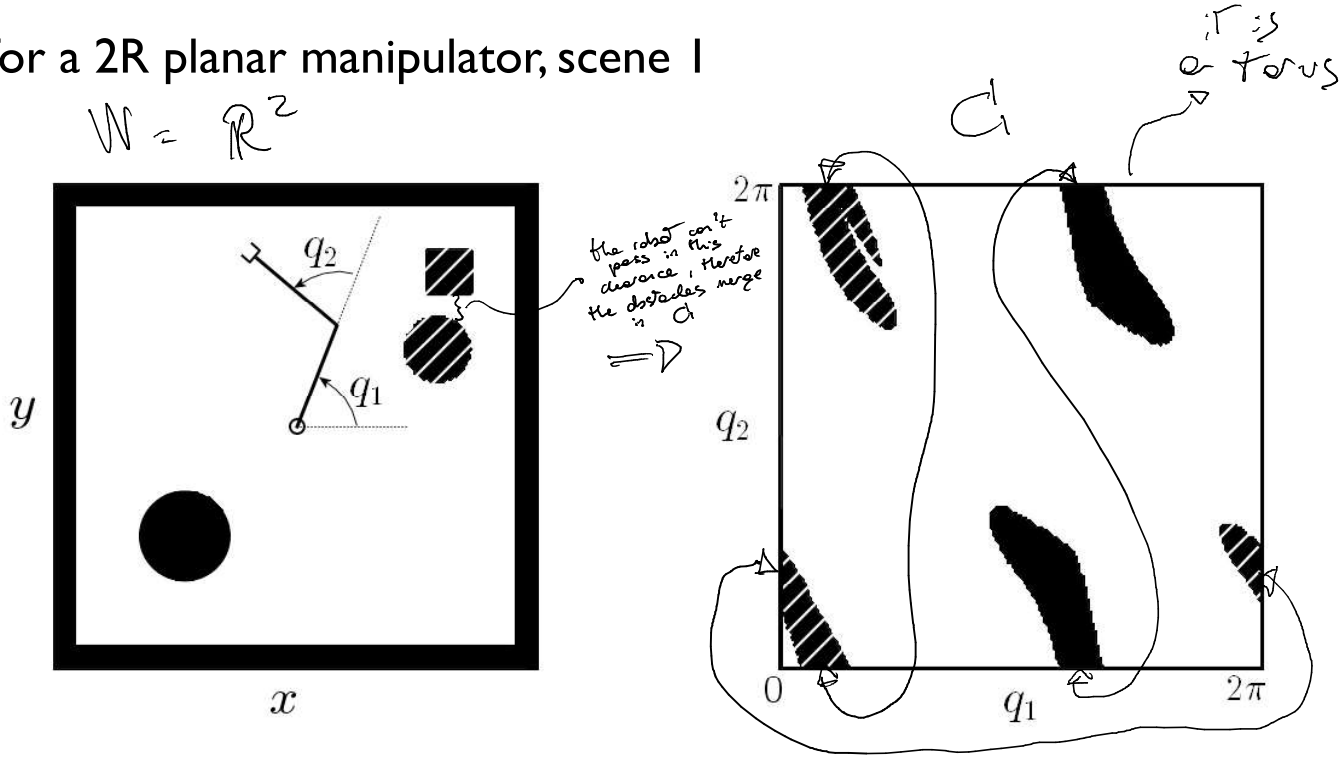
stewed deck of cards

↓

rotation of a robot with an obstacle of the shape of a rectangle

C-obstacles when rotations are involved

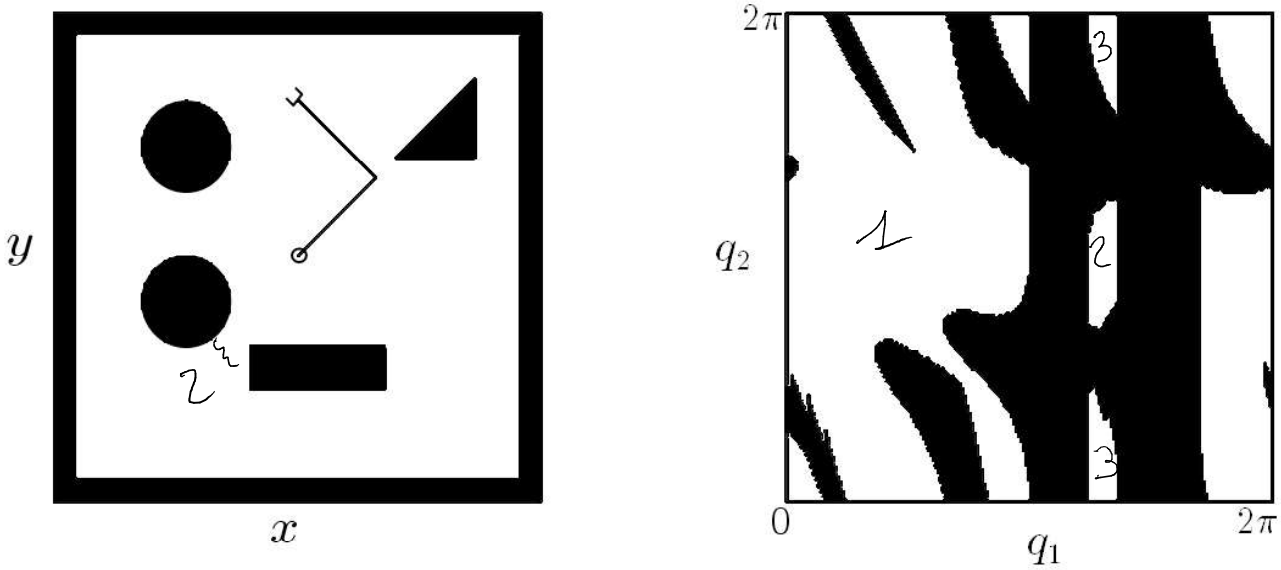
for a 2R planar manipulator, scene 1



disjoint workspace obstacles may merge in C-space

C-obstacles when rotations are involved

for a 2R planar manipulator, scene 2



the free configuration space may be disconnected

- single-body robot in $\mathbb{R}^2$: piano movers' problem
  single-body robot in $\mathbb{R}^3$: generalized movers' problem

- extensions to the canonical problem:
  - moving obstacles
  - on-line planning
  - kinematic (e.g., nonholonomic) constraints
  - manipulation planning (requires contact)

- many methods that can solve the canonical problem can be appropriately modified to address one or more of these extensions

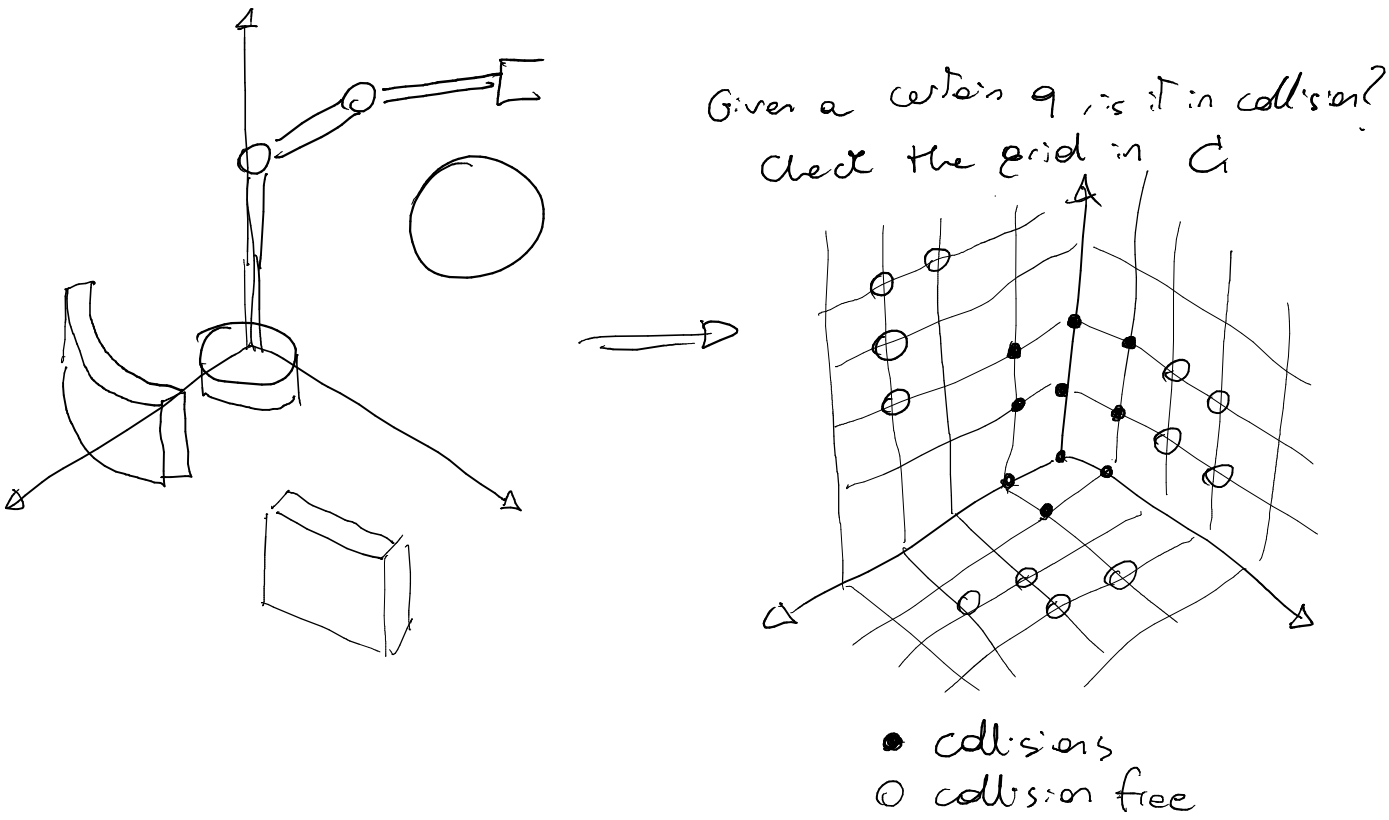
motion planning methods

- all work in the configuration space $\mathcal{C}$
- most need preliminary computation of the $\mathcal{C}$-obstacle region $\mathcal{CO}$, a highly expensive procedure (complexity is exponential in dim $\mathcal{C}$) → We have to solve the inverse kinematics problem for an infinity of points (the obstacle) and find the region that they represent in $\mathcal{C}$.
- computation of $\mathcal{CO}$ can be

  exact: requires an algebraic model of $\mathcal{O}_1,...,\mathcal{O}_p$ Rarely used

  approximate: e.g., sample $\mathcal{C}$ using a regular grid, compute the volume occupied by the robot at each sample, and check for collisions between this volume and the obstacles Can be done very efficiently
  Discretization of the configuration space
- efficient collision-checking algorithms exist, such as V-collide in R^2 and I-collide in R^3

Collision Checking



classification

1. roadmap methods

   represent the connectivity of $\mathcal{C}_{\text{free}}$ by a sufficiently rich network of safe paths

   e.g., retraction, cell decomposition

2. probabilistic methods

   a particular instance of sampling-based methods where samples of $\mathcal{C}$ are randomly extracted

   e.g., PRM, RRT

3. artificial potential field methods

   a heuristic approach which is particularly suitable for on-line planning

retraction method

the plane (no rotation, only translation)

- assume $\mathcal{C} = \mathrm{R}^2$ and $\mathcal{C}_{\text{free}}$ a polygonal limited subset (its boundary $\partial\mathcal{C}_{\text{free}}$ is entirely made of line segments)

- define the clearance of a configuration $\boldsymbol{q}$ in $\mathcal{C}_{\text{free}}$ as

minimum distance from the boundary

$$\gamma(\boldsymbol{q}) = \min_{\boldsymbol{s} \in \partial\mathcal{C}_{\text{free}}} \|\boldsymbol{q} - \boldsymbol{s}\|$$

any other configuration on the boundary of $\mathcal{C}_{\text{free}}$

$\boldsymbol{s}$ configuration

- define the neighbors of $\boldsymbol{q}$ as

$$N(\boldsymbol{q}) = \{\boldsymbol{s} \in \partial\mathcal{C}_{\text{free}} : \|\boldsymbol{q} - \boldsymbol{s}\| = \gamma(\boldsymbol{q})\}$$

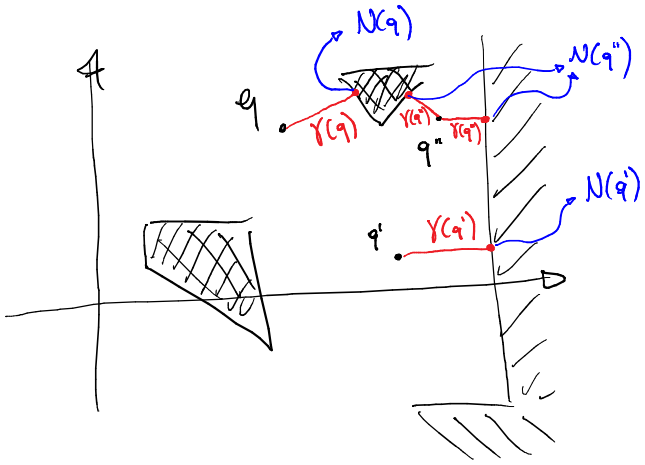
set of points at minimum distance from q

- the generalized Voronoi diagram of $\mathcal{C}_{\text{free}}$ is

cardinality

$$\mathcal{V}(\mathcal{C}_{\text{free}}) = \{\boldsymbol{q} \in \mathcal{C}_{\text{free}} : \text{card}(N(\boldsymbol{q})) > 1\}$$

Clearance



assumption: the boundary of C_{free} is made up of only segments!

The clearance of the configuration q is the minimum distance between q and any point of the boundary of C_{free}

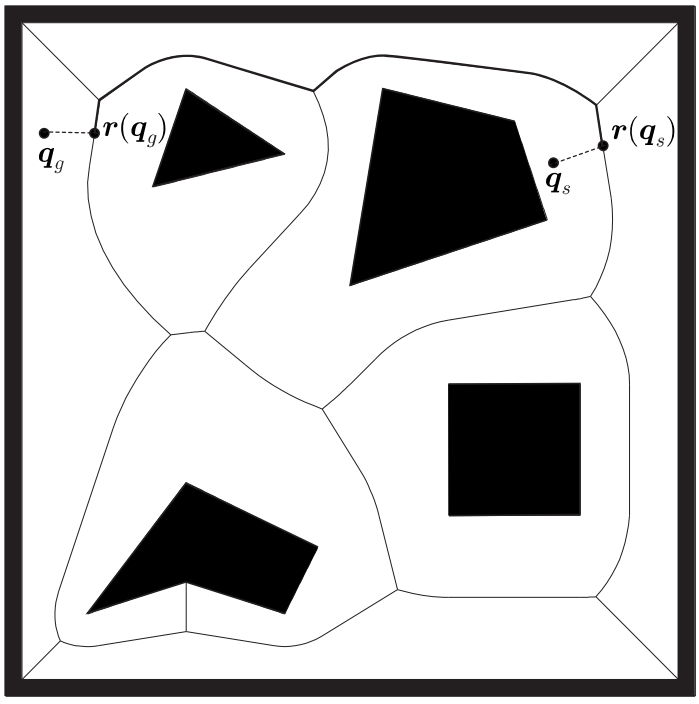
Neighbours: set of configurations whose distance from the point is equal to the clearance. There could be more neighbours in case of equidistance from 2 points or more. Therefore the cardinality of this set ≥ 1

$\Downarrow$

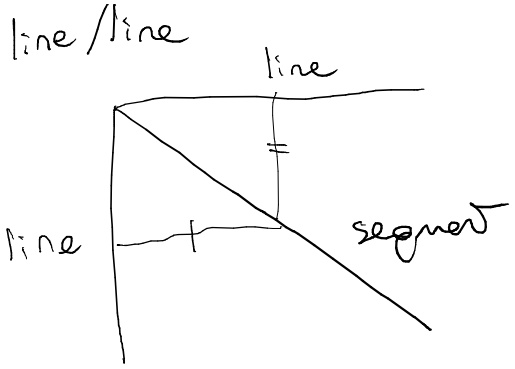
Generalized Voronoi

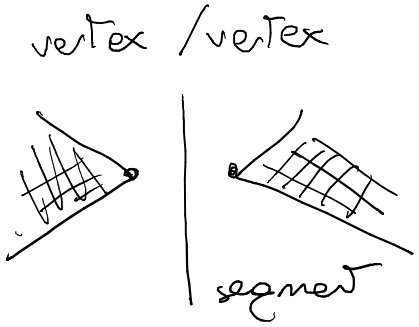
Generalized Voronoi diagram (defined by points): arcs represent the maximum clearance among the obstacles

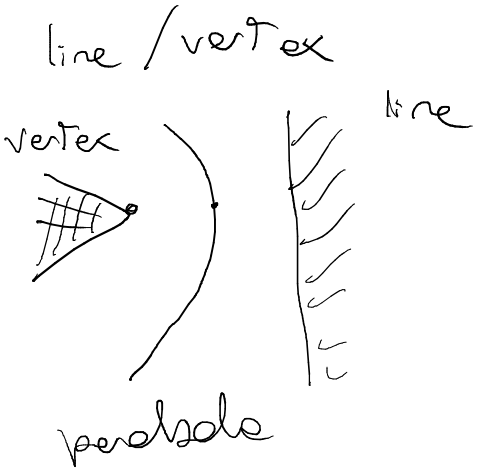
- its elementary arcs are
  - rectilinear (edge-edge, vertex-vertex)
  - parabolic (edge-vertex)
- can be seen as a graph
  - elementary arcs as arcs
  - arc endpoints as nodes
- a natural roadmap as it maximizes safety

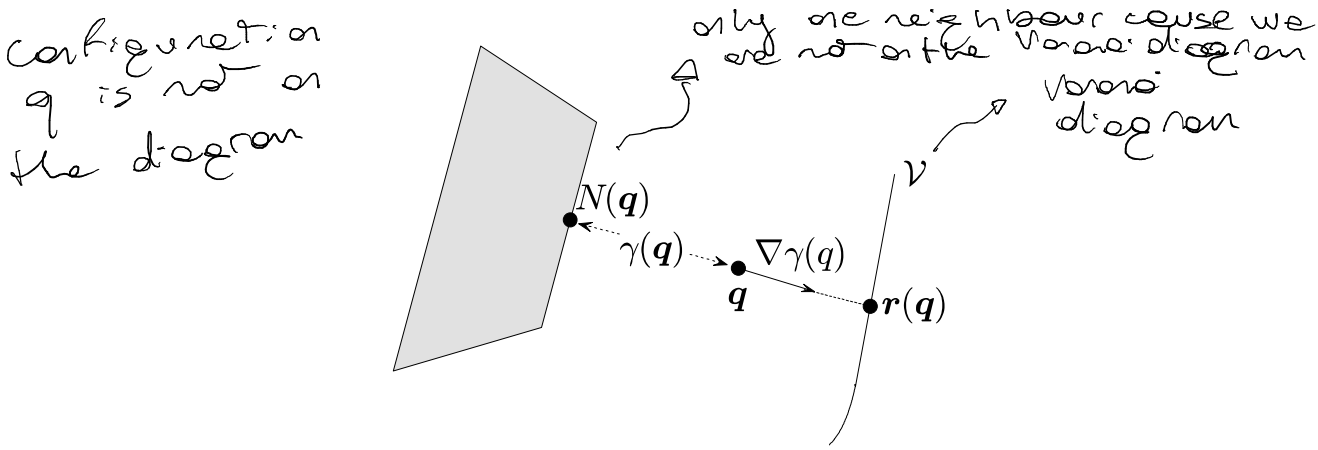


Voronoi diagram









- to connect any q to $\mathcal{V}(\mathcal{C}_{\text{free}})$, use retraction: from q, follow $\nabla\gamma$ up to the first intersection $r(q)$ with $\mathcal{V}(\mathcal{C}_{\text{free}})$
- $r(\cdot)$ preserves the connectivity of $\mathcal{C}_{\text{free}}$, i.e., q and $r(q)$ lie in the same connected component of $\mathcal{C}_{\text{free}}$
- hence, a safe path exists between q_s and q_g if and only if a path exists on $\mathcal{V}(\mathcal{C}_{\text{free}})$ between $r(q_s)$ and $r(q_g)$

algorithm

1. build the generalized Voronoi diagram $\mathcal{V}(\mathcal{C}_{\text{free}})$
2. compute the retractions $\boldsymbol{r}(\boldsymbol{q}_s)$ and $\boldsymbol{r}(\boldsymbol{q}_g)$
3. search $\mathcal{V}(\mathcal{C}_{\text{free}})$ for a sequence of arcs such that $\boldsymbol{r}(\boldsymbol{q}_s)$ belongs to the first and $\boldsymbol{r}(\boldsymbol{q}_g)$ to the last
4. if successful, return the solution path consisting of
   a. line segment from $\boldsymbol{q}_s$ to $\boldsymbol{r}(\boldsymbol{q}_s)$
   b. portion of first arc from $\boldsymbol{r}(\boldsymbol{q}_s)$ to its end
   c. second, third, ... , penultimate arc
   d. portion of last arc from its start to $\boldsymbol{r}(\boldsymbol{q}_g)$
   e. line segment from $\boldsymbol{r}(\boldsymbol{q}_g)$ to $\boldsymbol{q}_g$

   otherwise, report a failure → the problem cannot be solved

- graph search at step 3: if a minimum-length path is desired, label each arc with a cost equal to its length, and use $A^\star$ to compute a minimum-cost solution
  Graph search algorithms : Dijkstra etc...
- the retraction method is complete, i.e., finds a solution when one exists and reports failure otherwise; and multiple-query, as one can build $\mathcal{V}(\mathcal{C}_{\text{free}})$ once for all → this algorithm can be used also for the q start q goal
- complexity: if $\mathcal{C}_{\text{free}}$ has v vertices, $\mathcal{V}(\mathcal{C}_{\text{free}})$ has $O(v)$ arcs
  - step 1: $O(v \log v)$ → computing voronoi diagram
  - step 2: $O(v)$ → computing the retractions on q_s & q_g
  - step 3: $O(v \log v)$ ($A^\star$ on a graph with $O(v)$ arcs) → computing A^*

  altogether, the time complexity is $O(v \log v)$
- extensions (e.g., to higher-dimensional configuration spaces) are possible but quite complicated

this method is only practical for motion of single body robots

cell decomposition methods

off-line method : geometry known in advance

- idea: decompose $\mathcal{C}_{\text{free}}$ in cells, i.e., regions such that
  - it is easy to compute a safe path between two configurations in the same cell
  - it is easy to compute a safe path between two configurations in adjacent cells
- once a cell decomposition of $\mathcal{C}_{\text{free}}$ is computed, find a sequence of cells (channel) with $\boldsymbol{q}_s$ in the first and $\boldsymbol{q}_g$ in the last
- different methods are obtained depending on the type of cells used for the decomposition

exact decomposition

collection of regions

- assume $\mathcal{C} = \mathrm{R}^2$ and $\mathcal{C}_{\text{free}}$ a polygonal limited subset

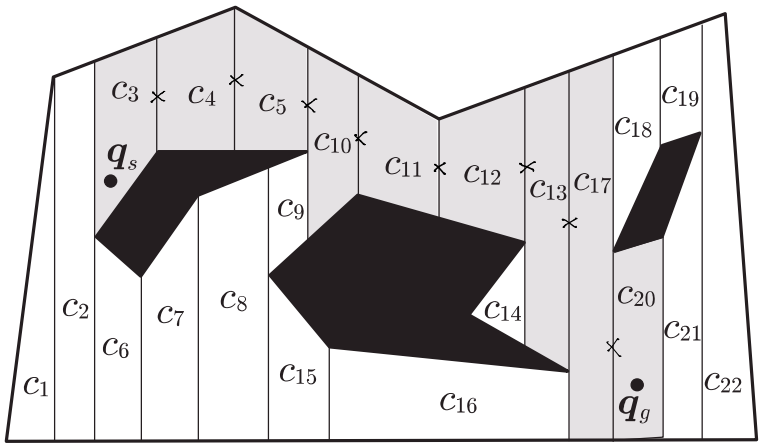
↳ Boundary made of line segments

- variable-shape cells are needed to decompose exactly $\mathcal{C}_{\text{free}}$; a typical choice are convex polygons
- convexity guarantees that it is easy to plan in a cell and between adjacent cells

Importance of convexity

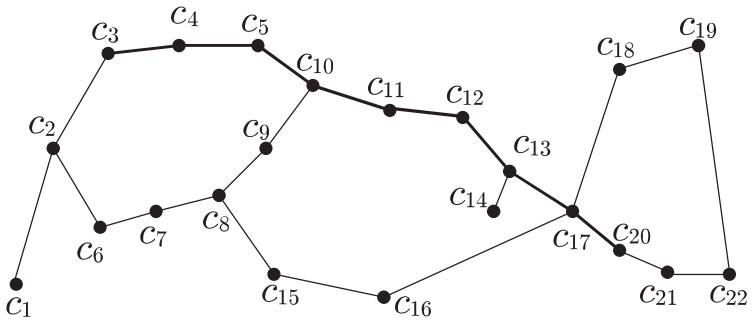
- the sweep-line algorithm can be used to compute a decomposition of $\mathcal{C}_{\text{free}}$ into convex polygons

sweep-line algorithm



every cell is a trapezoid

- sweep a line over $\mathcal{C}_{\text{free}}$; when it goes through a vertex, two segments (extensions) originate at the vertex
- an extension lying in $\mathcal{C}_{\text{free}}$ is part of the boundary of a cell; the rest are other extensions and/or parts of $\partial\mathcal{C}_{\text{free}}$
- the result is a trapezoidal decomposition



- build the associated connectivity graph C
- identify nodes (cells) c_s and c_g where $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$ are
- use graph search to find a path on C from c_s to c_g; this represents a channel of cells
- extract from the channel a safe solution path, e.g., joining $\boldsymbol{q}_s$ to $\boldsymbol{q}_g$ via midpoints of common boundaries

algorithm

1. compute a convex polygonal decomposition of $\mathcal{C}_{\text{free}}$
2. build the associated connectivity graph C
3. search C for a channel of cells from c_s to c_g
4. if successful, extract and return a solution path consisting of
   a. line segment from $\boldsymbol{q}_s$ to the midpoint of the common boundary between the first two cells
   b. line segments between the midpoints of consecutive cells
   c. line segment from the midpoint of the common boundary between the last two cells and $\boldsymbol{q}_g$

   otherwise, report a failure → there exists no solution

- if a minimum-length channel is desired, define a modified connectivity graph with $\boldsymbol{q}_s$, $\boldsymbol{q}_g$ and all the midpoints as nodes, and line segments between nodes in the same cell as arcs, each with a cost equal to its length, and use $A^\star$ to compute a minimum-cost path
- the exact cell decomposition method is complete and multiple-query, as one can build the connectivity graph once for all
- complexity: if $\mathcal{C}_{\text{free}}$ has v vertices, C has $O(v)$ arcs
  - step 1: $O(v \log v)$
  - step 2: $O(v)$
  - step 3: $O(v \log v)$ ($A^\star$ on a graph with $O(v)$ arcs)

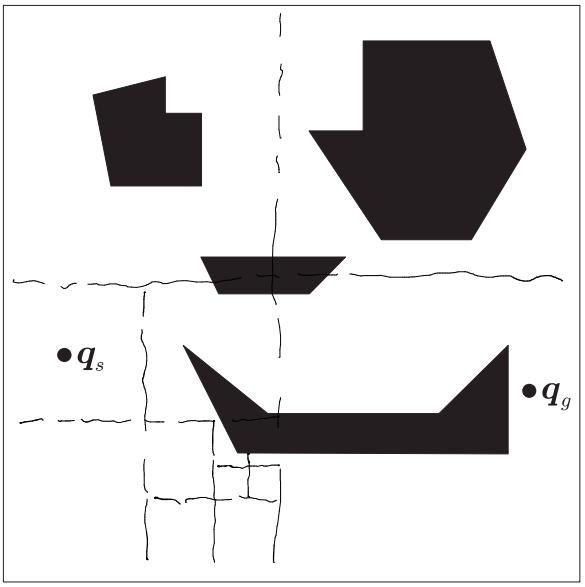
  altogether, the time complexity is $O(v \log v)$

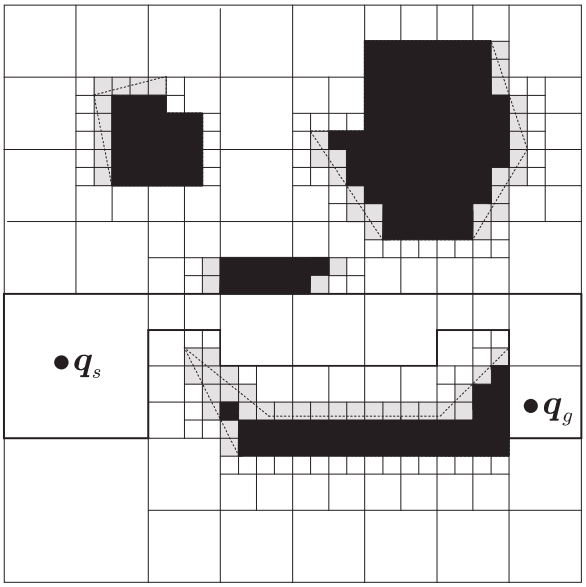
- a channel is more flexible than a roadmap because it contains an infinity of paths; this may be exploited to take into account nonholonomic constraints or to avoid unexpected obstacles during the motion
- the solution path is a broken line, but smoothing may be performed in a post-processing phase
- if $\mathcal{C} = \mathrm{R}^3$ and $\mathcal{C}_{\mathrm{free}}$ is a polyhedral limited subset, the sweep-plane algorithm may be used to compute a decomposition of $\mathcal{C}_{\mathrm{free}}$ into convex polyhedra
- an extension to configuration spaces of arbitrary dimension exists but it is very inefficient: in fact, complexity is exponential in the dimension of $\mathcal{C}$

approximate decomposition

- assume $\mathcal{C} = \mathrm{R}^2$ and $\mathcal{C}_{\mathrm{free}}$ a polygonal limited subset
- fixed-shape cells are used to obtain an approximate decomposition (by defect) of $\mathcal{C}_{\mathrm{free}}$; e.g., squares
- as in exact decomposition, convexity guarantees that it is easy to plan in a cell and between adjacent cells
- a recursive algorithm is used for decomposition to reach a trade-off between simplicity and accuracy

4 cells at the start

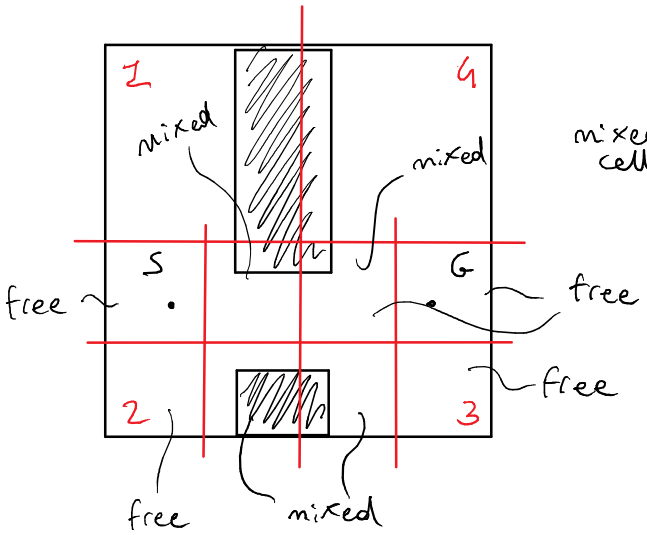


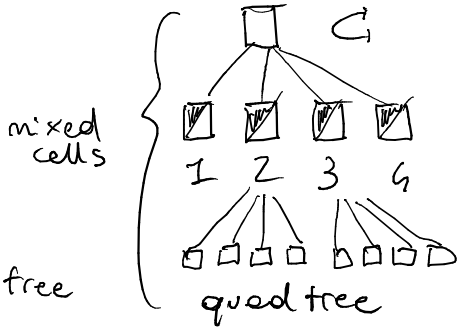


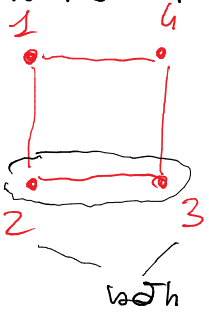
- start by dividing $\mathcal{C}$ in 4 cells and classifying each cell as
  - free, if its interior is completely in $\mathcal{C}_{\text{free}}$
  - occupied, if it is completely in $\mathcal{CO}$
  - mixed, if it is neither free nor occupied

Quadtree : tree with 0 or 4 children

→ C is the radix node
→ leaf nodes are the free and occupied cells
→ transitions "father - 4 children" are done of mixed cells and their decomposition

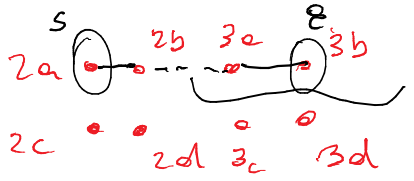






We want to go from 2 (start) to 3 (goal)

1 4



mixed → recursion of the algorithm and divide again the squares 2b 3a until no mixed squares appear

associated graph

- build the connectivity graph C associated to the current level of decomposition, with free and mixed cells as nodes and arcs between adjacent nodes
- identify nodes (cells) c_s and c_g where $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$ are, and use graph search to look for a path (channel) on C from c_s to c_g; if it does not exist, report failure
- if a path (channel) exists on C from c_s to c_g, take any mixed cell in the path and decompose it as before
- repeat the above steps (build C, look for a path and decompose mixed cells) until a path is found made only of free cells, or until a minimum size has been reached for the cells; in the latter case, backtrack

- the above planning method is
  - resolution complete, in the sense that a solution is found if and only if one exists at the maximum allowed resolution
  - single-query, because the decomposition is guided by the given $\boldsymbol{q}_s$, $\boldsymbol{q}_g$
- recursive decomposition of cells can be implemented efficiently using quadtrees (trees whose internal nodes have exactly four children)
- the method is conceptually applicable to configuration spaces of arbitrary dimension: however, complexity is still exponential in the dimension of $\mathcal{C}$

Final comparisons

A channel is more flexible than a roadmap because it contains an infinity of paths; this may be exploited to take into account non holonomic constraints or to avoid unexpected obstacles during the motion.

Approximate decomposition is more a general approach than the exact decomposition, in fact it approaches to all the shapes in a more simple way.

Autonomous and Mobile Robotics

Prof. Giuseppe Oriolo

Motion Planning 2

Probabilistic Planning

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

sampling-based methods

it should not be built by knowing the $\mathcal{C}_{free}$ in advance

- build a roadmap of the configuration space $\mathcal{C}$ by repeating this basic iteration: (Algorithm)
  - extract a sample q of $\mathcal{C}$ a random point or chosen along the grid
  - use forward kinematics to compute the volume $\mathcal{B}(q)$ occupied by the robot $\mathcal{B}$ at q
  - check collision between $\mathcal{B}(q)$ and obstacles $\mathcal{O}_1,...,\mathcal{O}_p$ in the workspace (Yes, it's in collision / No, it's free)
  - if $q \in \mathcal{C}_{\text{free}}$, add q to the roadmap; else, discard it

It is a graph where each node represents a configuration of $\mathcal{C}_{free}$

- preliminary computation of $\mathcal{CO}$ is completely avoided: an approximate representation of $\mathcal{C}_{\text{free}}$ is directly built as a collection of connected configurations (roadmap)

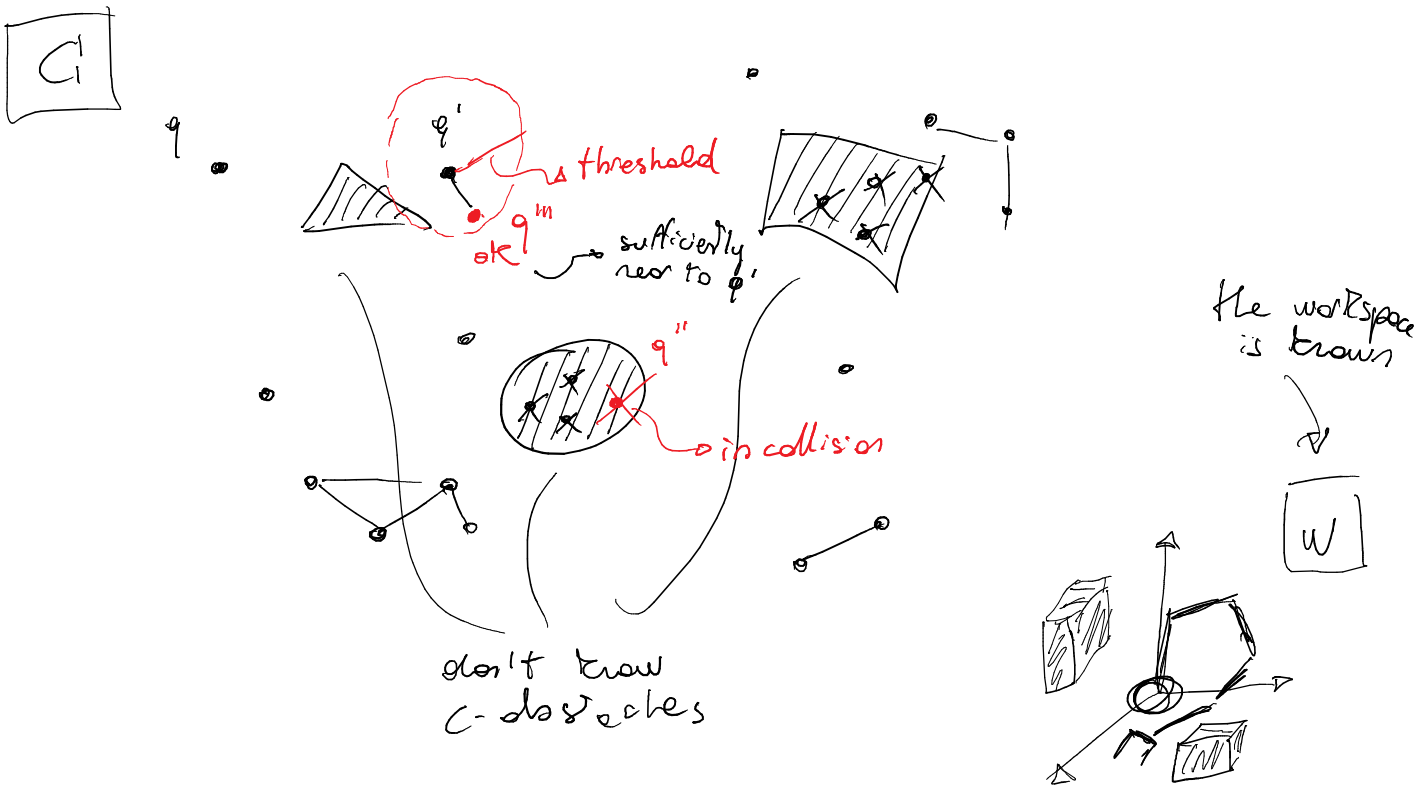
two nodes are connected if there exists a free node between them

- different criteria for sampling lead to different methods: in general, randomized outperforms deterministic

if we choose the sample q randomly, the performances are much faster than a deterministic choice

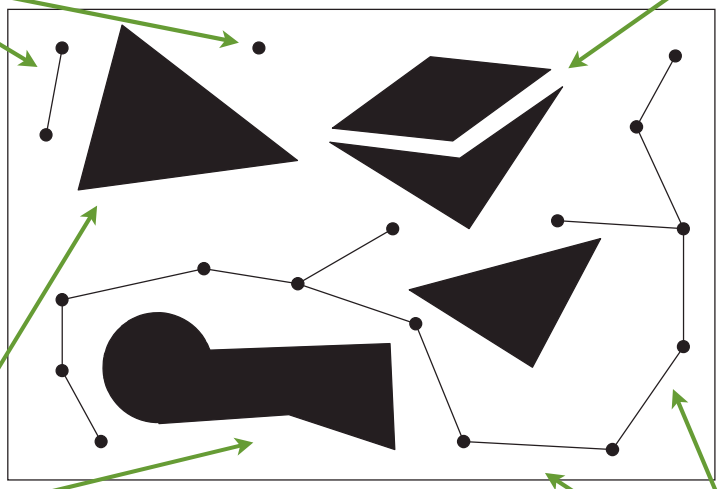
PRM (Probabilistic Roadmap)

- basic iteration to build the PRM:
  - extract a sample q of $\mathcal{C}$ with uniform probability distribution
  - compute $\mathcal{B}(q)$ and check for collision
  - if $q \in \mathcal{C}_{\text{free}}$, add q to the PRM; else, discard it
  - search the PRM for "sufficiently near" configurations q_{near}
  - if possible, connect q to q_{near} with a free local path
- the generation of a free path between q and q_{near} is delegated to a procedure called local planner: e.g., throw a linear path and check it for collision
- the chosen metric in $\mathcal{C}$ plays a role in identifying q_{near}



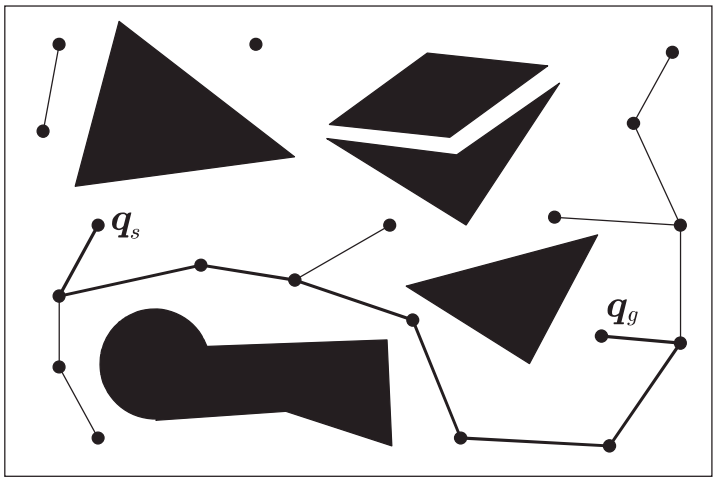
disconnected components

narrow passages are scarcely sampled



$\mathcal{C}$-obstacles are never computed

local paths



- construction of the PRM is arrested when
  1. disconnected components become less than a threshold, or
  2. a maximum number of iterations is reached
- if q_s and q_g can be connected to the same component, a solution can be found by graph search; else, enhance the PRM by performing more iterations

- the PRM method is probabilistically complete, i.e., the probability of finding a solution whenever one exists tends to 1 as the execution time tends to ∞; and is multiple-query (new queries enhance the PRM)

- the main advantage is speed; the time PRM needs to find a solution in high-dimensional spaces can be orders of magnitude smaller than previous planners

Weakness of the algorithm

- narrow passages are critical; heuristics may be used to design biased (non-uniform) probability distributions aimed at increasing sampling in such areas

PRM algorithm

1. Extract $q \in C$ with uniform probability distribution
2. Compute $B(q)$ to avoid collisions with $O_1 \dots O_p \rightarrow q \in C_{free}$
3. If $q \notin C_{free}$ discard it and restart from 1, otherwise:
   - add q as a node to the graph
   - Find the set $N(q)$ of nodes sufficiently near to q basing on a certain threshold and for each node $q_{near} \in N(q)$:
     4. Verify if there exists a free path which connects q and q_{near}; if so, add an arc to the graph between the two.

* This algorithm is called "local planner". In the most simple case, it verifies wether a segment between q and q_{free} belongs to C_{free} or not

At the end, if the maximum number of iterations OR if the graph is sufficiently connected:

Add q_s and q_g to the graph with the same procedure used for the generic q.

Finally, we verify if there exist a neighbor of q_s and one of q_g connected (with a graph search algorithm).

If not, we exclude again q_s and q_g and we proceed with the iterations to extend the graph

RRT (Rapidly-exploring Random Tree)

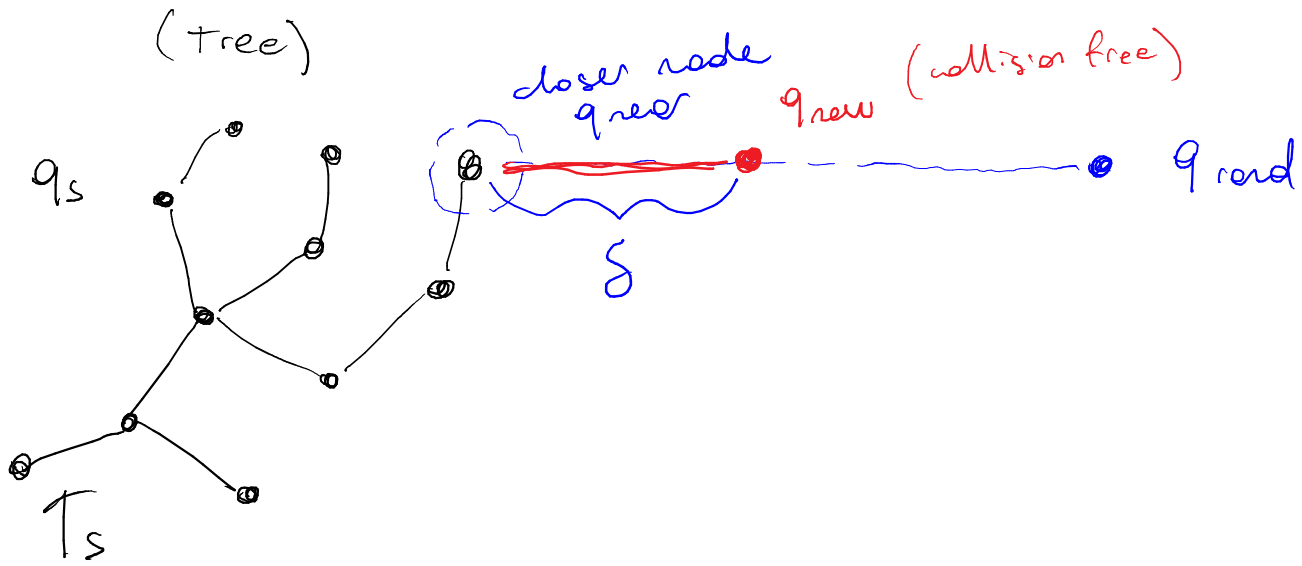
most popular in robotics

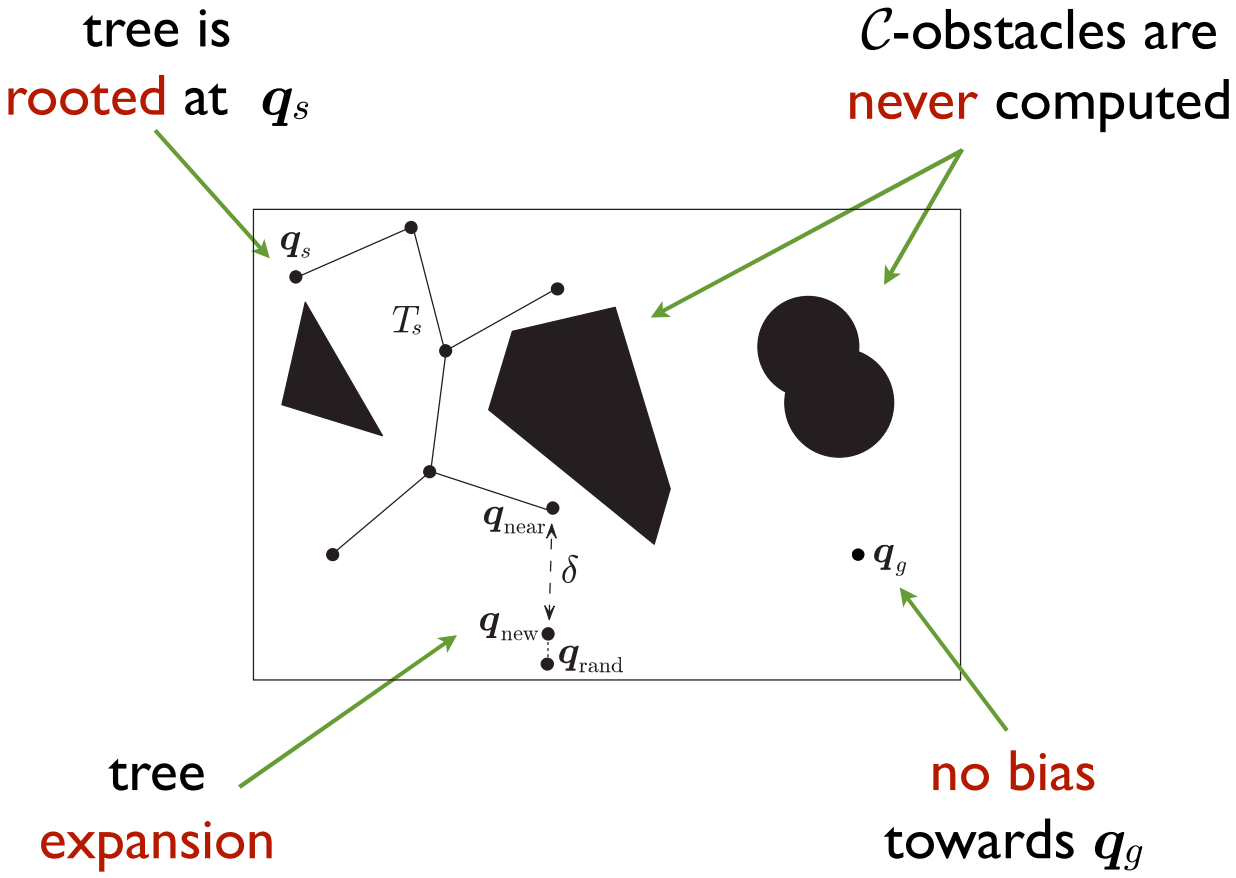
- basic iteration to build the tree T_s rooted at q_s:
  - generate q_{rand} in $\mathcal{C}$ with uniform probability distribution
  - search the tree for the nearest configuration q_{near}
  - choose q_{new} at a distance δ from q_{near} in the direction of q_{rand}
  - check for collision q_{new} and the segment from q_{near} to q_{new}
  - if check is negative, add q_{new} to T_s (expansion)

- the chosen metric in $\mathcal{C}$ plays a role in identifying q_{near}

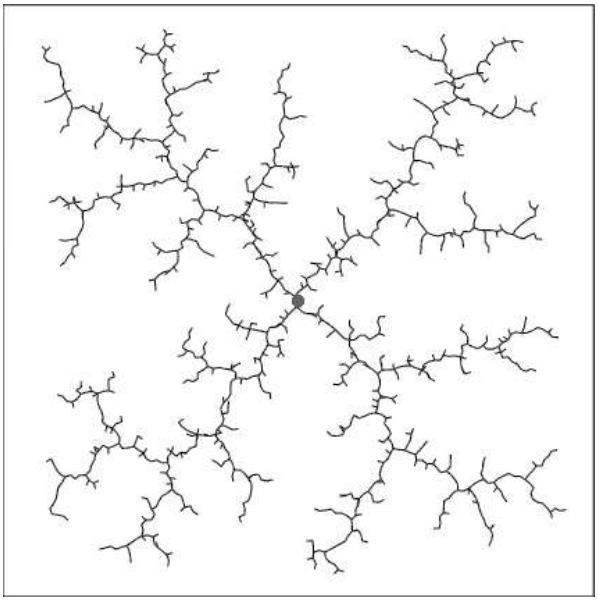
- T_s rapidly covers $\mathcal{C}_{\text{free}}$ because the expansion is biased towards unexplored areas (actually, towards larger Voronoi regions)

RRT



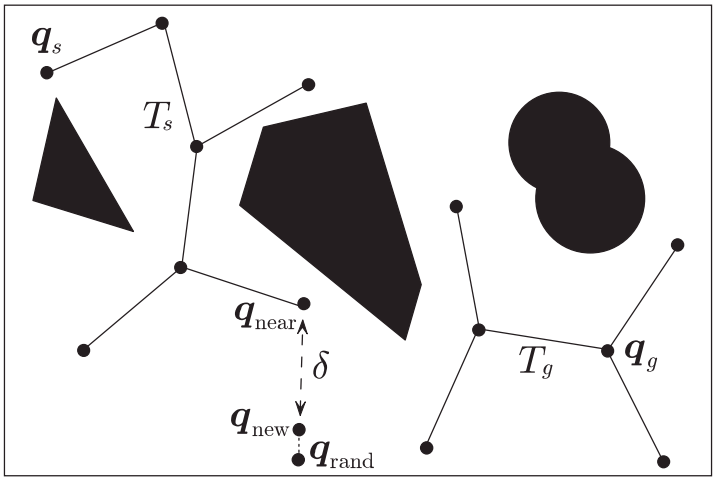


RRT in empty 2D space



quickly explores all areas, much more efficiently than other simple strategies, e.g., random walks

- to introduce a bias towards q_g, one may grow two trees T_s and T_g, respectively rooted at q_s and q_g (bidirectional RRT)
- alternate expansion and connection phases: use the last generated q_{new} of T_s as a q_{rand} for T_g, and then repeat switching the roles of T_s and T_g



Another possibility for biasing the exploration towards the goal (alternative to bidirectional version)

- ε - greedy exploration

choose an $\varepsilon \in (0,1)$

at each step k of the algorithm (number)

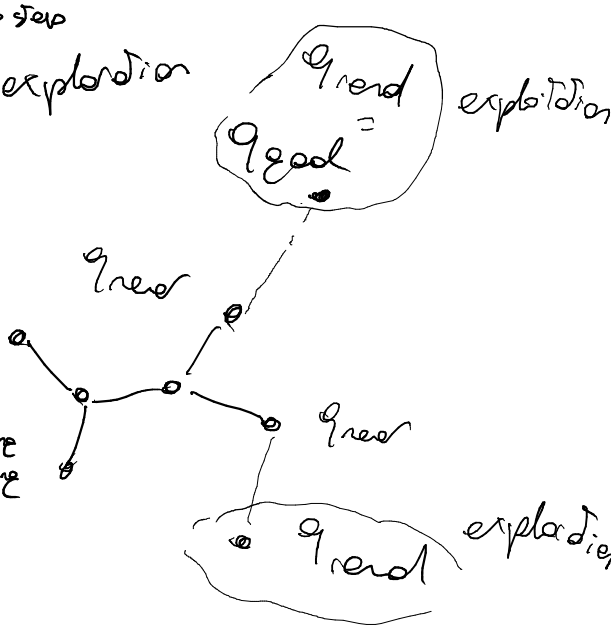
generate a random $r_k \in (0,1)$

if $r_k < \varepsilon$ then (to step)

$q_{rand} = \text{random}(q)$ → exploration

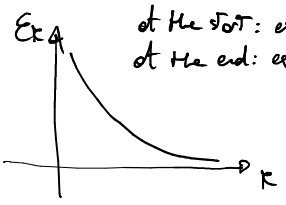
else

$q_{rand} = q_{goal}$



exploitation phase

(greedy



at the start: exploring
at the end: exploiting

can also set

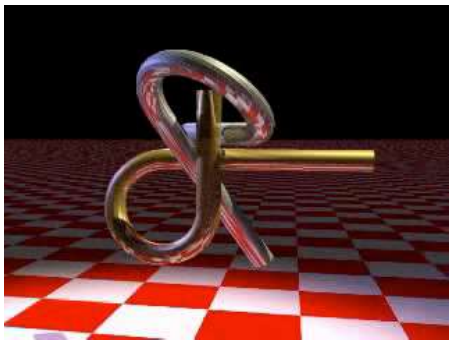
$$\varepsilon_k = \frac{\varepsilon_0}{1 + \alpha k}$$

- bidirectional RRT is probabilistically complete and single-query (trees are rooted at q_s and q_g, and in any case new queries may require significant work)

- many variations are possible: e.g., one may use an adaptive stepsize δ to speed up motion in wide open areas (greedy exploration)

- can be modified to address many extensions of the canonical planning problem, e.g., moving obstacles, nonholonomic constraints, manipulation planning

a benchmark problem: the Alpha Puzzle



- 6-dof configuration space + narrow passages
- solved by bidirectional RRT in few mins (average)
- in practice, this problem is not solvable by classical methods such as retraction or cell decomposition

RRT: extension to nonholonomic robots

- motion planning for a unicycle in $\mathcal{C} = \mathrm{R}^2 \times SO(2)$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \omega$$

- linear paths in $\mathcal{C}$ such as those used to connect $\boldsymbol{q}_{\mathrm{near}}$ to $\boldsymbol{q}_{\mathrm{rand}}$ are not admissible in general

- one possibility is to use motion primitives, i.e., a finite set of admissible local paths, produced by a specific choice of the velocity inputs

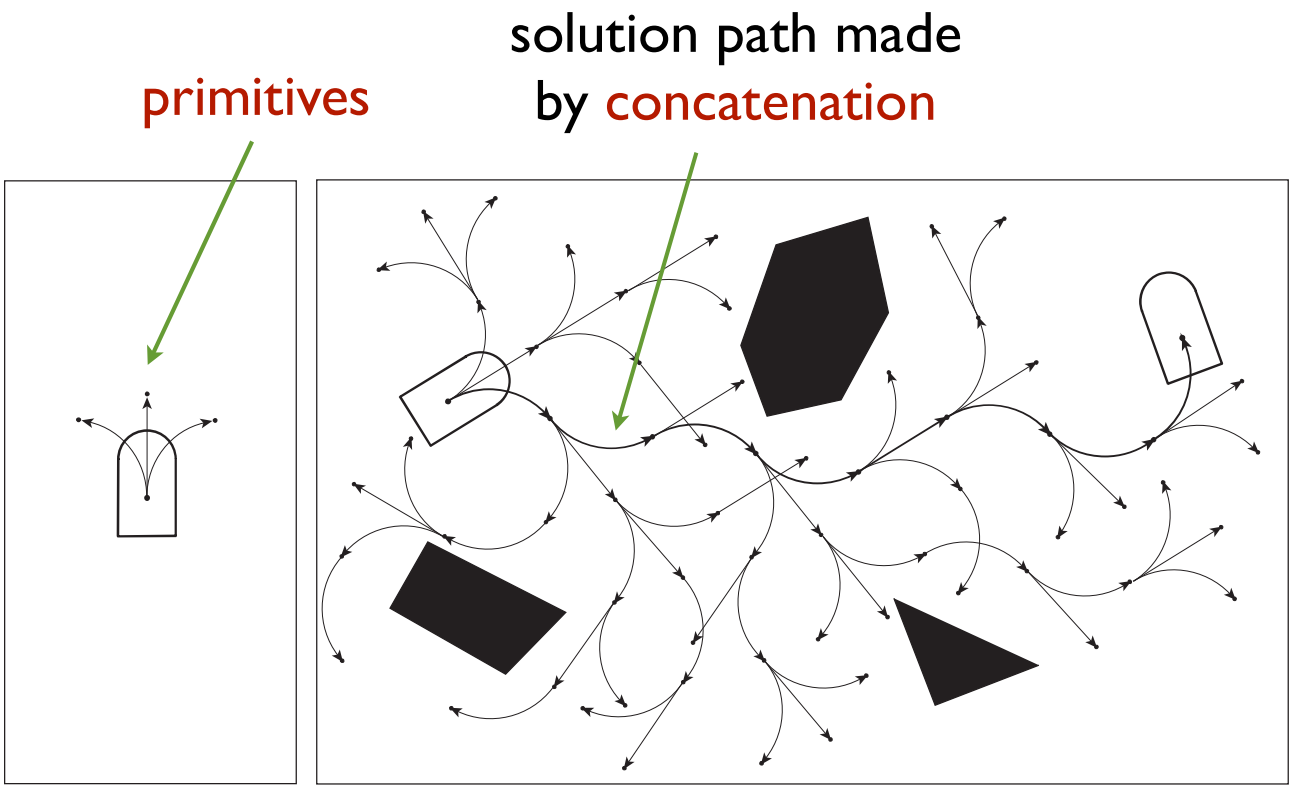
- for example, one may use (Dubins car)

$$v = \bar{v} \qquad \omega = \{-\bar{\omega}, 0, \bar{\omega}\} \qquad t \in [0, \Delta]$$

resulting in 3 possible paths in forward motion

- the algorithm is the same with the only difference that $\boldsymbol{q}_{\mathrm{new}}$ is generated from $\boldsymbol{q}_{\mathrm{near}}$ selecting one of the possible paths (either randomly or as the one that leads the unicycle closer to $\boldsymbol{q}_{\mathrm{rand}}$)

- if $\boldsymbol{q}_g$ can be reached from $\boldsymbol{q}_s$ with a collision-free concatenation of primitives, the probability that a solution is found tends to 1 as the time tends to ∞


primitives
solution path made
by concatenation

Autonomous and Mobile Robotics

Prof. Giuseppe Oriolo

Motion Planning 3

Artificial Potential Fields

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti
Sapienza
Università di Roma

on-line planning

- autonomous robots must be able to plan on line, i.e, using partial workspace information collected during the motion via the robot sensors We don't have a map
- incremental workspace information may be integrated in a map and used in a sense-plan-move paradigm (deliberative navigation)
  → First plan and then move
- alternatively, incremental workspace information may be used to plan motions following a memoryless stimulus-response paradigm (reactive navigation)

artificial potential fields

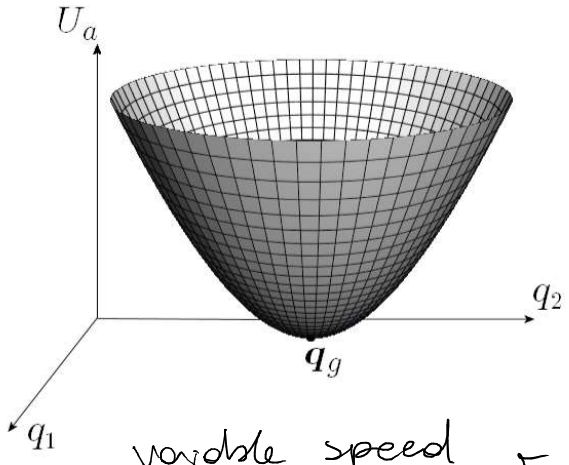
- idea: build potential fields in $\mathcal{C}$ so that the point that represents the robot is attracted by the goal $\boldsymbol{q}_g$ and repelled by the $\mathcal{C}$-obstacle region $\mathcal{CO}$

- the total potential U is the sum of an attractive and a repulsive potential, whose negative gradient $-\nabla U(\boldsymbol{q})$ indicates the most promising local direction of motion

- the chosen metric in $\mathcal{C}$ plays a role

attractive potential

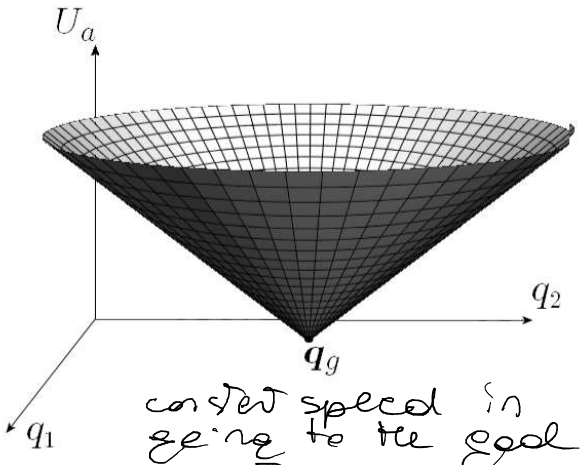
- objective: to guide the robot to the goal $\boldsymbol{q}_g$
- two possibilities; e.g., in $\mathcal{C} = \mathrm{R}^2$



variable speed in going to the target

paraboloidal

linear gradient



constant speed in going to the goal

constant gradient

conical

(be careful to not overshoot the goal)

- paraboloidal: let $\boldsymbol{e} = \boldsymbol{q}_g - \boldsymbol{q}$ and choose $k_a > 0$

$$U_{a1}(\boldsymbol{q}) = \frac{1}{2} k_a \boldsymbol{e}^T(\boldsymbol{q})\boldsymbol{e}(\boldsymbol{q}) = \frac{1}{2} k_a \|\boldsymbol{e}(\boldsymbol{q})\|^2$$

- the resulting attractive force is linear in $\boldsymbol{e}$

$$\boldsymbol{f}_{a1}(\boldsymbol{q}) = -\nabla U_{a1}(\boldsymbol{q}) = k_a \boldsymbol{e}(\boldsymbol{q})$$

- conical:

$$U_{a2}(\boldsymbol{q}) = k_a \|\boldsymbol{e}(\boldsymbol{q})\|$$

- the resulting attractive force is constant

$$\boldsymbol{f}_{a2}(\boldsymbol{q}) = -\nabla U_{a2}(\boldsymbol{q}) = k_a \frac{\boldsymbol{e}(\boldsymbol{q})}{\|\boldsymbol{e}(\boldsymbol{q})\|}$$

$U_{a1} = \frac{1}{2} k_a e^T e = \frac{1}{2} k_a (e_1^2 + e_2^2 + \dots + e_n^2)$ $\quad e_i = q_{g,i} - q_i$

$-\nabla U_{a1} = -\begin{pmatrix} \frac{\partial U_{a1}}{\partial q} \\ \vdots \\ \frac{\partial U_{a1}}{\partial q_n} \end{pmatrix}$ ↓ differentiating wrt q_i $\quad \frac{\partial U_{a1}}{\partial q_i} = -\frac{1}{2} k_a 2 e_i = -k_a e_i$

$-\nabla U_{a1} = -\begin{pmatrix} -k_a e_1 \\ \vdots \\ -k_a e_n \end{pmatrix} = k_a \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = k_a e_i$

- $\boldsymbol{f}_{a1}$ behaves better than $\boldsymbol{f}_{a2}$ in the vicinity of $\boldsymbol{q}_g$ but increases indefinitely with $\boldsymbol{e}$

- a convenient solution is to combine the two profiles: conical away from $\boldsymbol{q}_g$ and paraboloidal close to $\boldsymbol{q}_g$

$$U_a(\boldsymbol{q}) = \begin{cases} \frac{1}{2} k_a \|\boldsymbol{e}(\boldsymbol{q})\|^2 & \text{if} \quad \|\boldsymbol{e}(\boldsymbol{q})\| \leq \rho \\ k_b \|\boldsymbol{e}(\boldsymbol{q})\| & \text{if} \quad \|\boldsymbol{e}(\boldsymbol{q})\| > \rho \end{cases}$$

continuity of $\boldsymbol{f}_a$ at the transition requires

$$k_a \boldsymbol{e}(\boldsymbol{q}) = k_b \frac{\boldsymbol{e}(\boldsymbol{q})}{\|\boldsymbol{e}(\boldsymbol{q})\|} \qquad \text{for} \quad \|\boldsymbol{e}(\boldsymbol{q})\| = \rho$$

i.e., $k_b = \rho k_a$

Continuity

Continuity is given by the force and not by the potential, that is:

$$\nabla\left(\tfrac{1}{2} k_a \|e\|^2\right)\Big|_{\|e\|=\rho} = \nabla\left(k_b \|e\|\right)\Big|_{\|e\|=\rho}$$

Therefore

$$k_a e\Big|_{\|e\|=\rho} = k_b \frac{e}{\|e\|}\Big|_{\|e\|=\rho}$$

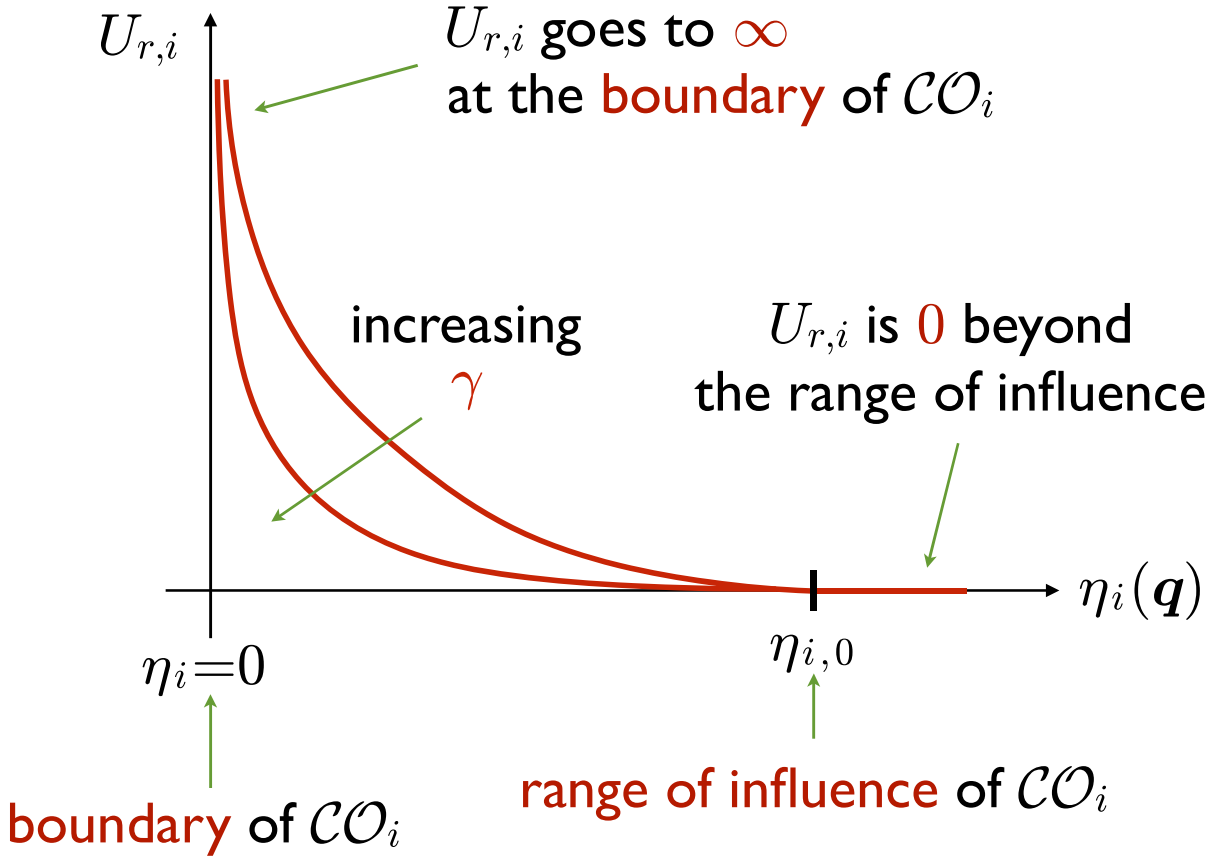
repulsive potential

- objective: keep the robot away from $\mathcal{CO}$
- assume that $\mathcal{CO}$ has been partitioned in advance in convex components $\mathcal{CO}_i$ (otherwise…)
- for each $\mathcal{CO}_i$ define a repulsive field

$$U_{r,i}(\boldsymbol{q}) = \begin{cases} \frac{k_{r,i}}{\gamma}\left(\frac{1}{\eta_i(\boldsymbol{q})} - \frac{1}{\eta_{0,i}}\right)^{\gamma} & \text{if } \eta_i(\boldsymbol{q}) \leq \eta_{0,i} \\ 0 & \text{if } \eta_i(\boldsymbol{q}) > \eta_{0,i} \end{cases}$$

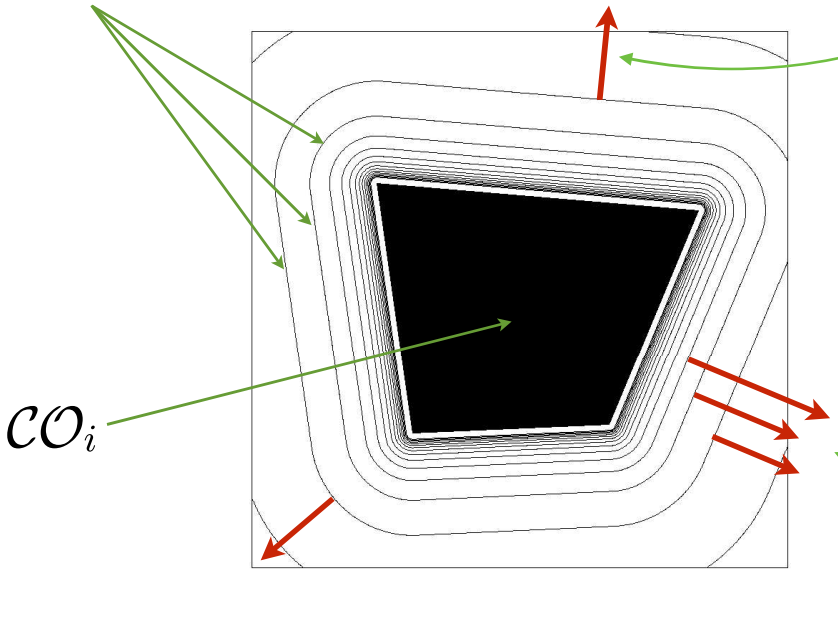
where $k_{r,i} > 0$; $\gamma = 2,3,\ldots$; $\eta_{0,i}$ is the range of influence of $\mathcal{CO}_i$; and $\eta_i(\boldsymbol{q})$ is the clearance

$$\eta_i(\boldsymbol{q}) = \min_{\boldsymbol{q}' \in \mathcal{CO}_i} \|\boldsymbol{q} - \boldsymbol{q}'\|$$



equipotential contours
follow the obstacle shape

repulsive forces are orthogonal
to equipotential contours



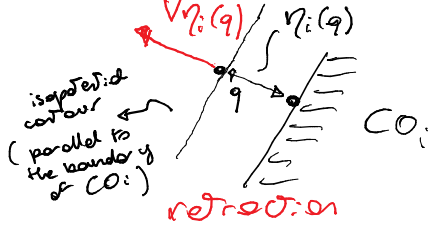
repulsive forces increase
approaching the boundary of CO_i

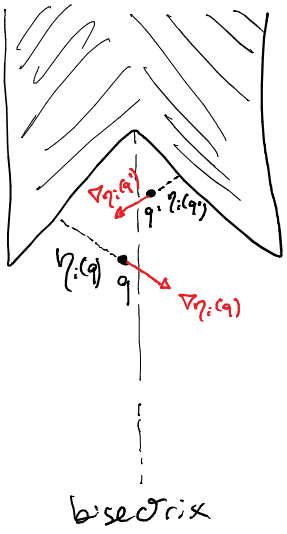
$$U_{r,i} = \begin{cases} \frac{k_{r,i}}{\gamma}\left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{i,0}}\right)^{\gamma} & \eta_i(q) \le \eta_{i,0} \\ 0 & \text{else} \end{cases}$$

$$\nabla U_{r,i}(q) = \begin{cases} \frac{k_{r,i}}{\gamma}\,\gamma\left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{i,0}}\right)^{\gamma-1}\left(-\frac{1}{\eta_i^2(q)}\right)\nabla\eta_i(q) & \eta_i(q) \le \eta_{i,0} \\ 0 & \text{else} \end{cases}$$

This is why γ cannot be 1 $\Rightarrow$ the force would not go to zero at the transition

$$f_{r,i} = \begin{cases} \underbrace{\frac{k_{r,i}}{\eta_i^2(q)}\left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{i,0}}\right)^{\gamma-1}}_{\text{scalar}}\ \underbrace{\nabla\eta_i(q)}_{\text{vector}} & \text{gradient of the clearance} \\ 0 \end{cases}$$

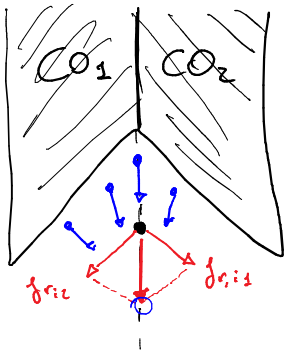




$f_{r,i} = \begin{cases} \alpha \nabla\eta_i(q) \\ 0 \end{cases}$ scalar gradient

along the bisectrix we have discontinuity!

Dividing this in two parts we solve the problem



convex decomposition of CO_i

$f_{r,i} = f_{r,i1} + f_{r,i2}$

is continuous everywhere!

- the resulting repulsive force is

$$\boldsymbol{f}_{r,i}(\boldsymbol{q}) = -\nabla U_{r,i}(\boldsymbol{q}) = \begin{cases} \dfrac{k_{r,i}}{\eta_i^2(\boldsymbol{q})}\left(\dfrac{1}{\eta_i(\boldsymbol{q})} - \dfrac{1}{\eta_{0,i}}\right)^{\gamma-1} \nabla\eta_i(\boldsymbol{q}) & \text{if } \eta_i(\boldsymbol{q}) \leq \eta_{0,i} \\ 0 & \text{if } \eta_i(\boldsymbol{q}) > \eta_{0,i} \end{cases}$$

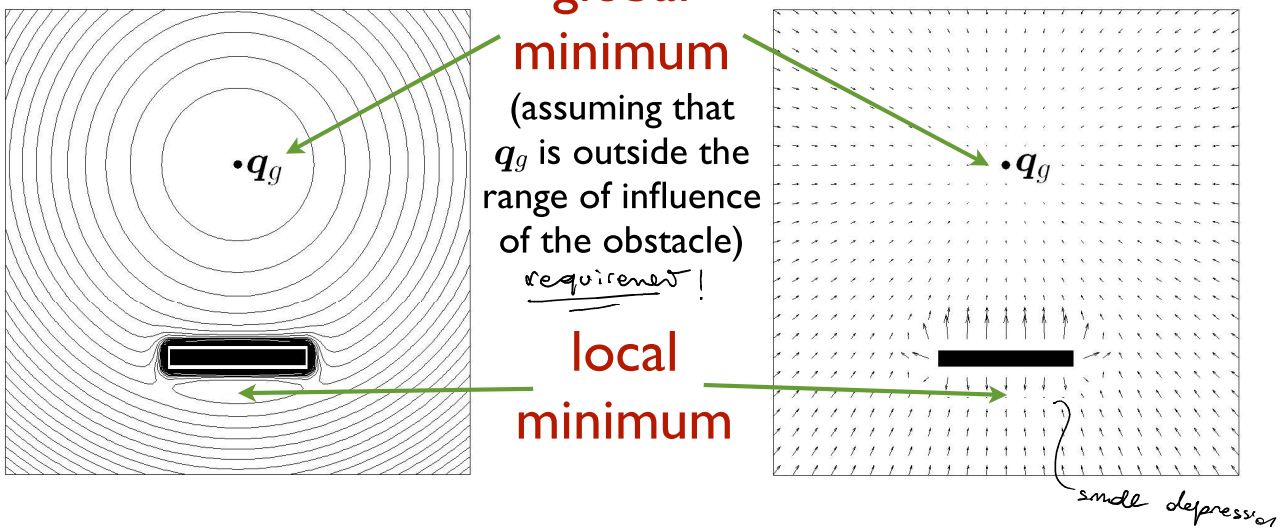
- $\boldsymbol{f}_{r,i}$ is orthogonal to the equipotential contour passing through $\boldsymbol{q}$ and points away from the obstacle
- $\boldsymbol{f}_{r,i}$ is continuous everywhere thanks to the convex decomposition of $\mathcal{CO}$
- aggregate repulsive potential of $\mathcal{CO}$

$$U_r(\boldsymbol{q}) = \sum_{i=1}^{p} U_{r,i}(\boldsymbol{q})$$

total potential

- superposition: $U_t(\boldsymbol{q}) = U_a(\boldsymbol{q}) + U_r(\boldsymbol{q})$
- force field: $\boldsymbol{f}_t(\boldsymbol{q}) = -\nabla U_t(\boldsymbol{q}) = \boldsymbol{f}_a(\boldsymbol{q}) + \sum_{i=1}^{p} \boldsymbol{f}_{r,i}(\boldsymbol{q})$

anti gradient



Local minima

Local minima are a problem, in fact here the robot is subjected to a force $f=0$, and then it stops.

This is caused by the fact that near the obstacles the repulsive force dominates (it goes to ∞), while, sufficiently far from them, the attractive force dominates (as the repulsive one is small or null).

This implies that motion planning based on artificial potential fields method is not complete.

However artificial potential fields method is mainly used for online motion planning, where completeness is often not required.

planning techniques

- three techniques for planning on the basis of f_t → as true force (they are artificial, they don't exist in reality)

1. consider f_t as generalized forces: $\tau = f_t(q)$

   the effect on the robot is filtered by its dynamics (generalized accelerations are scaled)

   a smaller robot will move faster, a larger slower

2. consider f_t as generalized accelerations: $\ddot{q} = f_t(q)$

   the effect on the robot is independent on its dynamics (generalized forces are scaled) and `slow'

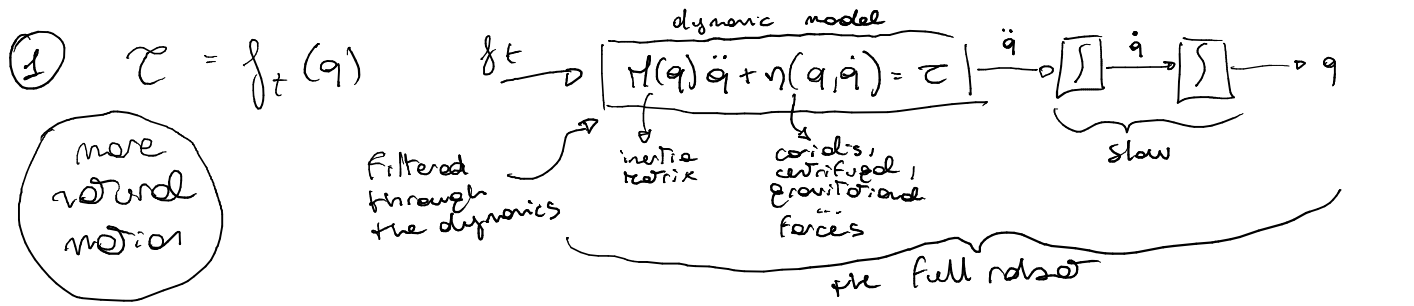
   the opposite of the case 1

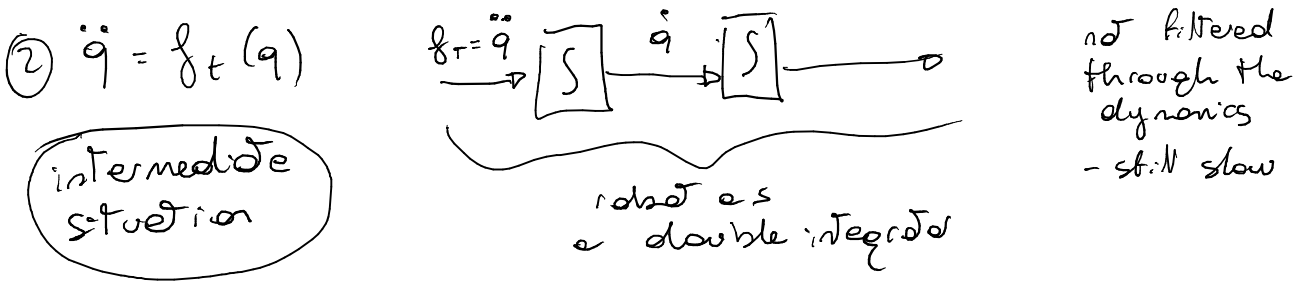
   I'm changing the velocity through the acceleration (integration)

3. consider f_t as generalized velocities: $\dot{q} = f_t(q)$

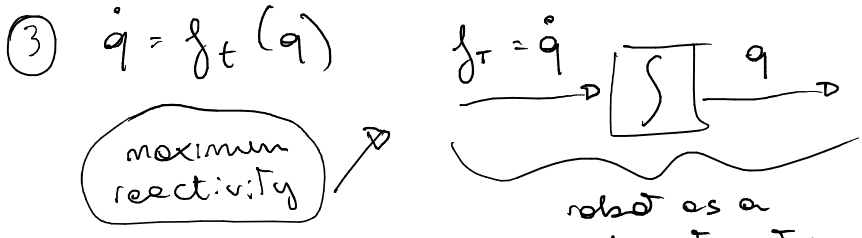
   the effect on the robot is independent on its dynamics (generalized forces are scaled) and `fast'

   ↓ I'm changing immediately the velocity (instantaneously)



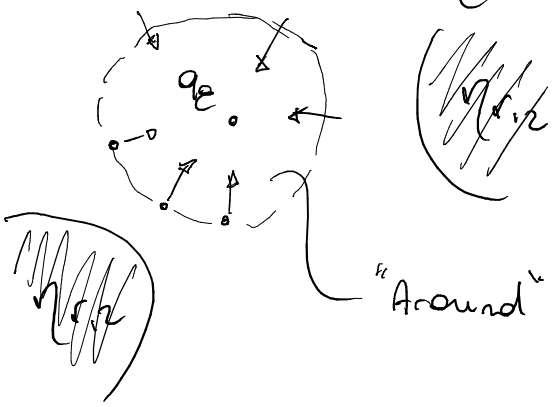


How to obtain τ which permits the q̈ acceleration? Inverting the dynamic model



How to obtain τ which permits q̇? Using the kinematic model

Behavior at the goal (close to the goal)



"Around" the goal $U_t = U_a$

$f_t = k_a e = k_a (q_g - q)$

① $\tau = k_a (q_g - q)$

② $\ddot{q} = k_a (q_g - q)$

③ $\dot{q} = k_a (q_g - q)$

Which one is asymptotically stable? → ③

Why? consider ② → $\ddot{q} = k_a (q_g - q)$ (not AS) # the same proof holds for case ①

$e = q_g - q$ (q_g constant)

$\dot{e} = -\dot{q}$ $\ddot{e} = -k_a e$ $\Rightarrow$ $\dot{x} = \begin{pmatrix} 0 & 1 \\ -k_a & 0 \end{pmatrix} x$

$x = \begin{pmatrix} e \\ \dot{e} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ $\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -k_a x_1 \end{cases}$ A

$\det(\lambda I - A) = \det \begin{pmatrix} \lambda & -1 \\ k_a & \lambda \end{pmatrix} = \lambda^2 + k_a$

$eigs(A) = \{ \pm j\sqrt{k_a} \}$ marginally stable!

- technique 1 generates smoother movements, while technique 3 is faster (irrespective of robot dynamics) in realizing motion corrections; technique 2 gives intermediate results

- strictly speaking, only technique 3 guarantees (in the absence of local minima) asymptotic stability of $\boldsymbol{q}_g$; velocity damping is necessary to achieve the same with techniques 1 and 2

$\tau = f_t(q) + \dots$ $\ddot{q} = f_t(q) + \dots$ → velocity damping term $(k_D \dot{e})$

- off-line planning

  paths in $\mathcal{C}$ are generated by numerical integration of the dynamic model (if technique 1), of $\ddot{\boldsymbol{q}} = \boldsymbol{f}_t(\boldsymbol{q})$ (if technique 2), of $\dot{\boldsymbol{q}} = \boldsymbol{f}_t(\boldsymbol{q})$ (if technique 3)

  the most popular choice is 3 and in particular

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + T\boldsymbol{f}_t(\boldsymbol{q}_k)$$

  sampling time

  i.e., the algorithm of steepest descent → gradient descent

- on-line planning (is actually feedback!)

  technique 1 directly provides control inputs, technique 2 too (via inverse dynamics), technique 3 provides reference velocities for low-level control loops

  the most popular choice is 3 → feedback scheme

local minima: a complication

- if a planned path enters the basin of attraction of a local minimum $\boldsymbol{q}_m$ of U_t, it will reach $\boldsymbol{q}_m$ and stop there, because $\boldsymbol{f}_t(\boldsymbol{q}_m) = -\nabla U_t(\boldsymbol{q}_m) = 0$; whereas saddle points are not an issue
- repulsive fields generally create local minima, hence motion planning based on artificial potential fields is not complete (the path may not reach $\boldsymbol{q}_g$ even if a solution exists)
- workarounds exist but keep in mind that artificial potential fields are mainly used for on-line motion planning, where completeness may not be required

workaround no. 1: best-first algorithm

- build a discretized representation (by defect) of $\mathcal{C}_{\text{free}}$ using a regular grid, and associate to each free cell of the grid the value of U_t at its centroid
- build a tree T rooted at $\boldsymbol{q}_s$: at each iteration, select the leaf of T with the minimum value of U_t and add as children its adjacent free cells that are not in T
- planning stops when $\boldsymbol{q}_g$ is reached (success) or no further cells can be added to T (failure)
- in case of success, build a solution path by tracing back the arcs from $\boldsymbol{q}_g$ to $\boldsymbol{q}_s$

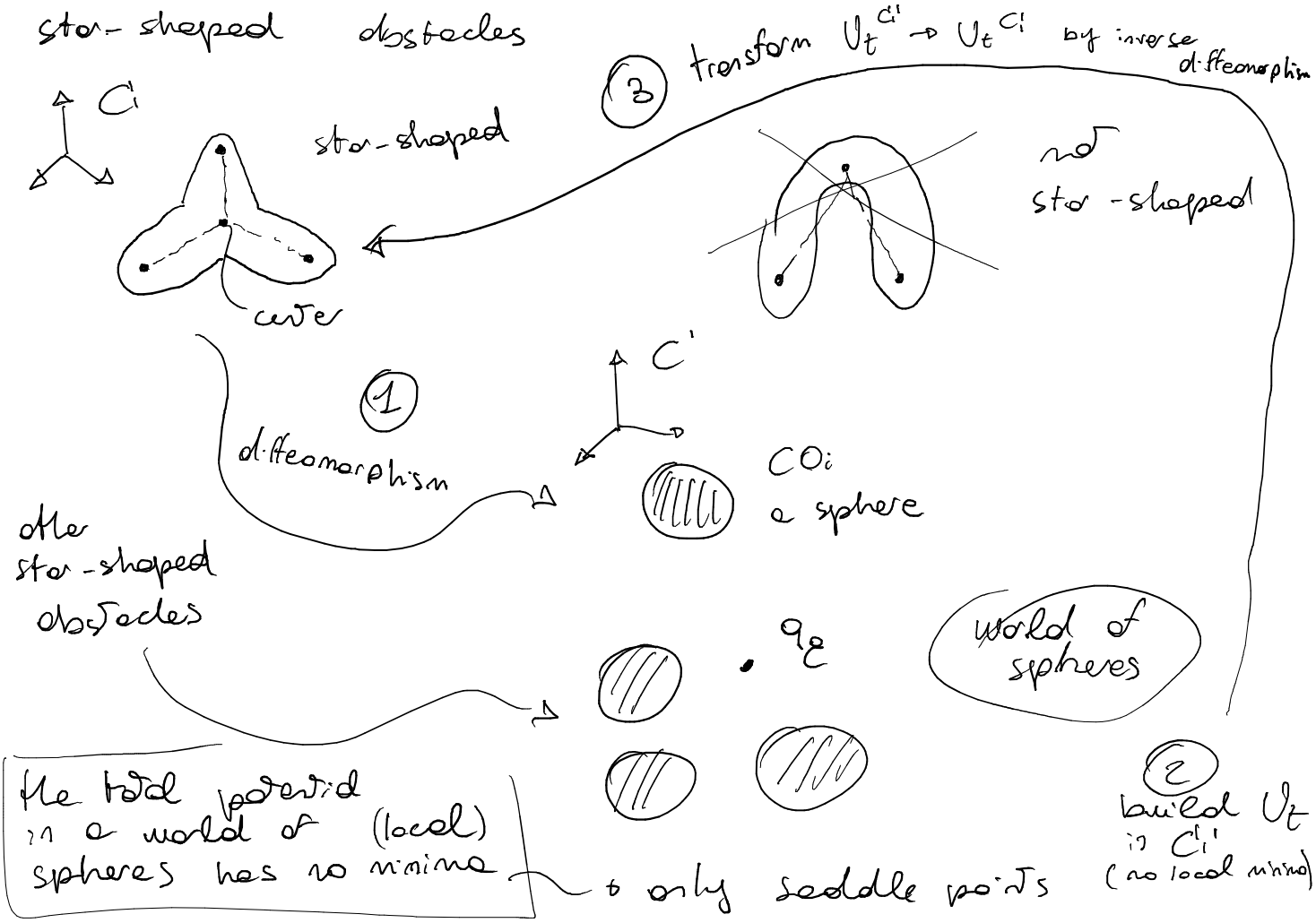
- best-first evolves as a grid-discretized version of steepest descent until a local minimum is met
- at a local minimum, best-first will "fill" its basin of attraction until it finds a way out
- the best-first algorithm is resolution complete
- its complexity is exponential in the dimension of $\mathcal{C}$, hence it is only applicable in low-dimensional spaces
- efficiency improves if random walks are alternated with basin-filling iterations (randomized best-first)

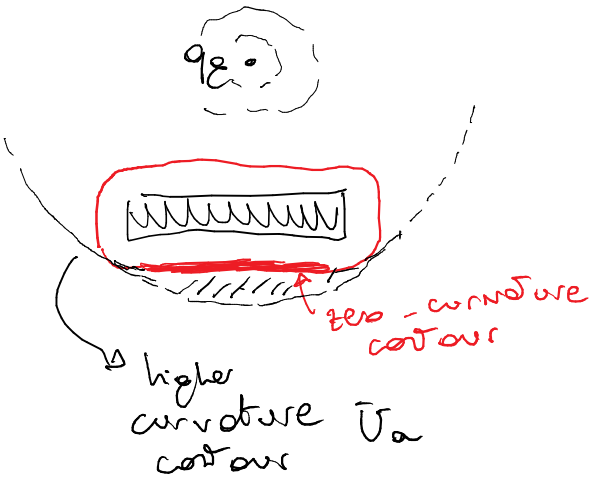
workaround no. 2: navigation functions

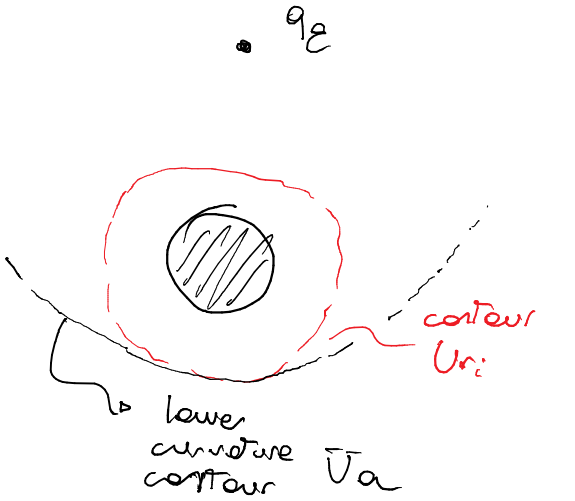
- paths generated by the best-first algorithm are not efficient (local minima are not avoided)
- a different approach: build navigation functions, i.e., potentials without local minima
- if the $\mathcal{C}$-obstacles are star-shaped, one can map $\mathcal{CO}$ to a collection of spheres via a diffeomorphism, build a potential in transformed space and map it back to $\mathcal{C}$
- another possibility is to define the potential as an harmonic function (solution of Laplace's equation)

Laplace's eq. : $\nabla^2 f(x) = 0 \rightarrow \frac{\partial^2 f(x)}{\partial^2 x_1} + \dots + \frac{\partial^2 f(x)}{\partial^2 x_n} = 0$

- all these techniques require complete knowledge of the environment: only suitable for off-line planning







- easy to build: numerical navigation function
- with $\mathcal{C}_{\text{free}}$ represented as a gridmap, assign 0 to start cell, 1 to cells adjacent to the 0-cell, 2 to unvisited cells adjacent to 1-cells, ... (wavefront expansion)

you need to know C_i in advance

2	1	2	3	4	5	6	7	8	9		19
1	0	1			6	7	8	9	10		18
2	1	2	3		7	8		10	11		17
3		3	4	5	6	7	8		12		16
4			5	6	7			12	13		15
5	6	7	6	7	8	9	10	11	12	13	14
6	7	8	7	8	9	10	11	12	13	14	15

start

This procedure is the only one computationally acceptable

goal

solution path:
steepest descent from the goal

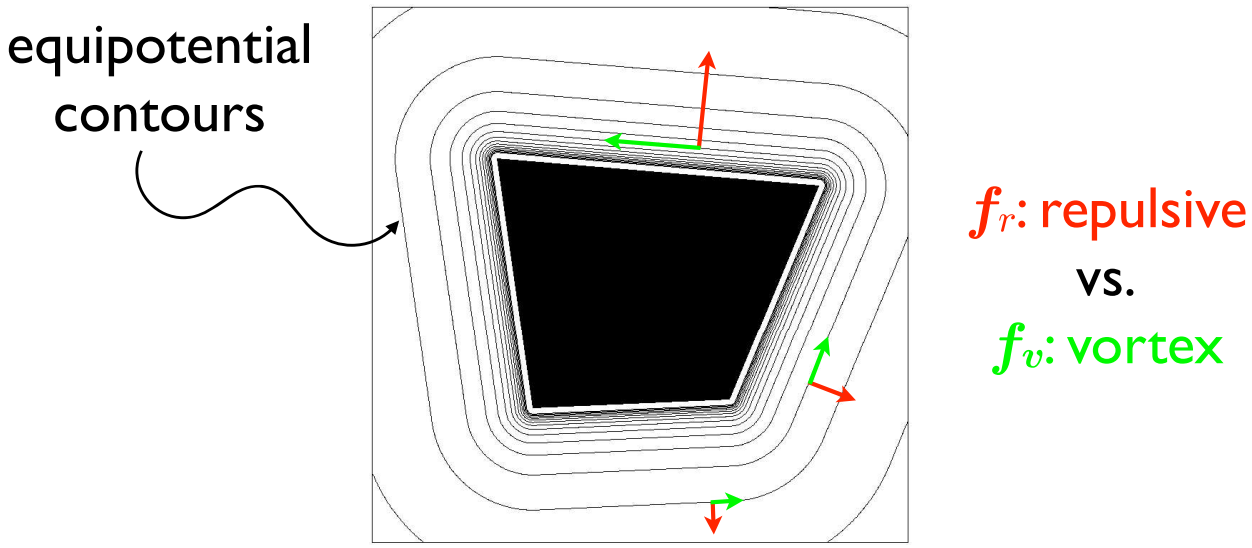
workaround no. 3: vortex fields

- an alternative to navigation functions in which one directly assigns a force field (rather than a potential)
- the idea is to replace the repulsive action (which is responsible for appearance of local minima) with an action forcing the robot to go around the $\mathcal{C}$-obstacle
- e.g., assume $\mathcal{C} = \mathrm{R}^2$ and define the vortex field for $\mathcal{CO}_i$ as

$$\boldsymbol{f}_v = \pm \begin{pmatrix} \frac{\partial U_{r,i}}{\partial y} \\ -\frac{\partial U_{r,i}}{\partial x} \end{pmatrix}$$

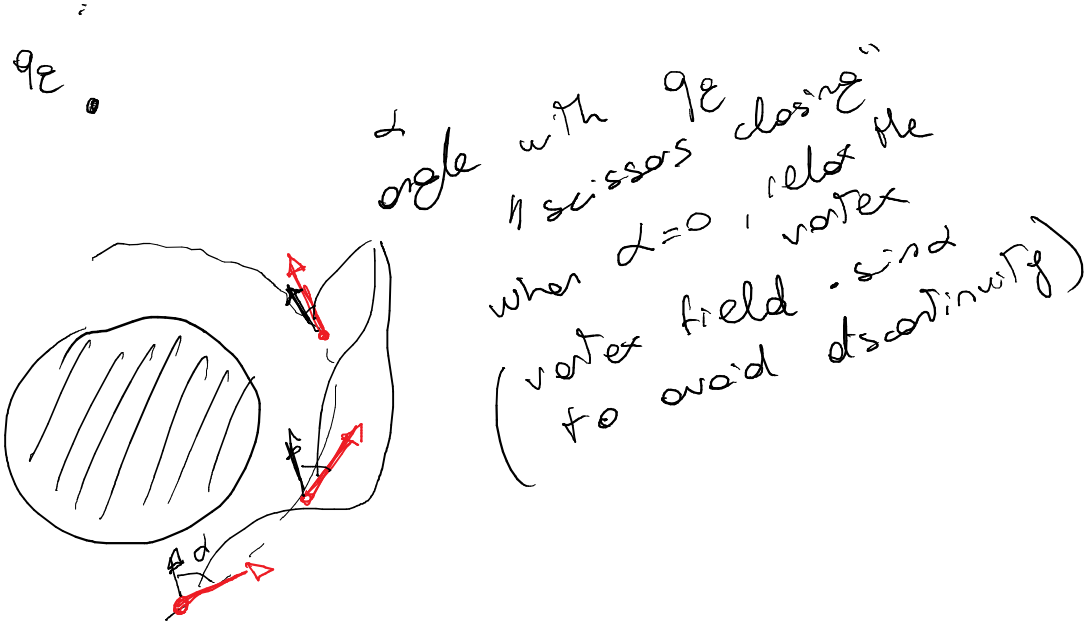
normal vector to the gradient

i.e., a vector which is tangent (rather than normal) to the equipotential contours



- the intensity of the two fields is the same, only the direction changes
- if $\mathcal{CO}_i$ is convex, the vortex sense (CW or CCW) can be always chosen in such a way that the total field (attractive+vortex) has no local minima

Vortex relaxation



- in particular, the vortex sense (CW or CCW) should be chosen depending on the entrance point of the robot in the area of influence of the $\mathcal{C}$-obstacle
- vortex relaxation must be performed so as to avoid orbiting around the obstacle
- both these procedures can be easily performed at runtime based on local sensor measurements
- complete knowledge of the environment is not required: also suitable for on-line planning

artificial potentials for wheeled robots

- since WMRs are typically described by kinematic models, artificial potential fields for these robots are used at the velocity level
- however, robots subject to nonholonomic constraints violate the free-flying assumption
- as a consequence, the artificial force $\boldsymbol{f}_t$ cannot be directly imposed as a generalized velocity $\dot{\boldsymbol{q}}$
- a possible approach: use $\boldsymbol{f}_t$ to generate a feasible $\dot{\boldsymbol{q}}$ via pseudoinversion

- the kinematic model of a WMR is expressed as

$$\dot{\boldsymbol{q}} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{u}$$

- since $\boldsymbol{G}$ is $n \times m$, with $n > m$, it is in general impossible to compute $\boldsymbol{u}$ so as to realize exactly a desired $\dot{\boldsymbol{q}}_{\text{des}}$
- however, a least-squares solution can be used

$$\boldsymbol{u} = \boldsymbol{G}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{q}}_{\text{des}} = \boldsymbol{G}^{\dagger}(\boldsymbol{q})\boldsymbol{f}_t$$

where

$$\boldsymbol{G}^{\dagger}(\boldsymbol{q}) = (\boldsymbol{G}^T(\boldsymbol{q})\boldsymbol{G}(\boldsymbol{q}))^{-1}\boldsymbol{G}^T(\boldsymbol{q})$$

In general a linear system

$n \{ Ax = b \sim n \times 1$; m, p, $p \times 1$

if $p < n$ overconstrained no solution

if $p > n$ underconstrained ∞^{n-p} solutions

if $p = n$ square case 1 solution if $\det A \neq 0$ we have the solution provided that A is non-singular

Overconstrained case

$x = A^{\dagger} b$ ← pseudoinverse

A is a tall matrix

$A^{\dagger} = (A^T A)^{-1} A^T$

$\min e^T e$, where $e = Ax - b$

- as an application, consider the case of a unicycle robot moving in a planar workspace; we have

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \boldsymbol{G}^{\dagger}(\boldsymbol{q}) = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

the least-squares solution corresponding to an artificial force $\boldsymbol{f}_t = (f_{t,x} \;\; f_{t,y} \;\; f_{t,\theta})^T$ is then

$$v = f_{t,x}\cos\theta + f_{t,y}\sin\theta$$
$$\omega = f_{t,\theta}$$

v may be interpreted as the orthogonal projection of the cartesian force $(f_{t,x} \;\; f_{t,y})^T$ on the sagittal axis

- assume that the unicycle robot has a circular shape, so that its orientation is irrelevant for collision

- one may build artificial potentials in a reduced $\mathcal{C}' = \mathrm{R}^2$ with $\mathcal{C}'$-obstacles simply obtained by growing the workspace obstacles by the robot radius

- in $\mathcal{C}'$, the attractive field pulls the robot towards (x_g, y_g) while repulsive fields push it away from $\mathcal{C}'$-obstacle boundaries (segments and arcs of circle)

- since $\mathcal{C}'$ does not contain the orientation, the total field will not include a component $f_{t,\theta}$

- this degree of freedom can be exploited by letting

$$\omega = f_{t,\theta} = k_\theta \left(\mathrm{atan2}(f_{t,y}, f_{t,x}) - \theta \right)$$

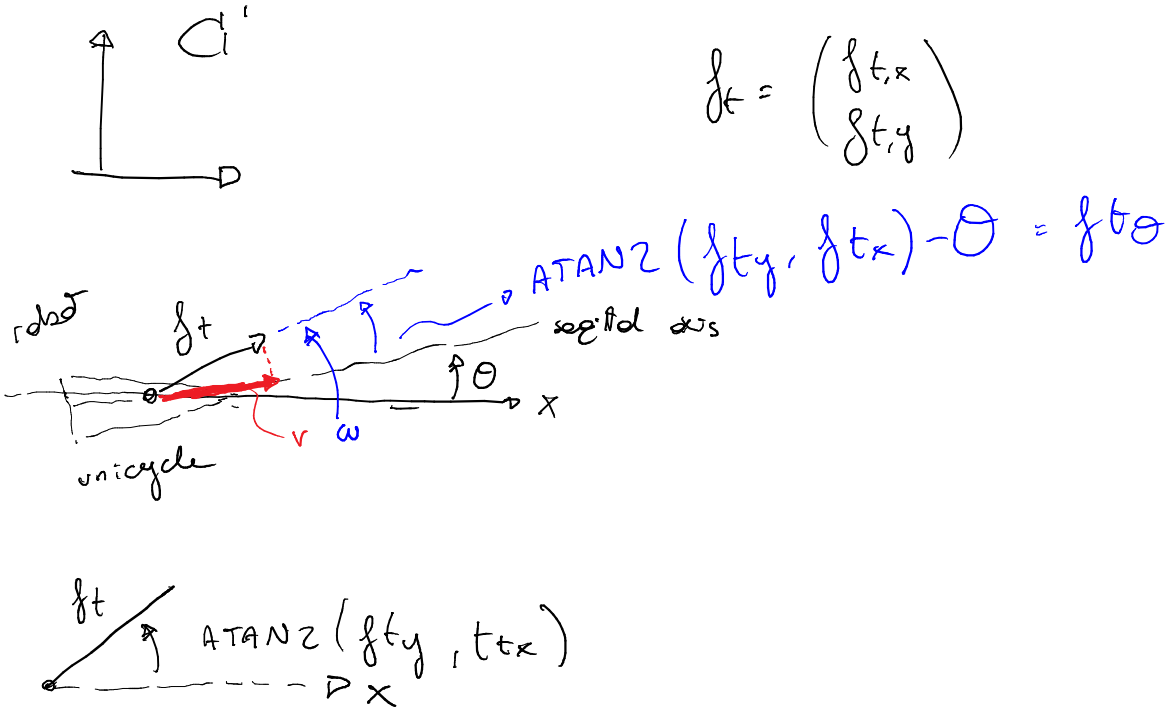
whose rationale is to force the unicycle to align with the total field, so that $\boldsymbol{f}_t$ can be better reproduced

- overall, a feedback control scheme is obtained where v and ω are computed in real time from $\boldsymbol{f}_t$

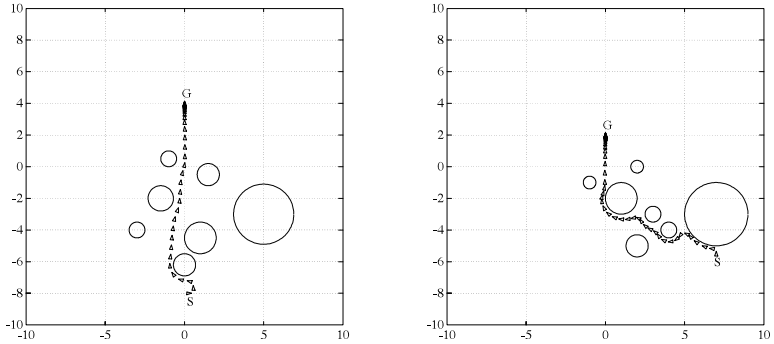
- assume w.l.o.g. $(x_g, y_g) = (0,0)$; close to the goal, where $\boldsymbol{f}_t = \boldsymbol{f}_a$, the controls become

$$v = -k_a (x \cos\theta + y \sin\theta)$$
$$\omega = k_\theta \left(\mathrm{Atan2}(-y, -x) - \theta \right)$$

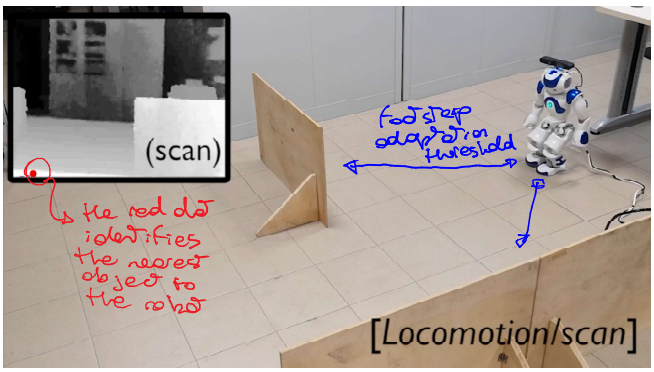
i.e., a cartesian regulator! (see slides Wheeled Mobile Robots 5)



- results on unicycle (using vortex fields)



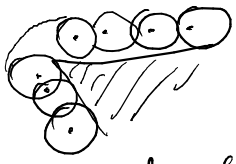
- can be applied to robots moving unicycle-like



Considerations on unicycle

If the unicycle is circular then there's not a defined orientation. This simplify the operations:

- $C = W = \mathbb{R}^2$
- Building CO_i is very simple:
  - Along the edge of the obstacle we extend them with the measure of the radius of the robot
  - At the vertex we interpolates the cycles on the unit with an arc of circle



- We build a bidimensional force field $\begin{pmatrix} f_x \\ f_y \end{pmatrix}$; this force is seen as the Cartesian velocity $\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$ to be imposed to the robot through the actuators, leaving $\dot{\theta}$ free.

This degree of freedom is useful to define f_θ differently, in order to face the unicycle to align with the total field.

In this way, when we are sufficiently far from the obstacles, and the repulsive force is null, the control law is the same of a Cartesian regulator.

motion planning for robot manipulators

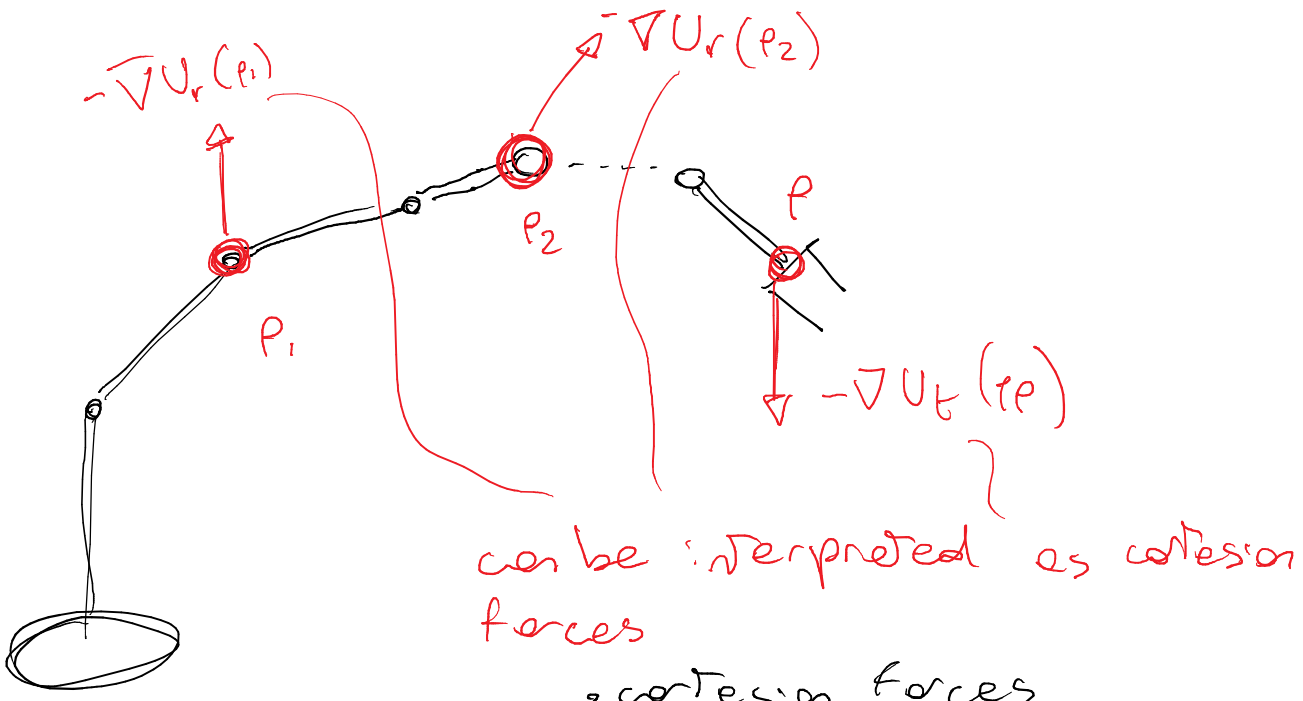
- complexity of motion planning is high, because the configuration space has dimension typically ≥ 4
- try to reduce dimensionality: e.g., in 6-dof robots, replace the wrist with the total volume it can sweep (a conservative approximation)
- both the construction and the shape of $\mathcal{CO}$ are complicated by the presence of revolute joints
- off-line planning: probabilistic methods are the best choice (although collision checking is heavy)
- on-line planning: adaptation of artificial potential fields

artificial potentials for robot manipulators

- to avoid the computation of $\mathcal{CO}$ and the "curse of dimensionality", the potential is built in $\mathcal{W}$ (rather than in $\mathcal{C}$) and acts on a set of control points $\boldsymbol{p}_1,...,\boldsymbol{p}_P$ distributed on the robot body

- in general, control points include one point per link ($\boldsymbol{p}_1,...,\boldsymbol{p}_{P\text{-}1}$) and the end-effector (to which the goal is typically assigned) as $\boldsymbol{p}_P$

- the attractive potential U_a acts on $\boldsymbol{p}_P$ only, while the repulsive potential U_r acts on the whole set $\boldsymbol{p}_1,...,\boldsymbol{p}_P$; hence, $\boldsymbol{p}_P$ is subject to the total $U_t = U_a + U_r$



can be interpreted as cartesian forces

→ cartesian forces

$\Rightarrow \quad \tau = J^T(q) F \quad \Rightarrow$

torques

$P-1$ → Jacobian J part p_i → Jacobian at the end-effector

$$\tau = -\sum_{i=1} J_i^T(q) \nabla U_r(p_i) - J_e^T \nabla U_t(p_P)$$

- two techniques for planning with control points:

  1. impose to the robot joints the generalized forces resulting from the combined action of force fields

$$\boldsymbol{\tau} = -\sum_{i=1}^{P-1} \boldsymbol{J}_i^T(\boldsymbol{q}) \nabla U_r(\boldsymbol{p}_i) - \boldsymbol{J}_P^T(\boldsymbol{q}) \nabla U_t(\boldsymbol{p}_P)$$

     where $\boldsymbol{J}_i(\boldsymbol{q}), i = 1,\ldots,P$, is the Jacobian matrix of the direct kinematics function associated to $\boldsymbol{p}_i(\boldsymbol{q})$

  2. use the above expression as reference velocities to be fed to the low-level control loops (actuators)

$$\dot{\boldsymbol{q}} = -\sum_{i=1}^{P-1} \boldsymbol{J}_i^T(\boldsymbol{q}) \nabla U_r(\boldsymbol{p}_i) - \boldsymbol{J}_P^T(\boldsymbol{q}) \nabla U_t(\boldsymbol{p}_P)$$

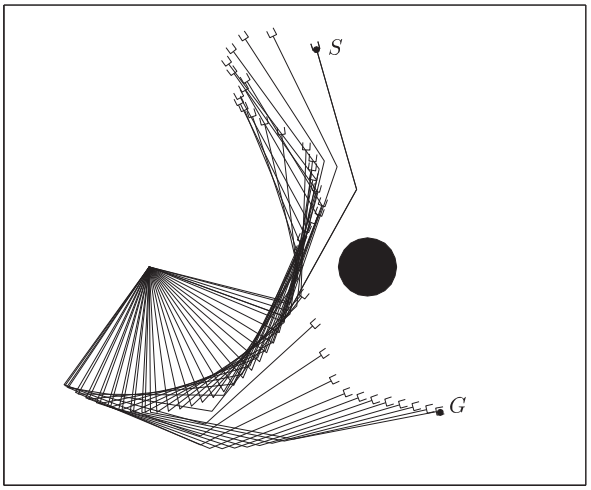
- technique 2 is actually a gradient-based minimization step in $\mathcal{C}$ of a combined potential in $\mathcal{W}$; in fact

$$\nabla_{\boldsymbol{q}} U(\boldsymbol{p}_i) = \left(\frac{\partial U(\boldsymbol{p}_i(\boldsymbol{q}))}{\partial \boldsymbol{q}} \right)^T = \left(\frac{\partial U(\boldsymbol{p}_i)}{\partial \boldsymbol{p}_i} \frac{\partial \boldsymbol{p}_i}{\partial \boldsymbol{q}} \right)^T = \boldsymbol{J}_i^T(\boldsymbol{q}) \nabla U(\boldsymbol{p}_i)$$

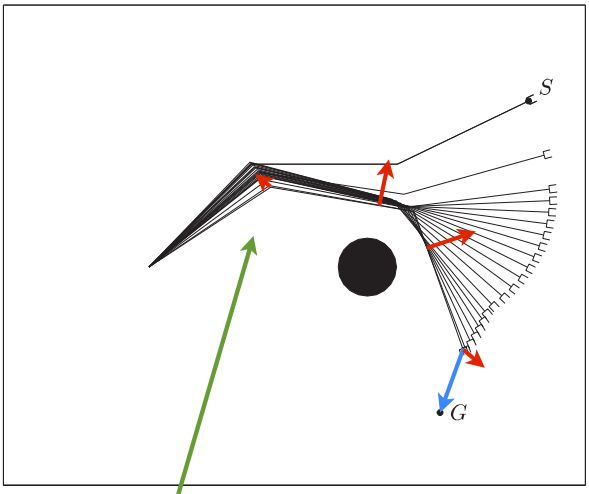
- technique 1 generates smoother movements, while technique 2 is faster (irrespective of robot dynamics) in realizing motion corrections → see technique 1 & 3 of slide 16

- both can stop at force equilibria, where the various forces balance each other even if the total potential U_t is not at a local minimum; hence, this method should be used in conjunction with a best-first algorithm

success
(with vortex field and
folding heuristic for cw/ccw sense)



failure
(with repulsive field)



a force equilibrium
between attractive and repulsive forces

Find considerations about motion planning

Roadmap methods build C_{free} and CO exactly (more or less).

Probabilistic methods build an extremely approximated version of C_{free} (e.g. the roadmap) and they don't build CO: they compute $B(q)$ volume to check the collisions.

Roadmap method and probabilistic method require an a priori knowledge of the geometry and poses of the obstacles, while the artificial potential method don't; therefore the latter is used for online planning.

→ Mobile planning for manipulators:
$\dim(C) \geq 4$, difficult computation of CO due to the revolute joints.
- Offline planning: probabilistic method
- Online planning: artificial potential field method