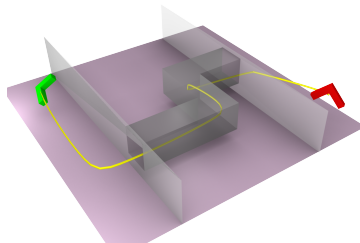
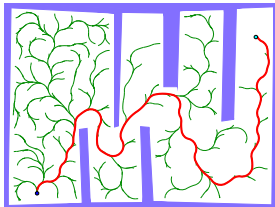
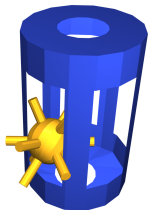


# Motion planning: basic concepts

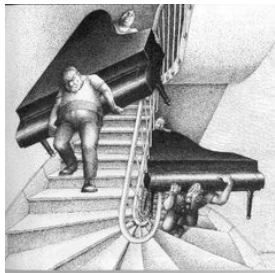
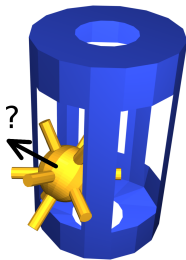
**Vojtěch Vonásek**

Department of Cybernetics  
Faculty of Electrical Engineering  
Czech Technical University in Prague



**Informal definition:** Motion planning is about automatic finding of ways how to move an object (robot) while avoiding obstacles (and considering other constraints).

- “Piano mover’s problem”
- Studied also in other fields
- Robotics uses results from other research fields
  - Mathematics: graph theory & topology
  - Computational geometry: collision detection
  - Computer graphics: visualizations
  - Control theory: feedback controllers required to navigate along paths
- Many applications in robotics and even beyond



## World $\mathcal{W}$

- is space where the robot operates
- $\mathcal{W}$  is usually  $\mathcal{W} \subseteq \mathbf{R}^2$  or  $\mathcal{W} \subseteq \mathbf{R}^3$
- $\mathcal{O} \subseteq \mathcal{W}$  are obstacles

## Robot $\mathcal{A}$

- $\mathcal{A}$  is the geometry of the robot
- $\mathcal{A} \subseteq \mathbf{R}^2$  (or  $\mathcal{A} \subseteq \mathbf{R}^3$ )
- or set of links  $\mathcal{A}_1, \dots, \mathcal{A}_n$  for  $n$ -body robot

## Configuration $q$

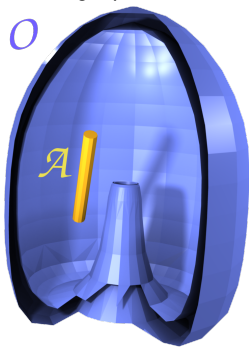
- Specifies position of **every** point of  $\mathcal{A}$  in  $\mathcal{W}$
- Usually a vector of **Degrees of freedom (DOF)**

$$q = (q_1, q_2, \dots, q_n)$$

## Configuration space $\mathcal{C}$ (aka C-Space or $\mathcal{C}$ -space)

- $\mathcal{C}$  is a set of **all** possible configurations

3D Bugtraj benchmark



$$\mathcal{W} \subseteq \mathbf{R}^3, \mathcal{A} \subseteq \mathbf{R}^3$$

$$\mathcal{O} \subseteq \mathbf{R}^3$$

$(x, y, z)$  is 3D position

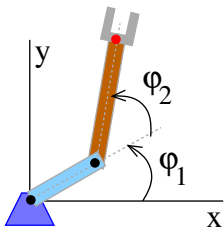
$(r_x, r_y, r_z)$  is 3D rotation

$$q = (x, y, z, r_x, r_y, r_z)$$

$\mathcal{C}$ -space is 6D

- A configuration is a **point** in  $\mathcal{C}$
- $\mathcal{A}(q)$  is set of **all points** of the robot determined by configuration  $q \in \mathcal{C}$
- Therefore, point  $q \in \mathcal{C}$  **fully** describes how the robot looks in  $\mathcal{W}$
- $\mathcal{C}$  has as many dimensions as robot's DOFs
- $\mathcal{C}$  is considered “high-dimensional” if number of DOFs  $> 4$

**Example:** a robotic arm with two revolute joints;  $q = (\varphi_1, \varphi_2) \rightarrow 2D$   $\mathcal{C}$ -space  
Robot geometry has two rigid shapes:  $\mathcal{A}_1$  and  $\mathcal{A}_2$



## Obstacles in the configuration space: $\mathcal{C}_{\text{obs}}$

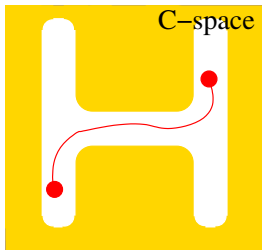
$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}, \quad \mathcal{C}_{\text{obs}} \subseteq \mathcal{C}$$

- $\mathcal{C}_{\text{obs}}$  contains robot-obstacle collisions and self-collisions
- Self-collisions: robotic arms, tethered robots
- Collision-free configurations

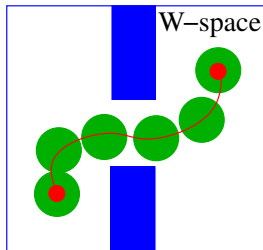
$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$$

- $q \in \mathcal{C}_{\text{free}} \rightarrow q$  is feasible

- A **path** in  $\mathcal{C}$  is a continuous curve connecting two configurations  $q_{\text{init}}$  and  $q_{\text{goal}}$ :  
$$\tau : s \in [0, 1] \rightarrow \tau(s) \in \mathcal{C}; \quad \tau(0) = q_{\text{init}} \text{ and } \tau(1) = q_{\text{goal}}$$
- A **trajectory** is a path parameterized by time  
$$\tau : t \in [0, T] \rightarrow \tau(t) \in \mathcal{C}$$
- Trajectory/path defines motion in workspace



Path in  $\mathcal{C}$



Workspace motion

## Given

- model of the world  $\mathcal{W}$  and robot  $\mathcal{A}$
- start  $q_{\text{init}} \in \mathcal{C}_{\text{free}}$
- goal region  $\mathcal{C}_{\text{goal}} \subseteq \mathcal{C}_{\text{free}}$

## Path planning

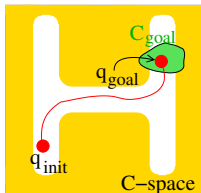
- To find a collision-free path  $\tau(s)$  from  $q_{\text{init}}$  to  $\mathcal{C}_{\text{goal}}$
- i.e.,  $q(s) \in \mathcal{C}_{\text{free}}$  for all  $s \in [0, 1]$ ,  $s(0) = q_{\text{init}}$ ,  $s(1) \in \mathcal{C}_{\text{goal}}$

## Motion planning

- To find a collision-free trajectory  $\tau(t)$  from  $q_{\text{init}}$  to  $\mathcal{C}_{\text{goal}}$
- i.e.,  $q(t) \in \mathcal{C}_{\text{free}}$  for all  $t \in [0, T]$ ,  $s(0) = q_{\text{init}}$ ,  $s(T) \in \mathcal{C}_{\text{goal}}$

## Notes

- The above definition is considered as **feasible path/motion planning**
- Using  $\mathcal{C}_{\text{goal}}$  instead of single  $q_{\text{goal}} \in \mathcal{C}_{\text{free}}$  is more practical
- No optimality criteria is considered



## Completeness

- Algorithm is complete, if for any input it correctly reports in **finite time if there is a solution or no**
- If a solution exists, it **must** return one **in a finite time**
- Computationally very hard (P-Space complete)
- Complete methods exist only for low-dimensional problems

## Probabilistic completeness

- Algorithm is prob. complete if for scenarios with an existing solution the probability of finding that solution converges to one
- If solution does not exist, the method can run forever

## Optimal vs. non-optimal

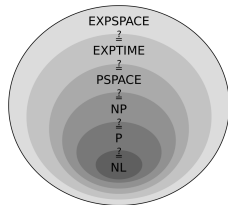
- Optimal planning: algorithm ensures finding of the optimal solution (according to a criterion)
- Non-optimal: any feasible solution is returned

## Asymptotically optimal

- With increasing runtime, a solution provided by the algorithm converges to the optimal solution

## Configuration space

- “Converts” planning tasks to a search of path for a **point** in  $\mathcal{C}$
- Once we can search  $\mathcal{C}$ , we can solve any planning problem
- Motion planning is P-Space complete!



## Why is planning so difficult?

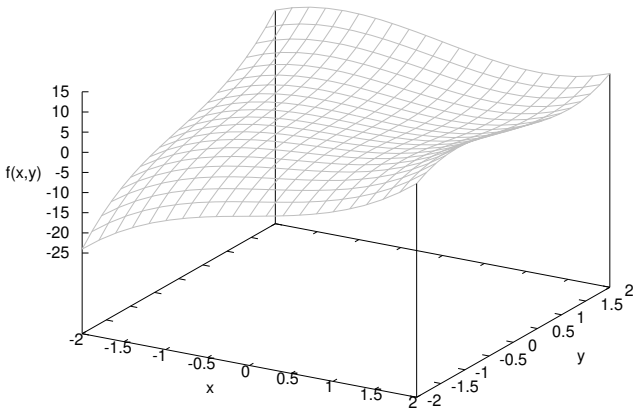
- Because we have to explicitly know  $\mathcal{C}$ ,  $\mathcal{C}_{\text{obs}}$  and  $\mathcal{C}_{\text{free}}$
- The most important are obstacles  $\mathcal{C}_{\text{obs}}$ , but they are given implicitly:

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}, \quad \mathcal{C}_{\text{obs}} \subseteq \mathcal{C}$$

- Implicit definition does not allow to enumerate points in  $\mathcal{C}_{\text{obs}}$
- Difficult to determine the nearest colliding configuration

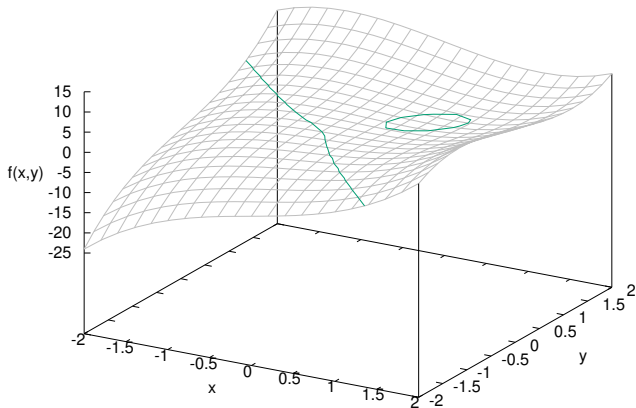
• J. Canny. The complexity of robot motion planning. MIT press, 1988.

$$f(x, y) = x^3 - 2xy + y^3$$



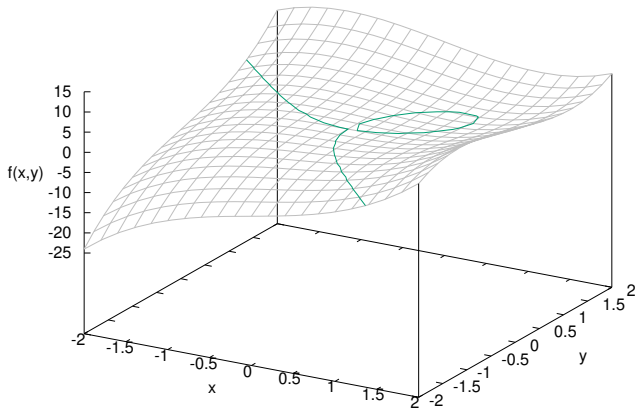
$f(x, y)$

$$f(x, y) = x^3 - 2xy + y^3$$



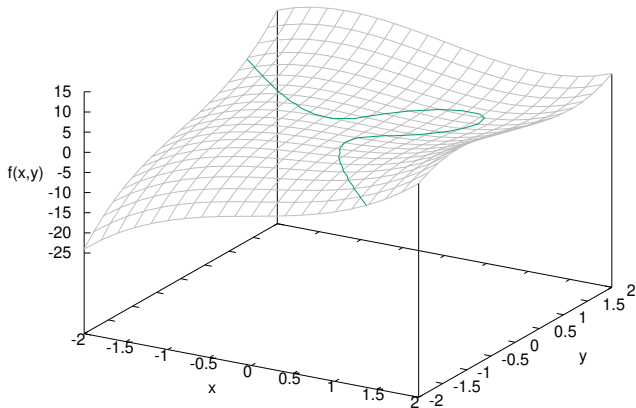
$$f(x, y) = -0.1$$

$$f(x, y) = x^3 - 2xy + y^3$$



$$f(x, y) = 0$$

$$f(x, y) = x^3 - 2xy + y^3$$



$$f(x, y) = 0.1$$

- How to get explicit list of obstacles from the implicit obstacles

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}, \quad \mathcal{C}_{\text{obs}} \subseteq \mathcal{C}$$

- i.e., how to enumerate points on the border of the obstacles?

## Explicit construction of $\mathcal{C}_{\text{obs}}$

- $\mathcal{A}(0)$  is the robot at origin
- $-\mathcal{A}(0)$  is achieved by replacing all  $x \in \mathcal{A}(0)$  by  $-x$
- Obstacles in  $\mathcal{C}$  are determined by the Minkowski sum

$$\mathcal{C}_{\text{obs}} = \mathcal{O} \oplus -\mathcal{A}(0)$$

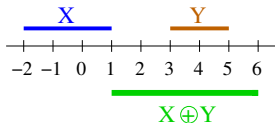
- Theoretical principle, not used in practise (you will see why)

Minkowski sum  $\oplus$  of two sets  $X, Y \subset \mathbb{R}^n$  is

$$X \oplus Y = \{x + y \in \mathbb{R}^n \mid x \in X \text{ and } y \in Y\}$$

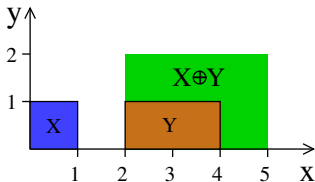
**1D example:**  $X = [-2, 1]$ ,  $Y = [3, 5]$

$$X \oplus Y = [1, 6]$$



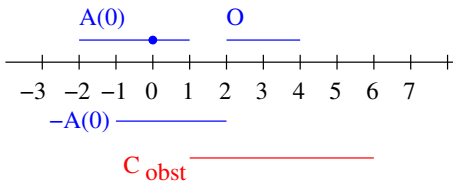
**2D example:**  $X = [0, 1] \times [0, 1]$ ,  $Y = [2, 4] \times [0, 1]$

$$X \oplus Y = [2, 5] \times [0, 2]$$



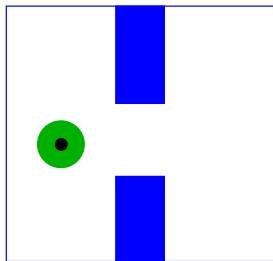
**Example:** 1D robot  $\mathcal{A} = [-2, 1]$  and obstacle  $\mathcal{O} = [2, 4]$ :

$$\mathcal{C}_{\text{obs}} = \mathcal{O} \oplus -\mathcal{A}(0)$$

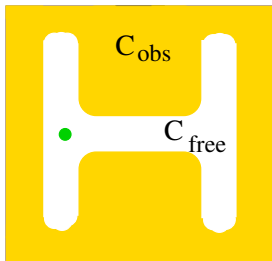


$$\mathcal{C}_{\text{obs}} = [1, 6]$$

- 2D workspace  $\mathcal{W} \subseteq \mathbf{R}^2$
- 2D disc robot  $\mathcal{A} \subseteq \mathbf{R}^2$ , reference point in the disc's center
- We assume **only translation**
- Therefore, configuration  $q = (x, y)$  and  $\mathcal{C}$  is 2D



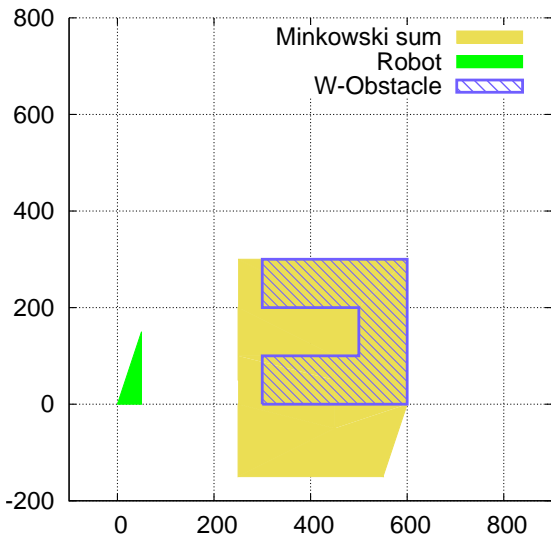
Workspace



Configuration space

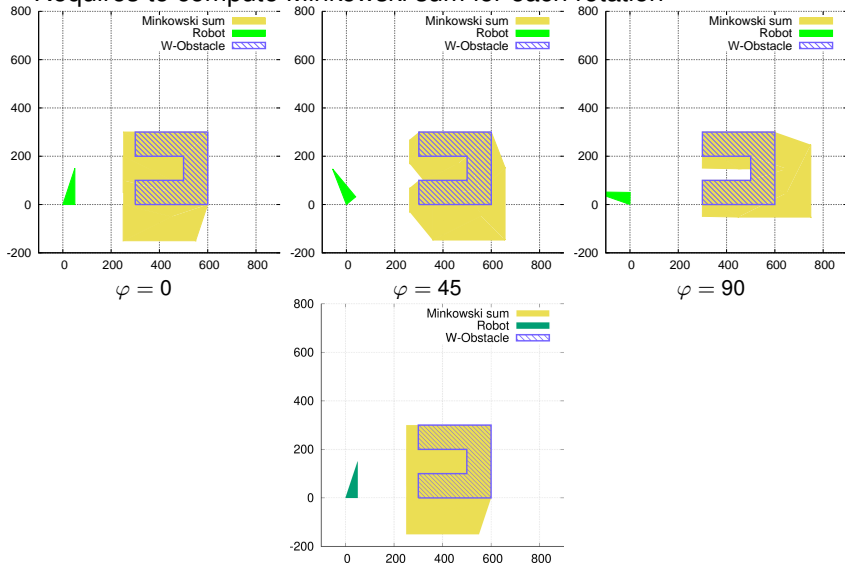
- All  $q \in \mathcal{C}_{\text{free}}$  are collision-free  $\rightarrow \mathcal{A}(q) \cap \mathcal{O} = \emptyset$
- Volume of  $\mathcal{C}_{\text{free}}$  depends both on the robot and obstacles
- What happens if the robot is a point?

- 2D robot, only translation,  $q = (x, y) \rightarrow 2D \mathcal{C}$



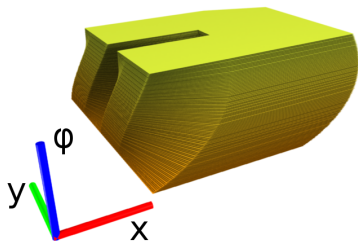
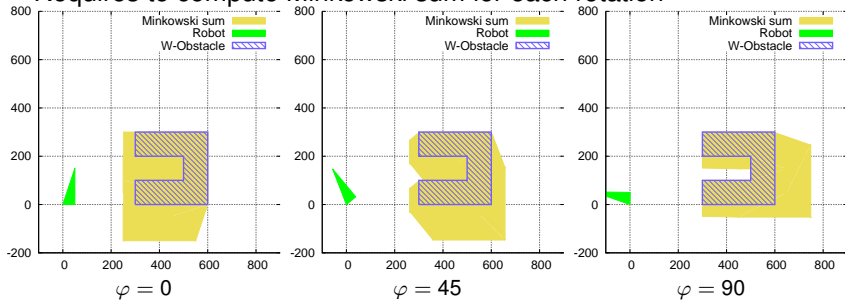
# Configuration space: 2D robot II

- 2D robot, translation + rotation,  $q = (x, y, \varphi) \rightarrow 3D \mathcal{C}$
- Requires to compute Minkowski sum for each rotation



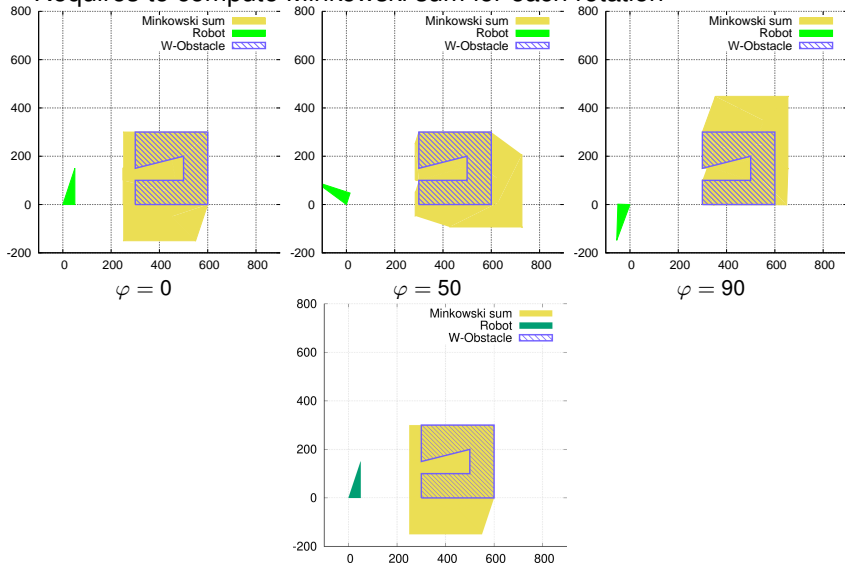
# Configuration space: 2D robot II

- 2D robot, translation + rotation,  $q = (x, y, \varphi) \rightarrow 3D \mathcal{C}$
- Requires to compute Minkowski sum for each rotation



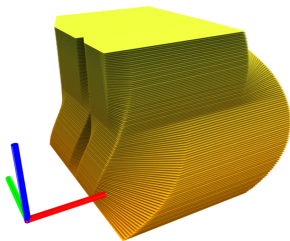
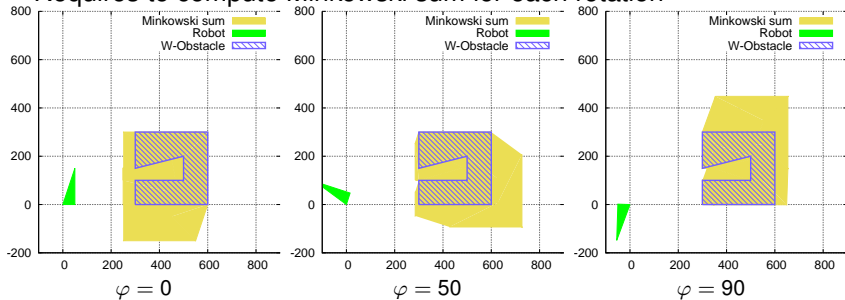
# Configuration space: 2D rotating robot III

- 2D robot, translation + rotation,  $q = (x, y, \varphi) \rightarrow 3D \mathcal{C}$
- Requires to compute Minkowski sum for each rotation



# Configuration space: 2D rotating robot III

- 2D robot, translation + rotation,  $q = (x, y, \varphi) \rightarrow 3D \mathcal{C}$
- Requires to compute Minkowski sum for each rotation



Minkowski sum of two objects of  $n$  and  $m$  complexity

## 2D polygons

- convex  $\oplus$  convex,  $O(m + n)$
- convex  $\oplus$  arbitrary,  $(mn)$
- arbitrary  $\oplus$  arbitrary,  $(m^2n^2)$

## 3D polyhedrons

- convex  $\oplus$  convex,  $O(mn)$
- arbitrary  $\oplus$  arbitrary,  $(m^3n^3)$

- Construction of  $\mathcal{C}$  Minkowski sums is straightforward, but . . .
- We have only 2D/3D models of robots and obstacles
- directly we can construct  $\mathcal{C}$  only for “translation only” systems
- Other DOFS need to be discretized and Minkowski sum computed for each combination (!)
- Explicit construction of  $\mathcal{C}$  is computationally demanding!
- Not practical for high-dimensional systems
- Explicit construction of  $\mathcal{C}_{\text{obs}}$  using Minkowski sum is (generally) too difficult, and it is not practically used.

## Robots (usually) cannot move arbitrarily

- Kinematic constraints (e.g. 'car-like' vehicle)
- Dynamic constraints (e.g. maximal acceleration)
- Task constraints (e.g. 'do not spill the beer')
- These are considered as additional constraints that must be satisfied in path/motion planning

## Motion model

- describes how the robot's state changes when input  $u \in \mathcal{U}$  is applied at  $q \in \mathcal{C}$
- $\mathcal{U}$  is a set of all possible inputs

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

- Discrete version is often used:

$$q_{k+1} = f(q_k, u), \quad q_{k+1}, q_k \in \mathcal{C}, u \in \mathcal{U}$$

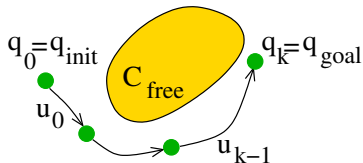
## Given

- model of the world  $\mathcal{W}$  and robot  $\mathcal{A}$ , configurations  $q_{\text{init}}, q_{\text{goal}} \in \mathcal{C}_{\text{free}}$
- motion model  $q' = f(q, u)$  with inputs  $\mathcal{U}$

## Discrete feasible planning

- Find a finite sequence of actions  $\pi_k = (u_0, \dots, u_{k-1}), u \in \mathcal{U}$  such that

$$\begin{aligned}q_{k+1} &= f(q_k, u_k) \\q_0 &= q_{\text{init}} \\q_k &= q_{\text{goal}} \\q_k &\in \mathcal{C}_{\text{free}}\end{aligned}$$



- The sequence of states  $(q_1, \dots, q_k)$  can be derived from the motion model starting from  $q_0$  and applying  $q_{k+1} = f(q_k, u_k)$  subsequently
- Is this plan optimal?

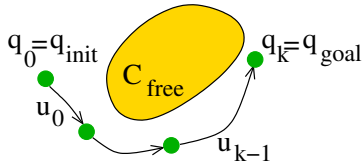
- Let  $L(\pi_k)$  is the cost of the sequence  $\pi_k = (u_0, \dots, u_{k-1})$

$$L(\pi_k) = l_f(q_k) + \sum_{i=0}^{k-1} l(q_i, u_i)$$

- the final term  $l_f(q_k) = 0$  if  $q_k = q_{\text{goal}}$ ; it is  $\infty$  otherwise

## Discrete optimal planning

$$\begin{aligned} & \underset{\pi_k=(u_0, \dots, u_{k-1})}{\text{minimize}} && L(\pi_k) \\ & \text{subject to} && q_{k+1} = f(q_k, u_k) \\ & && q_0 = q_{\text{init}} \\ & && q_k = q_{\text{goal}} \\ & && q_k \in \mathcal{C}_{\text{free}} \end{aligned}$$



- $L(\pi_k) = \infty$  means infeasible solution
- $L(\pi_k) < \infty$  means a feasible solution with the cost  $L(\pi_k)$

- Optimal control for a discrete-time (and finite horizon)
- initial state is  $x_i$ , goal state  $x_n$  may be given (or not)

$$\underset{u_i, \dots, u_{N-1}, (x_i), \dots, x_n}{\text{minimize}} \quad \left( \phi(x_n, N) + \sum_{k=i}^{N-1} L_k(x_k, u_k) \right)$$

$$\begin{aligned} \text{subject to} \quad & x_{k+1} = f_k(x_k, u_k) \\ & u_{lb} \leq u_k \leq u_{ub} \\ & x_{lb} \leq x_k \leq x_{ub} \end{aligned}$$

## Discrete optimal control (generally)

$$\underset{x \in \mathbf{R}^{n(N-i)}, u \in \mathbf{R}^{m(N-i)}}{\text{minimize}} \quad J(x, u)$$

$$\begin{aligned} \text{subject to} \quad & g(x, u) = 0 \\ & h(x, u) \leq 0 \end{aligned}$$

equations by Z. Hurak: Discrete-time optimal control — direct approach (lectures notes of ORR)

- Optimal control and optimal (path/motion) planning are (generally) the same
- Both can find path/trajectory from start to goal
- What is the practical difference?

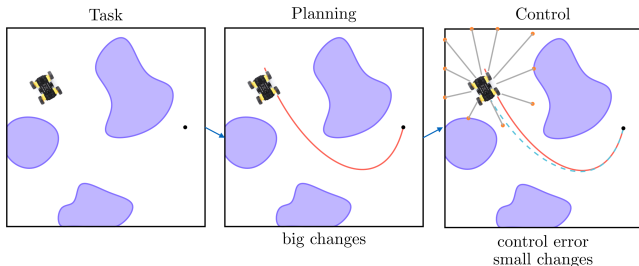
## Path planning

- Solution is achieved by searching  $\mathcal{C}$ -space
- Can work with explicit (combinatorial planning) or implicit obstacles (sampling-based planning)
- Difficult to react on changes (robot control error, dynamic obstacles) → replanning
- Replanning requires to solve the problem from scratch → slow

- Optimal control and optimal (path/motion) planning are (generally) the same
- Both can find path/trajectory from start to goal
- What is the practical difference?

## Control

- Trajectory is achieved via mathematical optimization
  - we (typically) need “a gradient”  $\rightarrow$ , e.g. ‘distance to the nearest obstacle’, its derivative etc.
  - this requires an explicit representation of  $\mathcal{C}$  resp.  $\mathcal{C}_{\text{obs}}$
- Difficult to find first (feasible) solution  $\rightarrow$  large search space
- Suitable for following reference, e.g. reference trajectory from motion planning



- Global plan delivered by motion planning
- Sensing (actual position, speed, etc.) controlled along planned path
- i.e., errors in actuation are handled by control
- Replanning when global change occurs (e.g. new obstacle that cannot be handled by control)

**Does not make sense to solve motion plan by control-theory methods**

**Does not make sense to control via planning!**

- Path/motion planning are studied in several disciplines
  - Robotics, computation geometry, mathematics, biology
  - ... since 1950's !
- Each field uses different meaning for “path” and “trajectory”  
... and different meaning for path/motion planning
- this continues up to now

## What is a “trajectory”?

- Robotics (including this lecture): path + time
- Control-oriented part of robotics: path + time + control inputs
- Computational biology: 3D path of atom(s) (with or without time)

**Before you start to solve a planning problem, define (or agree on) the basic terms first!**

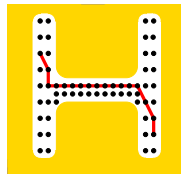
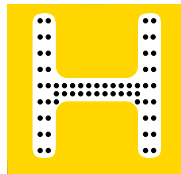
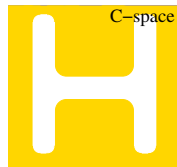
Continuous space

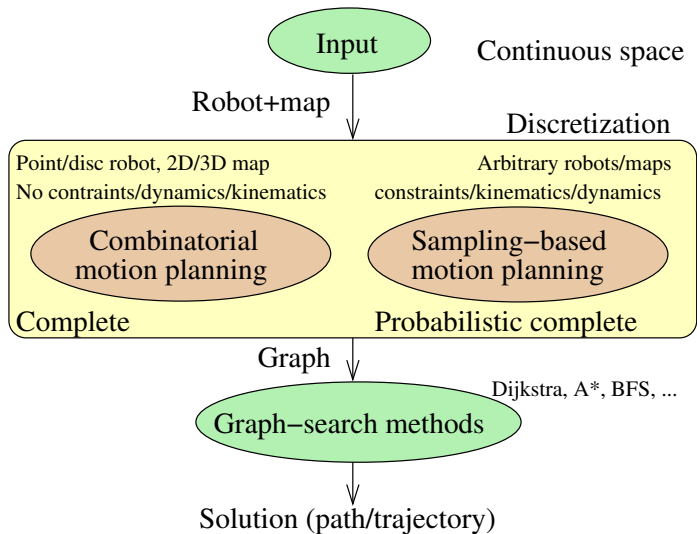


Discretization



Search





- Motion planning: how to move objects and avoid obstacles
- Configuration space  $\mathcal{C}$
- Generally, planning leads to search in continuous  $\mathcal{C}$
- But we (generally) don't have explicit representation of  $\mathcal{C}$
- We have to first create a discrete representation of  $\mathcal{C}$
- and search it by graph-search methods