Short introduction to motion control of manipulators

Karel Zimmermann

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics
Center for Machine Perception
http://cmp.felk.cvut.cz/~zimmerk, zimmerk@fel.cvut.cz

Preliminaries

- Many different robots with various complexity of dynamics and dimensionality of state-action space:
  - triple pendulum,
  - two-arm manipulator,
  - mobile platform with auxiliary articulated sub-tracks and lockable differential.
- Usually given
  - **states**: $\mathbf{x} \in X$,
  - **actions**: $\mathbf{u} \in U$,
  - sometimes **motion model**: $\mathbf{x}' = f(\mathbf{x}, \mathbf{u}) : X \times U \to X$

Problem definition

- **Hard problem:** Find the optimal collision-free trajectory from start state(s) $\mathbf{x}_s$ to goal state(s) $\mathbf{x}_g$ which respect dynamic constraints of the robot. (e.g. LQR-trees or Guided Policy Search)

Problem definition

- **Hard problem:** Find the optimal collision-free trajectory from start state(s) $\mathbf{x}_s$ to goal state(s) $\mathbf{x}_g$ which respect dynamic constraints of the robot. (e.g. LQR-trees or Guided Policy Search)
- **Easier problem:** Find a feasible collision-free path from start position $\mathbf{x}_s$ to goal position $\mathbf{x}_g$ given a kinematic model of the robot. (e.g. Dijkstra, RRT, PRM)

Problem definition

- **Hard problem:** Find the optimal collision-free trajectory from start state(s) $\mathbf{x}_s$ to goal state(s) $\mathbf{x}_g$ which respect dynamic constraints of the robot. (e.g. LQR-trees or Guided Policy Search)
- **Easier problem:** Find a feasible collision-free path from start position $\mathbf{x}_s$ to goal position $\mathbf{x}_g$ given a kinematic model of the robot. (e.g. Dijkstra, RRT, PRM)
- **If robot's dynamic is weak:** one possible approximation to the hard problem is to solve the easier problem and interpolate the path by a smooth trajectory.

Outline

First part of this lecture: solve the easier problem and replace the path by a smooth curve.

1. Short summary of path planning via Rapidly Exploring Random Trees (RRT).
2. Introduce cubic spline interpolation of the planned path.

Second part of this lecture: what to do if motion model is not available?

3. Short intro to reinforcement learning for robotics.

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl

Planning

- Open Motion Planning Library: `http://wiki.ros.org/ompl`
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra, RRT

Planning

- Open Motion Planning Library: `http://wiki.ros.org/ompl`
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra, RRT
- Real robots usually operate in a continuous space.

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra, RRT
- Real robots usually operate in a continuous space.
- Search in continuous high-dimensional space can get stuck in a local minimum, since you can expand infinite number of nodes in a small region.

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra, RRT
- Real robots usually operate in a continuous space.
- Search in continuous high-dimensional space can get stuck in a local minimum, since you can expand infinite number of nodes in a small region.
- RRT [1] efficiently search non-convex, high-dimensional spaces by randomly building a space-filling tree.
- Initial publication [1] has over 1200 citations.

[1] LaValle, Steven M. (October 1998). "Rapidly-exploring random trees: A new tool for path planning"(PDF). Technical Report (Computer Science Department, Iowa State University) (TR 98-11).

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)
4. Find the closest node to $\mathbf{x}$ (e.g. KD tree) $\mathbf{y}^* = \arg\min_{\mathbf{y}} \|\mathbf{y} - \mathbf{x}\|$

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)
4. Find the closest node to $\mathbf{x}$ (e.g. KD tree) $\mathbf{y}^* = \arg\min_{\mathbf{y}} \|\mathbf{y} - \mathbf{x}\|$
5. Try connect $\mathbf{y}^*$ with $\mathbf{x}$
   - if possible: add node $\mathbf{x}$ and edge $[\mathbf{y}^*, \mathbf{x}]$ to the search tree.
   - otherwise: throw $\mathbf{x}$ away.

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)
4. Find the closest node to $\mathbf{x}$ (e.g. KD tree) $\mathbf{y}^* = \arg\min_{\mathbf{y}} \|\mathbf{y} - \mathbf{x}\|$
5. Try connect $\mathbf{y}^*$ with $\mathbf{x}$
   - if possible: add node $\mathbf{x}$ and edge $[\mathbf{y}^*, \mathbf{x}]$ to the search tree.
   - otherwise: throw $\mathbf{x}$ away.
6. Repeat from 2 until a feasible path is found.

Output: a collision-free path if exist (after infinite number of nodes expanded)

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)
- If optimality is important:

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)
- If optimality is important:
  - informed-RRT

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)
- If optimality is important:
  - informed-RRT
  - RRT*

Properties of Rapidly Exploring Random Tree

- What might be time consuming?

Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).

Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.

Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.
- Probabilistic completeness: if nodes number reaches infinity than you find a feasible path (if exists) with $P = 1$.

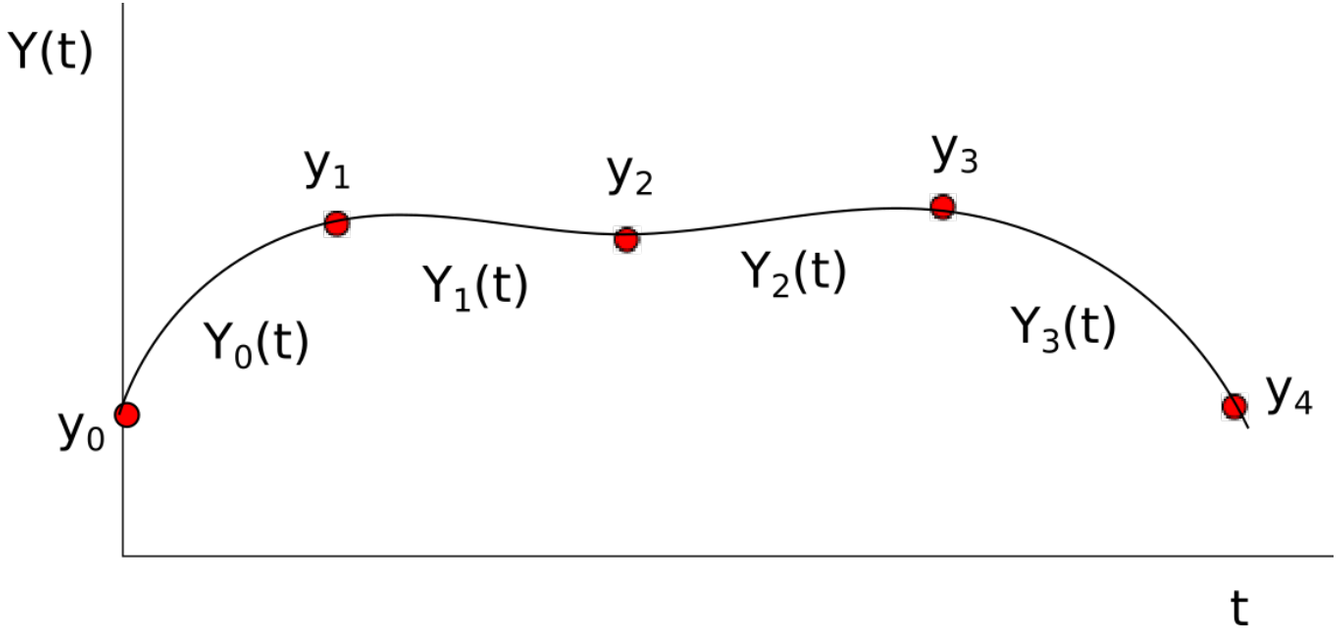
Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.
- Probabilistic completeness: if nodes number reaches infinity than you find a feasible path (if exists) with $P = 1$.
- if nodes number reaches infinity then RRT* find the optimal path

Properties of Rapidly Exploring Random Tree

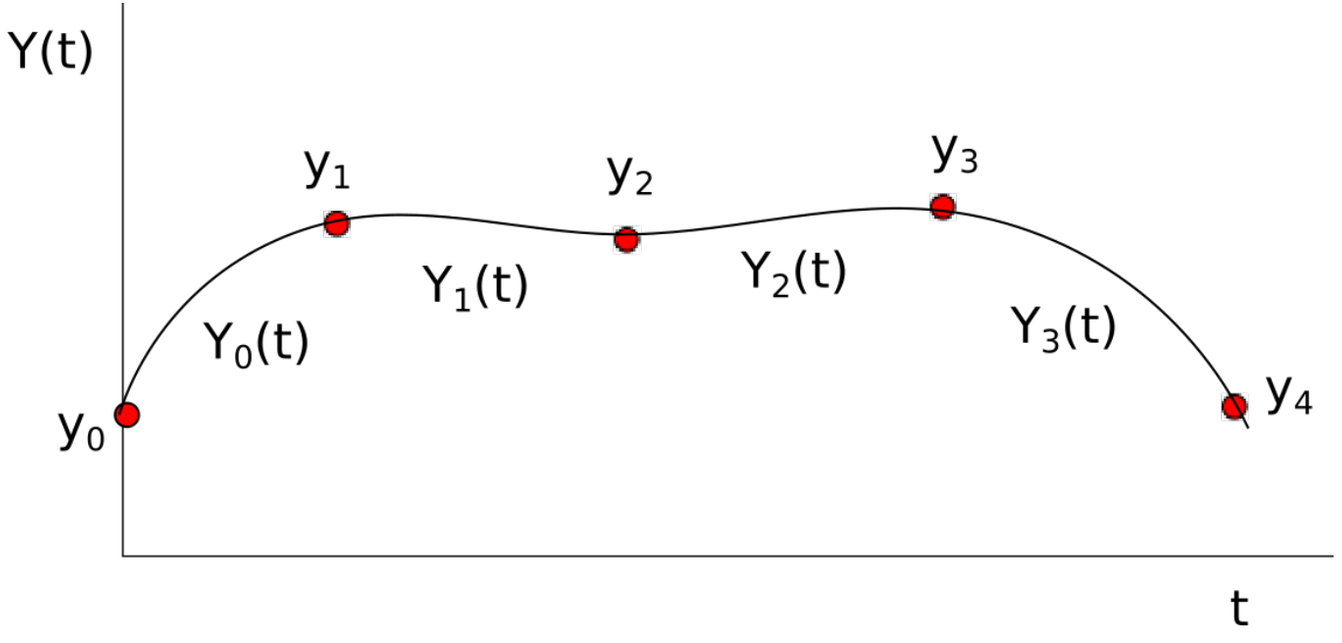
- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.
- Probabilistic completeness: if nodes number reaches infinity than you find a feasible path (if exists) with $P = 1$.
- if nodes number reaches infinity then RRT* find the optimal path
- if accurate dynamic motion model is known (i.e. $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$ relations are determined), then RRT searches for trajectory in lifted state-space $[\mathbf{x},\ \dot{\mathbf{x}},\ \ddot{\mathbf{x}}]^\top$.
- if only kinematic model is available we have to generate smooth trajectory

Smooth path interpolation



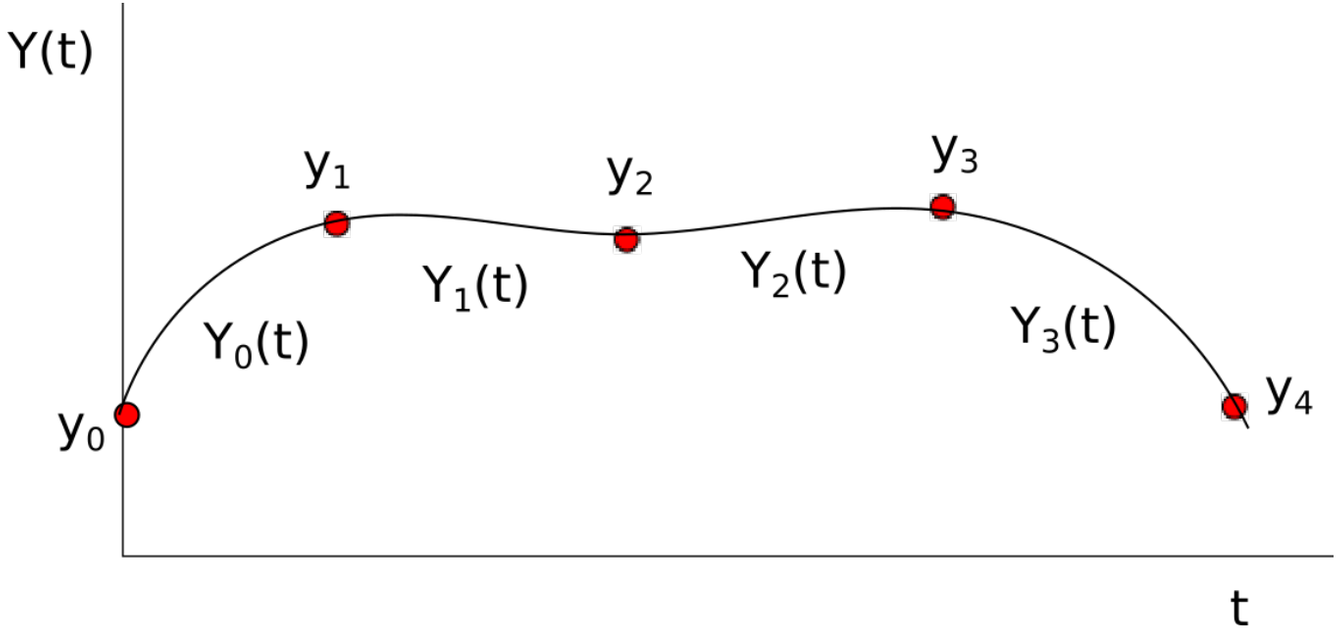
- Let us have a path given by N points in state-space $\mathbf{y}_0 \in X, \ldots \mathbf{y}_{N-1} \in X$.

Smooth path interpolation



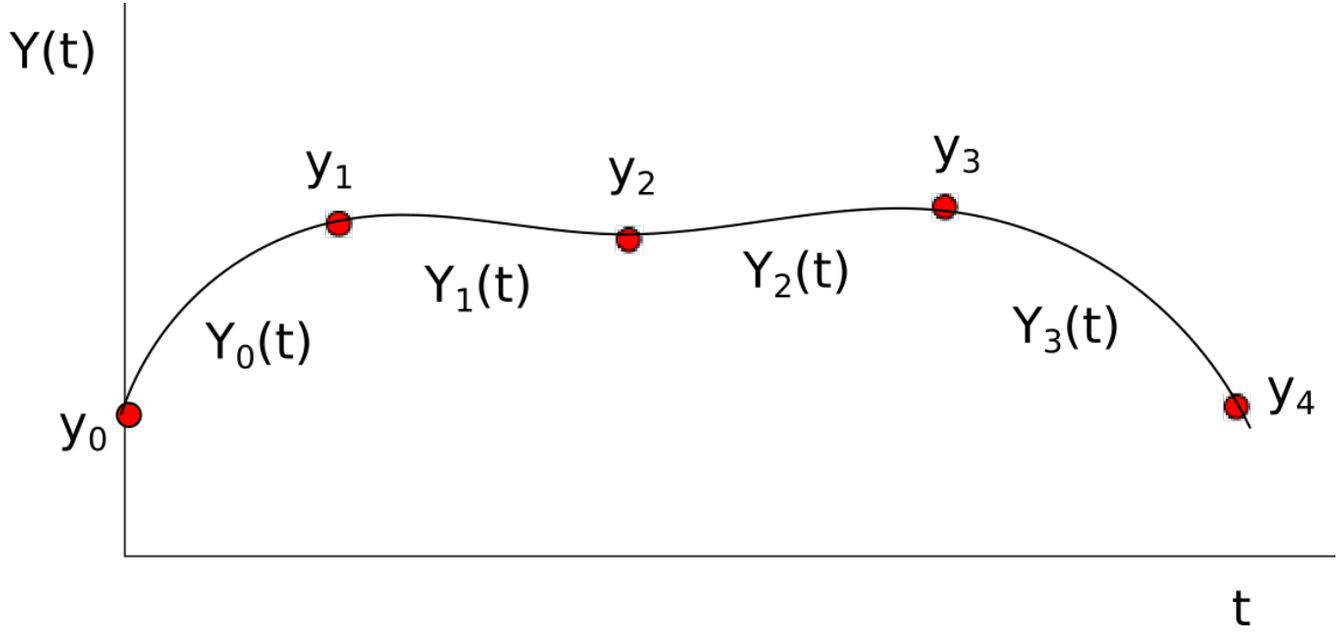
- Let us have a path given by N points in state-space $\mathbf{y}_0 \in X, \ldots \mathbf{y}_{N-1} \in X$.
- We define trajectory as pair-wise cubic function Y, i.e sequence of $N-1$ cubic functions $Y_i(t) : [0,1] \to X$ such that:
  - $Y_i(t)$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$.
  - Y has continuous first and second derivatives.

Smooth path interpolation



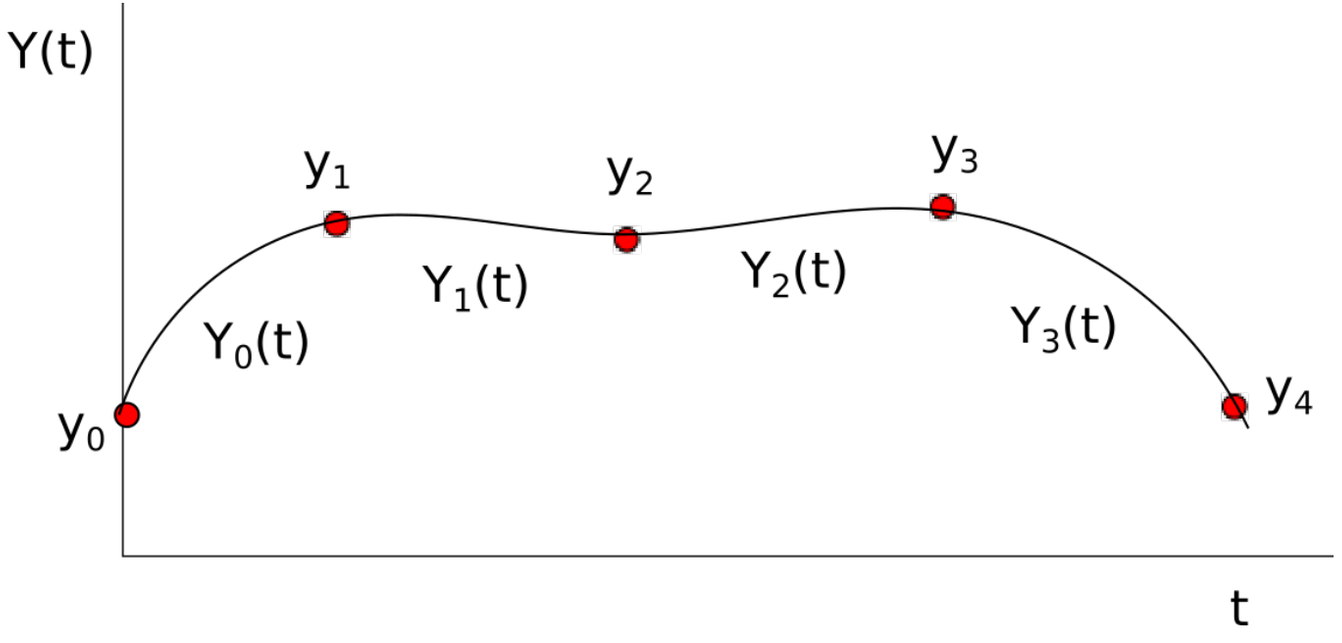
- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ - How many unknowns??

Smooth path interpolation



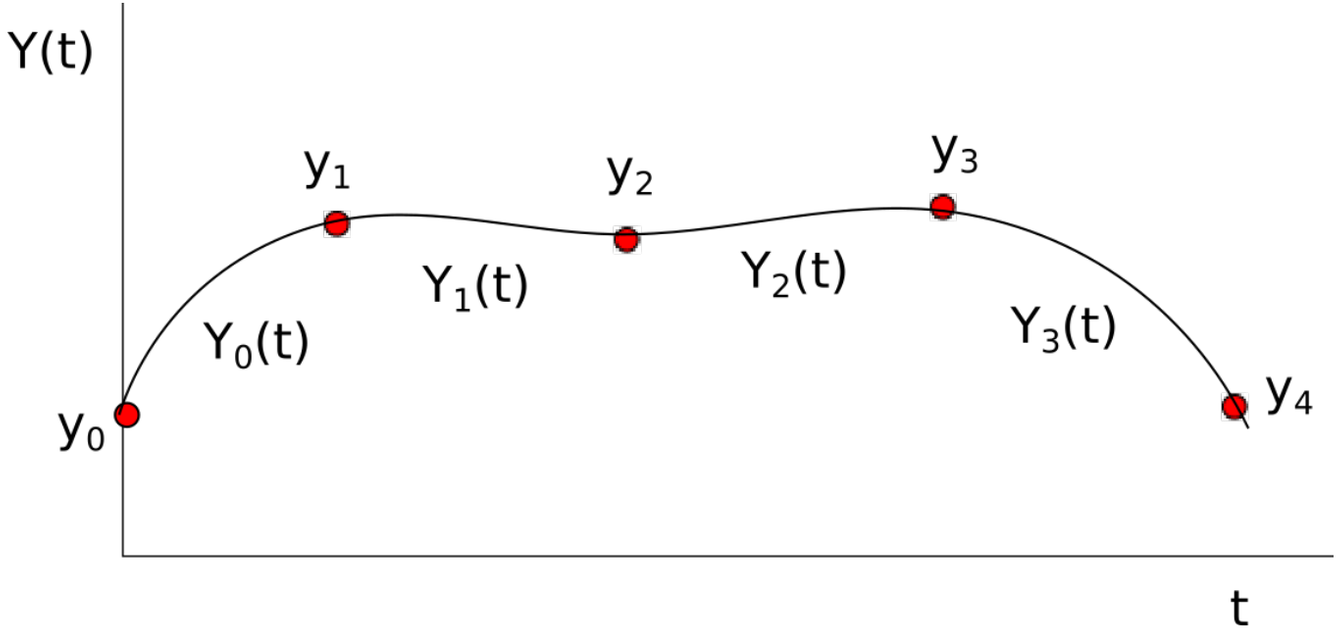
- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ - How many unknowns??
  - $4(N-1)$ unknowns.

Smooth path interpolation



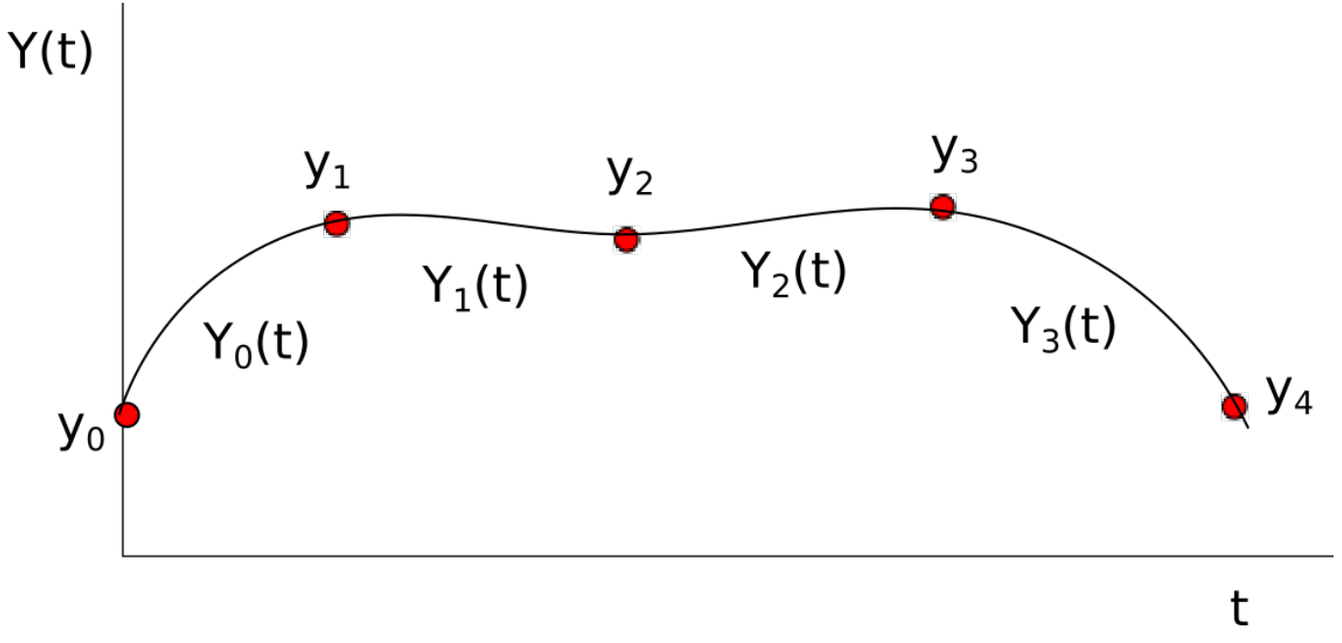
- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ - How many unknowns??
  - $4(N-1)$ unknowns.
- $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$ - How many eqs??

Smooth path interpolation



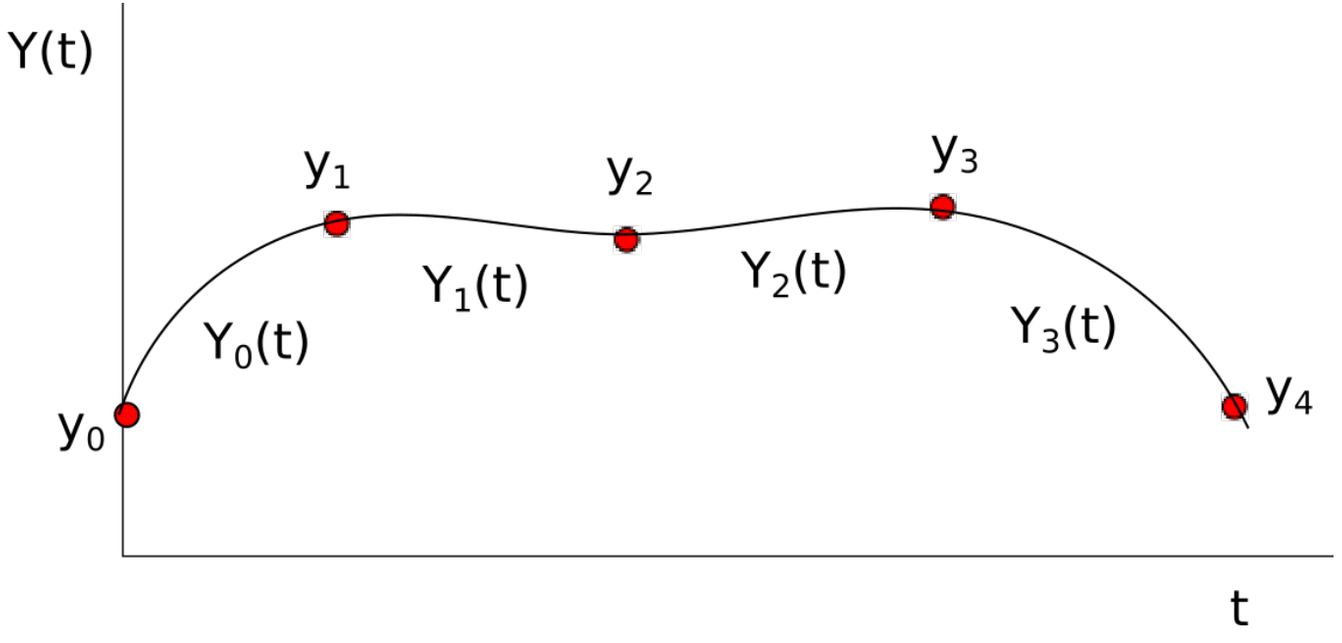
- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ - How many unknowns??
  - $4(N-1)$ unknowns.
- $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$ - How many eqs??
  - $Y_i(0) = a_i = \mathbf{y}_i$
  - $Y_i(1) = b_i + c_i + d_i = \mathbf{y}_{i+1}$
  - $2 \times (N-1)$ linear equations.

Smooth path interpolation



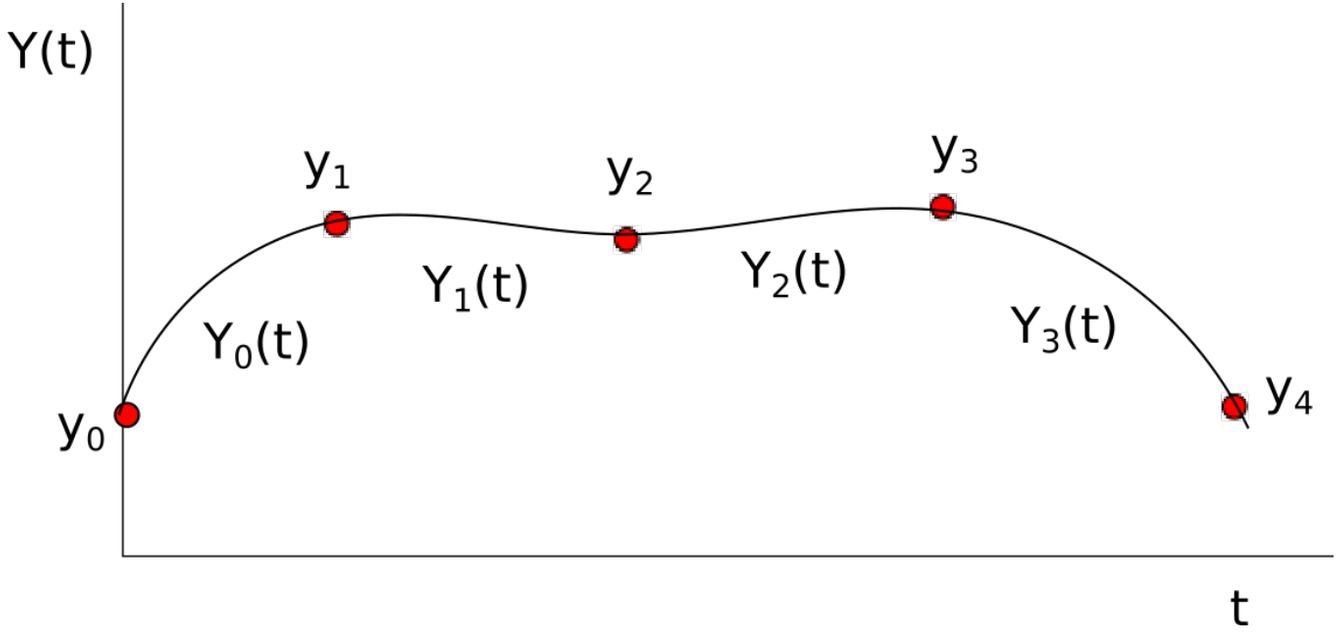
- has continuous first derivative $\frac{\partial Y_i(t)}{\partial t} = b_i + 2c_i t + 3d_i t^2$ - How many eqs??

Smooth path interpolation



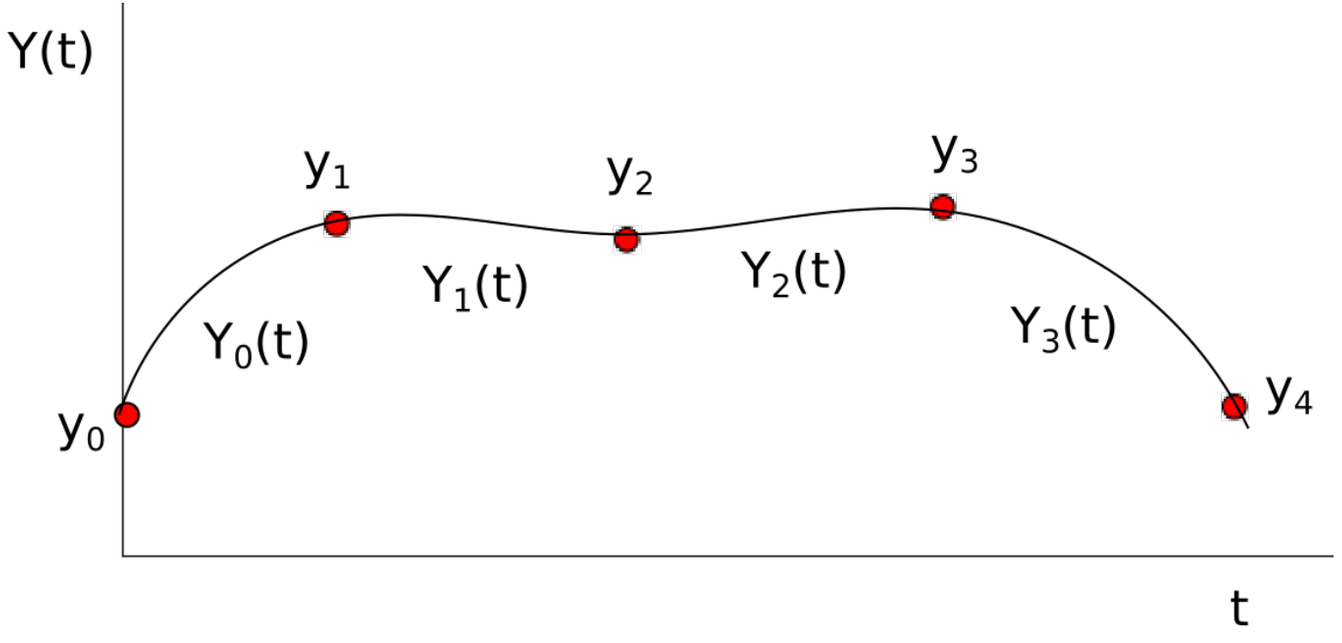
- has continuous first derivative $\frac{\partial Y_i(t)}{\partial t} = b_i + 2c_i t + 3d_i t^2$ - How many eqs??
  - $\frac{\partial Y_i(1)}{\partial t} = \frac{\partial Y_{i+1}(0)}{\partial t} \quad \Rightarrow \quad 2c_i + 3d_i = b_{i+1}$
  - $(N-1)$ linear equations.

Smooth path interpolation



- has continuous second derivative $\frac{\partial^2 Y_i(t)}{\partial t^2} = 2c_i + 6d_i t$ - How many eqs??

Smooth path interpolation



- has continuous second derivative $\frac{\partial^2 Y_i(t)}{\partial t^2} = 2c_i + 6d_i t$ - How many eqs??
  - $\frac{\partial^2 Y_i(1)}{\partial t^2} = \frac{\partial^2 Y_{i+1}(0)}{\partial t^2} \quad \Rightarrow \quad 6d_i = 2c_{i+1}$
  - $(N-1)$ linear equations.

Smooth path interpolation

- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$
  - $4 \times (N-1)$ unknowns.

Smooth path interpolation

- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$
  - $4 \times (N-1)$ unknowns.
- $Y_i(t)$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$.
  - $2 \times (N-1)$ linear equations.

Smooth path interpolation

- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$
  - $4 \times (N-1)$ unknowns.
- $Y_i(t)$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$.
  - $2 \times (N-1)$ linear equations.
- has continuous first and second derivative
  - $2 \times (N-1)$ linear equations.

Smooth path interpolation

- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$
  - $4 \times (N-1)$ unknowns.
- $Y_i(t)$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$.
  - $2 \times (N-1)$ linear equations.
- has continuous first and second derivative
  - $2 \times (N-1)$ linear equations.
- total number of equations is ??

Smooth path interpolation

- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$
  - $4 \times (N-1)$ unknowns.
- $Y_i(t)$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$.
  - $2 \times (N-1)$ linear equations.
- has continuous first and second derivative
  - $2 \times (N-1)$ linear equations.
- total number of equations is ??
  - $4 \times (N-1) - 2$ linear equations - What is missing ??

Smooth path interpolation

- $(N-1)$ cubics: $Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$
  - $4 \times (N-1)$ unknowns.
- $Y_i(t)$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$.
  - $2 \times (N-1)$ linear equations.
- has continuous first and second derivative
  - $2 \times (N-1)$ linear equations.
- total number of equations is ??
  - $4 \times (N-1) - 2$ linear equations - What is missing ??
- last two equation comes from requirements on initial and final speed, e.g.
  - $\frac{\partial Y_0(0)}{\partial t} = \frac{\partial Y_{N-1}(1)}{\partial t} = 0$

What if motion model is unknown

- Linear equations are re-arranged into $4(N-1) \times 4(N-1)$ matrix $\mathbf{A}$ and $4(N-1)$ dimensional vector right-hand side vector $\mathbf{b}$.
- Solution is given by inversion of trdiagonal matrix.

What if motion model is unknown

- Linear equations are re-arranged into $4(N-1) \times 4(N-1)$ matrix $\mathbf{A}$ and $4(N-1)$ dimensional vector right-hand side vector $\mathbf{b}$.
- Solution is given by inversion of trdiagonal matrix.
- Can you design a motion control algorithm without motion model?

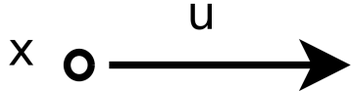
MDP definition

- States: $\mathbf{x} \in X$

x o

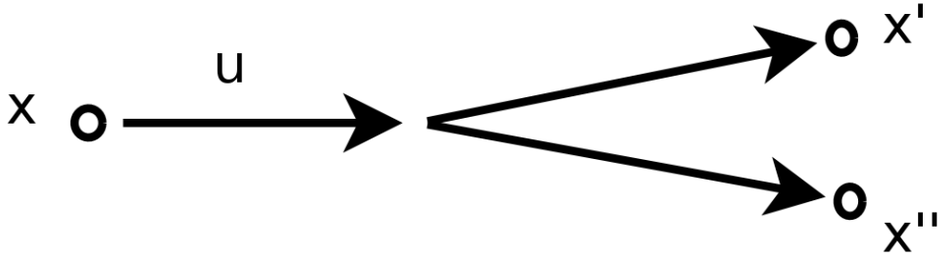
MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$



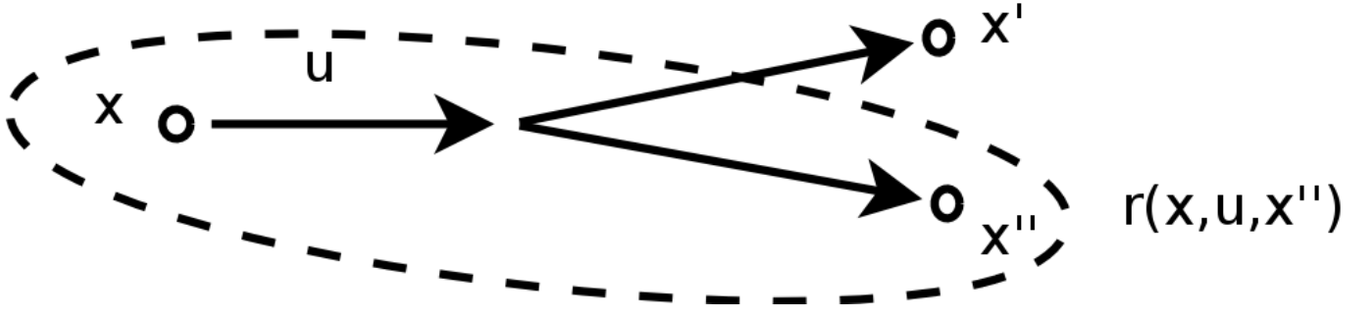
MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$
- Transition probability: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) : X \times U \times X \to [0; 1]$



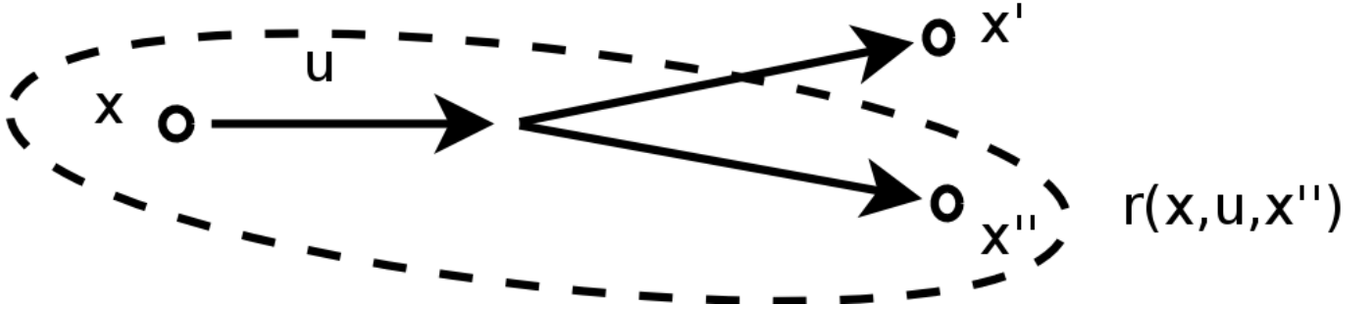
MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$
- Transition probability: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) : X \times U \times X \to [0; 1]$
- Reward: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') : X \times U \times X \to \mathbb{R}$



MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$
- Transition probability: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) : X \times U \times X \to [0; 1]$
- Reward: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') : X \times U \times X \to \mathbb{R}$
- Policy: $\pi_\theta(\mathbf{x}) : X \to U$



MDP definition

- Trajectory is sequence of visited states and performed actions:
  $\tau = (\mathbf{x}_0, \mathbf{u}_0, \ \mathbf{x}_1, \mathbf{u}_1, \ \mathbf{x}_2, \dots)$

MDP definition

- Trajectory is sequence of visited states and performed actions:
  $\tau = (\mathbf{x}_0, \mathbf{u}_0, \ \mathbf{x}_1, \mathbf{u}_1, \ \mathbf{x}_2, \dots)$
- Sum of rewards with limited horizont:

$$r(\tau) = \sum_{i=0}^{H} r(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{i+1})$$

MDP definition

- Trajectory is sequence of visited states and performed actions:
  $\tau = (\mathbf{x}_0, \mathbf{u}_0, \ \mathbf{x}_1, \mathbf{u}_1, \ \mathbf{x}_2, \dots)$
- Sum of rewards with limited horizont:

$$r(\tau) = \sum_{i=0}^{H} r(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{i+1})$$

- Sum of discounted rewards:

$$r(\tau) = \sum_{i=0}^{\infty} \gamma^i \cdot r(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{i+1})$$

Problem definition

- We have a robot and we have no idea how to control it.

Problem definition

- We have a robot and we have no idea how to control it.
- Nevertheless, we know what is good and bad state - we have a definition of rewards.

Problem definition

- We have a robot and we have no idea how to control it.
- Nevertheless, we know what is good and bad state - we have a definition of rewards.
- We control it somehow (e.g. with some initial policy) and record the trajectory τ (or several trajectories).

Problem definition

- We have a robot and we have no idea how to control it.
- Nevertheless, we know what is good and bad state - we have a definition of rewards.
- We control it somehow (e.g. with some initial policy) and record the trajectory τ (or several trajectories).
- Given these trajectories, change the policy to increase expected sum of rewards

$$J(\theta) = E\{r(\tau)\}$$

Problem definition

- Denote $p(\tau|\theta)$ probability of trajectory τ occurs when following policy π_θ

Problem definition

- Denote $p(\tau|\theta)$ probability of trajectory τ occurs when following policy π_θ
- Criterion to be maximized is the expected sum of rewards

$$J(\theta) = E\{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau) \, \mathrm{d}\tau$$

Problem definition

- Denote $p(\tau|\theta)$ probability of trajectory τ occurs when following policy π_θ
- Criterion to be maximized is the expected sum of rewards

$$J(\theta) = E\{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau) \, \mathrm{d}\tau$$

- We solve the following optimization problem

$$\theta^* = \arg\max_\theta J(\theta)$$

Problem solution

- As usually, you can:

Problem solution

- As usually, you can:
  - either solve **primal task** e.g. by following gradient ∇J to maximize $J(\theta)$ directly.
  - primal is often solved in the optimal control community (e.g. LQR),

Problem solution

- As usually, you can:
  - either solve **primal task** e.g. by following gradient ∇J to maximize $J(\theta)$ directly.
  - primal is often solved in the optimal control community (e.g. LQR),
  - or solve **dual task** to find state-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$ which tells expected sum rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
  - optimal policy $\pi^* = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$
  - dual is often employed by AI community as heuristics for state-space search
  - Q-values could provide metrics for RRT [Tedrake-LQR-trees-2015]

Dual task provides alternative point-of-view (e.g. shadow prices in LP or sparse feature selection for SVM)

Primal task - approximating criterion.

- Use $\pi_\theta(\mathbf{x})$ to get several trajectories τ_i.

Primal task - approximating criterion.

- Use $\pi_\theta(\mathbf{x})$ to get several trajectories τ_i.
- Approximate criterion value in θ as average reward of trajectories $\tau_i \sim p(\tau|\theta)$ generated with policy π_θ

$$J(\theta) = E\{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau)\, \mathrm{d}\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$$

Primal task - approximating criterion.

- Use $\pi_\theta(\mathbf{x})$ to get several trajectories τ_i.
- Approximate criterion value in θ as average reward of trajectories $\tau_i \sim p(\tau|\theta)$ generated with policy π_θ

$$J(\theta) = E\{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau) \, \mathrm{d}\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$$

- We can approximate criterion value, what about gradient?

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?
- Of course, but doing it from one sample is quite unstable (especially for high dimensional θ).

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?
- Of course, but doing it from one sample is quite unstable (especially for high dimensional θ).
- Perform several small random perturbations $\Delta\theta_i$ and compute $J(\theta + \Delta\theta_i)$.

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?
- Of course, but doing it from one sample is quite unstable (especially for high dimensional θ).
- Perform several small random perturbations $\Delta\theta_i$ and compute $J(\theta + \Delta\theta_i)$.
- Relation to gradient $\nabla J(\theta)$ is given by the first order Taylor polynom

$$
\begin{aligned}
J(\theta + \Delta\theta_i) &= J(\theta) + \nabla J(\theta)^\top \Delta\theta_i \\
\Delta\theta_i^\top \nabla J(\theta) &= J(\theta) - J(\theta + \Delta\theta_i) \\
\underbrace{\begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}}_{\text{matrix } \mathbf{A}} \nabla J(\theta) &= \underbrace{\begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}}_{\text{vector } \mathbf{b}}
\end{aligned}
$$

Primal task - solution

- Gradient is solution of overdetermined set of linear equations:

$$\nabla J(\theta) = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}$$

Primal task - solution

- Gradient is solution of overdetermined set of linear equations:

$$\nabla J(\theta) = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}$$

- Algorithm is simple:
  - Randomly initialize θ
  - Use $\pi_\theta(\mathbf{x})$ to get trajectories.
  - Compute $\nabla J(\theta)$ using pseudo-inverse.
  - Update $\theta \leftarrow \theta + \alpha \frac{\nabla J(\theta)}{\|\nabla J(\theta)\|}$

Primal task - pros and cons

- No motion model required.

Primal task - pros and cons

- No motion model required.
- Converges to local minima - good initialization needed (e.g. imitation learning).

Primal task - pros and cons

- No motion model required.
- Converges to local minima - good initialization needed (e.g. imitation learning).
- High-dimensional θ requires many real-world samples.
- There are better gradient approximations - see survey [2]

[2] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters, A Survey on Policy Search for Robotics, NOW, 2013

Primal task - likelihood ratio trick

- Let us introduce stochastic policy:

$$\pi_\theta(\mathbf{u}|\mathbf{x}) : X \times U \to [0; 1]$$

- For example: $\pi_\theta(\mathbf{u}|\mathbf{x}) \sim \mathcal{N}(\theta^\top \mathbf{x}, \Sigma)$

Primal task - likelihood ratio trick

- Let us introduce stochastic policy:

$$\pi_\theta(\mathbf{u}|\mathbf{x}) : X \times U \to [0; 1]$$

- For example: $\pi_\theta(\mathbf{u}|\mathbf{x}) \sim \mathcal{N}(\theta^\top \mathbf{x}, \Sigma)$
- Gradient of the expected sum of rewards remains:

$$\nabla_\theta J(\theta) = \int_T \nabla_\theta p(\tau|\theta) r(\tau) d\tau$$

- Problem is that $\nabla_\theta p(\tau|\theta)$ is gradient of unknown probability distribution.

Primal task - likelihood ratio trick

- Let us introduce stochastic policy:

$$\pi_\theta(\mathbf{u}|\mathbf{x}) : X \times U \to [0; 1]$$

- For example: $\pi_\theta(\mathbf{u}|\mathbf{x}) \sim \mathcal{N}(\theta^\top \mathbf{x}, \Sigma)$
- Gradient of the expected sum of rewards remains:

$$\nabla_\theta J(\theta) = \int_T \nabla_\theta p(\tau|\theta) r(\tau) d\tau$$

- Problem is that $\nabla_\theta p(\tau|\theta)$ is gradient of unknown probability distribution.
- Using likelihood ratio trick we rewrite it as follows:

$$\nabla_\theta p(\tau|\theta) = p(\tau|\theta) \nabla_\theta \log p(\tau|\theta)$$

Primal task - likelihood ratio trick

- and substitute it back to $\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta) = \int_T p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) r(\tau) d\tau =$$

Primal task - likelihood ratio trick

- and substitute it back to $\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta) = \int_T p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) r(\tau) d\tau =$$

$$= E[\nabla_\theta \log p(\tau|\theta) r(\tau)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(\tau_i|\theta) r(\tau_i)$$

Primal task - likelihood ratio trick

- and substitute it back to $\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta) = \int_T p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) r(\tau) d\tau =$$

$$= E[\nabla_\theta \log p(\tau|\theta) r(\tau)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(\tau_i|\theta) r(\tau_i)$$

- Why?

Primal task - likelihood ratio trick

- and substitute it back to $\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta) = \int_T p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) r(\tau) d\tau =$$

$$= E[\nabla_\theta \log p(\tau|\theta) r(\tau)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(\tau_i|\theta) r(\tau_i)$$

- Why? Because $\nabla_\theta \log p(\tau|\theta)$ can be expressed using known $\pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$:

$$p(\tau|\theta) = p(\mathbf{x_0}) \prod_k p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) \pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$$

$$\nabla_\theta \log p(\tau|\theta) = \nabla_\theta [\, \log p(\mathbf{x_0}) + \sum_k \log p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) + \sum_k \log \pi_\theta(\mathbf{u}_k|\mathbf{x}_k) \,]$$

$$= \sum_k \nabla_\theta \log \pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$$

Primal task - likelihood ratio trick

- Collect N trajectories of length M:

$$\tau_1 = [(\mathbf{u}_{1,1}, \mathbf{x}_{1,1}) \dots \mathbf{u}_{M,1}, \mathbf{x}_{M,1})]$$

$$\vdots$$

$$\tau_N = [(\mathbf{u}_{1,N}, \mathbf{x}_{1,N}) \dots \mathbf{u}_{M,N}, \mathbf{x}_{M,N})]$$

- Compute gradient (main result):

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{M} \nabla_\theta \log \pi_\theta(\mathbf{u}_{k,i} | \mathbf{x}_{k,i})$$

- update θ according to the gradient and repeat until convergence.

What you can do?

- Work with us on:
  - real Search&Rescue platform
  - better IRO tasks
- TORCS - Racing **and demolishon derby** simulator competition.
  http://en.wikipedia.org/wiki/TORCS
- Starcraft competition
  http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/