Motion Planning Lecture 11

Optimization-Based Motion Planning: Differential Flatness and SCP

Wolfgang Hönig (TU Berlin) and Andreas Orthey (Realtime Robotics)

July 3, 2024

Recap (1)

Foundations

2 Weeks (problem formulation, terminology, collision checking)

Search-based

2 Weeks (A* and variants; state-lattice-based planning)

Sampling-based

5 Weeks (RRT, PRM, OMPL, Sampling Theory)

Optimization-based

2.5 Weeks (Splines, CHOMP, SCP)

Current and Advanced Topics

1.5 Weeks (Comparative Analysis, Machine Learning and Motion Planning, Hybrid- and Multi-Robot approaches)

Recap (2)

- Post-processing / Smoothing of existing solutions:
  - Shortcutting (gradient-free)
  - Splines (B-Splines)
- Geometric Motion Planning
  - Spline Optimization (Polynomials, Bézier Curves)
  - CHOMP (Optimization on Signed Distance Fields)

Today

Kinodynamic Motion Planning

- Splines by using differential flatness
- Sequential Convex Programming (SCP)

Geometric to Kinodynamic Motion Planning via Differential Flatness

Differential Flatness

- Observation: Splines have a temporal component and are smooth
- Can we use them for kinodynamic motion planning?

Differentially Flat System

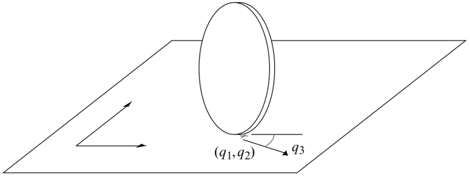
A robot with dynamics $\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u})$ is differentally flat if we can find flat outputs $\mathbf{z}(t)$ such that:

$$\mathbf{q}(t) = g_q(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \ldots)$$
$$\mathbf{u}(t) = g_u(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \ldots).$$

That is, we can compute the configuration and action sequence from $\mathbf{z}(t)$ and a finite number of derivatives of $\mathbf{z}(t)$.

Differential Flatness Example (1)

Unicycle



$\mathbf{u} = (s, \omega) \in \mathcal{U}$ (speed, angular velocity) and $\mathbf{q} = (x, y, \theta) \in \mathcal{Q}$ (position and orientation) The dynamics $\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u})$ are:

$$\dot{x} = s\cos\theta \qquad \dot{y} = s\sin\theta \qquad \dot{\theta} = \omega$$

Differential Flatness Example (2)

Unicycle

The dynamics $\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u})$ are:

$$\dot{x} = s\cos\theta \qquad \dot{y} = s\sin\theta \qquad \dot{\theta} = \omega$$

Pick flat outputs $\mathbf{z}(t) = (x, y)$, i.e., the position of the unicycle.

$$\frac{\dot{y}}{\dot{x}} = \frac{s\sin\theta}{s\cos\theta}$$

$$\frac{\dot{y}}{\dot{x}} = \tan\theta$$

$$\theta = \arctan\left(\frac{\dot{y}}{\dot{x}}\right)$$

$$\begin{aligned} s &= \frac{\dot{x}}{\cos\theta} \\ &= \frac{\dot{x}}{\cos\left(\arctan\left(\frac{\dot{y}}{\dot{x}}\right)\right)} \\ &= \dot{x}\sqrt{\frac{\dot{y}^2}{\dot{x}^2} + 1} = \dot{x}\sqrt{\frac{\dot{y}^2}{\dot{x}^2} + \frac{\dot{x}^2}{\dot{x}^2}} \\ &= \pm\sqrt{\dot{y}^2 + \dot{x}^2} \end{aligned}$$

$$\begin{aligned} \omega = \dot{\theta} &= \frac{d}{dt}\arctan\left(\frac{\dot{y}}{\dot{x}}\right) \\ &= \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2} \end{aligned}$$

Differential Flatness Example (3)

Unicycle

The dynamics $\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u})$ are:

$$\dot{x} = s\cos\theta \qquad \dot{y} = s\sin\theta \qquad \dot{\theta} = \omega$$

Pick flat outputs $\mathbf{z}(t) = (x, y)$, i.e., the position of the unicycle. Then we can compute

$$\mathbf{q}(t) = g_q(\mathbf{z}, \dot{\mathbf{z}}) = \left(x, y, \arctan\left(\frac{\dot{y}}{\dot{x}}\right)\right)$$

$$\mathbf{u}(t) = g_u(\dot{\mathbf{z}}, \ddot{\mathbf{z}}) = \left(\pm\sqrt{\dot{y}^2 + \dot{x}^2}, \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}\right)$$

How does this help for kinodynamic motion planning?

Differential Flatness Applications

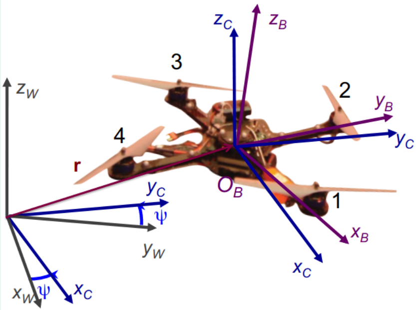
- We can plan a smooth trajectory for $\mathbf{z}(t)$. By applying g_q and g_u, we can compute the configuration and action sequences for the original motion planning problem!
- If $\mathbf{z}(t)$ is lower-dimensional, the planning problem is simplified.

Many robotic systems are differentially flat:

- Unicycle
- Omnidirectional robots
- Differential drive (e.g, Roomba)
- Multirotor
- Car (even when pulling k trailers)

Case Study: Multirotors [1]

Multirotor



- Configuration space: 12+ dimensions (position, orientation, velocity, angular velocity)
- Action space: 4+ dimensions (angular velocity of each propeller)
- Flat output $\mathbf{z}(t) = (x, y, z, \psi)$, i.e., only 4 dimensions (position and yaw angle)

Sufficient to optimize polynomial splines for x, y, z, and ψ, even for aggressive maneuvers.

Video

https://doi.org/10.1109/ICRA.2011.5980409

Differential Flatness Challenges (1)

How can we handle obstacles?

If $\mathbf{z}(t)$ contains the position, we can handle it like before (e.g., adding additional way-points for polynomials or using Bézier splines).

How can we handle dynamic constraints (e.g., maximum speed limit)?

Unicycle: $s(z) = \pm\sqrt{\dot{y}^2 + \dot{x}^2}$

We need to ensure that $\dot{z}$ is bounded appropriately! This can by done by temporal scaling as postprocessing.

Temporal Scaling (1)

Polynomial

$$p(t) = \sum_{k=0}^{n} a_k t^k \quad t \in [0, 1]$$

- Consider a time horizon T, i.e., $t \in [0, T]$:

$$\tilde{p}(t) = \sum_{k=0}^{n} a_k \left(\frac{t}{T}\right)^k = \sum_{k=0}^{n} \frac{a_k}{T^k} t^k = \sum_{k=0}^{n} \tilde{a}_k t^k \text{ where } \tilde{a}_k = \frac{a_k}{T^k}$$

- Time derivative for cubic spline:

$$\tilde{p}'(t) = \tilde{a}_1 + 2\tilde{a}_2 t + 3\tilde{a}_3 t^2 = \frac{a_1}{T} + 2\frac{a_2}{T^2} t + 3\frac{a_3}{T^3} t^2$$

- The derivative gets smaller for $T > 1$:

$$\lim_{T \to \infty} \tilde{p}'(t) = 0$$

Temporal Scaling (2)

```
def temporalScaling():
  z(t) = solveQP() # use arbitrary T
  while True:
    max_mag =
    ↪ computeMaxDerivativeMagnitude(z)
    if lower_bound > max_mag: # too slow
      decrease(T)
    elif upper_bound < max_mag: # too fast
      increase(T)
    else:
      return z(t)
    z(t) = UpdateCoefficients(z(t), T)
```

- Binary search on T can be an efficient way, if stepsize of T is unknown
- Rescaling is fast, since rescaling is just updating the coefficients
- Computing the maximum magnitude is “costly” (numeric methods)
- Similar approach for Bézier curves

Temporal Scaling (3)

```
def temporalScaling():
  z(t) = solveQP() # use arbitrary T
  while True:
    max_mag = computeMaxDerivativeMagnitude(z)
    if lower_bound > max_mag: # too slow
      decrease(T)
    elif upper_bound < max_mag: # too fast
      increase(T)
    else:
      return z(t)
    z(t) = UpdateCoefficients(z(t), T)
```

What can happen if we plan for a 2D plane (unicycle with minimum and maximum speed)?

There might be no feasible T!

Differential Flatness Challenges (2)

The optimization might not minimize the cost we want.

- For the optimization, we might minimize a component of $\boldsymbol{z}(t)$ (or derivatives thereof)
- For our motion planning problem, we might have a different objective, e.g., $J = T$

Differential Flatness Summary

1. Verify that your robot has differentially flatness in the workspace
2. Optimize collision-free splines in workspace (polynomial or Bézier) (QP, i.e., efficient and global optimal solution)
3. Temporally scale to obey dynamic limits (e.g., maximum speed)
4. Extract $\mathbf{q}(t)$ and $\mathbf{u}(t)$ from the spline or use a differentially-flat controller

Differential Flatness Summary (2)

Optimization:

$$\underset{\mathbf{z}_0^1,\ldots\mathbf{z}_n^1,\ldots,\mathbf{z}_0^m,\ldots\mathbf{z}_n^m}{\operatorname{argmin}} \sum_{k=1}^{m} \int_{t=0}^{1} \ddot{\mathbf{z}}^k(t) dt \text{s.t.}$$

$$\mathbf{z}_0^k, \ldots \mathbf{z}_n^k \in \mathit{SafeConvexRegion}(k) \quad \forall k \in \{1, \ldots, m\}$$

$$\mathbf{z}_0^1 = g_q^{-1}(\mathbf{q}_{start})$$

$$\mathbf{z}_n^m = g_q^{-1}(\mathbf{q}_{goal})$$

$$\mathbf{z}^k(1) = \mathbf{z}^{k+1}(0), \quad \dot{\mathbf{z}}^k(1) = \dot{\mathbf{z}}^{k+1}(0), \quad \ddot{\mathbf{z}}^k(1) = \ddot{\mathbf{z}}^{k+1}(0), \ldots \forall k \in \{1, \ldots, m-1\}$$

Sequential Convex Programming (SCP)

Motivation

- Consider a 2D double integrator $\mathbf{q} = (x, y, v_x, v_y)$, $\mathbf{u} = (a_x, a_y)$, $\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u}) = (v_x, v_y, a_x, a_y)$

Kinodynamic Motion Planning

$$\begin{aligned} \underset{T, \mathbf{u}(t), \mathbf{q}(t)}{\operatorname{argmin}} \quad & J(T, \mathbf{u}(t), \mathbf{q}(t)) \quad \text{s.t.} \\ \mathbf{q}(0) = \mathbf{q}_{start} \quad & \mathbf{q}(T) = \mathbf{q}_{goal} \\ \mathcal{B}(\mathbf{q}(t)) \subset \mathcal{W}_{free} \quad & \forall t \in [0, T] \\ \dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) \quad & \forall t \in [0, T) \end{aligned}$$

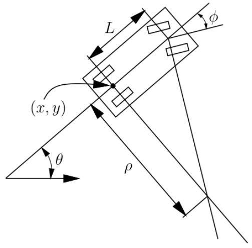
Discrete-Time Optimization

$$\begin{aligned} \underset{\mathbf{u}_0, \ldots \mathbf{u}_{T-1}; \mathbf{q}_0, \ldots \mathbf{q}_T}{\operatorname{argmin}} \quad \sum_{k=1}^{T} \|\mathbf{u}_k\|^2 \quad & \text{s.t.} \\ \mathbf{q}_0 = q_{start} \quad & \mathbf{q}_T = q_{goal} \\ \mathbf{q}_k \geq \mathbf{q}_{min} \quad \mathbf{q}_k \leq \mathbf{q}_{max} \quad & \forall k = 0, \ldots, T \\ \mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{f}(\mathbf{q}_k, \mathbf{u}_k)\Delta t \quad & \forall k = 0, \ldots, T-1 \\ \mathbf{u}_k \geq \mathbf{u}_{min} \quad \mathbf{u}_k \leq \mathbf{u}_{max} \quad & \forall k = 0, \ldots, T-1 \end{aligned}$$

Linear constraints; quadratic cost $\Rightarrow$ convex (QP)

Nonlinear Dynamics

Car Dynamics



$\mathbf{u} = (s, \phi) \in \mathcal{U}$ (speed, steering wheel angle)
$\mathbf{q} = (x, y, \theta) \in \mathcal{Q}$ (position and orientation)
The dynamics $\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u})$ are:

$$\dot{x} = s\cos\theta \qquad \dot{y} = s\sin\theta \qquad \dot{\theta} = \frac{s}{L}\tan\phi$$

Dynamics constraint is not convex!

$$\begin{aligned}\mathbf{q}_{k+1} &= \mathbf{q}_k + \mathbf{f}(\mathbf{q}_k, \mathbf{u}_k)\Delta t \\ &= \mathbf{q}_k + [s\cos\theta, s\sin\theta, \frac{s}{L}\tan\phi]\Delta t\end{aligned}$$

Linearizing Dynamics

If we have a guess $\bar{\mathbf{q}}$ and $\bar{\mathbf{u}}$, we can linearize around those using the first order Taylor expansion:

$$\begin{aligned}\dot{\mathbf{q}} &= \mathbf{f}(\mathbf{q}, \mathbf{u}) \\ &\approx \mathbf{f}(\bar{\mathbf{q}}, \bar{\mathbf{u}}) + \frac{\partial}{\partial \mathbf{q}}\mathbf{f}(\bar{\mathbf{q}}, \bar{\mathbf{u}})(\mathbf{q} - \bar{\mathbf{q}}) + \frac{\partial}{\partial \mathbf{u}}\mathbf{f}(\bar{\mathbf{q}}, \bar{\mathbf{u}})(\mathbf{u} - \bar{\mathbf{u}})\end{aligned}$$

Car Dynamics

$$\mathbf{f}(\mathbf{q}, \mathbf{u}) = \begin{bmatrix} s\cos\theta \\ s\sin\theta \\ \frac{s}{L}\tan\phi \end{bmatrix} \qquad \frac{\partial}{\partial \mathbf{q}}\mathbf{f} = \begin{bmatrix} 0 & 0 & -s\sin\theta \\ 0 & 0 & s\cos\theta \\ 0 & 0 & 0 \end{bmatrix} \quad \frac{\partial}{\partial \mathbf{u}}\mathbf{f} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ \frac{1}{L}\tan\phi & \frac{s}{L\cos^2\phi} \end{bmatrix}$$

$\frac{\partial}{\partial \mathbf{q}}\mathbf{f}(\bar{\mathbf{q}}, \bar{\mathbf{u}})$ is $\frac{\partial}{\partial \mathbf{q}}\mathbf{f}$ evaluated at $\bar{\mathbf{q}}, \bar{\mathbf{u}}$, i.e., a static matrix

Sequential Convex Programming (SCP)

```
def basicSCP():
  x_bar, u_bar = InitialGuess()
  while x_bar, u_bar not valid:
    dyn = LinearizeDynamics(x_bar, u_bar)
    CP = ConstructConvexProblem(dyn, ...)
    x_bar, u_bar = Solve(CP)
```

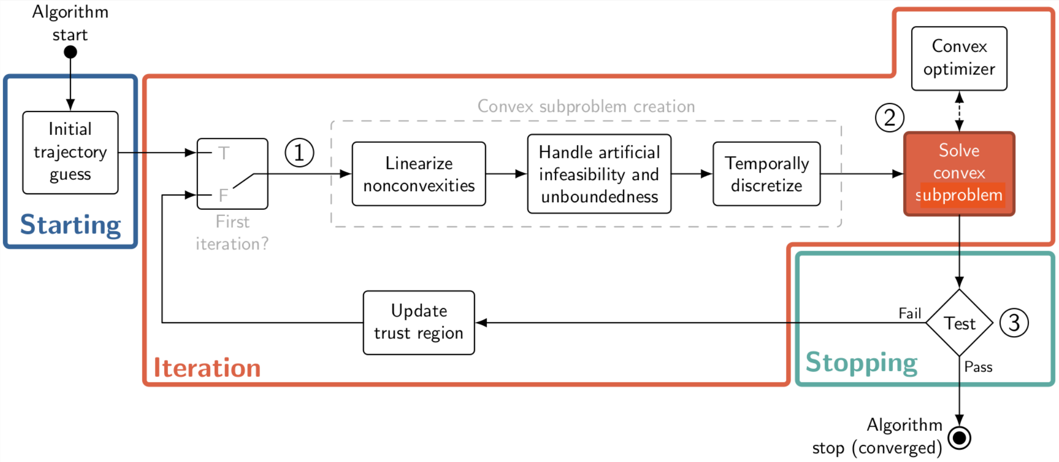
Challenges of the basic version:

1. Linearized dynamics are only valid *around* $\bar{\mathbf{q}}, \bar{\mathbf{u}} \Rightarrow$ add trust region constraints:

$$\bar{\mathbf{q}}_k - r_q \leq \mathbf{q}_k \leq \bar{\mathbf{q}}_k + r_q \quad \forall k = 0, \dots, T$$
$$\bar{\mathbf{u}}_k - r_u \leq \mathbf{u}_k \leq \bar{\mathbf{u}}_k + r_u \quad \forall k = 0, \dots, T-1$$

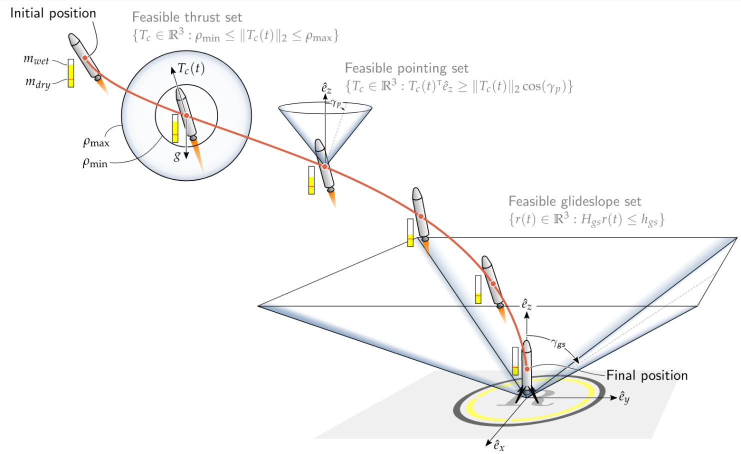
2. CP might be infeasible (even if nonlinear original formulation is feasible) $\Rightarrow$ use soft constraints (with slack variables), rather than hard constraints

Sequential Convex Programming (SCP)



Source: [2]

Sequential Convex Programming (SCP): Case Study



Source: [2]

Demo: `https://github.com/UW-ACL/SCPToolbox_tutorial`

Practical Tips

- Computing Jacobians is mechanical, yet error-prone. Let a computer do it!
  - sympy, Wolfram Alpha, Mathematica if you need an analytic expression
  - jax, pytorch if you need a numeric expression at a specific point (like in SCP)
- A Julia Toolbox with examples is available at:
  `https://github.com/UW-ACL/SCPToolbox.jl`

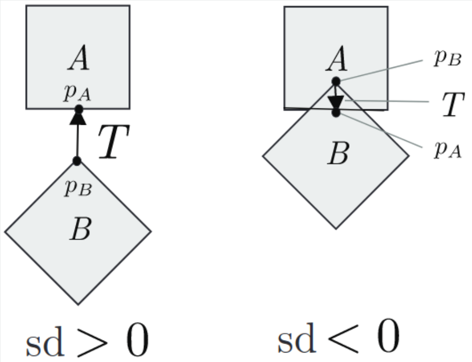
Will SCP converge to a global minimum?

No, since we linearize around an initial guess.

Sequential Convex Programming and Obstacles (1)

- Spline optimization and obstacles: add waypoints or use Bézier curves
- SCP: ?

Signed Distance



Source: [3]

- Let A be a robot and B an obstacle
- Positive: non-overlapping; length of the smallest translation T that puts the two shapes in contact
- Negative: overlapping; length of the smallest translation that takes the two shapes out of contact
- $p_A \in A$ and $p_B \in B$ are the contact points

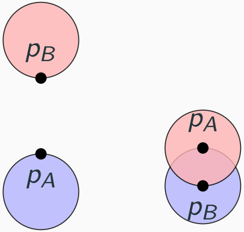
Signed Distance

- FCL can efficiently compute p_A, p_B and sd
  - set `enable_nearest_points` in `DistanceRequest` to `true`
- Define contact normal

$$\mathbf{n} = \begin{cases} \frac{p_A - p_B}{\|p_A - p_B\|} & sd > 0 \\ \frac{p_B - p_A}{\|p_B - p_A\|} & sd \leq 0 \end{cases}$$

- Then:

$$sd = \mathbf{n} \cdot (p_A - p_B)$$



Signed Distance

The robot is moving!

Thus, signed distance depends on configuration $\mathbf{q}$:

$$sd(\mathbf{q}) = \mathbf{n}(\mathbf{q}) \cdot (p_A(\mathbf{q}) - p_B).$$

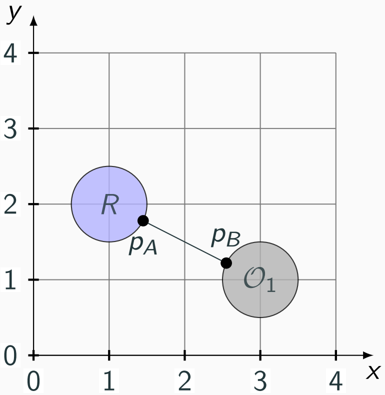
- Assume $\mathbf{n}$ and p_B are static

  Strong assumption that is common in research [3, 4].

- If we have a guess $\bar{\mathbf{q}}$, we can linearize around it using the first order Taylor expansion:

$$\begin{aligned} sd(\mathbf{q}) &\approx sd(\bar{\mathbf{q}}) + \frac{\partial}{\partial \mathbf{q}} sd(\bar{\mathbf{q}})(\mathbf{q} - \bar{\mathbf{q}}) \\ &= sd(\bar{\mathbf{q}}) + \mathbf{n}^\top \frac{\partial}{\partial \mathbf{q}} p_A(\bar{\mathbf{q}})(\mathbf{q} - \bar{\mathbf{q}}) \end{aligned}$$

Sequential Convex Programming and Obstacles (2)



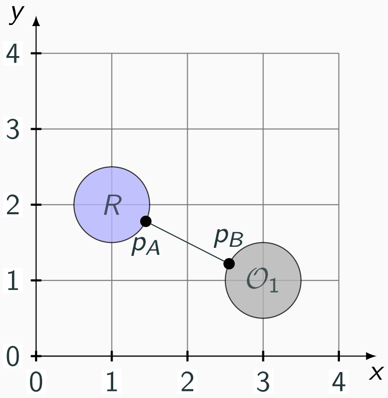
- Step 1: Compute contact points and *sd* (e.g. FCL):

$$p_A \approx [1.45, 1.78]^\top$$
$$p_B \approx [2.55, 1.22]^\top$$
$$sd = \sqrt{5} - 1 \approx 1.23$$

- Step 2: Compute normal vector between contact points:

$$\mathbf{n} = \begin{cases} \frac{p_A - p_B}{\|p_A - p_B\|} & sd > 0 \\ \frac{p_B - p_A}{\|p_B - p_A\|} & sd \leq 0 \end{cases} = \frac{[-2, 1]^\top}{\sqrt{5}}$$

Sequential Convex Programming and Obstacles (3)



- Step 3: Linearize around current state $\bar{\mathbf{q}}$

$$
\begin{aligned}
sd(\mathbf{q}) &\approx sd(\bar{\mathbf{q}}) + \mathbf{n}^\top \frac{\partial}{\partial \mathbf{q}} p_A(\bar{\mathbf{q}})(\mathbf{q} - \bar{\mathbf{q}}) \\
&= 1.23 + \frac{[-2, 1]}{\sqrt{5}} \frac{\partial}{\partial \mathbf{q}} p_A(\bar{\mathbf{q}})(\mathbf{q} - \bar{\mathbf{q}})
\end{aligned}
$$

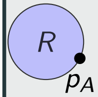
What is the partial derivative?

$p_A(\mathbf{q})$ is the transformation of the robot-local point p_A^L into the global/common coordinate system:

$$p_A(\mathbf{q}) = T^{L \rightarrow W}(\mathbf{q}) \cdot p_A^L$$

Sequential Convex Programming and Obstacles (4)

Double Integrator



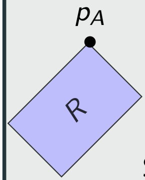
State $\mathbf{q} = [x, y, \dot{x}, \dot{y}]^\top = [1, 2, 0, 0]^\top$

$$p_A = [1.45, 1.78]^\top$$
$$p_A^L = [x_L, y_L]^\top = [0.45, -0.22]^\top$$
$$p_A(x, y, \dot{x}, \dot{y}) = [x_L + x, y_L + y]^\top$$

Only "extract" the position part of $\mathbf{q}$:

$$\frac{\partial}{\partial \mathbf{q}} p_A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Car-like Robot



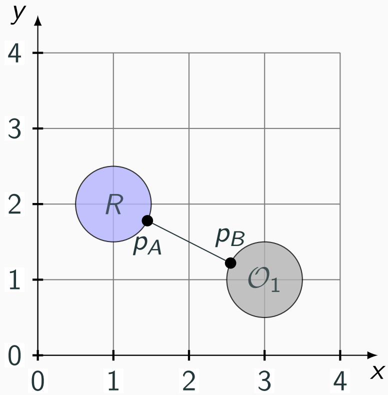
State $\mathbf{q} = [x, y, \theta]^\top$

$$p_A(x, y, \theta) = \begin{bmatrix} x_L \cos(\theta) - y_L \sin(\theta) + x \\ x_L \sin(\theta) + y_L \cos(\theta) + y \end{bmatrix}$$

Exercise

$$\frac{\partial}{\partial \mathbf{q}} p_A = \begin{bmatrix} 1 & 0 & ? \\ 0 & 1 & ? \end{bmatrix}$$

Sequential Convex Programming and Obstacles (5)



Assume double integrator

- Step 3: Linearize around current state
  $\bar{\mathbf{q}} = [1, 2, 0, 0]^\top$

$$sd(\mathbf{q}) \approx sd(\bar{\mathbf{q}}) + \mathbf{n}^\top \frac{\partial}{\partial \mathbf{q}} p_A(\bar{\mathbf{q}})(\mathbf{q} - \bar{\mathbf{q}})$$

$$= 1.23 + \frac{[-2, 1]}{\sqrt{5}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} (\mathbf{q} - [1, 2, 0, 0]^\top)$$

Sanity Checks

- Dimensions $(1, 2) \times (2, 4) \times (4, 1)$
- $q \rightarrow [2, 2, 0, 0]^\top \Rightarrow sd = 0.35$
- $q \rightarrow [0, 2, 0, 0]^\top \Rightarrow sd = 2.12$

Sequential Convex Programming and Obstacles (6)

- Step 4: Add linear constraint of form $sd(\mathbf{q}) \geq 0$
- Step 5: Repeat for each obstacle and timestep (i.e., $(T+1) \cdot (|\mathcal{O}|)$ constraints)

If obstacles are not convex...

... split into the union of convex obstacles first.

SCP and Time-Optimal Planning

Approach 1 (Naive)

1. Solve SCP with arbitrary guess
2. Reduce T, use (interpolated) result as initial guess
3. Repeat, until SCP becomes infeasible

Issues?

- Does not work with arbitrary dynamics (e.g., airplane)
- Potentially slow
- Difficult initial guess

Approach 2

Can we add Δt as a decision variable?

$$
\begin{aligned}
\mathbf{q}_{k+1} &= \mathbf{q}_k + f(\mathbf{q}_k, \mathbf{u}_k)\Delta t \\
&\approx \mathbf{q}_k + (f(\bar{\mathbf{q}}_k, \bar{\mathbf{u}}_k) + \\
&\quad \mathbf{A}(\mathbf{q}_k - \bar{\mathbf{q}}_k) + \mathbf{B}(\mathbf{u}_k - \bar{\mathbf{u}}_k))\Delta t
\end{aligned}
$$

($\mathbf{A}$, $\mathbf{B}$ are fixed Jacobian matrices)

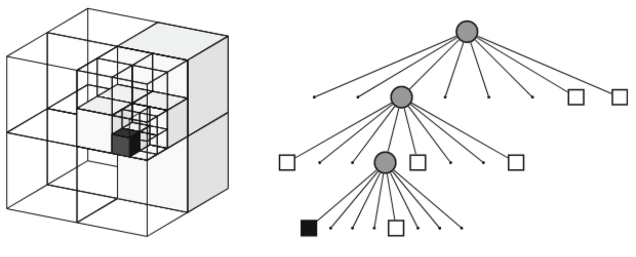
Quadratic constraint!

Trick

$$
\begin{aligned}
\mathbf{q}_{k+1} &= \tilde{f}(\mathbf{q}_k, \mathbf{u}_k, \Delta t) \\
&\approx \tilde{f}(\bar{\mathbf{q}}_k, \bar{\mathbf{u}}_k, \bar{\Delta} t) + \tilde{\mathbf{A}}(\mathbf{q}_k - \bar{\mathbf{q}}_k) + \\
&\quad \tilde{\mathbf{B}}(\mathbf{u}_k - \bar{\mathbf{u}}_k) + \tilde{\mathbf{C}}(\Delta t - \bar{\Delta} t)
\end{aligned}
$$

Useful Datastructures

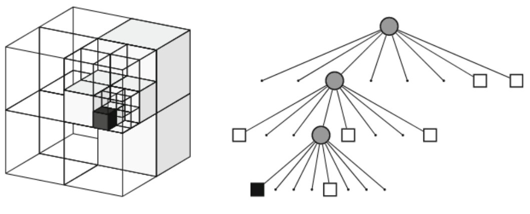
OctoMap [5]



- Octree to store occupancy probability
- Efficient update from noisy sensor data (LIDAR, RGB-D)
- Very compact map size, efficient update and query

Demo `octovis fr_campus.bt`

OctoMap [5]



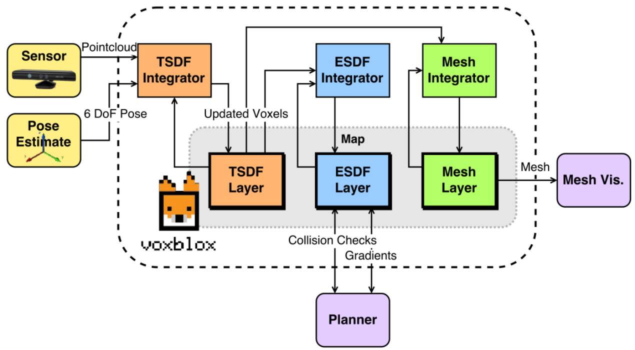
Why is this useful for optimization-based motion planning?

Each cube is convex, i.e., we can use it for Bézier Curve constraints.

https://octomap.github.io

Faster, newer alternative: UFOMap [6]

VoxBlox [7]



- Directly maintain a Euclidean Signed Distance Field (ESDF) from RGB-D data
- Gradient-based optimization similar to CHOMP for planning

https://github.com/ethz-asl/voxblox

Video

https://doi.org/10.1109/IROS.2017.8202315

Conclusion

- Differential flatness to use geometric motion planning for kinodynamic systems
  - Find a mapping from workspace to configuration space
  - Works for many robotic systems (car, multirotor, ...)
  - Enables very efficient, globally optimal planning (perhaps not for the desired optimization criteria)
- Sequential Convex Programming (SCP)
  - Linearize dynamics, constraints, objective around some solution
  - Repeat
- Useful Datastructures: OctoMap and Voxblox

Next Time

- More optimization-based motion planning
- Search / Sampling / Optimization-based motion planning comparison

Suggested Reading

1. Steven M. LaValle. **Planning algorithms.** Cambridge University Press, 2006. ISBN: 978-0-521-86205-9. URL: http://planning.cs.uiuc.edu, Section 15.5.3
2. Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. **Principles of Robot Motion: Theory, Algorithms, and Implementations.** Intelligent Robotics and Autonomous Agents Series. Cambridge, MA, USA: A Bradford Book, 2005. 630 pp. ISBN: 978-0-262-03327-5, Section 12.5.5
3. Russ Tedrake. **Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation.** 2022. URL: http://underactuated.mit.edu, Chapter 10

References i

[1] Daniel Mellinger and Vijay Kumar. **"Minimum Snap Trajectory Generation and Control for Quadrotors".** In: *2011 IEEE International Conference on Robotics and Automation*. 2011 IEEE International Conference on Robotics and Automation. May 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[2] Danylo Malyuta, Taylor P. Reynolds, Michael Szmuk, Thomas Lew, Riccardo Bonalli, Marco Pavone, and Behçet Açıkmeşe. **"Convex Optimization for Trajectory Generation: A Tutorial on Generating Dynamically Feasible Trajectories Reliably and Efficiently".** In: *IEEE Control Systems Magazine* 42.5 (2022), pp. 40–113. DOI: 10.1109/MCS.2022.3187542.

[3] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. **"Motion Planning with Sequential Convex Optimization and Convex Collision Checking".** In: *The International Journal of Robotics Research* 33.9 (Aug. 1, 2014), pp. 1251–1270. ISSN: 0278-3649. DOI: 10.1177/0278364914528132.

[4] Josep Virgili-Llop, Costantinos Zagaris, Richard Zappulla Ii, and Marcello Romano. **"Convex Optimization for Proximity Maneuvering of a Spacecraft with a Robotic Manipulator".** In: *AAS/AIAA Spaceflight Mechanics Meeting*. 2017.

[5] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. **"OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees".** In: *Autonomous Robots* 34.3 (2013), pp. 189–206. DOI: 10.1007/s10514-012-9321-0.

[6] Daniel Duberg and Patric Jensfelt. **"UFOMap: An Efficient Probabilistic 3D Mapping Framework That Embraces the Unknown".** In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 6411–6418. ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3013861.

[7] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. **“Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-Board MAV Planning”.** In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[8] Steven M. LaValle. **Planning algorithms.** Cambridge University Press, 2006. ISBN: 978-0-521-86205-9. URL: http://planning.cs.uiuc.edu.

[9] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. **Principles of Robot Motion: Theory, Algorithms, and Implementations.** Intelligent Robotics and Autonomous Agents Series. Cambridge, MA, USA: A Bradford Book, 2005. 630 pp. ISBN: 978-0-262-03327-5.

[10] Russ Tedrake. **Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation.** 2022. URL: http://underactuated.mit.edu.