

Robotics: Motion planning

Vladimír Petrík

vladimir.petrik@cvut.cz

9 November 2026

Motivation

- Imagine a task in which a robot has to carry a full mug from one place to another
- How would you solve it with the methods you know?
  - connect the start and goal points by a line segment – position interpolation
  - fix the orientation of the mug
  - compute the inverse kinematics

Motivation

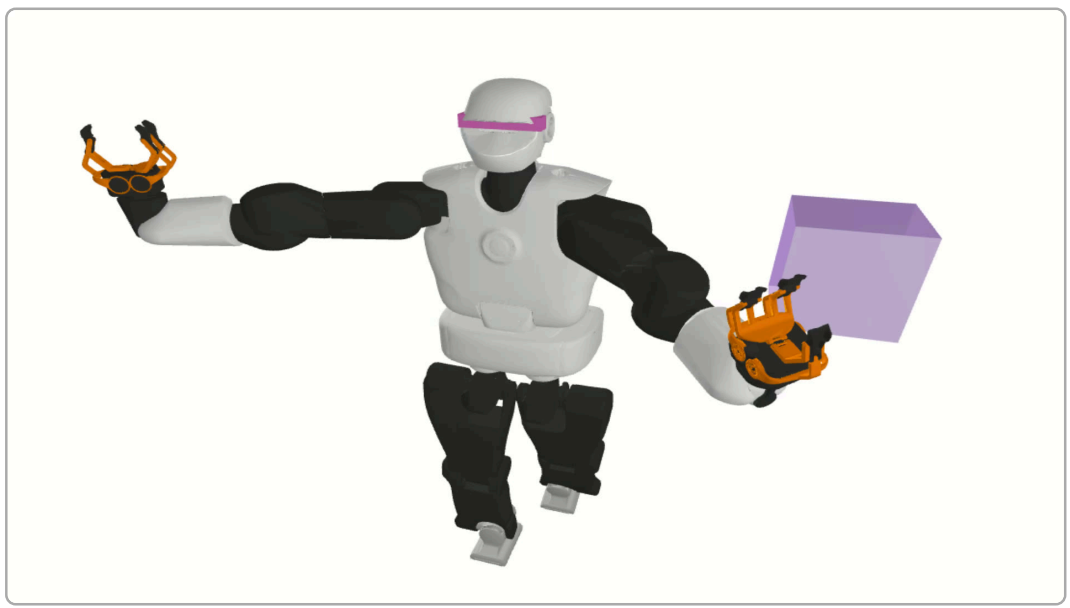
- Imagine a task in which a robot has to carry a full mug from one place to another
- How would you solve it with the methods you know?
  - connect the start and goal points by a line segment – position interpolation
  - fix the orientation of the mug
  - compute the inverse kinematics
- Possible problems
  - inverse kinematics gives several solutions
  - inverse kinematics may find no solution
  - an obstacle in the direct path

Motivation

- Imagine a task in which a robot has to carry a full mug from one place to another
- How would you solve it with the methods you know?
  - connect the start and goal points by a line segment – position interpolation
  - fix the orientation of the mug
  - compute the inverse kinematics
- Possible problems
  - inverse kinematics gives several solutions
  - inverse kinematics may find no solution
  - an obstacle in the direct path
- Define the admissible orientations and use planning

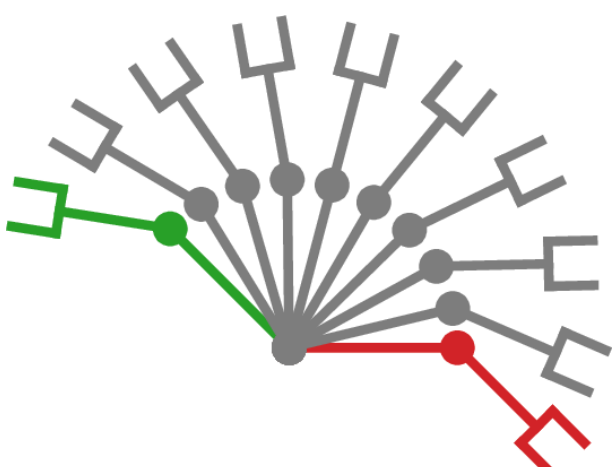
Goal of today's lecture

- Motion planning
  - what is planning?
  - planning in mobile robotics
  - planning for a 2D manipulator
  - planning for a real 7D manipulator
- Based on chapter 10 of Lynch & Park: Modern Robotics



Problem formulation

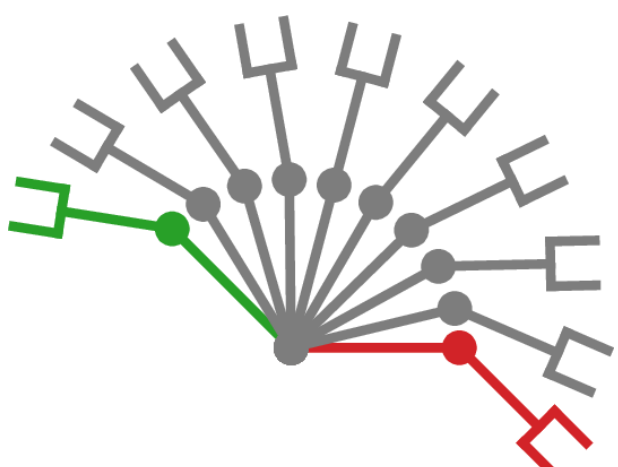
- Motion planning for a robot from a given start to a given goal
  - avoiding obstacles
  - respecting the joint limits
  - respecting the maximum velocities and forces/torques
  - other constraints of the environment (e.g. do not spill the water from the mug $\rightarrow$ orientation constraint)



Task-space planners exist, but we will not deal with them today.

Problem formulation

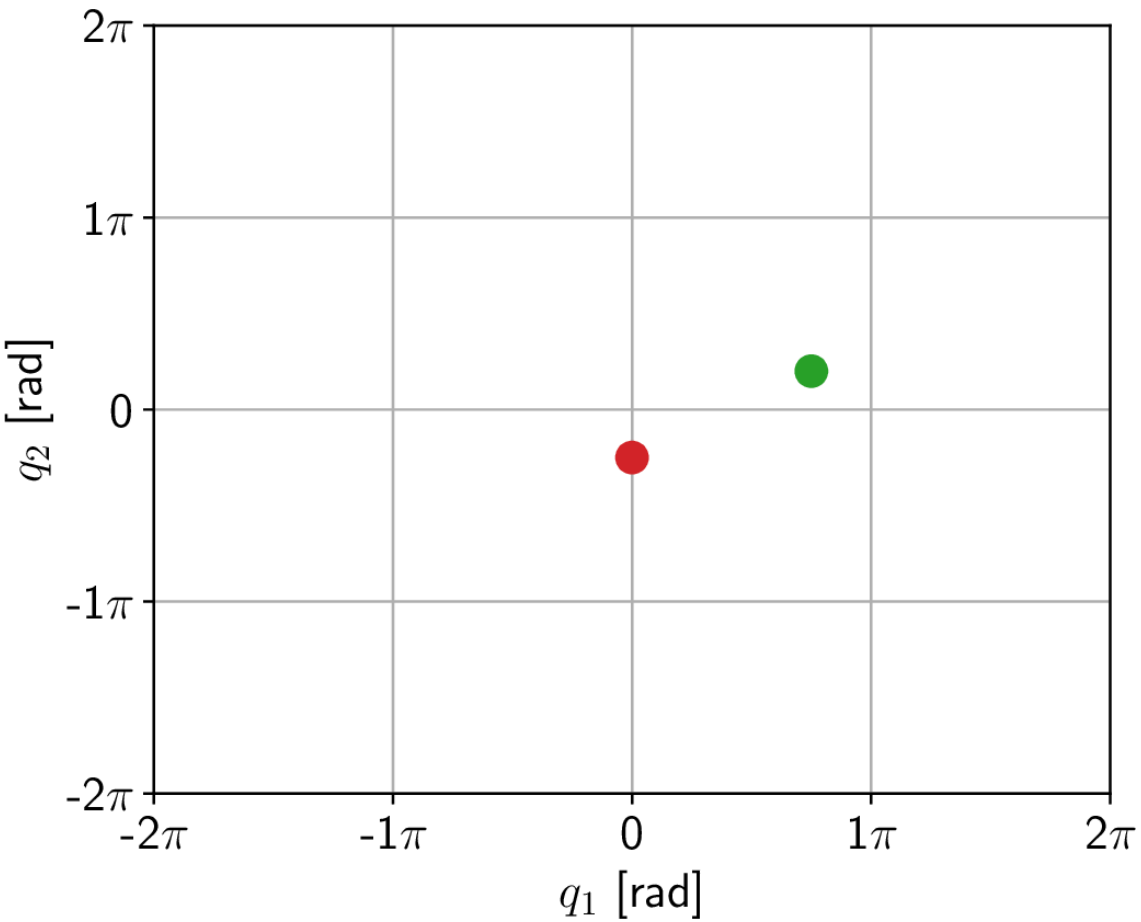
- Motion planning for a robot from a given start to a given goal
  - avoiding obstacles
  - respecting the joint limits
  - respecting the maximum velocities and forces/torques
  - other constraints of the environment (e.g. do not spill the water from the mug $\rightarrow$ orientation constraint)
- Planning takes place in the configuration space of the robot
  - configuration space $\mathcal{C}$ (C-space)
  - robot configuration $q \in \mathcal{C}$
  - both the start and the goal are defined in the configuration space, $(\boldsymbol{q}_s, \boldsymbol{q}_g) \in \mathcal{C} \times \mathcal{C}$

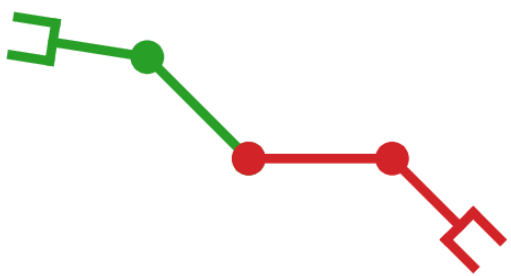


Task-space planners exist, but we will not deal with them today.

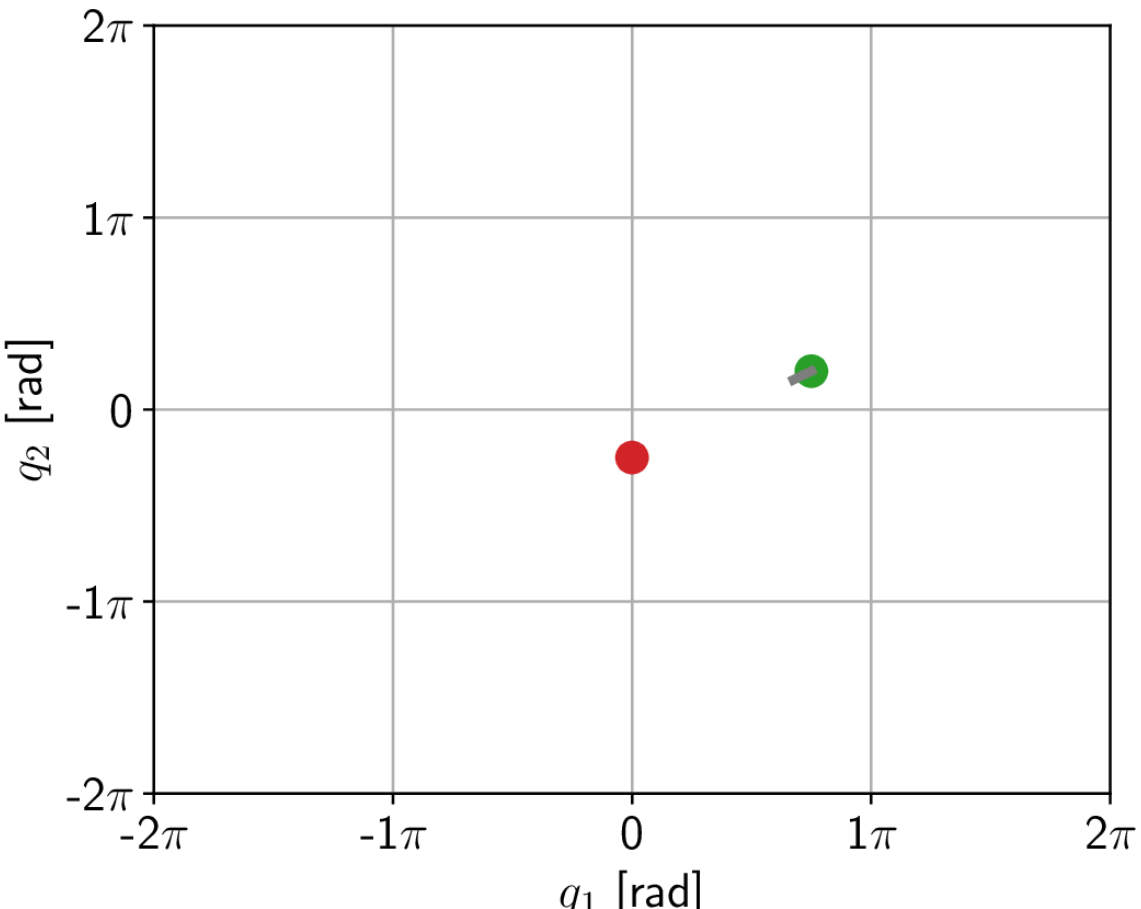
Configuration space

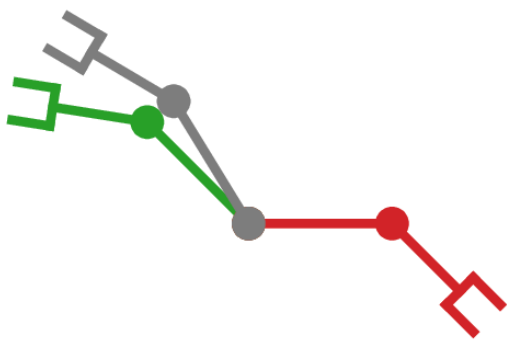
Configuration space: interpolation



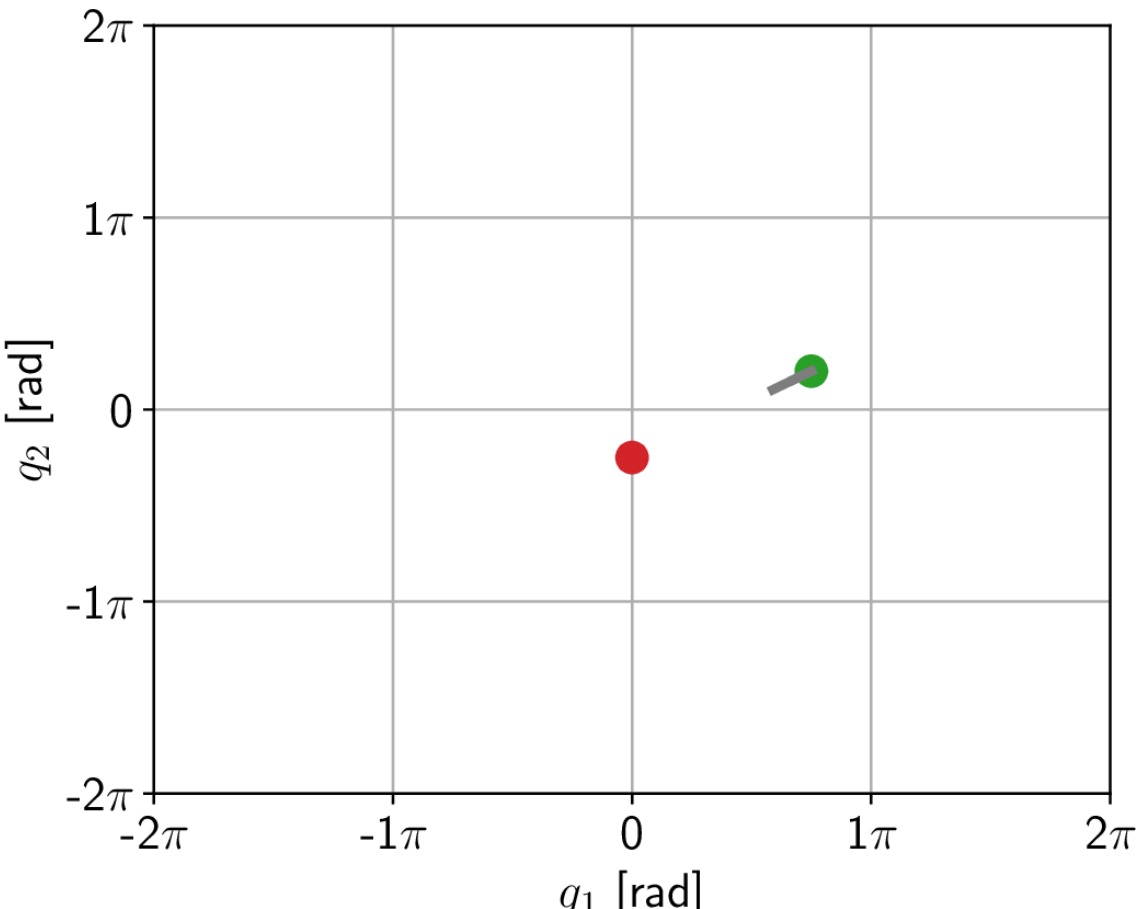


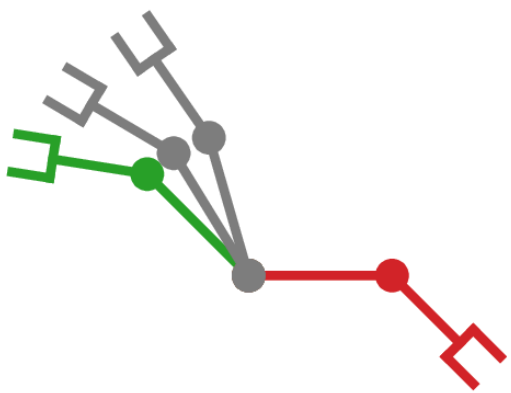
Configuration space: interpolation



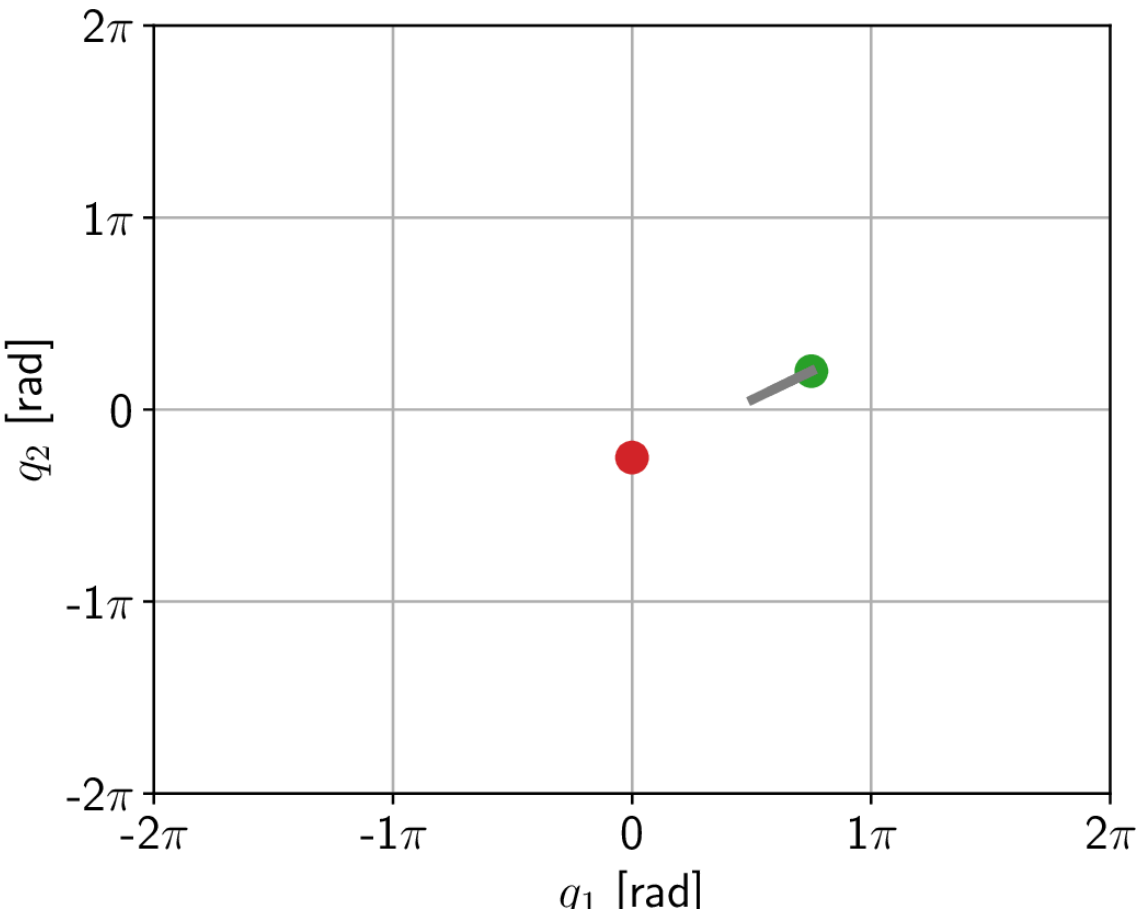


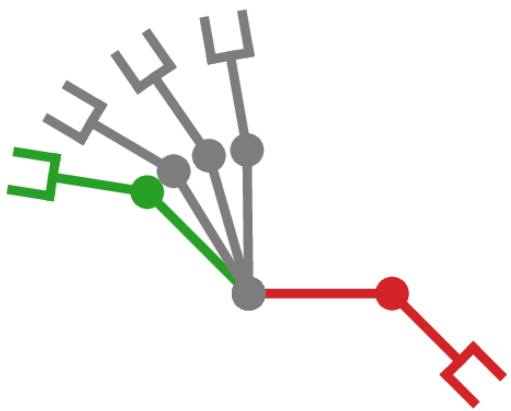
Configuration space: interpolation



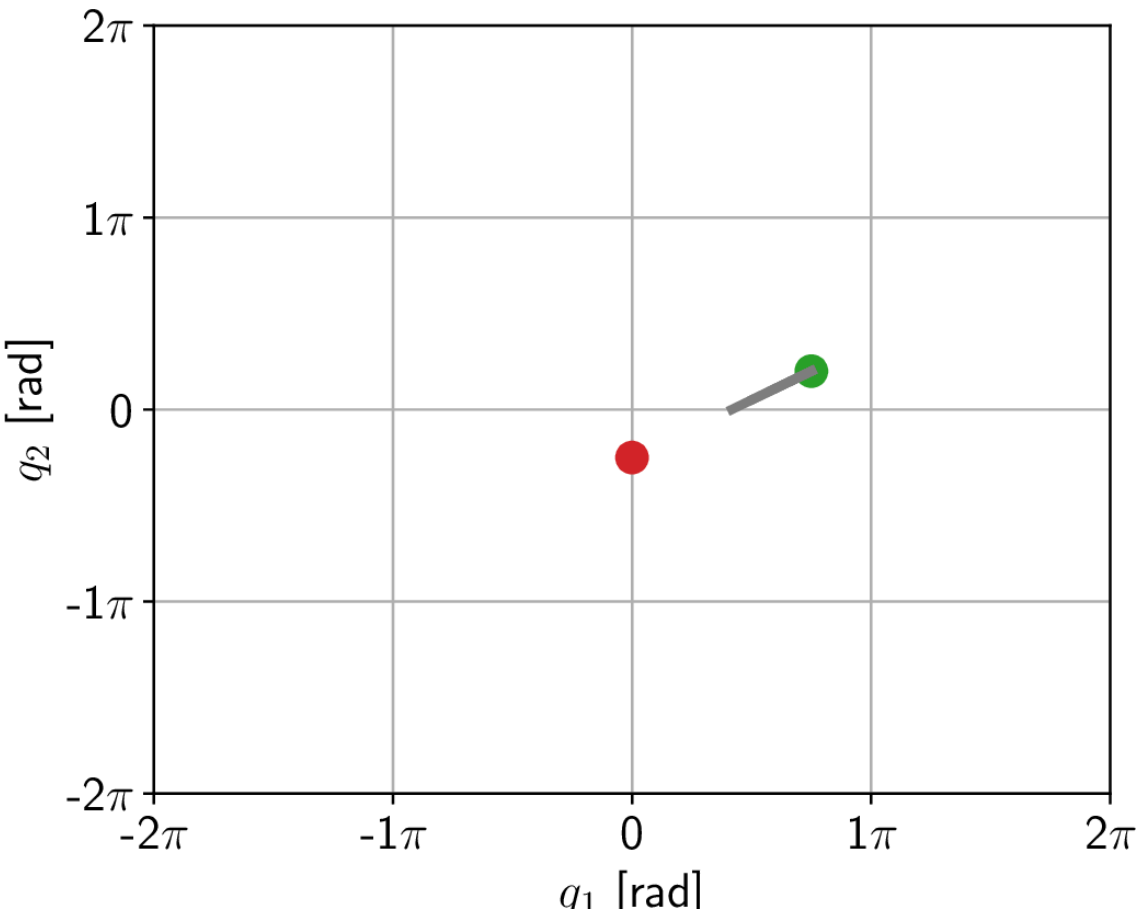


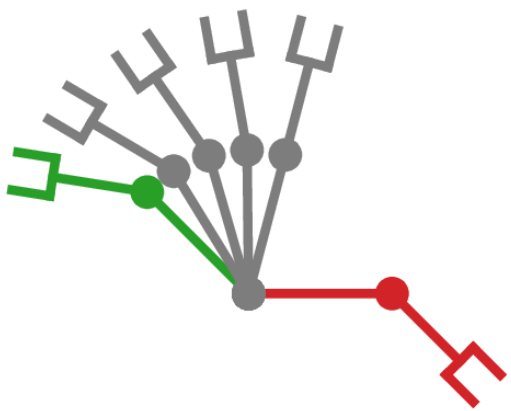
Configuration space: interpolation



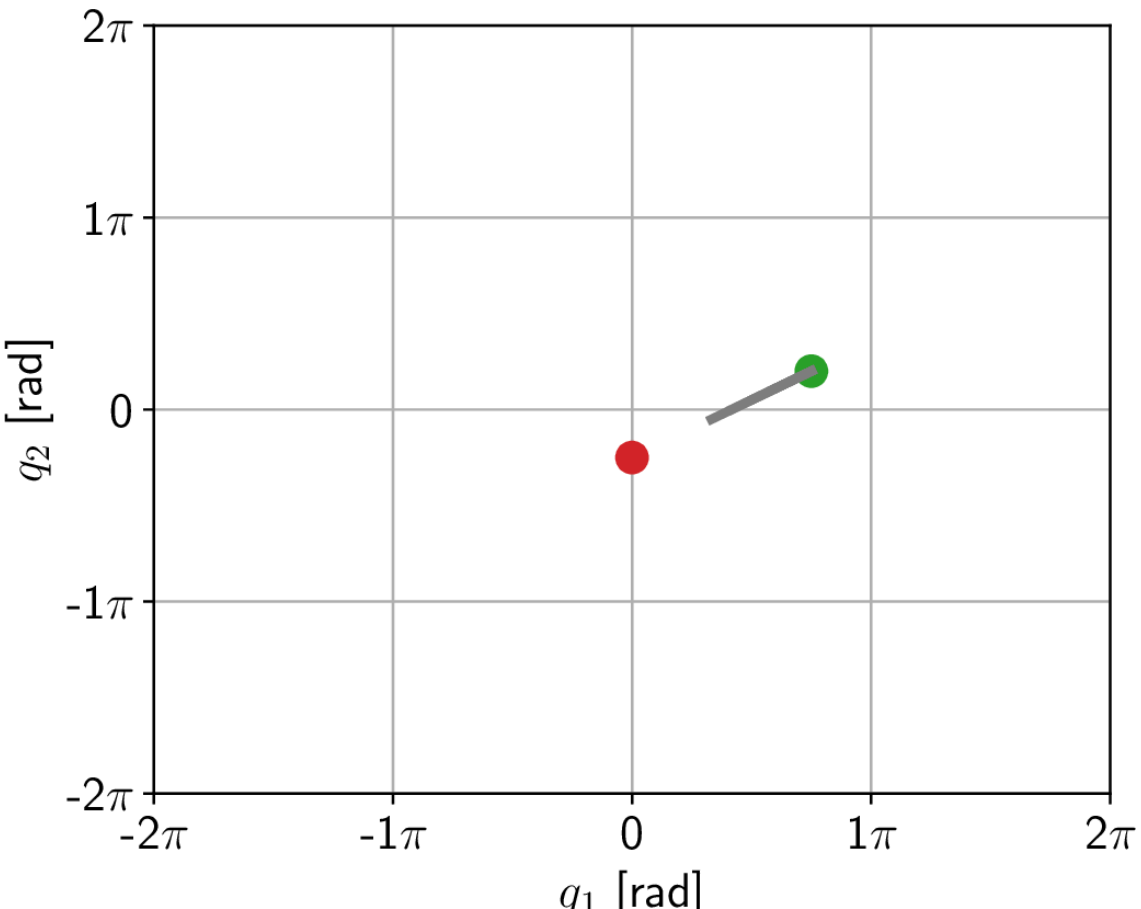


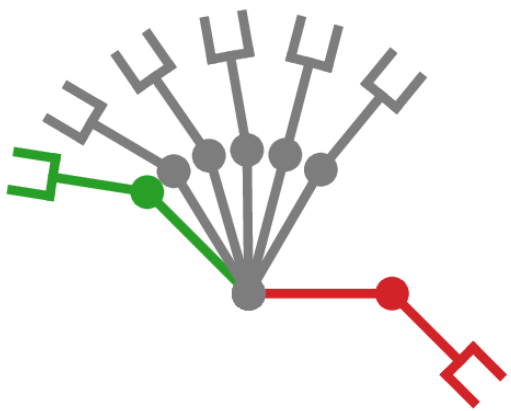
Configuration space: interpolation



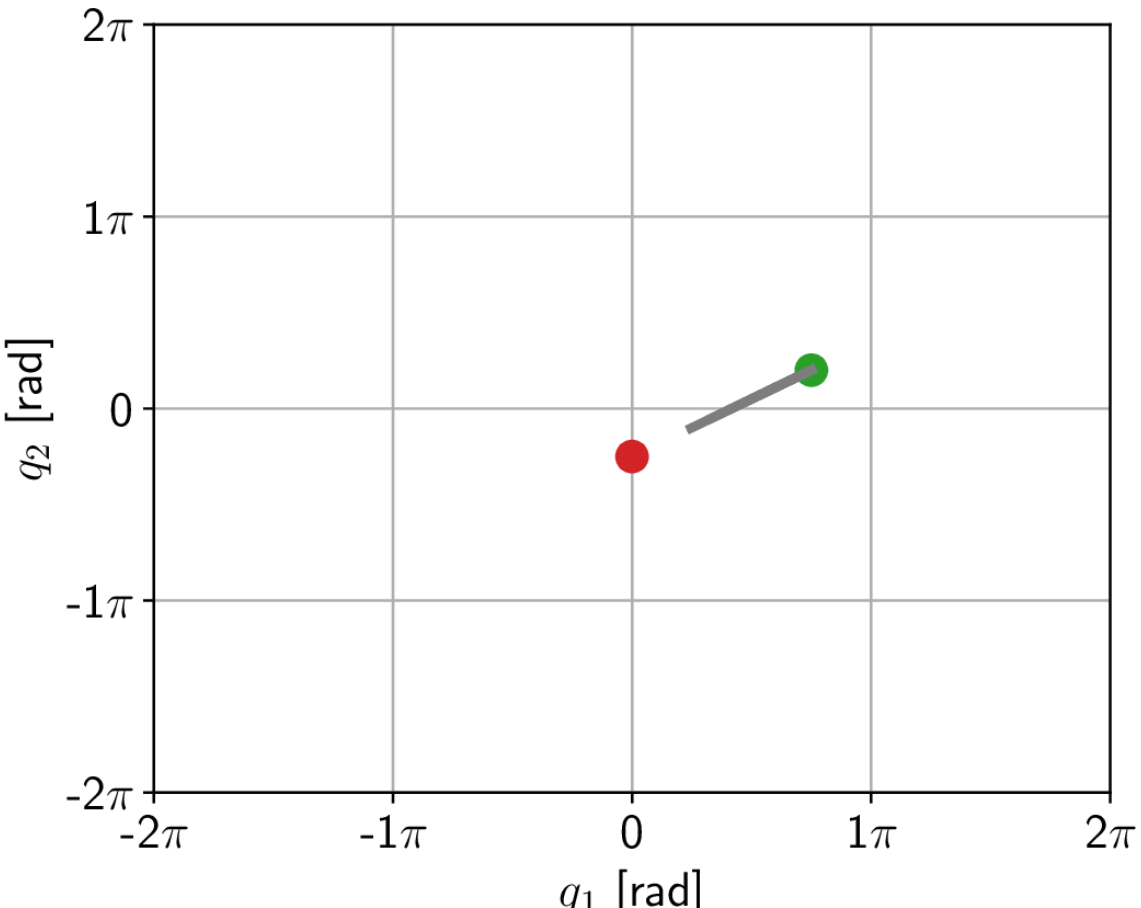


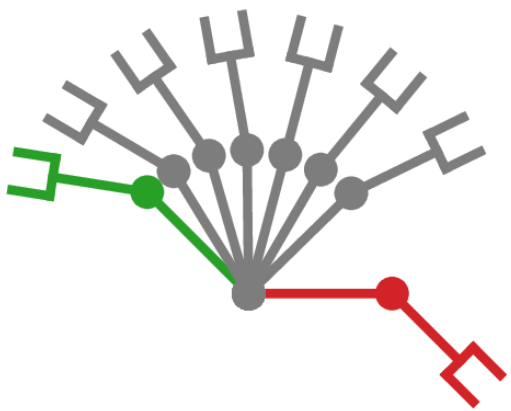
Configuration space: interpolation



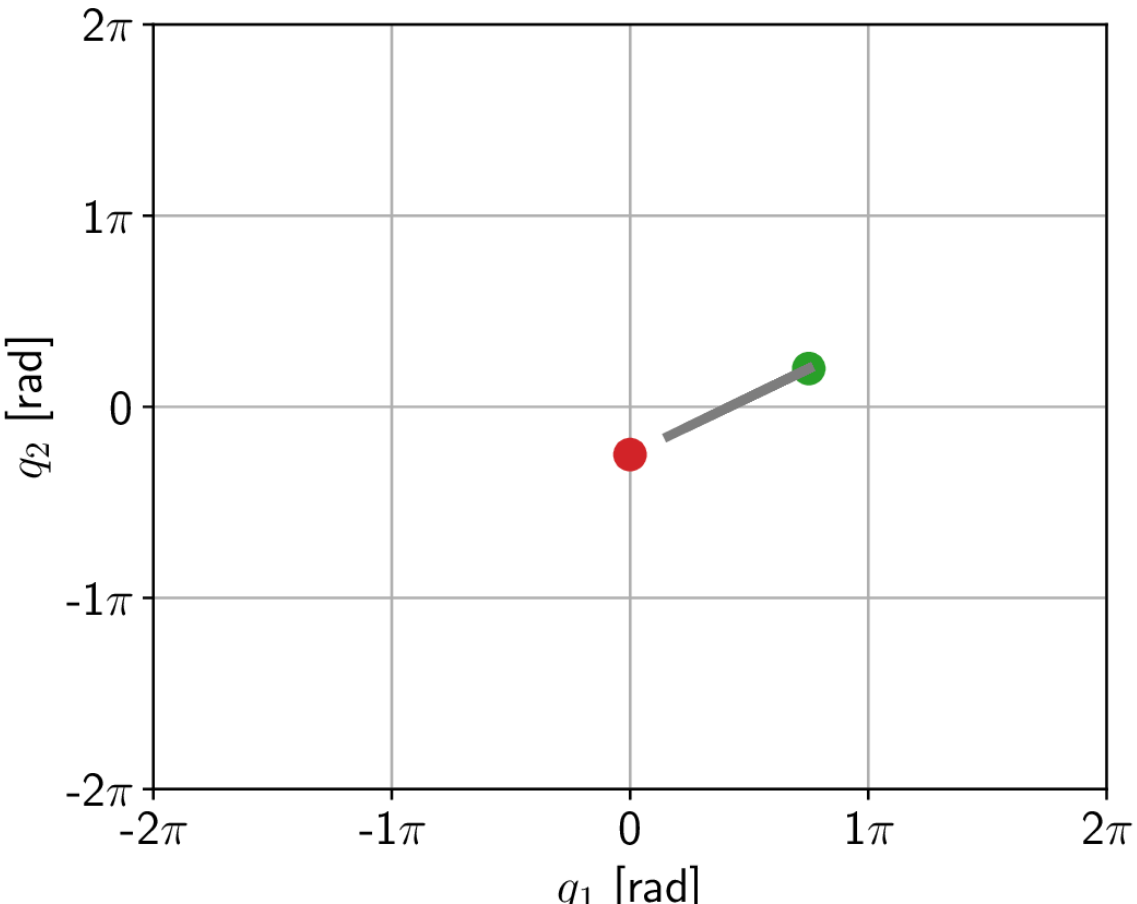


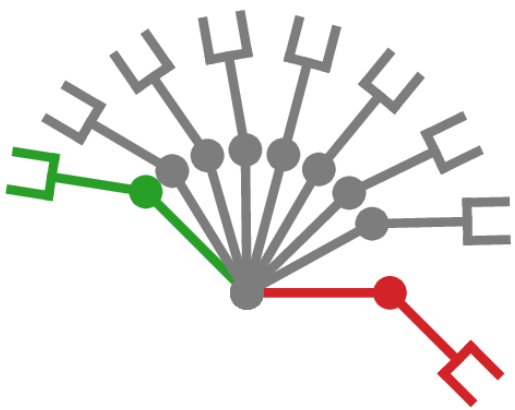
Configuration space: interpolation



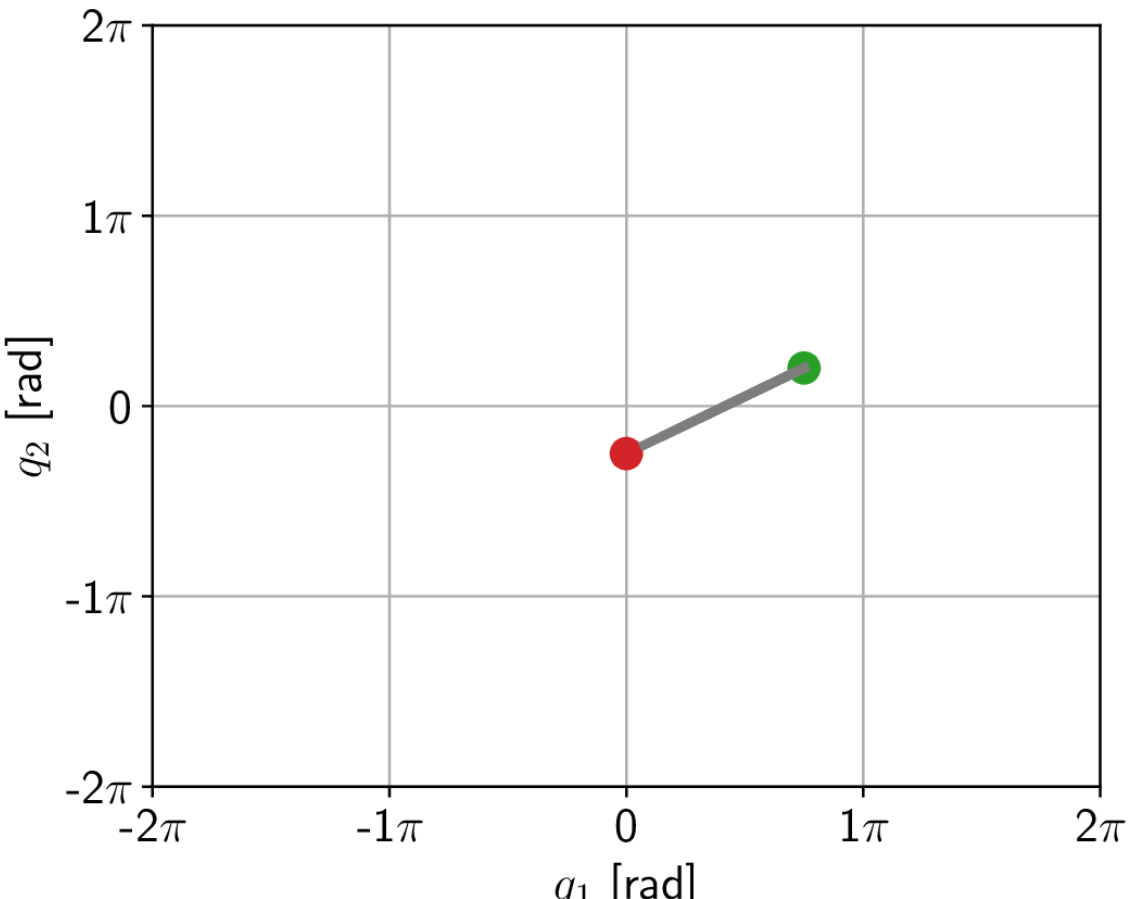


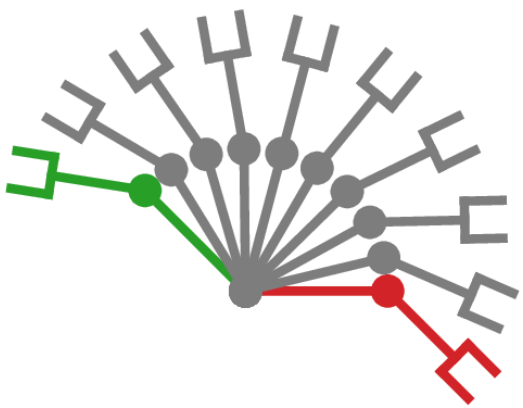
Configuration space: interpolation



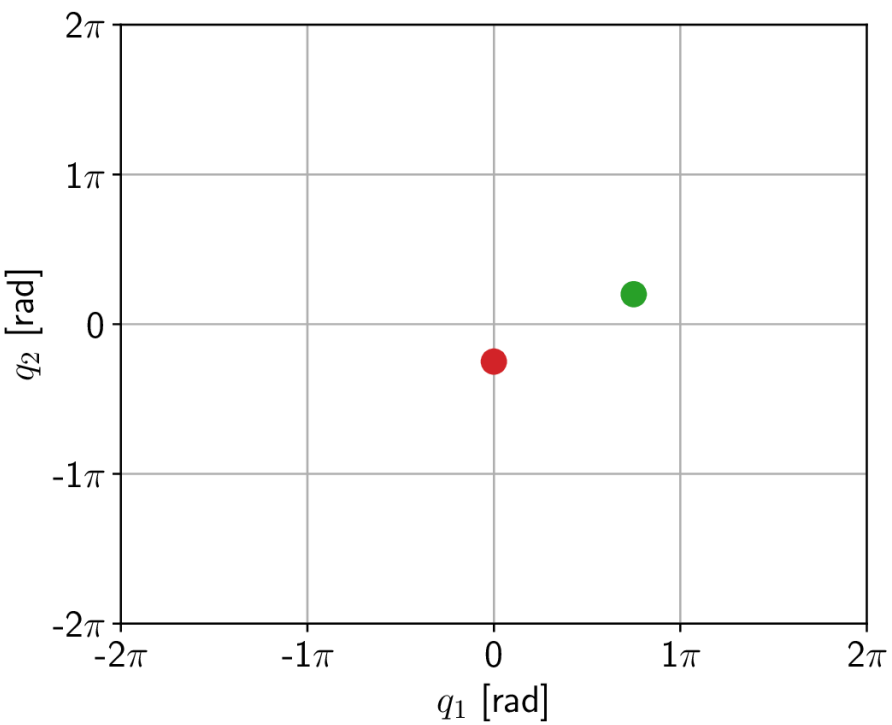


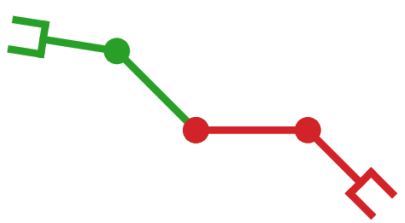
Configuration space: interpolation





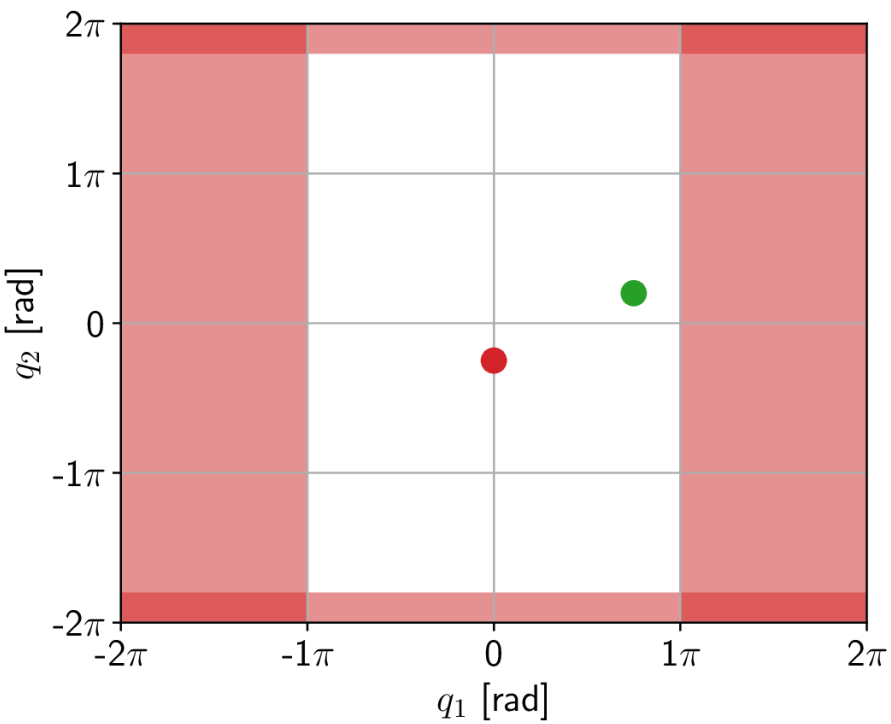
Constraints and the configuration space

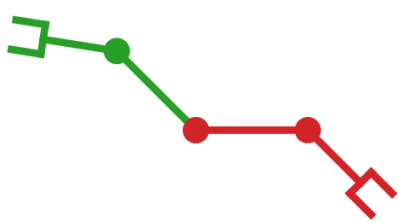




Constraints and the configuration space

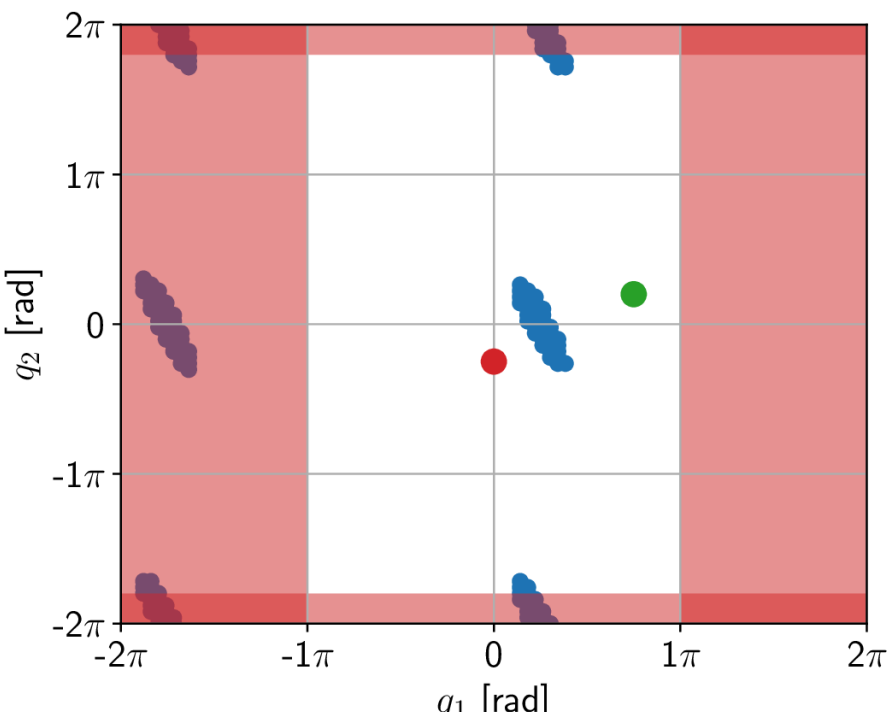
- Joint limits

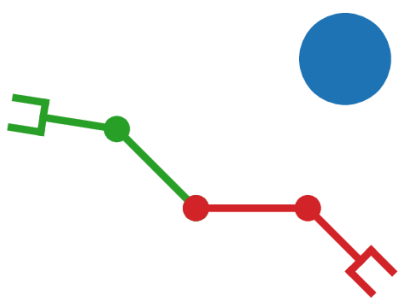




Constraints and the configuration space

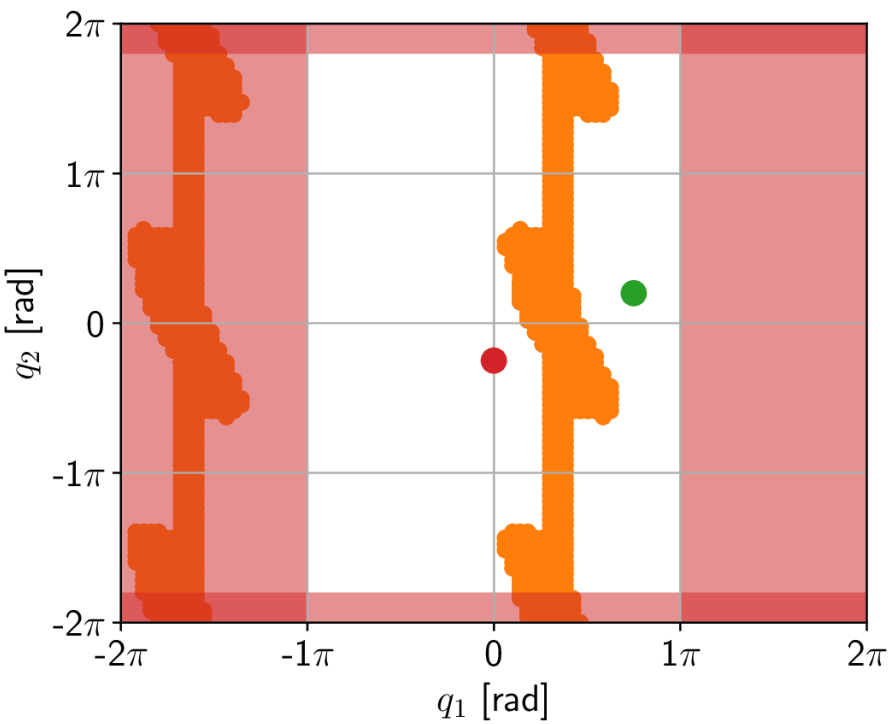
- Joint limits
- Obstacles in the environment (note: the shape of the gripper is ignored)

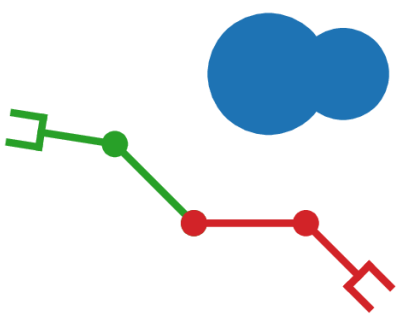




Constraints and the configuration space

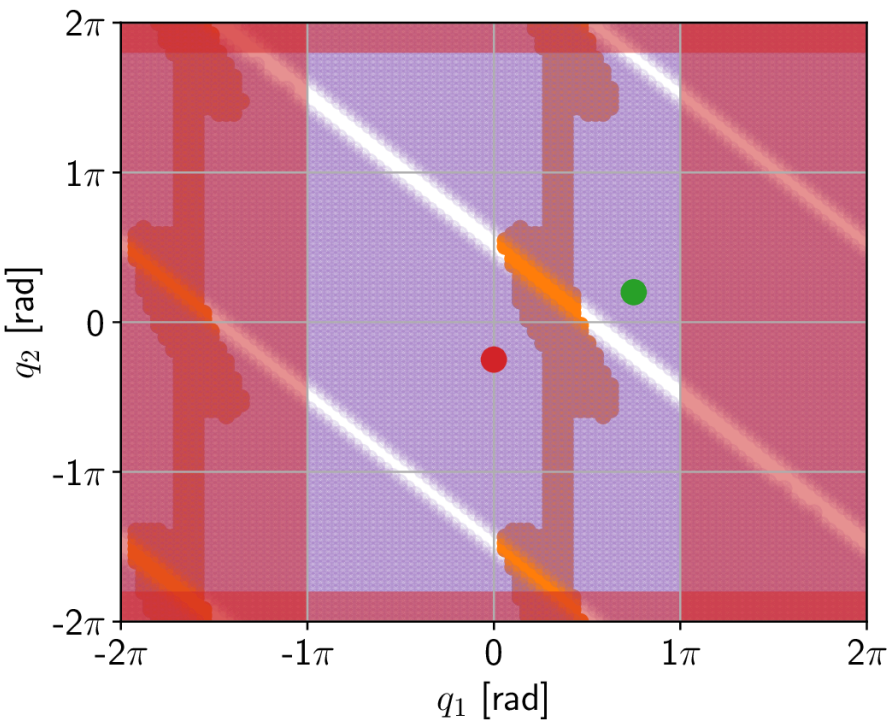
- Joint limits
- Obstacles in the environment (note: the shape of the gripper is ignored)

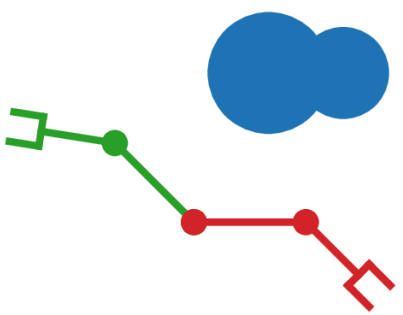




Constraints and the configuration space

- Joint limits
- Obstacles in the environment (note: the shape of the gripper is ignored)
- Constraint on the gripper orientation ($80° < \theta < 100°$)



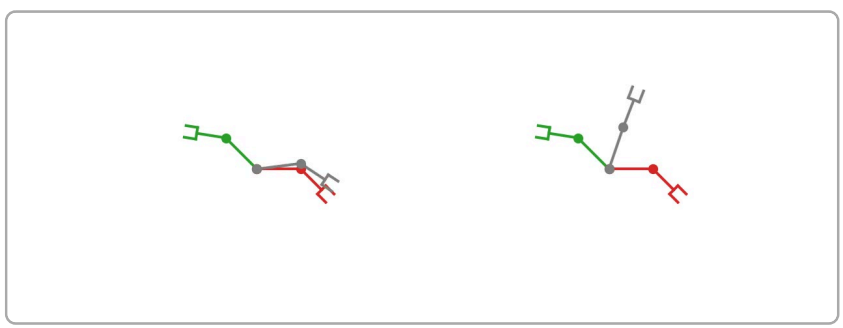


Configuration space: conclusion

- We divide the configuration space into
  - the space of admissible configurations $\mathcal{C}_{\text{free}}$
  - the space of inadmissible configurations $\mathcal{C}_{\text{occ}}$
  - $\mathcal{C} = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{occ}}$
- The configuration space is (usually) hard to describe explicitly
- Even if both the start and the goal lie in $\mathcal{C}_{\text{free}}$, a motion connecting them need not exist

What is motion? Path $\times$ trajectory

- Path
  - geometric description (a sequence of configurations)
  - no time stamps, no dynamics, no constraints on the control signal
  - find $q(s) \in \mathcal{C}_{\text{free}}$, $s \in [0, 1]$ such that
  - $q(0) = \boldsymbol{q}_s, \quad q(1) = \boldsymbol{q}_g$
  - the main assumption is that a trajectory can be obtained by post-processing
- Trajectory
  - describes the robot position in time
  - $q(t) \in \mathcal{C}_{\text{free}}$, $t \in [0, T]$



Can we plan the trajectory directly?

- The problem definition includes the equation of motion of the system
- Find the control signal in time $\boldsymbol{u}(t)$ such that
- $q(0) = \boldsymbol{q}_s, \quad q(T) = \boldsymbol{q}_g, \quad \dot{q} = f(q(t), \boldsymbol{u}(t))$
- A more complicated task than finding a path
- When is it preferred?
  - only some joints are actuated $\rightarrow$ it may be impossible to follow the planned path
- In robotic motion planning, a path is typically planned and a trajectory is computed from it

Path planning in mobile robotics

Path planning in mobile robotics

- How planning works in ROS: MoveBase
- The robot moves in the plane, $(v_x, v_\theta)^\top$
- We know:
  - the dimensions of the robot (footprint)
  - the map of the environment
  - the position of the robot in the map, $(x, y, \theta)^\top$
  - the goal position in the map

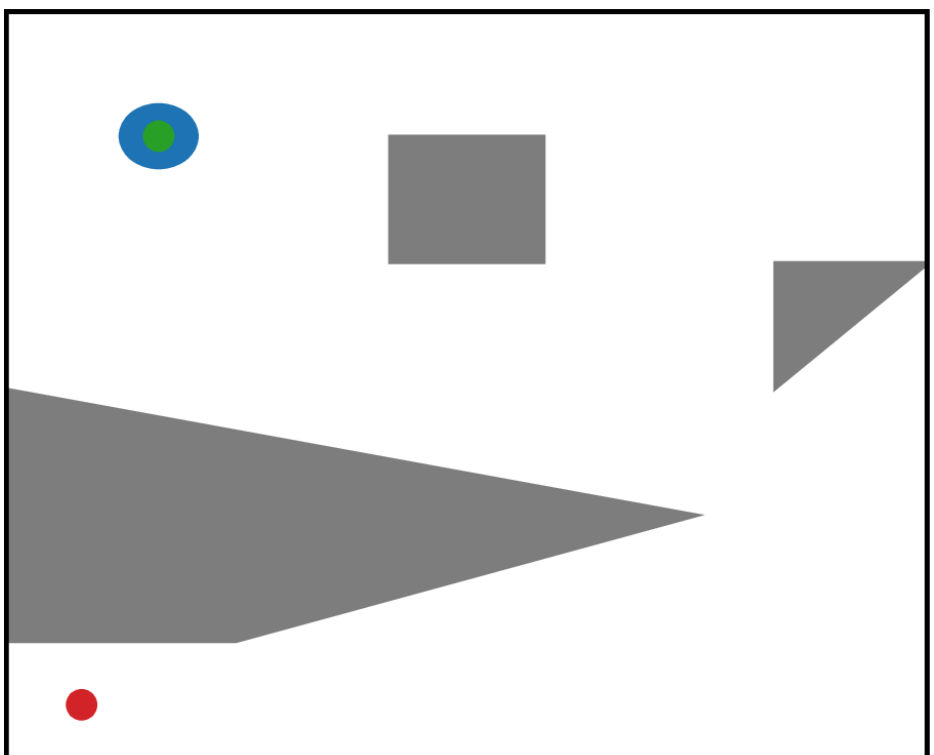
Path planning in mobile robotics

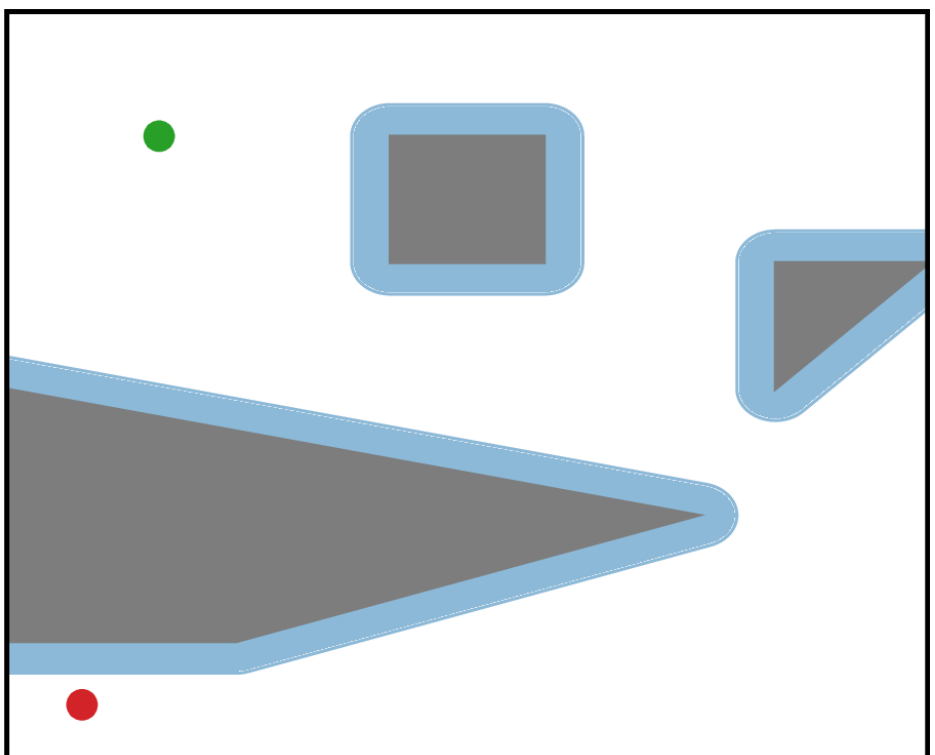
- How planning works in ROS: MoveBase
- The robot moves in the plane, $(v_x, v_\theta)^\top$
- We know:
  - the dimensions of the robot (footprint)
  - the map of the environment
  - the position of the robot in the map, $(x, y, \theta)^\top$
  - the goal position in the map
- Navigation to a given goal in ROS: MoveBase:
  - global path planner in 2D
  - computing the orientation (the robot heads forward along the path – why?)
  - local planner (avoids dynamic obstacles, uses the lidar input)
  - sending the command to the robot

Path planning in mobile robotics

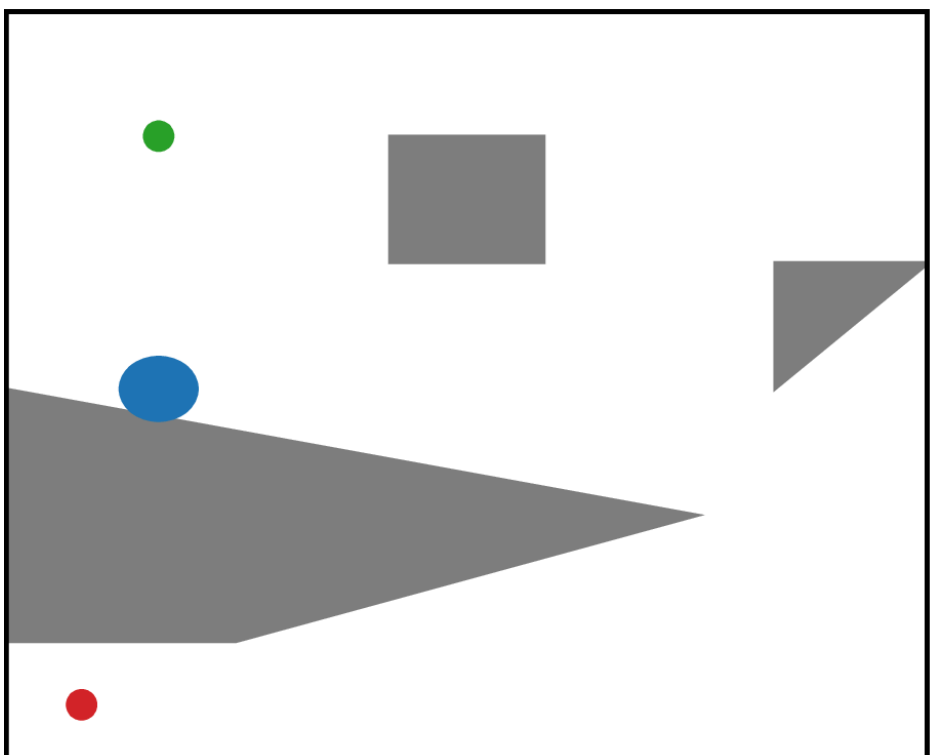
- How planning works in ROS: MoveBase
- The robot moves in the plane, $(v_x, v_\theta)^\top$
- We know:
  - the dimensions of the robot (footprint)
  - the map of the environment
  - the position of the robot in the map, $(x, y, \theta)^\top$
  - the goal position in the map
- Navigation to a given goal in ROS: MoveBase:
  - **global path planner in 2D**
  - computing the orientation (the robot heads forward along the path – why?)
  - local planner (avoids dynamic obstacles, uses the lidar input)
  - sending the command to the robot

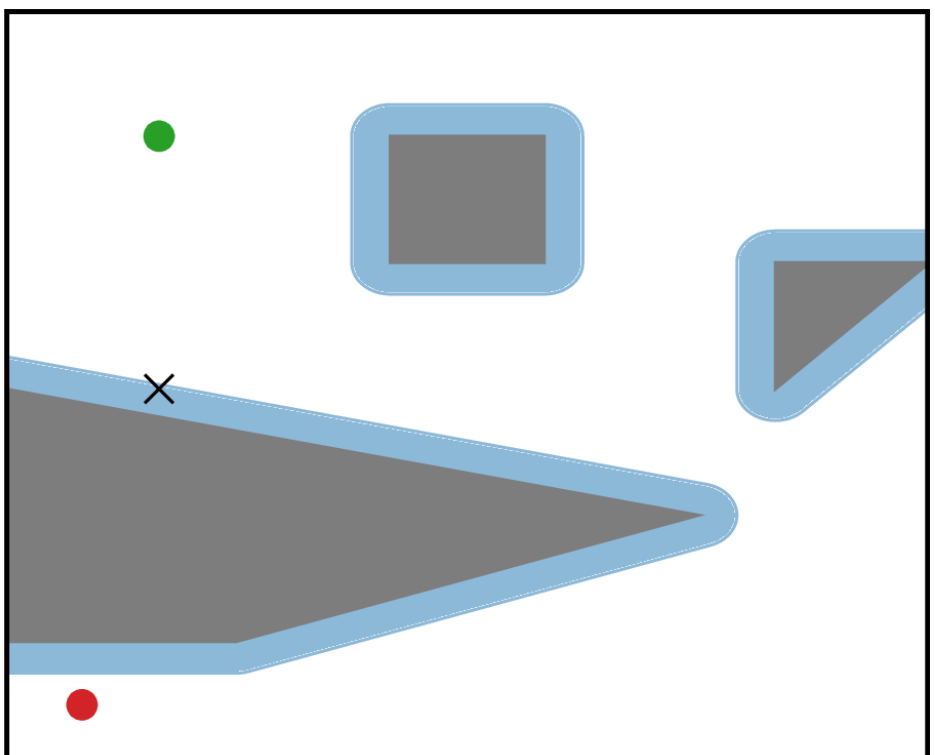
What do the map and the C-space look like?





What do the map and the C-space look like?

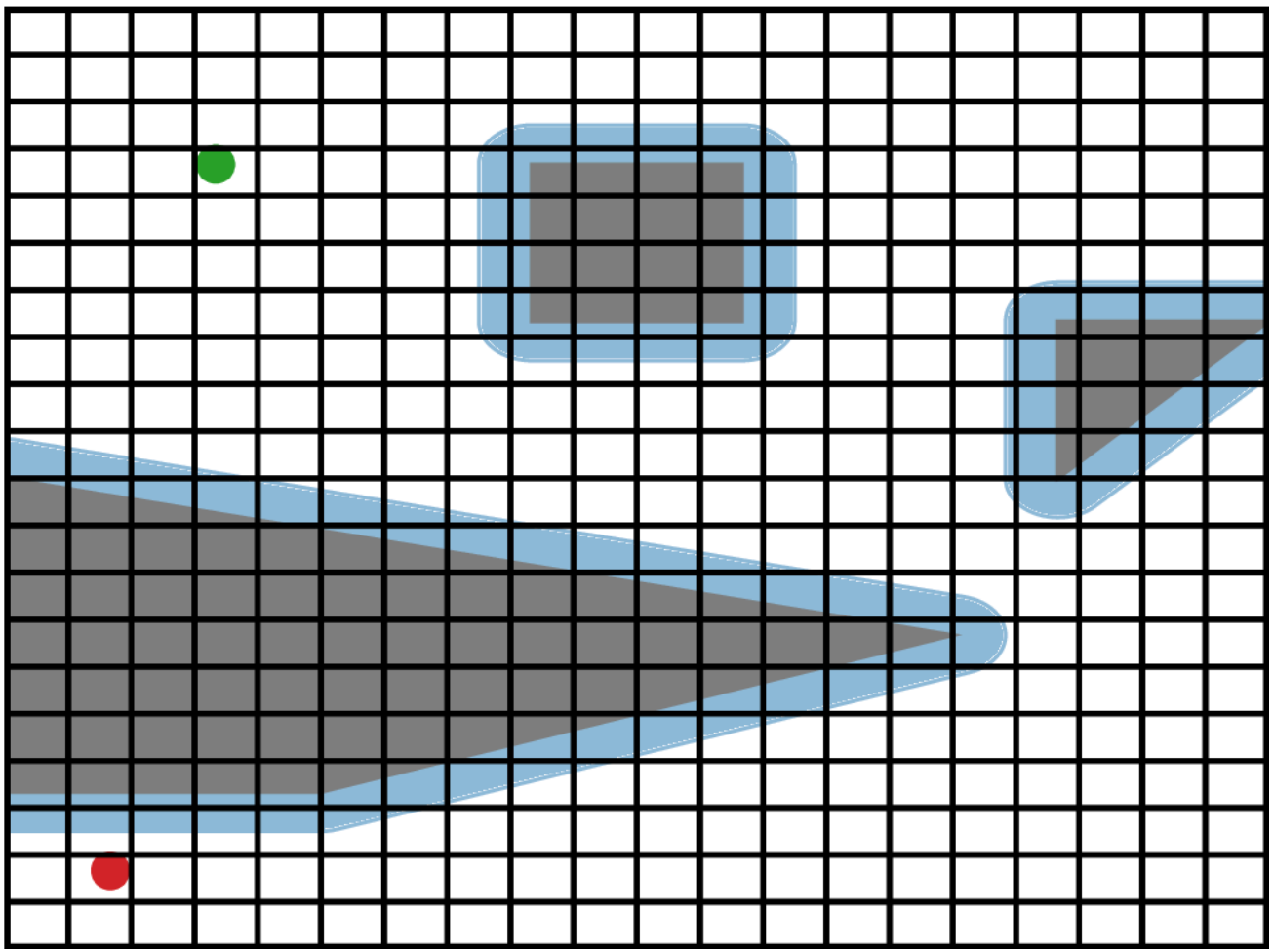




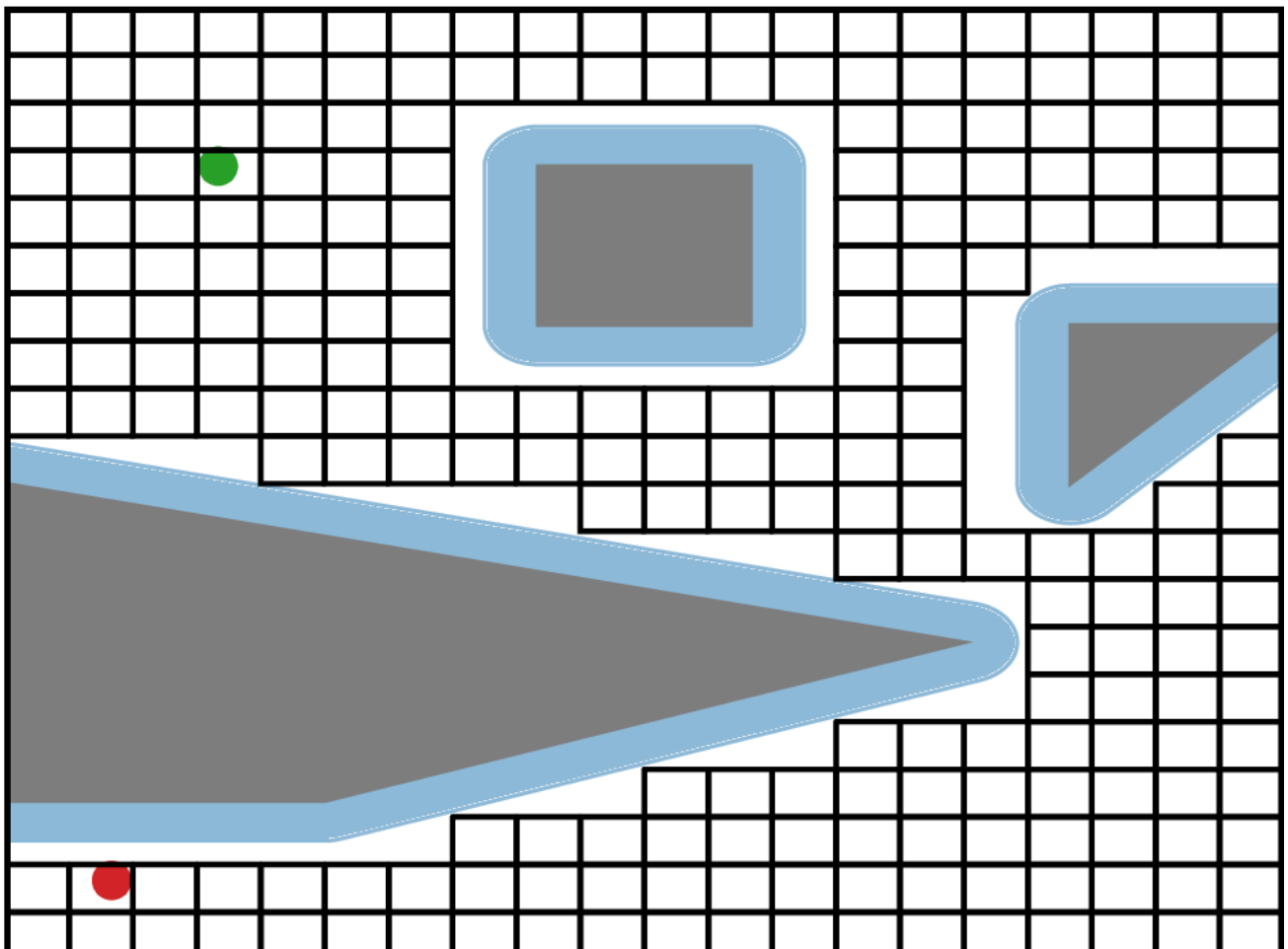
Planners #1: planning on a discrete grid

- Discretize the configuration space (the map is often discretized already on input)
- The simplest discretization is a grid
- The goal is to represent the free configuration space by a graph

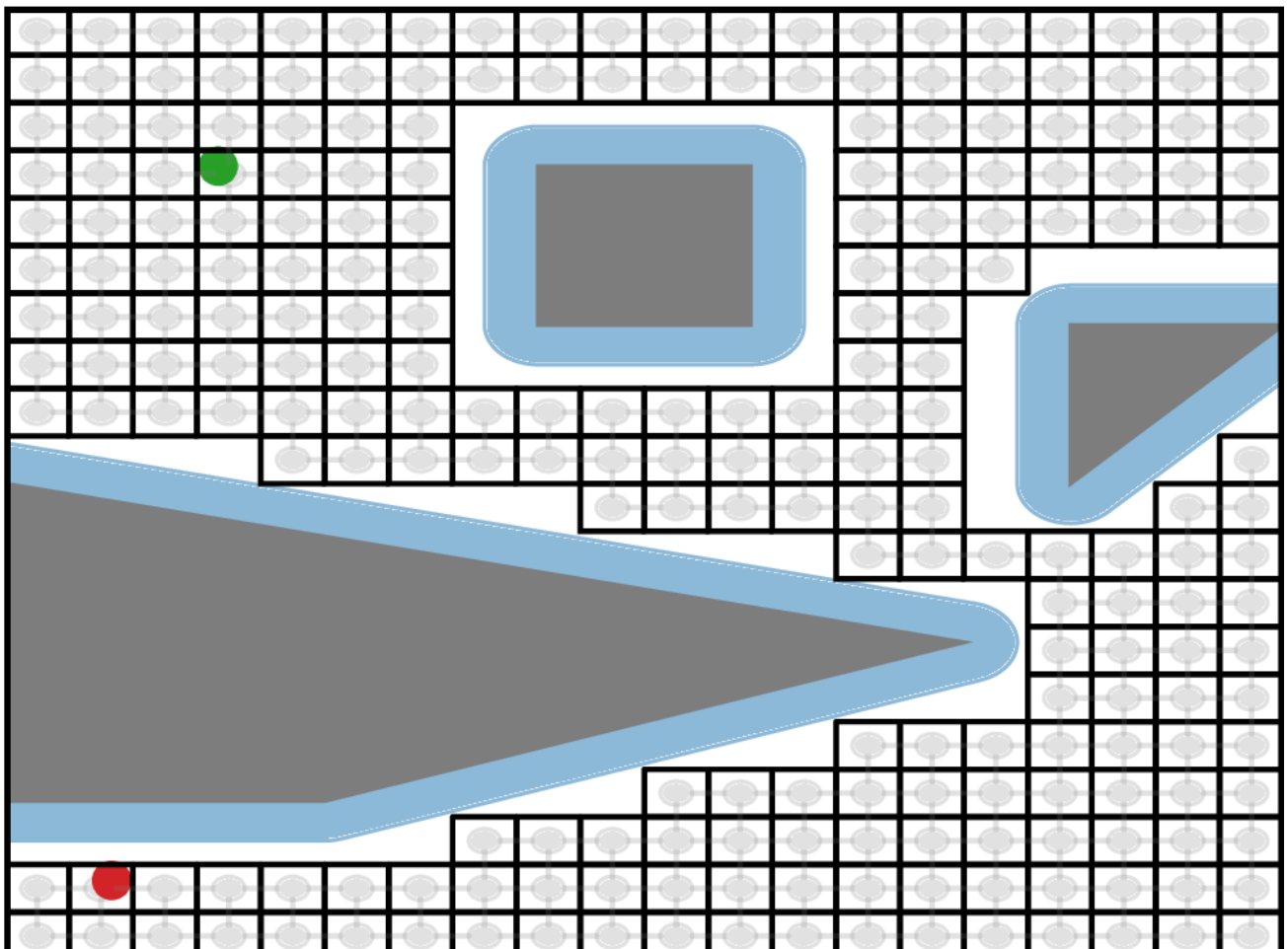
Planners #1: planning on a discrete grid



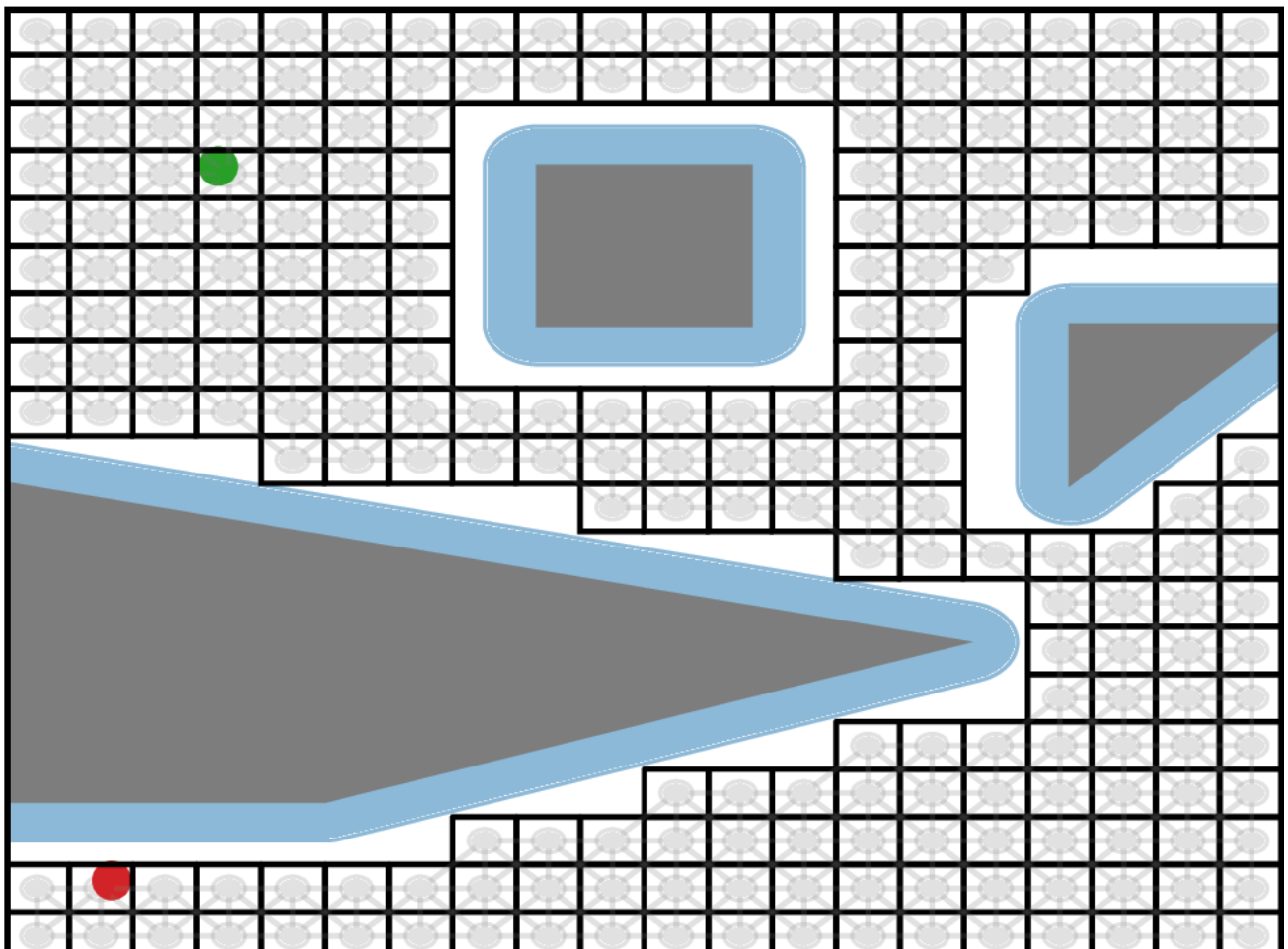
Planners #1: planning on a discrete grid



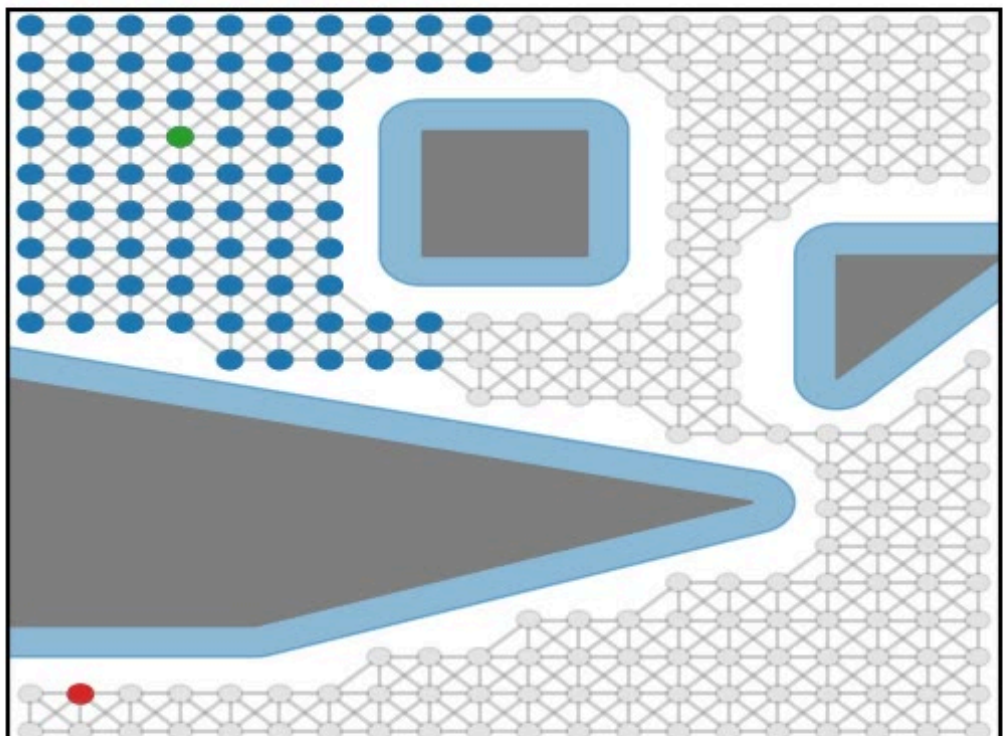
Planners #1: planning on a discrete grid



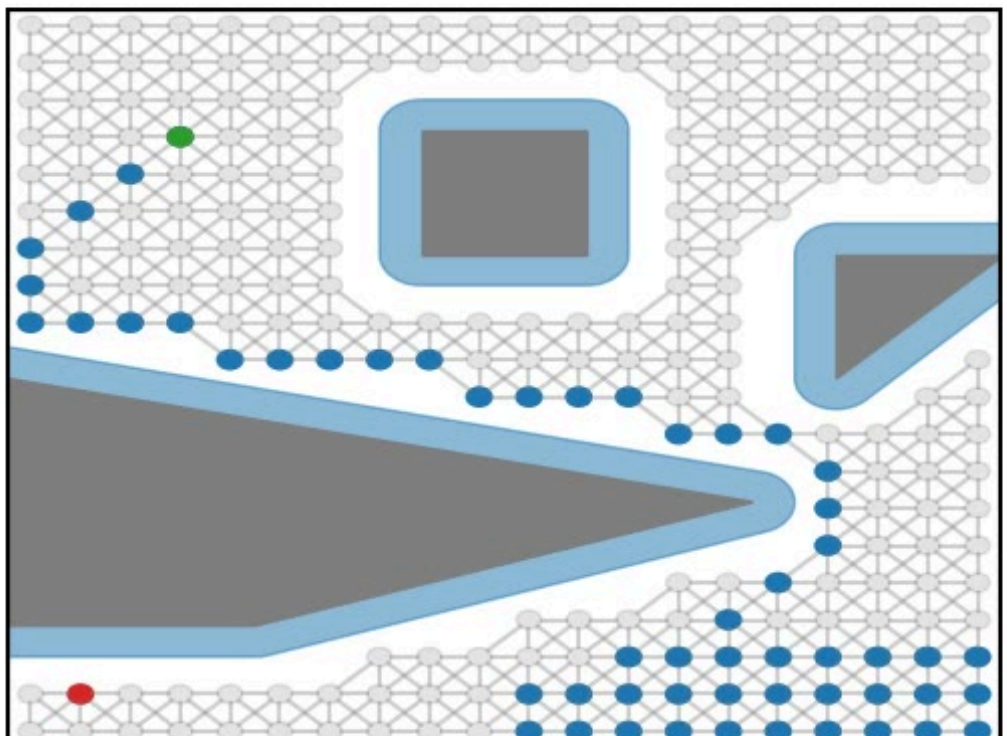
Planners #1: planning on a discrete grid



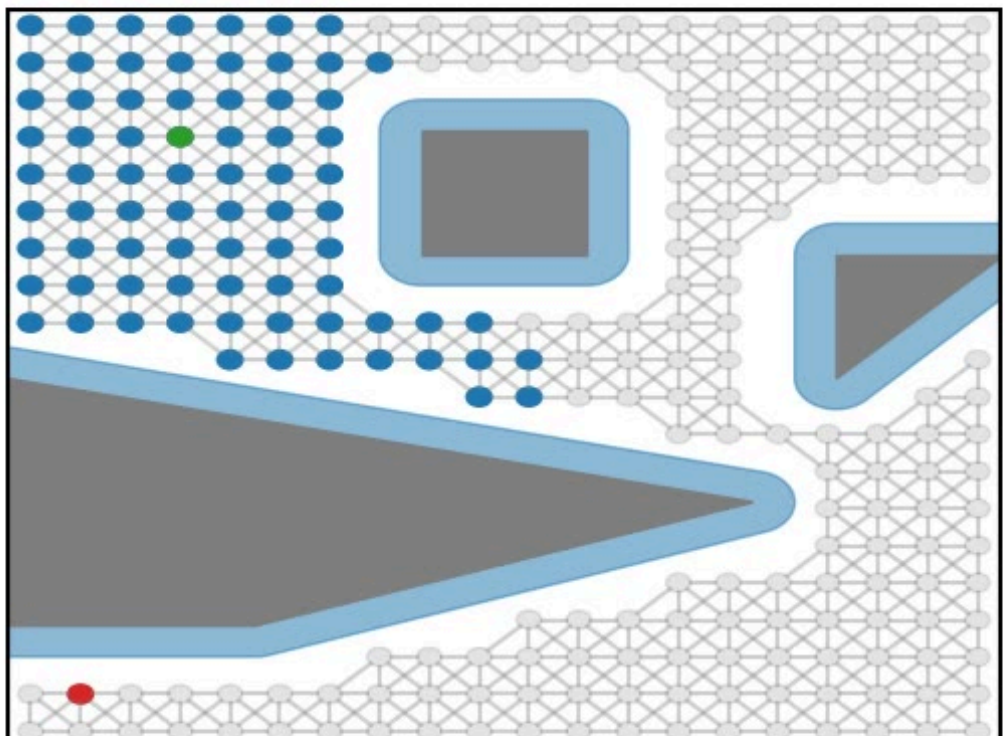
Graph search: breadth-first



Graph search: depth-first



Graph search: informed search A*

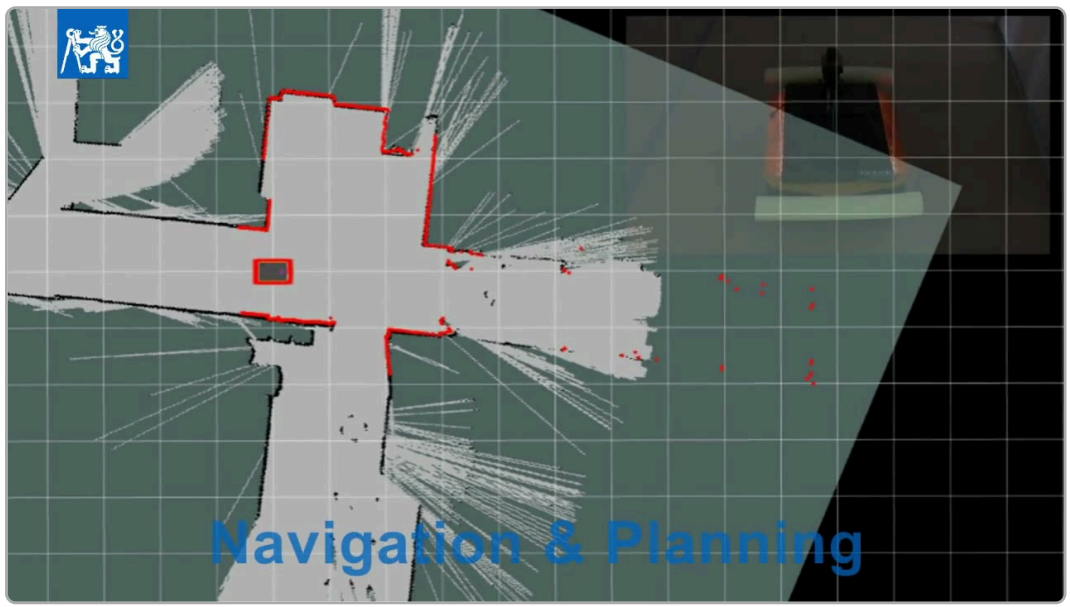


Graph search

- Uninformed breadth-first search
  - complete – finds a solution if one exists
  - optimal
- Uninformed depth-first search
  - complete only in finite graphs
  - not optimal
- Informed A*
  - uses a heuristic to guide the search and thus speeds it up
  - complete, optimal

Graph search

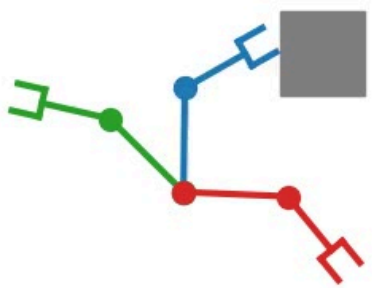
- Uninformed breadth-first search
  - complete – finds a solution if one exists
  - optimal
- Uninformed depth-first search
  - complete only in finite graphs
  - not optimal
- Informed A*
  - uses a heuristic to guide the search and thus speeds it up
  - complete, optimal

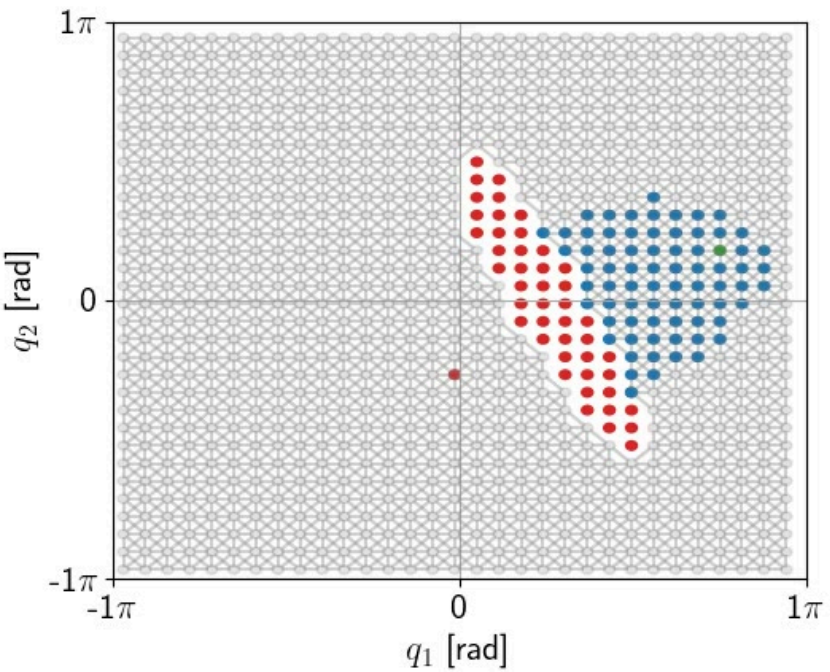


Graph search

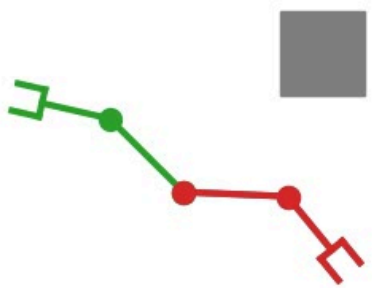
- Uninformed breadth-first search
  - complete – finds a solution if one exists
  - optimal
- Uninformed depth-first search
  - complete only in finite graphs
  - not optimal
- Informed A*
  - uses a heuristic to guide the search and thus speeds it up
  - complete, optimal
  - can we use A* for a robotic manipulator?

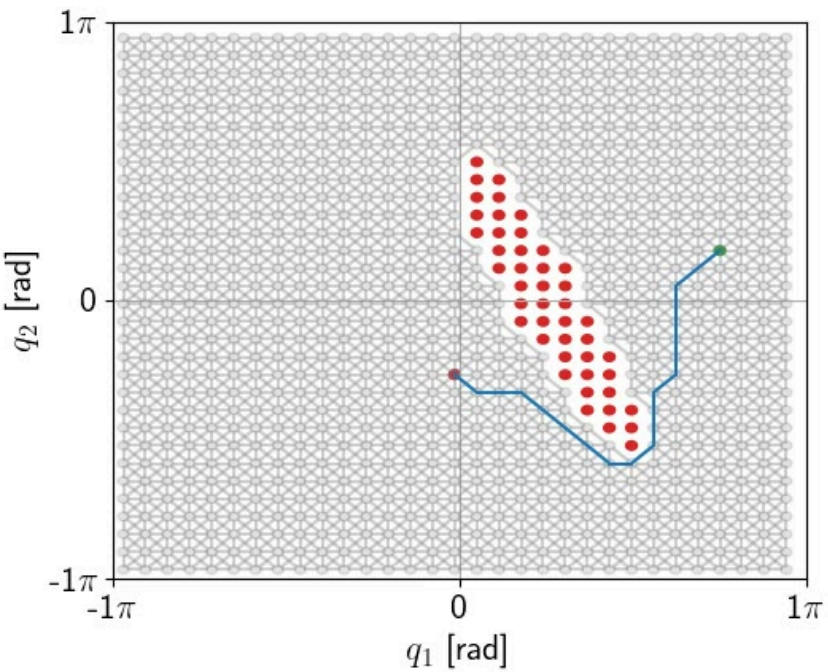
A* for a 2D manipulator





A* for a 2D manipulator



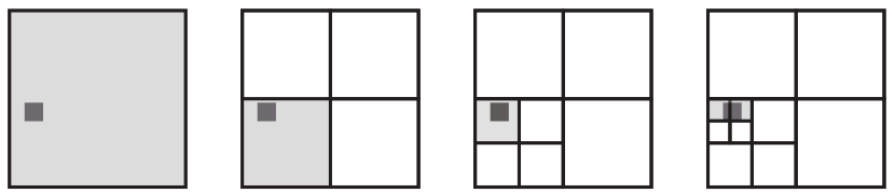


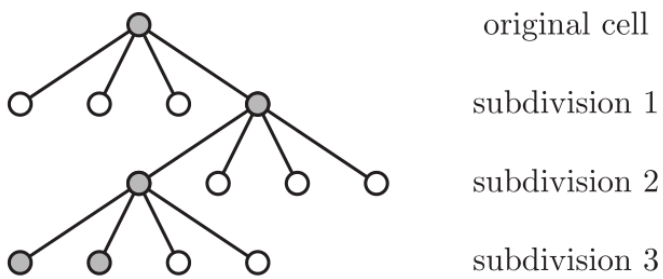
How large a grid is needed?

- Example of the mobile robot: $20^2 = 400$
- Example of the manipulator: $30^2 = 900$
- A more complex manipulator: $30^6 = 729\,000\,000$
- Humanoid: $30^{20} = 3.5 \cdot 10^{29}$
- The completeness of the algorithms depends on the grid resolution – *resolution complete*

How large a grid is needed?

- Example of the mobile robot: $20^2 = 400$
- Example of the manipulator: $30^2 = 900$
- A more complex manipulator: $30^6 = 729\,000\,000$
- Humanoid: $30^{20} = 3.5 \cdot 10^{29}$
- The completeness of the algorithms depends on the grid resolution – *resolution complete*
- Grids with variable resolution exist (figure from Lynch & Park)
- Discretizing a high-dimensional space is neither easy nor efficient – other planning approaches are used





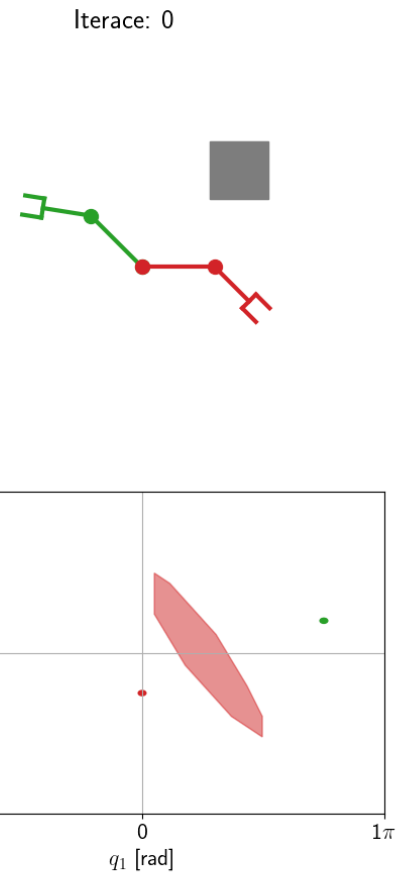
Path planning in manipulator robotics

Planners #2: sampling-based methods

- Based on
  - random sampling of $\mathcal{C}$
  - a function that decides whether a sample is collision-free
  - a local planner that finds a path to the random sample
- The graph is built randomly $\times$ the graph is built by discretization
- Less computationally demanding than the grid-based methods
- Probabilistically complete – the probability of finding a solution approaches 1 as the number of samples approaches infinity
- Random trees (RT) and probabilistic roadmaps (PRM)

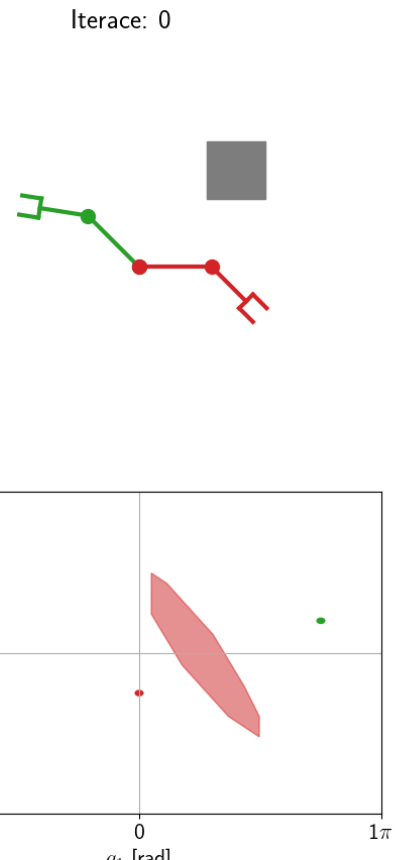
Random trees: the naive approach

1: Input: start q_s, goal q_g, maximum edge length Δq



Random trees: the naive approach

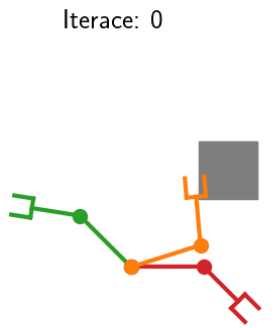
1: Input: start q_s, goal q_g, maximum edge length Δq
2: $\mathcal{V} \leftarrow \{q_s\}, \quad \mathcal{E} \leftarrow \{\}$

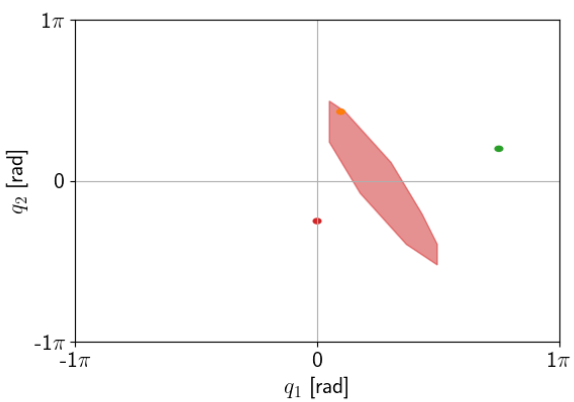


Random trees: the naive approach

1: Input: start q_s, goal q_g, maximum edge length Δq
2: $\mathcal{V} \leftarrow \{q_s\}, \quad \mathcal{E} \leftarrow \{\}$
3: **for** $k = 1$ **to** M **do**
4: $\quad q_{\text{rand}} \sim \mathcal{C}$

14: **end for**

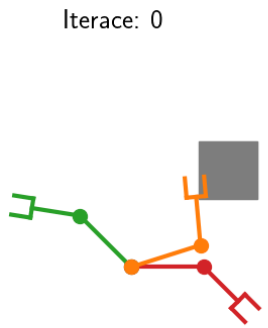


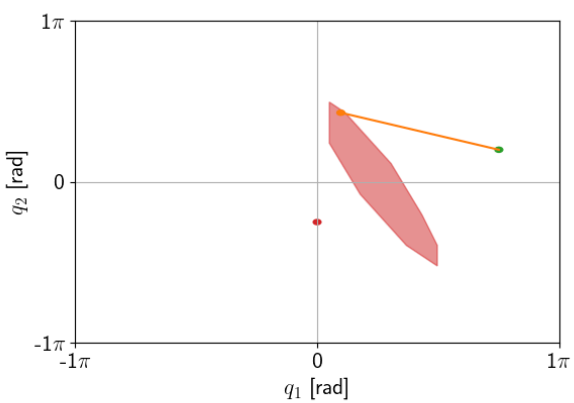


Random trees: the naive approach

1: Input: start q_s, goal q_g, maximum edge length Δq
2: $\mathcal{V} \leftarrow \{q_s\}, \quad \mathcal{E} \leftarrow \{\}$
3: **for** $k = 1$ **to** M **do**
4: $\quad q_{\text{rand}} \sim \mathcal{C}$
5: $\quad q_{\text{tree}} \sim \mathcal{V}$

14: **end for**

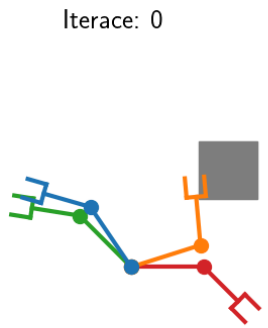


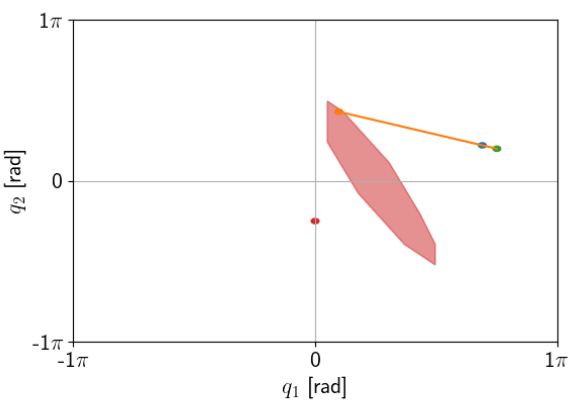


Random trees: the naive approach

1: Input: start q_s, goal q_g, maximum edge length Δq
2: $\mathcal{V} \leftarrow \{q_s\}, \quad \mathcal{E} \leftarrow \{\}$
3: **for** $k = 1$ **to** M **do**
4: $\quad q_{\text{rand}} \sim \mathcal{C}$
5: $\quad q_{\text{tree}} \sim \mathcal{V}$
6: $\quad q_{\text{new}} \leftarrow f_{\text{plan}}(q_{\text{tree}}, q_{\text{rand}}, \Delta q)$

14: **end for**

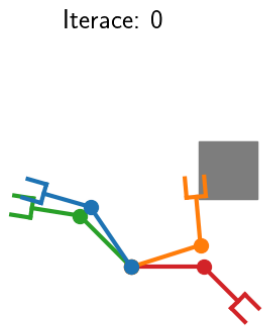


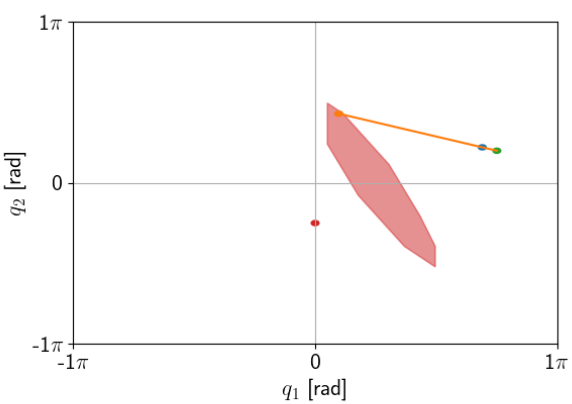


Random trees: the naive approach

1: Input: start q_s, goal q_g, maximum edge length Δq
2: $\mathcal{V} \leftarrow \{q_s\}, \quad \mathcal{E} \leftarrow \{\}$
3: **for** $k = 1$ **to** M **do**
4: $\quad q_{\text{rand}} \sim \mathcal{C}$
5: $\quad q_{\text{tree}} \sim \mathcal{V}$
6: $\quad q_{\text{new}} \leftarrow f_{\text{plan}}(q_{\text{tree}}, q_{\text{rand}}, \Delta q)$
7: $\quad$ **if** $q_{\text{new}} \in \mathcal{C}_{\text{free}}$ **then**
8: $\quad\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{q_{\text{new}}\}, \quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(q_{\text{tree}}, q_{\text{new}})\}$
9: $\quad$ **end if**

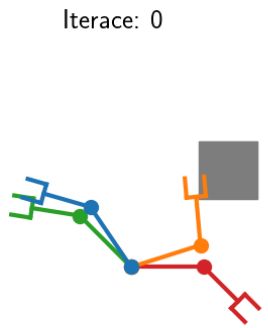
14: **end for**

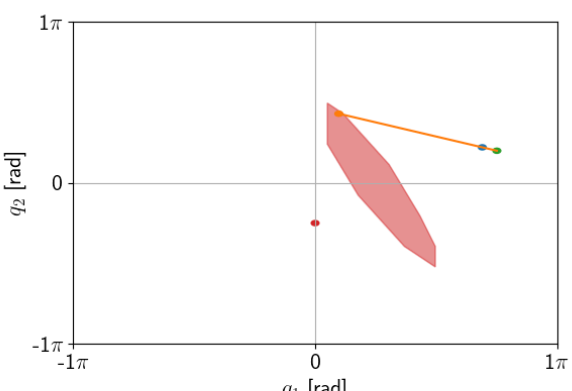




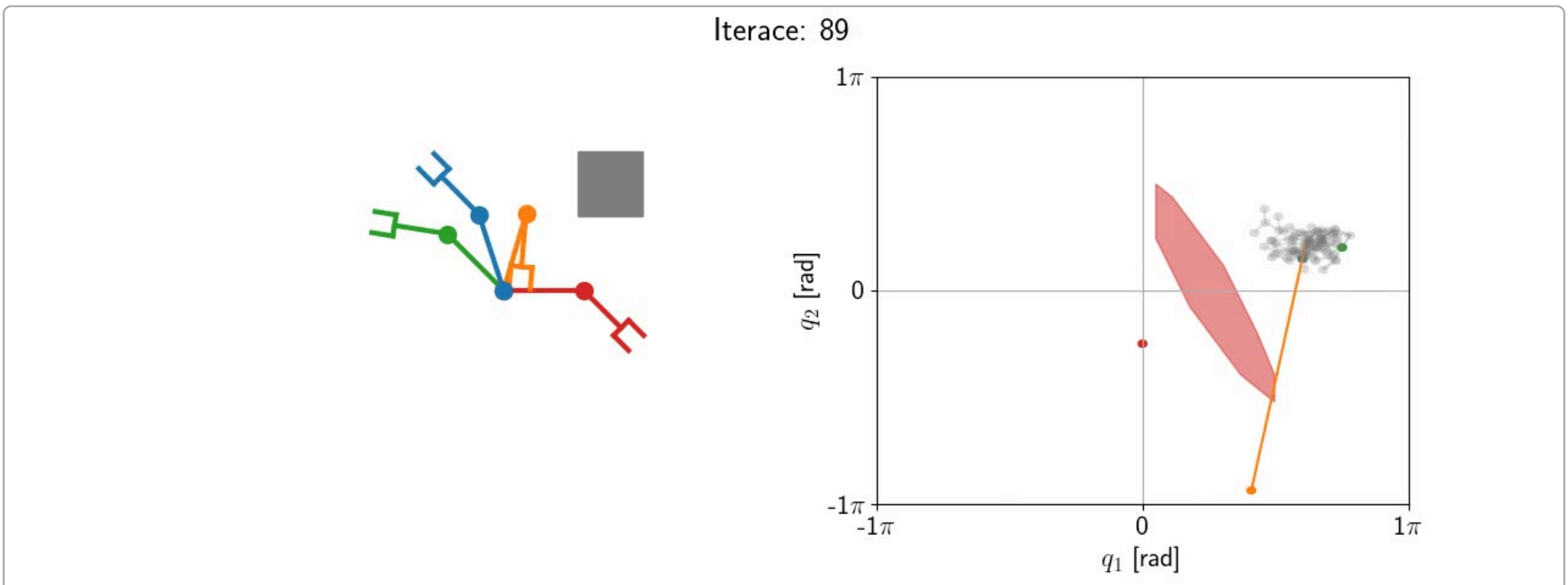
Random trees: the naive approach

1: Input: start q_s, goal q_g, maximum edge length Δq
2: $\mathcal{V} \leftarrow \{q_s\}, \quad \mathcal{E} \leftarrow \{\}$
3: **for** $k = 1$ **to** M **do**
4: $\quad q_{\text{rand}} \sim \mathcal{C}$
5: $\quad q_{\text{tree}} \sim \mathcal{V}$
6: $\quad q_{\text{new}} \leftarrow f_{\text{plan}}(q_{\text{tree}}, q_{\text{rand}}, \Delta q)$
7: $\quad$ **if** $q_{\text{new}} \in \mathcal{C}_{\text{free}}$ **then**
8: $\quad\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{q_{\text{new}}\}, \quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(q_{\text{tree}}, q_{\text{new}})\}$
9: $\quad$ **end if**
10: $\quad$ **if** $\|q_{\text{new}} - q_g\| < \varepsilon$ **then**
11: $\quad\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{q_g\}, \quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(q_{\text{new}}, q_g)\}$
12: $\quad\quad$ **break**
13: $\quad$ **end if**
14: **end for**





Random trees



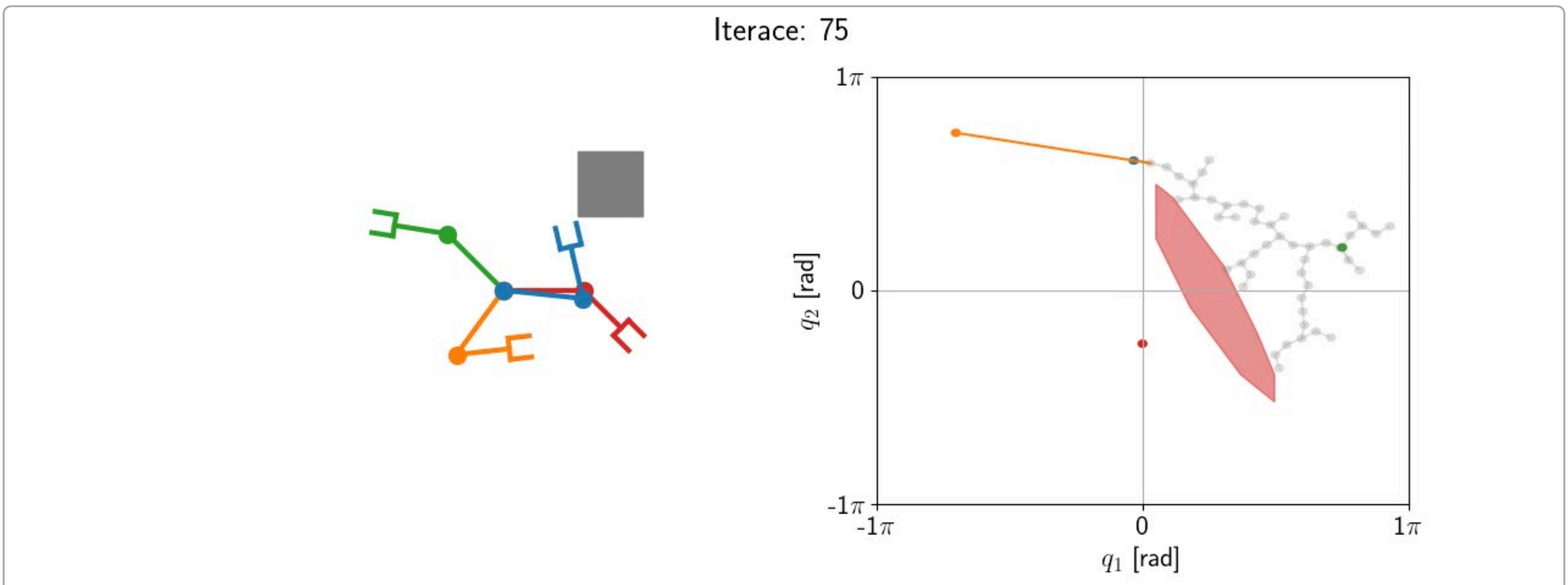
Rapidly-exploring random trees (RRT)

- Picking the tree vertex at random does not support exploration
- RRT (rapidly-exploring random tree) was designed for exploration
- To compute q_{new}, the random configuration is "connected" to the nearest vertex of the tree

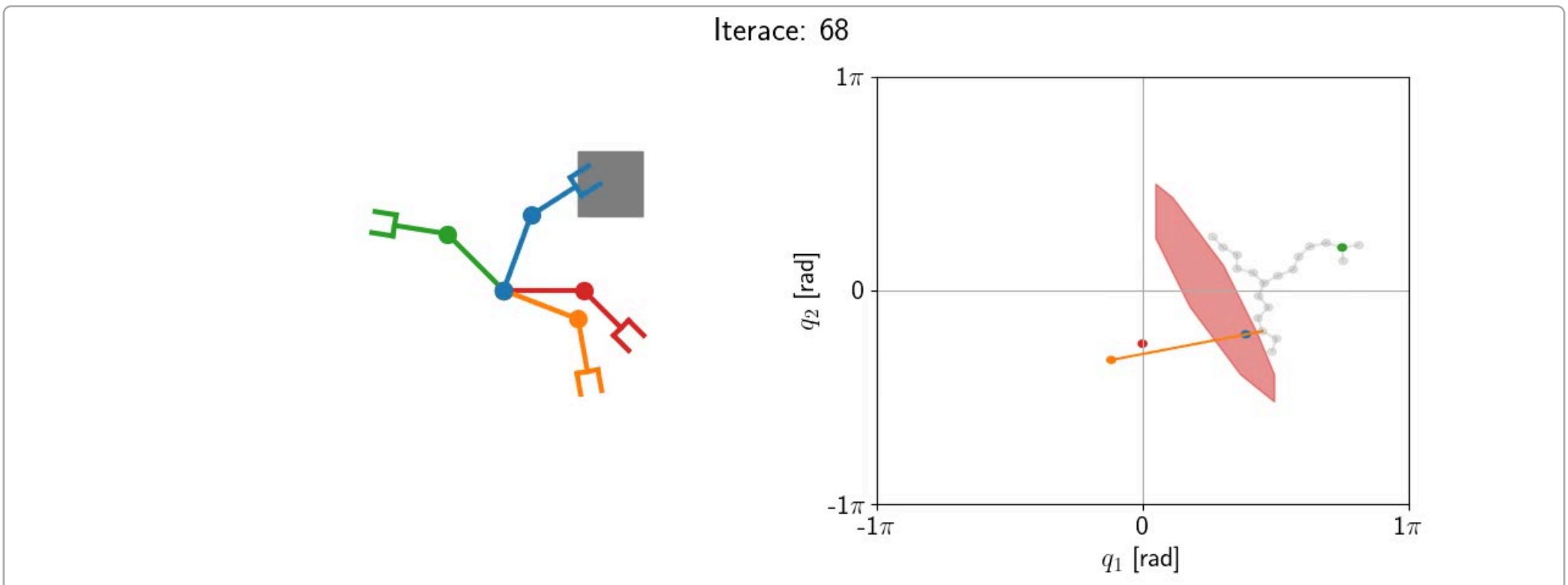
$$q_{\text{tree}} = \arg\min_{q \in \mathcal{V}} \|q - q_{\text{rand}}\| \qquad \cancel{q_{\text{tree}} \sim \mathcal{V}}$$

LaValle and Kuffner, *Randomized Kinodynamic Planning*, IJRR 2001

RRT



RRT: sampling bias



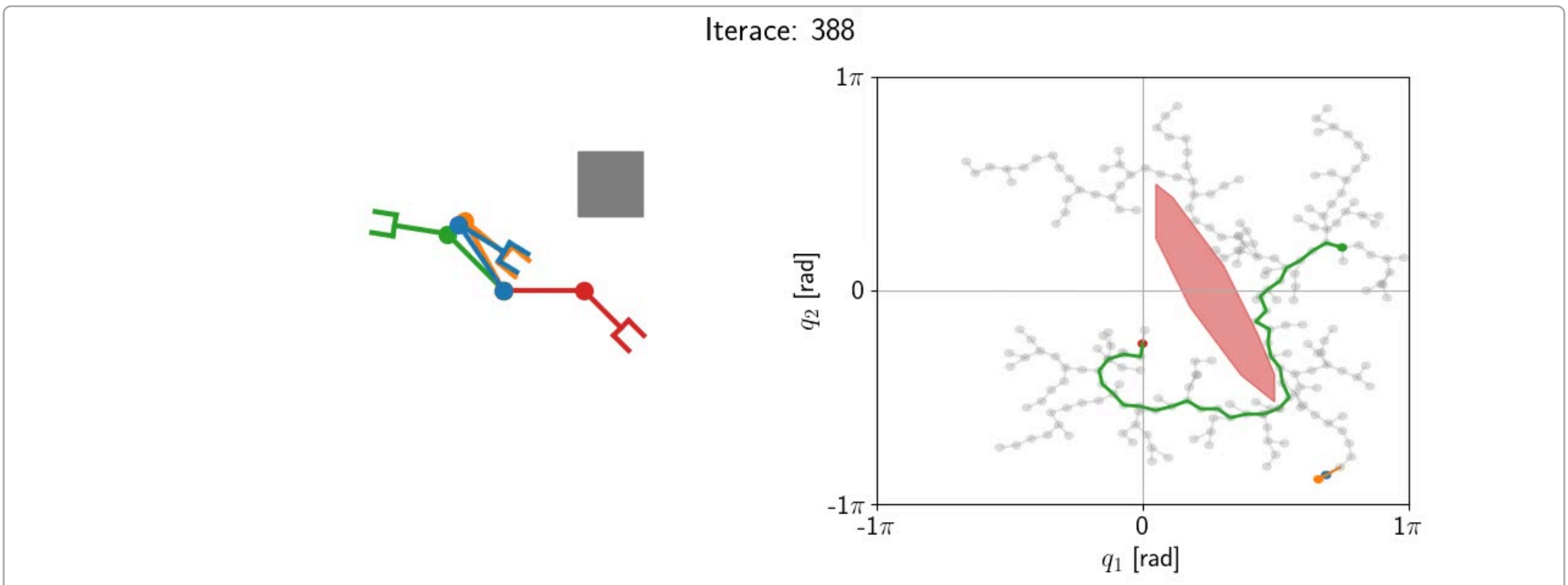
RRT

- The resulting path is not optimal – but often any solution is enough
- RRT is efficient even in high dimensions
- Suitable sampling increases the search efficiency (304 vs 257 iterations)
- Many modifications of the RRT algorithm exist
  - RRT*, BiDirectional RRT, RRG, PDRRTs, RRT*-Smart, A*-RRT, RRT*FN, RRT*-AR, RT-RRT*, RRT#, Theta*-RRT, RRT* FND, RRT-GPU, RRdT*

RRT*

- Improves the path using further samples
- Slightly slower than RRT (additional operations to maintain the tree)
- A termination criterion has to be defined
  - maximum number of samples
  - planning time

RRT*

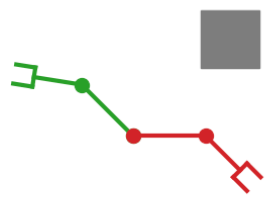


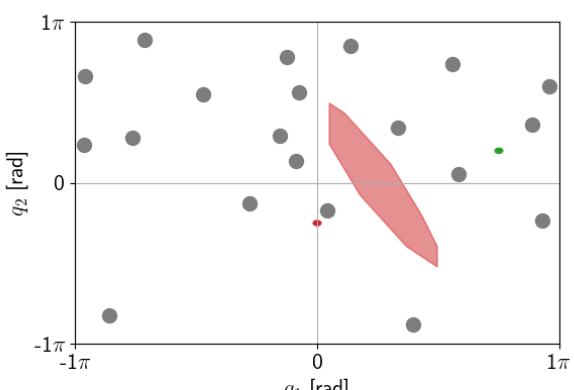
Probabilistic roadmaps (PRM)

- Builds an undirected graph covering $\mathcal{C}_{\text{free}}$ before the plan is searched for
- The graph is then searched by A*
- PRM vs RRT:
  - the graph need not be a tree
  - the built graph can be used repeatedly
  - multi-query vs single-query

PRM: building the graph

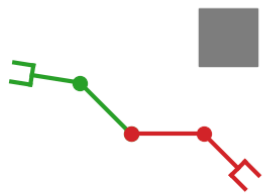
1: $\mathcal{V} \leftarrow \{q_i \mid q_i \sim \mathcal{C}_{\text{free}}\}_{i=0}^{M}$

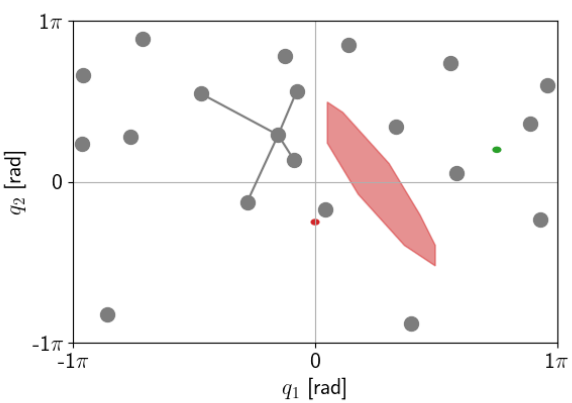




PRM: building the graph

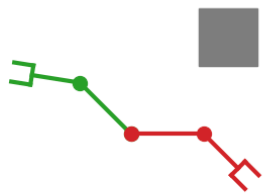
$\mathcal{V} \leftarrow \{q_i \mid q_i \sim \mathcal{C}_{\text{free}}\}_{i=0}^{M}$
$\mathcal{E} \leftarrow \{\}$
for $q_i \in \mathcal{V}$ **do**
    $\mathcal{N}(q_i) \leftarrow k$ nearest neighbours of q_i
    for $q_j \in \mathcal{N}(q_i)$ **do**
        if $\text{plan}(q_i, q_j) \in \mathcal{C}_{\text{free}} \wedge (q_i, q_j) \notin \mathcal{E}$ **then**
            $\mathcal{E} \leftarrow \mathcal{E} \cup \{(q_i, q_j)\}$
        end if
    end for
end for

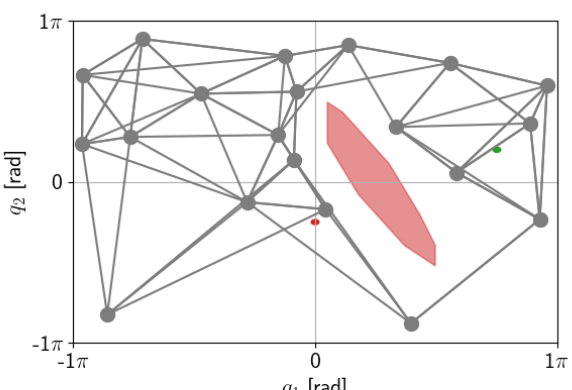




PRM: building the graph

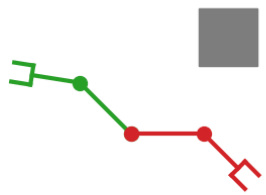
1: $\mathcal{V} \leftarrow \{q_i \mid q_i \sim \mathcal{C}_{\text{free}}\}_{i=0}^{M}$
2: $\mathcal{E} \leftarrow \{\}$
3: **for** $q_i \in \mathcal{V}$ **do**
4: $\quad \mathcal{N}(q_i) \leftarrow k$ nearest neighbours of q_i
5: $\quad$ **for** $q_j \in \mathcal{N}(q_i)$ **do**
6: $\quad\quad$ **if** $\text{plan}(q_i, q_j) \in \mathcal{C}_{\text{free}} \wedge (q_i, q_j) \notin \mathcal{E}$ **then**
7: $\quad\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(q_i, q_j)\}$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: **end for**

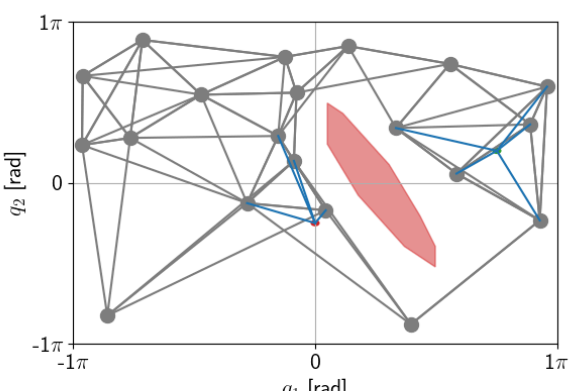




PRM: building the graph

1: $\mathcal{V} \leftarrow \{q_i \mid q_i \sim \mathcal{C}_{\text{free}}\}_{i=0}^{M}$
2: $\mathcal{E} \leftarrow \{\}$
3: **for** $q_i \in \mathcal{V}$ **do**
4: $\quad \mathcal{N}(q_i) \leftarrow k$ nearest neighbours of q_i
5: $\quad$ **for** $q_j \in \mathcal{N}(q_i)$ **do**
6: $\quad\quad$ **if** $\text{plan}(q_i, q_j) \in \mathcal{C}_{\text{free}} \wedge (q_i, q_j) \notin \mathcal{E}$ **then**
7: $\quad\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(q_i, q_j)\}$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: **end for**

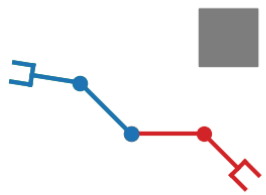


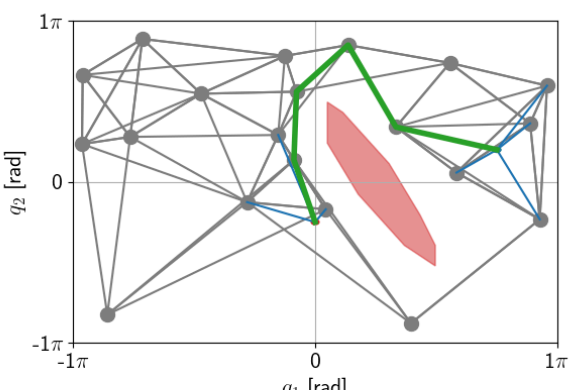


- When planning, q_s and q_g are inserted into the graph

PRM: building the graph

1: $\mathcal{V} \leftarrow \{q_i \mid q_i \sim \mathcal{C}_{\text{free}}\}_{i=0}^{M}$
2: $\mathcal{E} \leftarrow \{\}$
3: **for** $q_i \in \mathcal{V}$ **do**
4: $\quad \mathcal{N}(q_i) \leftarrow k$ nearest neighbours of q_i
5: $\quad$ **for** $q_j \in \mathcal{N}(q_i)$ **do**
6: $\quad\quad$ **if** $\text{plan}(q_i, q_j) \in \mathcal{C}_{\text{free}} \wedge (q_i, q_j) \notin \mathcal{E}$ **then**
7: $\quad\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(q_i, q_j)\}$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: **end for**

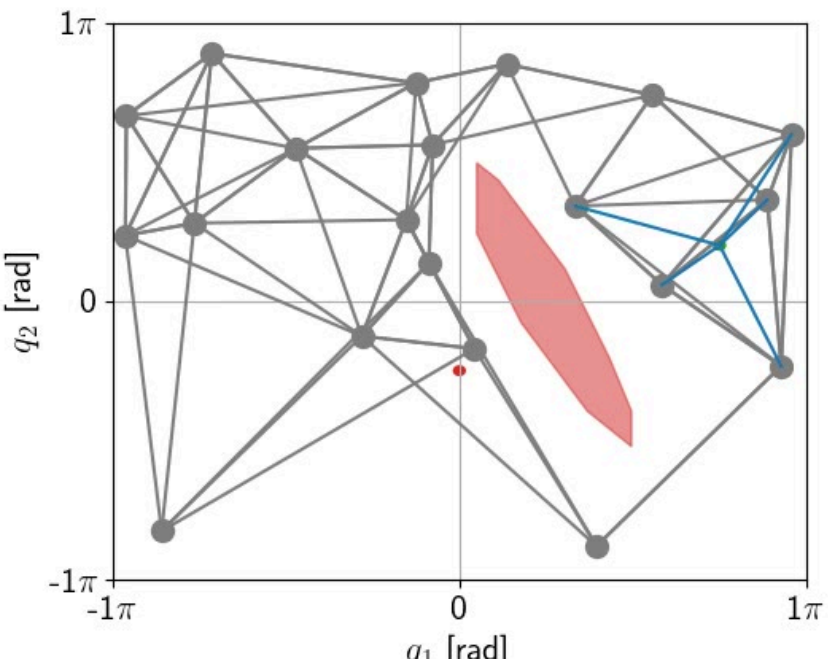




- When planning, q_s and q_g are inserted into the graph
- The path is found by graph search

PRM





What would planning look like for a real manipulator?

Planning for a real manipulator

- Franka Emika Panda robot
- Collision computation is more complicated in 3D
  - input: q and the kinematic model including the 3D models of the links
  - output: is the configuration in collision?
- Otherwise RRT/PRM can be applied without change
- The searched space has more dimensions (7 DoF + 1 DoF)

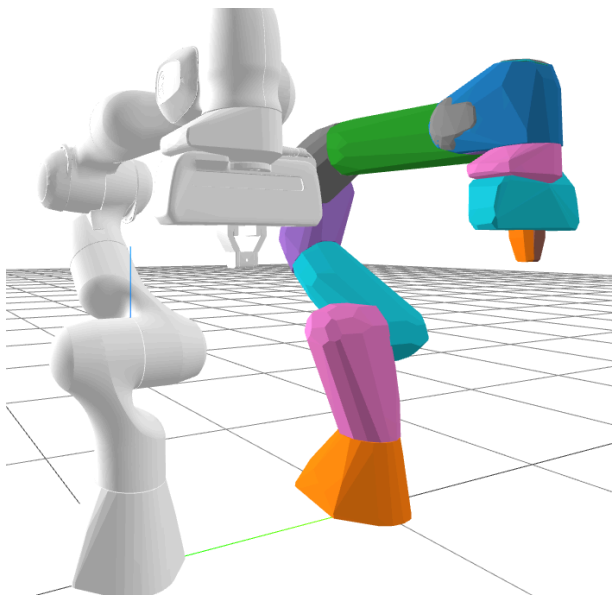


Collision computation

- We look for the position and orientation of every link
  - forward kinematics
  - the collision is computed for every link separately

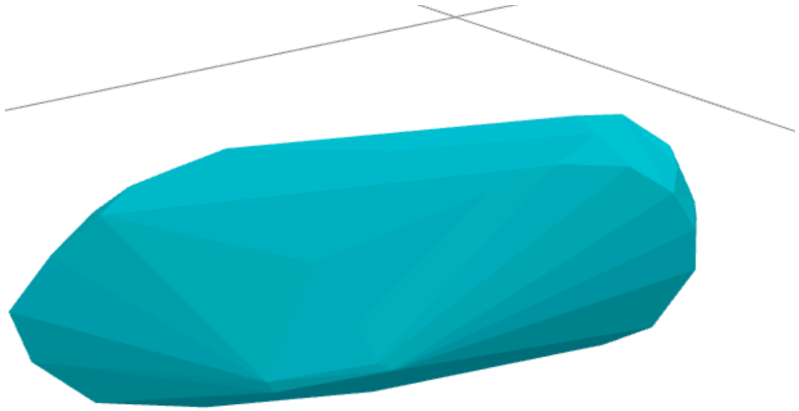
Collision computation

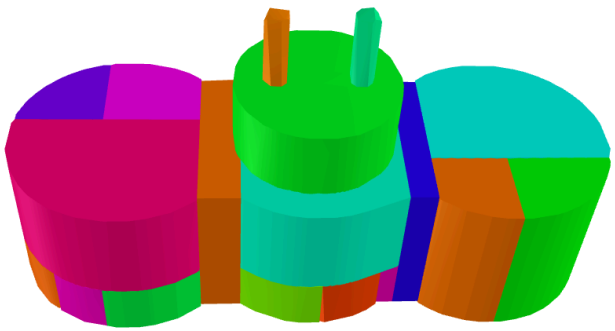
- We look for the position and orientation of every link
  - forward kinematics
  - the collision is computed for every link separately
- 3D models are usually very detailed (and non-convex)
  - a convex approximation is computed to speed up the computation
  - convex hull or convex decomposition
  - the convex decomposition is computed for the objects of the scene as well
- Collision-freeness of two convex objects can be verified e.g. by the GJK algorithm



Convex decomposition examples

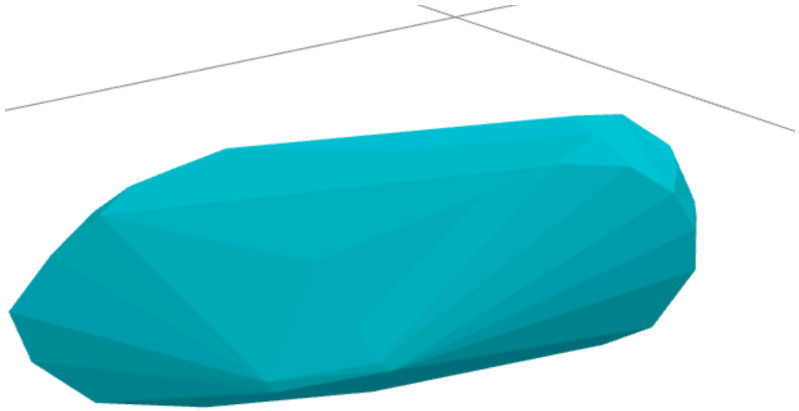


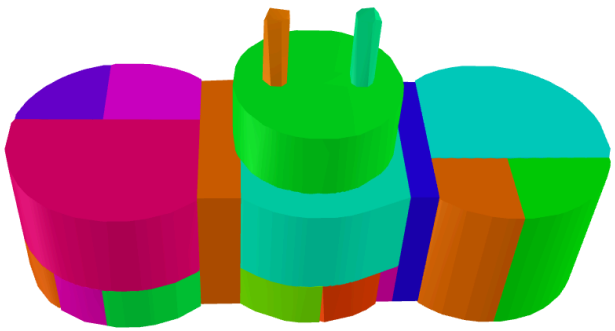




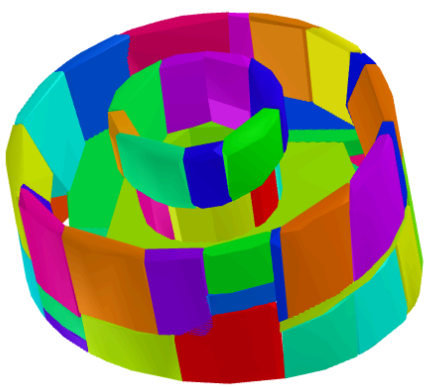
Convex decomposition examples

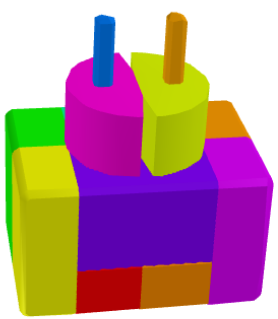












Collisions between links



- Are the links in the figure in collision?

Collisions between links

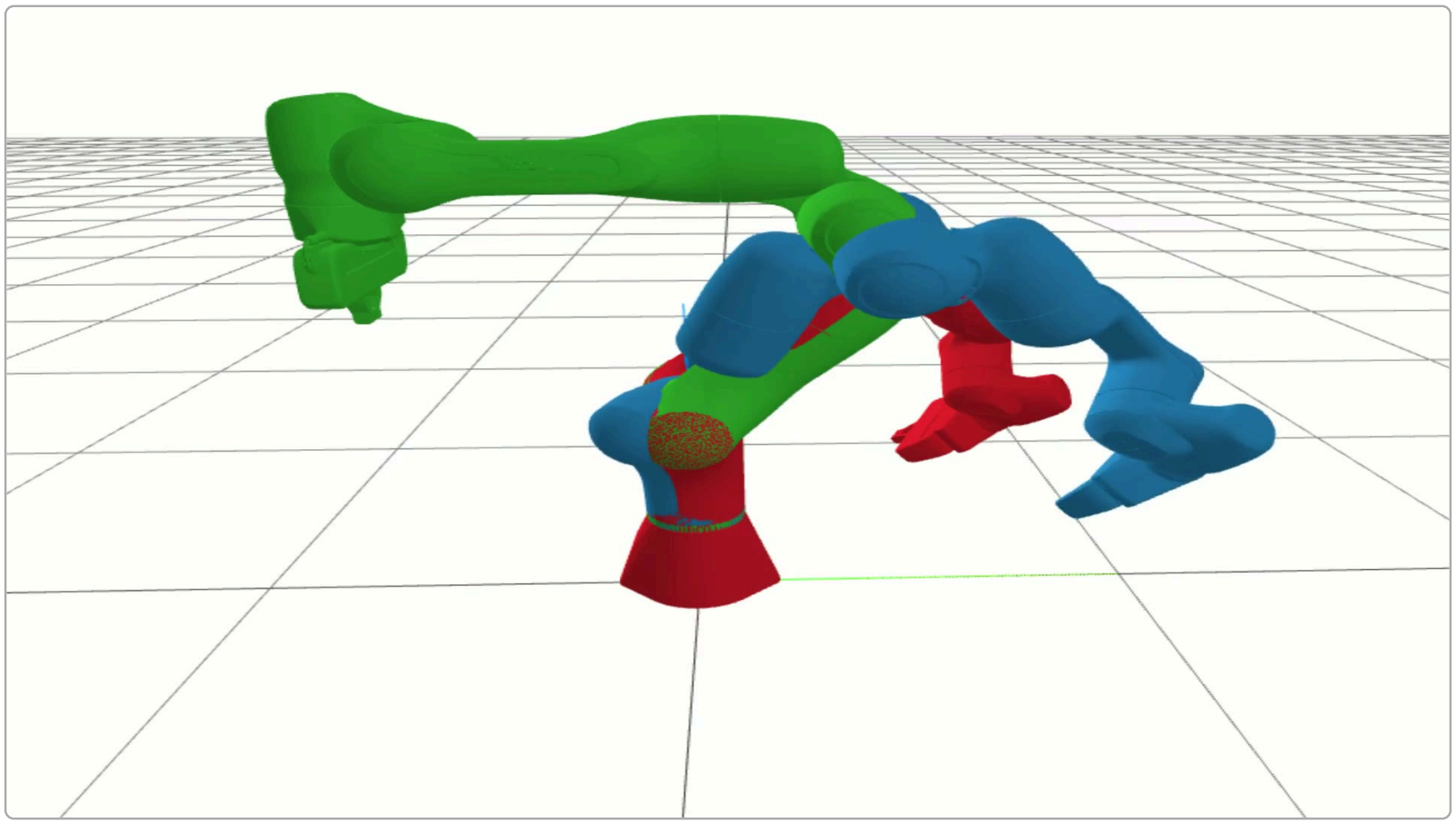


- Are the links in the figure in collision?
- Collision computation is time-consuming (especially for objects that are almost in contact)
- We try to reduce the number of collision pairs
- In practice we disable collisions between objects that
  - are always in collision, e.g. neighbouring links
  - can never be in collision, e.g. separate static collision objects
  - ROS: MoveIt! detects these collision pairs when the configuration files are created

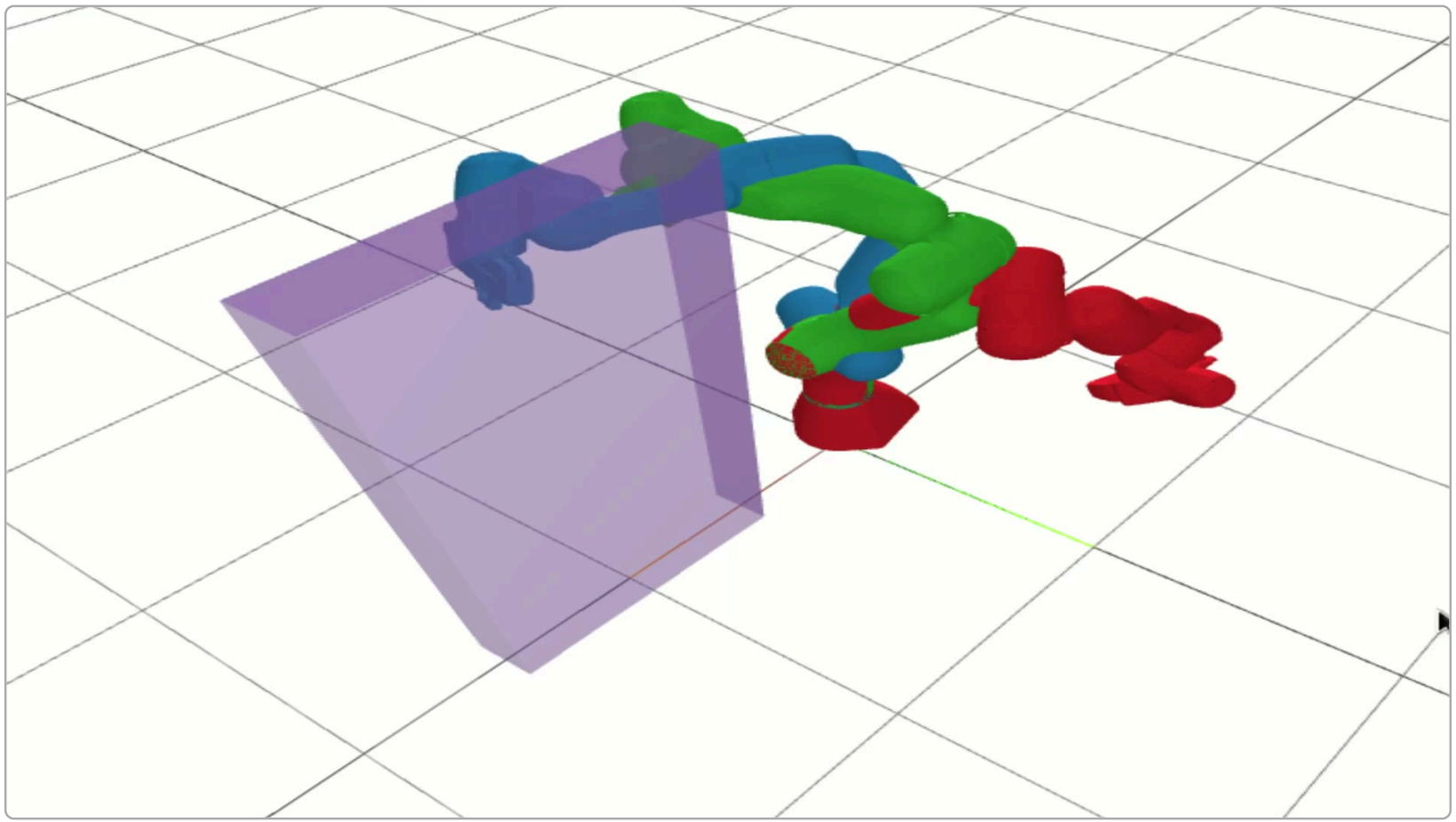
Collision detection for a robot: conclusion

- Preparation phase
  - compute the convex hulls
  - define the collision pairs
- Computation phase
  - compute the position and orientation of every link by forward kinematics
  - check collision-freeness for every collision pair
  - if any pair is in collision, the configuration is inadmissible

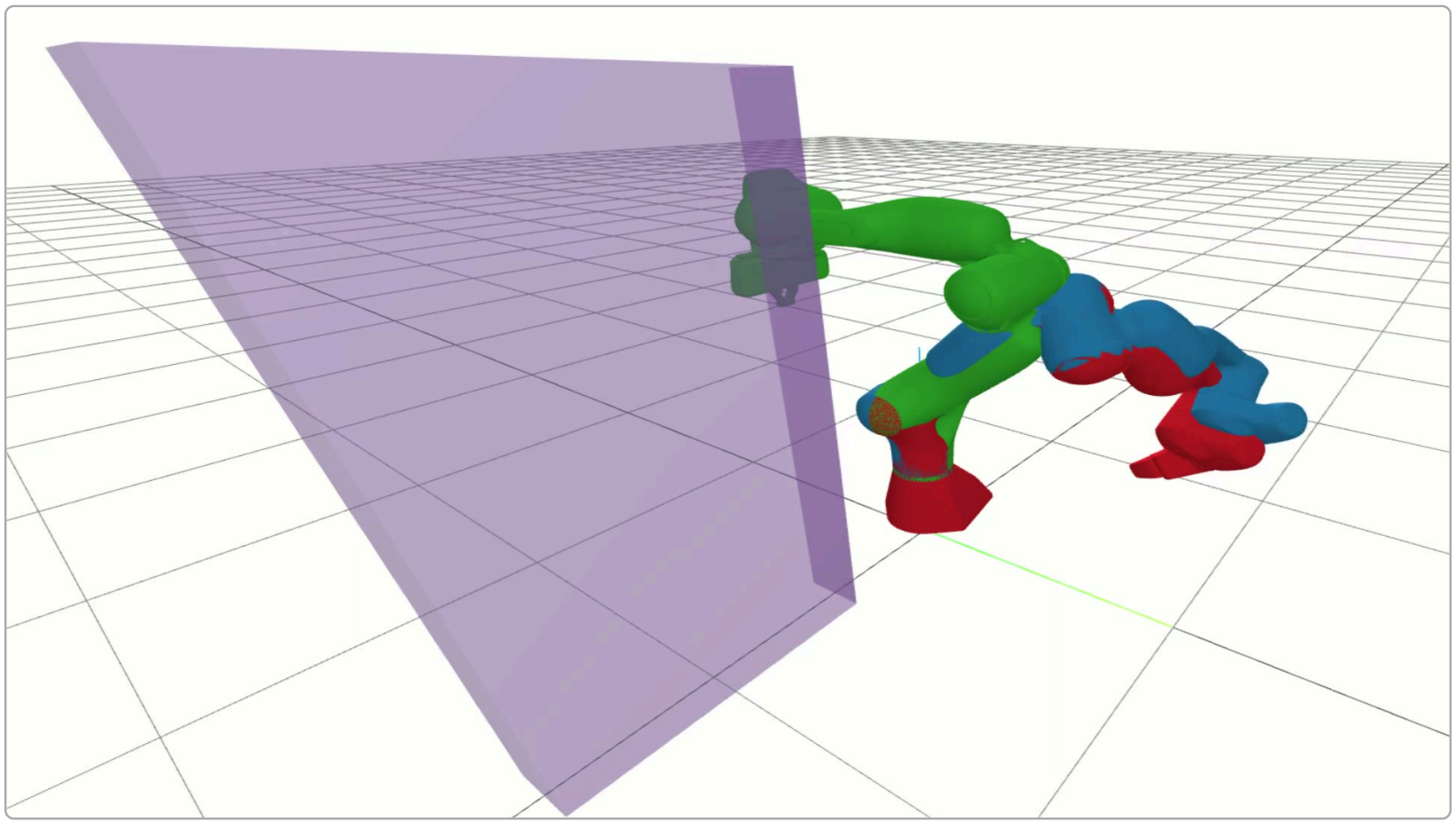
Planned motion



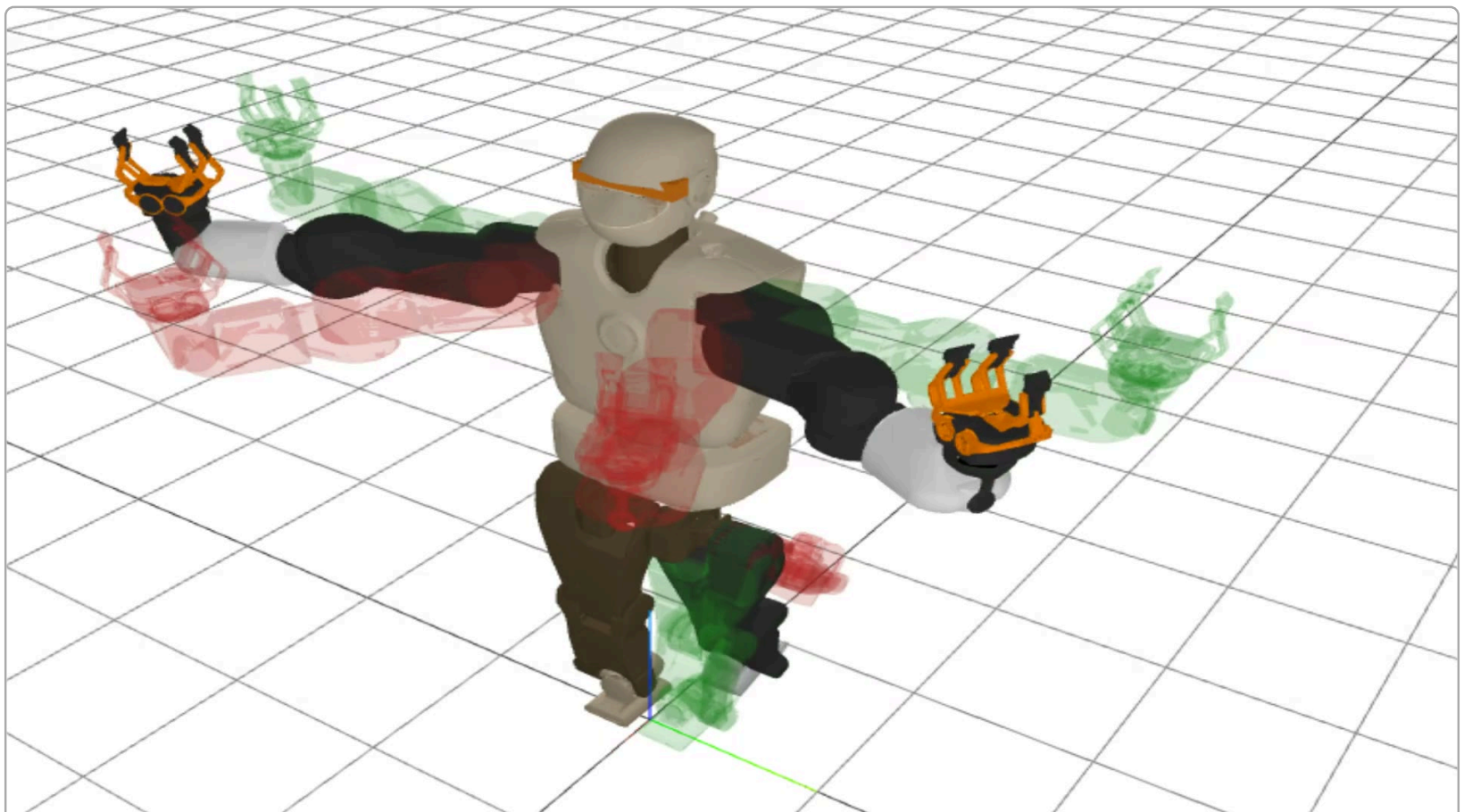
Planned motion: RRT – 3132 iterations



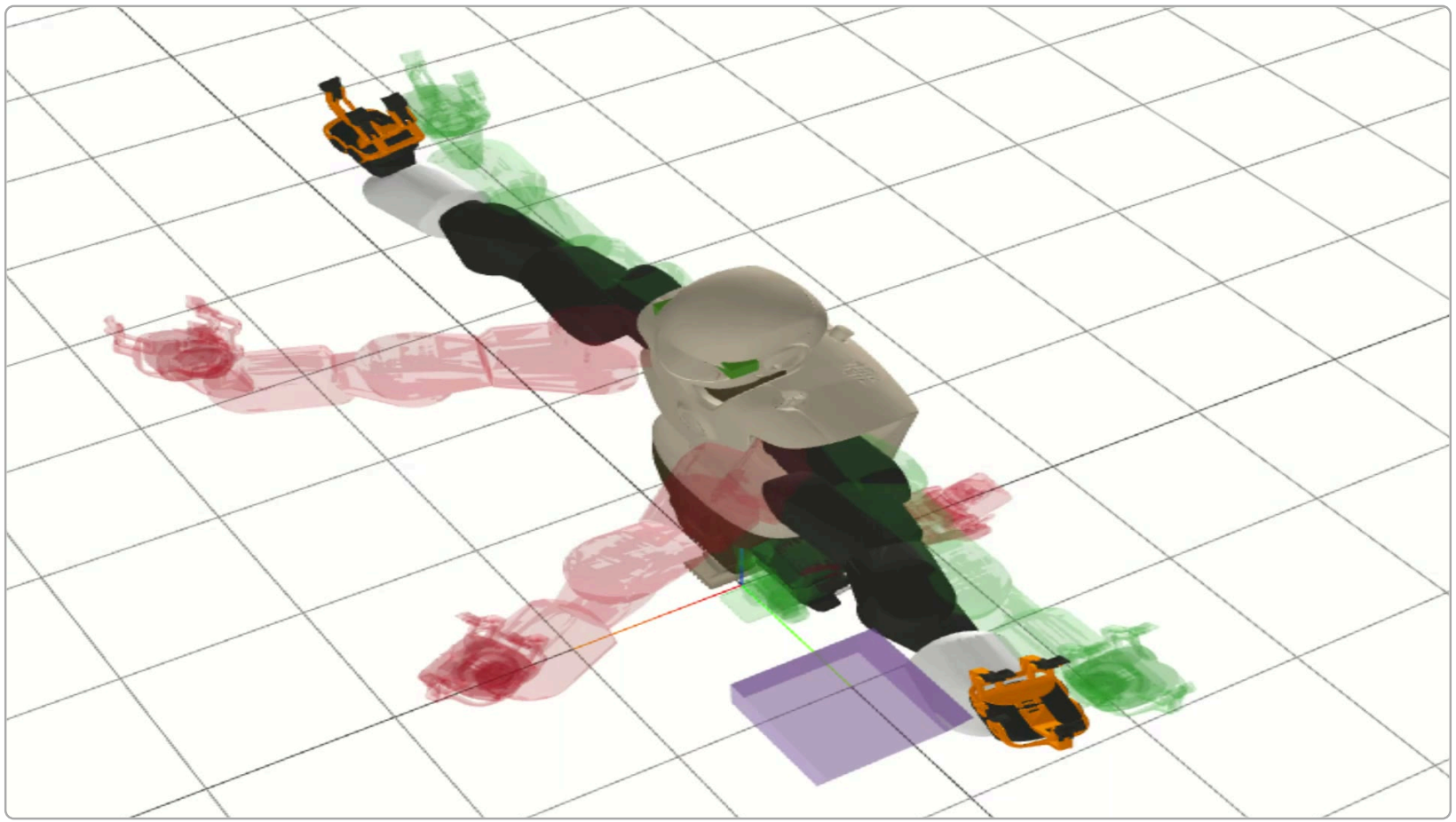
Planned motion: RRT* – 10000 iterations



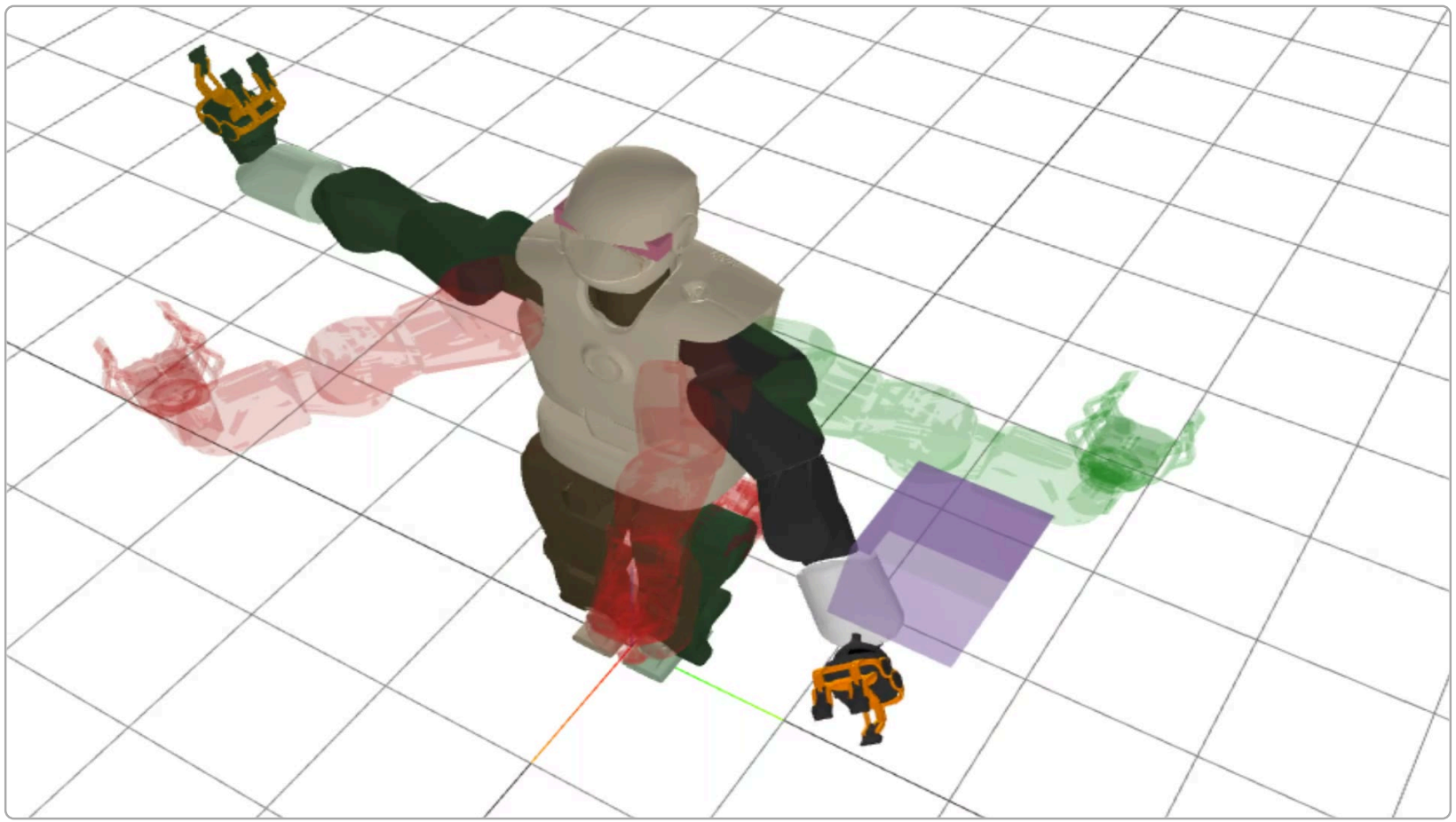
Planned motion: Talos



Planned motion



Planned motion



Path shortening: the random shortcut method

- Random shortcut
- Shortens the path using random samples and straight-line connections
- Algorithm:
  - pick a random point q_1 on the path
  - pick a random point q_2 on the path
  - if the interpolation between q_1 and q_2 is collision-free, replace the path between q_1 and q_2 by the straight line in the configuration space

Summary

- What the configuration space is
- The difference between a path and a trajectory
- What path planning is
- The difference between the planning methods:
  - discretized grid
  - RRT
  - RRT*
  - PRM
- What was left out:
  - there are many more planners
  - optimization-based planning methods

Laboratory

- This week (10.–11. 11.): inverse kinematics, hw05 and opt_hw05 due 22. 11.
  - numerical IK in $SE(2)$
  - analytical IK in $SE(2)$ for the RRR manipulator
  - analytical IK in $SE(2)$ for the PRR manipulator (optional)
- Next week (17.–18. 11.): individual work on the final project, consultations
- In two weeks (24.–25. 11.): motion planning, opt_hw06 due 6. 12.
  - implementation of PRM
  - homework: implementation of RRT and random shortcut (optional)