SCP for trajectory optimization

- Basic problem
  - minimize\(_{\text{traj}}\) path_length + other costs
  - subject to pose constraints, joint limits, “no collisions”

- Why use optimization for planning?
  - Solve high-DOF problems
  - Smooth solutions
  - Encode preferences
  - It’s wicked fast

- Why SCP rather than some other descent method?
  - Deals with hard constraints and discontinuous costs stably and robustly
  - Solver isn’t the bottleneck anyway
SCP in general

minimize \( f(x) \)
subject to \( g(x) \leq 0 \)

where \( f, g \), may not be convex

- repeat until convergence:
  - convexify objective and constraints
  - solve convex approximation to problem
  - recalculate actual objective
  - if objective decreased
    - shrink trust region
  - else
    - accept update
Non-overlap constraints

- Any kind of collision cost/constraint is non-convex, but we can locally approximate it as convex
  - simple example: consider constraint $x \notin C$

- For convex $C$, this is an “OR” of linear constraints
- Approximation: only impose constraint/cost from closest side to current $x$
Signed distance

- distance(shape1, shape2) = length of shortest translation that puts them in contact. (for non-overlapping shapes)
- penetration_depth(shape1, shape2) = length of shortest translation that takes them out of contact (for overlapping shapes)
- signed_distance(shape1, shape2) =
  - if overlapping: - penetration_depth
  - else: + distance

There are efficient algorithms for convex shapes, based on considering Minkowski difference
- GJK: find if convex set contains the origin
- EPA: find distance from origin to exterior
Collision cost

- Decompose the robot into convex parts
- Cost:
  \[ \sum_t \sum_{i,j} |d_{safe} - \text{signeddist}(\text{part}_i, \text{obstacle}_j)|^+ \]
- Convexification
  - detect all near-collisions
  - for each near-collision, linearize position of closest point using Jacobian

\[ \Delta p = J \Delta \theta \]
\[ \Delta d = \hat{n} : \dot{J} \Delta \theta \]
Two problems

- Need to make collision cost high enough to get out of all collisions
  - solution: increase collision cost coefficient
- Need to make sure trajectory is continuous-time safe
  - solution: subdivide trajectory in collision intervals
Two problems

- Need to make collision cost high enough to get out of all collisions
  - solution: increase collision cost coefficient
  - since it’s an L1 penalty, cost $\rightarrow$ zero for finite coeff
- Need to make sure trajectory is **continuous-time** safe
  - solution: subdivide trajectory in collision intervals
Two problems

- Need to make collision cost high enough to get out of all collisions
  - solution: increase collision cost coefficient
  - since it’s an L1 penalty, cost -> zero for finite coeff
- Need to make sure trajectory is **continuous-time** safe
  - solution: subdivide trajectory in collision intervals
while true:
  do sqp optimization
  if trajectory is not discrete-time safe:
    increase penalty parameter
    continue
  if traj is not continuous-time safe:
    subdivide collision intervals
    continue
  break
Demo videos
How to make SCP fast

- Convexification
  - If func evaluation is expensive, use analytic gradients
- Solving
  - Warm-start
  - Use a fast solver that exploits sparsity (any trajectory problem has banded-diagonal structure)
- Fast convergence
  - Use adaptive trust region adjustment
    
    If exact_improvement > 0.2 * approx_improvement:
    expand trust region
    Else:
    shrink trust region
Robot LfD: comparison of techniques

- Inverse Optimal Control
  - Learn the objective function from human demonstrations, then do optimal control
  - e.g. Abbeel & Ng, 2004

- Trajectory learning
  - Learn a trajectory, the control inputs that achieve it, and a dynamics model
  - e.g. Abbeel, Coates, and Ng 2010

- Behavioral cloning
  - Learn mapping between states and actions
  - e.g. Calinon, Guenter, and Billard 2007
  - the following work
When can’t we use traditional planning & opt. ctrl?

- Planning problem is hard
  - state space is big and you don’t get any gradient info
  - e.g. with deformable objects like rope or cloth
- Can’t simulate
  - e.g. we don’t want to do a fluid simulation to figure out how to pour liquid
- Can simulate, but unable to perceive the full state
  - e.g. crumpled up clothing article
Generalizing trajectories

- Abstract problem: given a bunch of demonstrations of a task, (scene_1, traj_1), (scene_2, traj_2) ..., learn to generate a correct trajectory given a new scene
Knot tying

- very hard to program
- To my knowledge, no one has gotten a robot to autonomously and robustly tie knots with a closed-loop procedure
- The most basic problem:

  given a demonstrated motion on this rope...

  generate an appropriate motion for this rope
Cartoon Problem Setting
Cartoon Problem Setting

demonstration: --- trajectory
Train situation:

Test situation:

Demonstration: --- trajectory

How to perform action here?
Cartoon Problem Setting

Train situation:

Test situation:

demonstration: --- trajectory

How to perform action here?
Cartoon Problem Setting

Train situation:

Test situation:

How to perform action here?

demonstration: --- trajectory
Cartoon Problem Setting

Train situation:

Test situation:

demonstration: ____ trajectory

How to perform action here?
Cartoon Problem Setting

Train situation:

Test situation:

demonstration: --- trajectory

How to perform action here?
Cartoon Problem Setting

Train situation:

Test situation:

How to perform action here?

demonstration: --- trajectory
Cartoon Problem Setting

Train situation:

Test situation:

demonstration: --- trajectory

How to perform action here?

Wednesday, October 31, 12
Cartoon Problem Setting

Train situation:

Test situation:

\[ f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \]

Samples of \( f \) vs. \( \text{demonstration: --- trajectory} \)

How to perform action here?
Cartoon Problem Setting

Train situation:  
Samples of $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

demonstration: --- trajectory

Test situation:  
How to perform action here?
Cartoon Problem Setting

Train situation:

Test situation:

Samples of $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

demonstration: --- trajectory

How to perform action here?
Cartoon Problem Setting

Train situation: 

Test situation:  

demonstration: --- trajectory

Samples of \( f : \mathbb{R}^2 \to \mathbb{R}^2 \)

How to perform action here?
Thin plate splines

- Global smoothness is very important, since this function will determine the gripper trajectory and orientation
- Thin plate splines: regularize function by Frobenius norm of second derivatives matrix

\[ J(f) = \sum_i (y_i - f(x_i))^2 + \lambda \int d^3x \| D_2 f(x) \|^2 \]

- Kernel expansion (1D):

\[ f(x) = \sum_{i=1}^{m} a_i K(x_i, x) + b^\top x + c, \]

\[ K(x, y) = \begin{cases} c_0 r^{4-d} \ln r, & d = 2 \text{ or } d = 4 \\ c_1 r^{4-d}, & \text{otherwise} \end{cases} \quad \text{with } r = \| x - y \|_2.\]
Knot tying procedure

- Look up nearest demonstration
  \[ \text{ClosestDemoRope} = \arg \min_i \text{dist}(\text{DemoRope}_i, \text{NewRope}) \]
- Fit a non-rigid transformation \( f \) that maps from ClosestDemoRope to NewRope
- Apply \( f \) to the end-effector trajectory (positions and orientations) to get a “warped” trajectory
- Execute warped trajectory
Visualization during knot tie
Point cloud registration

- Find a non-rigid transformation between two point clouds
- Given two point clouds $X$, $Y$, find a non-rigid transformation $f$ that minimizes $\text{dist}(f(X), Y)$
  - for some meaningful distance measure $\text{dist}(.)$ on unorganized point clouds
- TPS-RPM Algorithm (Chui & Ragnaran, 2003)
  - Correspondence: find matrix of correspondences between $X$ and $Y$ points
    - $C_{ij} = \text{correspondence between } x_i \text{ and } y_j$
  - Fit thin plate spline transformation that maps each $x_i$ to weighted sum of points $y_j$ it corresponds to
Application to other tasks

- Want to apply this method to a wide assortment of everyday tasks. e.g. in the kitchen:
  - pour, open container, pour, sprinkle, dip, stir, scoop, skewer, unskewer, stack, toss, cover, uncover, press, shake, grind, dump out, slice
- Still need to use non-rigid registration, even if the objects themselves are rigid