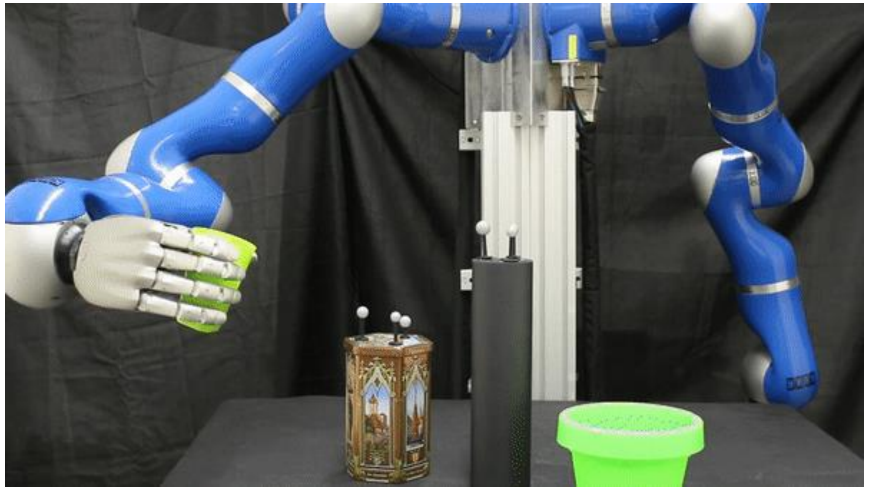
Accelerating Motion Planning Via Optimal Transport

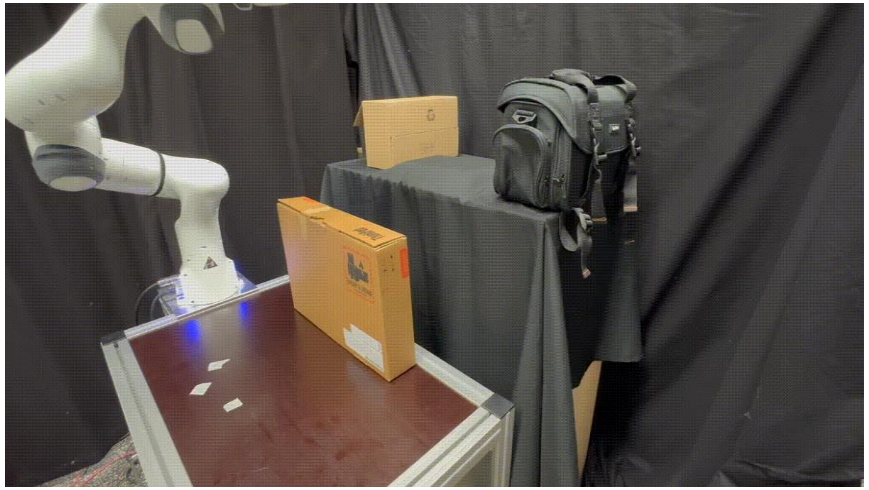
An T. Le, Georgia Chalvatzaki, Armin Biess, Jan Peters

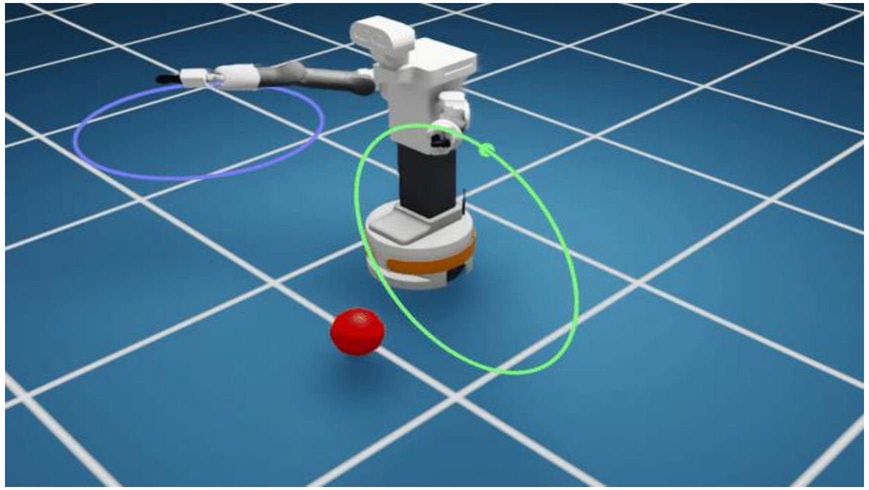
NEURAL INFORMATION PROCESSING SYSTEMS

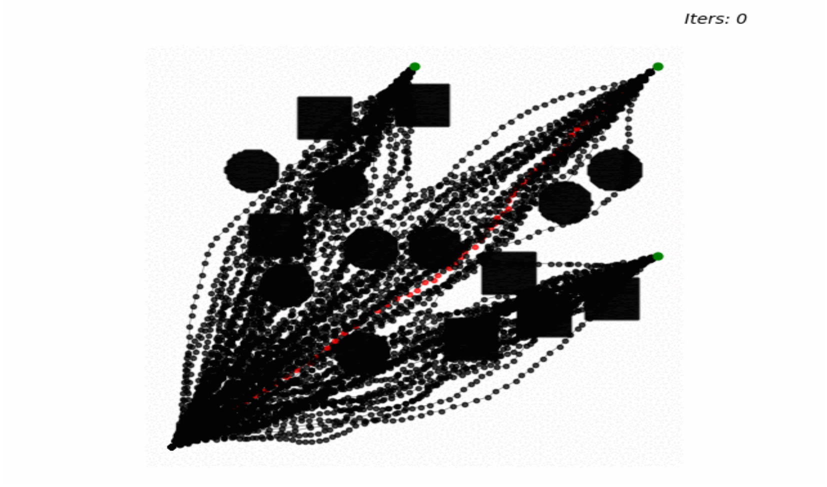
Optimal Transport and Machine Learning Workshop 2023

Planning is reliable ☺







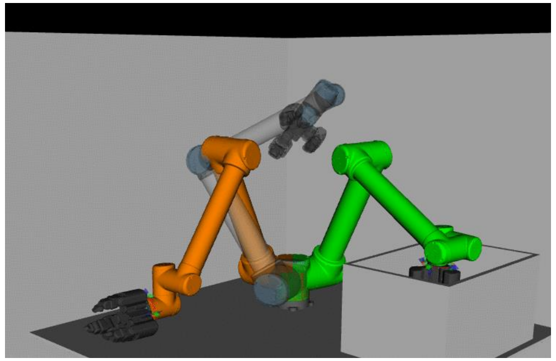


Urain, J.; **Le, A.T.**; Lambert, A.; Chalvatzaki, G.; Boots, B.; Peters, J. (2022). Learning Implicit Priors for Motion Optimization, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
Carvalho, J.; **Le, A.T.**; Baierl, M.; Koert, D.; Peters, J. (2023). Motion Planning Diffusion: Learning and Planning of Robot Motions with Diffusion Models, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
Le, A. T.; Hansel, K.; Peters, J.; Chalvatzaki, G. (2023). Hierarchical Policy Blending As Optimal Transport, 5th Annual Learning for Dynamics & Control Conference (L4DC), PMLR.
Le, A. T.; Chalvatzaki, G.; Biess, A.; Peters, J. (2023). Accelerating Motion Planning via Optimal Transport, NeurIPS 2023.

Motion planning: Overview

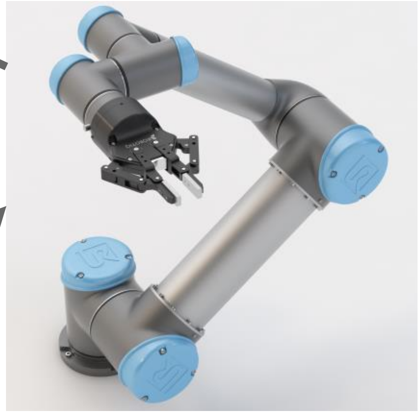
Task-level objectives

High-level planner, Typically 0-10Hz



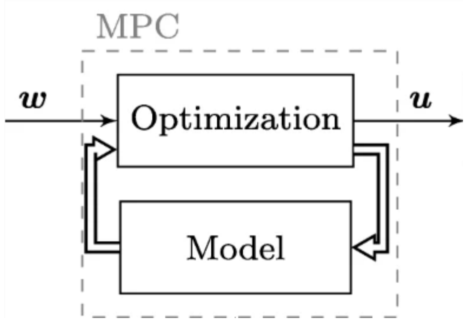
Constraints

Real Robot



Reference trajectory

Low-level controller, Typically > 100Hz



Control signals

Trajectory Optimization: Collocation method

$$\boldsymbol{\tau} = \left[\mathbf{x}_0, \boldsymbol{u}_0, ..., \mathbf{x}_{T-1}, \boldsymbol{u}_{T-1}, \mathbf{x}_T\right]^\intercal$$

$$\boldsymbol{\tau}^* = \arg\min_{\boldsymbol{\tau}} \sum_i \lambda_i c_i(\boldsymbol{\tau})$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \text{ and } \boldsymbol{\tau}(0) = \boldsymbol{x}_0$$

Model function

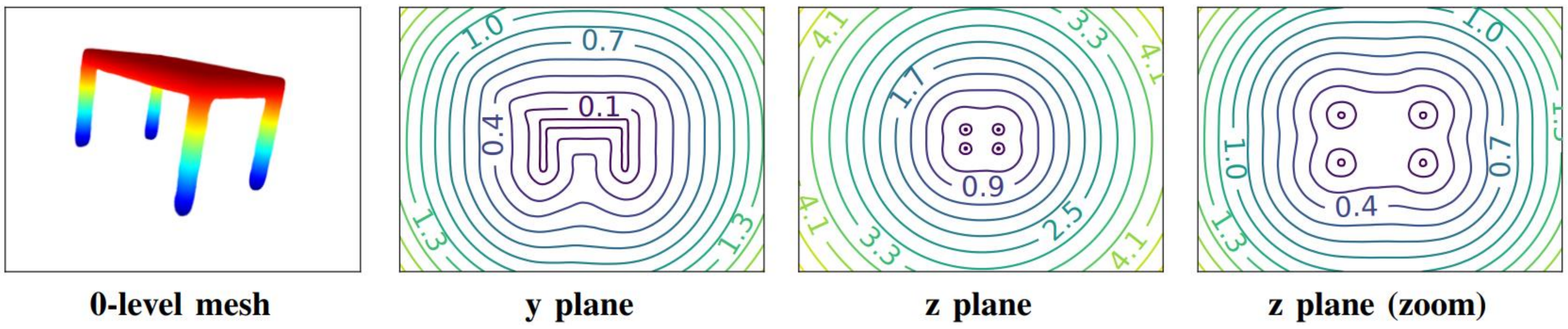
(self)-collision avoidance, joint limit, target ee-pose, etc.

Gradient is okay but...

Trajectory gradients are costly, especially in vectorization settings!

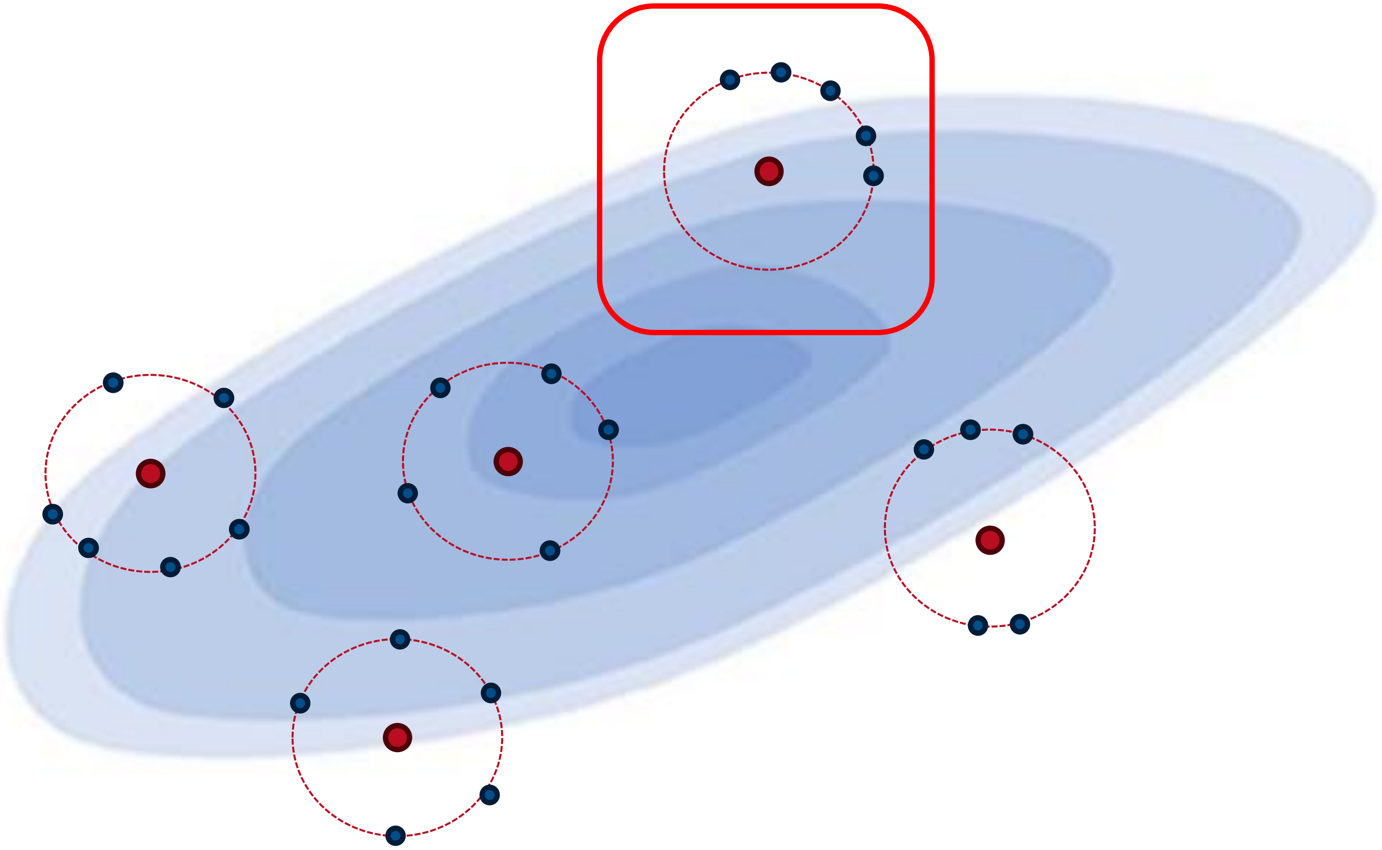
- Need to make sure all costs are differentiable, e.g., obstacle signed distant field

- Dynamics function is also needed to be differentiable



Liu, P.; Zhang, K.;Tateo D.; Jauhri S.; Peters J.; Chalvatzaki G.; (2022). Regularized Deep Signed Distance Fields for Reactive Motion Generation, *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

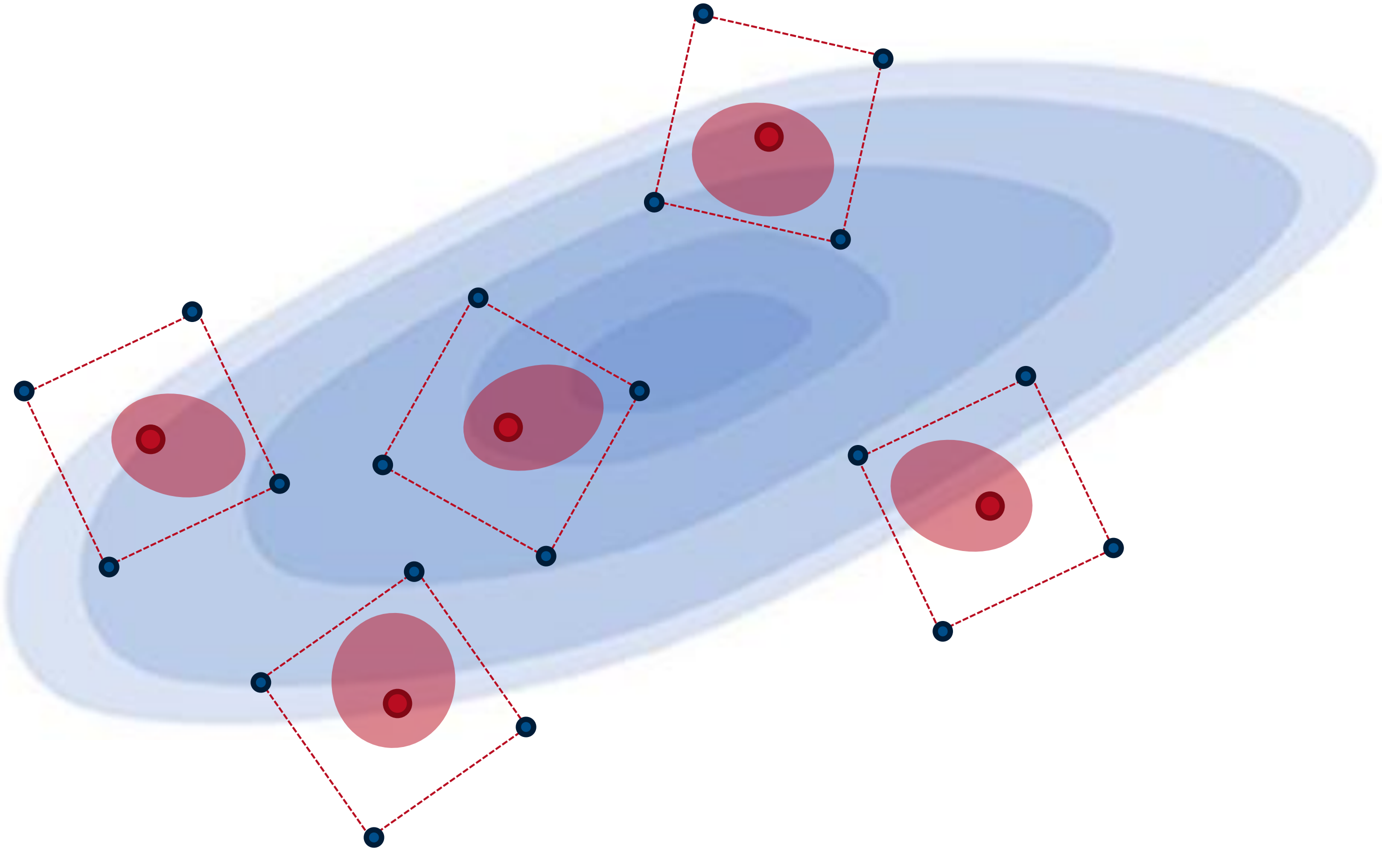
How can we solve trajectory optimization efficiently without gradients?



Requirements:

- **Batch update**
- **Fast**
- **No gradient access**

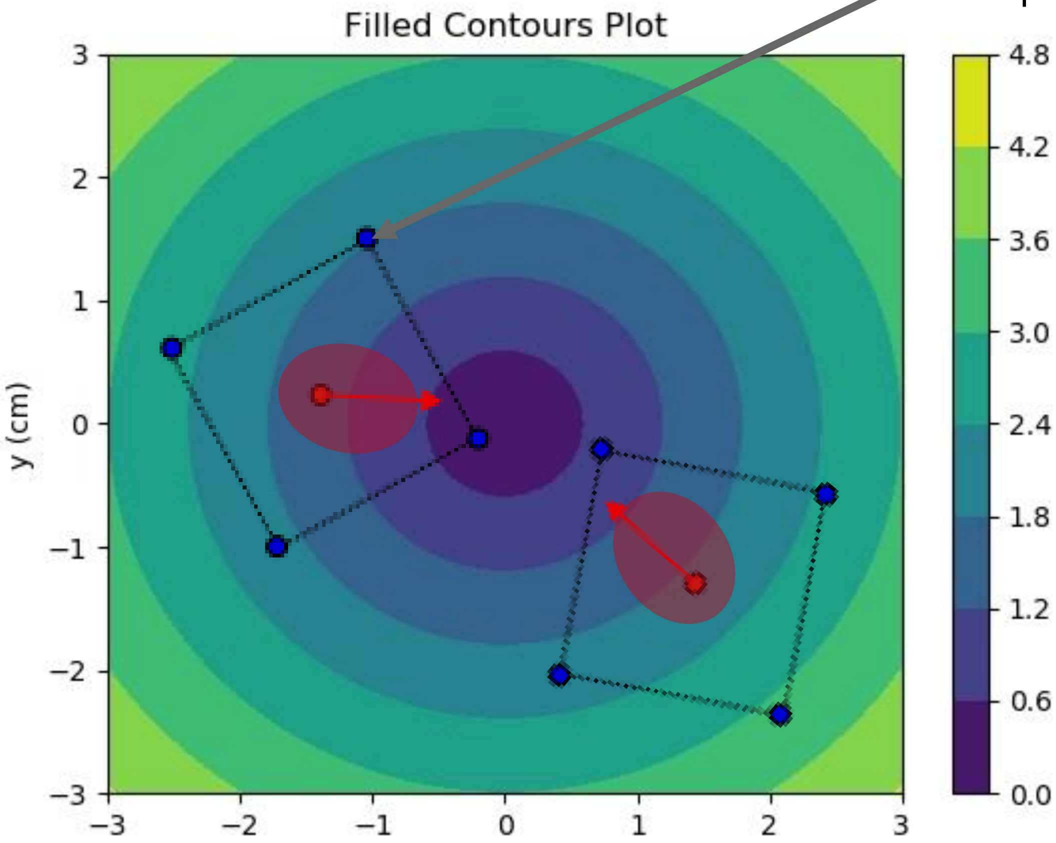
$$\min_X f(X) = \min_X \sum_{i=1}^{n} f(\boldsymbol{x}_i)$$



The regular polytopes are unbiased search direction sets!

Sinkhorn Step

Randomly Rotated Polytope Vertices as Step Direction Bases



n optimizing points, m vertices in polytope

$$\boldsymbol{n} \in \Sigma_n \qquad \boldsymbol{m} \in \Sigma_m$$

$$U(\boldsymbol{n}, \boldsymbol{m}) := \{\boldsymbol{W} \in \mathbb{R}_+^{n \times m} \mid \boldsymbol{W}\mathbf{1}_m = \boldsymbol{n}, \boldsymbol{W}^\intercal \mathbf{1}_n = \boldsymbol{m}\}$$

$$n \times m \ \text{cost matrix} \ \boldsymbol{C}$$

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \boldsymbol{S}_k, \ \boldsymbol{S}_k = \alpha_k \text{diag}(\boldsymbol{n})^{-1} \boldsymbol{W}_\lambda^* \boldsymbol{D}^P$$

$$\text{s.t. } \boldsymbol{W}_\lambda^* = \underset{\boldsymbol{W} \in U(\boldsymbol{n}, \boldsymbol{m})}{\text{argmin}} \ \langle \boldsymbol{W}, \boldsymbol{C} \rangle - \boxed{\lambda H(\boldsymbol{W})}$$

Step vectors (red) are the barycentric projection w.r.t. the polytope family!

Cuturi, Marco. "Sinkhorn distances: Lightspeed computation of optimal transport." *Advances in neural information processing systems* 26 (2013).
Peyré, Gabriel, and Marco Cuturi. "Computational optimal transport: With applications to data science." *Foundations and Trends® in Machine Learning* 11.5-6 (2019): 355-607.

Sinkhorn Step

Sinkhorn-Knopp algorithm

$$\mathrm{OT}_\lambda(\boldsymbol{n}, \boldsymbol{m}) := \min_{\boldsymbol{W} \in U(\boldsymbol{n}, \boldsymbol{m})} \langle \boldsymbol{W}, \boldsymbol{C} \rangle - \lambda H(\boldsymbol{W})$$

$$\boldsymbol{P} = \exp(-\boldsymbol{C}/\lambda) \qquad \boldsymbol{v}^0 = \mathbf{1}_n$$

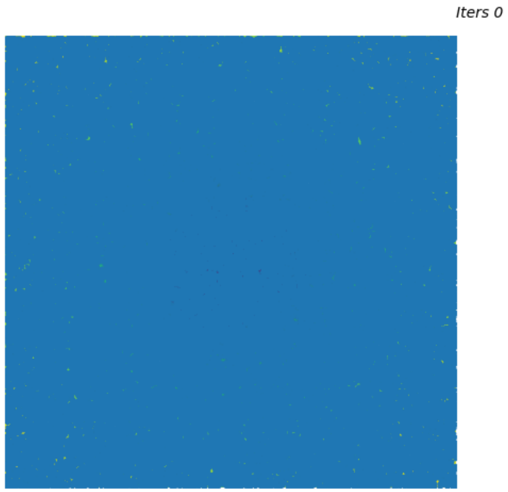
Until convergence: $\boldsymbol{u}^{i+1} = \frac{\boldsymbol{n}}{\boldsymbol{P}\boldsymbol{v}^i}, \quad \boldsymbol{v}^{i+1} = \frac{\boldsymbol{m}}{\boldsymbol{P}^\intercal \boldsymbol{u}^{i+1}},$

$$\boldsymbol{W}_\lambda^* = \mathrm{diag}(\boldsymbol{u}^*)\boldsymbol{P}\mathrm{diag}(\boldsymbol{v}^*)$$

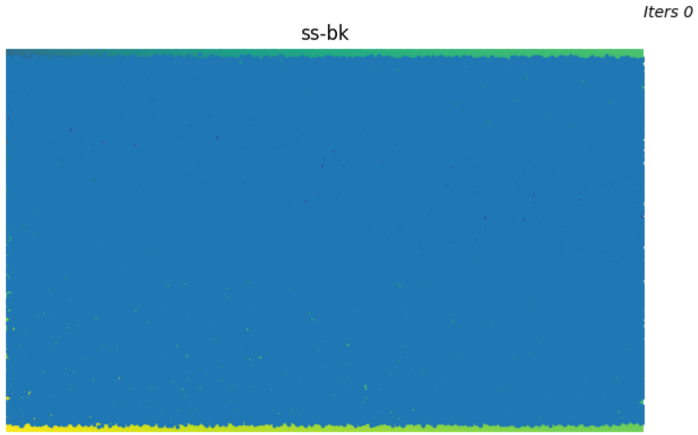
$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \boldsymbol{S}_k, \; \boldsymbol{S}_k = \alpha_k diag(\boldsymbol{n})^{-1} \boldsymbol{W}_\lambda^* \boldsymbol{D}^P$$

TECHNISCHE UNIVERSITÄT DARMSTADT

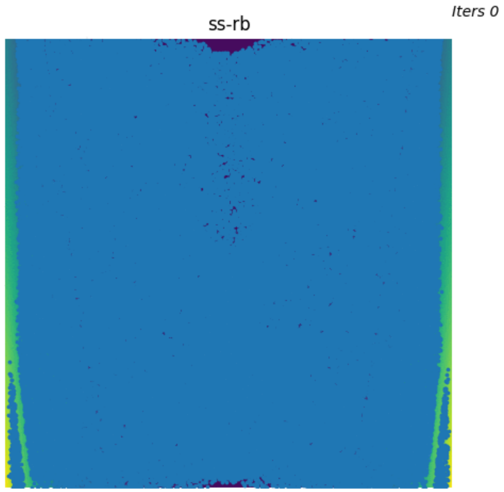
Code available at: https://github.com/anindex/ssax



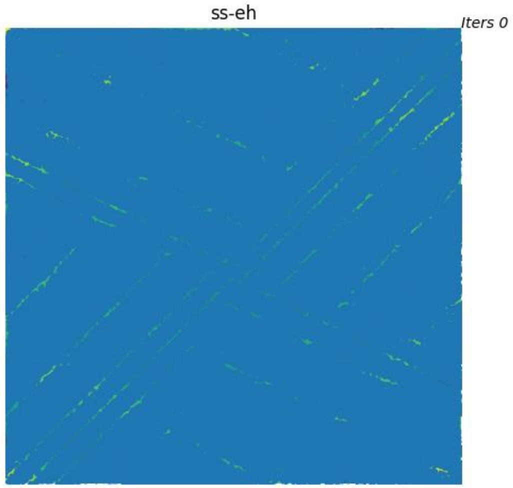
Ackley



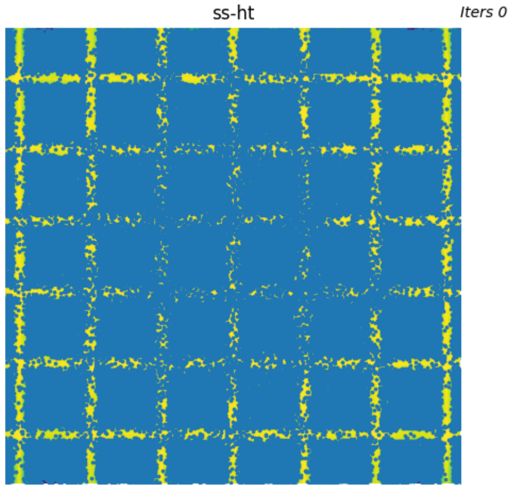
Bukin



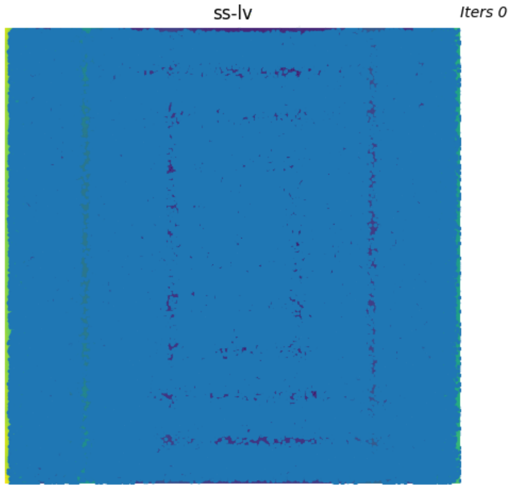
Rosenbrock



Egg Holder

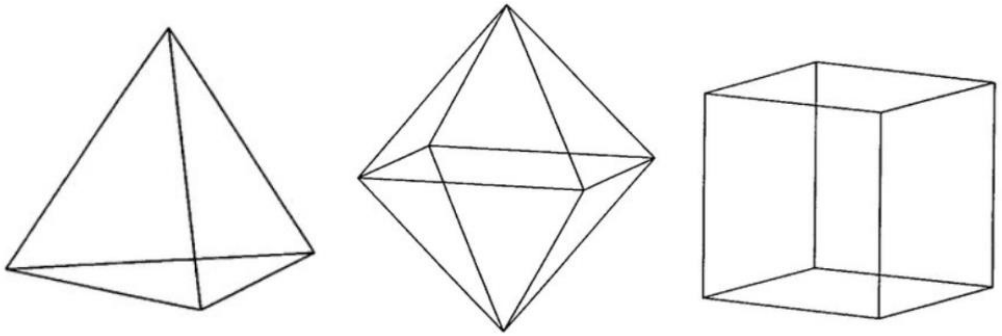


Holder Table



Levi

Uniform Polytope



	Parent	Truncated	Rectified	Bitruncated (tr. dual)	Birectified (dual)	Cantellated	Omnitruncated (Cantitruncated)	Snub
Tetrahedral 3-3-2	{3,3}	(3.6.6)	(3.3.3.3)	(3.6.6)	{3,3}	(3.4.3.4)	(4.6.6)	(3.3.3.3.3)
Octahedral 4-3-2	{4,3}	(3.8.8)	(3.4.3.4)	(4.6.6)	{3,4}	(3.4.4.4)	(4.6.8)	(3.3.3.3.4)
Icosahedral 5-3-2	{5,3}	(3.10.10)	(3.5.3.5)	(5.6.6)	{3,5}	(3.4.5.4)	(4.6.10)	(3.3.3.3.5)

https://en.wikipedia.org/wiki/Uniform_polytope

TECHNISCHE UNIVERSITÄT DARMSTADT

Now, applying Sinkhorn Step to trajectory optimization?

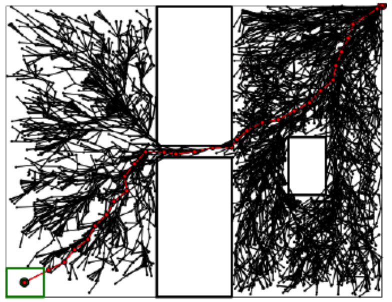
Problems?

Sampling-based algorithms

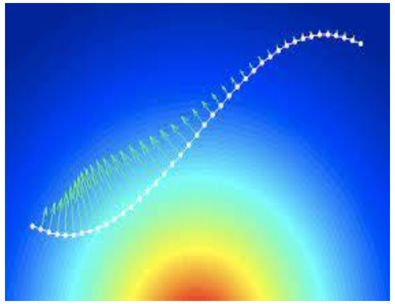
- Representatives: RRT*, PRM
- Completeness property
- Does not scale with state dimensions & objectives
- Require ad-hoc design to incorporate task objectives

Optimization-based algorithms

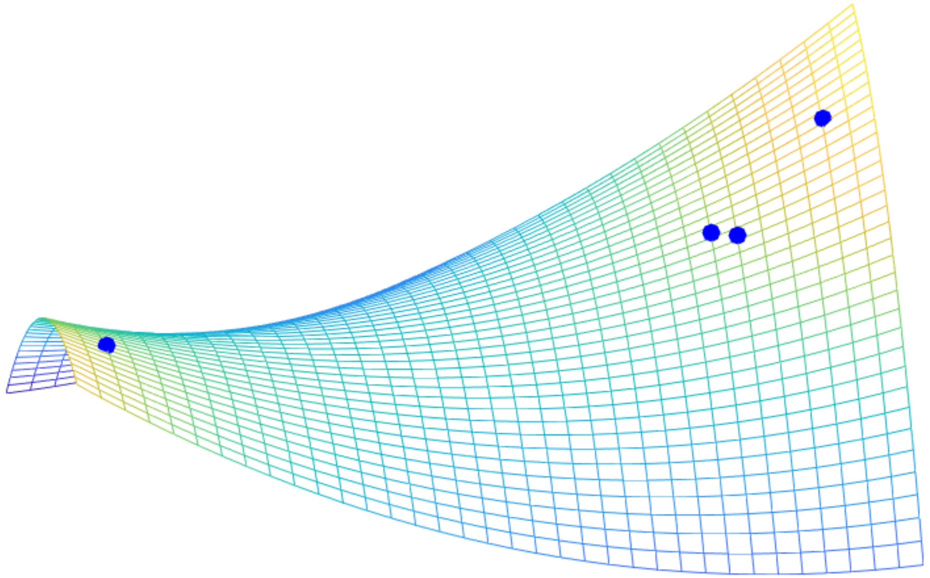
- Representatives: CHOMP, GPMP2, StochGPMP
- Does not guarantee to find solution
- Scale with state dimensions & objectives
- Easy to incorporate task objectives (as costs)

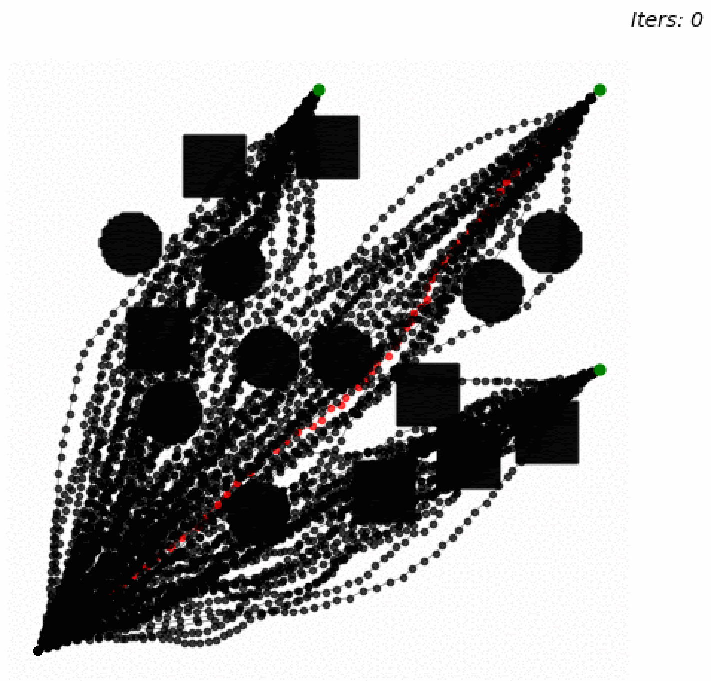


➢ **How can we approximate completeness while keeping scaling properties? ☺**



Anecdote: Go brute-force!





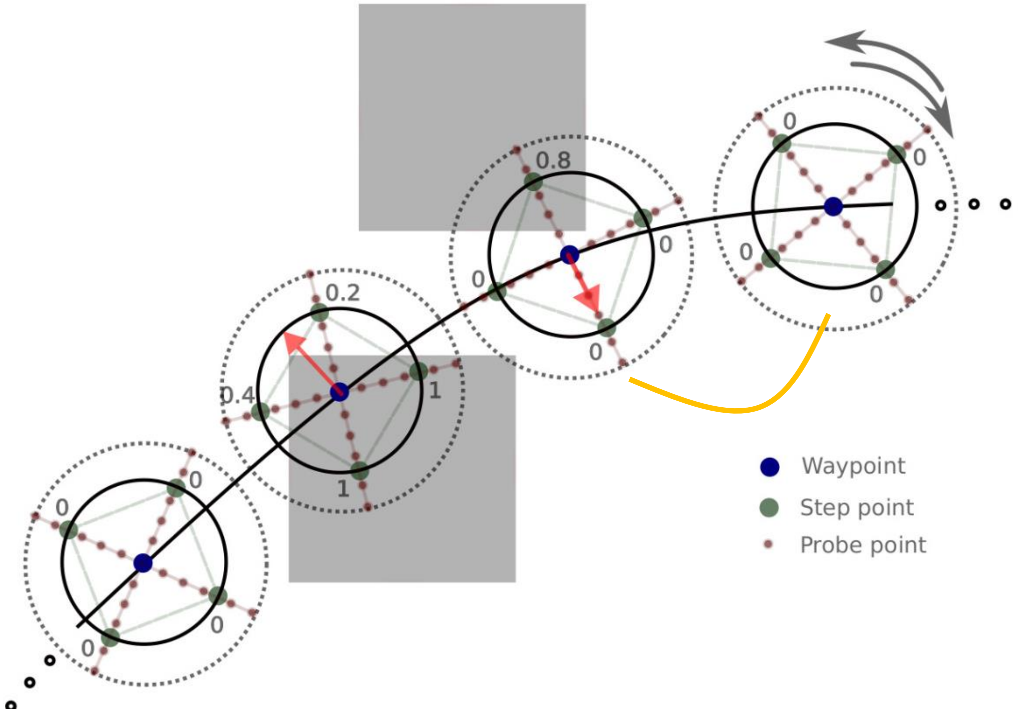
MPOT: Trajectory Optimization

$$\boldsymbol{\tau} = (X, U) = \{\boldsymbol{x}_t \in \mathbb{R}^d : \boldsymbol{x}_t = [\mathbf{x}_t, \dot{\mathbf{x}}_t]\}_{t=0}^T$$

Model constraint as cost (come from GP prior)

$$\boldsymbol{\tau}^* = \operatorname*{argmin}_{\boldsymbol{\tau}} \sum_{t=0}^{T-1} \underbrace{\eta C(\boldsymbol{x}_t)}_{\text{state cost}} + \underbrace{\frac{1}{2}\|\boldsymbol{\Phi}_{t,t+1}\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\|^2_{\mathbf{Q}^{-1}_{t,t+1}}}_{\text{transition model cost}}$$

MPOT: Procedure



1. **Construct uniform polytopes with current waypoints as their centers**

$$\boldsymbol{D}^P \in \mathbb{R}^{T \times m \times d}$$

2. **Populate probing points towards the polytope vertices**

$$\boldsymbol{H}^P \in \mathbb{R}^{T \times m \times h \times d}$$

3. **Compute local cost matrix**

$$\boldsymbol{C}_{t,i} = \frac{1}{h} \sum_{j=1}^{h} \eta\, c(\boldsymbol{x}_t + \boldsymbol{y}_{t,i,j}) + \frac{1}{2} \| \boldsymbol{\Phi}_{t,t+1} \boldsymbol{x}_t - (\boldsymbol{x}_{t+1} + \boldsymbol{y}_{t+1,i,j}) \|^2_{\boldsymbol{Q}^{-1}_{t,t+1}}$$

Probe points: $\boldsymbol{y}_{t,i,j} \in H^P$

4. **Do Sinkhorn Step!**

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \boldsymbol{S}_k,\ \boldsymbol{S}_k = \alpha_k \mathrm{diag}(\boldsymbol{n})^{-1} \boldsymbol{W}^*_\lambda \boldsymbol{D}^P$$

$$\text{s.t. } \boldsymbol{W}^*_\lambda = \underset{\boldsymbol{W} \in U(\boldsymbol{n}, \boldsymbol{m})}{\operatorname{argmin}} \langle \boldsymbol{W}, \boldsymbol{C} \rangle - \boxed{\lambda H(\boldsymbol{W})}$$

MPOT: Scale up!

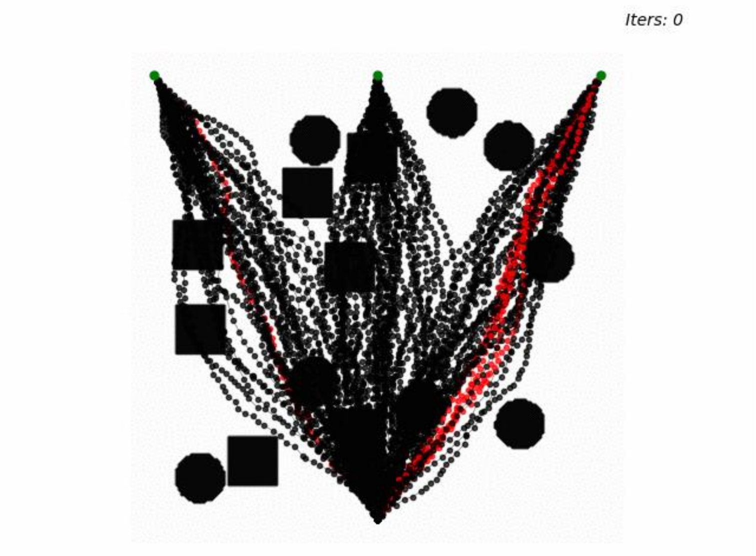
$$\mathcal{T} = \{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{N_p}\}$$

$$N = N_p \times T$$

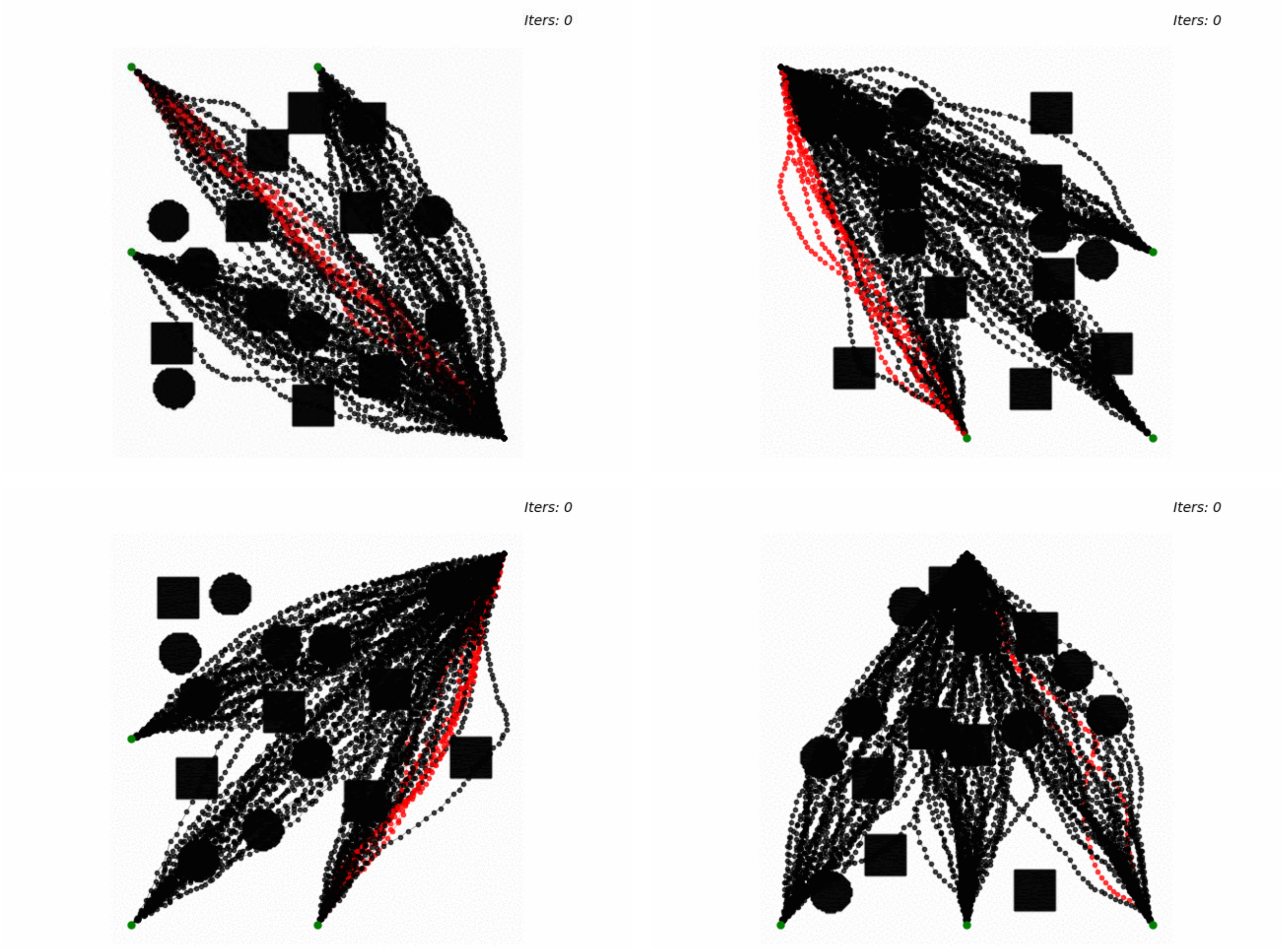
$$\boldsymbol{D}^P \in \mathbb{R}^{N \times m \times d}, \boldsymbol{H}^P \in \mathbb{R}^{N \times m \times h \times d}$$

Algorithm 1: Motion Planning via Optimal Transport

$\mathcal{T}^0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{K}_0)$ and $\boldsymbol{n} = \mathbf{1}_N/N$, $\boldsymbol{m} = \mathbf{1}_m/m$
while *termination criteria not met* **do**
  (Optional) $\alpha \leftarrow (1-\epsilon)\alpha$, $\beta \leftarrow (1-\epsilon)\beta$ `// Epsilon Annealing for Sinkhorn Step`
  Construct randomly rotated D^P, H^P and compute the cost matrix $\boldsymbol{C}$ as in Eq. (10)
  Perform Sinkhorn Step $\mathcal{T} \leftarrow \mathcal{T} + \mathbf{S}$
end

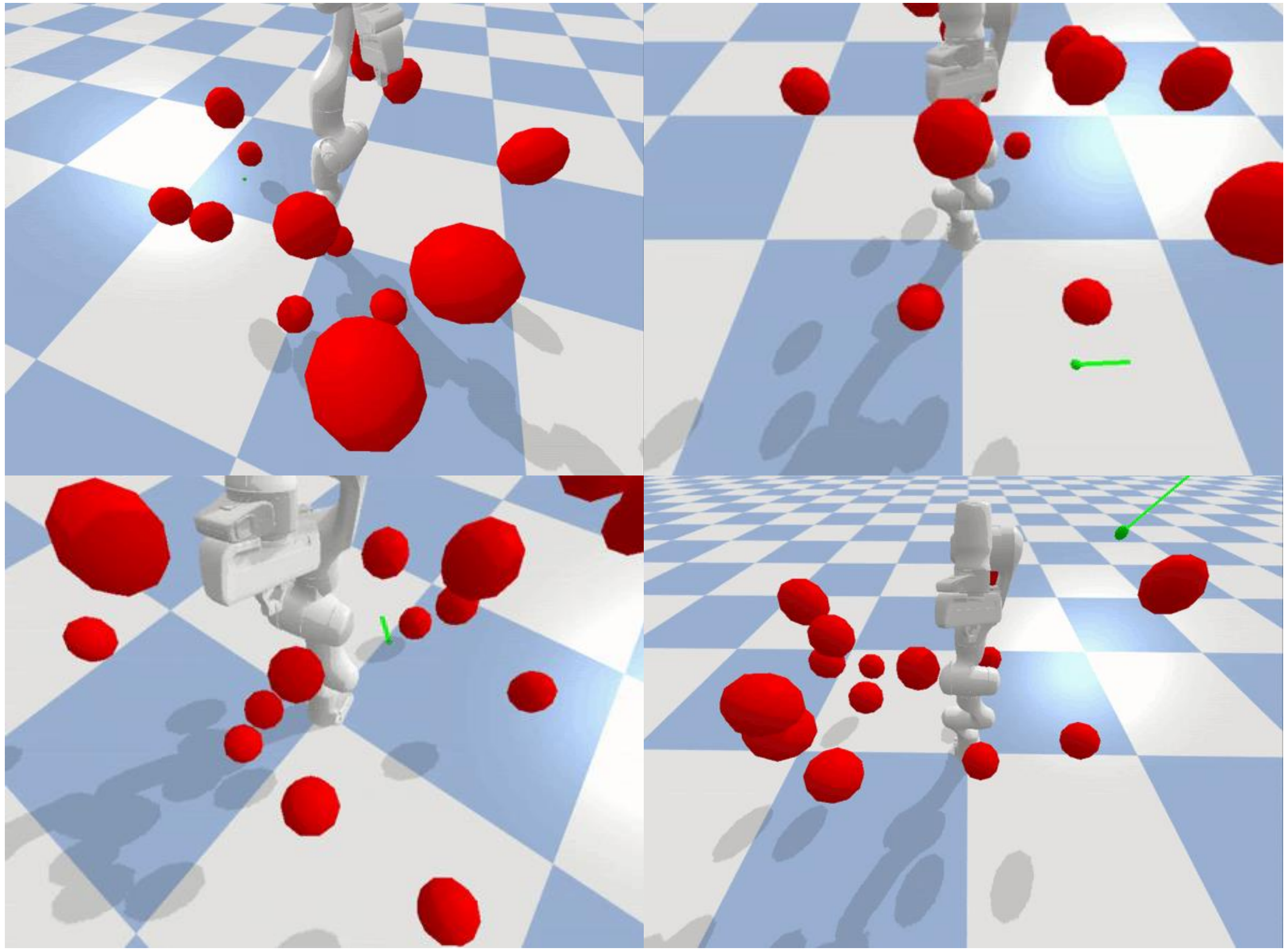


MPOT: Benchmark



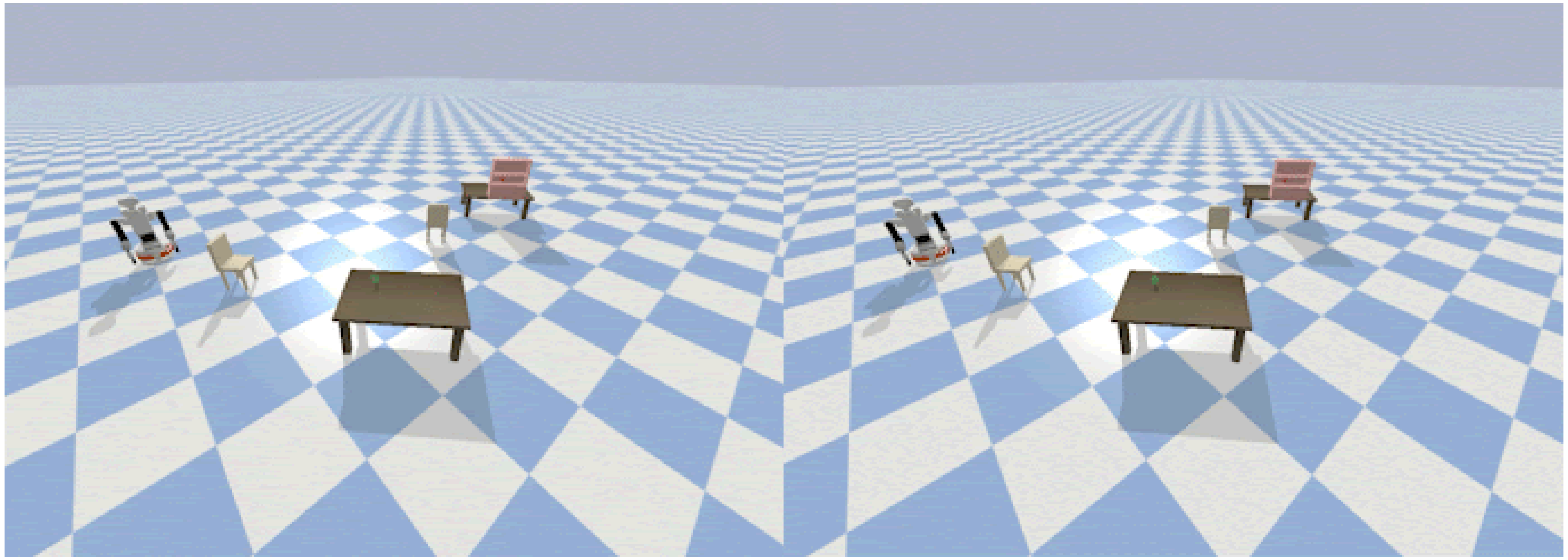
Planar navigation (4 dim): 99 plans in parallel took ~0.1-0.2 seconds with ~99% success rate on RTX3080Ti GPU (PyTorch).

MPOT: Benchmark



Panda case (14 dims): 10 plans in parallel took ~0.5-0.7 seconds with 71% success rate on RTX3080Ti GPU (PyTorch).

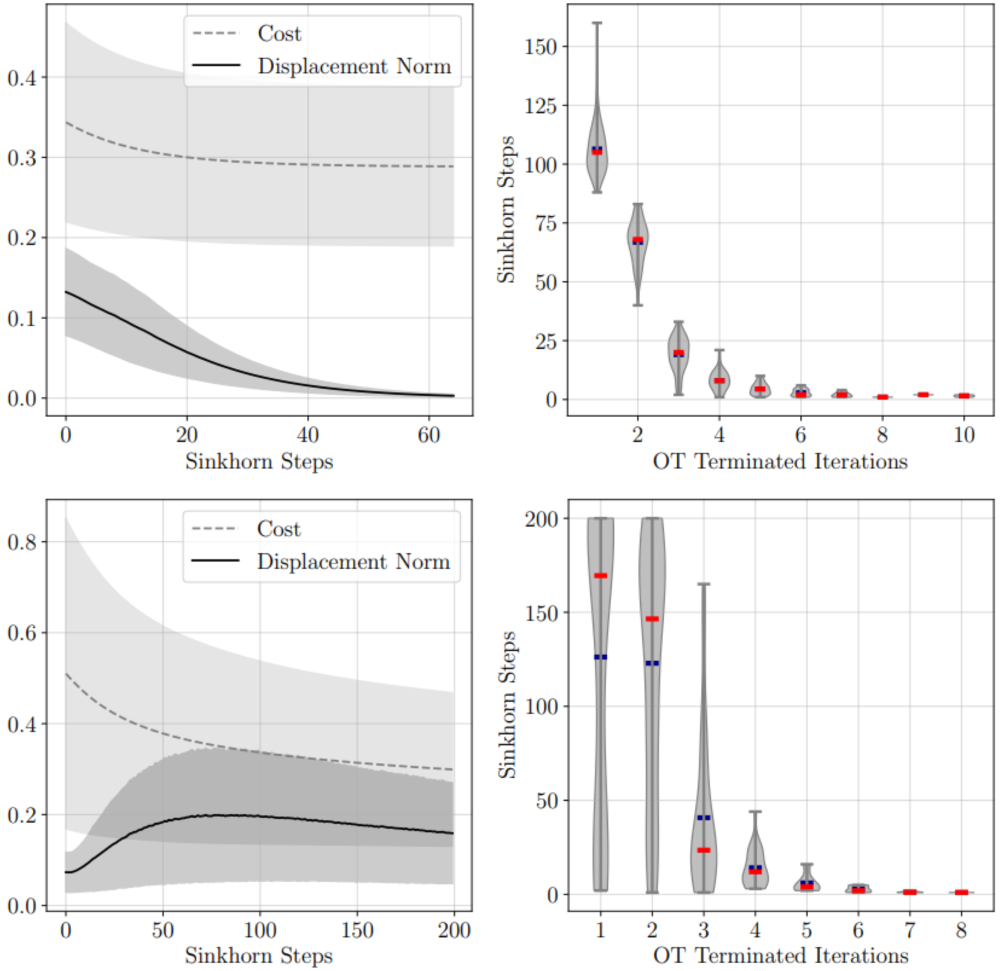
MPOT: Benchmark



	TF[s]	SUC[%]	S	PL
RRT*	1000 ± 0.00	0	-	-
I-RRT*	1000 ± 0.00	0	-	-
STOMP	-	0	-	-
SGPMP	27.75 ± 0.29	25	**0.010** ± 0.001	**6.69** ± 0.38
CHOMP	16.74 ± 0.21	40	0.015 ± 0.001	8.60 ± 0.73
GPMP2	40.11 ± 0.08	40	0.012 ± 0.015	8.63 ± 0.53
MPOT	**1.49** ± 0.02	**55**	0.022 ± 0.003	10.53 ± 0.62

This mobile manipulation case has the state space with 36 dimensions!

MPOT: Benchmark



Key takeaways

Sinkhorn Step is practically powerful but needs more theoretical understanding.

- No gradients are required anywhere!
- Surprisingly scalability and parallelization capability in batch planning!
- Many plans ~ more chance to get better modes!

Peoples

An T. Le

Georgia Chalvatzaki

Armin Biess

Jan Peters

Project website:
https://sites.google.com/view/sinkhorn-step/
Email: an@robot-learning.de
My website: anthaile.com

I am actively working on Optimal Transport methods applying for robotics problems.
Feel free to contact me to hear ranting about Optimal Transport in Robotics ☺