Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization

John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow and Pieter Abbeel

Pranay Pasula, Eugene Vinitsky

~~Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization~~

The world's fastest introduction to penalty methods and collision checking

Pranay Pasula, Eugene Vinitsky

Problem

Problem Statement

- Given object A and obstacles $\{O_i\}$
- Find a trajectory ξ that takes object A from q_{start} to q_{goal} with
  - Minimal computation time
  - Minimal likelihood of collision
  - Minimal cost

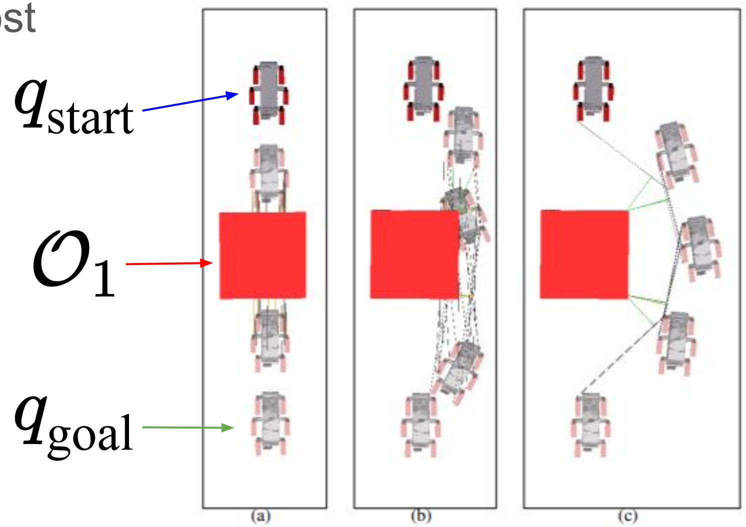


Figure from Xanthidis et al. 2019 “Navigation in the Presence of…”

Key Insights

- Mesh methods are really effective
- Convex-convex collision checking is “fast”
- Convex collision checking can be turned into a constraint
- Trust region methods can be used to efficiently solve constrained optimization problems

$$\phi(\xi;\mu) = \min_\xi f(\xi) + \mu \sum_i |g_i(\xi)|^+ + \mu \sum_j |h_j(\xi)|$$

Motion Objective

Collision constraints

Progress

- Fast signed distance checking
- Collision constraints
- Solving the constrained problem

Overview: Collision avoidance

Discrete-time

- Signed distance (SD) between objects
- SD constraints and penalty formulation
- Linearizing SD to enable optimization

Continuous-time

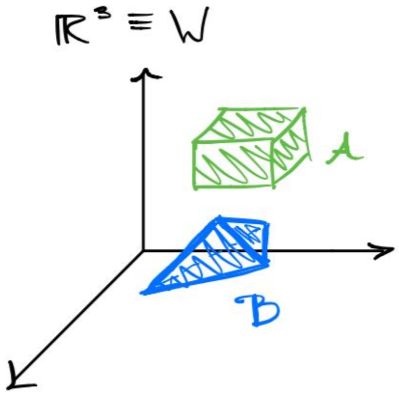
- Case 1: Translation only
- Case 2: Translation + rotation

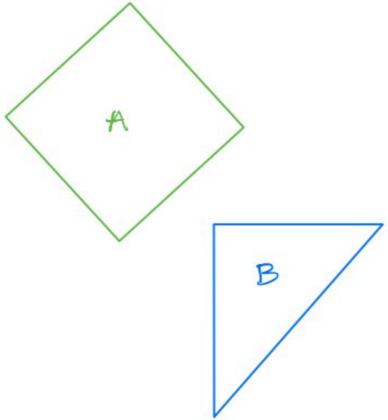
Preliminaries

A, B, O are labels for rigid objects/obstacles

$\mathcal{A}, \mathcal{B}, \mathcal{O} \subset \mathbb{R}^3$ are sets of points occupied by A, B, O

$W = \mathbb{R}^3$

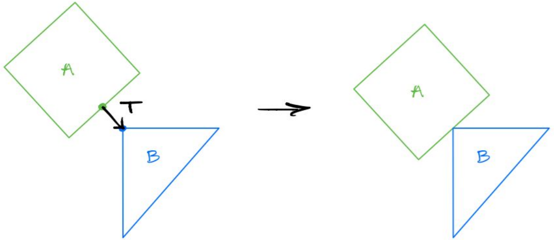




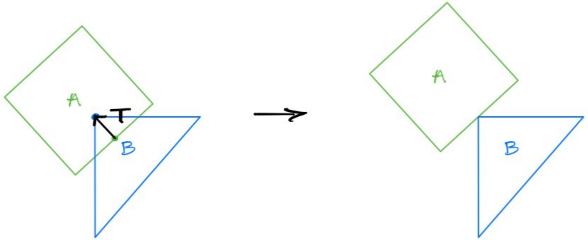
Penalizing collisions ideally

Collision penalty based on **minimum translation distance** $||T||$ between objects

$$\text{dist}(\mathcal{A}, \mathcal{B}) = \inf\{||T|| \mid (\mathcal{A} + T) \cap \mathcal{B} \neq \emptyset\}$$



$$\text{penetration}(\mathcal{A}, \mathcal{B}) = \inf\{||T|| \mid (\mathcal{A} + T) \cap \mathcal{B} = \emptyset\}$$



Penalizing collisions (kind of) practically

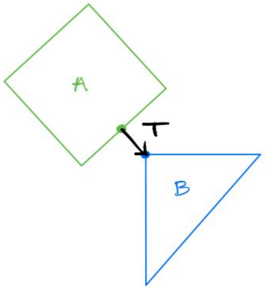
$$\mathrm{sd}(\mathcal{A},\mathcal{B}) = \mathrm{dist}(\mathcal{A},\mathcal{B}) - \mathrm{penetration}(\mathcal{A},\mathcal{B})$$

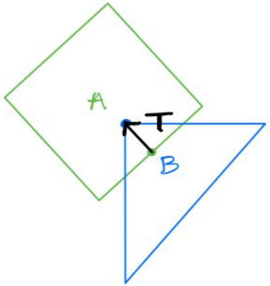
No collision: $\mathrm{sd}(\mathcal{A},\mathcal{B}) > 0$

Collision: $\mathrm{sd}(\mathcal{A},\mathcal{B}) < 0$

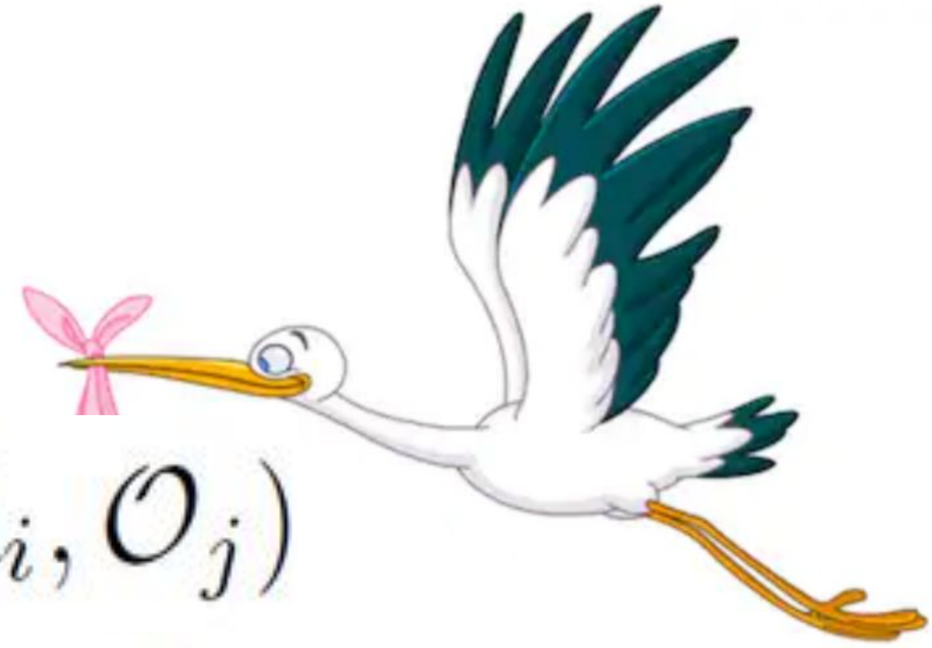
→ No room for error!

↓ Introduce safety margin $d_{\mathrm{safe}} > 0$





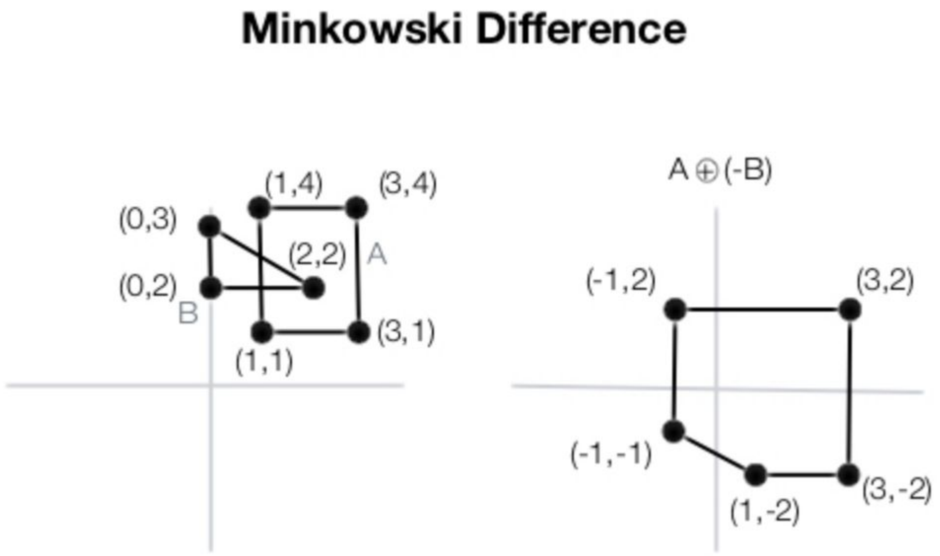
But where do signed distances come from?



$$\mathrm{sd}(\mathcal{A}_i, \mathcal{O}_j)$$

GJK

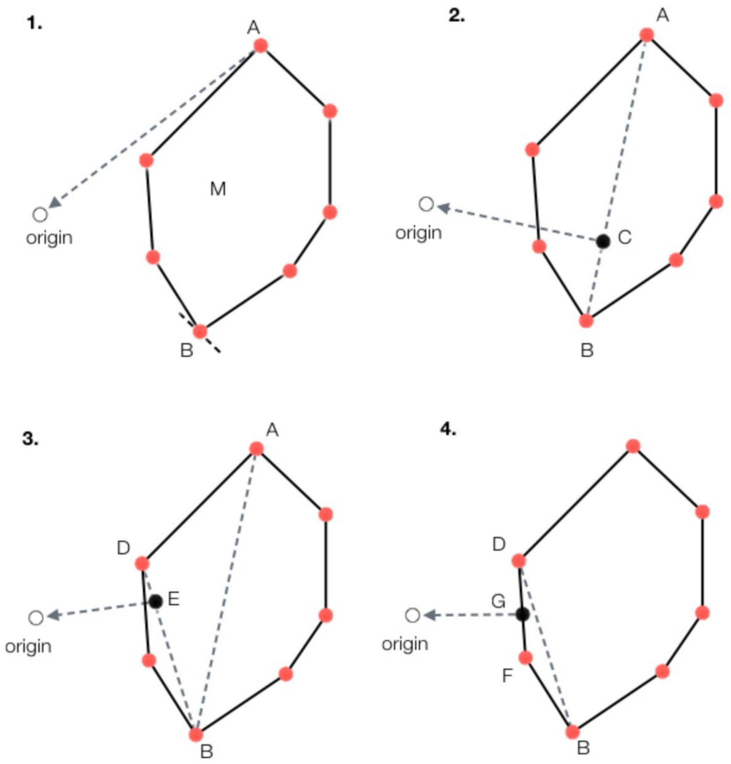
Two objects intersect if their Minkowski difference contains the origin



Source: “Real Time Collision Detection”

GJK basic idea: simple

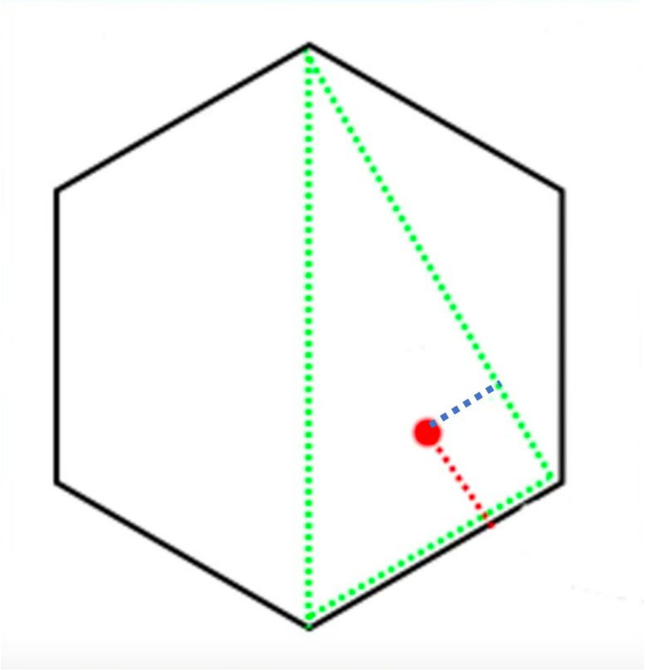
- Form Minkowski difference
- Iteratively find closest points
- Either gives closure or distance
- **NOT THE FASTEST WAY**



Source: “Real Time Collision Detection”

Expanding Polytope Method

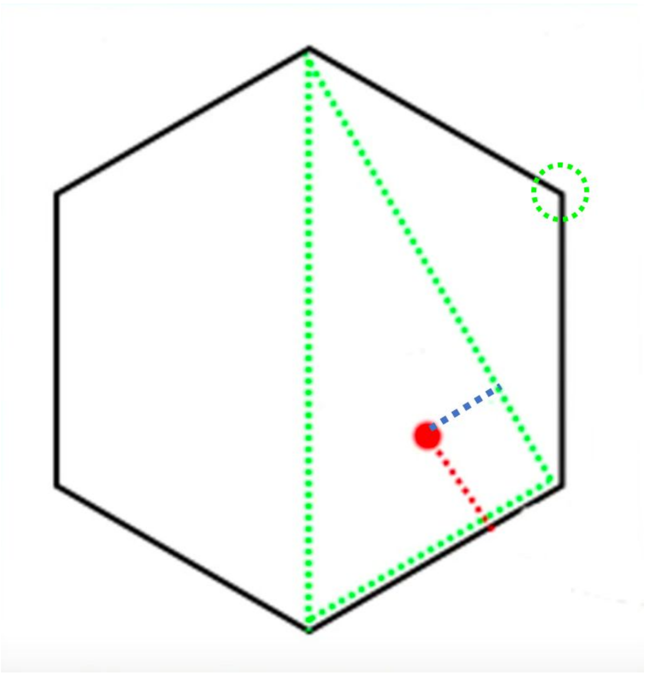
- Pick a polytope
- **Find closest point in polytope**
- If point on edge, done
- Else, expand polytope to include support vector



Source: https://www.youtube.com/watch?v=6rgiPrzqt9w

Expanding Polytope Method

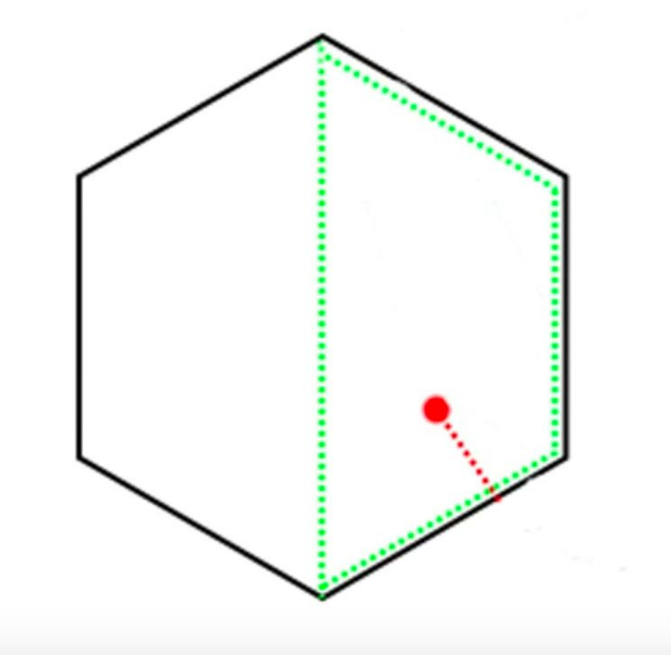
- Pick a polytope
- Find closest point in polytope
- **If point on edge, done**
- Else, expand polytope to include support vector



Source:
https://www.youtube.com/watch?v=6rgiPrzqt9w

Expanding Polytope Method

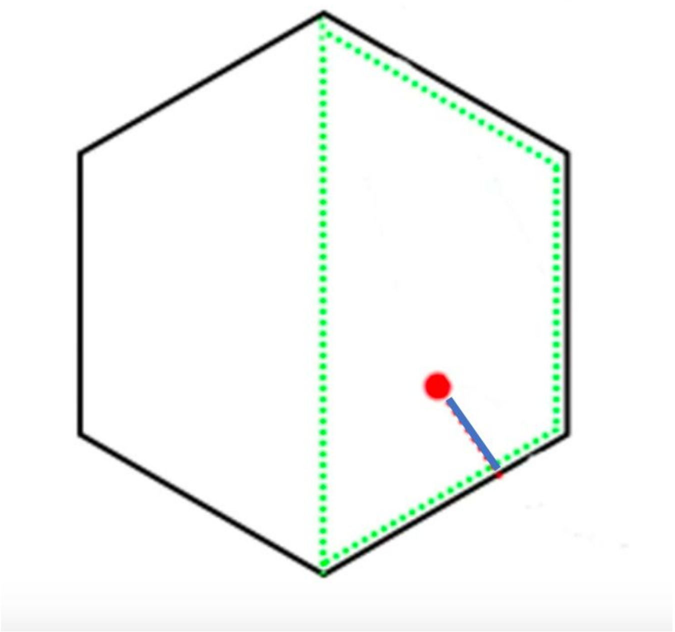
- Pick a polytope
- Find closest point in polytope
- If point on edge, done
- **Else, expand polytope to include support vector**



Source:
https://www.youtube.com/watch?v=6rgiPrzqt9w

Expanding Polytope Method

- Pick a polytope
- **Find closest point in polytope**
- **If point on edge, done**
- Else, expand polytope to include support vector



Source:
https://www.youtube.com/watch?v=6rgiPrzqt9w

Signed distance constraints and reformulation

$\{\mathcal{A}_i\}$: set of robot links

$\{\mathcal{O}_j\}$: set of obstacles

What do we want?

$$\mathrm{sd}(\mathcal{A}_i, \mathcal{O}_j) \geq d_{\mathrm{safe}}$$
$$\forall i \in \{1, 2, \ldots, N_{\mathrm{links}}\}$$
$$\forall j \in \{1, 2, \ldots, N_{\mathrm{obstacles}}\}$$

$$\mathrm{sd}(\mathcal{A}_i, \mathcal{A}_j) \geq d_{\mathrm{safe}}$$
$$\forall i, j \in \{1, 2, \ldots, N_{\mathrm{links}}\},\ i \neq j$$

Reformulate for our method

$$\sum_{i=1}^{N_{\mathrm{links}}} \sum_{j=1}^{N_{\mathrm{obs}}} |d_{\mathrm{safe}} - \mathrm{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+$$
$$+ \sum_{i=1}^{N_{\mathrm{links}}} \sum_{j=1}^{N_{\mathrm{links}}} |d_{\mathrm{safe}} - \mathrm{sd}(\mathcal{A}_i, \mathcal{A}_j)|^+$$

Penalizing collisions practically

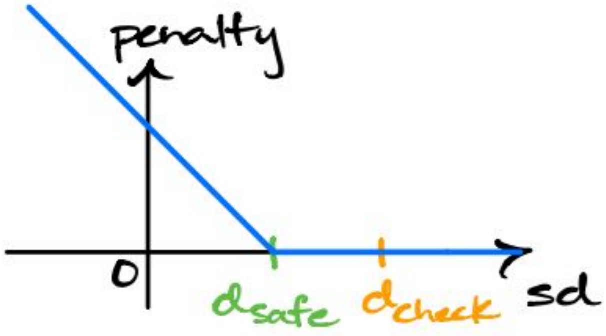
We have

$$\sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{obs}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+$$

$$+ \sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{links}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{A}_j)|^+$$

Enumerating over all combinations is prohibitive

Introduce $d_{\text{check}} > d_{\text{safe}}$



Compute cost only for pairs with $\text{sd}(\cdot, \cdot) < d_{\text{check}}$

Progress

- ~~Fast signed distance checking~~
- Collision constraints
- Solving the constrained problem

Making signed distance work for us

We have

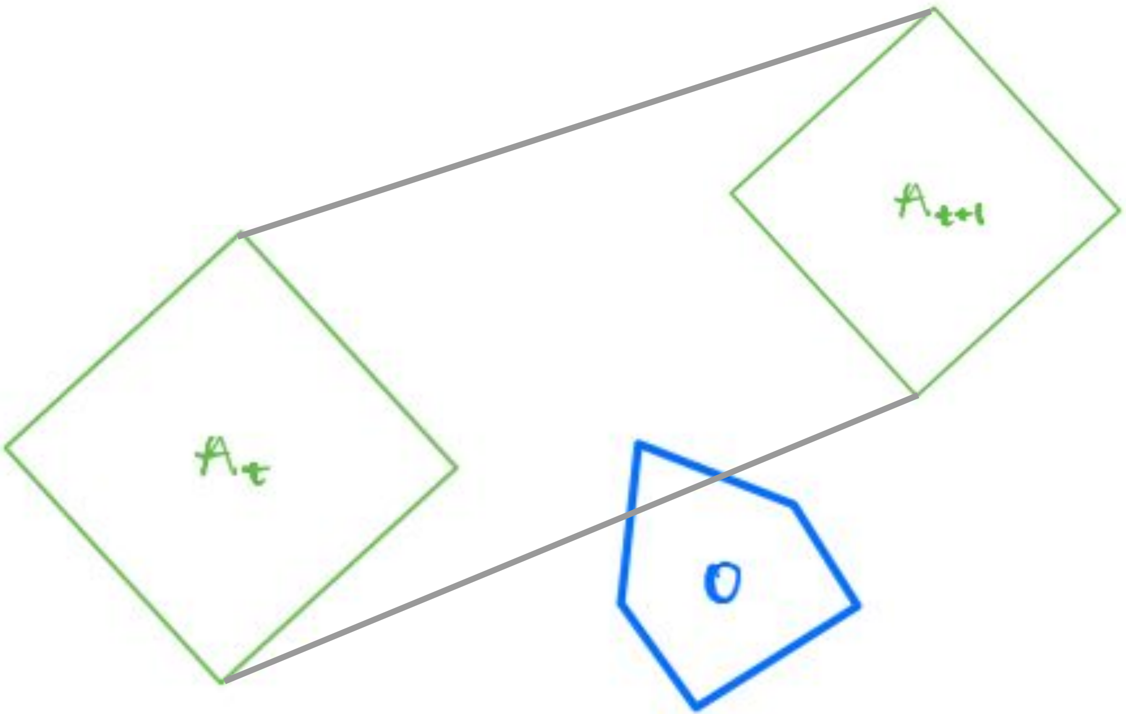
$$\sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{obs}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{O}_j)|^+$$

$$+ \sum_{i=1}^{N_{\text{links}}} \sum_{j=1}^{N_{\text{links}}} |d_{\text{safe}} - \text{sd}(\mathcal{A}_i, \mathcal{A}_j)|^+$$

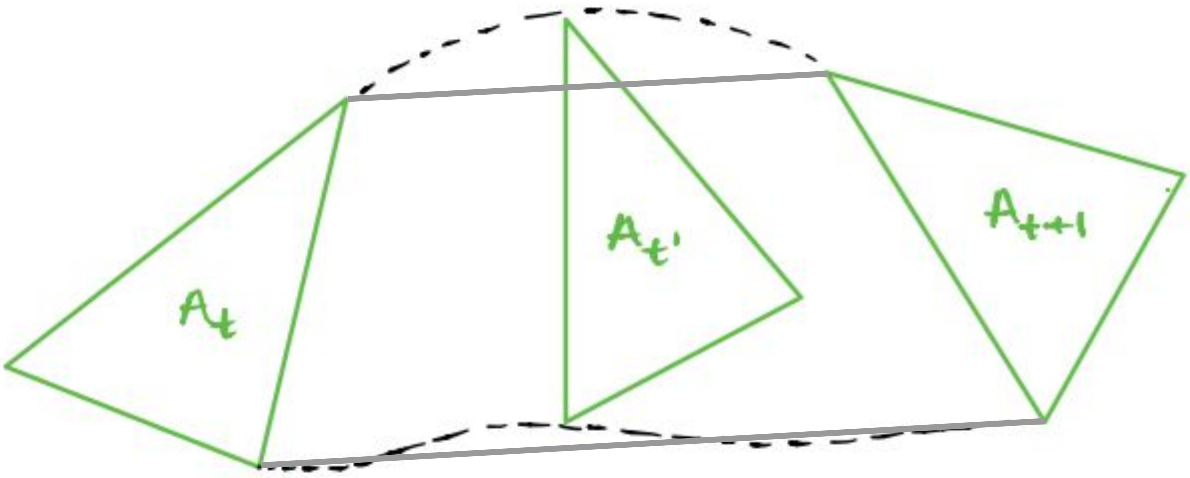
But $\text{sd}(\cdot, \cdot)$ is non-linear! ⇨

1. Reformulate $\text{sd}(\cdot, \cdot)$ as maximin problem
2. Approx with first-order Taylor expansion wrt q
3. Replace corresponding cost term with approx
4. Repeat for all pairs with distance $< d_{\text{check}}$

Continuous-Time Collision Avoidance (Translation only)



Continuous-Time Collision Avoidance (Translation + Rotation)



$$\mathrm{sd}(\mathrm{conv}(\mathcal{A}_t . \mathcal{A}_{t+1}), \mathcal{O}) > d_{\mathrm{safe}} + d_{\mathrm{arc}}$$

Recap: Collision avoidance

Discrete-time

- Signed distance (SD) between objects
- SD constraints and penalty formulation
- Linearizing SD to enable optimization

Continuous-time

- Case 1: Translation only
- Case 2: Translation + rotation

Progress

- ~~Fast signed distance checking~~
- ~~Collision constraint~~
- Solving the constrained problem

Penalty Optimization

- Start with a constrained problem

$$\min_x f(x)$$
$$g_i(x) \leq 0 \quad \forall i$$
$$h_j(x) = 0 \quad \forall j$$

- Move the constraints into the cost with a penalty coefficient

$$\phi(x; \mu) = \min_x f(x) + \mu \sum_i |g_i(x)|^+ + \mu \sum_j |h_j(x)|$$

- Optimize away

Penalty Optimization

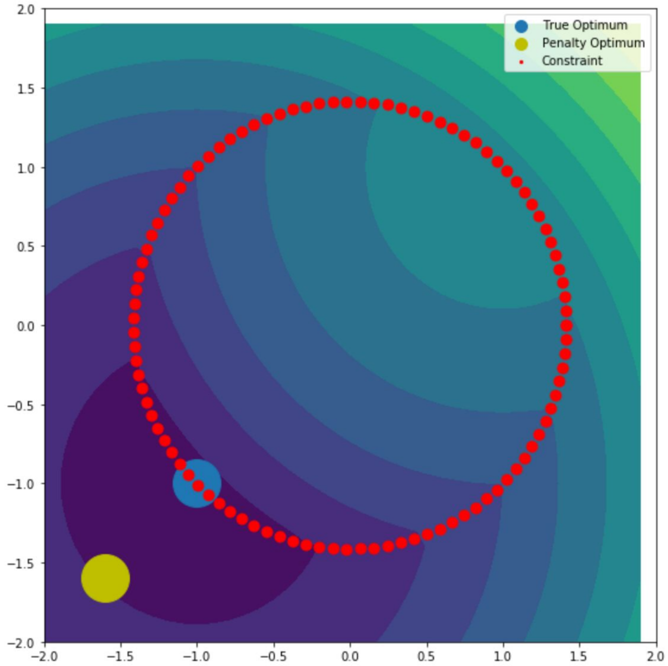
$$\phi(x;\mu) = \min_x f(x) + \mu \sum_i |g_i(x)|^+ + \mu \sum_j |h_j(x)|$$

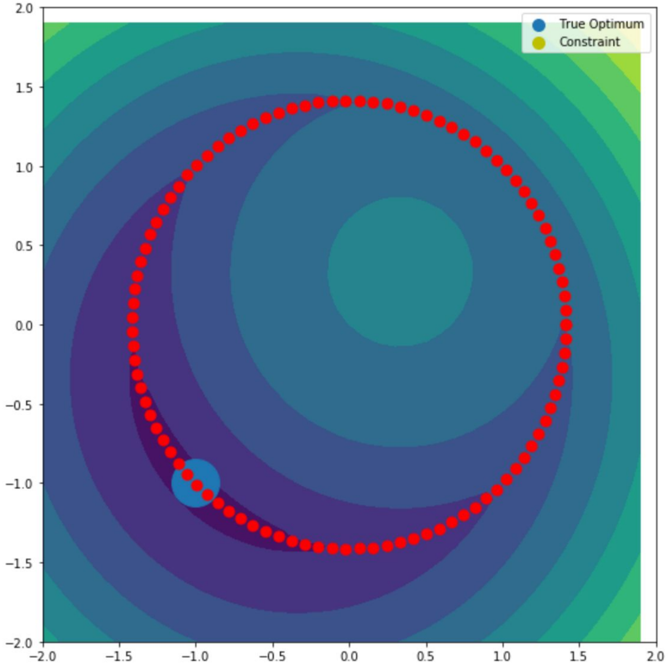
- But wait, won't this change the optimum?

Penalty Optimization

$\min x_1 + x_2 \quad \text{subject to } x_1^2 + x_2^2 - 2 = 0.$

But wait, won't this change the optimum? Yep!





Penalty Optimization

$$\phi(x;\mu) = \min_x f(x) + \mu \sum_i |g_i(x)|^+ + \mu \sum_j |h_j(x)|$$

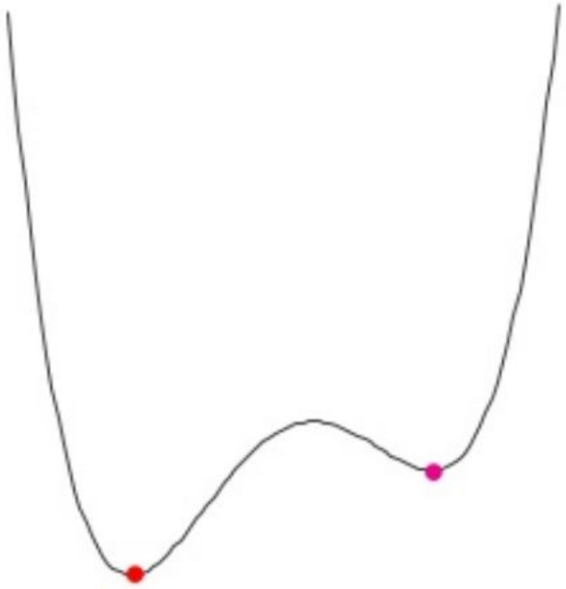
If you just make the penalty large enough, we'll find the constrained local minimum[1]

Note, this isn't necessarily true if we used quadratic penalties

Nocedal and Wright[1]

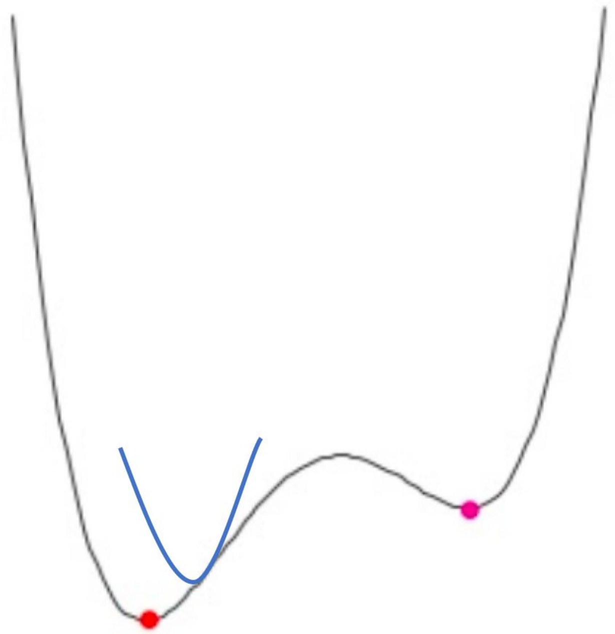
Sequential Quadratic Optimization w/ Trust Region

- Non-convex problem



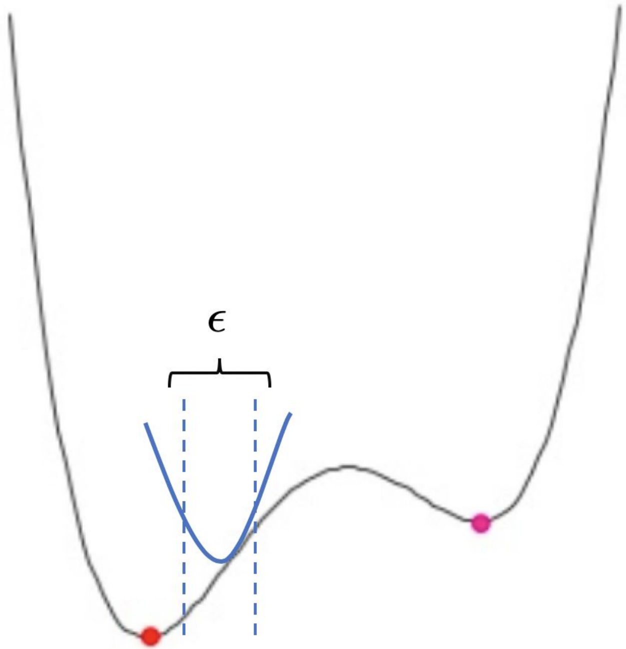
Sequential Quadratic Optimization w/ Trust Region

- Expand to second order around your point



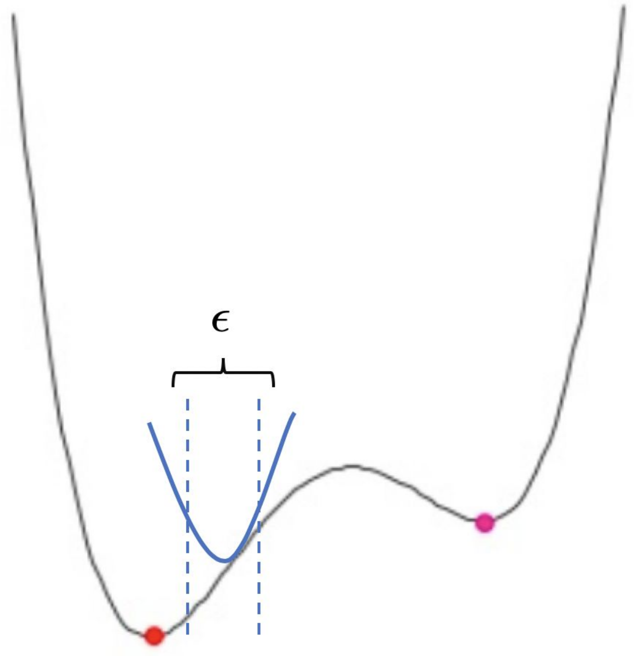
Sequential Quadratic Optimization w/ Trust Region

- Apply a trust region



Sequential Quadratic Optimization w/ Trust Region

- Apply a quadratic program solver like IPOPT



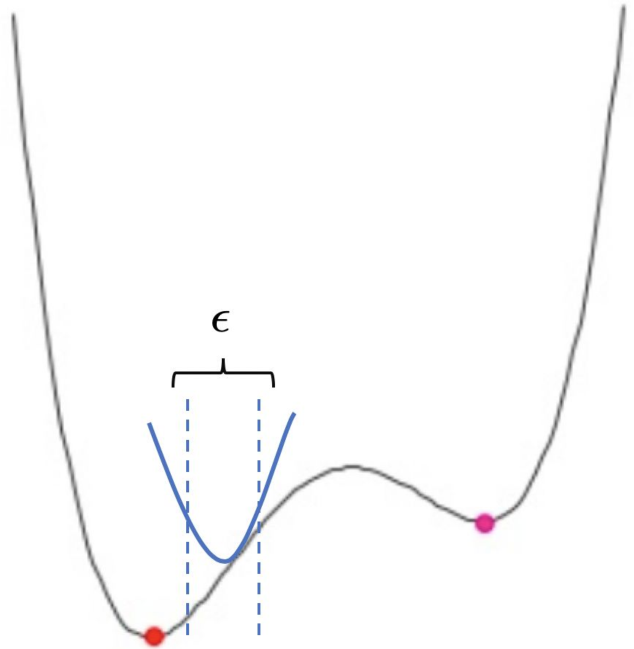
Trust Region Scaling

- Apply a quadratic program solver like IPOPT
- Improved on the true problem?

$$\epsilon \leftarrow c^{+}\epsilon,\ c > 1$$

- Didn't?

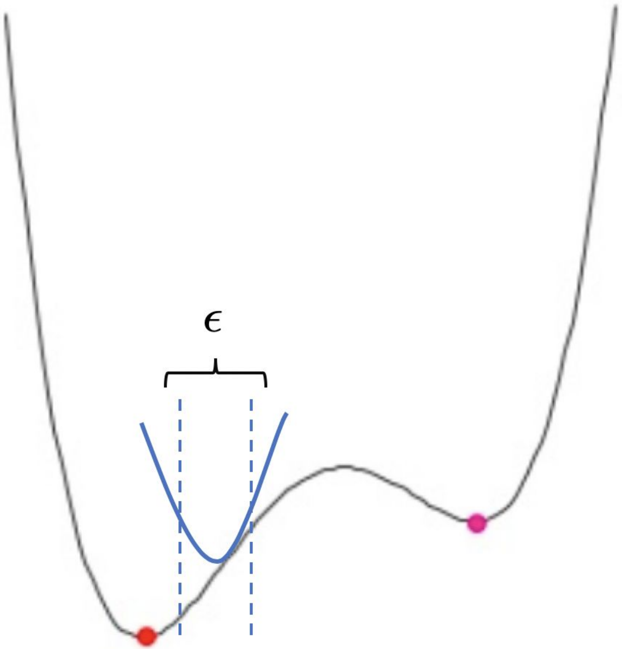
$$\epsilon \leftarrow c^{-}\epsilon,\ c < 1$$



Penalty Scaling

- Constraints unsatisfied?

$$\mu \leftarrow \kappa\mu, \ \ \kappa > 1$$



Progress

- ~~Fast signed distance checking~~
- ~~Collision constraints~~
- ~~Solving the constrained problem~~

What've we got

- A way to compute signed distance constraints
- A way to solve the optimization formula

	Trajopt	Trajopt-Multi	ompl-RRTConnect	ompl-LBKPIECE	CHOMP-HMC	CHOMP-HMC-Multi
success fraction	0.818	0.955	0.854	0.758	0.652	0.833
average time (s)	0.191	0.3	0.615	1.3	4.91	9.27
avg normed length	1.16	1.15	1.56	1.61	2.04	1.97

TABLE I
Results on 198 arm planning problems for a PR2, involving 7 degrees of freedom. Trajopt refers to our algorithm.

	Trajopt	Trajopt-multi	OMPL-RRTConnect	OMPL-LBKPIECE
success fraction	0.729	0.875	0.406	0.51
average time (s)	2.2	6.1	20.3	18.7
avg normed length	1.06	1.05	1.54	1.51

TABLE II
Results on 96 full-body planning problems for a PR2, involving 18 degrees of freedom (two arms, torso, and base).

CHOMP V. TrajOpt

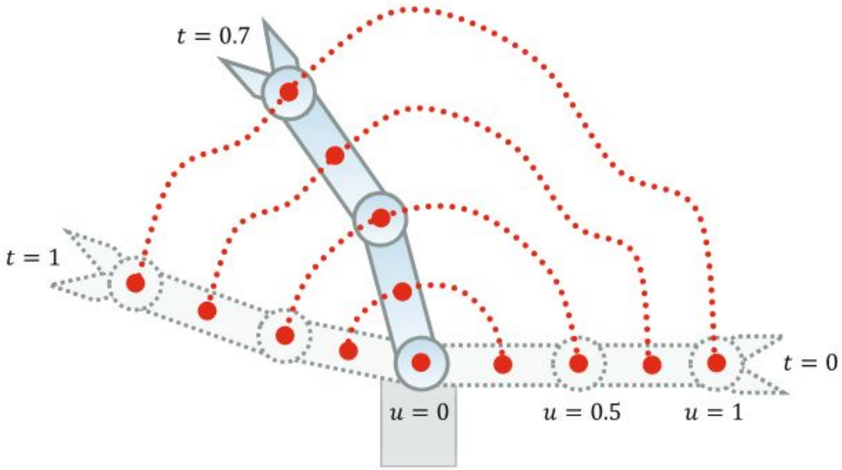
CHOMP

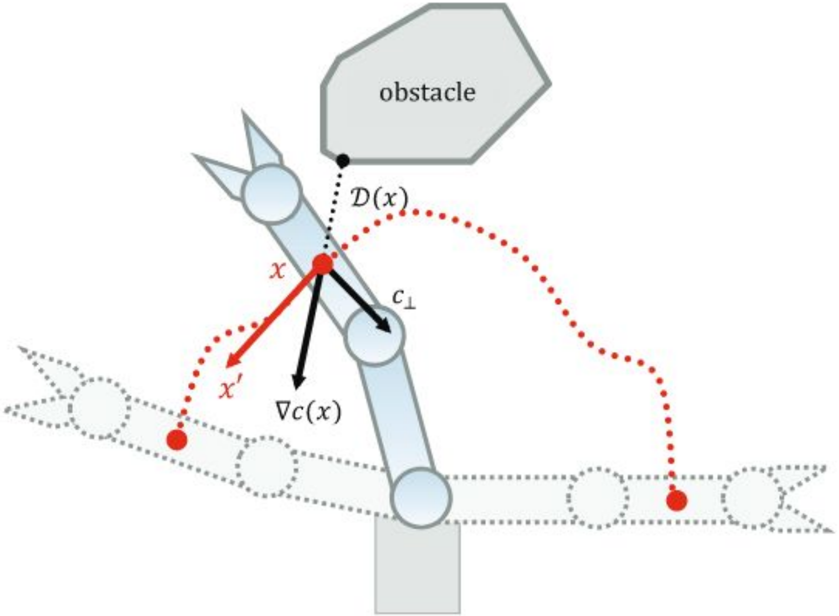
- Projected gradient descent
- Distance fields

TrajOpt

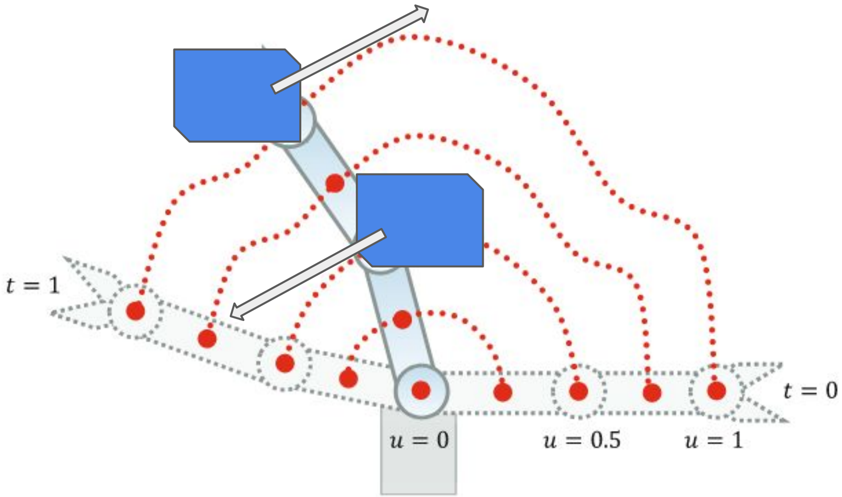
- Sequential Quadratic Programs
- Convex-Convex collision checking

What does trajopt help with? CHOMP



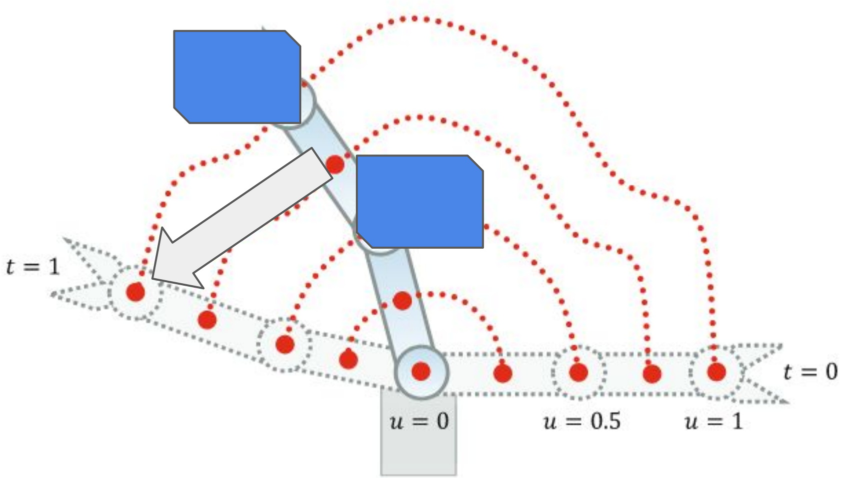


What does trajopt help with? CHOMP



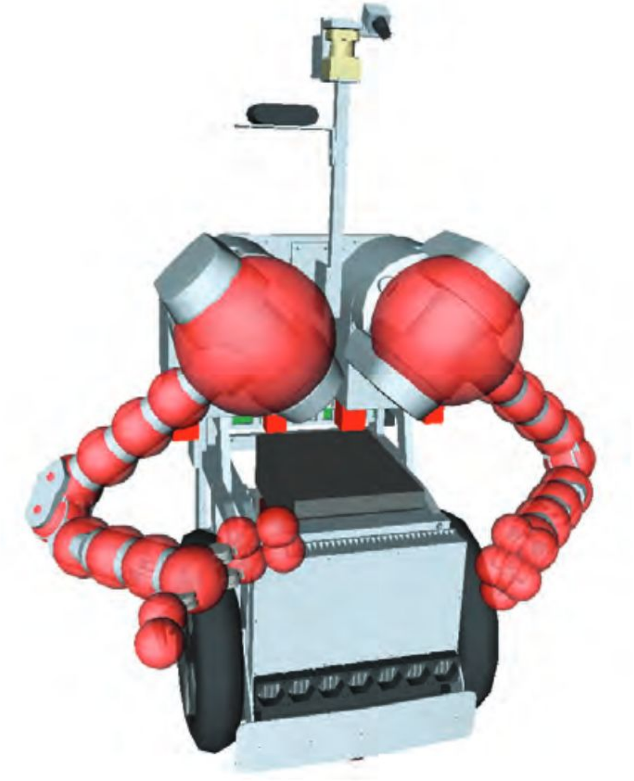
Conflicting costs on body points

What does trajopt help with? CHOMP



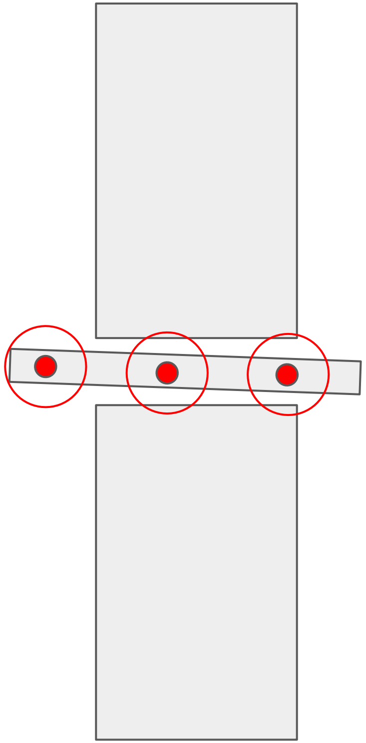
Just compute
minimal translation

What does trajopt help with? CHOMP



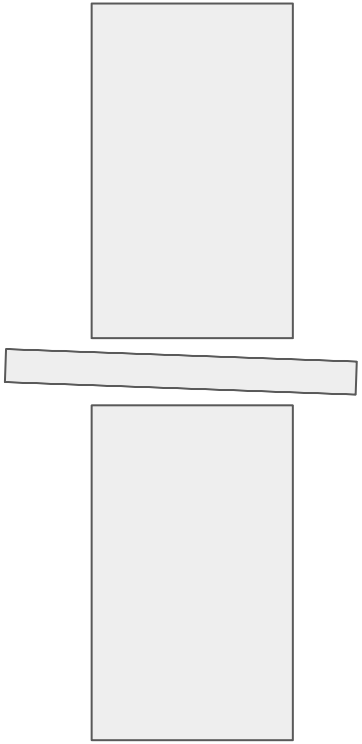
Fast distance checking using spheres

What does trajopt help with? CHOMP



Spheres over-approximate convex objects

What does trajopt help with? CHOMP



Don’t need to approximate for convex objects

Comparison with other motion planning algorithms

	Trajopt	Trajopt-Multi	ompl-RRTConnect	ompl-LBKPIECE	CHOMP-HMC	CHOMP-HMC-Multi
success fraction	0.818	0.955	0.854	0.758	0.652	0.833
average time (s)	0.191	0.3	0.615	1.3	4.91	9.27
avg normed length	1.16	1.15	1.56	1.61	2.04	1.97

198 arm planning problems with PR2 (7 DOF)

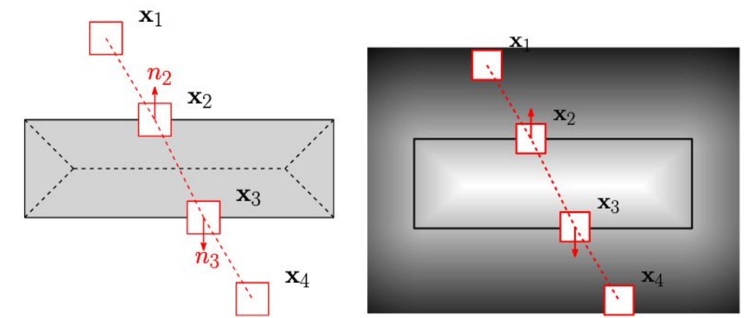
	Trajopt	Trajopt-multi	OMPL-RRTConnect	OMPL-LBKPIECE
success fraction	0.729	0.875	0.406	0.51
average time (s)	2.2	6.1	20.3	18.7
avg normed length	1.06	1.05	1.54	1.51

96 full body planning problems with PR2 (18 DOF)

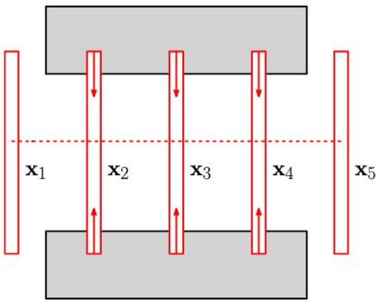
Pros + Cons

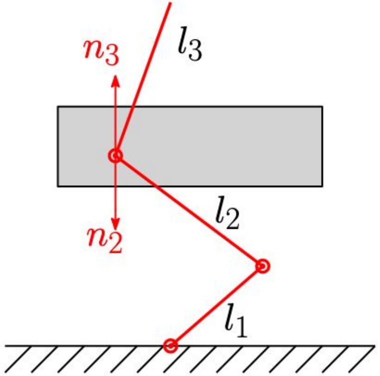
- \+ Typically returns high quality path
- \+ Works in high dimensions
- \+ Faster than comparable optimizers
- \+ Highly customizable
- \- Highly customizable
    - Must specific objective function, gradient descent step size, D_safe, etc.
- \- Not complete
- \- Not optimal
- \- Neglects structure of the problem during optimization
- \- Initialization dependent
- \- More complicated than many sampling based or graph search based methods

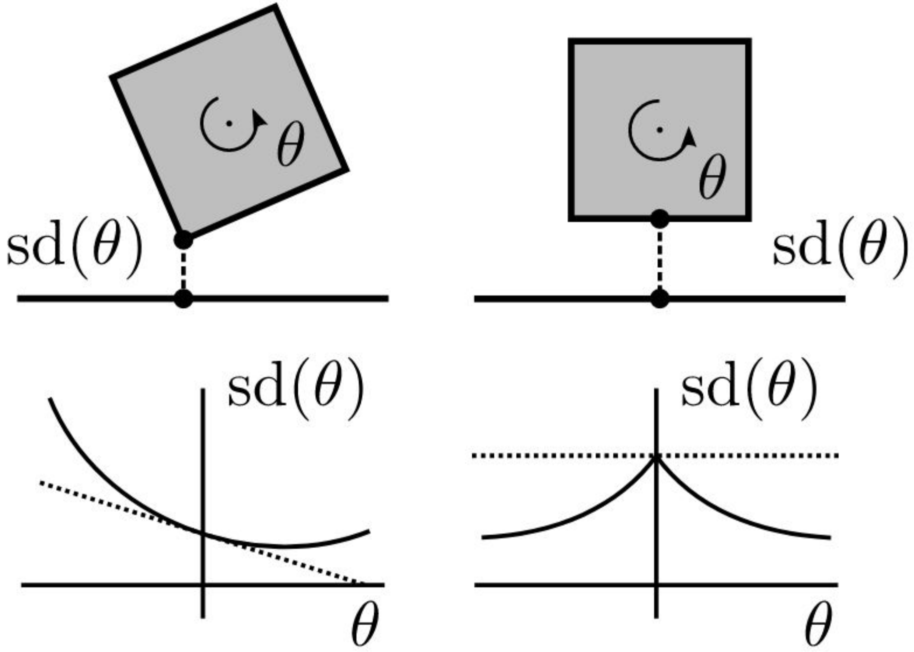
Failure Modes



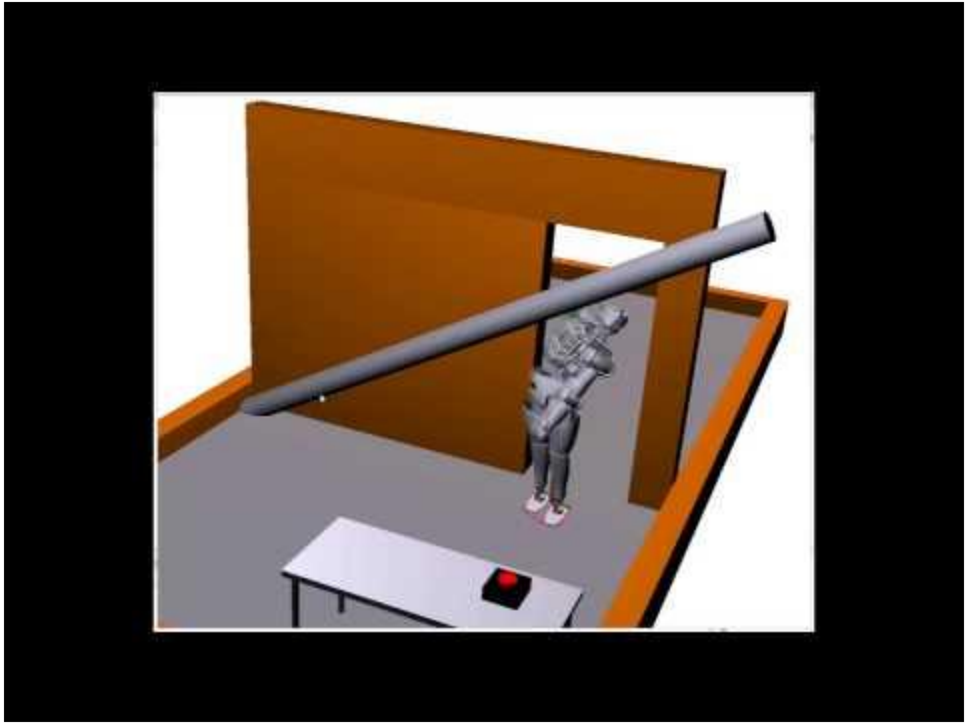
(a) (b)



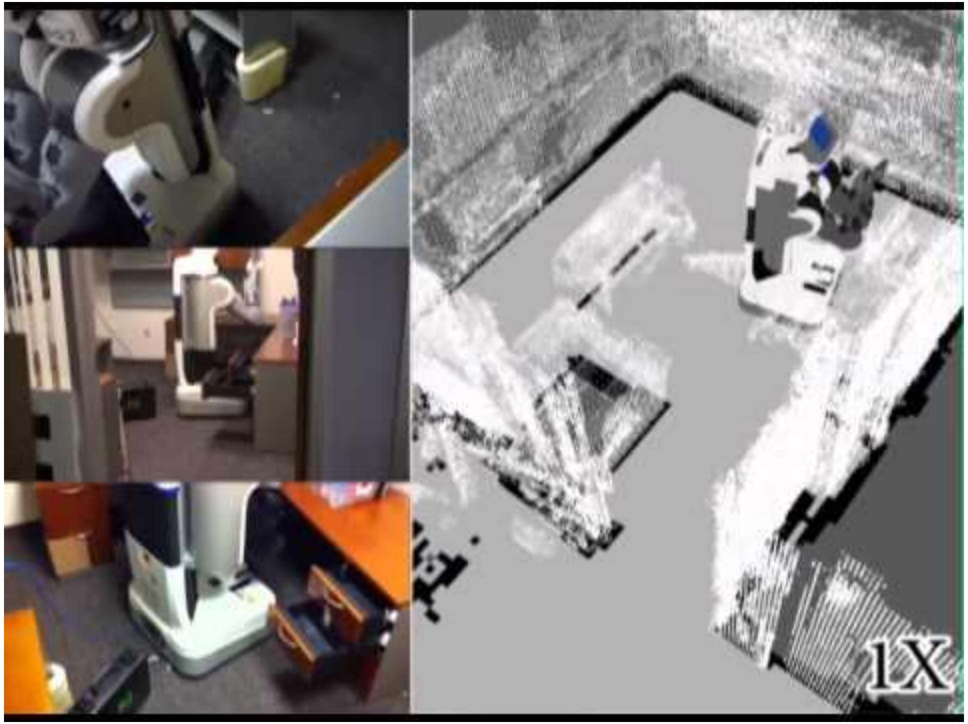




Epic 2013 resolution videos



Epic 2013 resolution videos



Points for discussion

- Other optimization schemes?
- Ideas for tackling failure modes
- GPU acceleration
- Where does the speedup come from? How to further speed up?