The trajectory planning problem

- design a reference trajectory $x(t) \in \mathbb{R}^n$ and control inputs $u(t) \in \mathbb{R}^m$ by solving the constrained optimal control problem:

\[
\min_{x(\cdot), u(\cdot), t_f} J = \phi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t) dt,
\]

subject to:

\[
x(t_0) = x_0, \quad x(t_f) \text{ and } t_f \text{ free}
\]

\[
\dot{x}(t) = f(x(t), u(t), t)
\]

\[
c(x(t), u(t), t) \leq 0, \text{ for all } t \in [t_0, t_f]
\]

\[
\psi(x(t_f), t_f) \leq 0,
\]

where

- $[t_0, t_f]$ - time horizon, $x_0$-initial state
- $L$ - trajectory cost: e.g. control effort, energy, time, distance
- $\phi$ - terminal cost: e.g. reaching a desired region
- $c$ - trajectory constraints: e.g. control bounds, forbidden regions in $X$
- $\psi$ - terminal constraint defines algebraically a goal region
Issues and Challenges

Generally, it’s a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques

- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages
- linearize/convexify the problem
  - might become too conservative or not realizable; might not scale to complex constraints
- discretize the space and use discrete search
  - not scalable: exponential in state dimension and time
- dynamic constraints difficult to handle
- sampling-based methods
  - randomized approximation of the space of trajectories (e.g. as a graph with randomly sampled nodes) and then search
  - By law of large numbers it approaches the optimal solution but typically at a very slow rate
Issues and Challenges

Generally, it’s a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques

- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
Issues and Challenges

Generally, it’s a hard problem:
- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques
- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages
Issues and Challenges

Generally, it’s a hard problem:

▶ no closed-form solution in general (beyond the linear-quadratic case)
▶ infinite dimensional; numerically NP-complete

Solution Techniques

▶ nonlinear optimization (over finite trajectory parameterization)
  ▶ could be slow, might not converge, only locally optimal
▶ stochastic trajectory optimization
  ▶ cannot handle complex constraints, e.g. narrow passages
▶ linearize/convexify the problem
  ▶ might become too conservative or not realizable; might not scale to complex constraints
Issues and Challenges

Generally, it’s a hard problem:
- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques
- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages
- linearize/convexify the problem
  - might become too conservative or not realizable; might not scale to complex constraints
- discretize the space and use discrete search
  - not scalable: exponential in state dimension and time
  - dynamic constraints difficult to handle
Issues and Challenges

Generally, it’s a hard problem:
- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques
- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages
- linearize/convexify the problem
  - might become too conservative or not realizable; might not scale to complex constraints
- discretize the space and use discrete search
  - not scalable: exponential in state dimension and time
  - dynamic constraints difficult to handle
- sampling-based methods
  - randomized approximation of the space of trajectories (e.g. as a graph with randomly sampled nodes) and then search
  - By law of large numbers it approaches the optimal solution but typically at a very slow rate
Example: Tree-based Sampling Motion Planning

optimal motion?  probabilistic roadmap (PRM)  kinodynamic planning
The basic algorithm: rapidly-exploring random tree (RRT)

Algorithm 1: $\mathcal{T} \leftarrow \text{RRT}(\eta_0)$

1. $\mathcal{T} \leftarrow \text{InitializeTree}()$
2. $\mathcal{T} \leftarrow \text{InsertNode}(\emptyset, \eta_0, \mathcal{T})$
3. for $i = 1 : N$ do
   4. $\eta_{\text{rand}} \leftarrow \text{Sample}$
   5. $\eta_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}, \eta_{\text{rand}})$
   6. $(\bar{x}_{\text{new}}, \bar{u}_{\text{new}}, T_{\text{new}}) \leftarrow \text{Steer}(\eta_{\text{nearest}}, \eta_{\text{rand}})$
   7. if ObstacleFree($\bar{x}_{\text{new}}$) then
      8. $\mathcal{T} \leftarrow \text{InsertNode}(\eta_{\text{nearest}}, \eta_{\text{new}}, \mathcal{T})$
   9. return $\mathcal{T}$

- A node is the tuple $\eta_i = (x_i, p_i, J_i) \in \mathcal{N} = \mathcal{X} \times \mathbb{N} \times \mathbb{R}_+$ where
  - $x_i \in \mathcal{X}$ is the state
  - $p_i \in \mathbb{N}$ is the index of the parent node of $i$, i.e. $\eta_{p_i}$ is the parent of $\eta_i$
  - $J_i$ is the cumulative cost from the start $\eta_0$ to $\eta_i$
- A tree $\mathcal{T} \subset \mathcal{N}$ is a particular arrangement of nodes
The basic algorithm: rapidly-exploring random tree (RRT)

Algorithm 2: $\mathcal{T} \leftarrow \text{RRT}(\eta_0)$

1. $\mathcal{T} \leftarrow \text{InitializeTree}()$
2. $\mathcal{T} \leftarrow \text{InsertNode}(\emptyset, \eta_0, \mathcal{T})$
3. for $i = 1 : N$ do
   4. $\eta_{\text{rand}} \leftarrow \text{Sample}$
   5. $\eta_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}, \eta_{\text{rand}})$
   6. $(\bar{x}_{\text{new}}, \bar{u}_{\text{new}}, \mathcal{T}_{\text{new}}) \leftarrow \text{Steer}(\eta_{\text{nearest}}, \eta_{\text{rand}})$
   7. if ObstacleFree($\bar{x}_{\text{new}}$) then
      8. $\mathcal{T} \leftarrow \text{InsertNode}(\eta_{\text{nearest}}, \eta_{\text{new}}, \mathcal{T})$
9. return $\mathcal{T}$

- A node is the tuple $\eta_i = (x_i, p_i, J_i) \in \mathcal{N} = \mathcal{X} \times \mathbb{N} \times \mathbb{R}_+$ where
  - $x_i \in \mathcal{X}$ is the state
  - $p_i \in \mathbb{N}$ is the index of the parent node of $i$, i.e. $\eta_{p_i}$ is the parent of $\eta_i$
  - $J_i$ is the cumulative cost from the start $\eta_0$ to $\eta_i$

- A tree $\mathcal{T} \subset \mathcal{N}$ is a particular arrangement of nodes
Key ingredients

- sampling routine $\text{Sample}$
- distance function $\rho(x_a, x_b) \geq 0$ for determining $\text{Nearest}(\mathcal{T}, \eta)$
- steering method $\text{Steer}(\eta_a, \eta_b)$
- collision detection $\text{ObstacleFree}(x)$
Key ingredients: sampling routine Sample

- **Baseline**: uniform sampling
  
  - **low-dispersion**: reduce largest unsampled space between all samples
    \[
    \delta(P) = \sup_{x \in \mathcal{X}} \min_{x' \in P} \{\rho(x, x')\},
    \]
    where \( P \) is a set of sampled points
  
  - **low-discrepancy**: # of samples inside a set are consistent with the volume of the set
    \[
    D(P, \mathcal{R}) = \sup_{R \in \mathcal{R}} \left\{ \left\| \frac{|P \cap R|}{k} - \frac{\mu(R)}{\mu(\mathcal{X})} \right\| \right\},
    \]
    where \( \mathcal{R} \) are all subsets of \( \mathcal{X} \) and \( \mu \) measures the volume of a set

- Non-uniform sampling: exploiting problem structure (more later)
Key ingredients: sampling routine Sample

- **Baseline**: uniform sampling

- **low-dispersion**: reduce largest unsampled space between all samples

\[
\delta(P) = \sup \{ \min_{x \in X} \min_{x' \in P} \{ \rho(x, x') \} \},
\]

where \( P \) is a set of sampled points

- **low-discrepancy**: # of samples inside a set are consistent with the volume of the set

\[
D(P, \mathcal{R}) = \sup_{R \in \mathcal{R}} \left\{ \left\| \frac{|P \cap R|}{k} - \frac{\mu(R)}{\mu(X)} \right\| \right\},
\]

where \( \mathcal{R} \) are all subsets of \( X \) and \( \mu \) measures the volume of a set

- **non-uniform sampling**: exploiting problem structure (more later)
Distance function $\rho(x_a, x_b) \geq 0$ for determining Nearest $(T, \eta)$

- ideal distance is the true cost-to-go from $x_a$ to $x_b$, i.e.
  $$\rho(x_a, x_b) = J(\bar{x}_{a\rightarrow b}, \bar{u}_{a\rightarrow b})$$
Key ingredients: distance function

Distance function $\rho(x_a, x_b) \geq 0$ for determining Nearest $(T, \eta)$

- ideal distance is the true cost-to-go from $x_a$ to $x_b$, i.e.
  $\rho(x_a, x_b) = J(\bar{x}_{a\rightarrow b}, \bar{u}_{a\rightarrow b})$

- which typically unavailable or expensive to compute so use a lower bound *heuristic cost*, e.g.

\[
\rho(x_a, x_b) = \sqrt{(x_b - x_a)^T W (x_b - x_a)},
\]

i.e. a weighted Euclidean distance (for some matrix $W > 0$)
Key ingredients: distance function

Distance function $\rho(x_a, x_b) \geq 0$ for determining Nearest $(\mathcal{T}, \eta)$

- ideal distance is the true cost-to-go from $x_a$ to $x_b$, i.e.
  \[ \rho(x_a, x_b) = J(\bar{x}_{a \rightarrow b}, \bar{u}_{a \rightarrow b}) \]
- which typically unavailable or expensive to compute so use a lower bound heuristic cost, e.g.
  \[ \rho(x_a, x_b) = \sqrt{(x_b - x_a)^T W (x_b - x_a)}, \]
  i.e. a weighted Euclidean distance (for some matrix $W > 0$)

- Nearest $(\mathcal{T}, \eta)$ can be set by:
  - $\rho(x_a, x_b)$: local distance ordering, i.e. standard RRT
  - $J_a + \rho(x_a, x_b)$: cost-to-come to parent + local distance ordering, i.e. RRT with optimal cost-to-come
Key ingredients: steering method $\text{Steer}(\eta_a, \eta_b)$

- Structured models (assume controllability)
  - open-loop trajectory generation: exploit nonholonomy, flatness, symmetries, if possible
  - employ efficient closed-form local methods, e.g. polynomial boundary value solutions

- Complicated / Black box models:
  - only possible to sample control space
  - observe/simulate generated trajectories
  - must be resolution complete: i.e. reach infinitely close to any state
  - typically implies a regularity condition: that small change in $u$ result in small change in $x$
Key ingredients: steering method $\text{Steer}(\eta_a, \eta_b)$

- **Structured models (assume controllability)**
  - open-loop trajectory generation: exploit nonholonomy, flatness, symmetries, if possible
  - employ efficient closed-form local methods, e.g. polynomial boundary value solutions
- **Complicated / Black box models:**
  - only possible to sample control space
  - observe/simulate generated trajectories
  - must be *resolution complete*: i.e. reach infinitely close to any state
  - typically implies a regularity condition: that small change in $u$ result in small change in $x$
Key ingredients: steering method $\text{Steer}(\eta_a, \eta_b)$

- Structured models (assume controllability)
  - open-loop trajectory generation: exploit nonholonomy, flatness, symmetries, if possible
  - employ efficient closed-form local methods, e.g. polynomial boundary value solutions

- Complicated / Black box models:
  - only possible to sample control space
  - observe/simulate generated trajectories
  - must be resolution complete: i.e. reach infinitely close to any state
  - typically implies a regularity condition: that small change in $u$ result in small change in $x$

- Steering using a finite set of primitives
  - primitives must be carefully chosen to satisfy controllability
  - in this case controllability is equivalent to resolution completeness
Key ingredients: collision checking \( \text{ObstacleFree}(\tilde{x}_{\text{new}}) \)

- ensure constraints \( c(t, x, u) \leq 0 \) are satisfied
- often the free configuration space is difficult to compute
- easiest to use a black-box collision checking package
- simulate controls \( u(t) \) and check collision

Example: Proximity Query Package (PQP)
http://gamma.cs.unc.edu/SSV/

PQP collision checking

PQP distance and direction
Tree-based planners

Various tree-based planners are possible (LaValle, 2006)

It is critical to solve the boundary value (steering) problem (BVP)

a) standard planning to a goal set $X_G$
b) reaching a specific goal
c) tree grown backwards from goal
d) bidirectional tree: forward from start and backward from goal
Key challenges in motion planning

- achieving efficiency even in high dimensions
- handling complicated constraints, e.g. narrow passages
- finding optimal not just feasible solutions
- hybrid and non-smooth systems
- distributed systems planning, parallel processing
- dealing with uncertainty
  - partially known system dynamics
  - unstructured dynamic uncertain environment
  - formal robustness guarantees
- holy grail: unifying planning, estimation, and control
From exploration to optimality

- Sampling-based methods are good at exploring the space to find “a path” but notoriously slow in converging to the “optimal” path.
- An important recent method: RRT* (Karaman, Frazzoli, 2011)
- Idea: rewire tree to maintain optimal cost-to-go
- Key result: only need to rewire by checking $\approx \log(n)$ neighbors
- Challenges: extend theory to complex dynamics; principled neighbor selection; CPU time?

![Images of RRT and RRT* iterations](images)
The RRT* algorithm

Algorithm 3: \( \mathcal{T} \leftarrow \text{RRT}^*(\eta_0, X_g) \)

1. \( \mathcal{T} \leftarrow \text{InitializeTree()} \)
2. \( \mathcal{T} \leftarrow \text{InsertNode}(\emptyset, \eta_0, \mathcal{T}) \)
3. for \( i = 1 : N \) do
   4. \( \eta_{\text{rand}} \leftarrow \text{Sample}(i) \)
   5. \( \eta_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}, \eta_{\text{rand}}) \)
   6. \( (x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \text{Steer}(\eta_{\text{nearest}}, \eta_{\text{rand}}) \)
   7. if \( \text{ObstacleFree}(x_{\text{new}}) \) then
      8. \( \mathcal{N}_{\text{near}} \leftarrow \text{Near}(\mathcal{T}, \eta_{\text{new}}, |V|) \)
      9. \( \eta_{\text{min}} = \text{ChooseParent}(\mathcal{N}_{\text{near}}, \eta_{\text{nearest}}, x_{\text{new}}) \)
     10. \( \mathcal{T} \leftarrow \text{InsertNode}(\eta_{\text{min}}, \eta_{\text{new}}, \mathcal{T}) \)
     11. \( \mathcal{T} \leftarrow \text{Rewire}(\mathcal{T}, \mathcal{N}_{\text{near}}, \eta_{\text{min}}, \eta_{\text{new}}) \)
4. return \( \mathcal{T} \)
The RRT* algorithm (cont.)

**Algorithm 4:** \( \eta_{\text{min}} \leftarrow \text{ChooseParent}(N_{\text{near}}, \eta_{\text{nearest}}, x_{\text{new}}) \)

1. \( \eta_{\text{min}} \leftarrow \eta_{\text{nearest}} \)
2. \( c_{\text{min}} \leftarrow \text{CostToCome}(\eta_{\text{nearest}}) + \text{Cost}(x_{\text{new}}) \)
3. for \( \eta_{\text{near}} \in N_{\text{near}} \) do
   4. \((x', u', T') \leftarrow \text{Steer}(\eta_{\text{near}}, \eta_{\text{new}}) \)
   5. if ObstacleFree\((x')\) and \(x'(T') = \eta_{\text{new}}\) then
      6. \(c' = \text{CostToCome}(\eta_{\text{near}}) + \text{Cost}(x')\)
      7. if \(c' < c_{\text{min}}\) then
         8. \(\eta_{\text{min}} \leftarrow \eta_{\text{near}}\)
         9. \(c_{\text{min}} \leftarrow c'\)
4. return \(\eta_{\text{min}}\)

**Algorithm 5:** \( T \leftarrow \text{Rewire}(T, N_{\text{near}}, \eta_{\text{min}}, x_{\text{new}}) \)

1. for \( \eta_{\text{near}} \in N_{\text{near}} \setminus \{\eta_{\text{min}}\} \) do
   2. \((x', u', T') \leftarrow \text{Steer}(\eta_{\text{new}}, \eta_{\text{near}}) \)
   3. if ObstacleFree\((x')\) and \(x'(T') = \eta_{\text{near}}\) and
      4. \(\text{CostToCome}(\eta_{\text{new}}) + \text{Cost}(x') < \text{CostToCome}(\eta_{\text{near}})\) then
      5. \(T \leftarrow \text{Reconnect}(\eta_{\text{new}}, \eta_{\text{near}}, T)\)
4. return \(T\)