EE211: Robotic Perception and Intelligence

Lecture 6 Advanced Planning Methods

Jiankun WANG

Department of Electronic and Electrical Engineering
Southern University of Science and Technology

Undergraduate Course, Nov 2024

Outline

Outline

RRT Algorithm

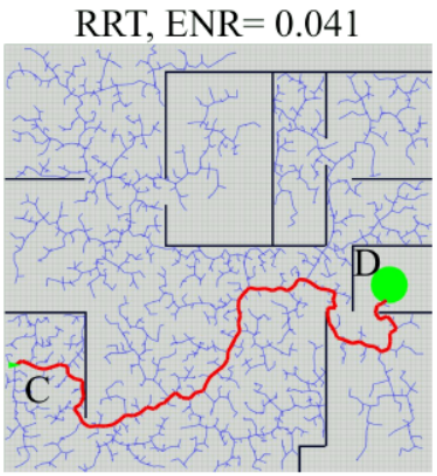
- The RRT algorithm has been shown to be probabilistically complete.
- The probability of success (if the problem is feasible) goes to 1 exponentially fast , if the environment satisfies certain "good visibility" conditions.

Algorithm 1: RRT

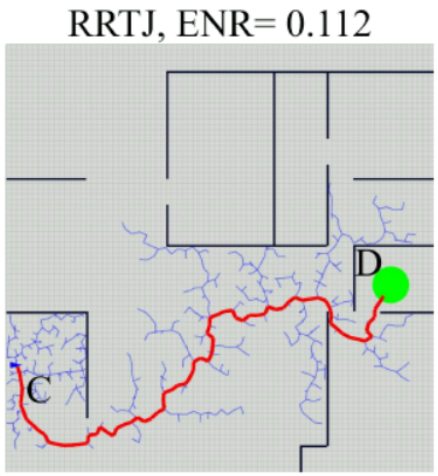
$V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$;
forall $i = 1, ..., n$ **do**
  $x_{rand} \leftarrow \text{SampleFree}_i$;
  $x_{nearest} \leftarrow \text{Nearest}(G = (V, E), x_{rand})$;
  $x_{new} \leftarrow \text{Steer}(x_{nearest}, x_{rand})$;
  if $\mathit{CollisonFree}(x_{nearest}, x_{new})$ **then**
    $V \leftarrow V \cup \{x_{new}\}$;
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$;
return $G = (V, E)$;

RRTs and Optimality

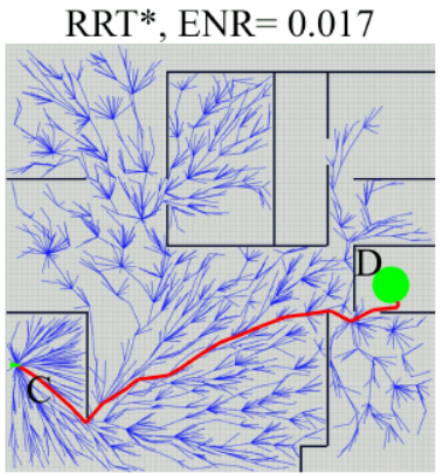
- RRTs are great at finding feasible trajectories quickly...
- However, RRTs are terrible at finding good trajectories, regardless of the obstacle configuration!



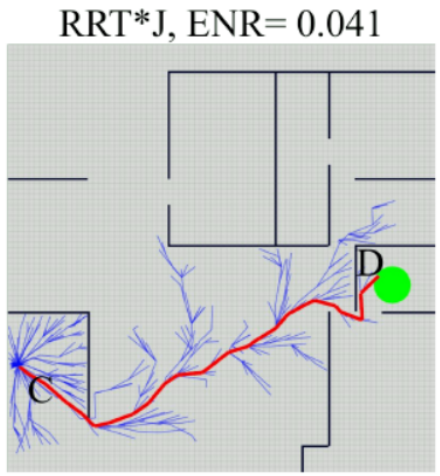
(e) Time = 165ms, Node = 2379.



(f) Time = 45ms, Node = 602.



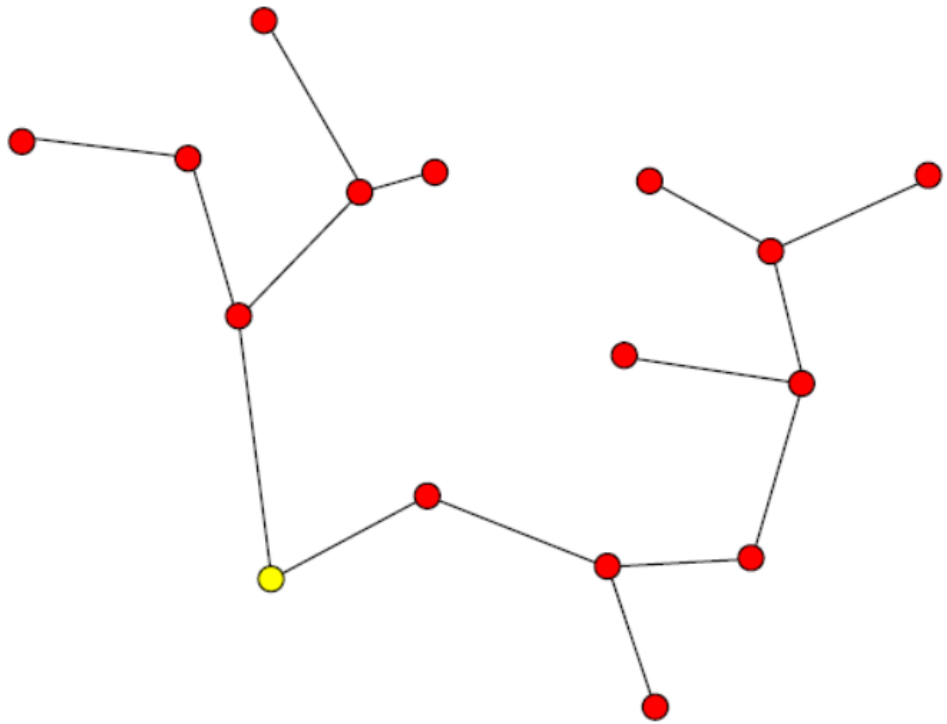
(g) Time = 254ms, Node = 2007.



(h) Time = 41ms, Node = 403.

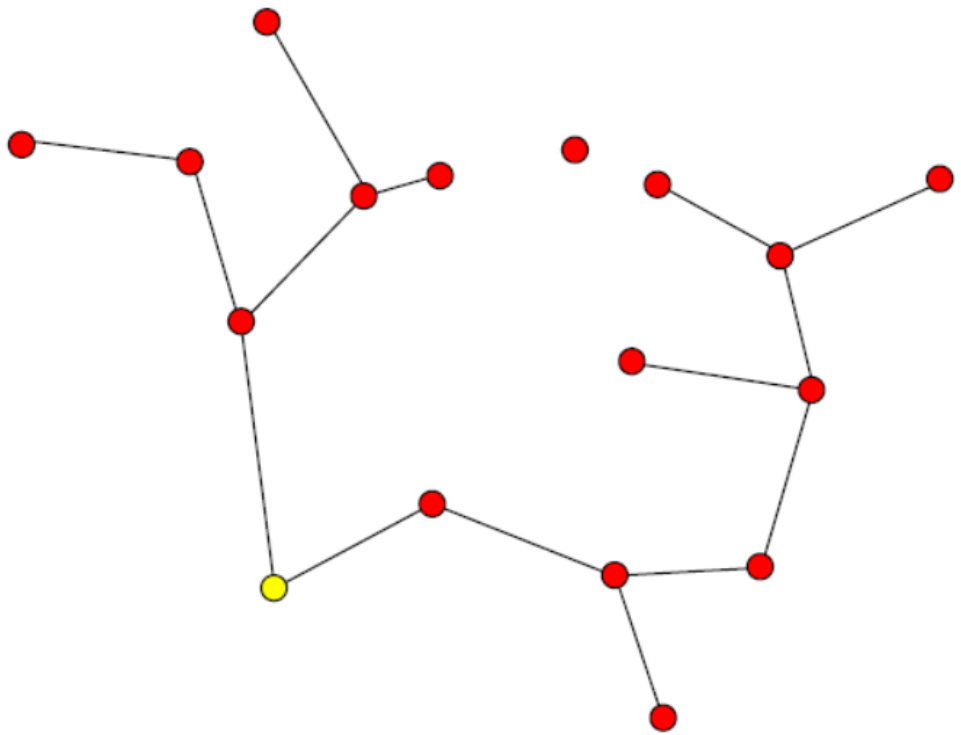
RRT* - Step 1

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



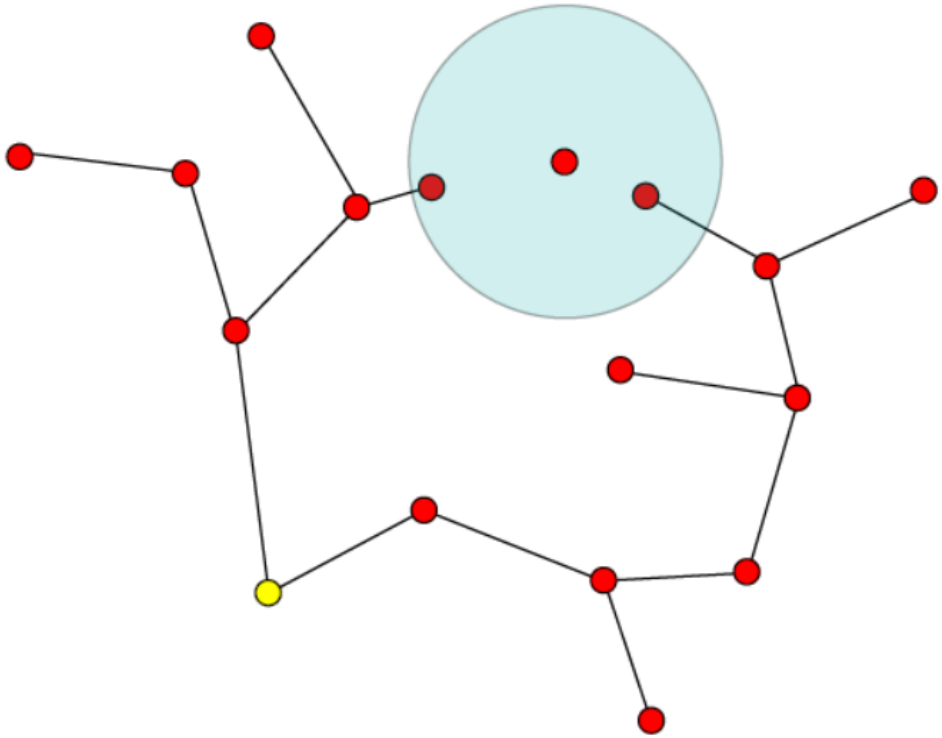
RRT* - Step 2

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



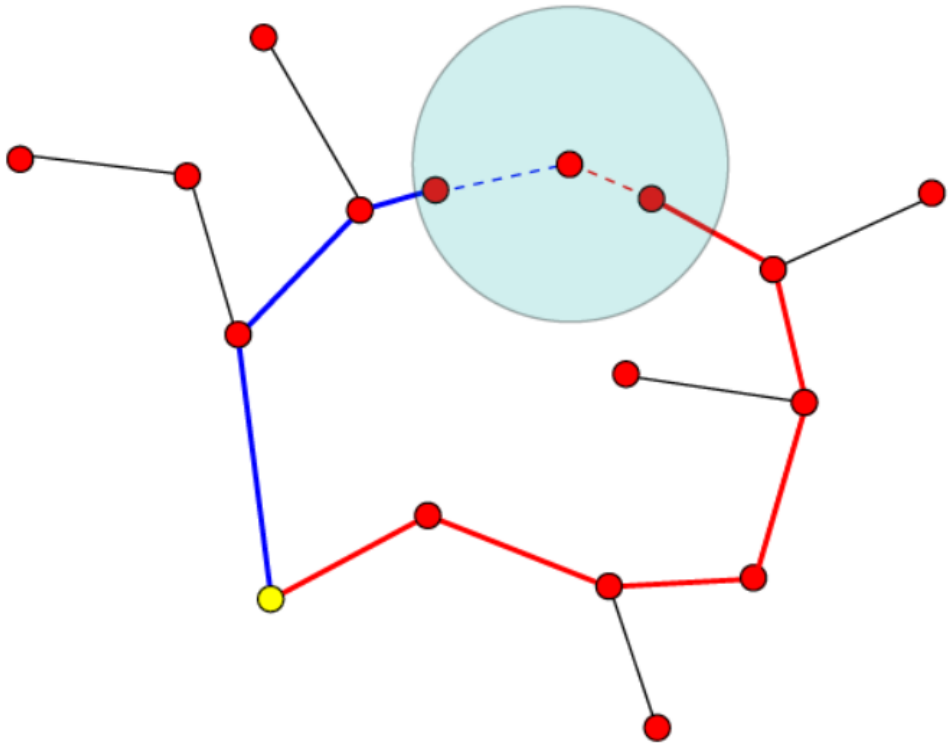
RRT* - Step 3

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



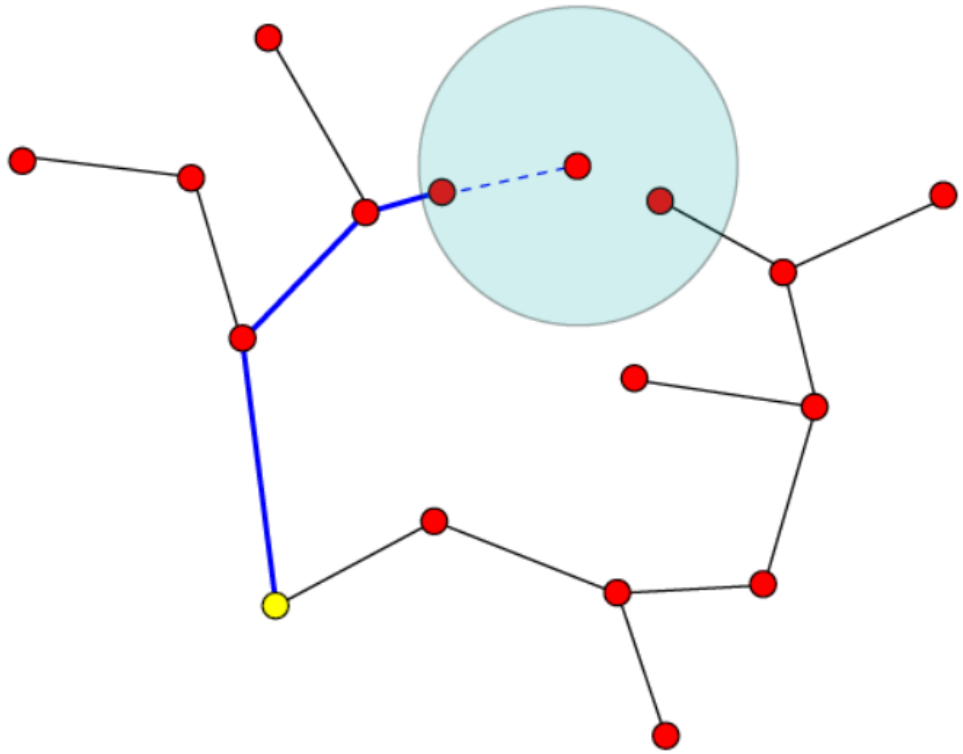
RRT* - Step 4

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



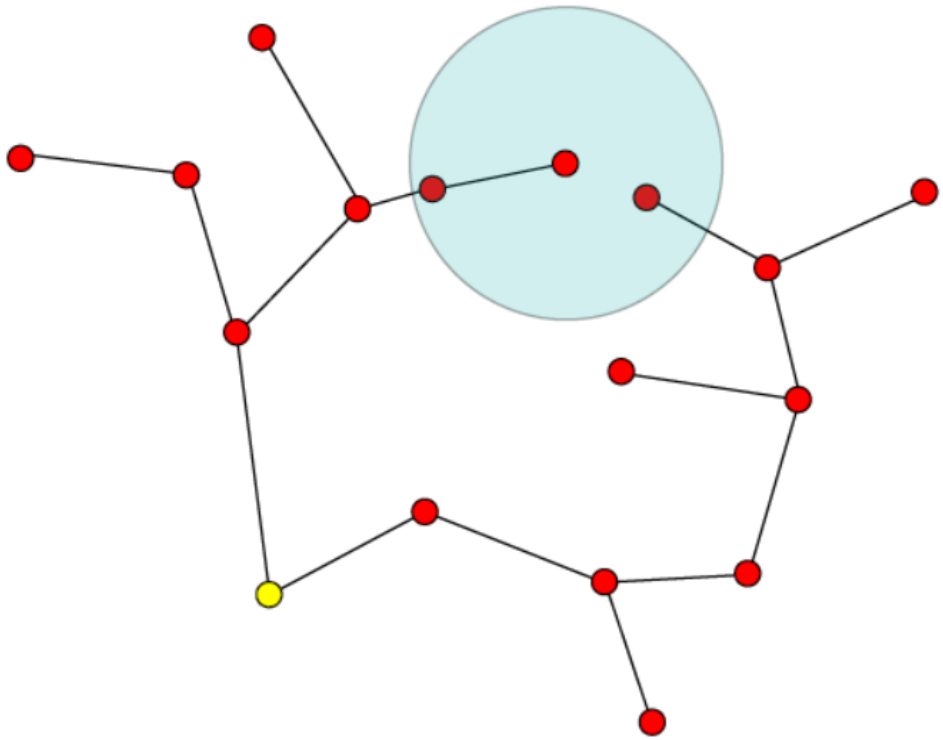
RRT* - Step 5

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



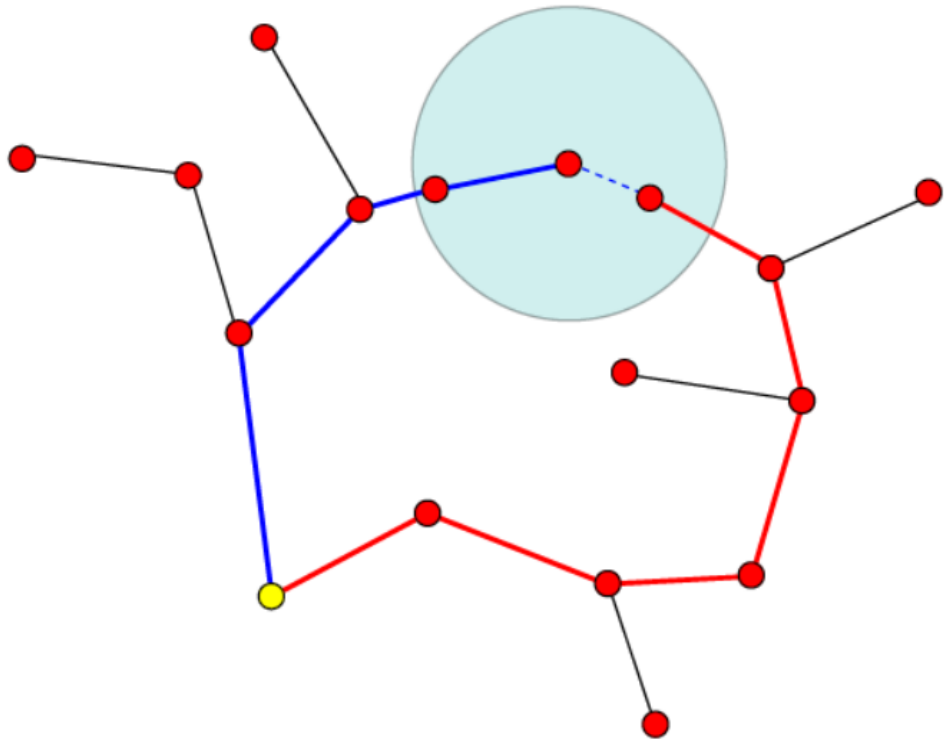
RRT* - Step 6

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



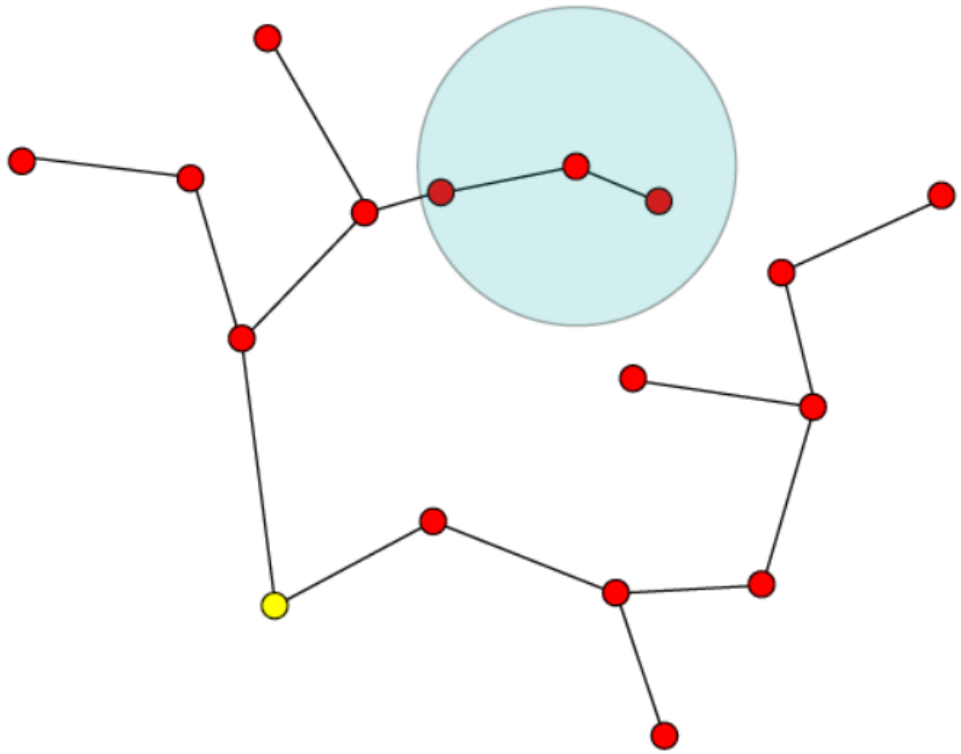
RRT* - Step 7

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



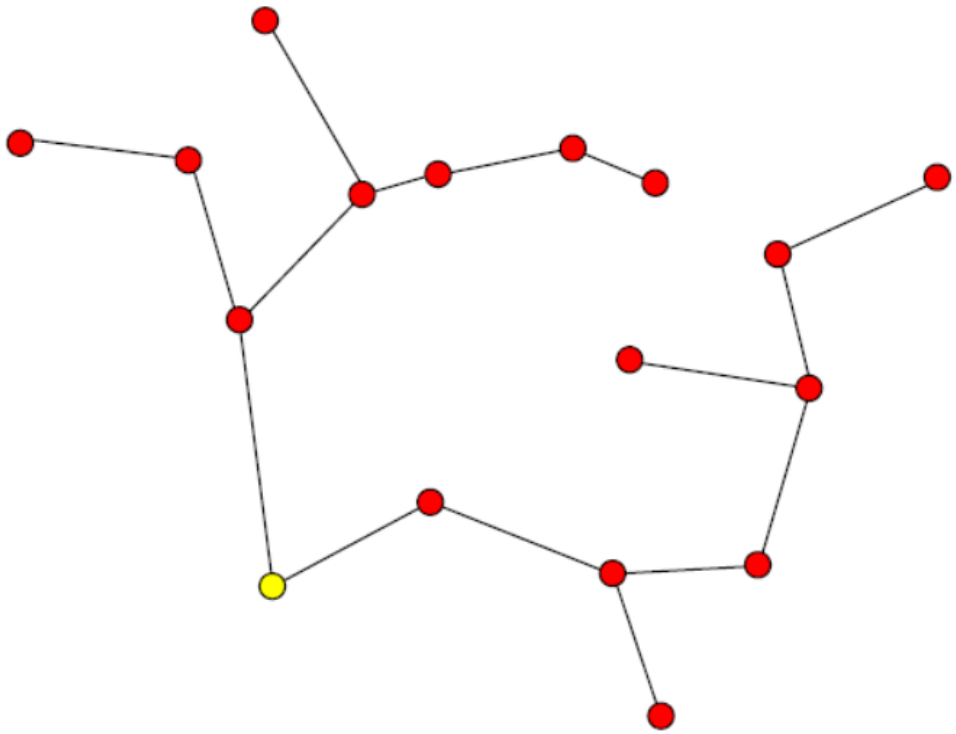
RRT* - Step 8

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.



RRT* - A Tree Version of The RRG: Step 9

- The RRT* algorithm maintains a tree structure by Choosing Parents, i.e., the optimal path to the new vertex, and Rewiring(eliminating "redundant" edges), i.e., edges that would be part of a non-optimal path to a vertex.

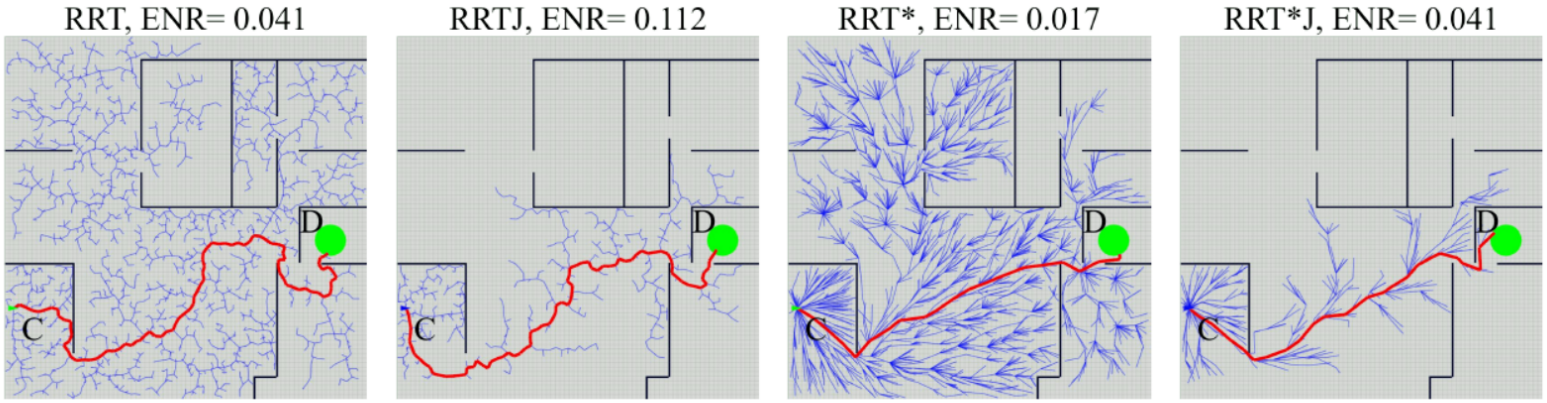


RRT* Algorithm

Algorithm 2: RRT*

$V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$; $G = (V, E)$;
forall $i = 1, ..., n$ **do**
  $x_{rand} \leftarrow \text{SampleFree}_i$; $x_{nearest} \leftarrow \text{Nearest}(G, x_{rand})$; $x_{new} \leftarrow \text{Steer}(x_{nearest}, x_{rand})$;
  if $CollisonFree(x_{nearest}, x_{new})$ **then**
    $X_{near} \leftarrow \text{Near}(G, x_{new}, \min\{\gamma_{RRG}(\log(card\ V)/card\ V)^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{new}\}$;
    $x_{min} \leftarrow x_{nearest}$; $c_{min} \leftarrow \text{Cost}(x_{nearest}) + \text{c}(\text{Line}(x_{nearest}, x_{new}))$;
    forall $x_{near} \in X_{near}$ **do**
      if $CollFree(x_{near}, x_{new})$ & $Cost(x_{near}) + c(Line(x_{near}, x_{new})) < c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$; $c_{min} \leftarrow \text{Cost}(x_{near}) + \text{c}(\text{Line}(x_{near}, x_{new}))$;
    $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
    forall $x_{near} \in X_{near}$ **do**
      if $CollFree(x_{new}, x_{near})$ & $Cost(x_{new}) + c(Line(x_{new}, x_{near})) < Cost(x_{near})$ **then**
        $x_{parent} \leftarrow \text{Parent}(x_{near})$;
        $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$;
return $G = (V, E)$;

RRT* Experiment Results



(e) Time = 165ms, Node = 2379. (f) Time = 45ms, Node = 602. (g) Time = 254ms, Node = 2007. (h) Time = 41ms, Node = 403.

Outline

Smoothing

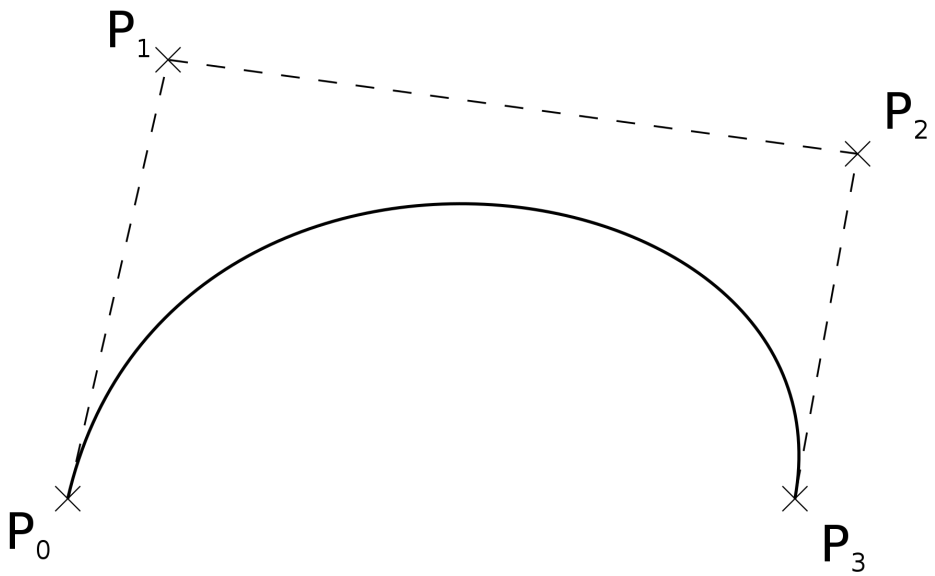
- The axis-aligned motions of a grid planner and the randomized motions of sampling planners may lead to jerky motion of a robot.
- Search globally for a solution, then post-process the resulting motion to make it smoother.
- Nonlinear Optimization: The initial motion must be converted to a parametrized representation of the controls. The following cost function

$$J = \frac{1}{2}\dot{u}^T(t)\dot{u}(t)dt,$$

  penalizes rapidly changing controls.
- Subdivide and Reconnect: A local planner can be used to attempt a connection between two distant points on a path.

Bézier Curve

- A Bézier curve is a parametric curve used in computer graphics and robotics.
- A set of discrete "control points" defines a smooth, continuous curve by means of a formula.



https://en.wikipedia.org/wiki/Bezier_curve

Bézier Curve

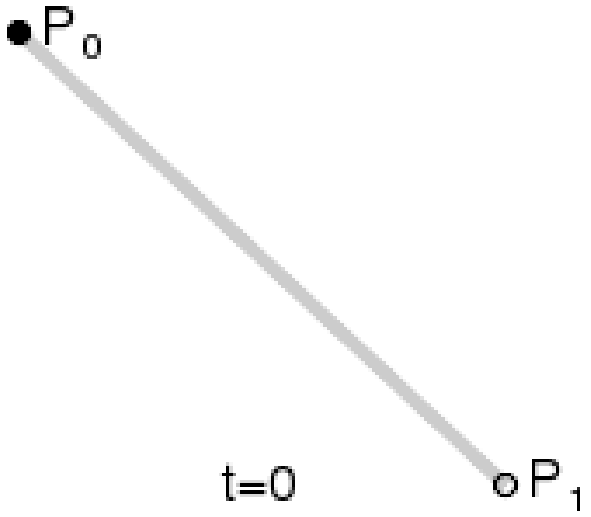
- A Bézier curve is defined by a set of control points P_0 through P_n, where n is called the order of the curve ($n = 1$ for linear, 2 for quadratic, 3 for cubic, etc.).
- The first and last control points are always the endpoints of the curve; however, the intermediate control points generally do not lie on the curve.
- The sums in the following sections are to be understood as affine combinations – that is, the coefficients sum to 1.

Linear Bézier Curve

- Given distinct points P_0 and P_1, a linear Bézier curve is simply a line between those two points. The curve is given by

$$B(t) = P_0 + t(P_1 - P_0) = (1 - t)P_0 + tP_1, \quad 0 \leq t \leq 1.$$

- It is equivalent to linear interpolation. The quantity $P_1 - P_0$ represents the displacement vector from the start point to the end point.



Quadratic Bézier Curve

- A quadratic Bézier curve is the path traced by the function $B(t)$, given points P_0, P_1, and P_2,

$$B(t) = (1-t)[(1-t)P_0 + tP_1] + t[(1-t)P_1 + tP_2], \quad 0 \leq t \leq 1,$$

  which can be interpreted as the linear interpolant of corresponding points on the linear Bézier curves from P_0 to P_1 and from P_1 to P_2 respectively.
- Rearranging the preceding equation yields:

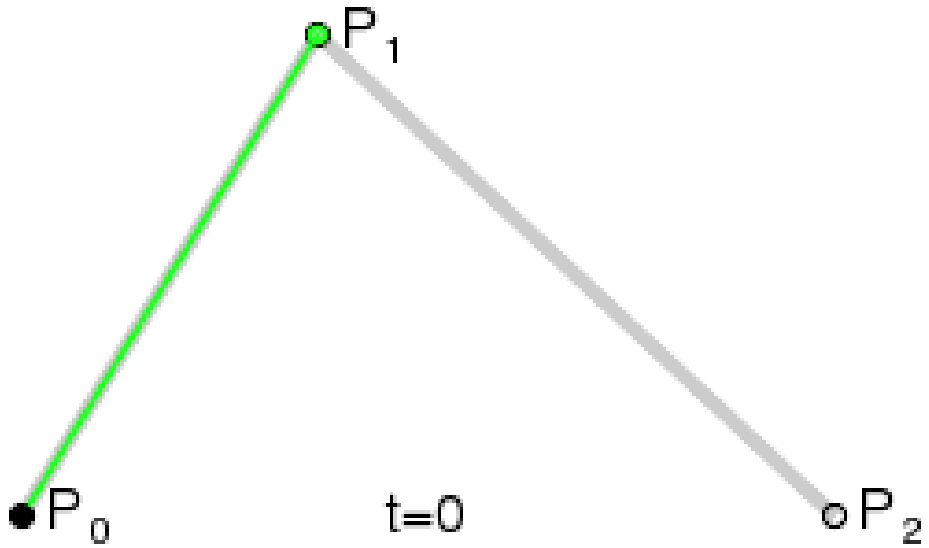
$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, \quad 0 \leq t \leq 1,$$

Quadratic Bézier Curve

- Which immediately gives the derivative of the Bézier curve with respect to t:

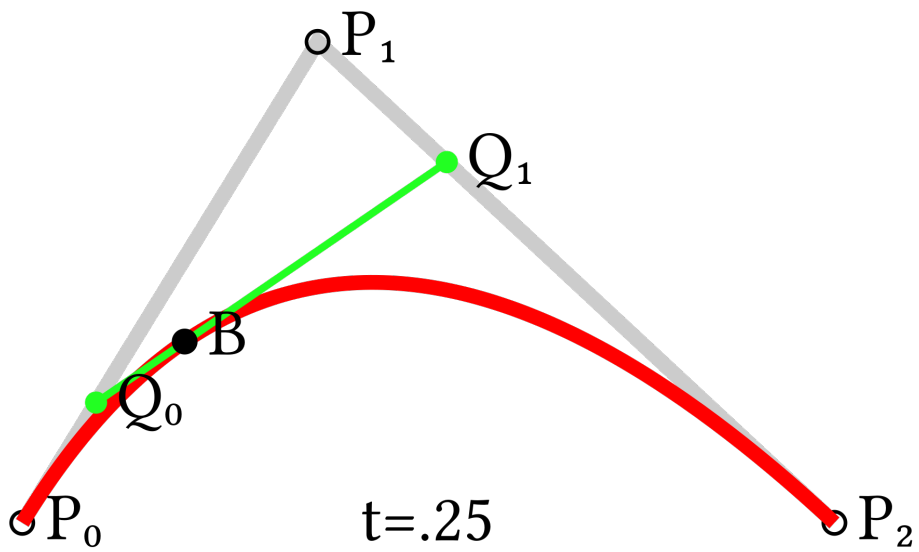
$$B'(t) = 2(1-t)[P_1 - P_0] + 2t[P_2 - P_1], \quad 0 \leq t \leq 1,$$

As t increases from 0 to 1, the curve departs from P_0 in the direction of P_1, then bends to arrive at P_2 from the direction of P_1.



Quadratic Bézier Curve

- As t increases from 0 to 1, the curve departs from P_0 in the direction of P_1, then bends to arrive at P_2 from the direction of P_1.
- Can be interpreted as the linear interpolant of corresponding points on the linear Bézier curves from P_0 to P_1 and from P_1 to P_2 respectively.



Cubic Bézier Curve

- The explicit form of the cubic Bézier curve is:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3, \quad 0 \leq t \leq 1.$$

- Any series of 4 distinct points can be converted to a cubic Bézier curve that goes through all 4 points in order.
- Given the starting and ending point of some cubic Bézier curve, and the points along the curve corresponding to $t = 1/3$ and $t = 2/3$, the control points for the original Bézier curve can be recovered.

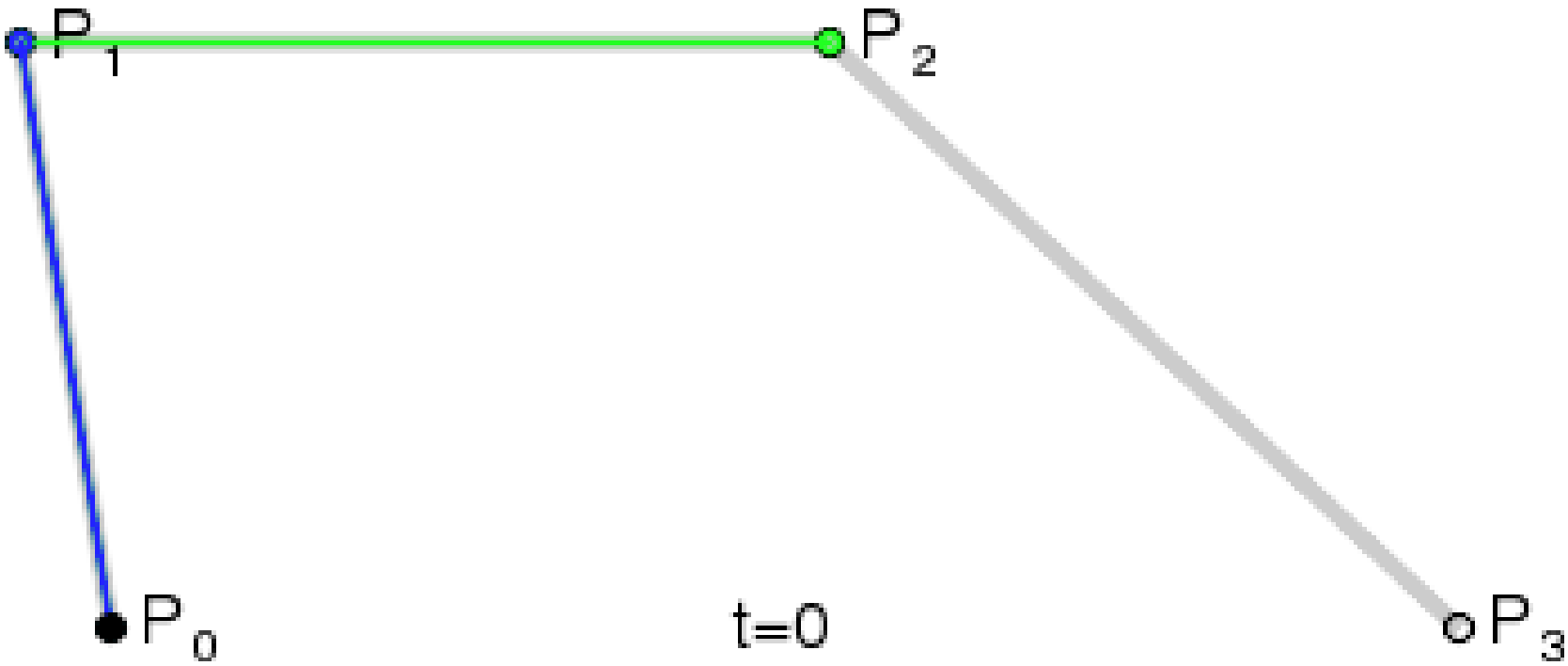
Cubic Bézier Curve

- The derivative of the cubic Bézier curve with respect to t is

$$B'(t) = 3(1-t)^2(P_1-P_0)+6(1-t)t(P_2-P_1)+3t^2(P_3-P_2), 0 \leq t \leq 1.$$

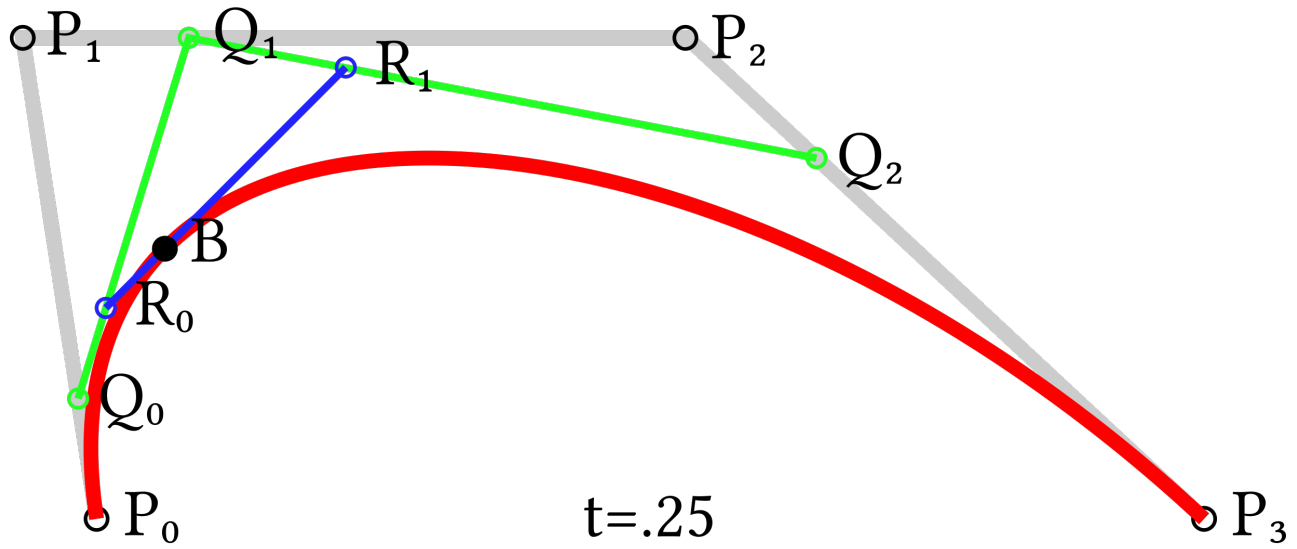
- Four points P_0, P_1, P_2 and P_3 in the plane or in higher-dimensional space define a cubic Bézier curve.
- The curve starts at P_0 going toward P_1 and arrives at P_3 coming from the direction of P_2. Usually, it will not pass through P_1 or P_2; these points are only there to provide directional information.
- The distance between P_1 and P_2 determines "how far" and "how fast" the curve moves towards P_1 before turning towards P_2.

Cubic Bézier Curve

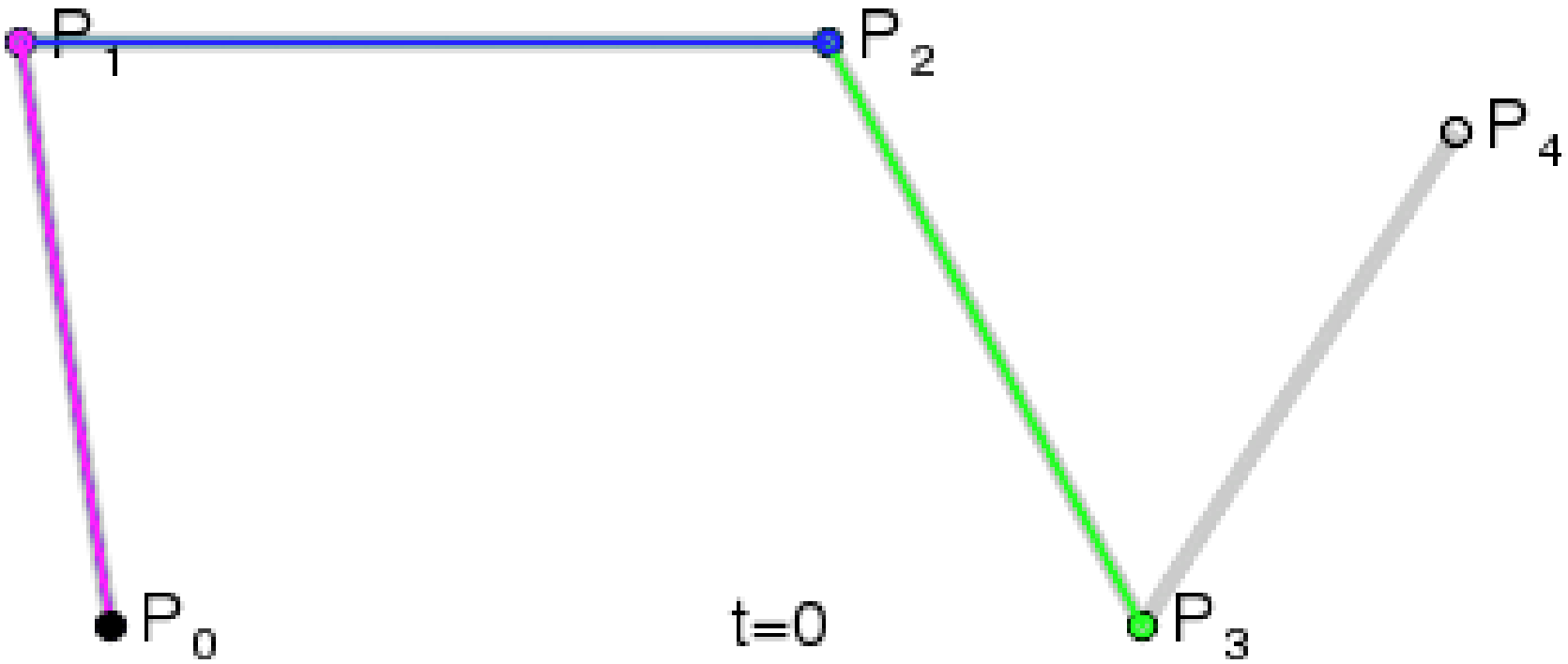


Cubic Bézier Curve

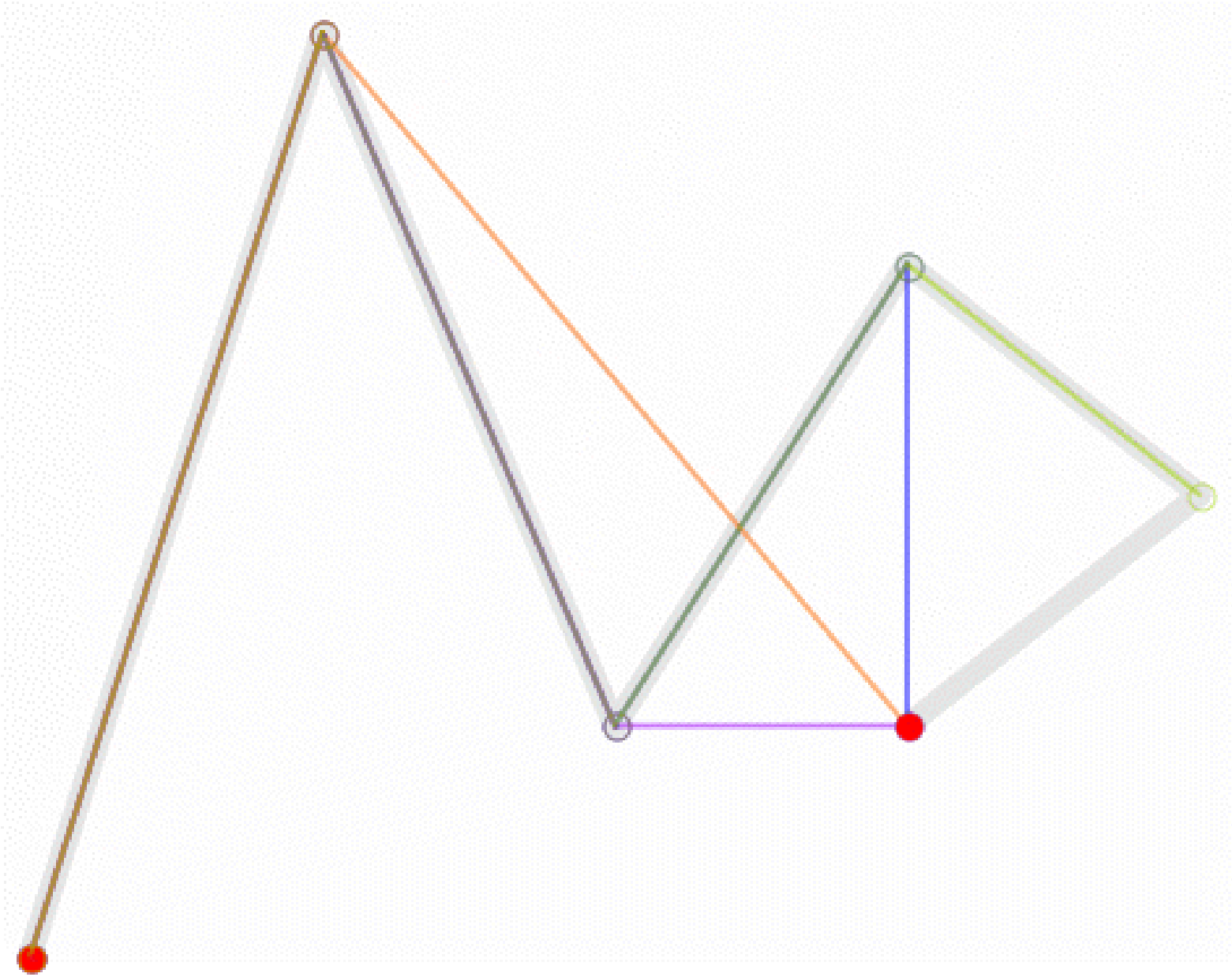
- For cubic curves one can construct intermediate points Q_0, Q_1, and Q_2 that describe linear Bézier curves, and points R_0 and R_1 that describe quadratic Bézier curves:



Fourth-order Bézier Curve

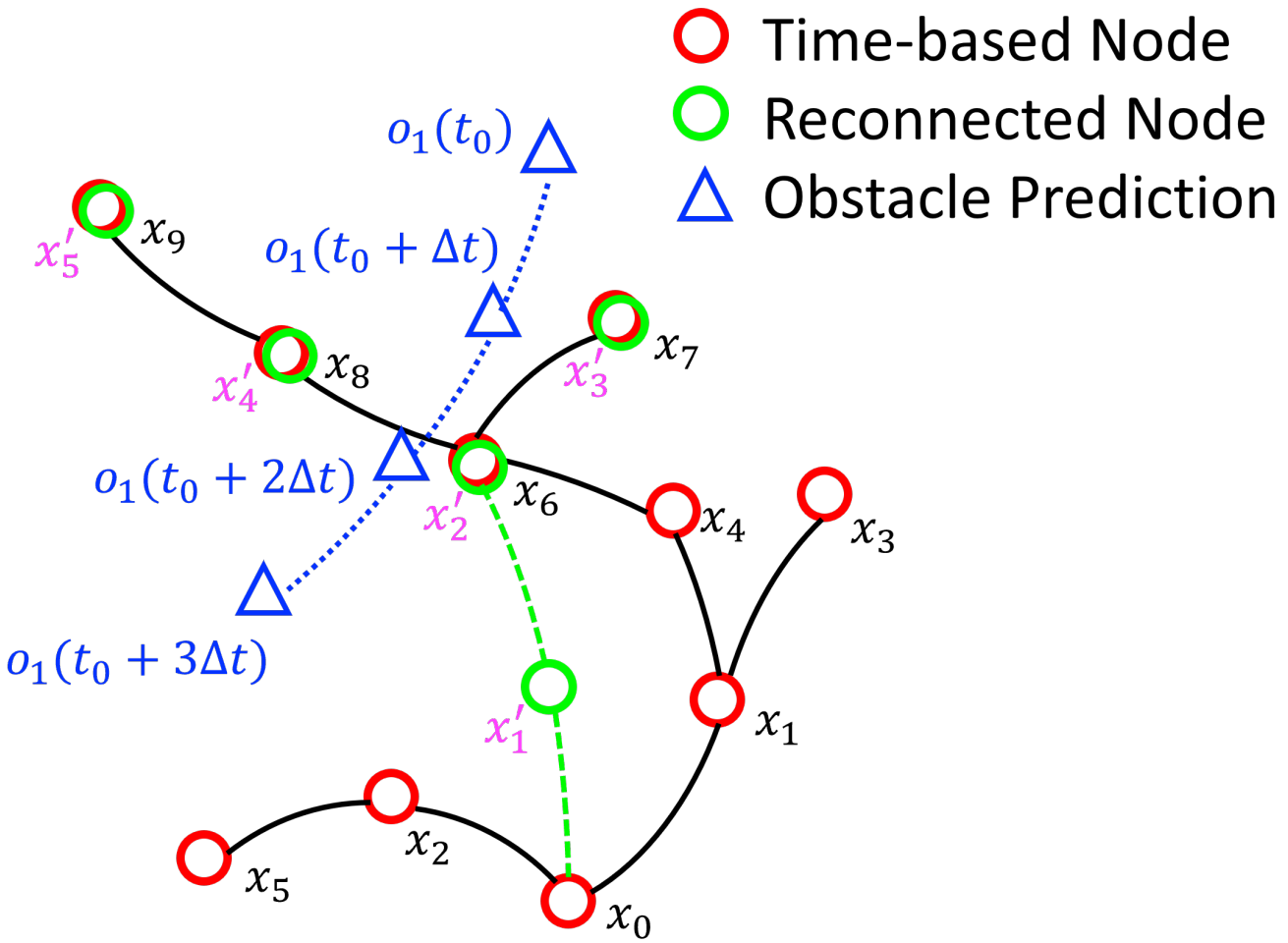


Fifth-order Bézier Curve



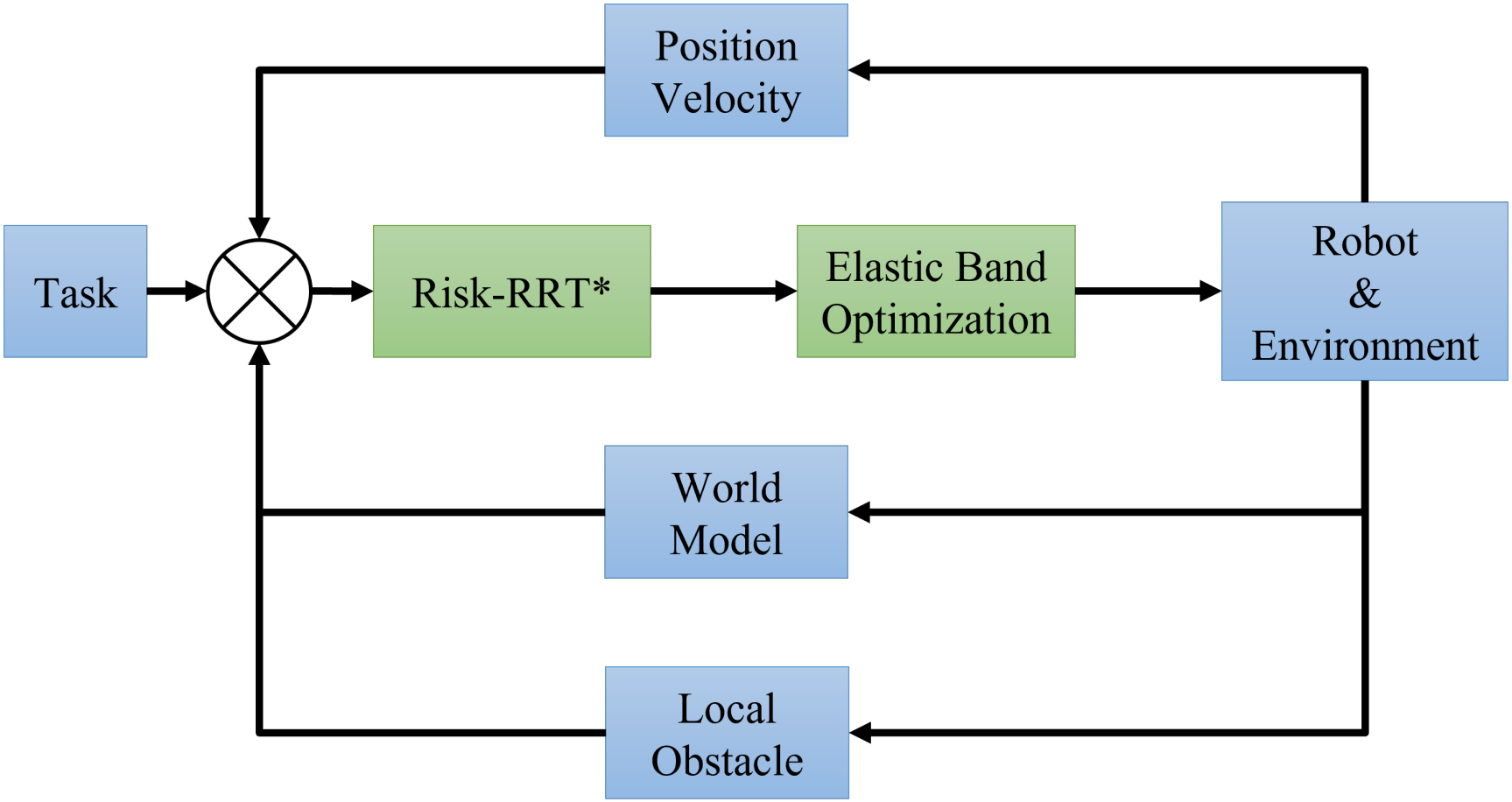
Outline

Elastic Band Based RRT

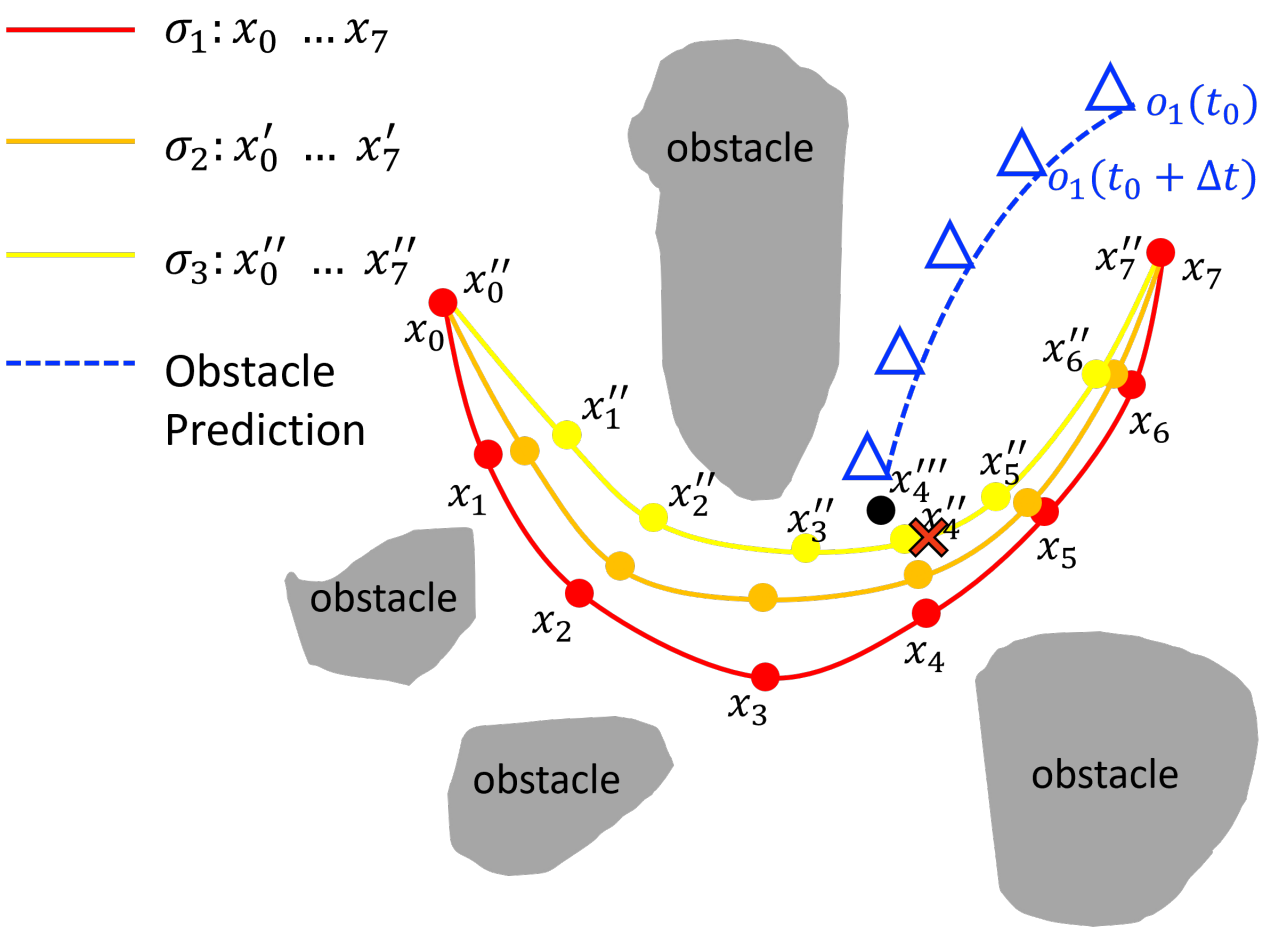


Wang, J., Meng, M. Q. H., & Khatib, O. (2020). EB-RRT: Optimal motion planning for mobile robots. IEEE Transactions on Automation Science and Engineering, 17(4), 2063-2073.

Elastic Band Based RRT

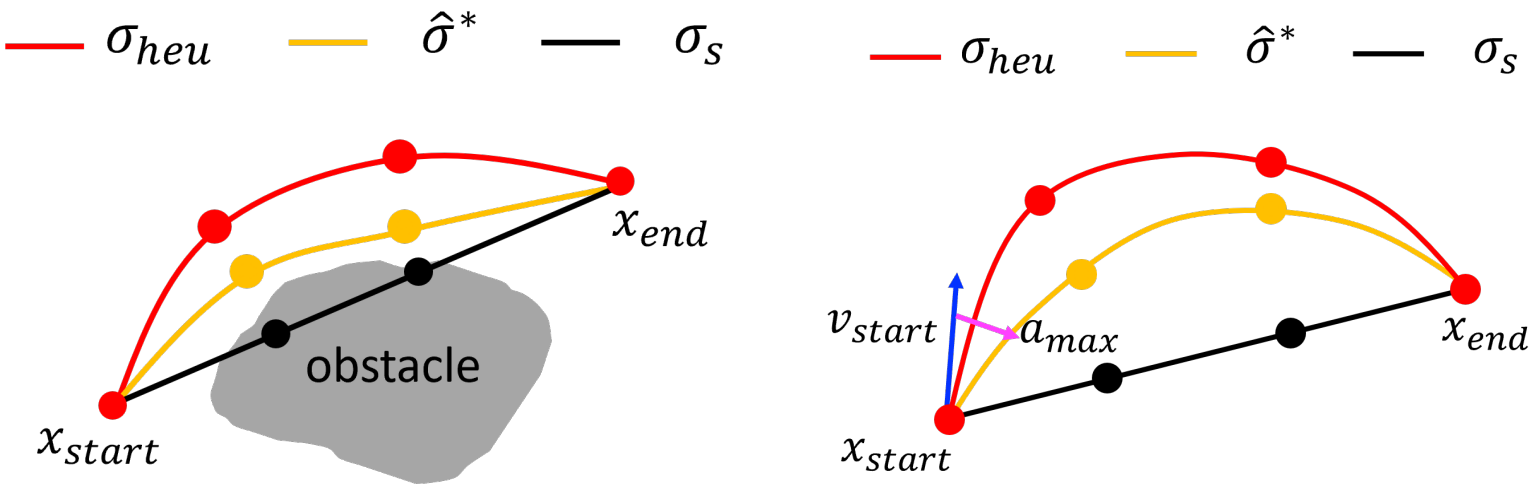


Elastic Band Based RRT



Elastic Band Based RRT

- Obstacle constraints and acceleration constraints serve as the external repulsive force.



Covariant Hamiltonian Optimization for Motion Planning

- Gradient optimization techniques for efficient motion planning.
- An obstacle term f_{obs}, which measures the cost of being near obstacles; and a prior term f_{prior}, which measures dynamical quantities of the robot such as smoothness and acceleration. The cost of a trajectory is

$$\mathcal{U}(\xi) = f_{prior}(\xi) + f_{obs}(\xi).$$

- f_{prior} is a simple quadratic form

$$f_{prior}(\xi) = \frac{1}{2}\xi^T A\xi + \xi^T b + c.$$

Ratliff, N., Zucker, M., Bagnell, J. A., & Srinivasa, S. (2009, May). CHOMP: Gradient optimization techniques for efficient motion planning. In 2009 IEEE international conference on robotics and automation (pp. 489-494). IEEE.

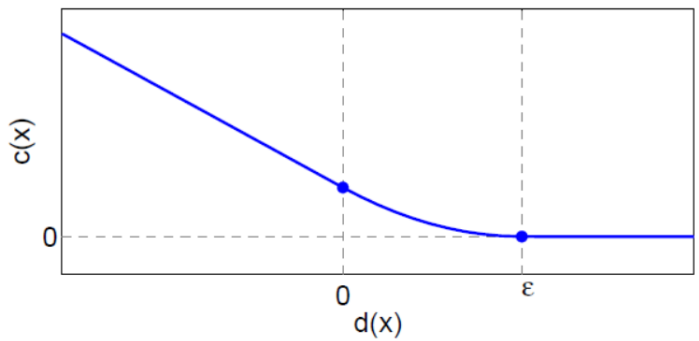
Covariant Hamiltonian Optimization for Motion Planning

- Assume the distance from robot to the nearest obstacle is greater than $\epsilon \geq 0$. The distance from a point $x \in \mathbb{R}^3$ to the boundary of the nearest obstacle is $d(x)$.
- $c(x)$ penalizes points of robots for being near obstacles

$$c(x) = \max(\epsilon - d(x), 0).$$

- A smoother version for the potential function

$$c(x) = \begin{cases} -d(x) + \frac{1}{2}\xi, & d(x) < 0 \\ \frac{1}{2\xi}(d(x) - \epsilon)^2, & 0 \leq d(x) \leq \epsilon \\ 0, & \text{otherwise} \end{cases}$$



Covariant Hamiltonian Optimization for Motion Planning

- The obstacle objective is defined as

$$f_{obs}[q] = \int_0^1 \int_{\mathcal{B}} c\Big(x(q(t), u)\Big) \left\| \frac{d}{dt} x(q(t), u) \right\| du dt.$$

- Recall the cost of a trajectory $\mathcal{U}$, we can approximate it using a first-order Taylor expansion:

$$\mathcal{U}(\xi) \approx \mathcal{U}(\xi_k) + g_k^T(\xi - \xi_k), \; g_k = \nabla \mathcal{U}(\xi_k).$$

- The update rule is

$$\xi_{k+1} = \arg\min_{\xi} \left\{ \mathcal{U}(\xi_k) + g_k^T(\xi - \xi_k) + \frac{\lambda}{2} ||\xi - \xi_k||_M^2 \right\}.$$

Exercise

- Consider a Quadratic Bézier Curve

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, \quad 0 \leq t \leq 1.$$

Given control points $P_0 = (1, 2), P_1 = (2, 4), P_2 = (4, 1)$, write out the explicit parametric equations for the Bézier Curve in the $xy-$plane.

Solution

- $x = t^2 + 2t + 1$
- $y = -5t^2 + 4t + 2$

Q & A

1. Further Discussion of RRT
2. Smoothing
3. Case Studies