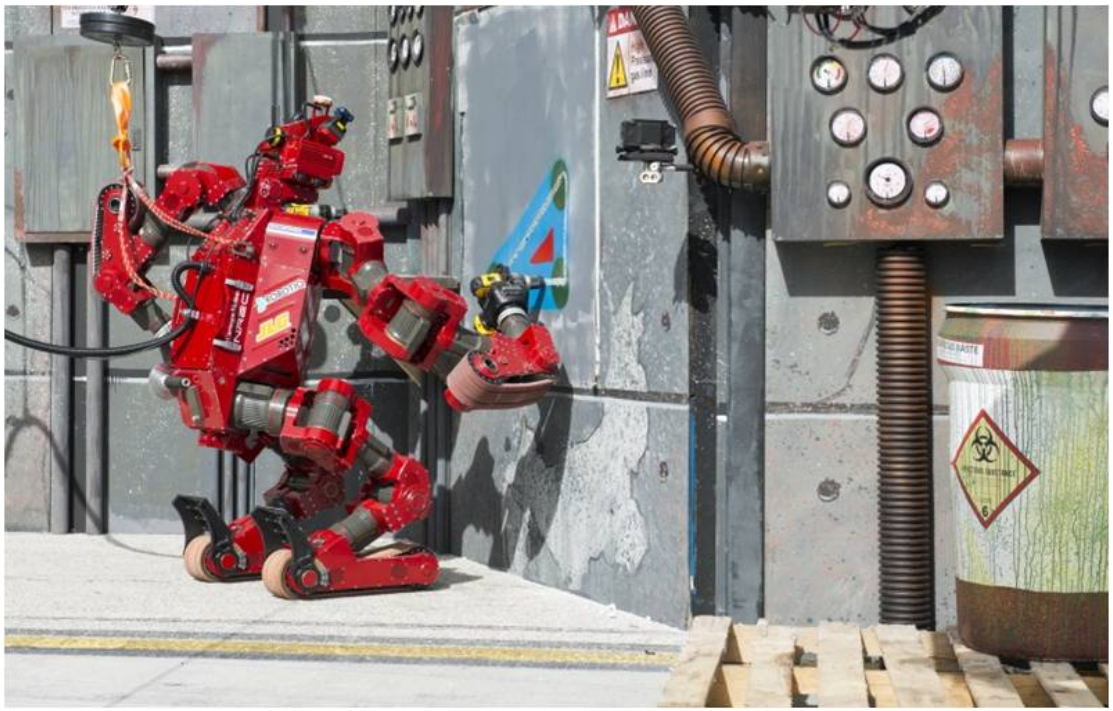
Algorithmic Motion Planning (236610)
Lecture 11—Task Space Planning

Oren Salzman

Computer Science Department, Technion

Today's lecture—Following Paths in Task Space

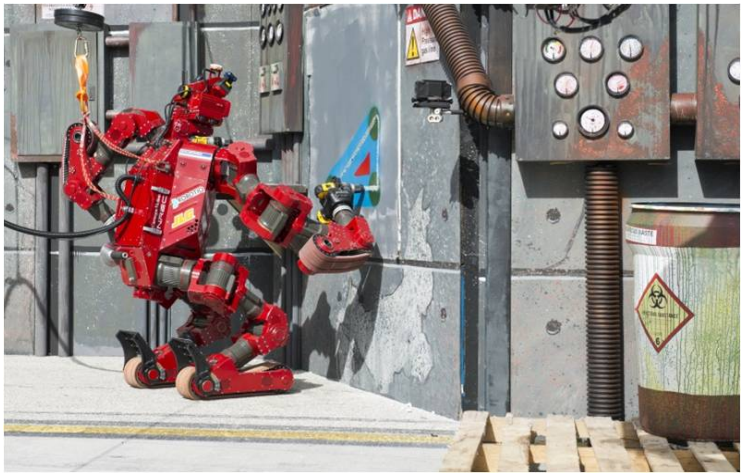
- Motivation & Problem definition
- Algorithmic approach
- Design optimization



Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



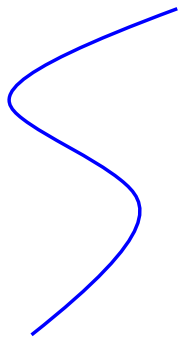
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



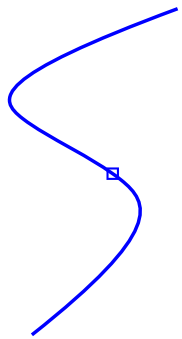
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



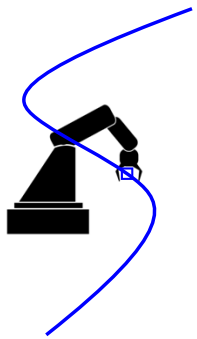
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



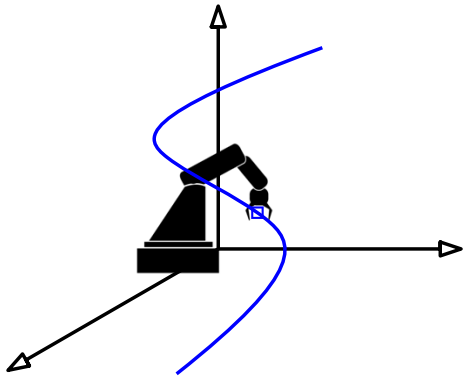
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



Task Space

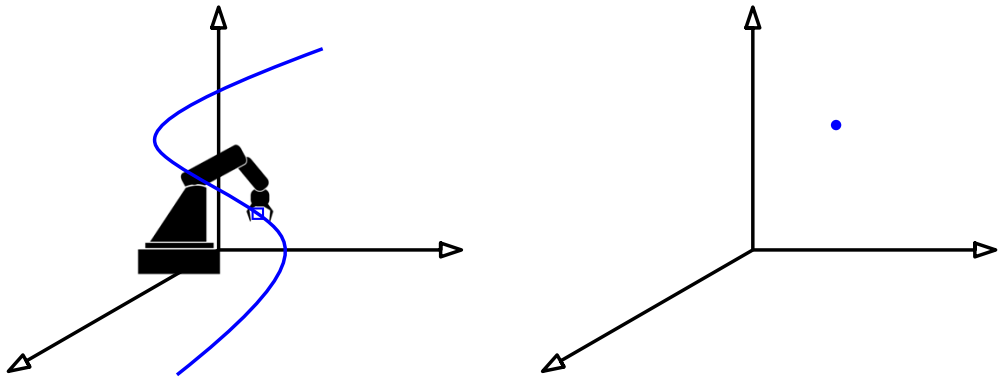
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



Task Space

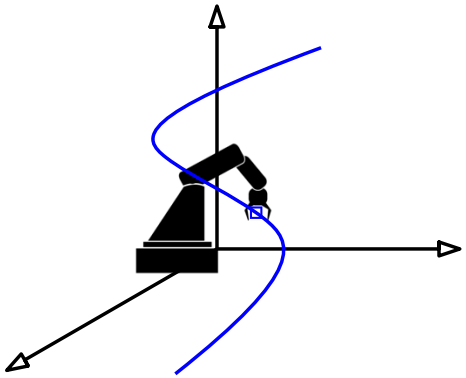
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

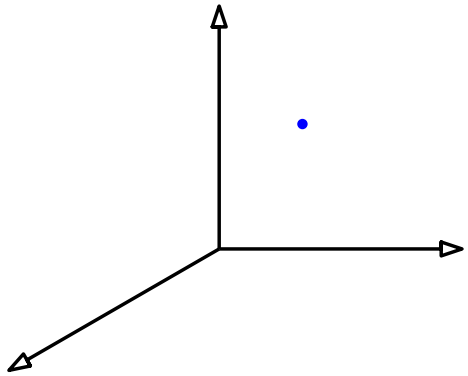
Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



Task Space



Configuration Space

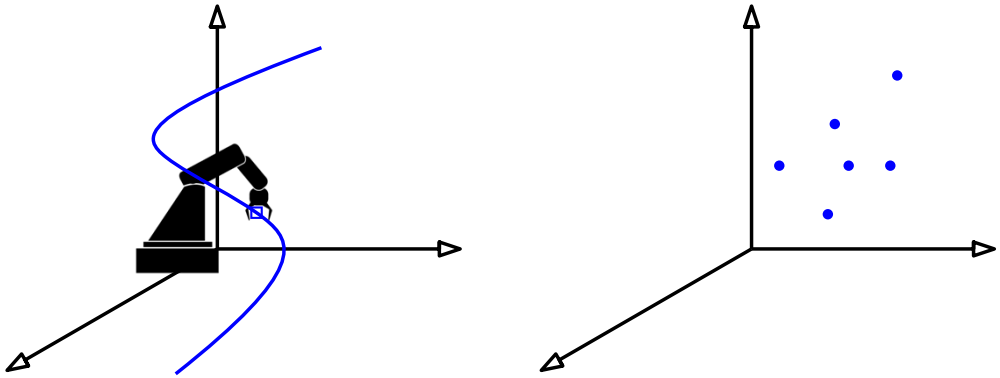
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



Task Space | Configuration Space

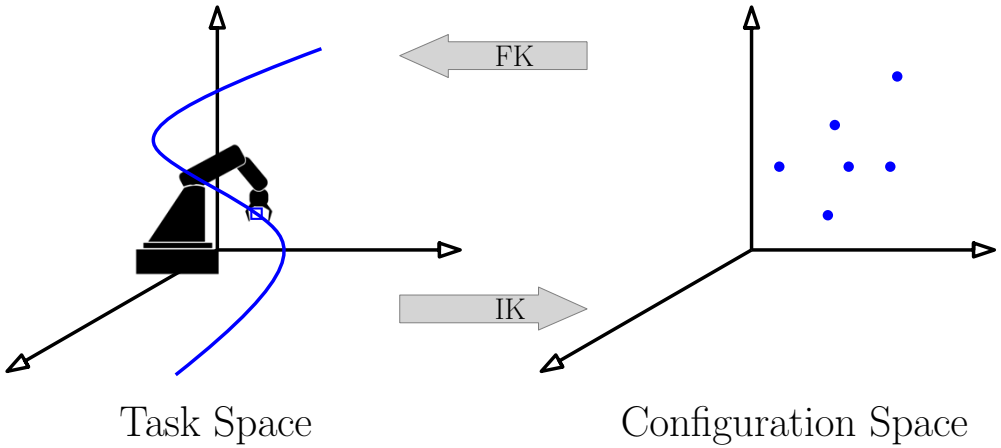
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

Motivation & Problem definition

(informal) definition of Task space

The space describing all poses of the robot's end effector



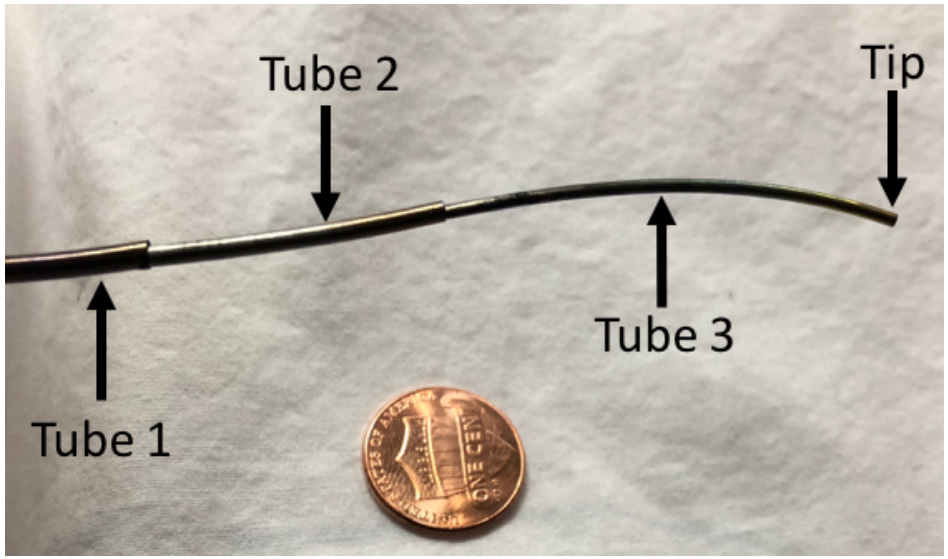
Following task-space paths

Given a path ζ_{ref} in task space, compute a collision-free C-space path that maps "as closely" as possible to ζ_{ref}

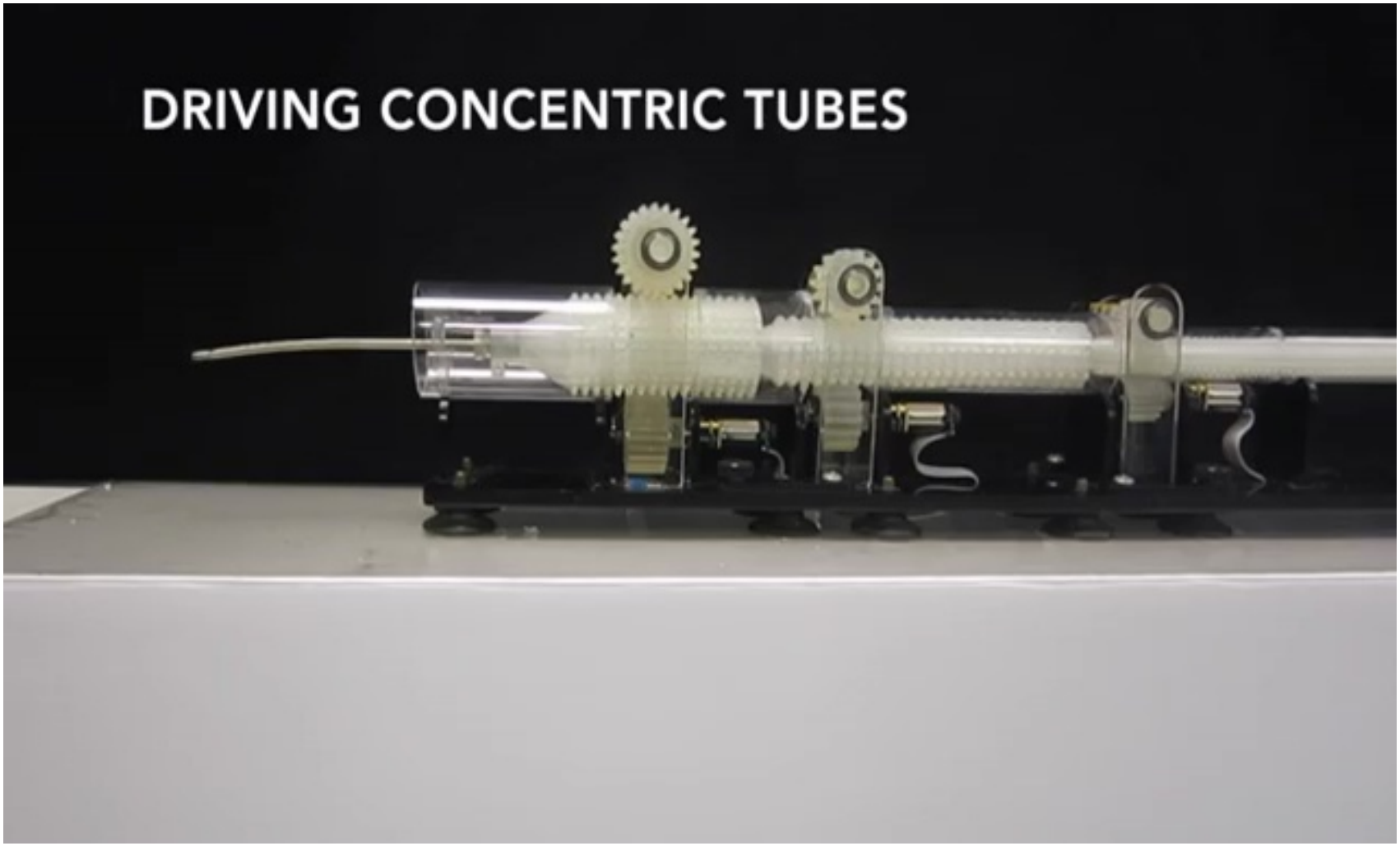
Motivation & Problem definition

Concentric-tube robots—tentacle-like medical devices composed of thin, pre-curved, telescoping tubes

- Controlled by axially rotating and/or telescopically translating individual tubes

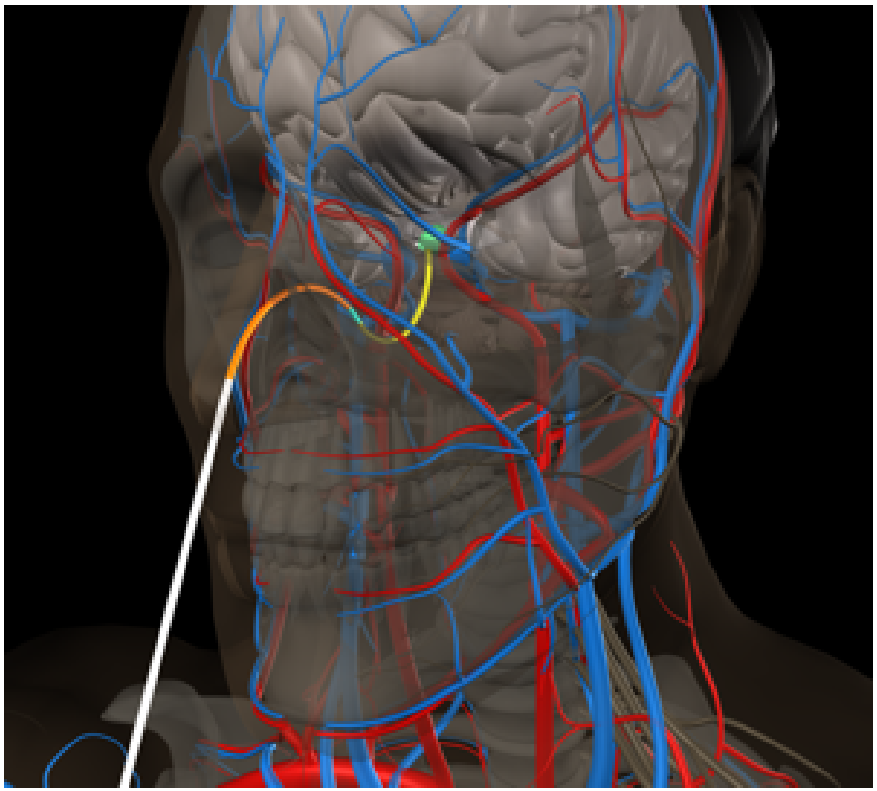


Motivation & Problem definition



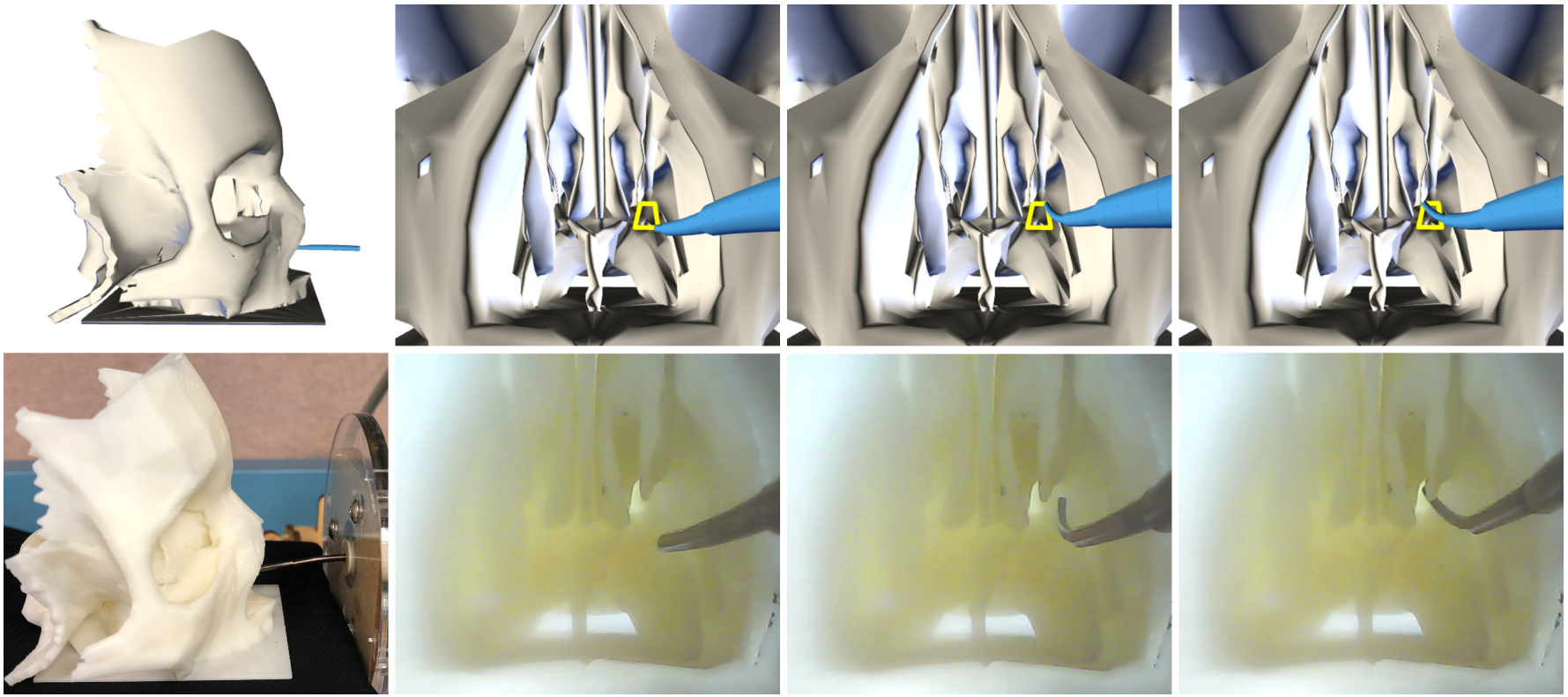
Motivation & Problem definition

Motivation & Problem definition



Use motion planning to follow surgeon's desired path as "closely-as-possible"

Motivation & Problem definition



Use motion planning to follow surgeon's desired path as "closely-as-possible"

Motivation & Problem definition

$$\zeta^* = \arg\min_{\zeta \in \mathcal{C}_{\text{free}}} ||\text{FK}(\zeta), \zeta_{\text{ref}}||$$

Motivation & Problem definition

$$\zeta^* = \arg\min_{\zeta \in \mathcal{C}_{\text{free}}} \underbrace{||\text{FK}(\zeta), \zeta_{\text{ref}}||}$$

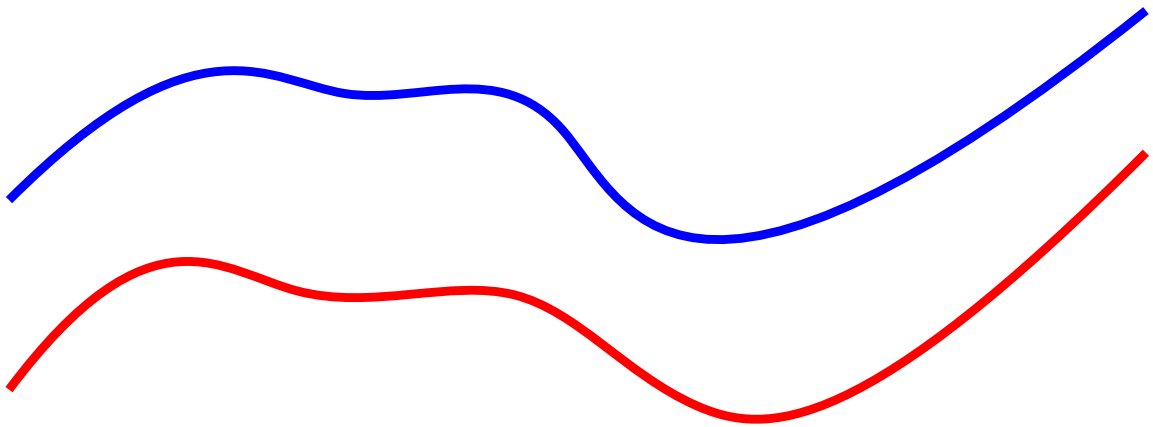
How do we measure distances ?

Motivation & Problem definition

$$\zeta^* = \underbrace{\arg\min_{\zeta \in \mathcal{C}_{\text{free}}}}_{\text{How do we plan ?}} \underbrace{||\text{FK}(\zeta), \zeta_{\text{ref}}||}_{\text{How do we measure distances ?}}$$

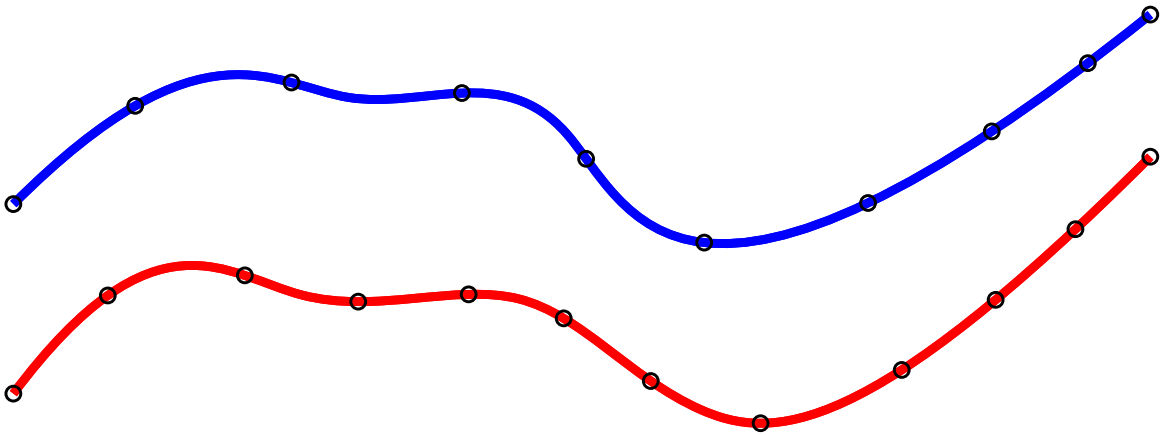
Measuring distances between paths

Key idea–borrow tools from Computational Geometry



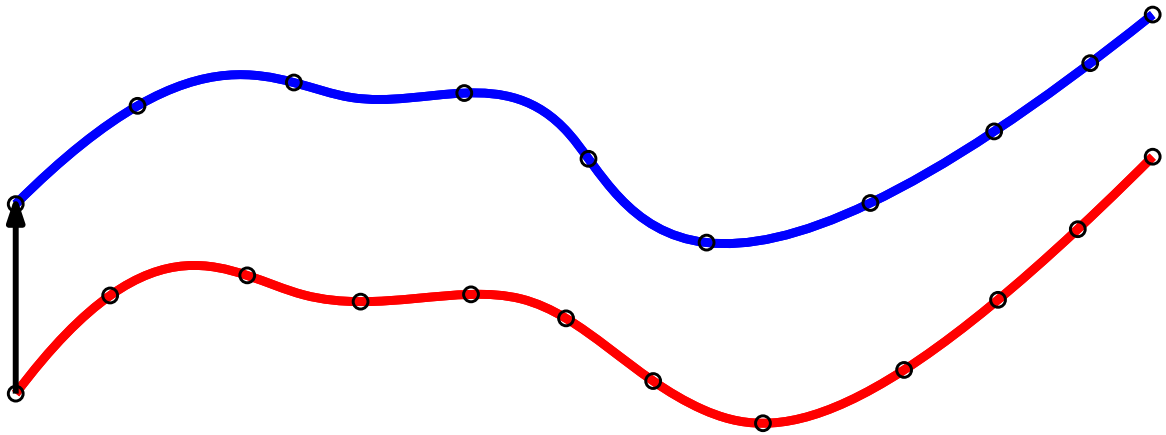
Measuring distances between paths

Key idea–borrow tools from Computational Geometry



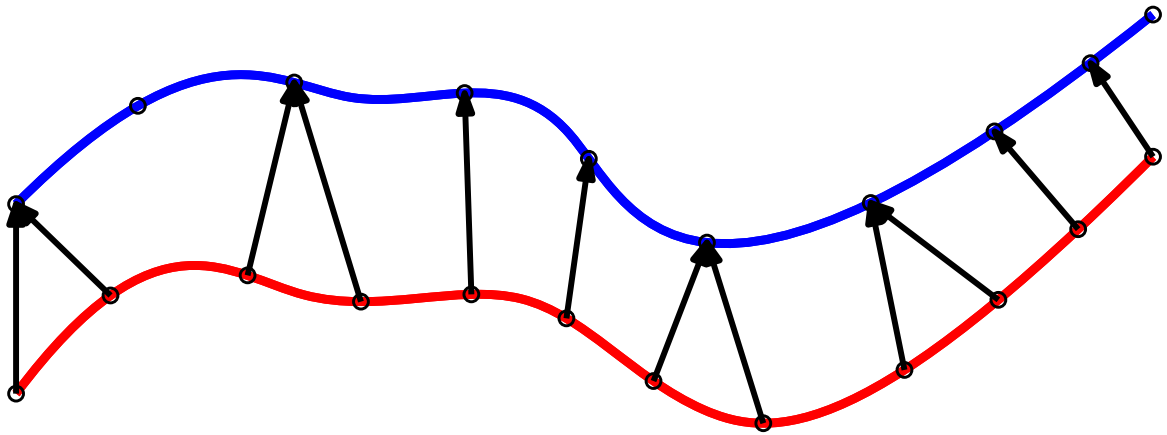
Measuring distances between paths

Key idea–borrow tools from Computational Geometry



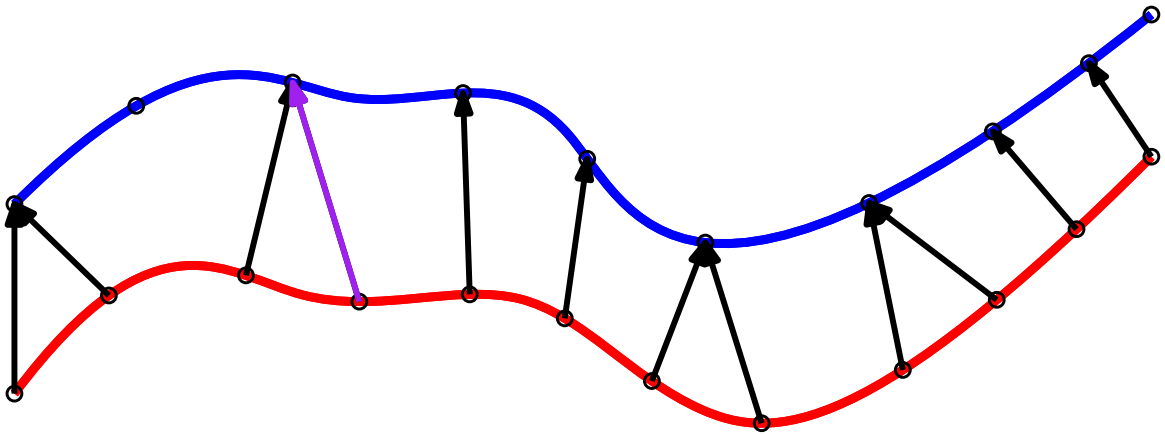
Measuring distances between paths

Key idea–borrow tools from Computational Geometry



Measuring distances between paths

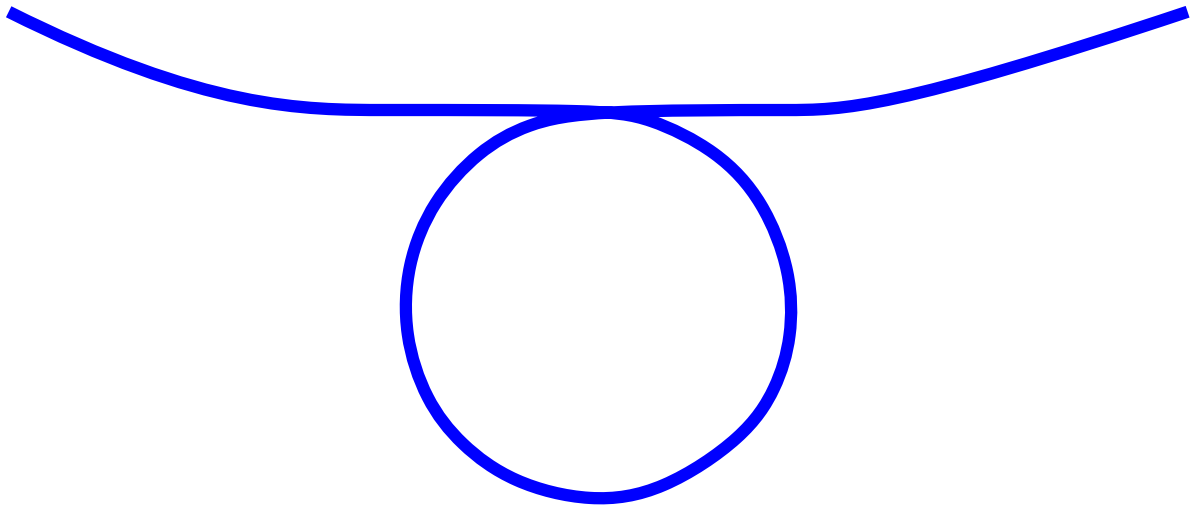
Key idea–borrow tools from Computational Geometry



One-way Hausdorff distance

Measuring distances between paths

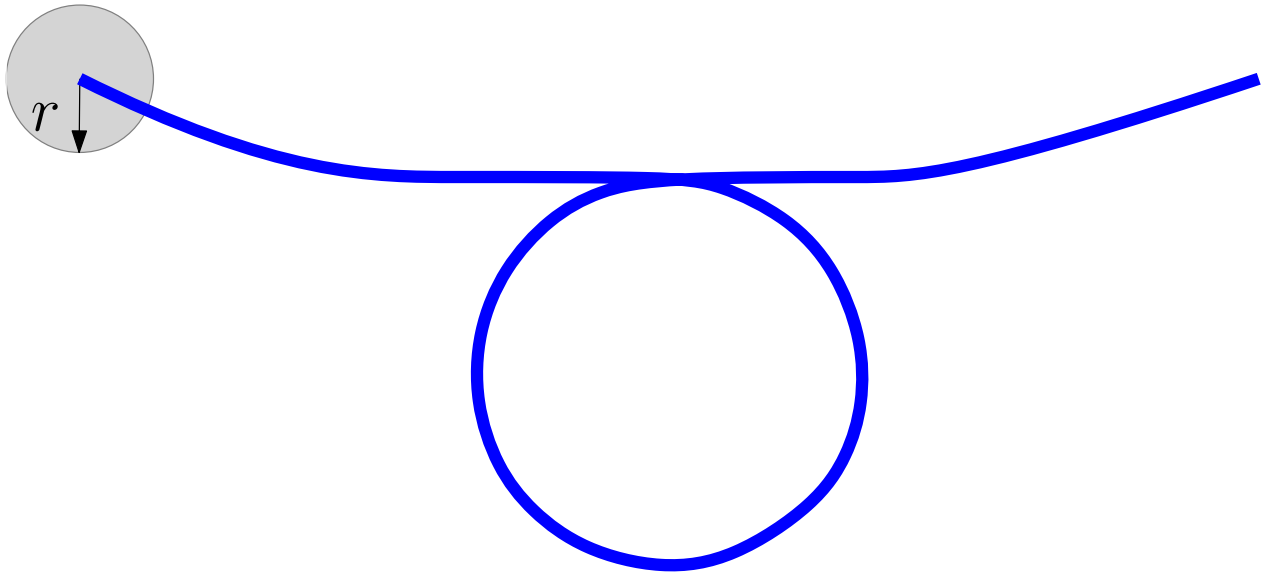
Key idea–borrow tools from Computational Geometry



One-way Hausdorff distance

Measuring distances between paths

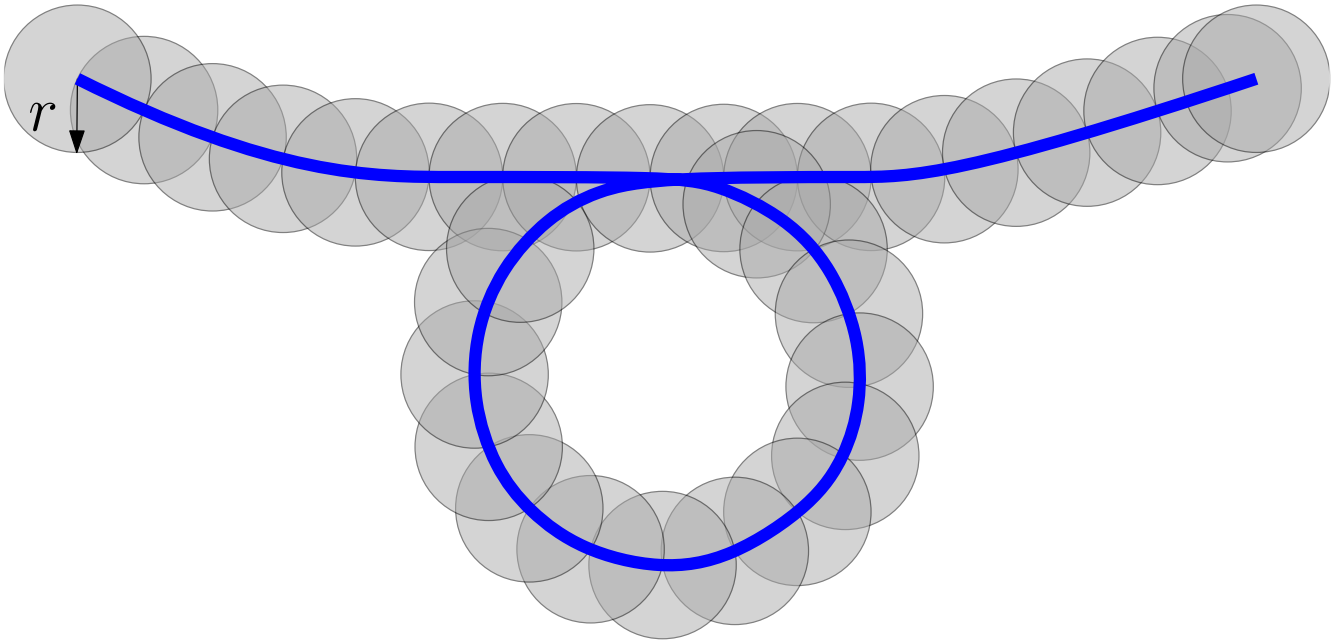
Key idea–borrow tools from Computational Geometry



One-way Hausdorff distance

Measuring distances between paths

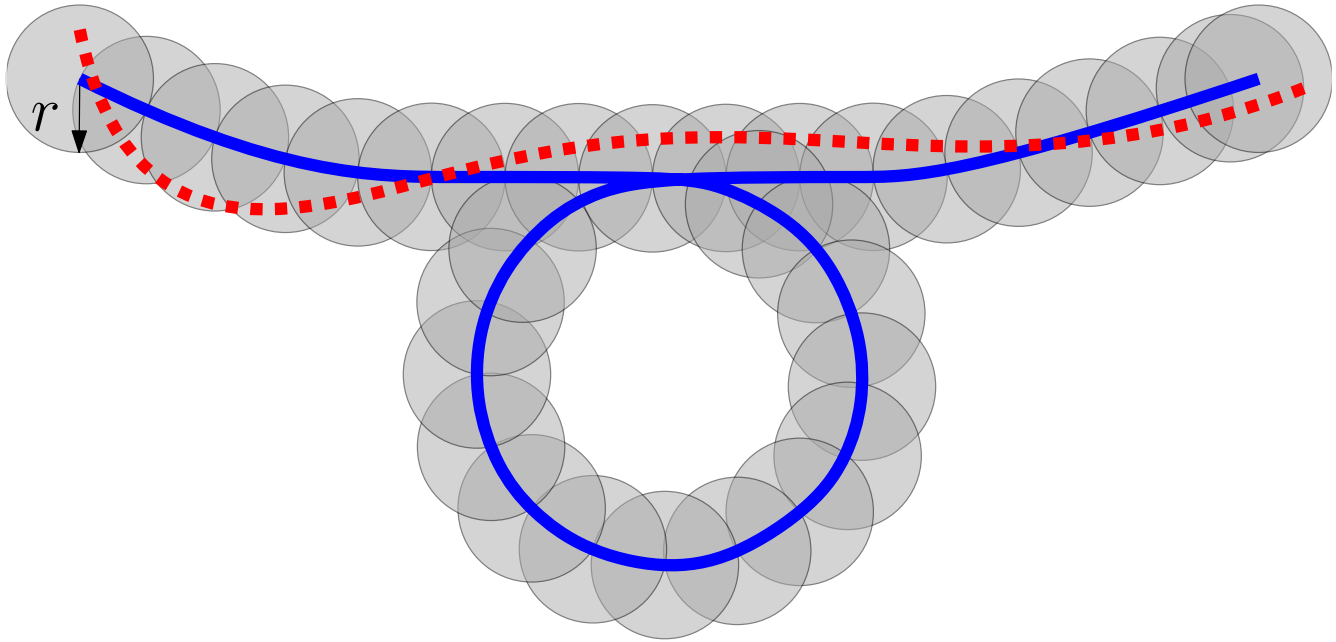
Key idea–borrow tools from Computational Geometry



One-way Hausdorff distance

Measuring distances between paths

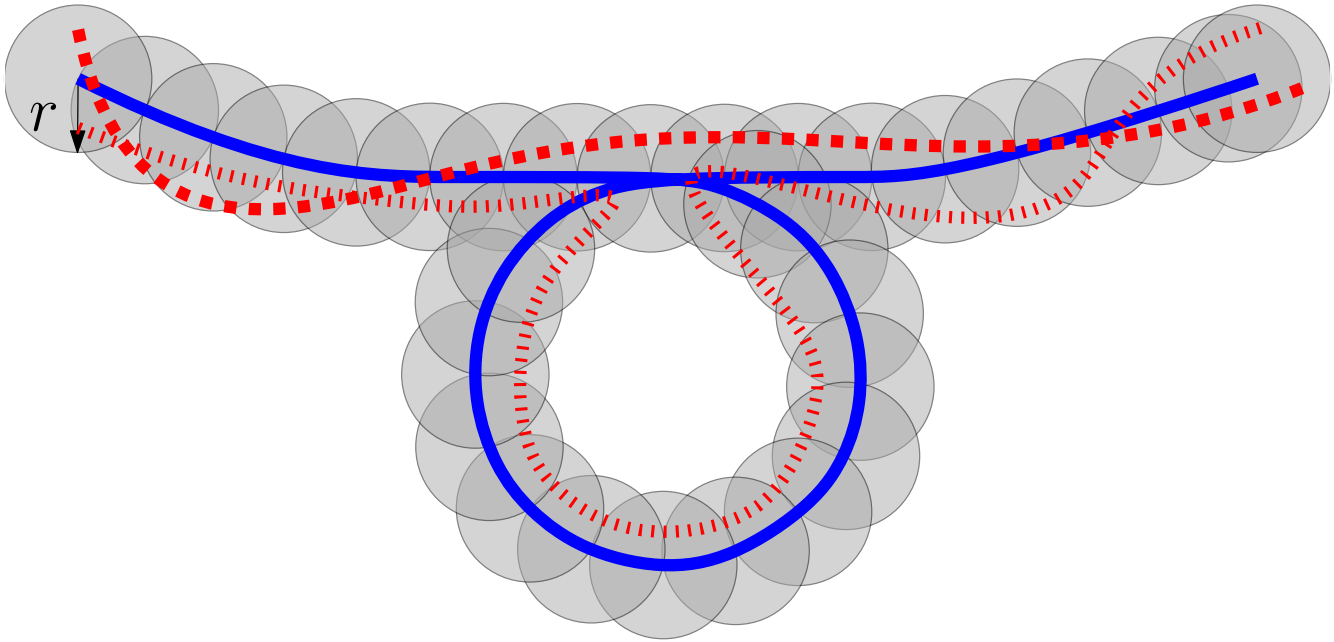
Key idea–borrow tools from Computational Geometry



One-way Hausdorff distance

Measuring distances between paths

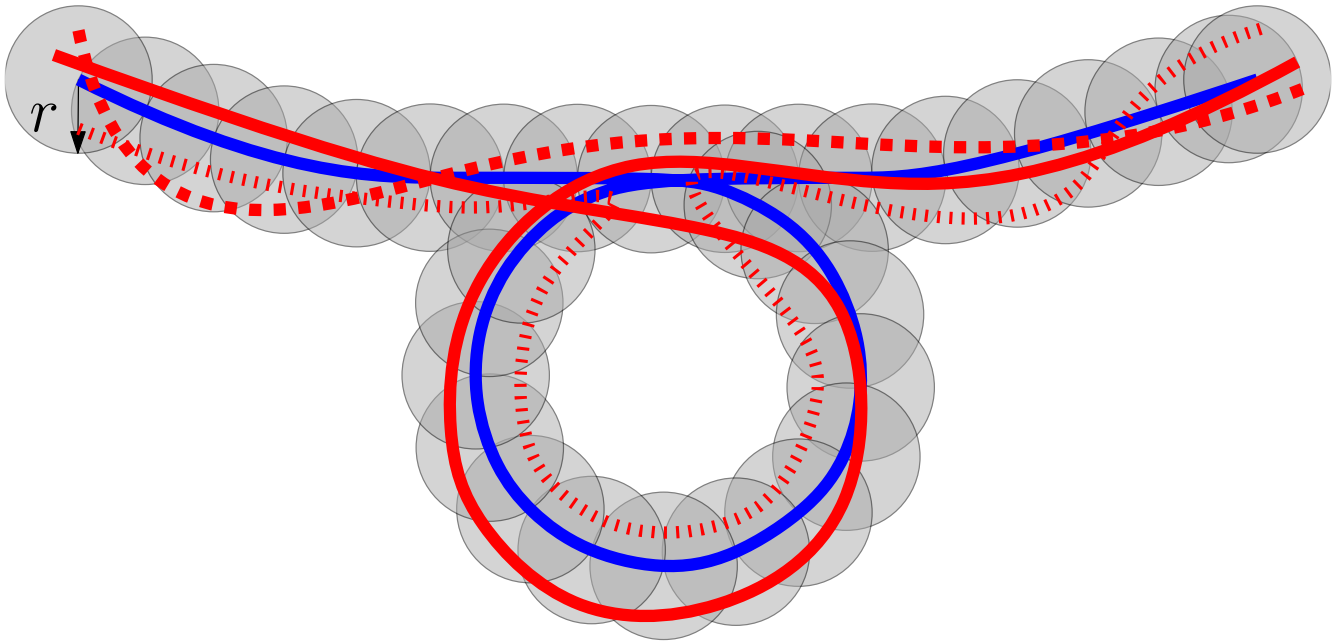
Key idea–borrow tools from Computational Geometry



Two-way Hausdorff distance

Measuring distances between paths

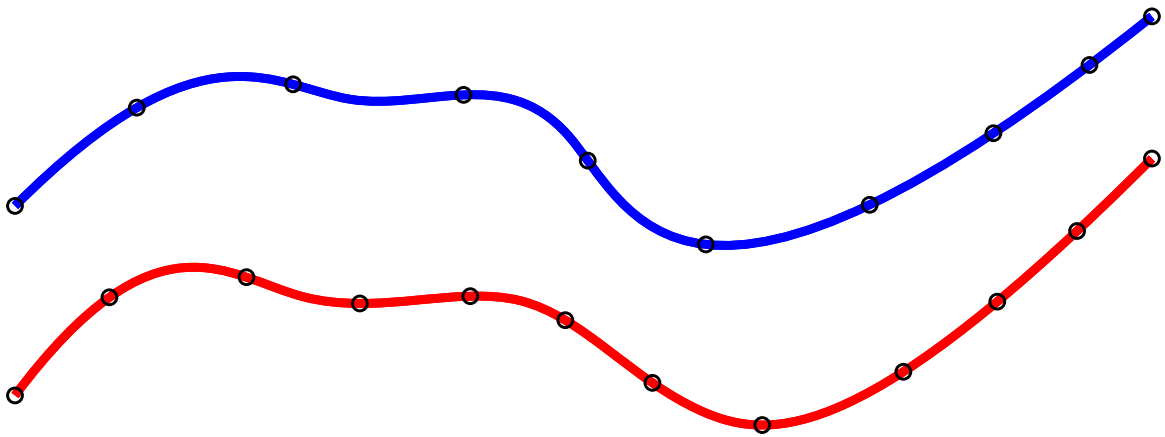
Key idea–borrow tools from Computational Geometry



We want to follow the balls in order—Frechet distance

Measuring distances between paths

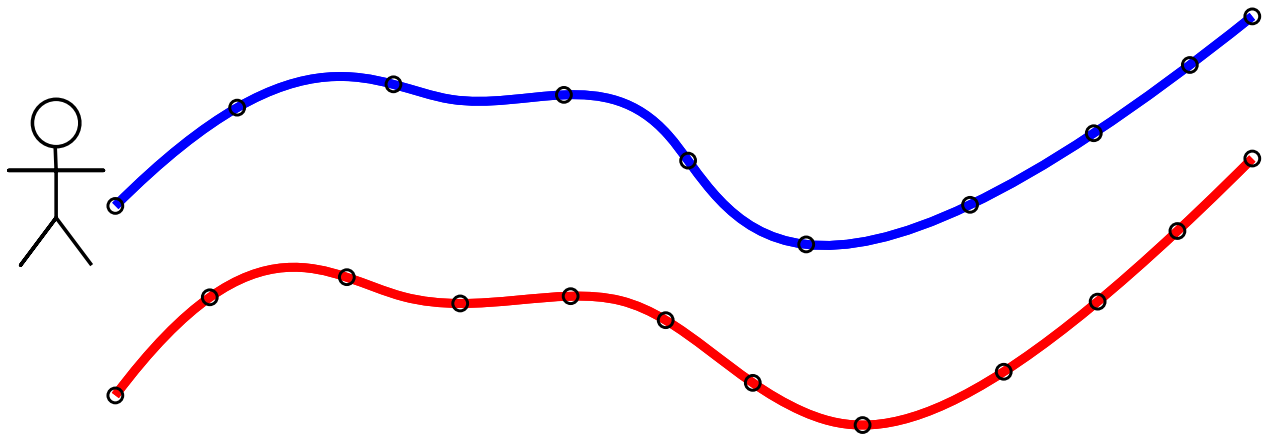
Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

Measuring distances between paths

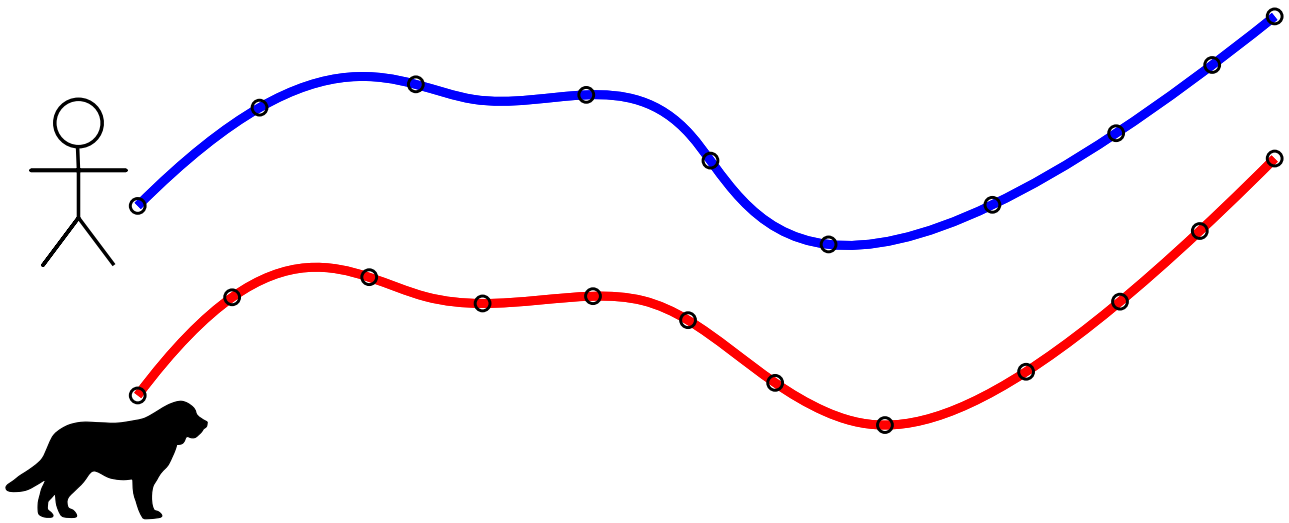
Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

Measuring distances between paths

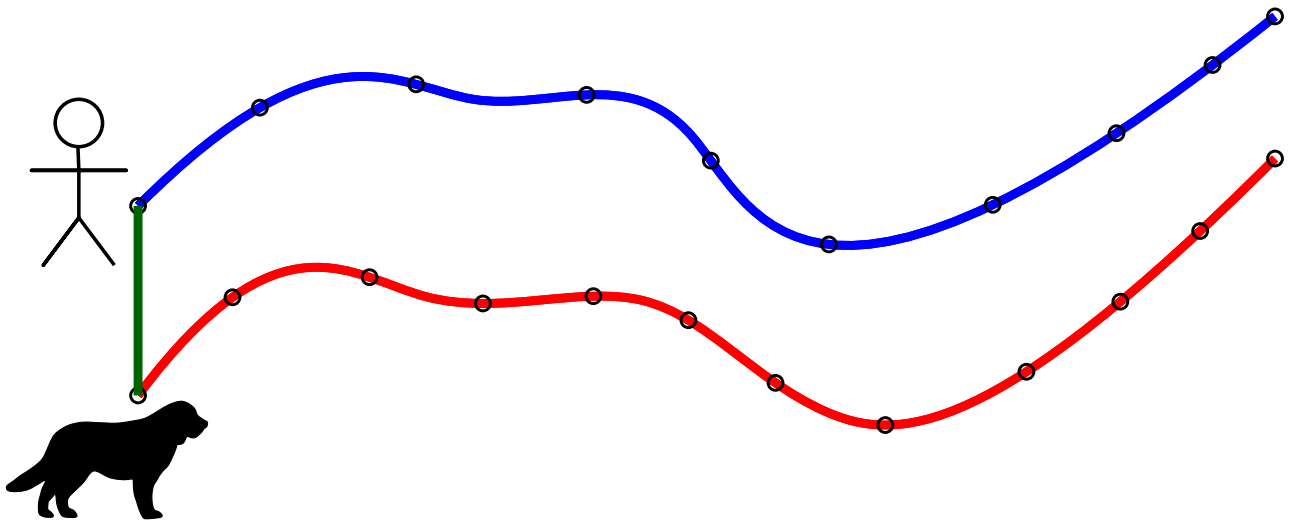
Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

Measuring distances between paths

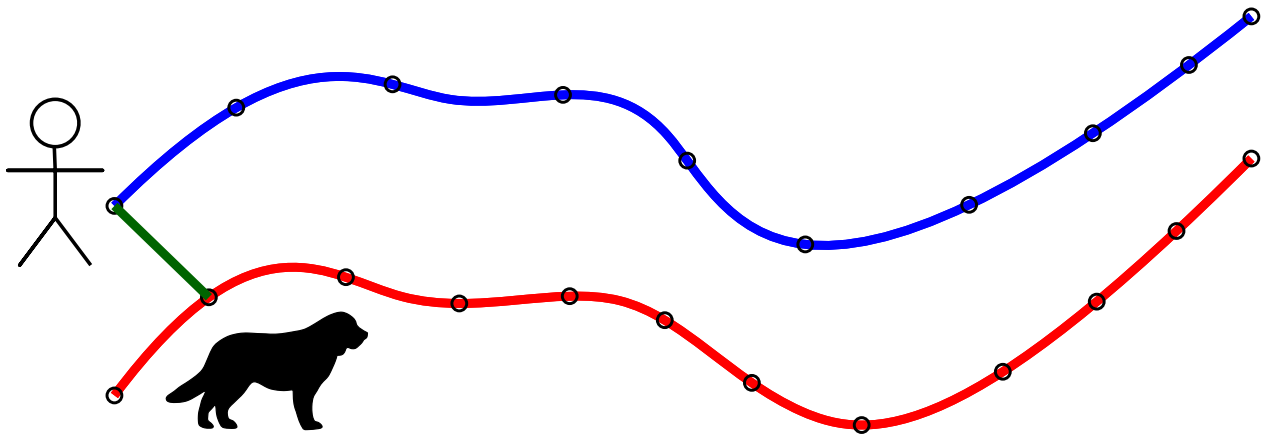
Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

Measuring distances between paths

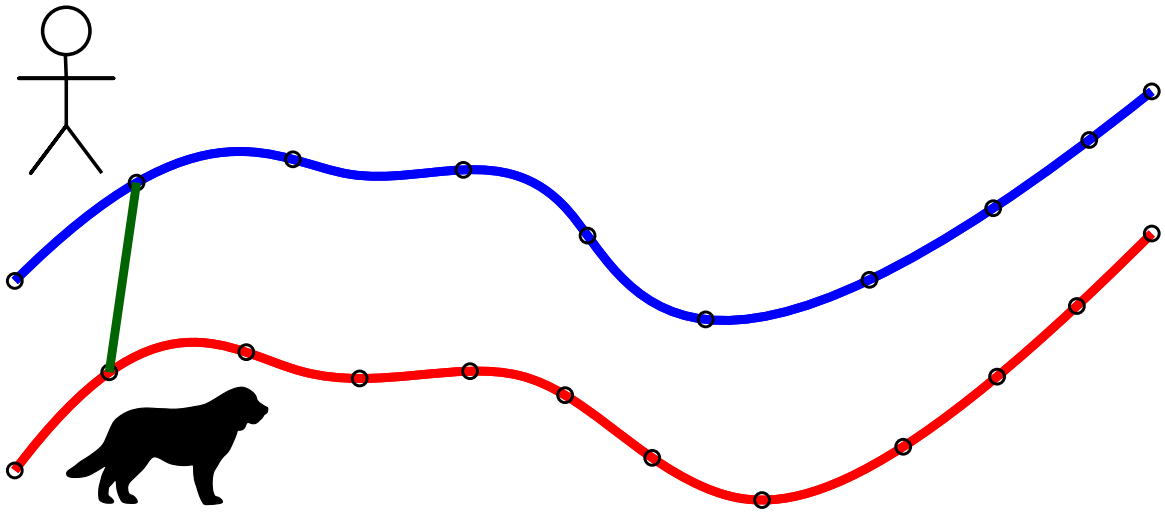
Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

Measuring distances between paths

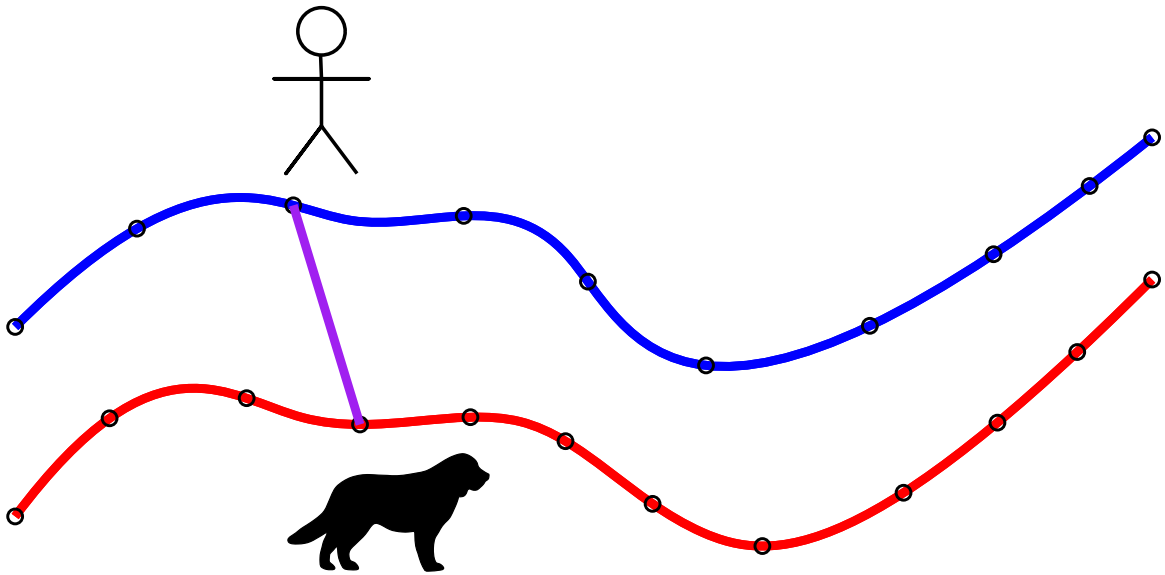
Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

Measuring distances between paths

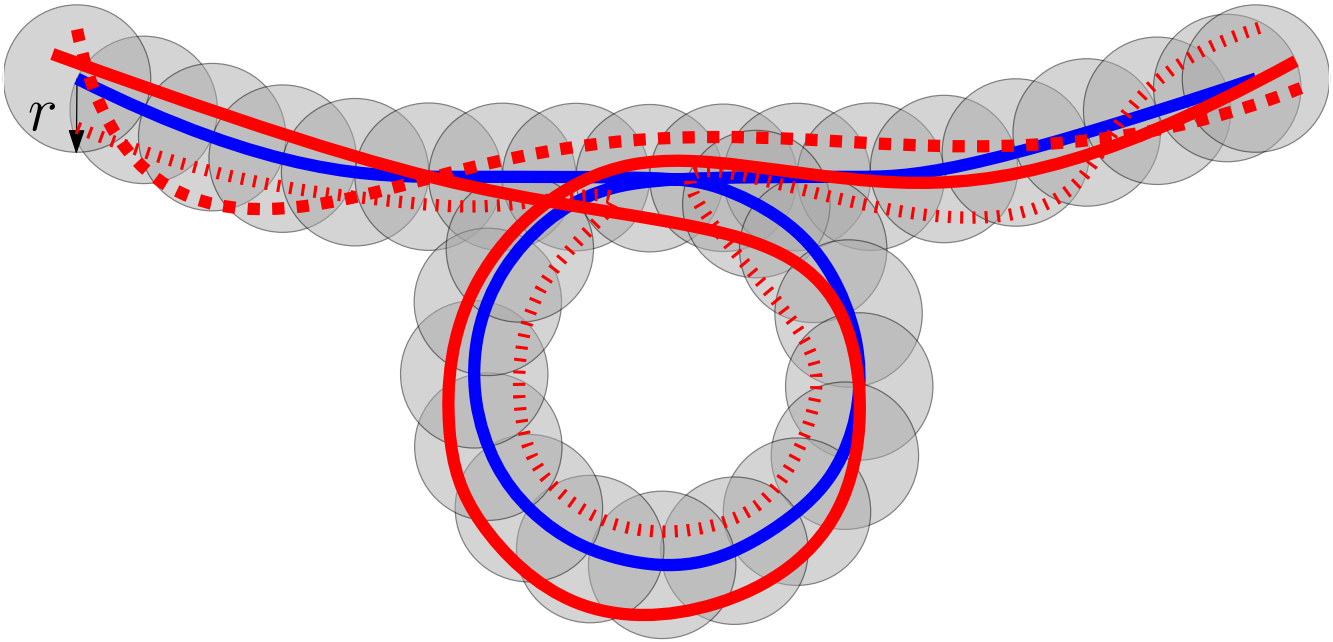
Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

Measuring distances between paths

Key idea–borrow tools from Computational Geometry



(discrete) Frechet distance—Similarity measure that accounts for location and ordering

(discrete) Frechet distance—computation

- Let $P := \{p_1, \ldots p_n\}$ and $Q := \{q_1, \ldots q_n\}$ be sequences of points
- $f(x, y)$—discrete Frechet distance between $\{p_1, \ldots p_x\}$ and $\{q_1, \ldots q_y\}$
- The last move in the optimal path was
  - $(x-1, y) \to (x, y)$ or
  - $(x, y-1) \to (x, y)$
- $\Rightarrow f(x, y) = \max(|p_x - q_y|, \min(f(x-1, y), f(x, y-1)))$
- Using dynamic program, $f(n, m)$ can be computed in $O(nm)$
- Slightly better algorithms exist [Agarwal et al. 14]

(discrete) Frechet distance—computation

- Let $P := \{p_1, \ldots p_n\}$ and $Q := \{q_1, \ldots q_n\}$ be sequences of points
- $f(x, y)$—discrete Frechet distance between $\{p_1, \ldots p_x\}$ and $\{q_1, \ldots q_y\}$
- The last move in the optimal path was
  - $(x-1, y) \to (x, y)$ or
  - $(x, y-1) \to (x, y)$
- $\Rightarrow f(x, y) = \max(|p_x - q_y|, \min(f(x-1, y), f(x, y-1)))$
- Using dynamic program, $f(n, m)$ can be computed in $O(nm)$
- Slightly better algorithms exist [Agarwal et al. 14]

(discrete) Frechet distance—computation

- Let $P := \{p_1, \ldots p_n\}$ and $Q := \{q_1, \ldots q_n\}$ be sequences of points
- $f(x, y)$—discrete Frechet distance between $\{p_1, \ldots p_x\}$ and $\{q_1, \ldots q_y\}$
- The last move in the optimal path was
  - $(x-1, y) \to (x, y)$ or
  - $(x, y-1) \to (x, y)$
- $\Rightarrow f(x, y) = \max\left(|p_x - q_y|, \min\left(f(x-1, y), f(x, y-1)\right)\right)$
- Using dynamic program, $f(n, m)$ can be computed in $O(nm)$
- Slightly better algorithms exist [Agarwal et al. 14]

Following reference paths—Aproach

$$\zeta^* = \underbrace{\arg\min_{\zeta \in \mathcal{C}_{\text{free}}}}_{\text{How do we plan ?}} \underbrace{||\text{FK}(\zeta), \zeta_{\text{ref}}||}_{\text{How do we measure distances ?}}$$

Following reference paths—Aproach

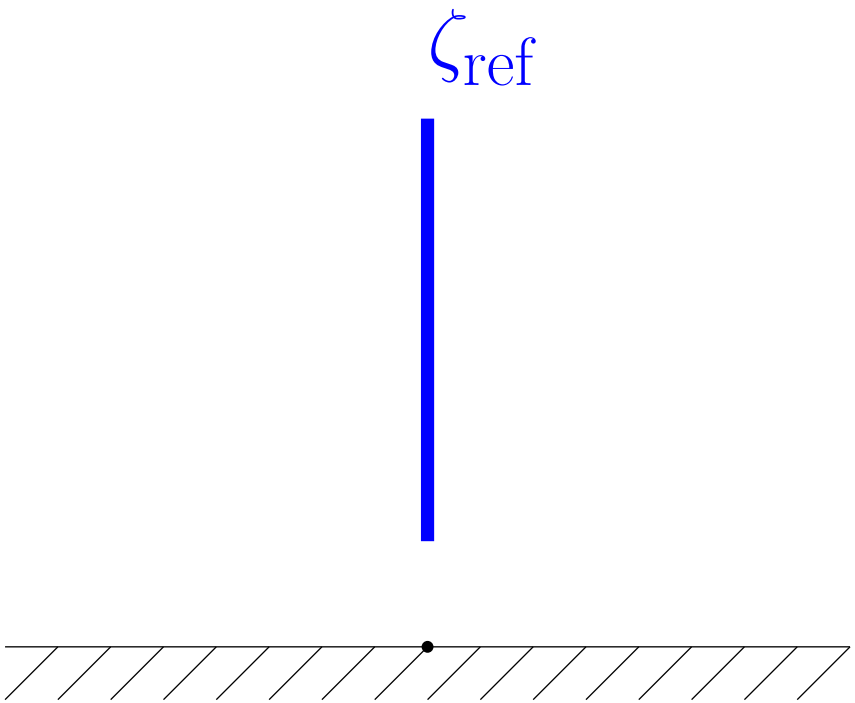
$$\zeta^* = \underbrace{\arg\min_{\zeta \in \mathcal{C}_{\text{free}}}}_{\text{How do we plan ?}} \underbrace{||\text{FK}(\zeta), \zeta_{\text{ref}}||}_{\text{How do we measure distances ?}}$$

dFrećhet distance

Following reference paths—Take (1)

Optimization-based approach [Holladay, Srinivassa 16]

- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples



Following reference paths—Take (1)

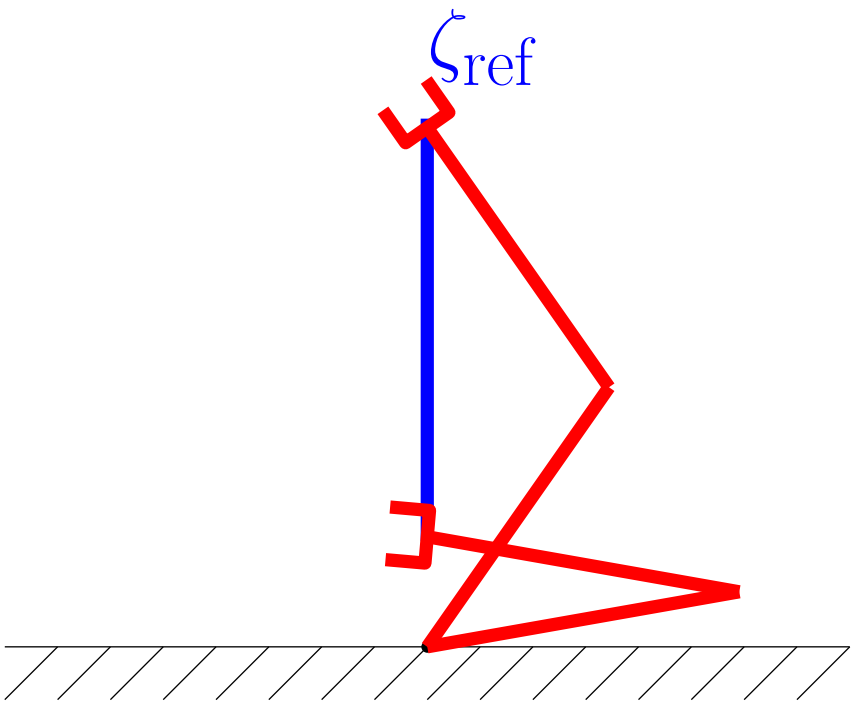
Optimization-based approach [Holladay, Srinivassa 16]

- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples

Following reference paths—Take (1)

Optimization-based approach [Holladay, Srinivassa 16]

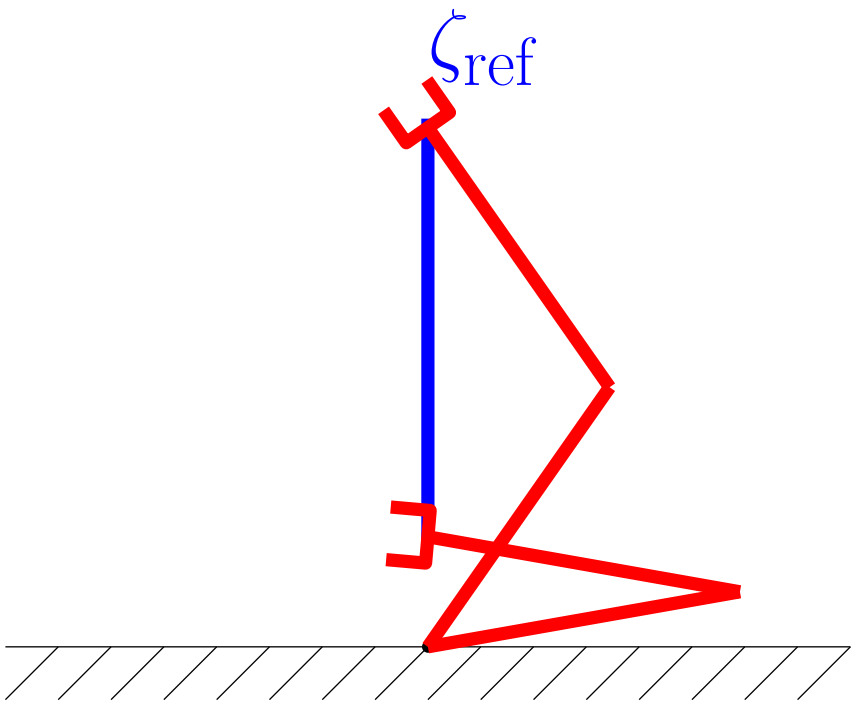
- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples



Following reference paths—Take (1)

Optimization-based approach [Holladay, Srinivassa 16]

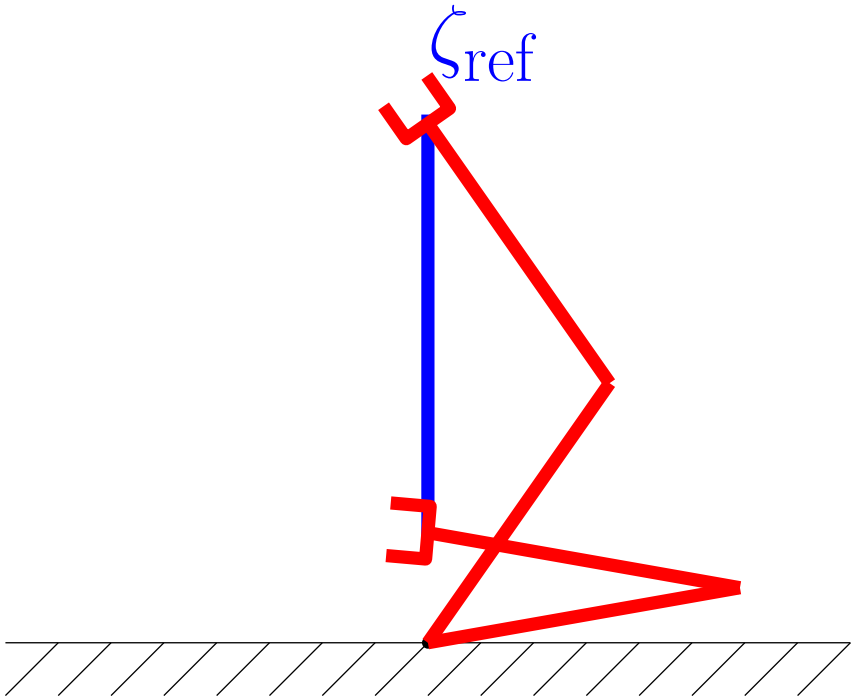
- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples

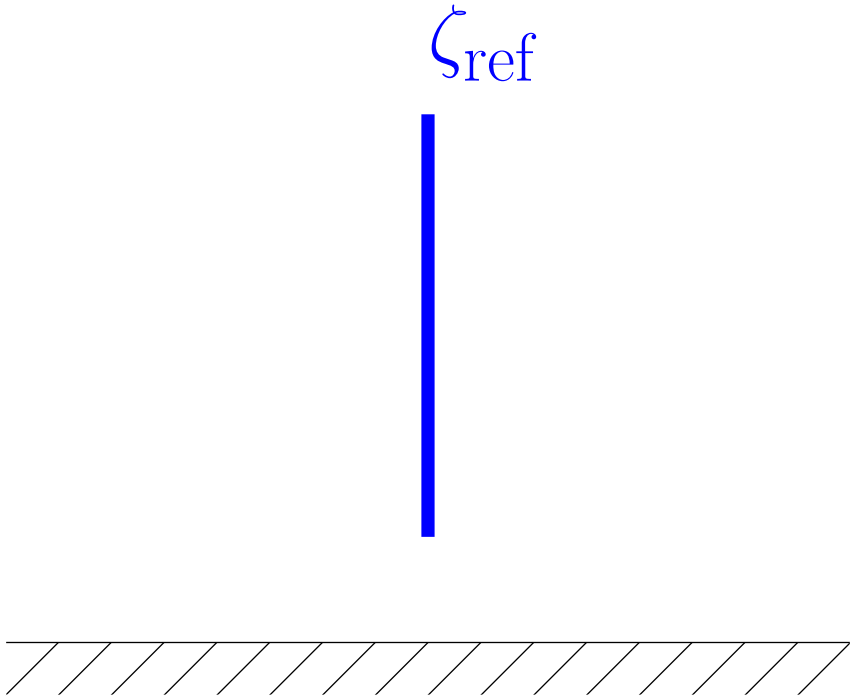


Following reference paths—Take (1)

Optimization-based approach [Holladay, Srinivassa 16]

- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples

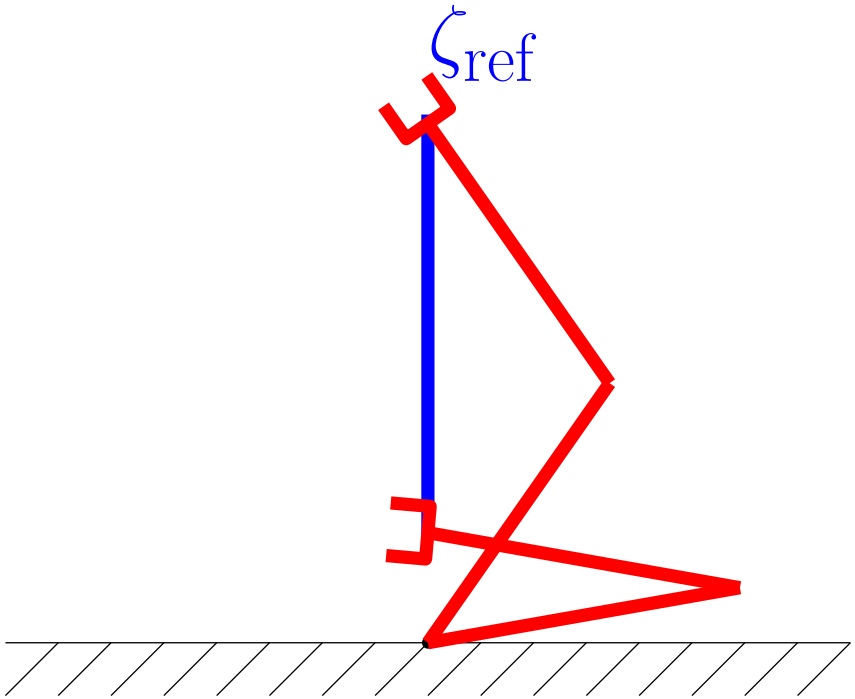


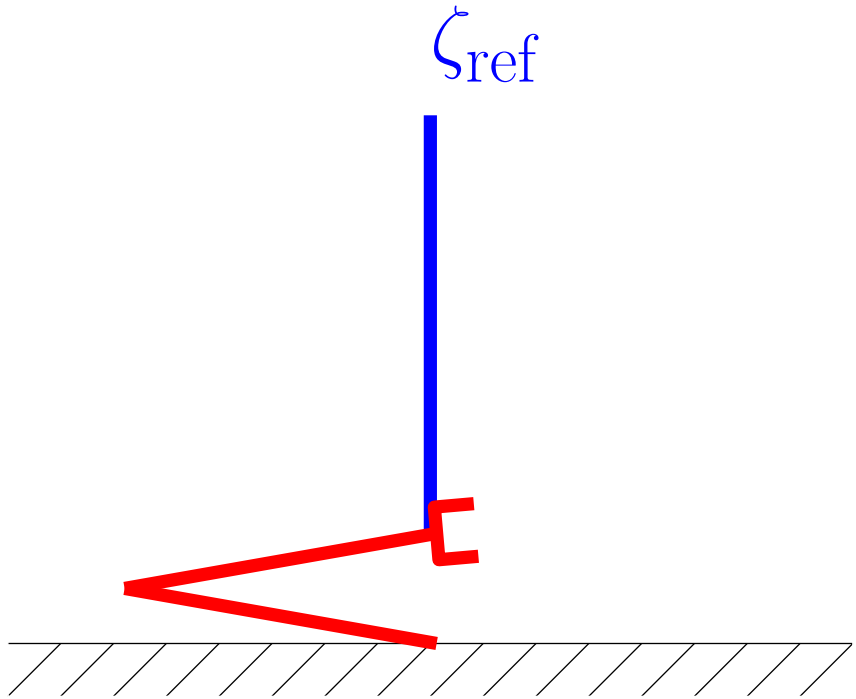


Following reference paths—Take (1)

Optimization-based approach [Holladay, Srinivassa 16]

- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples

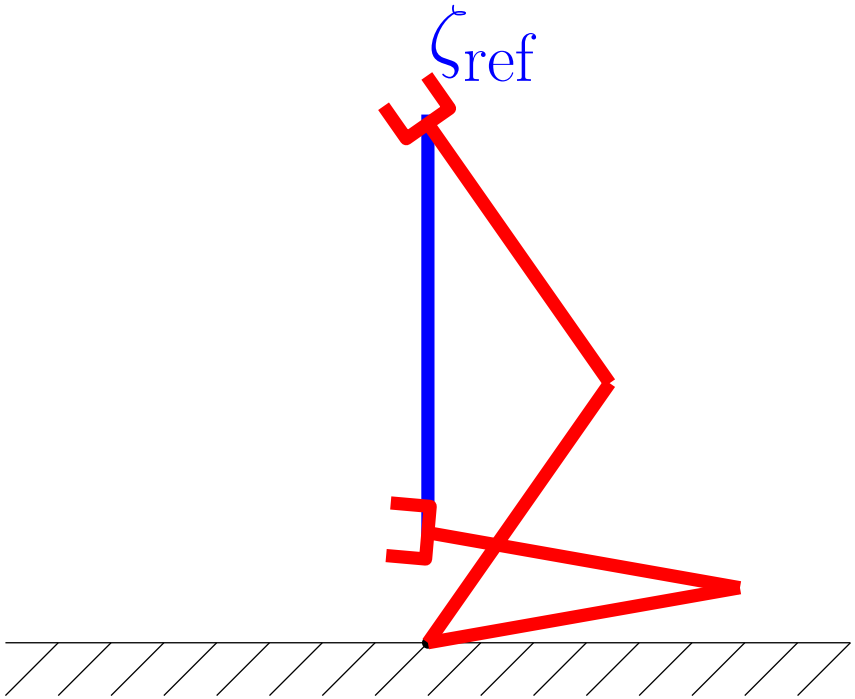


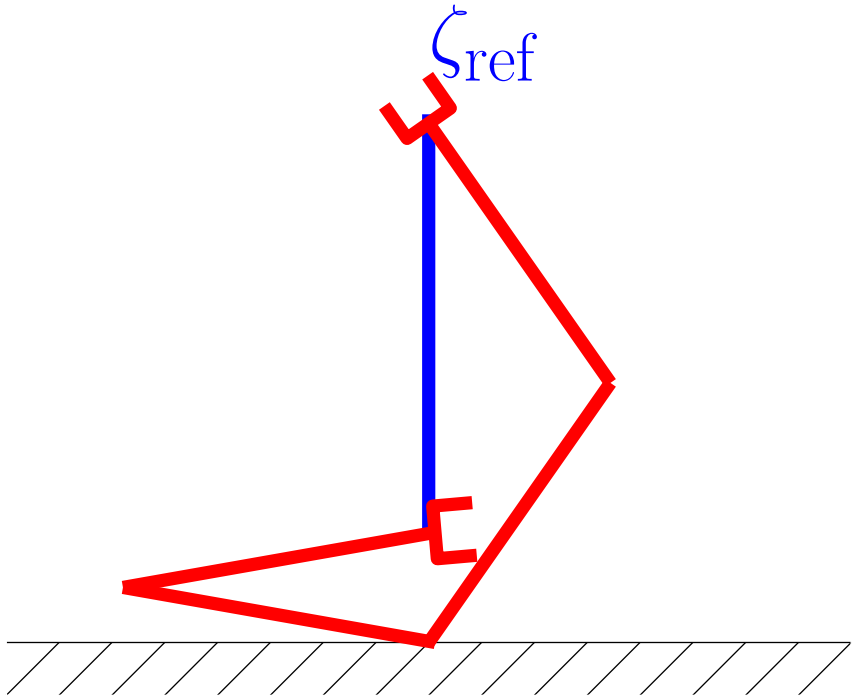


Following reference paths—Take (1)

Optimization-based approach [Holladay, Srinivassa 16]

- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples

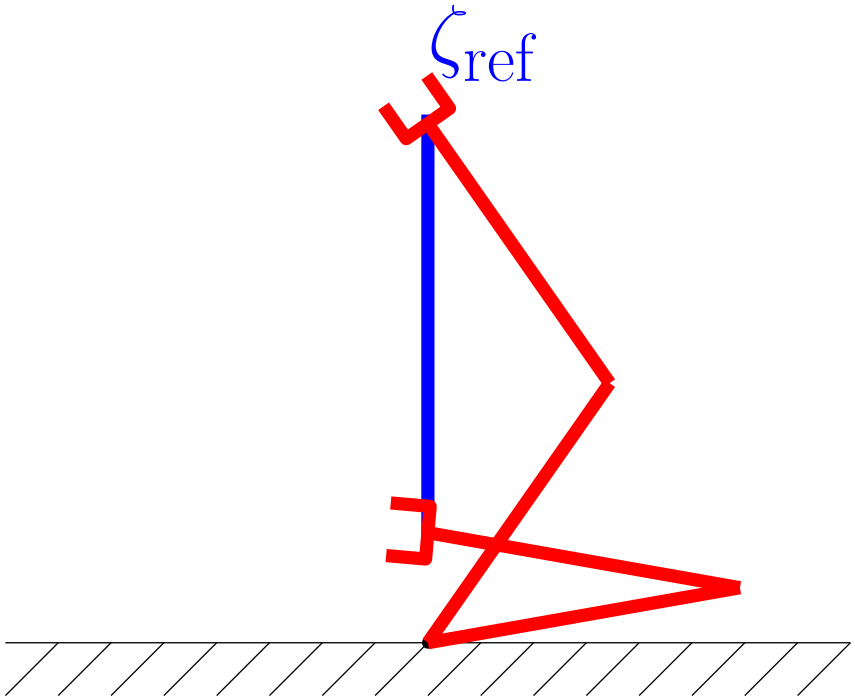


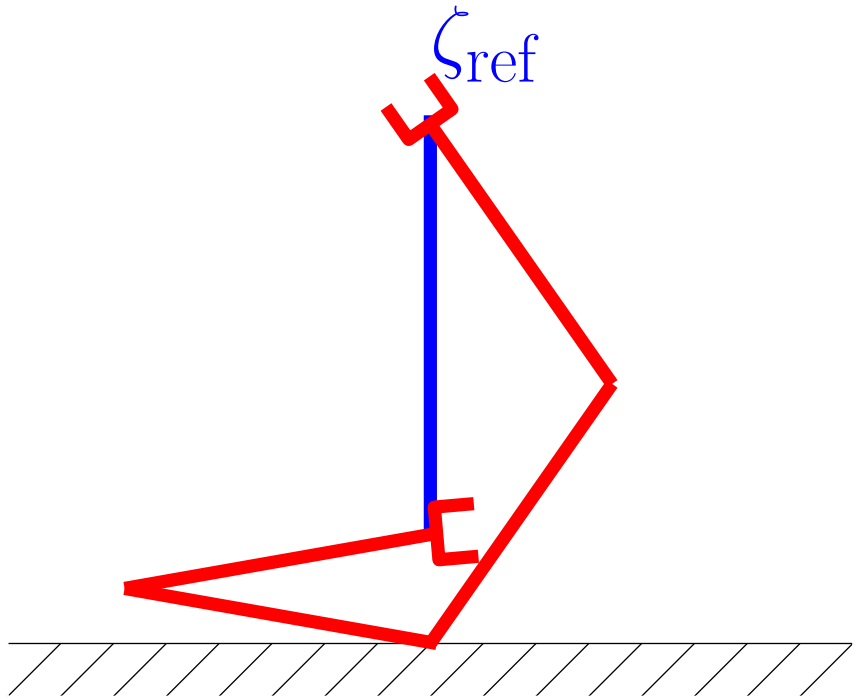


Following reference paths—Take (1)

Optimization-based approach [Holladay, Srinivassa 16]

- Sample along ζ_{ref}
- Compute IK of each sample
- Connect using local planner
- Refine samples

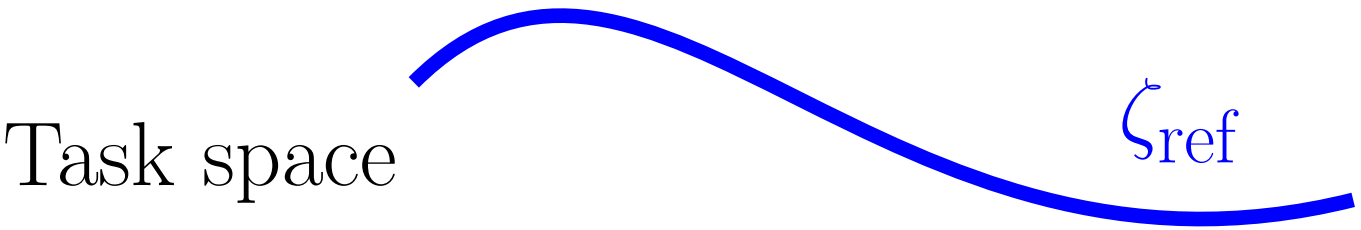




Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

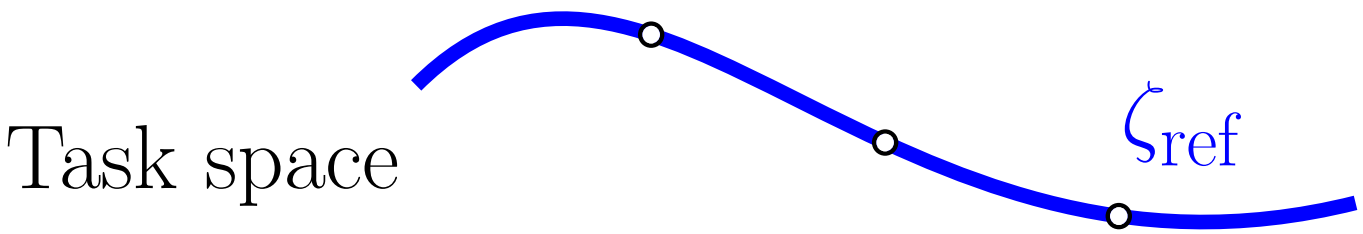
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

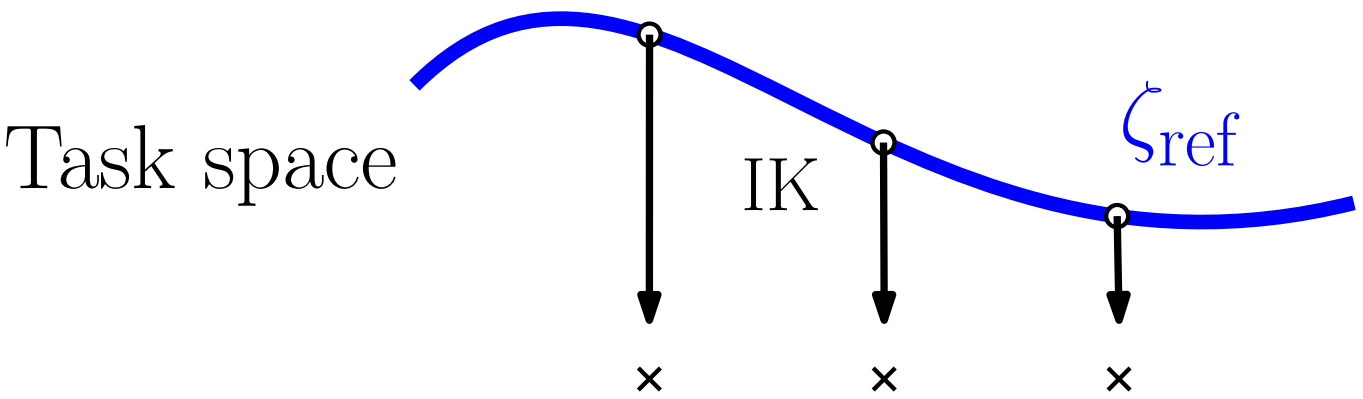
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

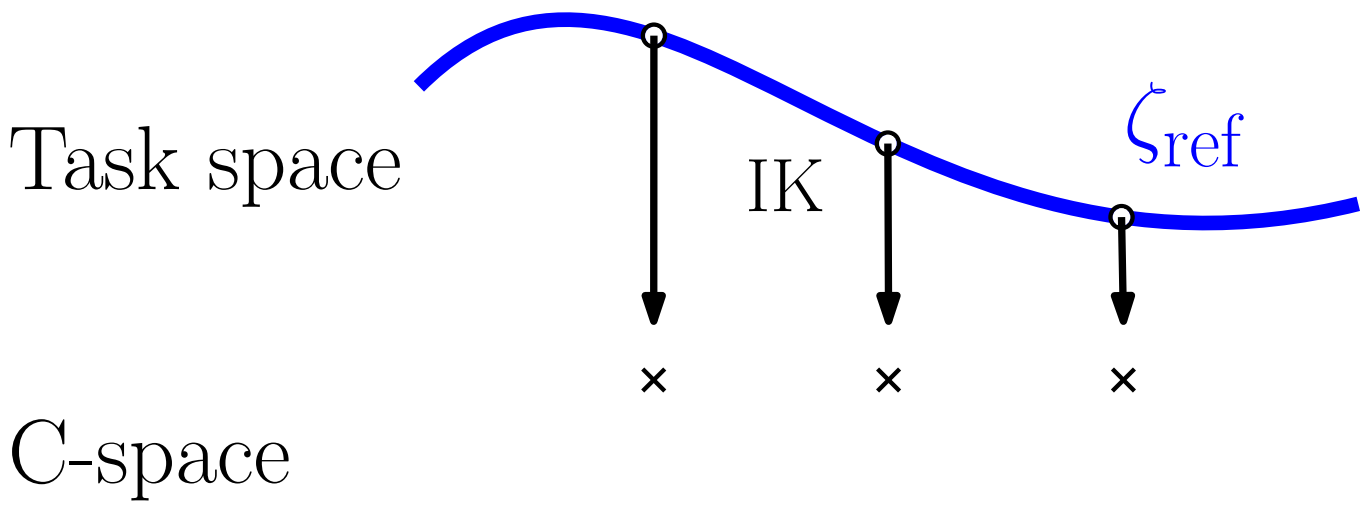
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

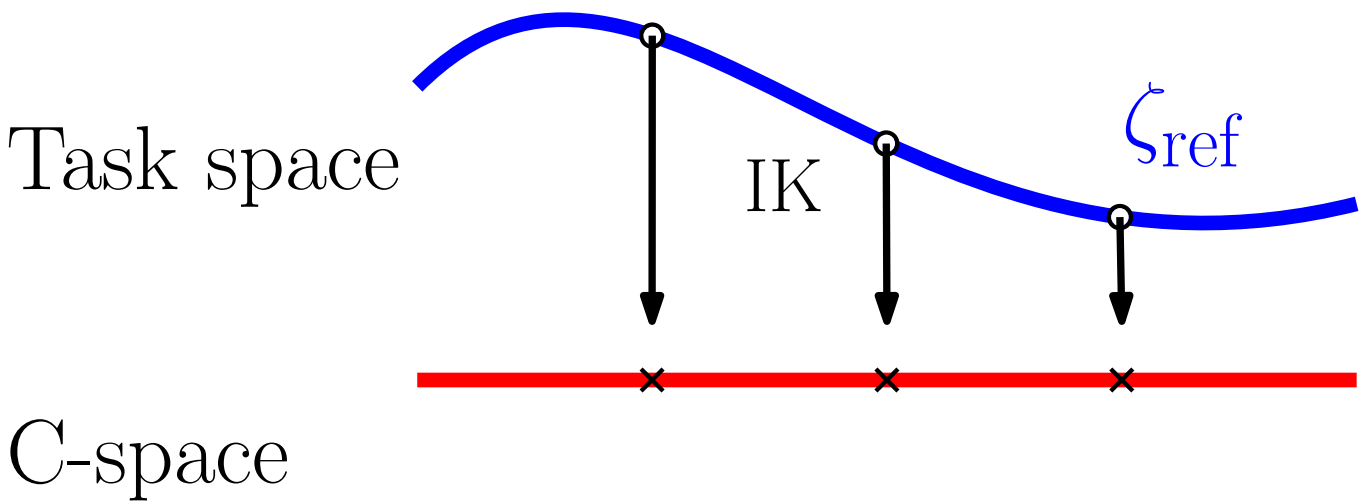
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

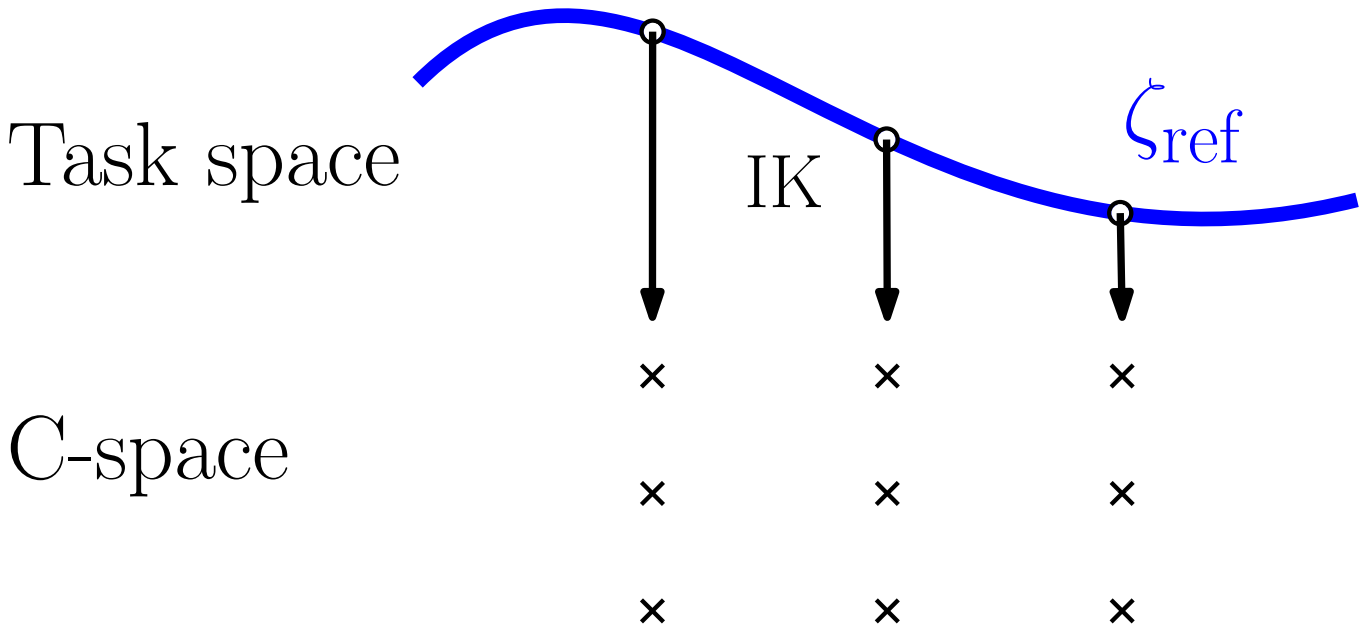
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

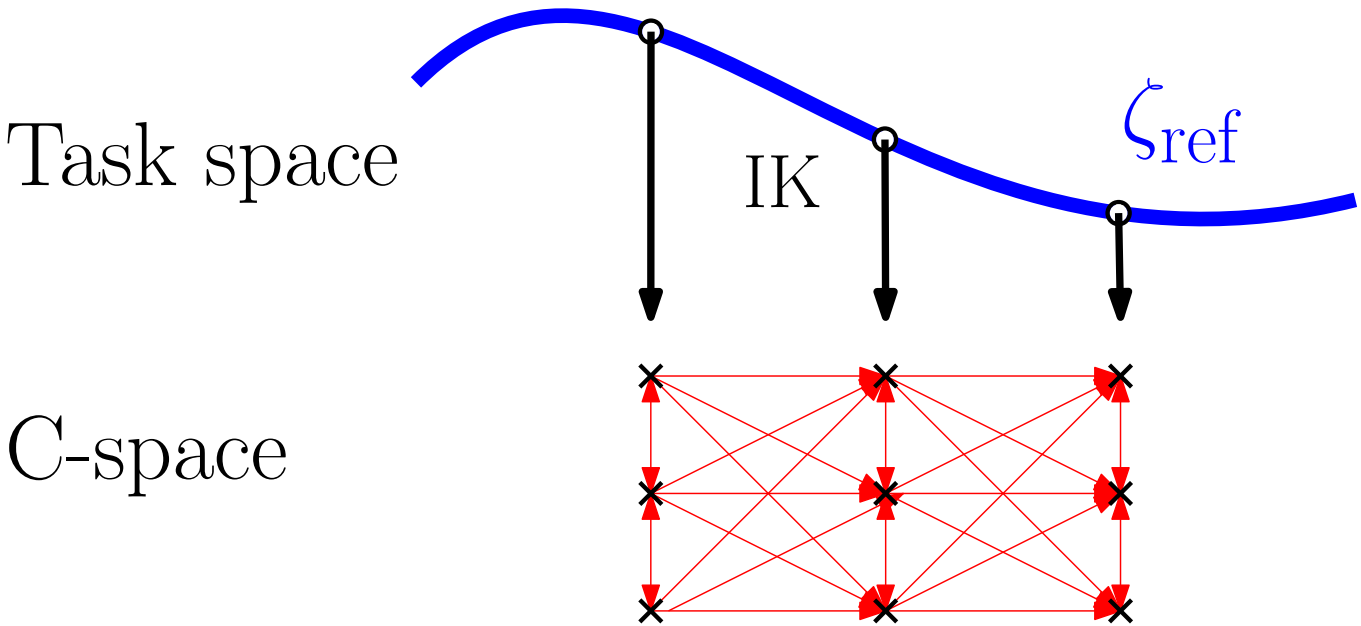
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

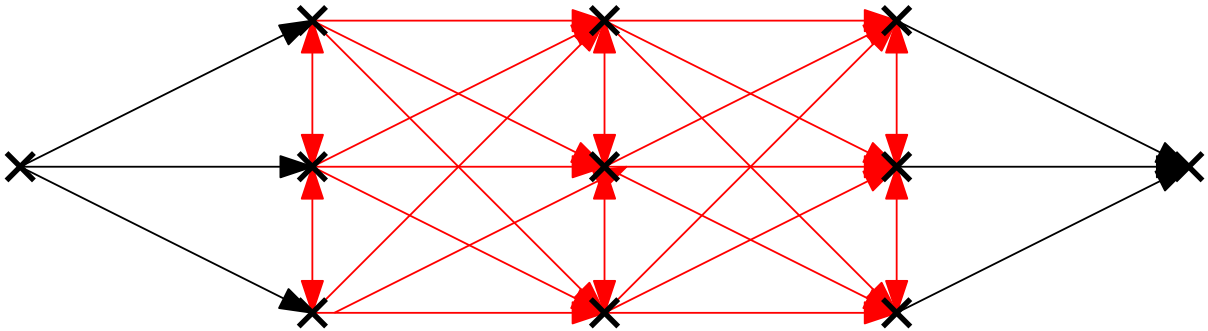
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

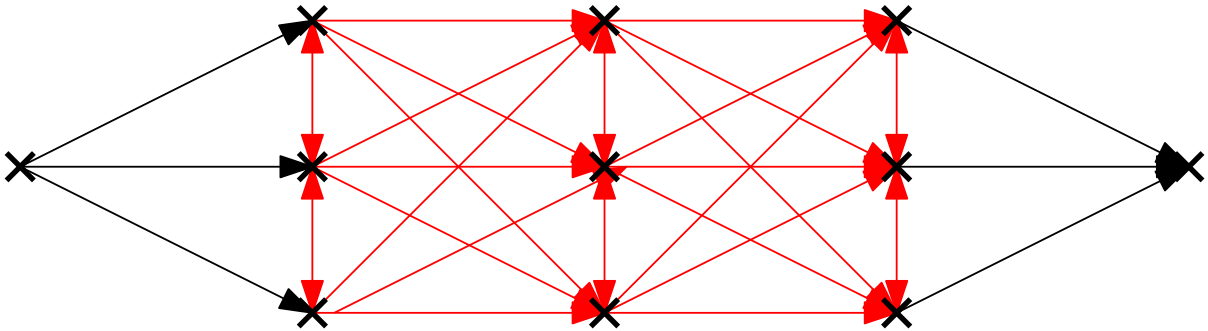
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



Following reference paths—Take (2)

Planning [Holladay, S., Srinivasa19]

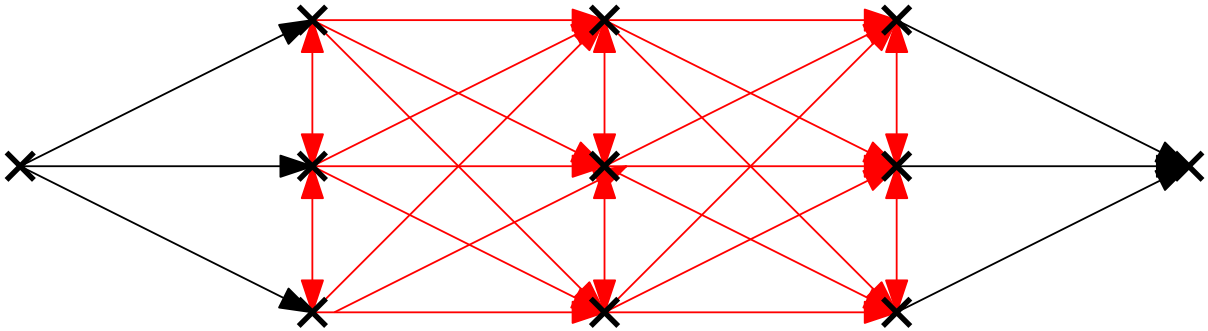
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L



$$\zeta^* = \arg\min_{p \in L} \text{dFrećhet}(\text{FK}(p), \zeta_{\text{ref}})$$

Domain-Specific challenges—following reference paths

How do we efficiently find minimal-cost path in L?



$$\zeta^* = \arg\min_{p \in L} \mathrm{dFrećhet}(\mathrm{FK}(p), \zeta_{\mathrm{ref}})$$

- Number of paths in L is $O(n^k)$
  - n is the number of layers
  - k is the maximal number of IK solutions
- By adapting existing machinery [Har-Peled, Raichel14] we can find ζ^* in $O(n^2k^2)$

Efficiently computing minimal-cost path in L

- We define the cross-product graph $\Phi = (V_\Phi, E_\Phi)$ of $L = (V_L, E_L)$ and $\zeta_{\text{ref}} = (V_{\zeta_{\text{ref}}}, E_{\zeta_{\text{ref}}})$

$$
\begin{aligned}
V_\Phi &= V_{\zeta_{\text{ref}}} \times V_L. \\
E_\Phi &= \{((w_1, q_1), (w_2, q_2)) \mid \text{ if} \\
&\quad ((w_1 = w_2) \text{ and } (q_1, q_2) \in E_L) \text{ or} \\
&\quad ((w_1, w_2) \in E_{\zeta_{\text{ref}}} \text{ and } (q_1 = q_2)) \text{ or} \\
&\quad ((w_1, w_2) \in E_{\zeta_{\text{ref}}} \text{ and } (q_1, q_2) \in E_L)\}. \\
c((w_1, q_1)) &= ||(w_1, \text{FK}(q_1))||. \\
c(u, v) &= \max\{c(u), c(v)\}.
\end{aligned}
$$

Definition

The bottleneck cost of a path is the cost of the edge with the maximal cost along the path

Efficiently computing minimal-cost path in L

- We define the cross-product graph $\Phi = (V_\Phi, E_\Phi)$ of $L = (V_L, E_L)$ and $\zeta_{\text{ref}} = (V_{\zeta_{\text{ref}}}, E_{\zeta_{\text{ref}}})$

$$
\begin{aligned}
V_\Phi &= V_{\zeta_{\text{ref}}} \times V_L. \\
E_\Phi &= \{((w_1, q_1), (w_2, q_2)) \mid \text{ if} \\
&\quad ((w_1 = w_2) \text{ and } (q_1, q_2) \in E_L) \text{ or} \\
&\quad ((w_1, w_2) \in E_{\zeta_{\text{ref}}} \text{ and } (q_1 = q_2)) \text{ or} \\
&\quad ((w_1, w_2) \in E_{\zeta_{\text{ref}}} \text{ and } (q_1, q_2) \in E_L)\}. \\
c((w_1, q_1)) &= ||(w_1, \text{FK}(q_1))||. \\
c(u, v) &= \max\{c(u), c(v)\}.
\end{aligned}
$$

Thm (adapted from [Raichel Har-Peled14])

Let $P = \{(w_1, q_1), \ldots, (w_k, q_k)\}$ be the bottleneck shortest path in Φ. Then the FK of $P_q = \{q_1, \ldots, q_k, \}$ minimizes the Frechet distance to $\{w_1, \ldots, w_k\}$ (discretization of ζ_{ref})

Computing the bottleneck shortest path

We can compute the bottleneck shortest path by a simple adaptation of Dijkstra's algorithm

Input:

- Graph $G = (V, E)$
- Start & target vertices s, t
- Cost function $c : V \to \mathbb{R}$

Output: Cost b^* of bottleneck shortest path connecting s and t

---

1: Open $\leftarrow s$ with b(s) = 0; ▷ Initialization
2: **while** Open is not empty **do**
3: $u \leftarrow$ Open.min_key() ▷ u has minimal $b(u)$ in Open
4: **if** $u = t$ **then**
5: **return** $b(t)$
6: **for** each v such that $(u, v) \in E$ **do**
7: $b' = \max(c(v), b(u))$ ▷ new bottleneck cost
8: **if** $v \in$ Open with $b(v) < b'$ **then continue**
9: **else** insert v to Open with $b(v) = b'$ ▷ updates $b(v)$ if in Open
10: **return**

Computing the bottleneck shortest path

We can compute the bottleneck shortest path by a simple adaptation of Dijkstra's algorithm

Input:

- Graph $G = (V, E)$
- Start & target vertices s, t
- Cost function $c : V \to \mathbb{R}$

Output: Cost b^* of bottleneck shortest path connecting s and t

- What is the complexity of the algorithm?

$$O(|V| \log |V| + |E|)$$

- Can we do better?

Computing the bottleneck shortest path (cont.)

We can actually compute the bottleneck shortest path in $O(|V| + |E|)$ [Har-Peled, Raichel14]

1: remove isolated vertices ▷ $O(|V|)$
2: check that s and t are connected ▷ $O(|E|)$
3: **if** G has constant size **then**
4: **compute** bottleneck shortest path brute force ▷ $O(1)$
5: **loop**
6: $e_{\text{med}} \leftarrow$ median weight edge ▷ $O(|E|)$
7: $E_{\leq \text{med}} \leftarrow \{e \in E | c(e) \leq c(e_{\text{med}})\}$; $G_{\leq \text{med}} \leftarrow (V, E_{\leq \text{med}})$ ▷ $O(|E|)$
8: compute the connected components of $G_{\leq \text{med}}$ ▷ $O(|E|)$
9: **if** s and t are in the same component of $G_{\leq \text{med}}$ **then**
10: recurse on $G_{\leq \text{med}}$ ▷ $b^* \leq c(e_{\text{med}})$
11: **else**
12: contract each connected component of $G_{\leq \text{med}}$ to a vertex
13: recurse on the graph with these vertices and the edges in $E_{>\text{med}} = E \setminus E_{\leq \text{med}}$

- In practice, Dijkstra's adaptation runs often faster than this (theoretically)-optimal variant

Computing the bottleneck shortest path (cont.)

We can actually compute the bottleneck shortest path in $O(|V| + |E|)$ [Har-Peled, Raichel14]

1: remove isolated vertices ▷ $O(|V|)$
2: check that s and t are connected ▷ $O(|E|)$
3: **if** G has constant size **then**
4: **compute** bottleneck shortest path brute force ▷ $O(1)$
5: **loop**
6: $e_{\text{med}} \leftarrow$ median weight edge ▷ $O(|E|)$
7: $E_{\leq \text{med}} \leftarrow \{e \in E | c(e) \leq c(e_{\text{med}})\}$; $G_{\leq \text{med}} \leftarrow (V, E_{\leq \text{med}})$ ▷ $O(|E|)$
8: compute the connected components of $G_{\leq \text{med}}$ ▷ $O(|E|)$
9: **if** s and t are in the same component of $G_{\leq \text{med}}$ **then**
10: recurse on $G_{\leq \text{med}}$ ▷ $b^* \leq c(e_{\text{med}})$
11: **else**
12: contract each connected component of $G_{\leq \text{med}}$ to a vertex
13: recurse on the graph with these vertices and the edges in $E_{>\text{med}} = E \setminus E_{\leq \text{med}}$

- In practice, Dijkstra's adaptation runs often faster than this (theoretically)-optimal variant

Computing the bottleneck shortest path (cont.)

We can actually compute the bottleneck shortest path in $O(|V| + |E|)$ [Har-Peled, Raichel14]

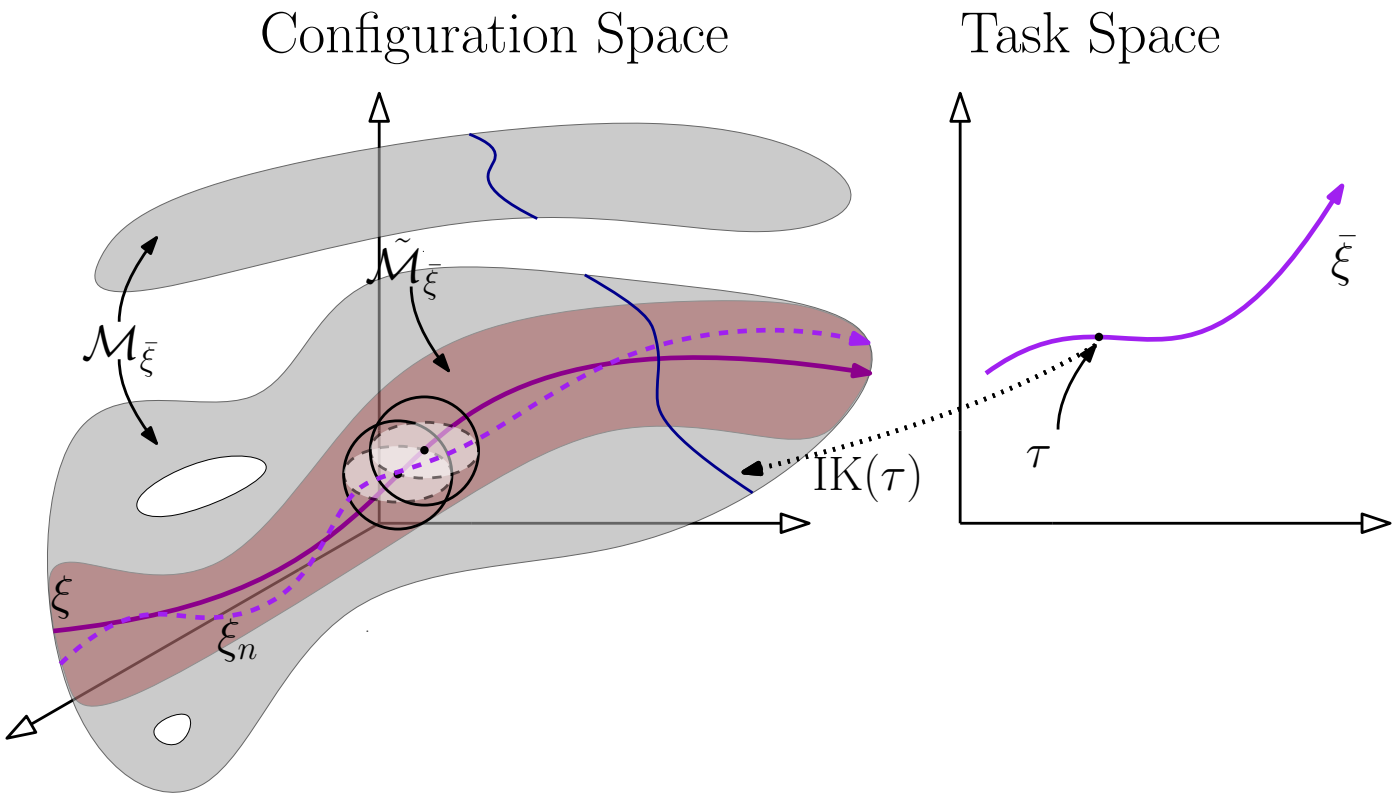
1: remove isolated vertices ▷ $O(|V|)$
2: check that s and t are connected ▷ $O(|E|)$
3: **if** G has constant size **then**
4: **compute** bottleneck shortest path brute force ▷ $O(1)$
5: **loop**
6: $e_{\text{med}} \leftarrow$ median weight edge ▷ $O(|E|)$
7: $E_{\leq \text{med}} \leftarrow \{e \in E | c(e) \leq c(e_{\text{med}})\}$; $G_{\leq \text{med}} \leftarrow (V, E_{\leq \text{med}})$ ▷ $O(|E|)$
8: compute the connected components of $G_{\leq \text{med}}$ ▷ $O(|E|)$
9: **if** s and t are in the same component of $G_{\leq \text{med}}$ **then**
10: recurse on $G_{\leq \text{med}}$ ▷ $b^* \leq c(e_{\text{med}})$
11: **else**
12: contract each connected component of $G_{\leq \text{med}}$ to a vertex
13: recurse on the graph with these vertices and the edges in $E_{>\text{med}} = E \setminus E_{\leq \text{med}}$

- In practice, Dijkstra's adaptation runs often faster than this (theoretically)-optimal variant

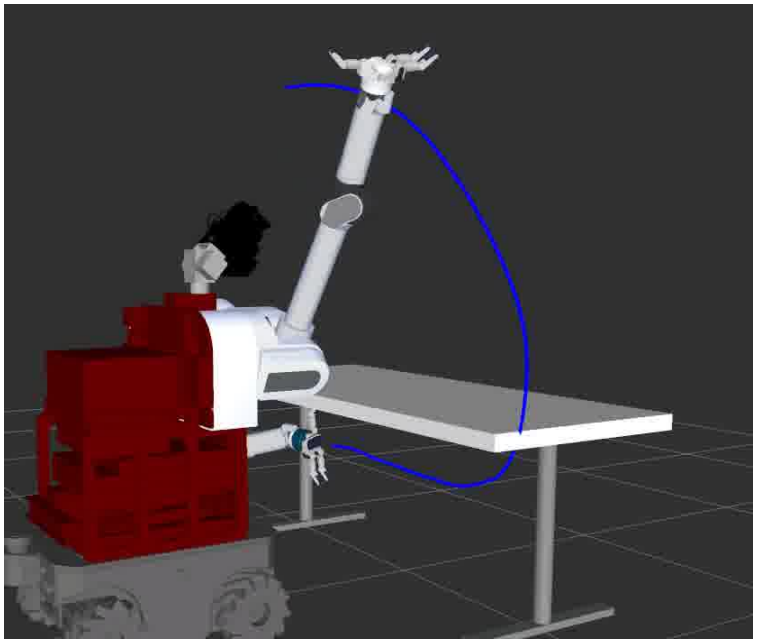
Properties [Holladay, S., Srinivassa19]

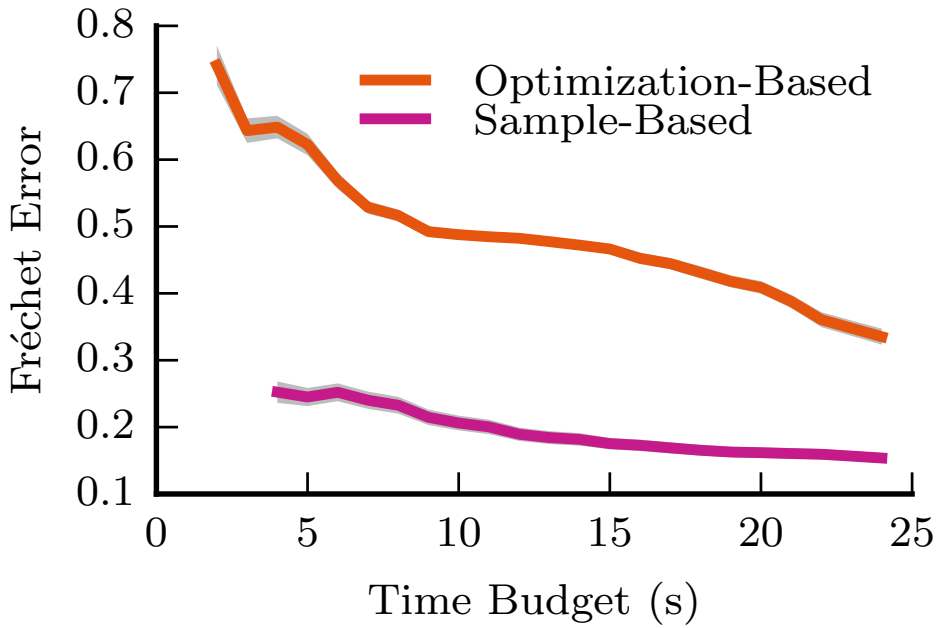
Thm

Let $M_{\zeta^*} \subset C_{\text{free}}$ be the the set of all configurations that directly map to ζ^*. If M_{ζ^*} contains a "well-behaved" portion, then the algorithm is asymptotically optimal (with respect to n and k)

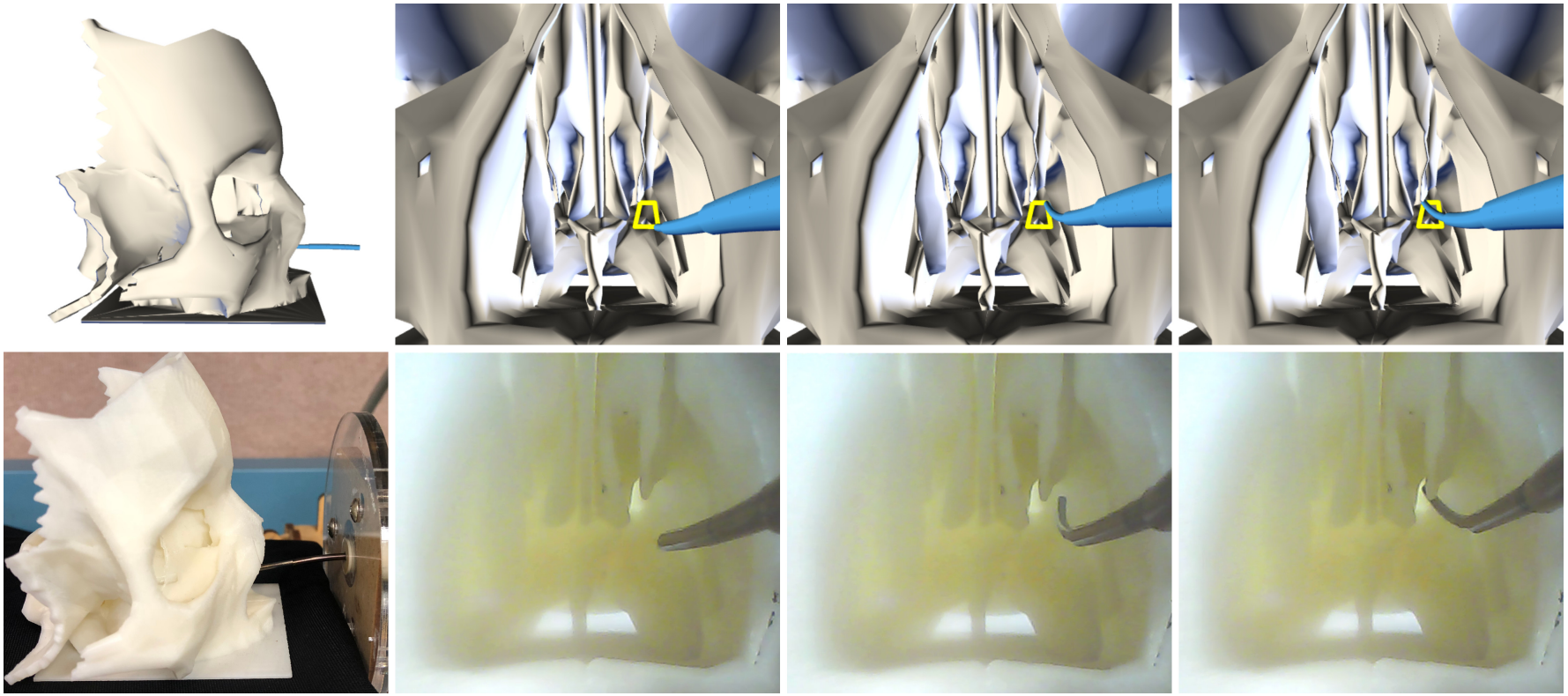


Evaluation [Holladay, S., Srinivassa19]

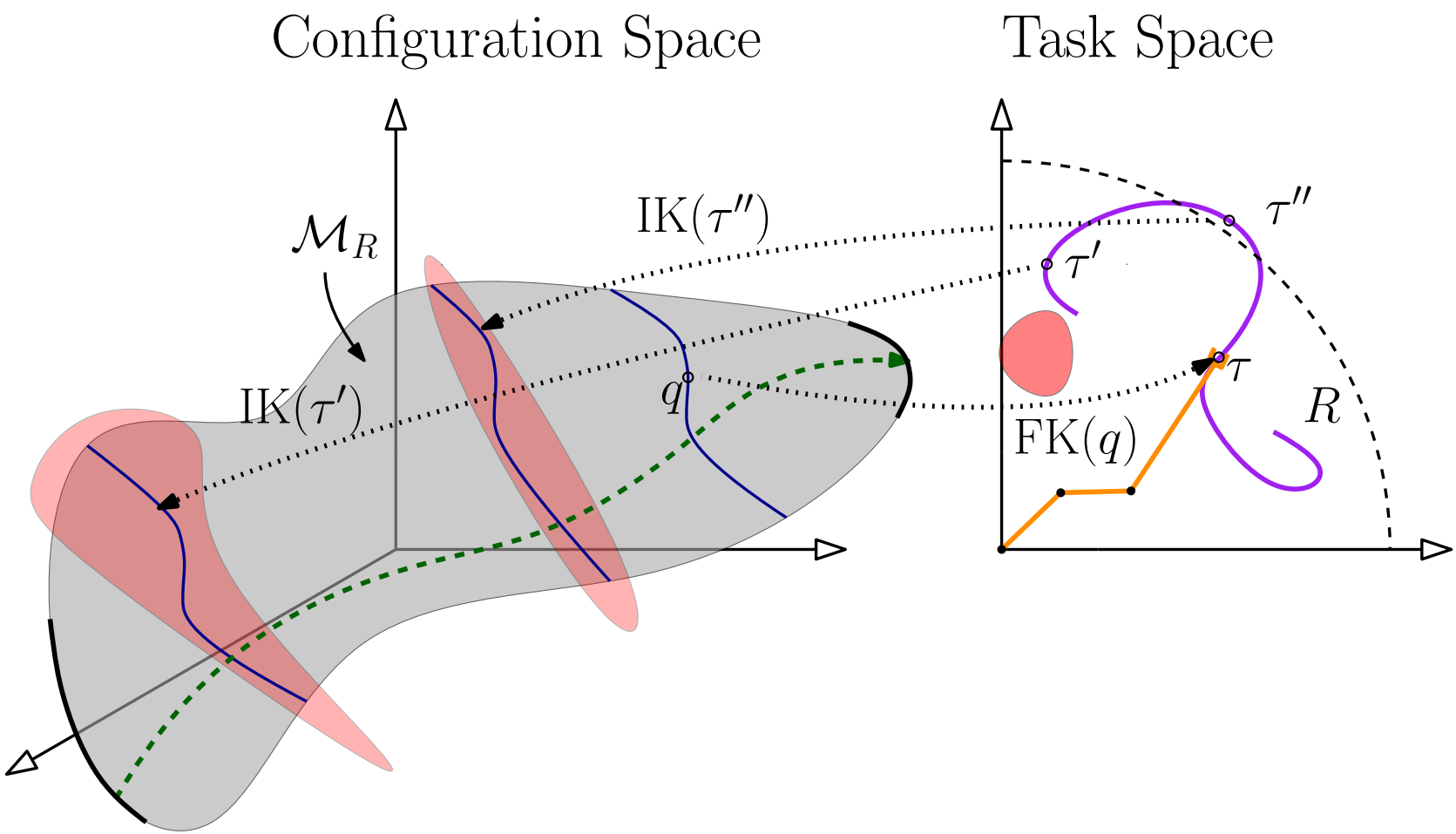




Key challenge—obstacles along the reference path



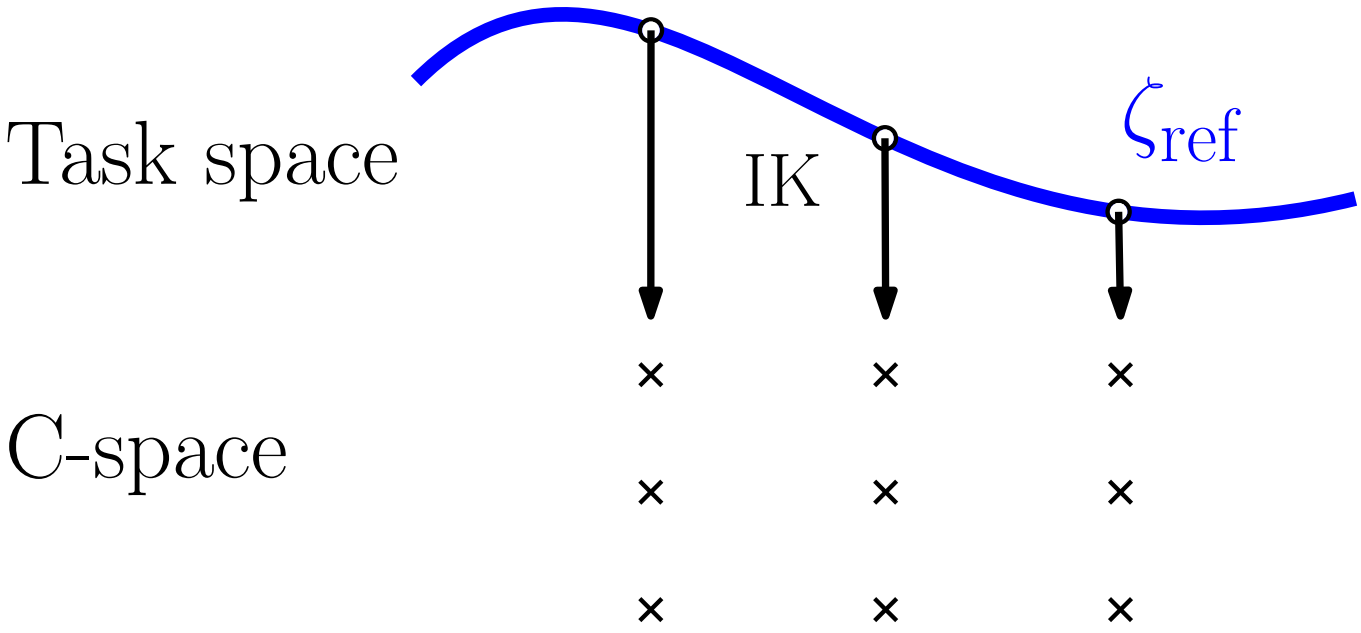
Key challenge—obstacles along the reference path



Approach—from layered graph to NN graph

Planning, take (2) [Holladay, Salzman, Srinivasa17]†

- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L

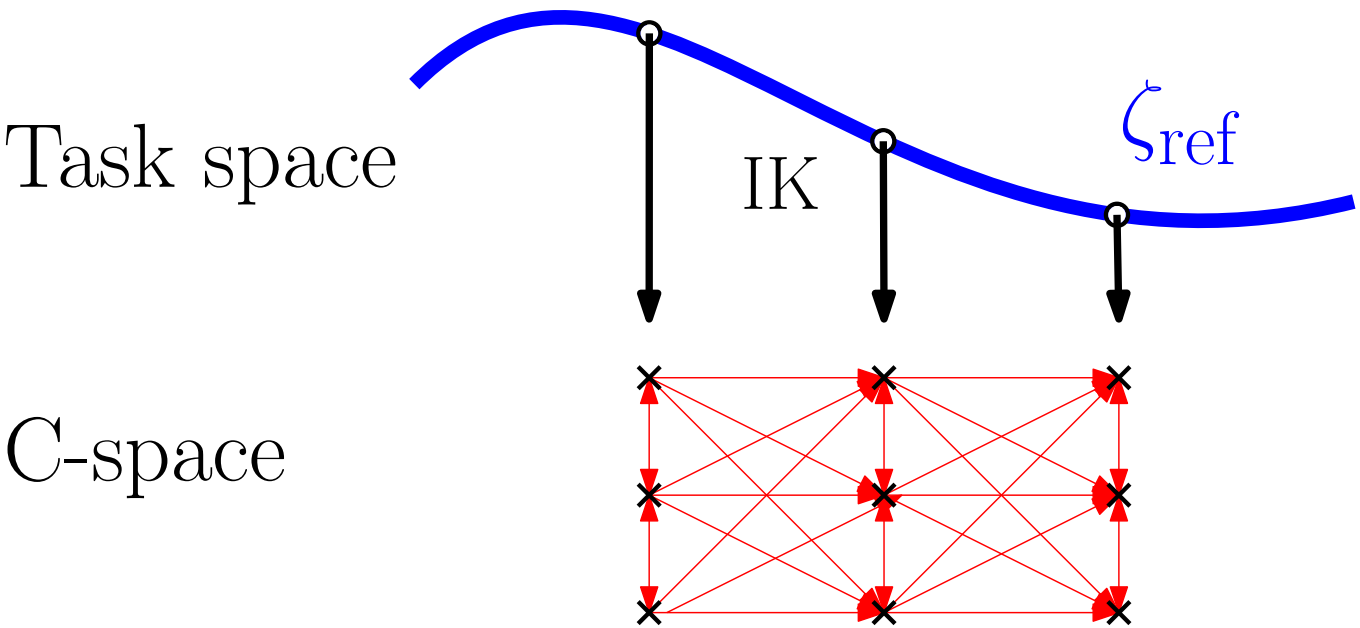


† Under review

Approach—from layered graph to NN graph

Planning, take (2) [Holladay, Salzman, Srinivasa17]†

- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L

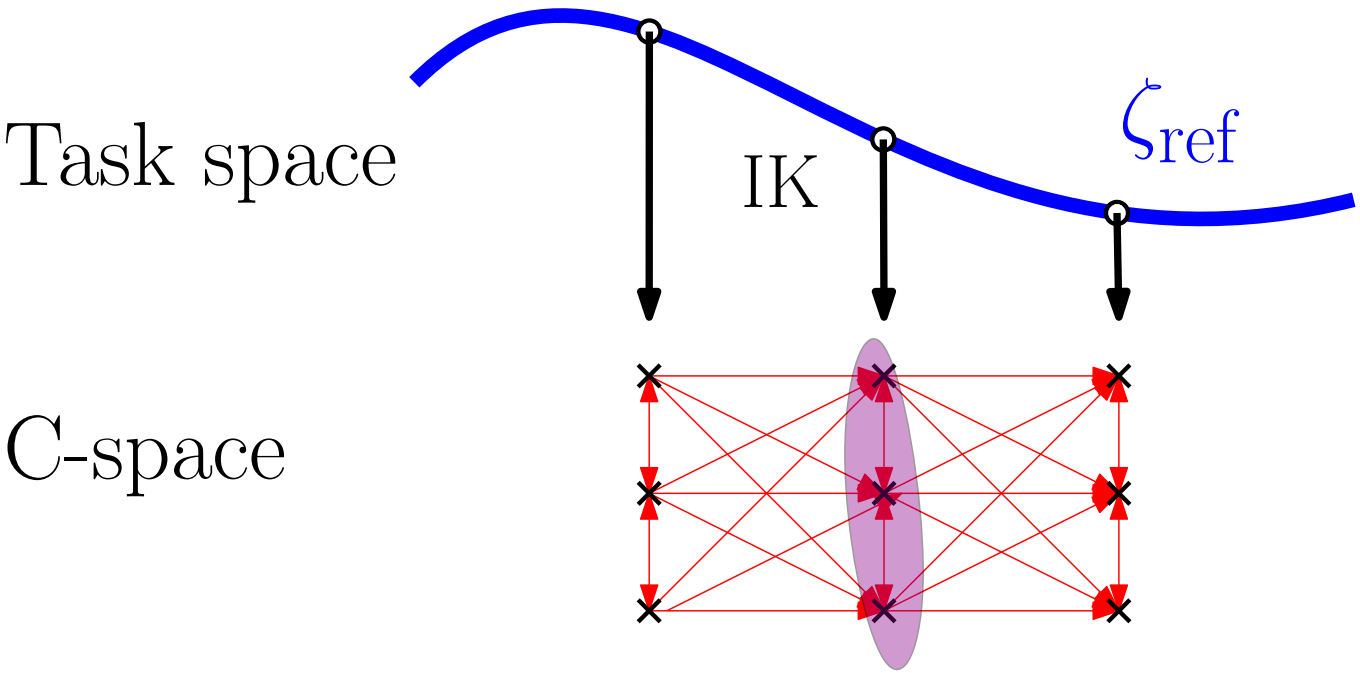


† Under review

Approach—from layered graph to NN graph

Planning, take (2) [Holladay, Salzman, Srinivasa17]†

- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a layered graph L
- Find minimal-cost path in L

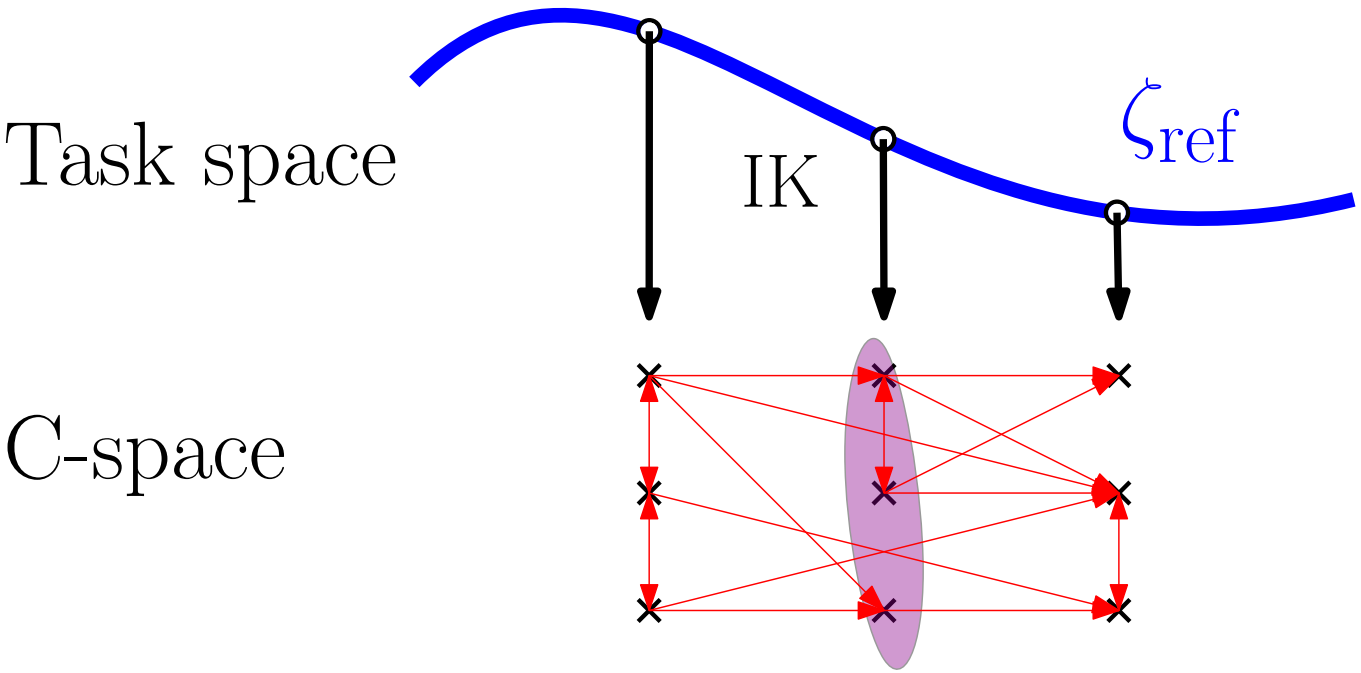


† Under review

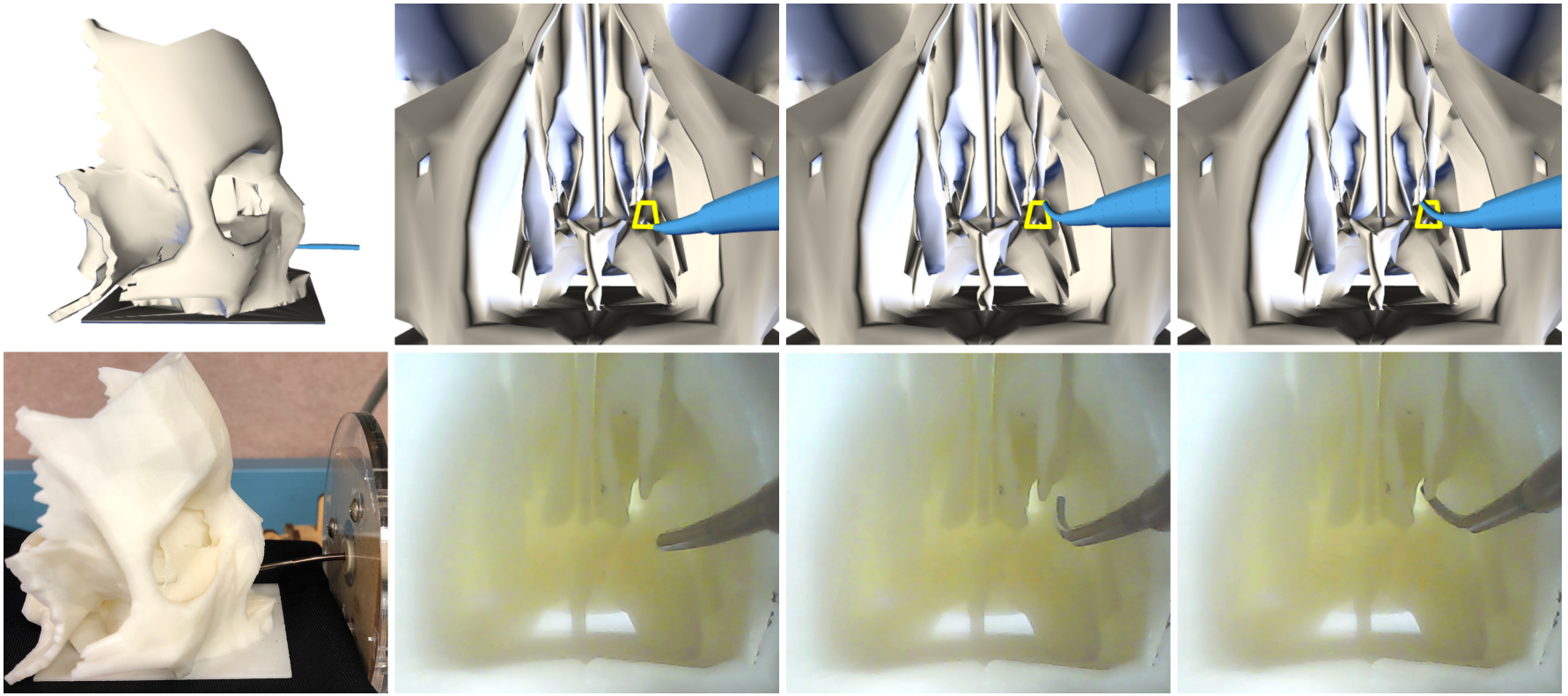
Approach—from layered graph to NN graph

Planning, take (3) [Niyaz, Kuntz, S., Alterovitz, Srinivasa18]

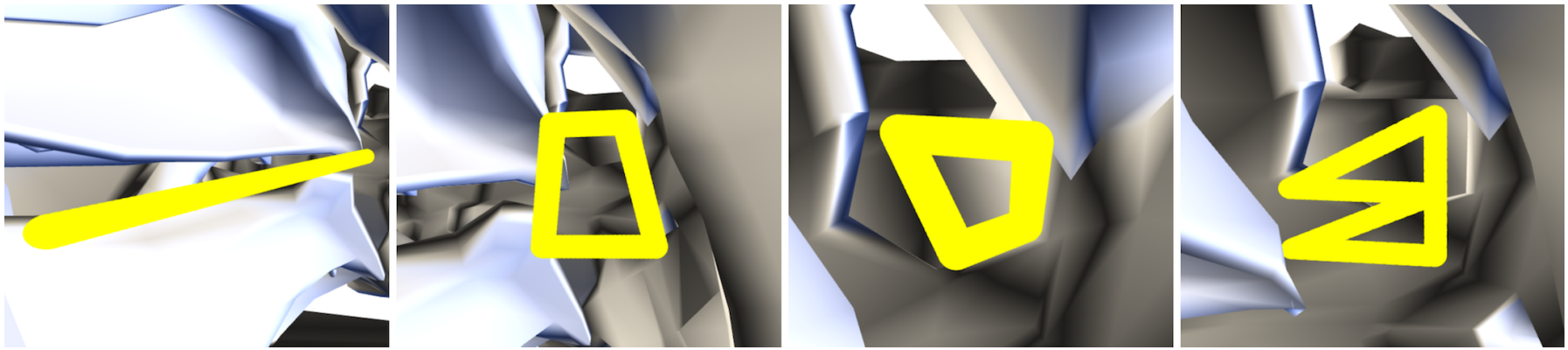
- Sample along ζ_{ref}
- Sample multiple IK solutions of each sample
- Construct a ~~layered~~ nearest-neighbor graph N
- Find minimal-cost path in N



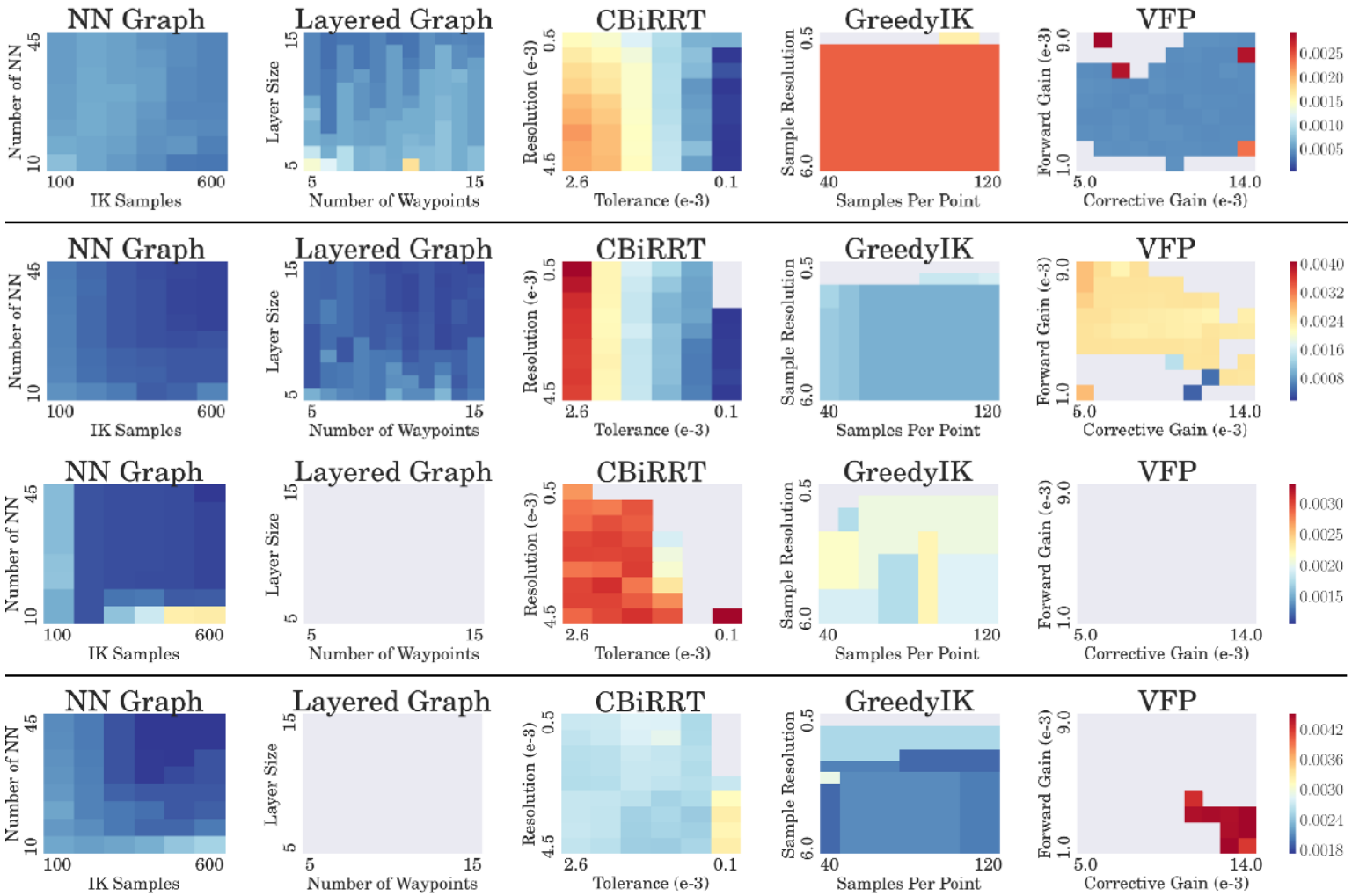
Experimental results—Simulation



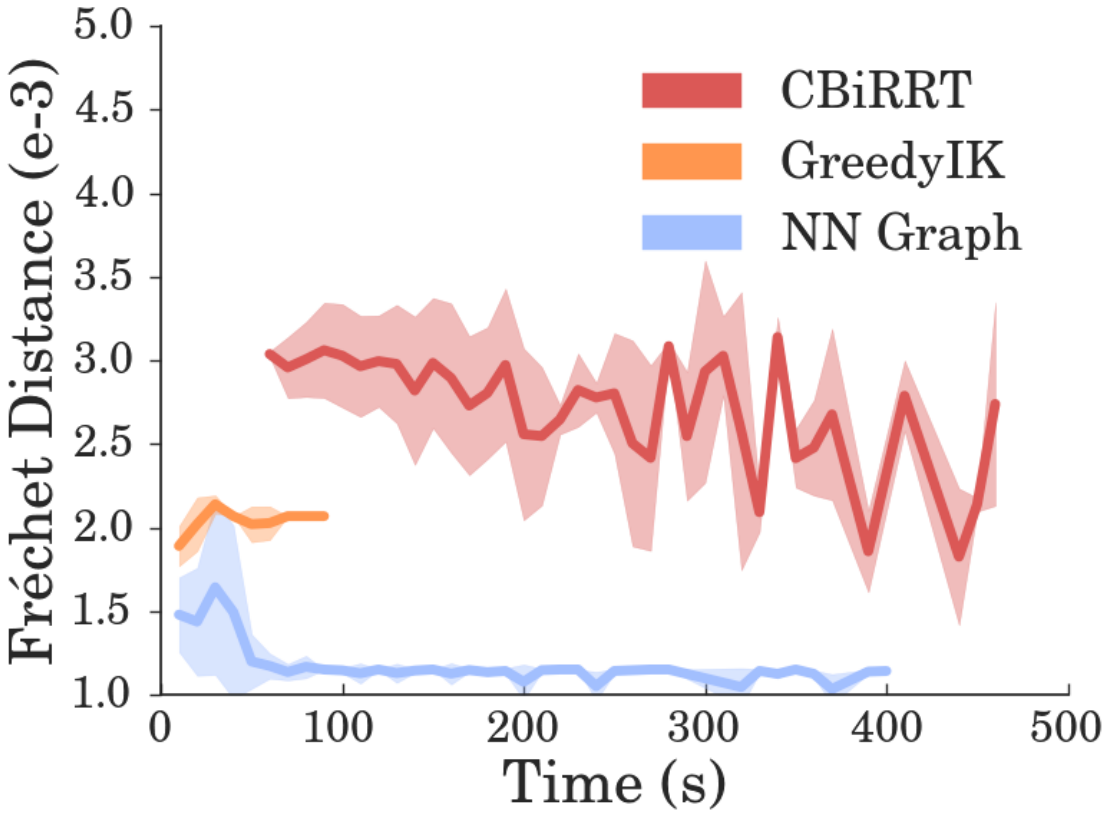
Experimental results—Simulation



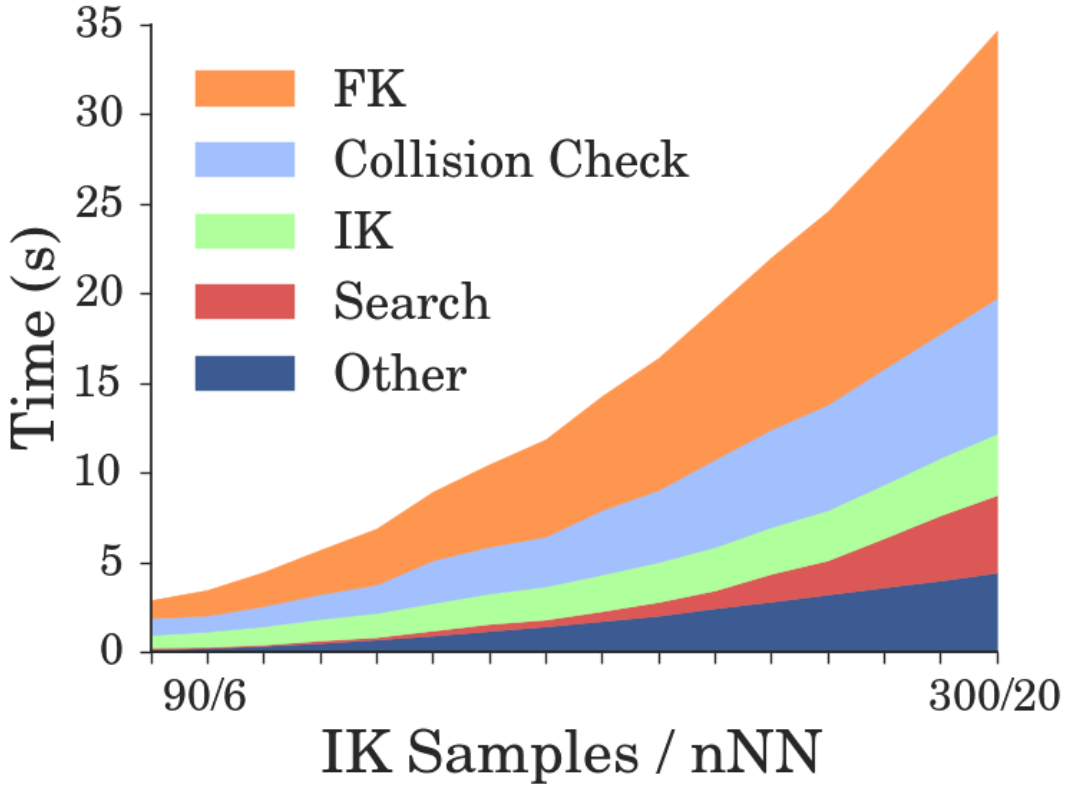
Experimental results—Simulation



Experimental results—Simulation



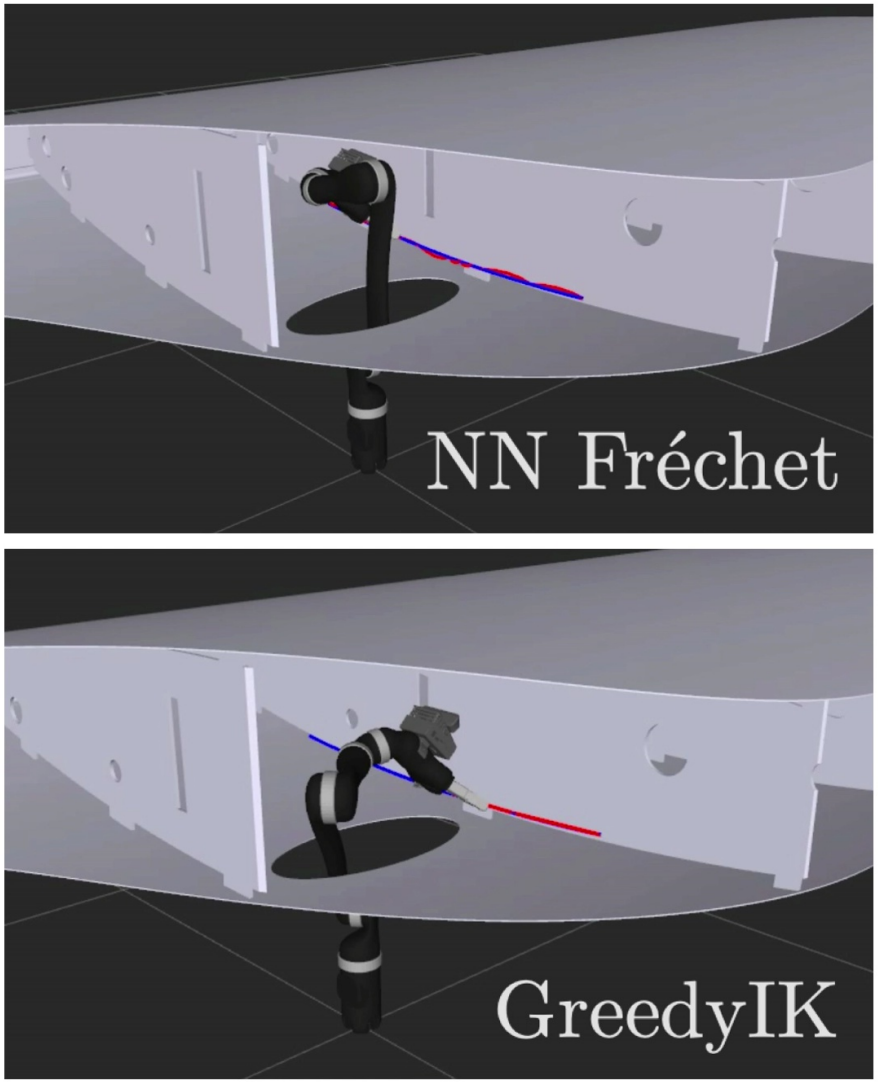
Experimental results—Simulation



Experimental results—Execution on CTR

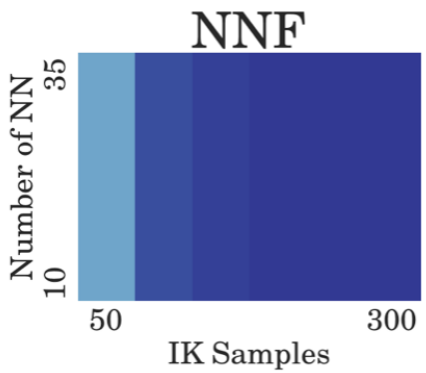
Frechet error: 8mm (real robot) vs. 2mm (simulation) due (primarily) to inaccuracies in kinematic model.

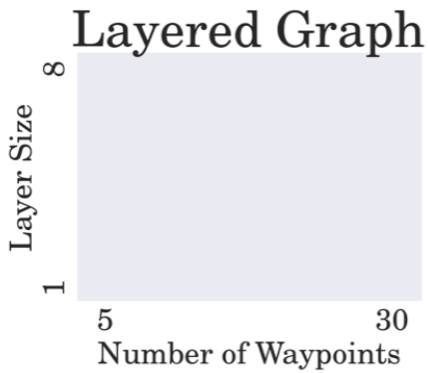
Experimental results—Aircraft wing inspection

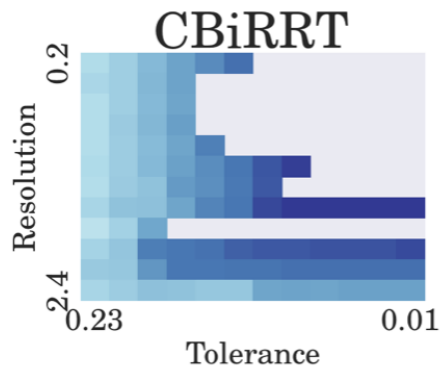


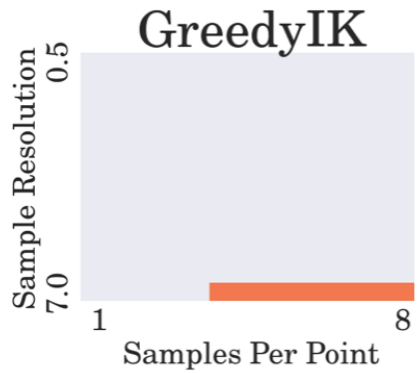
Experimental results—Aircraft wing inspection

Path M-1

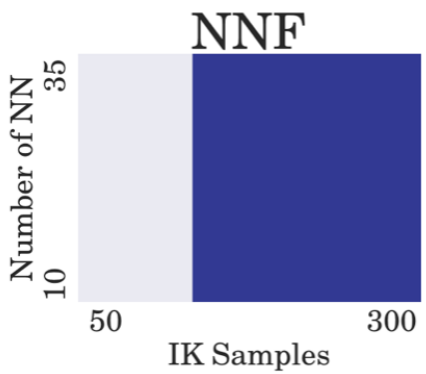


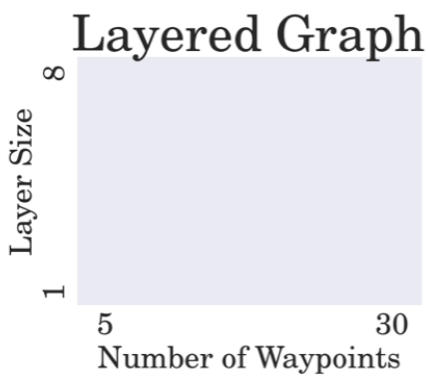


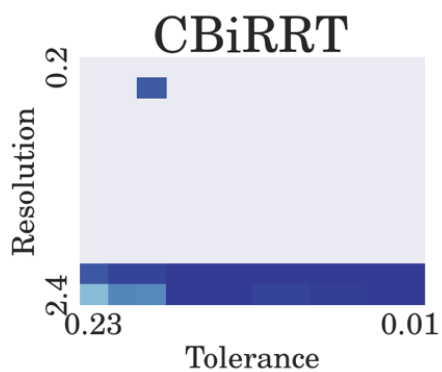


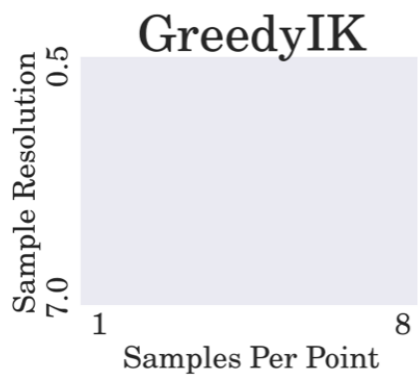


Path M-2

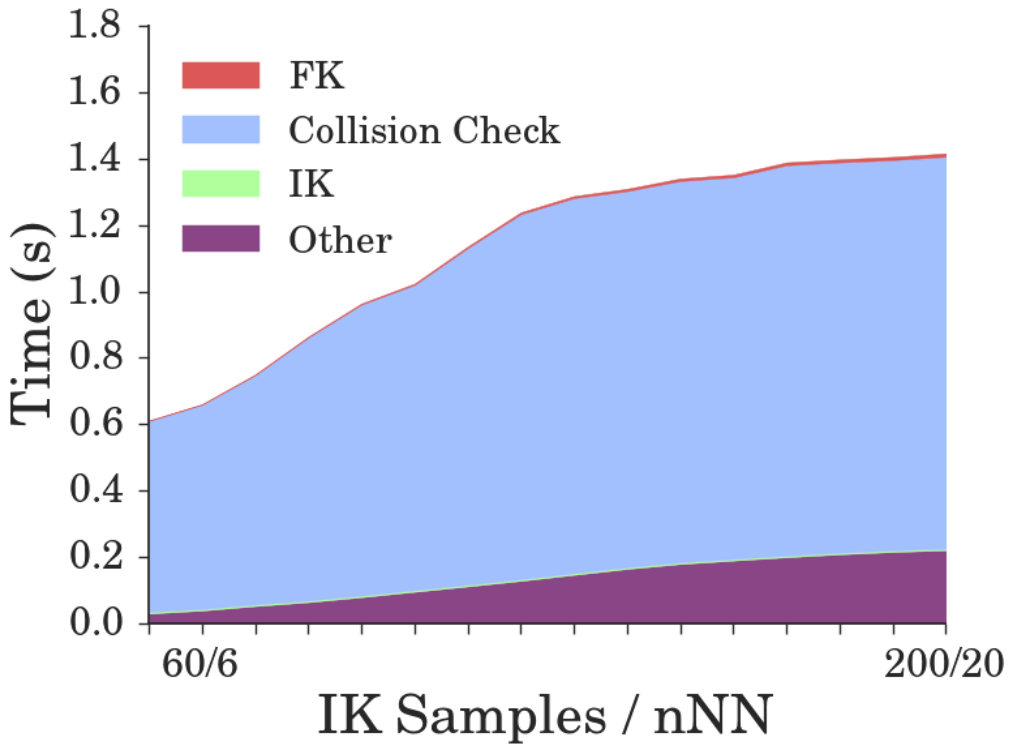




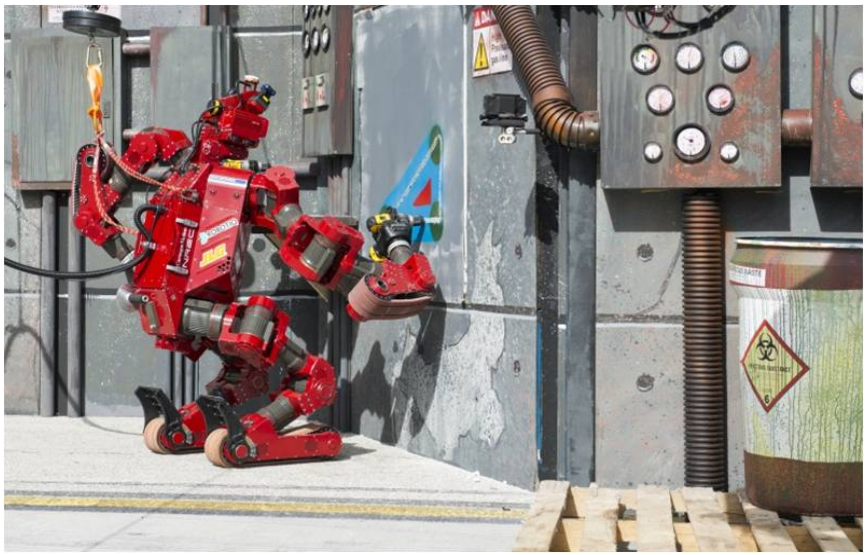


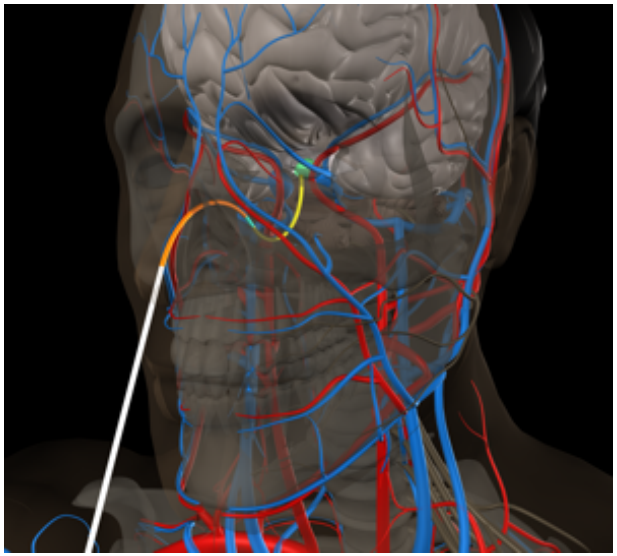


Experimental results—Aircraft wing inspection



Following Paths in Task Space

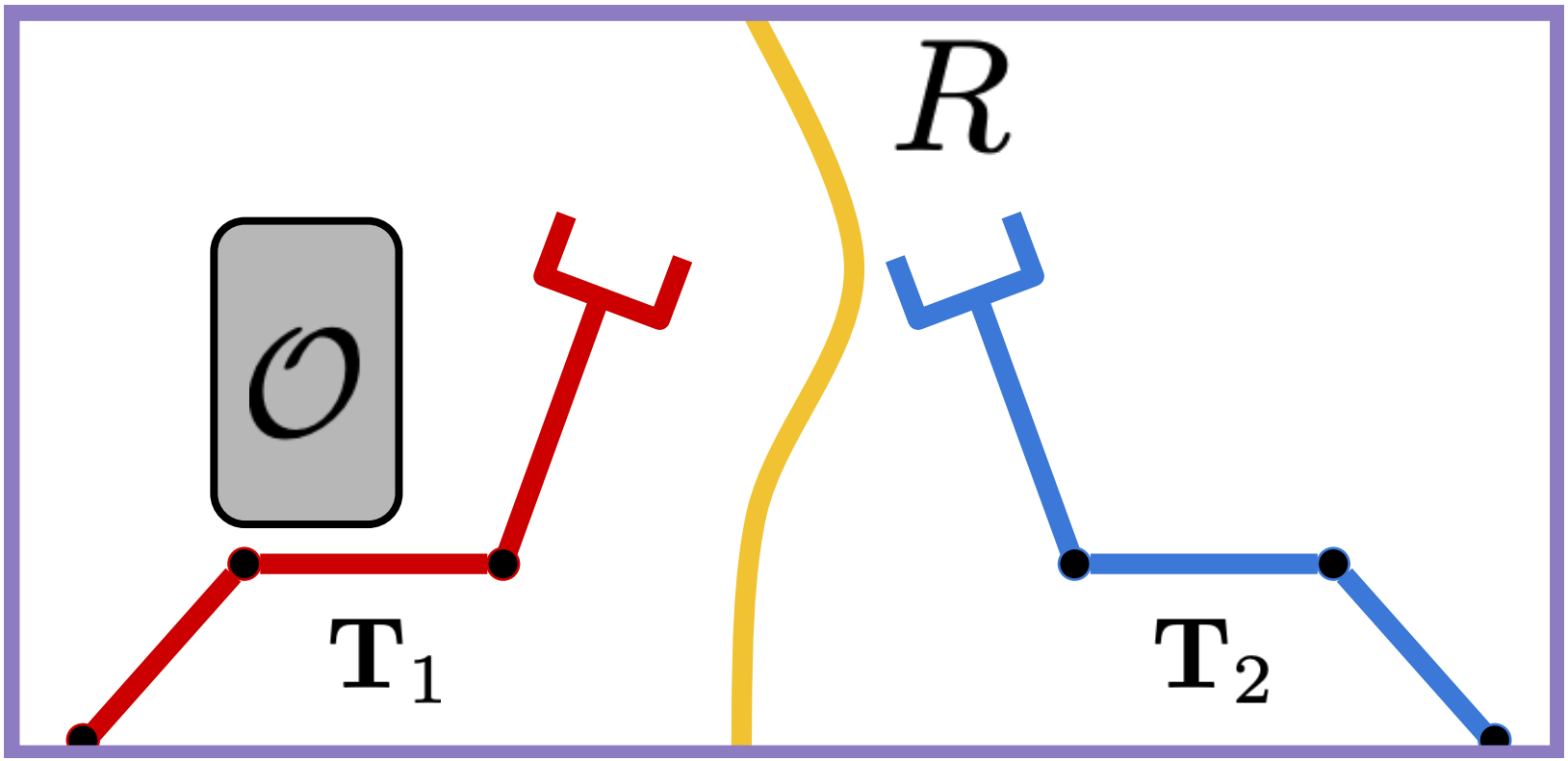




Can we do better?

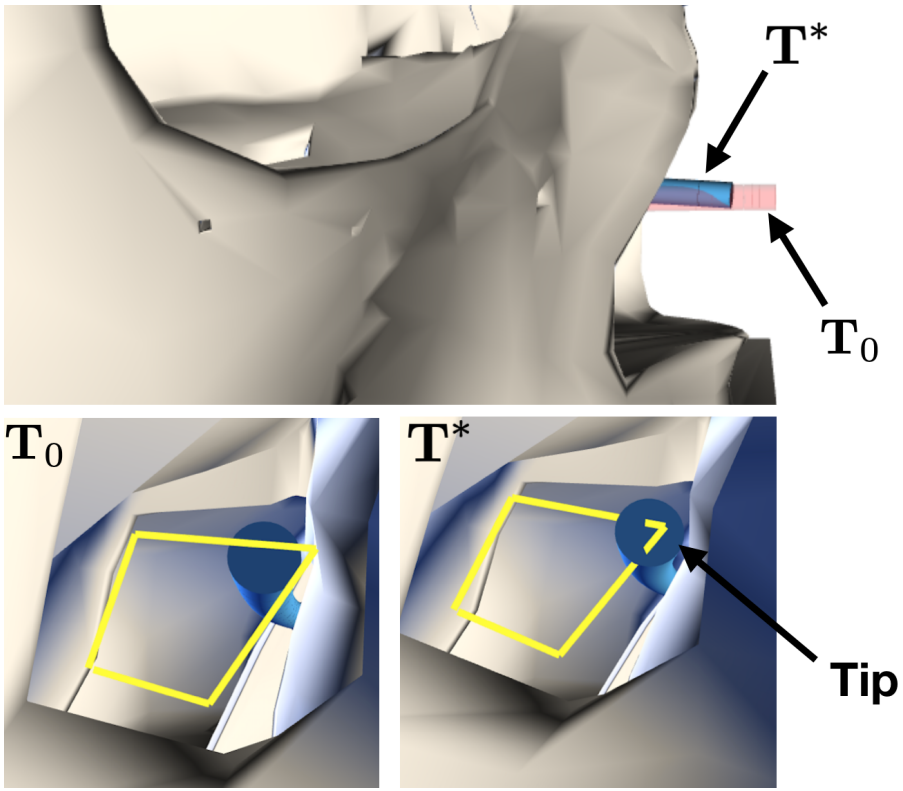
Following Paths in Task Space

Robot placement matters!



Following Paths in Task Space

Robot placement matters!



Following reference paths—Extending the optimization function

Let $T \in SE(3)$ be a placement of the robot base

$$\zeta^* = \underbrace{\arg\min_{\zeta \in \mathcal{C}_{\text{free}}}}_{\text{How do we plan ?}} \underbrace{||\text{FK}(\zeta), \zeta_{\text{ref}}||}_{\text{How do we measure distances ?}}$$

NN Graph

dFrećhet distance

Following reference paths—Extending the optimization function

Let $T \in SE(3)$ be a placement of the robot base

$$\zeta^* = \underbrace{\arg\min_{\zeta \in \mathcal{C}_{\text{free}}^T}}_{\text{How do we plan ?}} \underbrace{||\text{FK}^T(\zeta), \zeta_{\text{ref}}||}_{\text{How do we measure distances ?}}$$

NN Graph

dFrećhet distance

Following reference paths—Extending the optimization function

Let $T \in SE(3)$ be a placement of the robot base

$$\zeta^* = \arg\min_{T \in SE(3)} \underbrace{\arg\min_{\zeta \in \mathcal{C}^T_{\text{free}}}}_{\text{How do we plan ?}} \underbrace{||\text{FK}^{\text{T}}(\zeta), \zeta_{\text{ref}}||}_{\text{How do we measure distances ?}}$$

NN Graph

dFreć het distance

Following reference paths—Extending the optimization function

Let $T \in SE(3)$ be a placement of the robot base

$$\zeta^* = \underbrace{\arg\min_{T \in SE(3)}}_{\text{How do we optimzie ?}} \underbrace{\arg\min_{\zeta \in \mathcal{C}^T_{\text{free}}}}_{\substack{\text{How do we plan ?} \\ \text{NN Graph}}} \underbrace{||\text{FK}^{\text{T}}(\zeta), \zeta_{\text{ref}}||}_{\substack{\text{How do we measure distances ?} \\ \text{dFreć het distance}}}$$

Design or setup optimization

What we are actually trying to do is to optimize the setups of motionp-lanning problems to maximize the utility of a given task

(informal) definition of setup

The setup of a planning problem is the set of parameters we have control over that must be set before the planner is invoked

Design optimization via reachability maps

- Reachability maps can be used to approximate the cost of motions required by the robot as a function of placement [Vahrenkamp et al. 16; Makhal, Goins 17]
- They fail to account for actual motions required by the robot, especially in highly constrained tasks

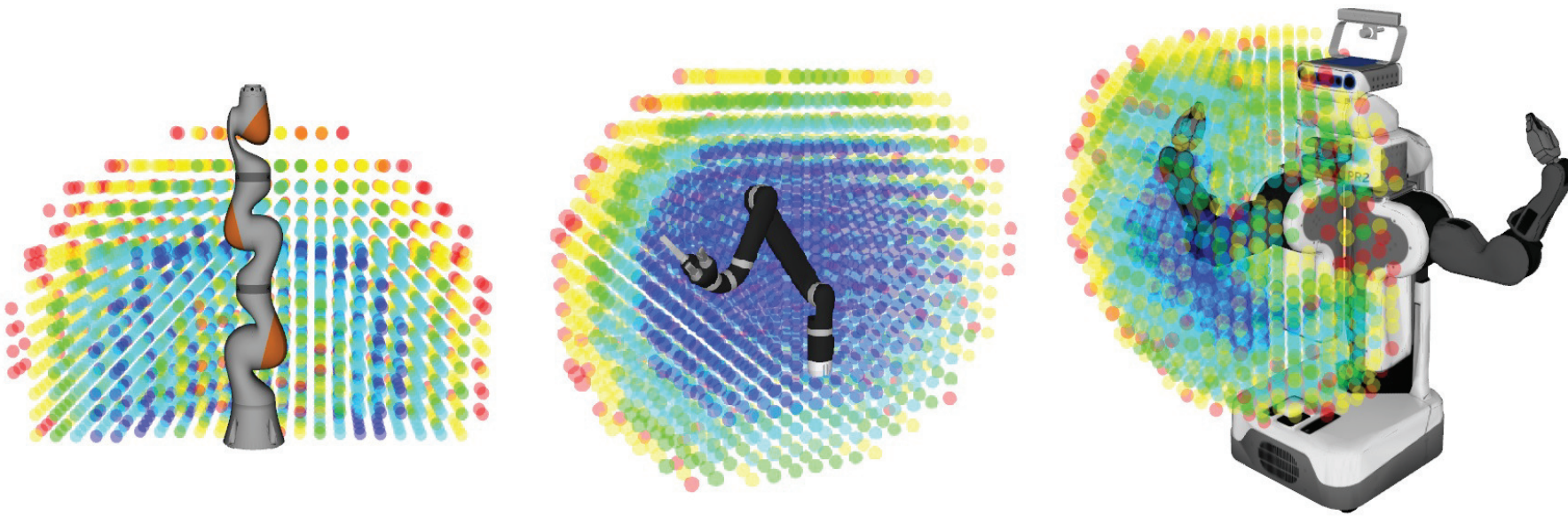
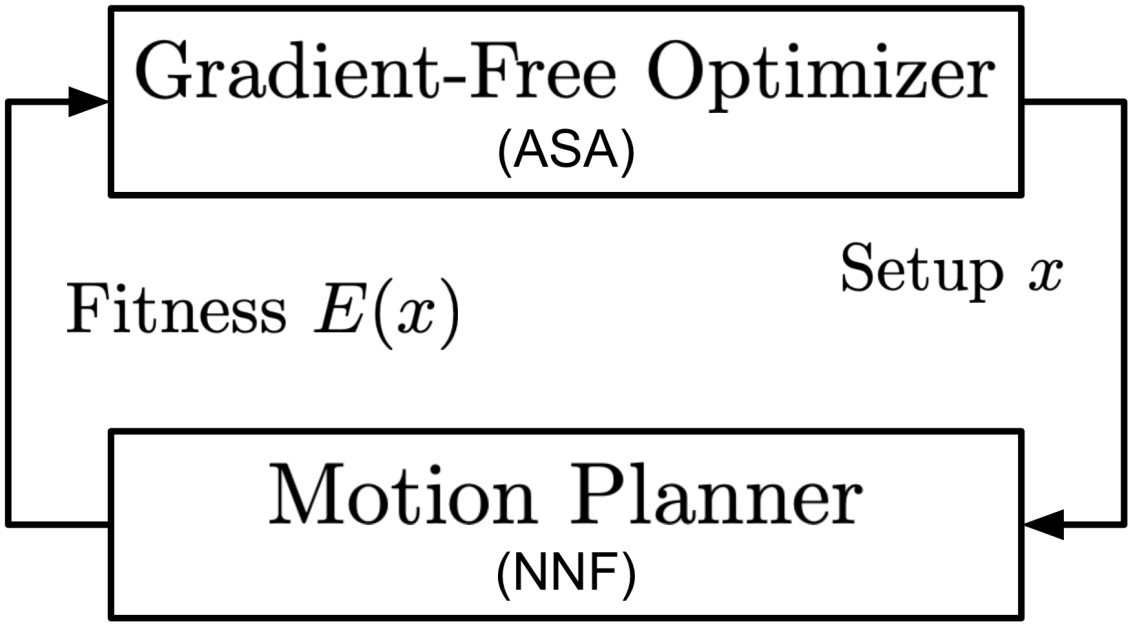


Figure adapted from [Makhal, Goins 17]

Design optimization via global optimization

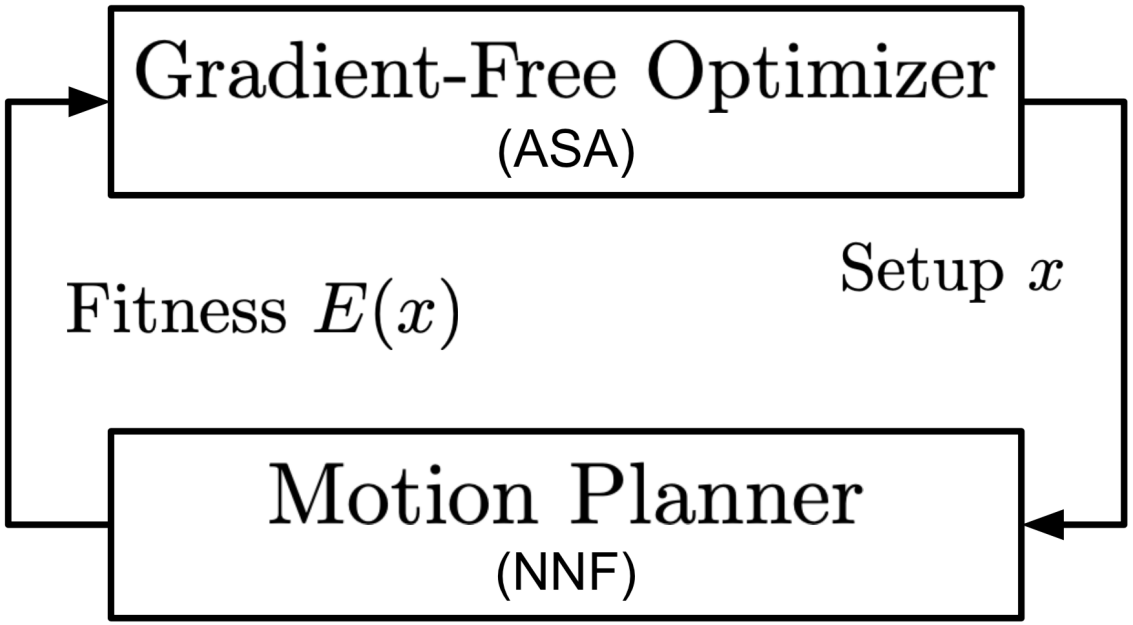
We want to use the motion planner to asses the quality of a design



- Combines gradient-free optimization over design (insertion pose) with a motion planner used to evaluate cost of each one

Design optimization via global optimization

We want to use the motion planner to asses the quality of a design



- Combines gradient-free optimization over design (insertion pose) with a motion planner used to evaluate cost of each one

Adaptive Simulated Annealing (ASA) [Locatelli 02]

- Annealing is a thermal process for obtaining low energy states of a solid in a heat bath
- Two-step process
  - Increase temperature of heat bath to a maximum value at which the solid melts
  - Carefully decrease temperature of heat bath until particles arrange themselves in solid's ground state
- Ground state is a minimum energy state of the solid
- The ground state is obtained only if the maximum temperature is high enough and the cooling is done slowly

Adaptive Simulated Annealing (ASA) [Locatelli 02]

- Analogy with thermodynamics
- Incorporate a temperature parameter into the minimization procedure
- At high temperatures, explore parameter space
- At lower temperatures, restrict exploration

Adaptive Simulated Annealing (ASA) [Locatelli 02]

- Consider decreasing series of temperatures
- For each temperature, iterate these steps:
  - Propose an update and evaluate function
  - Accept updates that improve solution
  - Accept some updates that don't improve solution
    - Acceptance probability depends on "temperature" parameter
- If cooling is sufficiently slow, the global minimum will be reached

Adaptive Simulated Annealing (ASA) [Locatelli 02]

Each iteration,

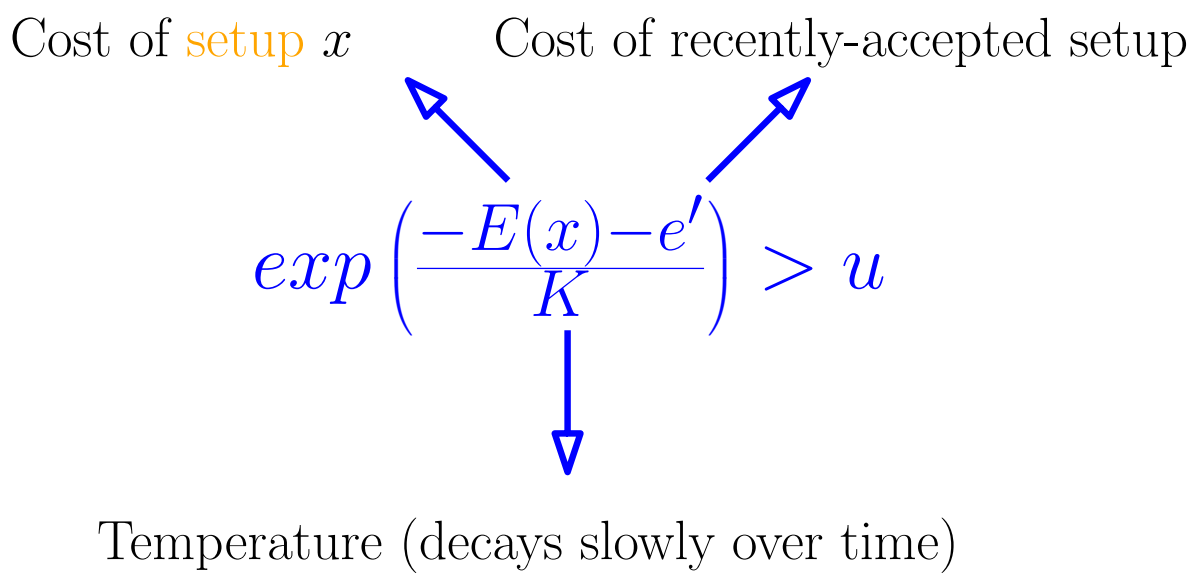
- sample some $u \in [0, 1]$ uniformly at random
- sample a new setup T "close" to the old setup
- accept T, if:

$$exp\left(\frac{-E(x)-e'}{K}\right) > u$$

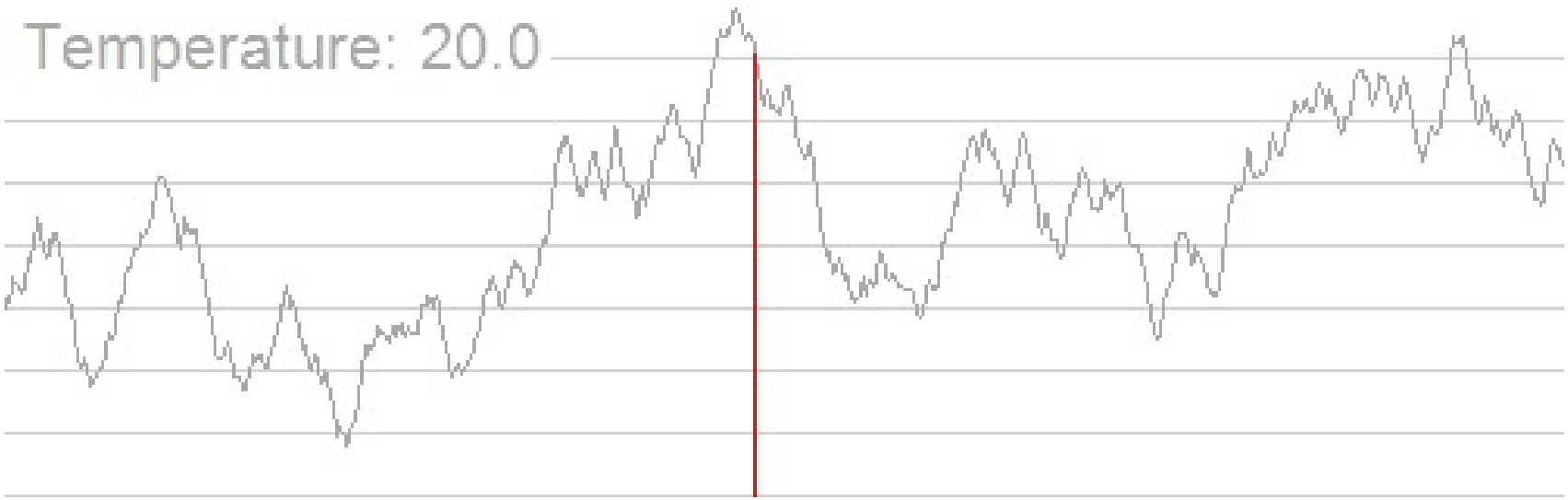
Adaptive Simulated Annealing (ASA) [Locatelli 02]

Each iteration,

- sample some $u \in [0, 1]$ uniformly at random
- sample a new setup T "close" to the old setup
- accept T, if:



Adaptive Simulated Annealing (ASA) [Locatelli 02]



Adapted from https://en.wikipedia.org/wiki/Simulated_annealing

Recap - ASA for design optimization

- ASA was used to optimize the kinematic design of surgical robots [Baykal, Bowen, Alterovitz 18; Kuntz et al. 18]
- These methods treat the planner as a black box
- However, the planner is computationally expensive

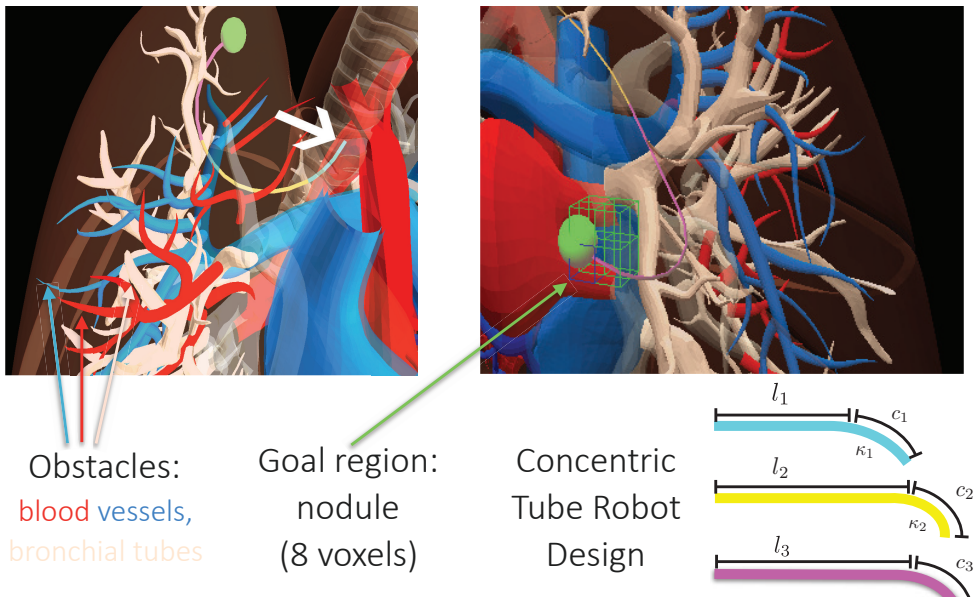


Figure adapted from [Baykal, Alterovitz 17]

ASA for efficient design optimization

- Key insight: open the black box optimizer
- We can show that a setup T cannot be accepted if

$$E(x) \geq \mathcal{B} \text{ s.t. } \mathcal{B} = e' - (K \cdot \ln(u))$$

- Useful!—Many motion planners are able to lower-bound the cost of their final solution as it is being computed
  - E.g. priority of node popped from priority queue in Dijkstra

ASA for efficient design optimization

- Key insight: open the black box optimizer
- We can show that a setup T cannot be accepted if

$$E(x) \geq \mathcal{B} \text{ s.t. } \mathcal{B} = e' - (K \cdot \ln(u))$$

- Useful!—Many motion planners are able to lower-bound the cost of their final solution as it is being computed
  - E.g. priority of node popped from priority queue in Dijkstra

ASA for efficient design optimization

- Key insight: open the black box optimizer
- We can show that a setup T cannot be accepted if

$$E(x) \geq \mathcal{B} \text{ s.t. } \mathcal{B} = e' - (K \cdot \ln(u))$$

- Useful!—Many motion planners are able to lower-bound the cost of their final solution as it is being computed
  - E.g. priority of node popped from priority queue in Dijkstra

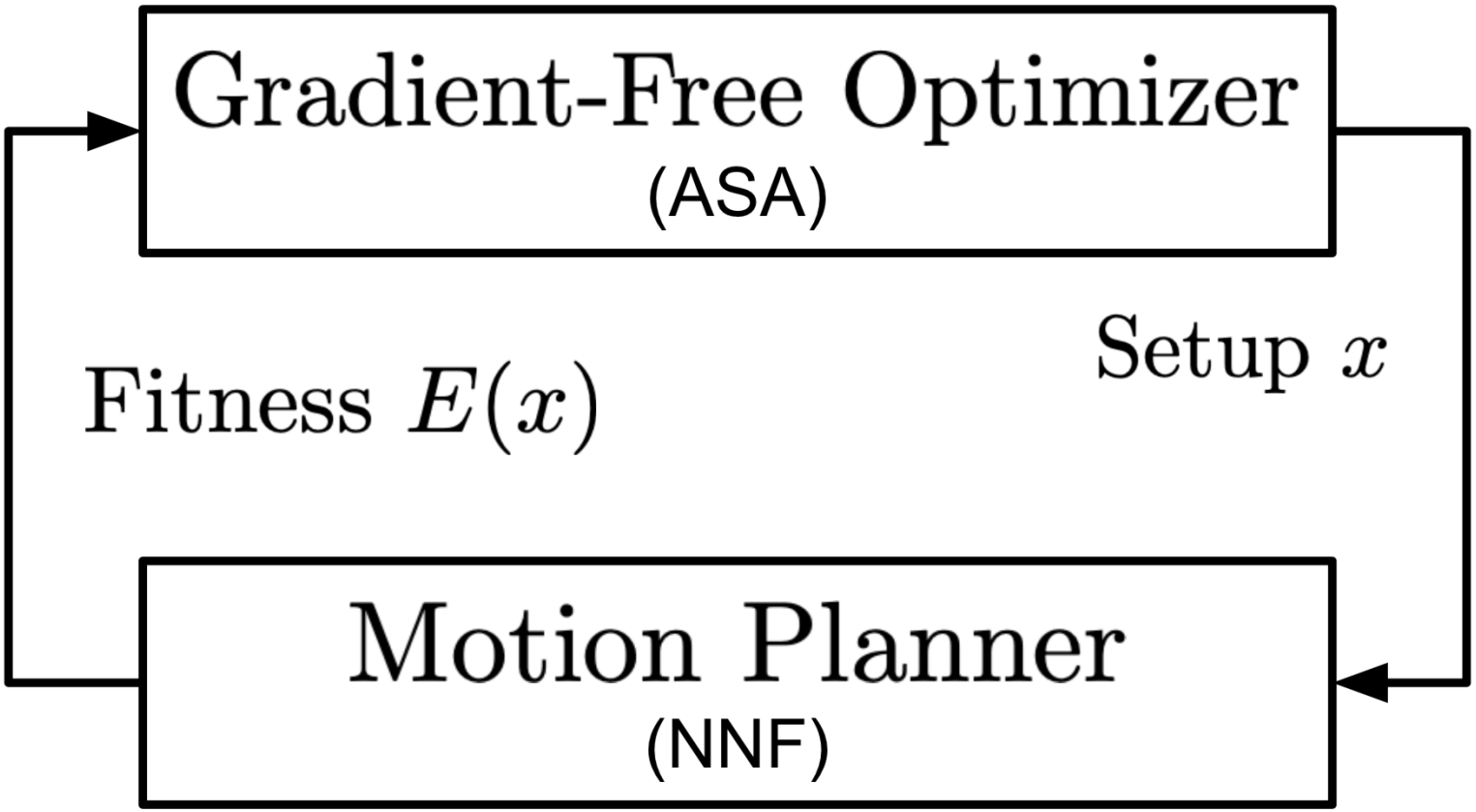
ASA for efficient design optimization

- Key insight: open the black box optimizer
- We can show that a setup T cannot be accepted if

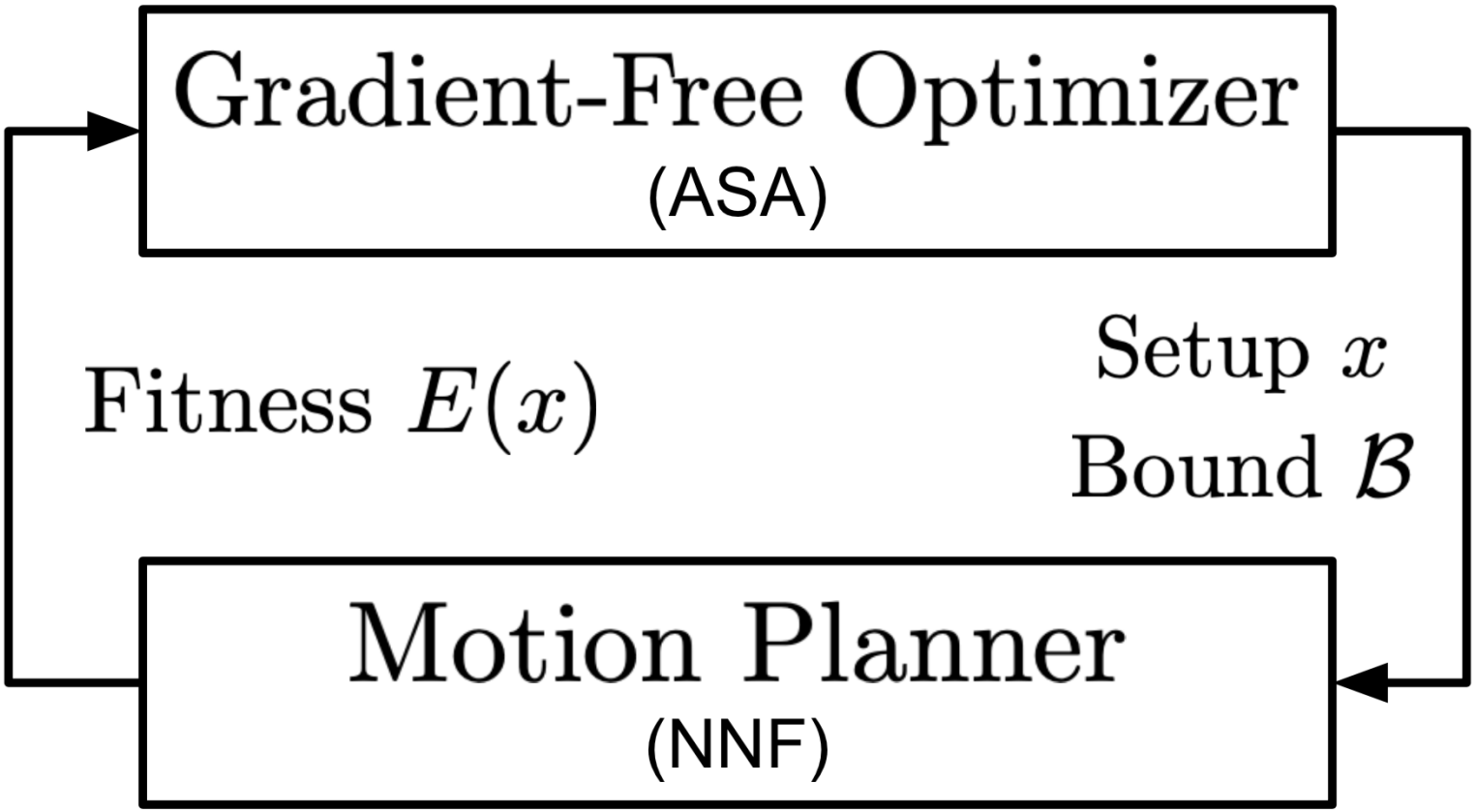
$$E(x) \geq \mathcal{B} \text{ s.t. } \mathcal{B} = e' - (K \cdot \ln(u))$$

- Useful!—Many motion planners are able to lower-bound the cost of their final solution as it is being computed
  - E.g. priority of node popped from priority queue in Dijkstra

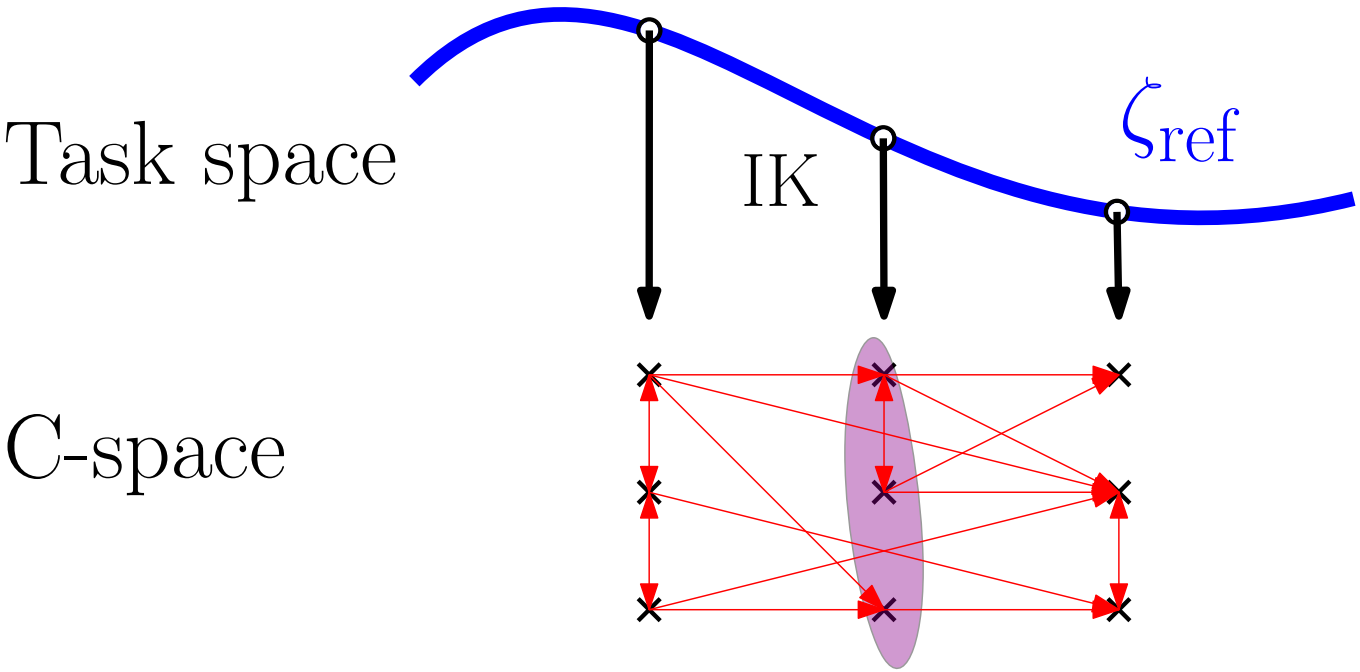
ASA for efficient design optimization



ASA for efficient design optimization

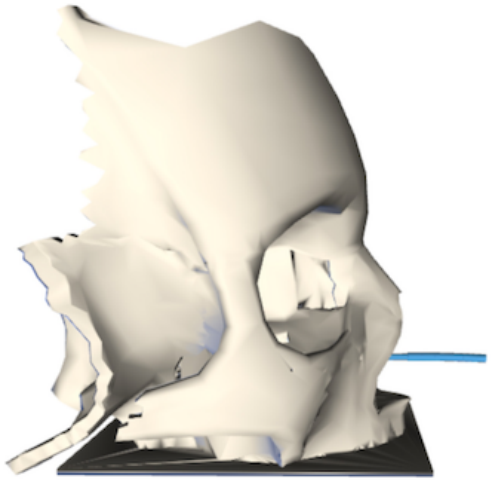


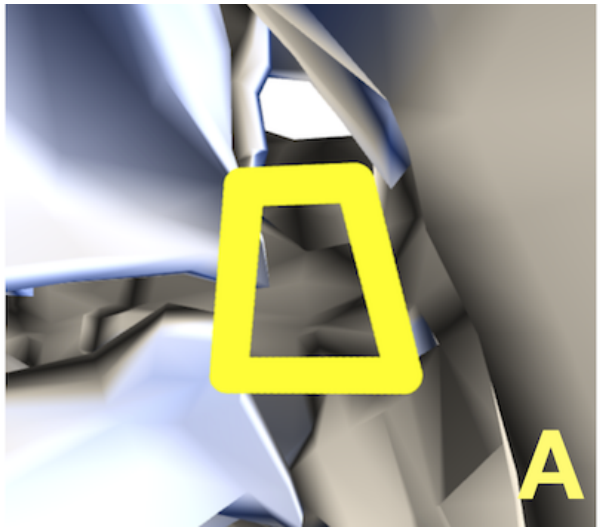
Opening the black box

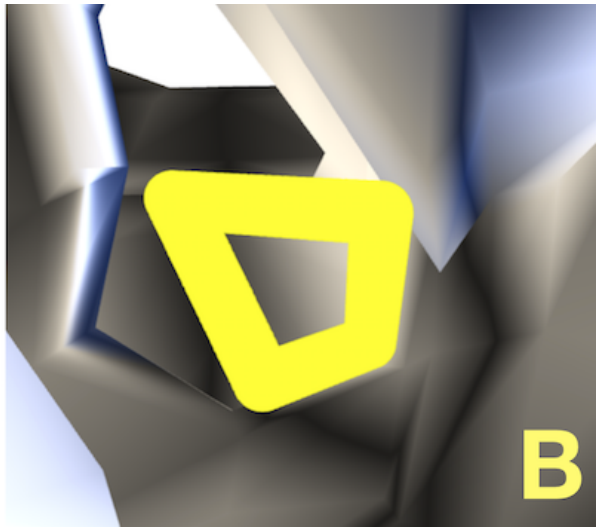


- For every node n expanded by the search algorithm, compare its priority $\text{key}(n)$ to $B = e' - (K \cdot \ln(u))$
- Abort if $\text{key}(n) > B$ (further computation is useless to the optimizer)

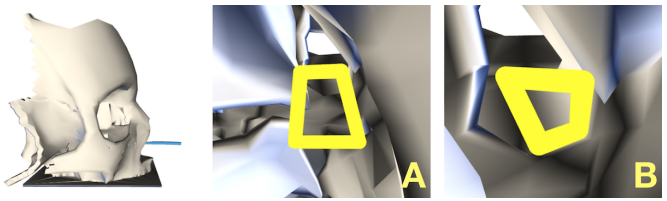
Evaluation

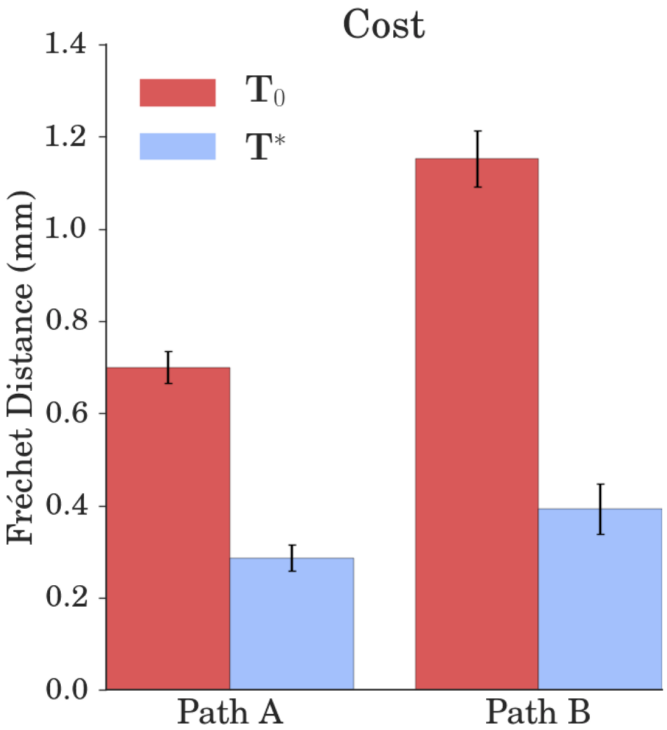


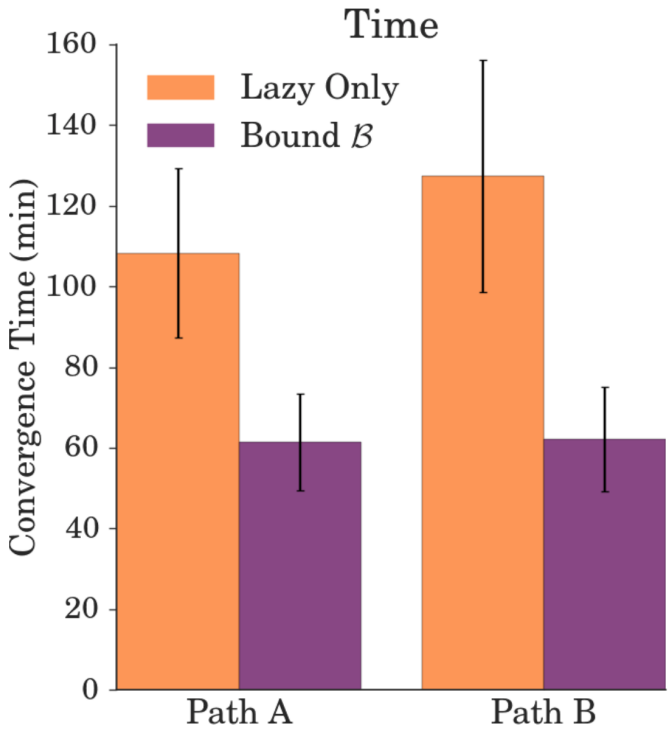




Evaluation


A
B


Cost
T_0
T^*
Fréchet Distance (mm)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Path A
Path B


Time
Lazy Only
Bound $\mathcal{B}$
Convergence Time (min)
160
140
120
100
80
60
40
20
0
Path A
Path B

Evaluation