Autonomous Vehicle Planning and Control

Wu Ning

Session 6

Vehicle Motion Planning/Local Planner

Outline

Motion planner

- Motion planner basic concept
- Functionality and Common method
- Fundamental Concepts and Key terminology

Stochastic Sampling Methods

- RRT
- RRT*
- Other improve methods

Lattice planner

- Frenet Coordinate
- Speed planning
- Trajectory planning

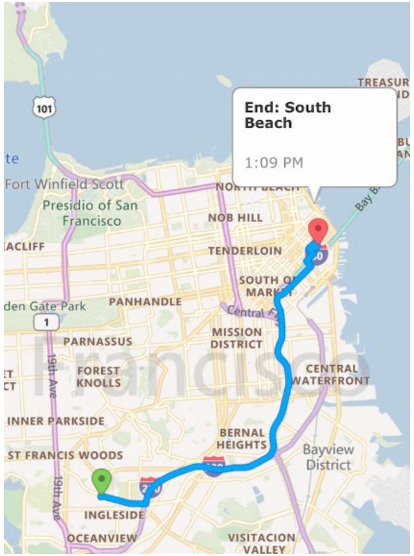
Motion Planner

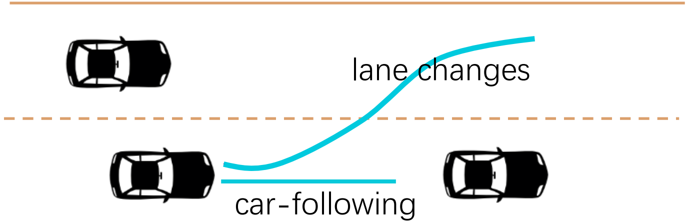
Part I: Main concept and key terminology

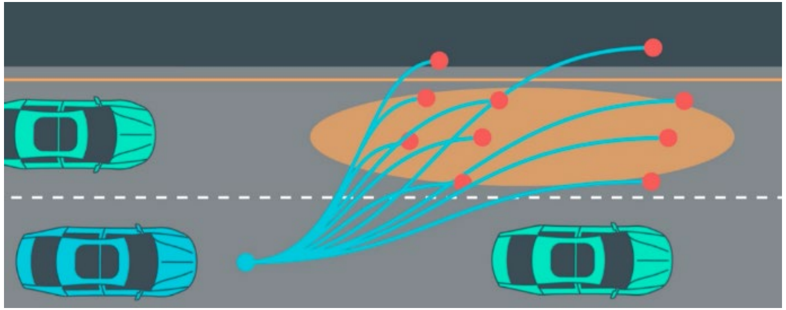
Motion Planning Problem

Determine a sequence of actions to reach a specified goal

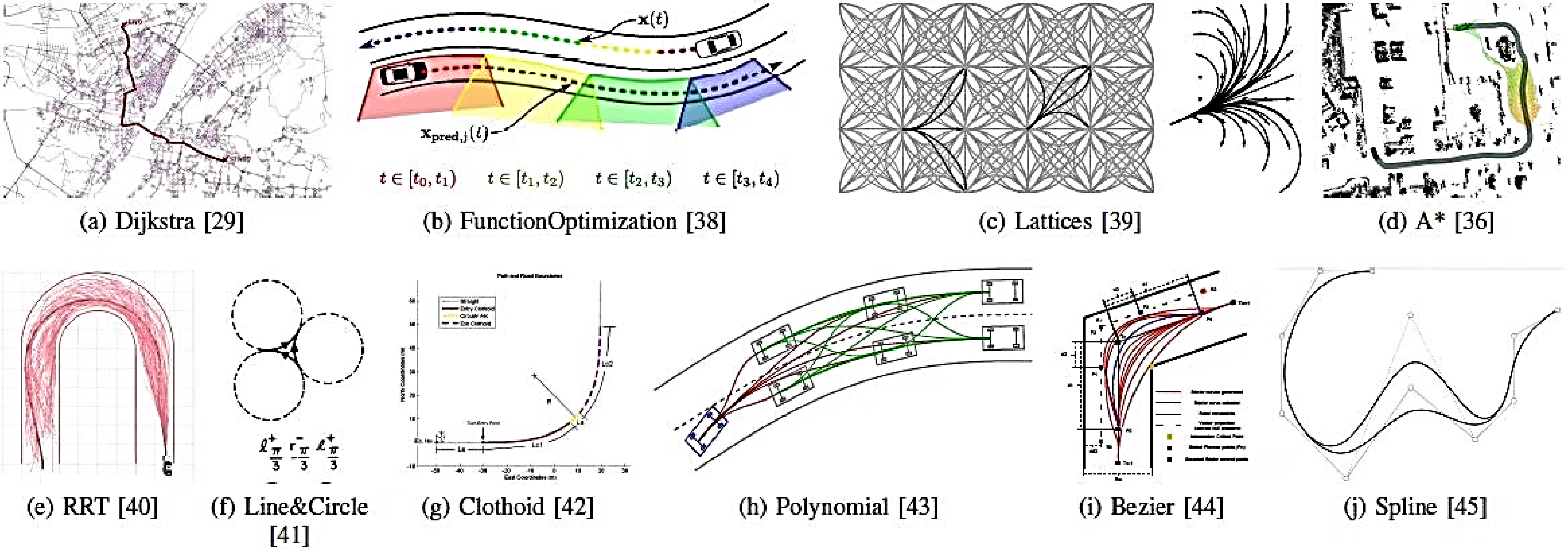
Mission/ Route Planning → Behavior Planner/ Decision making → Motion planning/ Local planner → Motion Control → Vehicle







The Common Motion Planning Methods

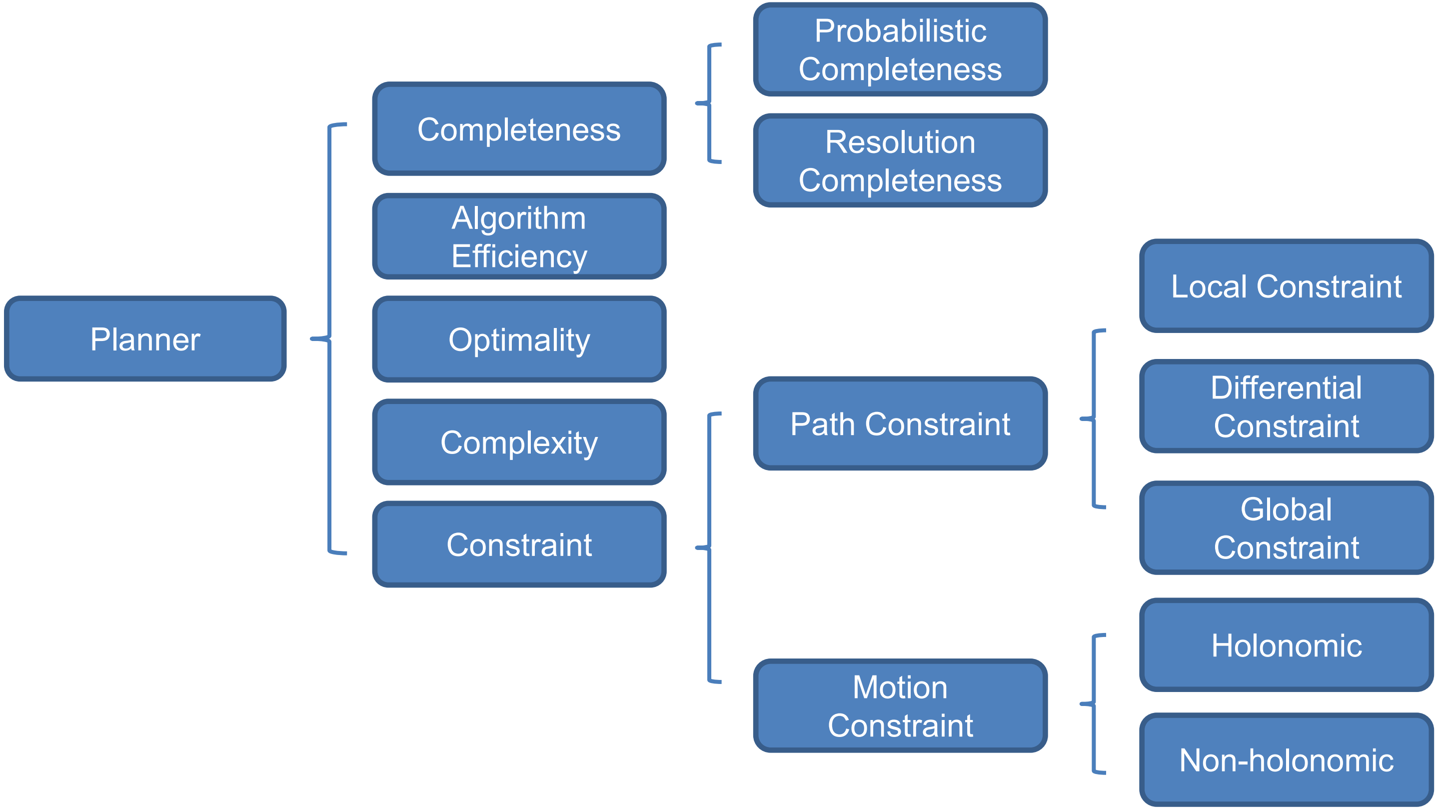


(a) Dijkstra [29] (b) FunctionOptimization [38] (c) Lattices [39] (d) A* [36]

(e) RRT [40] (f) Line&Circle [41] (g) Clothoid [42] (h) Polynomial [43] (i) Bezier [44] (j) Spline [45]

(a)Global path by Dijkastra. (b) Trajectory optimization considering a vehicle in the other lane. (c) Lattices and motion primitives. (d) Hybrid A* in DRAPA Junior. (e) RRT. (f) Optimal path to turn the vehicle around. (g) Planning a turn from Stanford. (h) Different motion states, planned with polynomial curves. (i) Evaluation of several Bezie curves. (j) Spline behaviour when a knot changes places

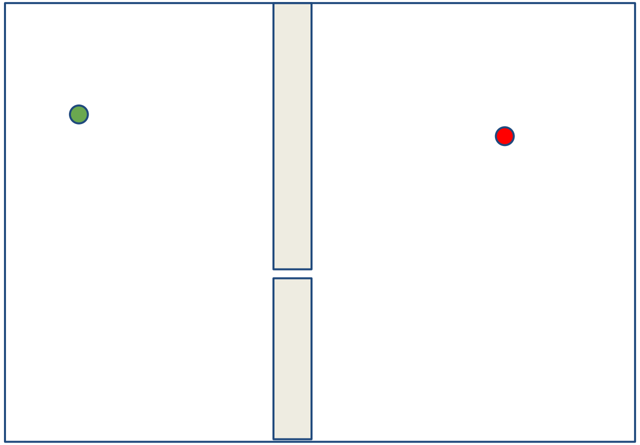
Planner Properties



Fundamental Concepts

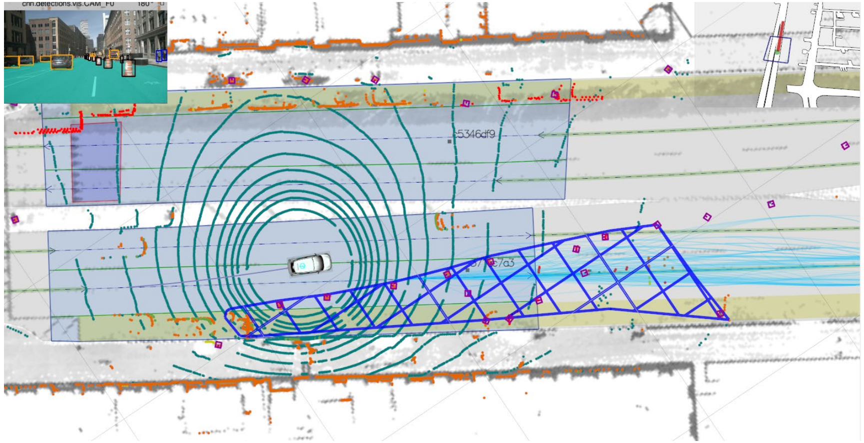
Completeness:

- Ability to find a solution if one exists,
- Narrow Gap Problem Examples.



2D space ($\mathbb{R}^2$), point robot

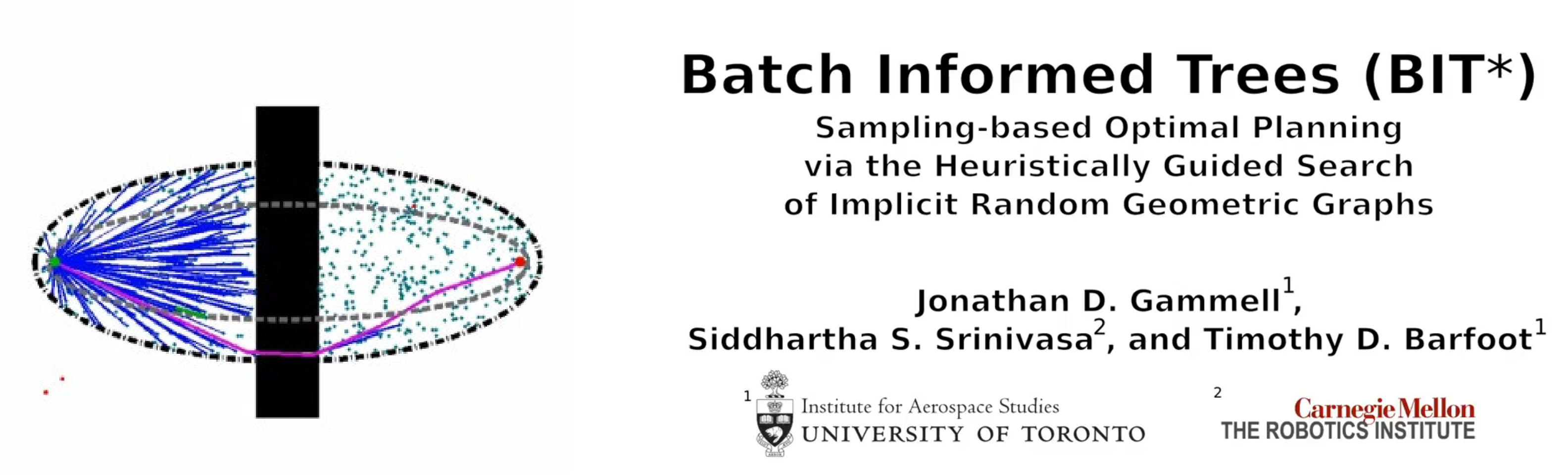
Self-driving car navigating around construction



Fundamental Concepts

Probabilistic Completeness:

- Probability of finding a solution (when one exists) increases as computational time spent on the problem increases



[Batch Informed Trees (BIT*) @ UTIAS / CMU (ICRA 2015)]

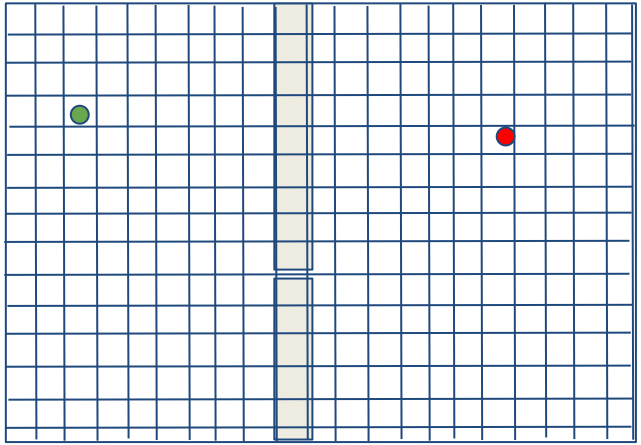
Fundamental Concepts

Resolution Completeness:

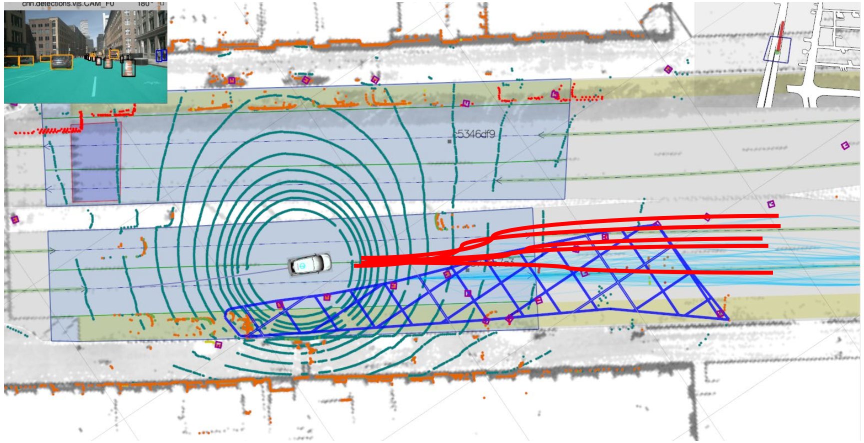
- Ability to find a solution if one exists AND using fine enough resolution in discretization of the state and/or control space

Narrow Gap Problem Examples:

2D space ($\mathbb{R}^2$), point robot



Self-driving car navigating around construction



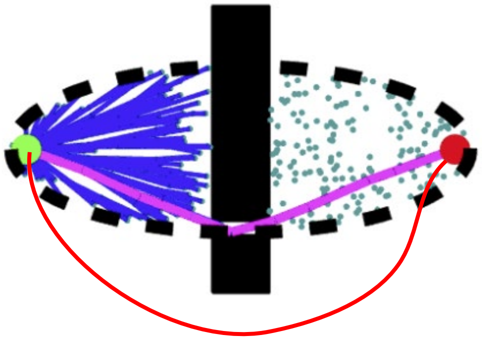
Fundamental Concepts

Algorithmic efficiency:

- How an algorithm solve time scales proportionally with respect to size of the input data,
- Big O Notation, $O(n)$ takes time proportional to number of elements.

Optimality:

- Optimal is able to find the lowest cost solution of all possible options,
- Suboptimal of a lower cost solution exists,
- Asymptotically optimal if guaranteed to converge to the optimal solution given increasing, computational time spent on the problem.



Fundamental Concepts

Complexity

- Space dimensionality
  - configuration space is a $\mathbb{R}^3$ for rigid body. But for the multi-bodies track, bicycle mode is not accurate enough to capture all the constraints.
- Geometric complexity
  - How bounding box and bounding box interact;
  - How to detect a path between polygons and interact with another obstacles;
  - ...

Fundamental Concepts

There are three types of **Path Constraints**:

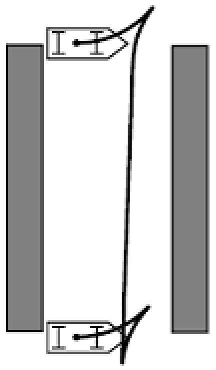
- **Local Constraints**
  - Avoid collision with static obstacles
- **Differential constraints**
  - Bounded curvature, limited steering angle for vehicle
- **Global constraints**
  - Find the shortest path by A*

Fundamental Concepts

Holonomic vs non-holonomic motion constraints

- Holonomic if # of controllable DOF = # of total DOF

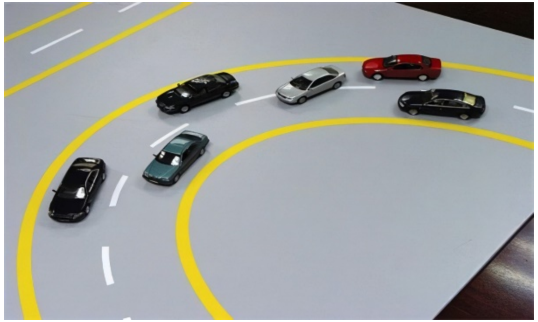
Cars are non-holonomic since control throttle and steering (2 DOF), but move in $SE(2)$ (x, y, θ)



Local Planning

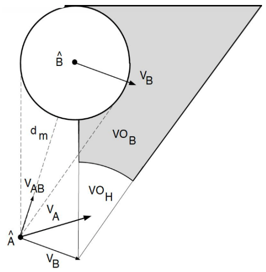
Find higher precision "good" (near optimal w.r.t. cost function) path to execute.

Where to search?



Configuration Space Parameterization Options

- Workspace (direct physical environment, traditional)
- Control Space (e.g. velocity space, see [link])
  - Only saves effort in simple problems
- Belief Space (POMDP, more later in Session 7)

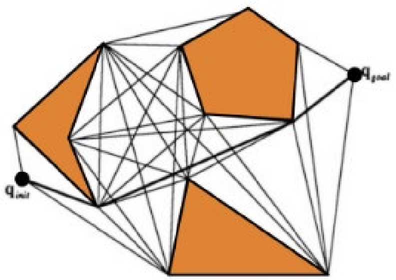


Velocity Obstacle (control space)

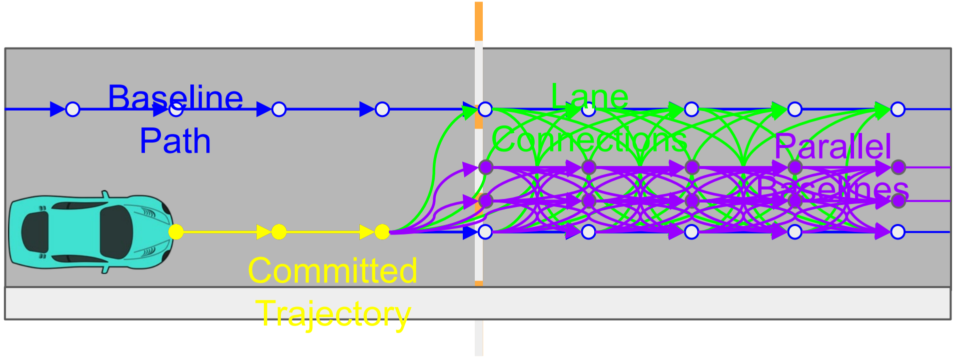
Local Planning

How to search?

- **Combinatorial methods (exact complete solution, e.g. visibility graph - link)**
  - Rarely exists to find optimal solution to complex problems
- **Sampling-based methods**
  - Deterministic (resolution complete), e.g. uniform grid or road structure graph, repeatable
  - Stochastic (probabilistic completeness), e.g. random sampling
- May need some **smoothing/post-process** to improve quality of solution



Visibility Graph (combinatorial)



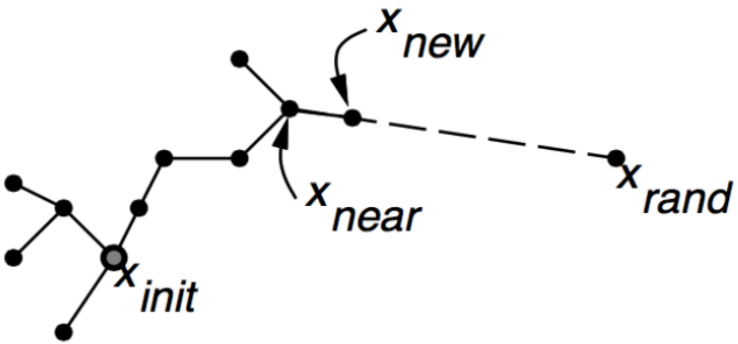
road “structural graph” (deterministic sampling)

Motion Planner

Part II Stochastic Sampling Methods

Stochastic Sampling Methods: Rapidly-exploring Random Trees

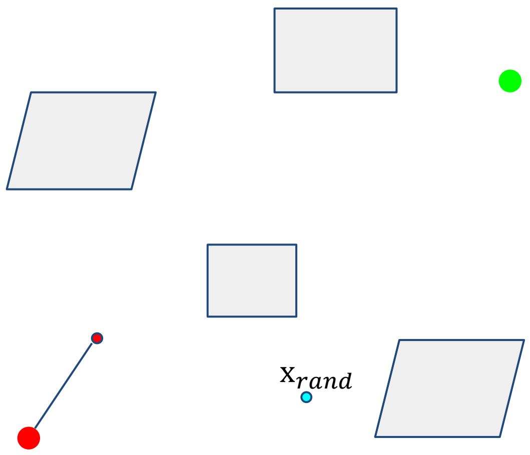
Build up a tree from start to goal through generating “next states” in the tree by executing random controls.



Stochastic Sampling Methods: RRT

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  $E_i \leftarrow Edge(x_{new}, x_{near})$;
  if *CollisionFree*($\mathcal{M}$, E_i) **then**
    $\mathcal{T}.addNode(x_{new})$;
    $\mathcal{T}.addEdge(E_i)$;
  if $x_{new} = x_{goal}$ **then**
    Success();

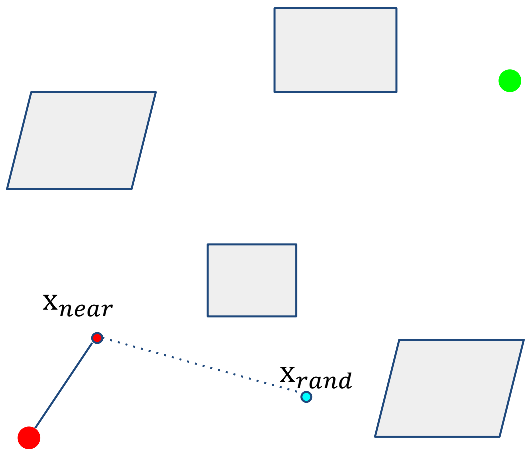


Sample a node X_{rand} in the free space

Stochastic Sampling Methods: RRT

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  $E_i \leftarrow Edge(x_{new}, x_{near})$;
  if *CollisionFree*($\mathcal{M}$, E_i) **then**
    $\mathcal{T}.addNode(x_{new})$;
    $\mathcal{T}.addEdge(E_i)$;
  if $x_{new} = x_{goal}$ **then**
    Success();

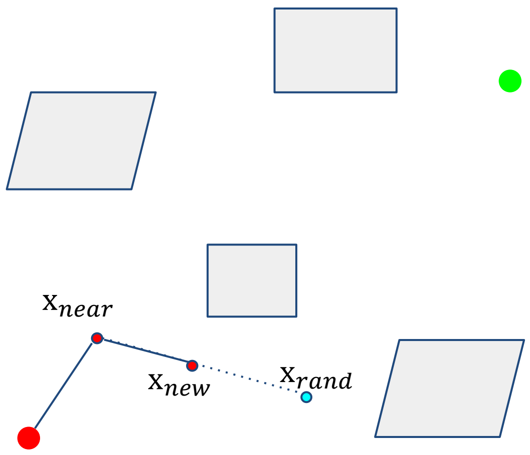


Find the nearest node X_{near} in current tree

Stochastic Sampling Methods: RRT

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  $E_i \leftarrow Edge(x_{new}, x_{near})$;
  if *CollisionFree*($\mathcal{M}$, E_i) **then**
    $\mathcal{T}.addNode(x_{new})$;
    $\mathcal{T}.addEdge(E_i)$;
  if $x_{new} = x_{goal}$ **then**
    Success();



Grow a new node X_{new} and path E_i from X_{near}

Stochastic Sampling Methods: RRT

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

  $x_{rand} \leftarrow Sample(\mathcal{M})$;

  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;

  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;

  $E_i \leftarrow Edge(x_{new}, x_{near})$;

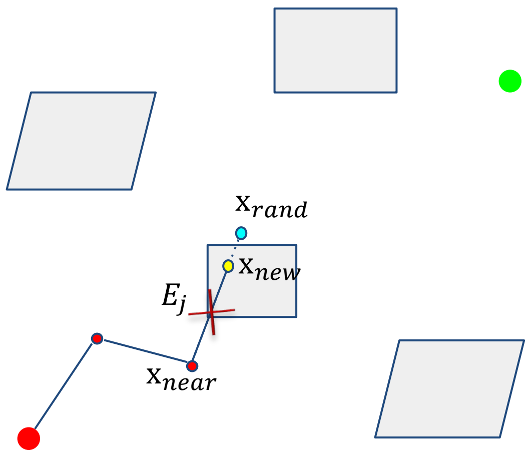
  if $CollisionFree(\mathcal{M}, E_i)$ **then**

    $\mathcal{T}.addNode(x_{new})$;

    $\mathcal{T}.addEdge(E_i)$;

  if $x_{new} = x_{goal}$ **then**

    Success();



Do not grow if collision

Stochastic Sampling Methods: RRT

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$, x_{init}, x_{goal}

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

  $x_{rand} \leftarrow Sample(\mathcal{M})$;

  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;

  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;

  $E_i \leftarrow Edge(x_{new}, x_{near})$;

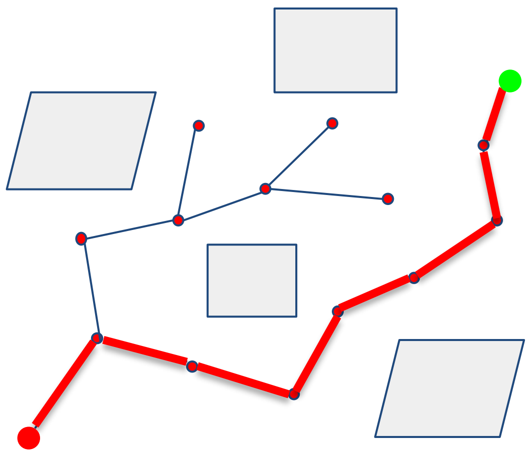
  if *CollisionFree*($\mathcal{M}$, E_i) **then**

    $\mathcal{T}.addNode(x_{new})$;

    $\mathcal{T}.addEdge(E_i)$;

  if $x_{new} = x_{goal}$ **then**

    Success();



Repeat sampling for n times until the tree reaches the goal or goal region

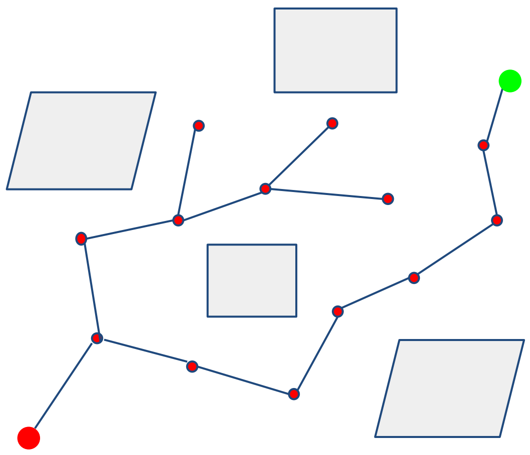
Stochastic Sampling Methods: RRT

Pros

- Easy to implement
- Aims to find a path from the start to the goal
- More target-oriented than PRM

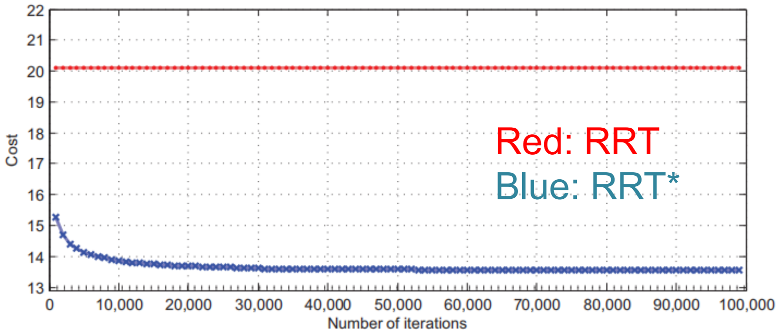
Cons

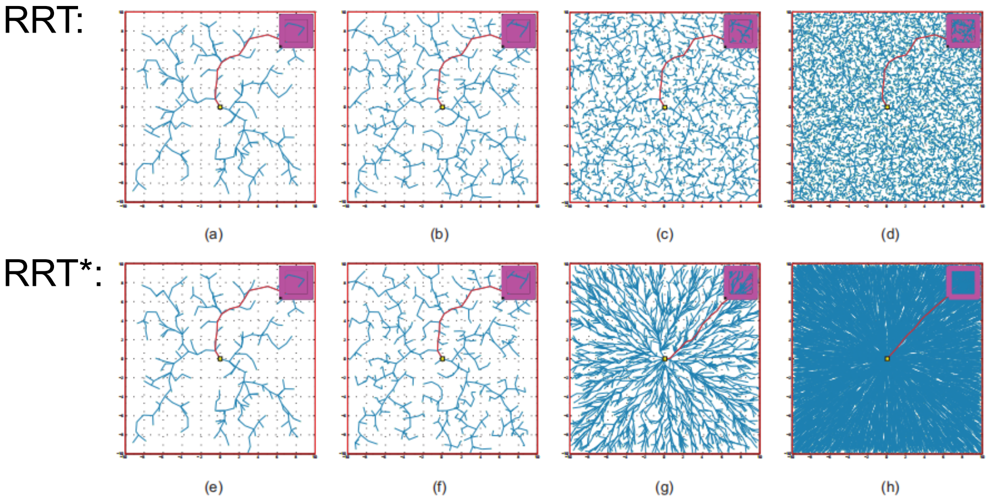
- Not optimal solution
- Not efficient (leave room for improvement)
- Sample in the whole space



Stochastic Sampling Methods: RRT*

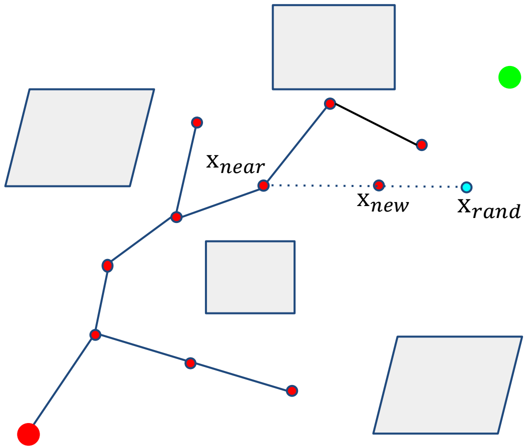
- An improvement of RRT
- Asymptotically optimal (under conditions)





Stochastic Sampling Methods: RRT*

Consider N nearing nodes

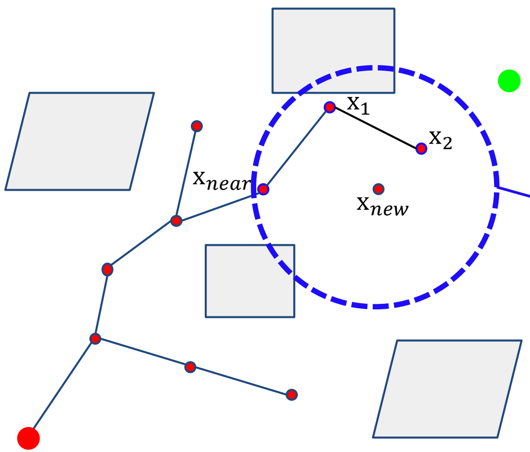


Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Stochastic Sampling Methods: RRT*

Consider N nearing nodes

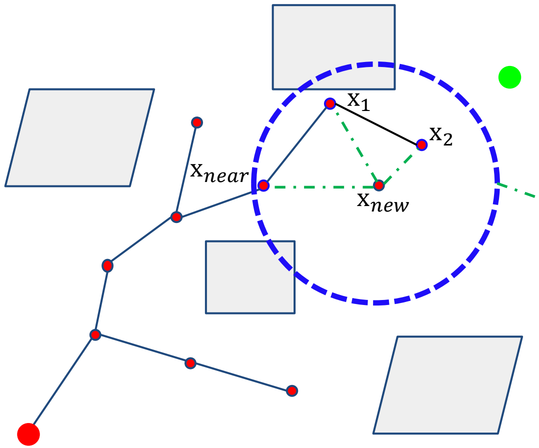


Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Stochastic Sampling Methods: RRT*

Consider history cost instead of only local information

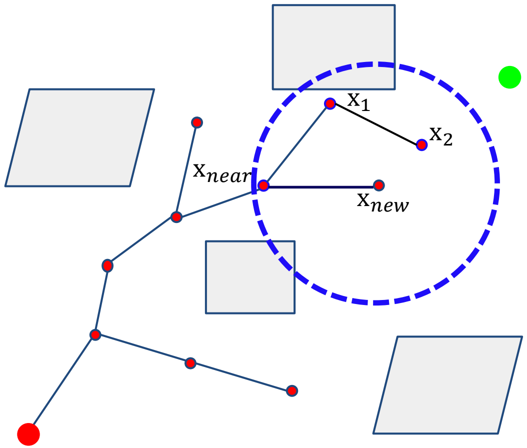


Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
    $x_{rand} \leftarrow Sample(\mathcal{M})$;
    $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
    $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
    if *CollisionFree*(x_{new}) **then**
        $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
        $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
        $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
        $\mathcal{T}.rewire()$;

Stochastic Sampling Methods: RRT*

Consider history cost instead of only local information

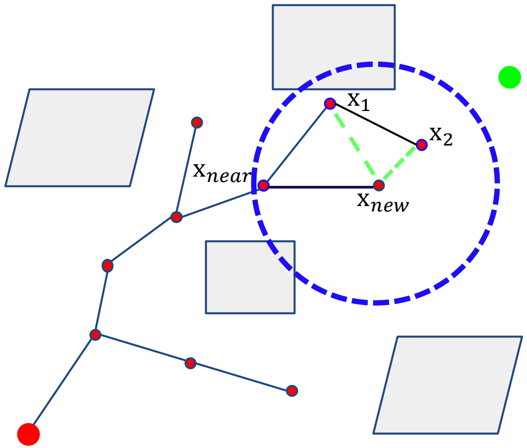


Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Stochastic Sampling Methods: RRT*

Rewire to improve local optimality

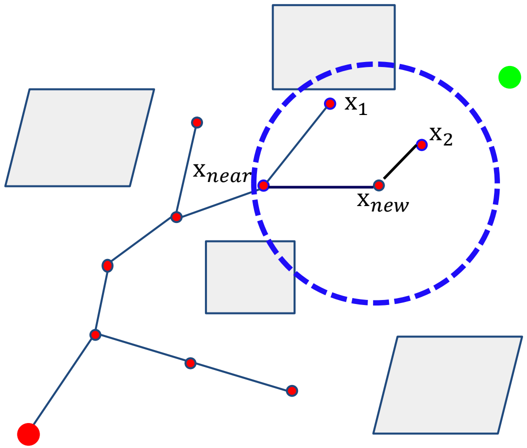


Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Stochastic Sampling Methods: RRT*

Rewire to improve local optimality

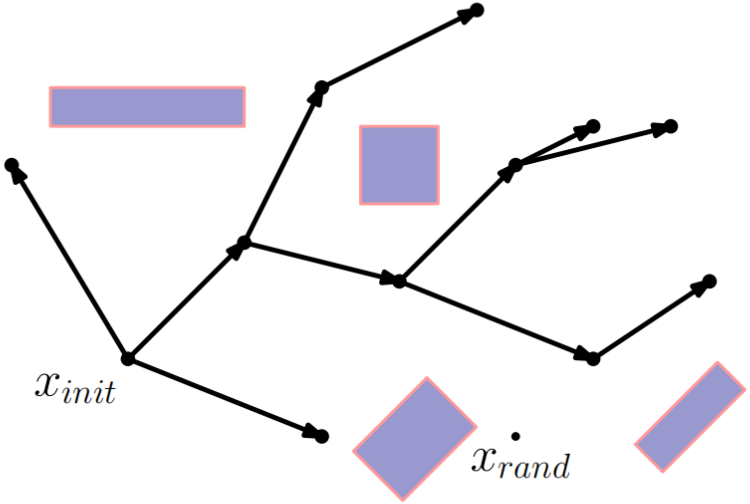


Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Stochastic Sampling Methods: RRT*

Generate a random point x_{rand}



Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

  $x_{rand} \leftarrow Sample(\mathcal{M})$;

  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;

  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;

  if *CollisionFree*(x_{new}) **then**

    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;

    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;

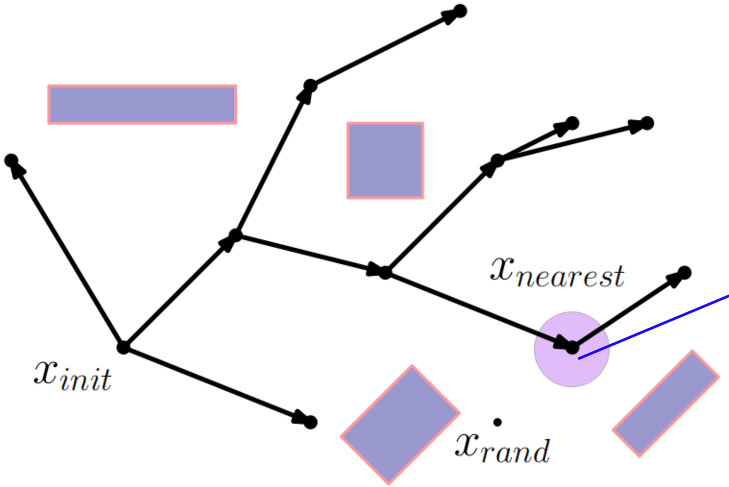
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;

    $\mathcal{T}.rewire()$;

https://blog.csdn.net/weixin_

Stochastic Sampling Methods: RRT*

Find the nearest nodes $x_{nearest}$



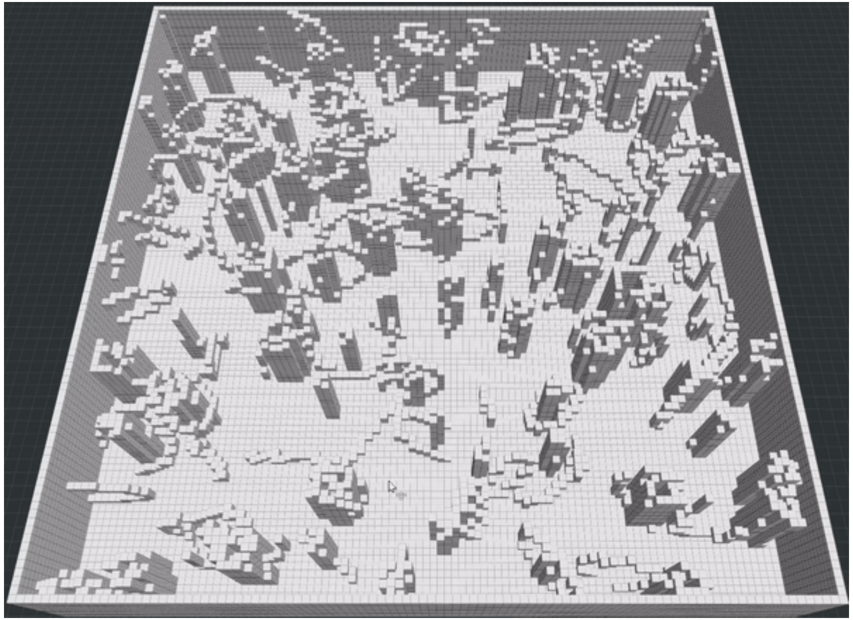
Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$, x_{init}, x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

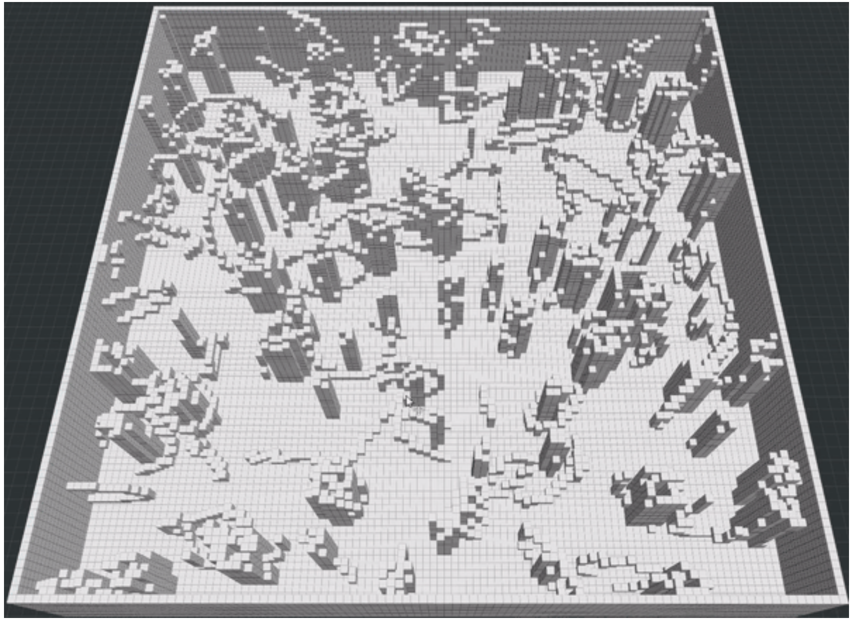
https://blog.csdn.net/weixin_43

Stochastic Sampling Methods: RRT*

RRT



RRT*



https://github.com/ZJU-FAST-Lab/sampling-based-path-finding

Stochastic Sampling Methods: Obstacle avoidance

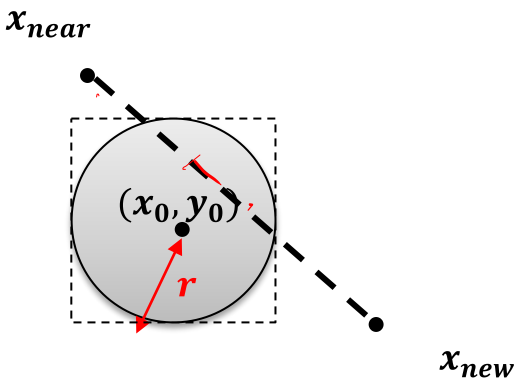
Obstacle avoidance check:

To simplify the check, we used circle or polygon (rectangle) to represent the objects

- For the circle object, we can check the obstacle easily by

$$x_0 - r - \varepsilon < x_{new,x} < x_0 + r + \varepsilon$$

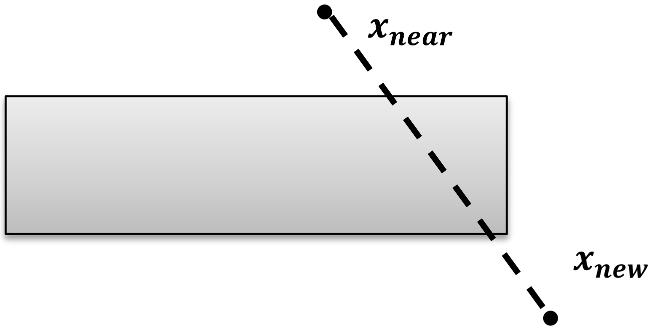
$$y_0 - r - \varepsilon < x_{new,y} < y_0 + r + \varepsilon$$



Stochastic Sampling Methods: Obstacle avoidance

Obstacle avoidance check: (polygon: rectangle)

- There are two steps for collision checking for polygon
- If x_{near} and x_{new} are in the same side of obstacle;
  - If they are at the same side, there will be no interaction with the obstacle;
- If x_{near} and x_{new} are not at the same side, there will be two situations:
  - x_{new} is inside the polygon, and there must be an intersection between the connection of x_{near} and x_{new}
  - If both x_{near} and x_{new} are at the outside of the polygon, we will need to use the connect line to check the collision.



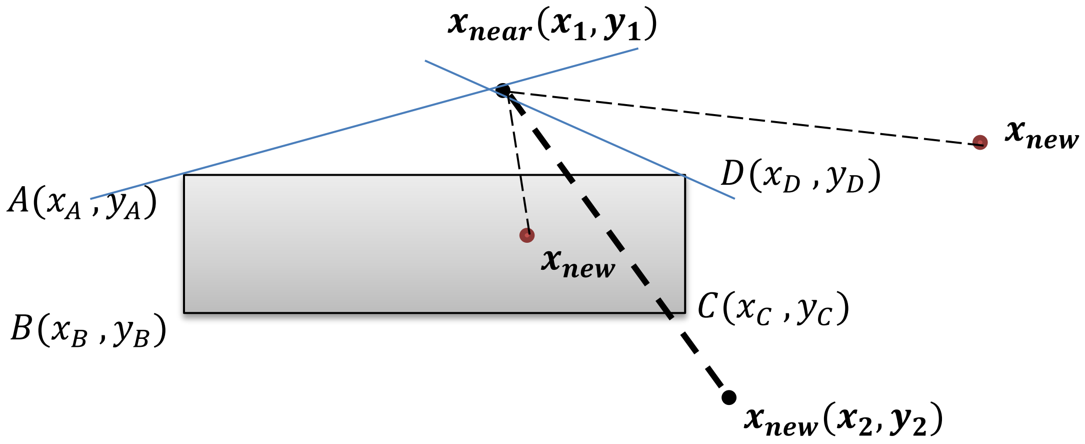
Stochastic Sampling Methods: Obstacle avoidance

Obstacle avoidance check:

• If both x_{near} and x_{new} are at the outside of the polygon, we will need to use the connect line to check the collision.

$$k_{x_{near}x_{new}} < k_{Dx_{near}} \,\&\&\, k_{x_{near}x_{new}} > k_{Ax_{near}}$$

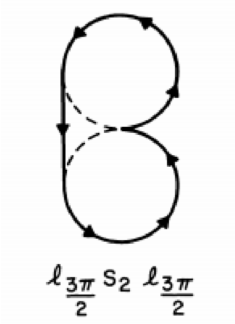
where k is the slope of line

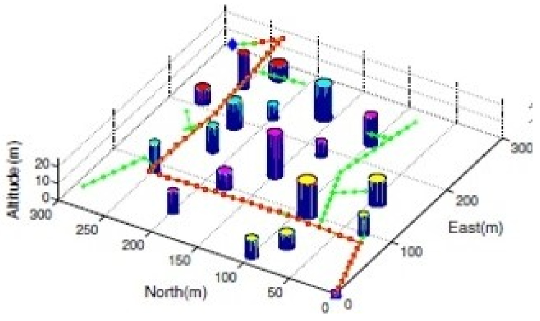


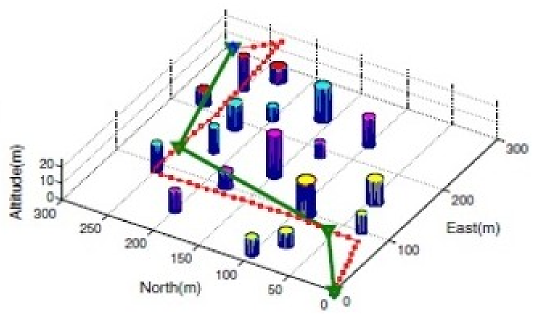
Stochastic Sampling Methods: Other improvement

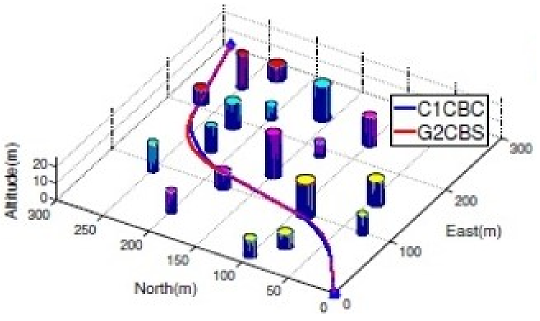
Improvements may come through focus on any of those steps individually, e.g.:

- Connect with edges of desirable properties
  - ✓ Dubins for shortest length w/fixed turn radius
  - ✓ Reeds-Shepp for forward/backward w/fixed turn radius
  - ✓ Spline, clothoid, Bezier for continuous curvature









Stochastic Sampling Methods: RRT^* Improvement

Bias Sampling

- Sample biasing toward the goal

Sample Rejection

- Reject samples that don't meet some threshold until you reach the number of samples you need

Tree Pruning

- Prune the non-promising sub trees to reduce neighbor query cost.

Graph Sparsify

- Reject samples by resolution. Introduce near optimality.

Delay Collision Check

- Sort the neighbours by potential cost-to-come values. Check collisions in order and stop once a collision-free edge is found.

Anytime RRT

- Store the collision-checking results for ChooseParent and Rewire.

Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic

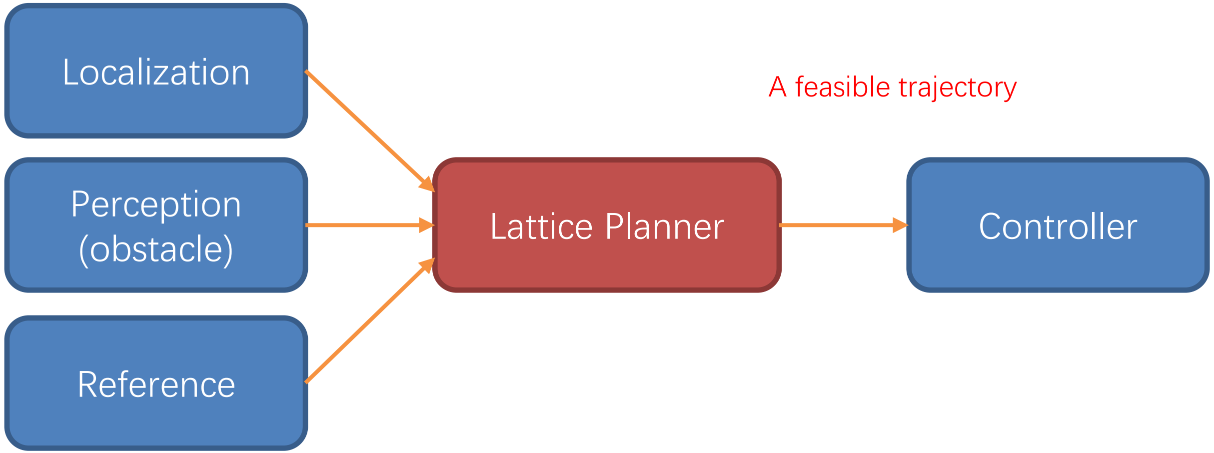
Motion Planner

Part III Lattice Planner

Lattice Planner

The Lattice Planner algorithm is a local motion planner,

- Sample based motion planner;
- Plan in Frenet coordinate;
- The output is a smooth, safe and collision-free local trajectory that satisfies the vehicle's kinematics and speed constraints which is directly feed into controller.



Lattice Planner

The basic process of Lattice planner:

Transform to Frenet coordinate, and calculate the look ahead

↓

Sample the path
(based on time, target speed and lateral displacement of the reference path)

↓

Construct lateral and longitudinal displacement planning function $s(t)$ and $d(s)$

↓

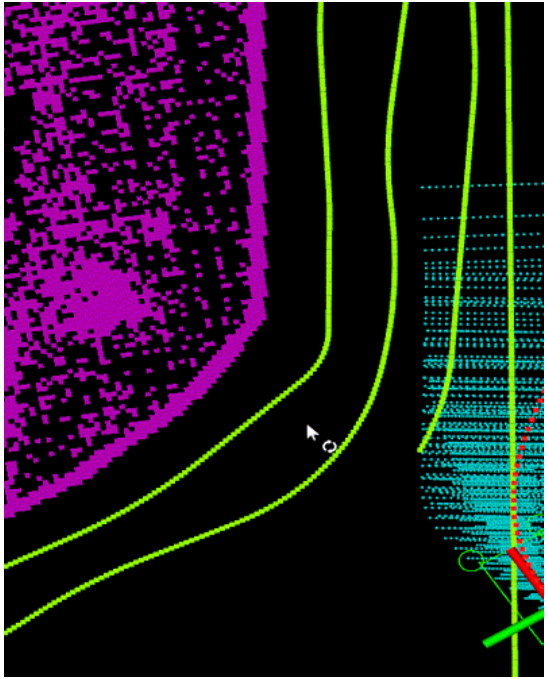
Calculate the reference paths in Frenet coordinate by sampling time t

↓

Transform the paths back to global Cartesian coordinate

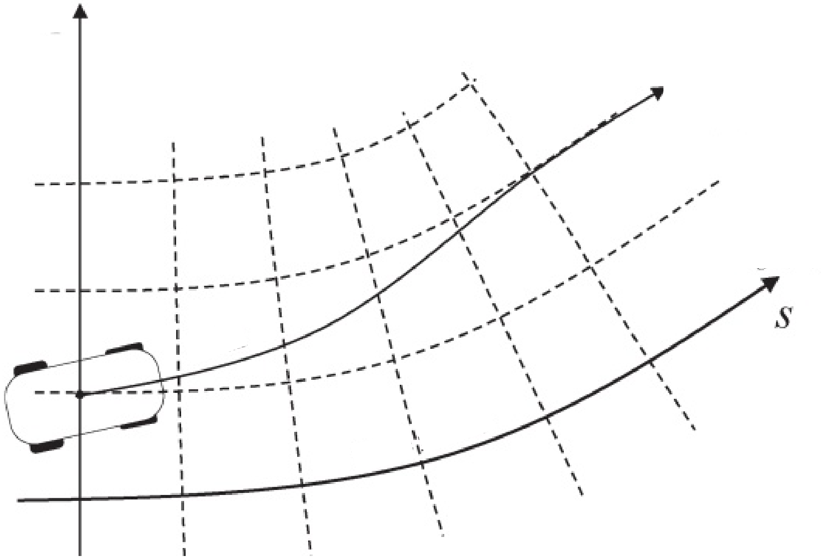
↓

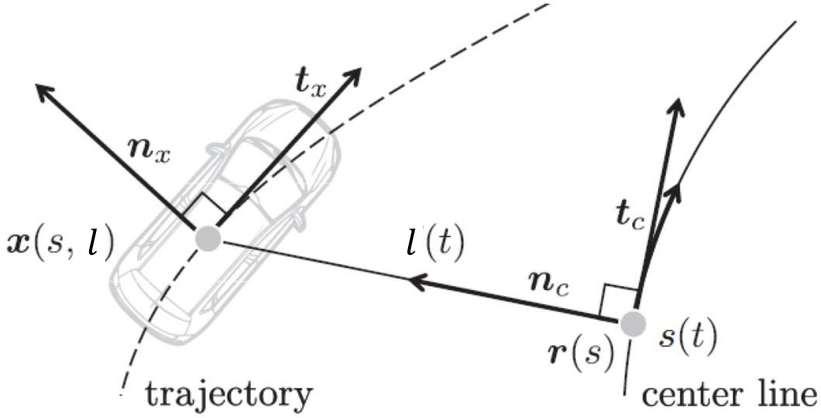
Trajectory scoring
(based on the cost and constraints, such as collision cost and vehicle dynamics constraints)



Frenet coordinate and Cartesian coordinate transformation

Frenet coordinate is a frame on the reference line, is a moving frame. Its origin *r* is the nearest point from the vehicle to the path. The *t*-axis is along the tangiential direction at *r* while *n*-axis is pependicular to *t*-axis.





Frenet coordinate and Cartesian coordinate transformation

Frenet coordinate: $[\mathbf{s}, \dot{\mathbf{s}}, \ddot{\mathbf{s}}, \boldsymbol{l}, \dot{\boldsymbol{l}}, \ddot{\boldsymbol{l}}, \boldsymbol{l}', \boldsymbol{l}'']$

s: longitudinal axis (T- axis) of Frenet coordinate

$\dot{s} = \frac{ds}{dt}$: differentiation of longitudinal axis w.r.t. to time, i.e. speed

$\dot{s} = \frac{d\dot{s}}{dt}$: longitudinal acceleration

l: lateral axis of Frenet coordinate

$\dot{l} = \frac{dl}{dt}$: lateral speed

$\dot{l} = \frac{d\dot{l}}{dt}$: lateral acceleration

l': differentiation of lateral axis w.r.t. to longitudinal axis

l'': 2nd derivative of lateral axis w.r.t. to longitudinal axis

Cartesian coordinate: $[\vec{\boldsymbol{x}}, \boldsymbol{v_x}, \boldsymbol{a_x}, \boldsymbol{\theta_x}, \boldsymbol{\kappa_x}]$

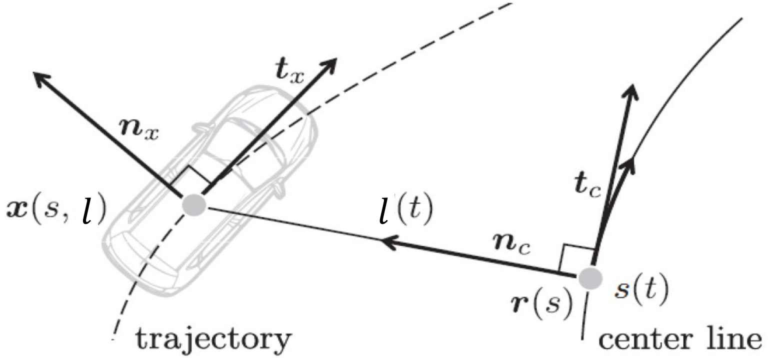
$\vec{x}$: a position vector in Cartesian coordinate

$v_x = \left\|\dot{x}\right\|_2$: speed in Cartesian coordinate

$a_x = \frac{v_x}{dt}$: acceleration in Cartesian coordinate

θ_x: heading in Cartesian coordinate

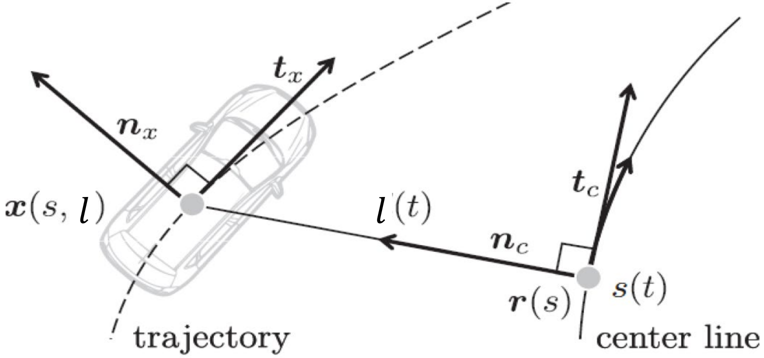
$\kappa_x = \frac{d\theta_x}{ds}$: curvature



Frenet coordinate and Cartesian coordinate transformation

Cartesian to Frenet

$$
\begin{cases}
s = s_r \\
\dot{s} = \dfrac{v_x cos(\theta_x - \theta_r)}{1 - k_r l} \\
\ddot{s} = \dfrac{a_x cos(\theta_x - \theta_r) - \dot{s}^2 \left[l' \left(k_x \dfrac{1 - k_r l}{cos(\theta_x - \theta_r)} - k_r \right) - (k_r' l + k_r l') \right]}{1 - k_r l} \\
l = sign\big((x_x - x_r)cos(\theta_r) - (y_x - y_r)sin(\theta_r)\big)\sqrt{(x_x - x_r)^2 + (y_x - y_r)^2} \\
l' = (1 - k_r l)tan(\theta_x - \theta_r) \\
l'' = -(k_r' l + k_r l')tan(\theta_x - \theta_r) + \dfrac{1 - k_r l}{cos^2(\theta_x - \theta_r)} \left(\dfrac{1 - k_r l}{cos(\theta_x - \theta_r)} k_x - k_r \right)
\end{cases}
$$



Frenet coordinate and Cartesian coordinate transformation

Frenet to Cartesian

$$
\begin{cases}
x_x = x_r - l\sin(\theta_r) \\
y_x = y_r + l\cos(\theta_r) \\
\theta_x = \arctan\left(\dfrac{l'}{1-k_r l}\right) + \theta_r \\
v_x = \sqrt{[\dot{s}(1-k_r l)]^2 + (\dot{s}l')^2} \\
a_x = \ddot{s}\dfrac{1-k_r l}{\cos(\theta_x-\theta_r)} + \dfrac{\dot{s}^2}{\cos(\theta_x-\theta_r)}\left[l'\left(k_x\dfrac{1-k_r l}{\cos(\theta_x-\theta_r)} - k_r\right) - (k_r' l + k_r l')\right] \\
k_x = \left(\left(l'' + (k_r' l + k_r l')\tan(\theta_x-\theta_r)\right)\dfrac{\cos^2(\theta_x-\theta_r)}{1-k_r l} + k_r\right)\dfrac{\cos(\theta_x-\theta_r)}{1-k_r l}
\end{cases}
$$

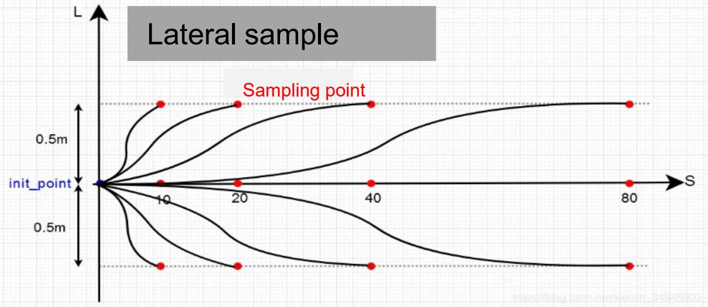
$$
\begin{cases}
x_x = x_r - l\sin(\theta_r) \\
y_x = y_r + l\cos(\theta_r) \\
\theta_x = \arctan\left(\dfrac{l'}{1-k_r l}\right) + \theta_r \\
v_x = \sqrt{[\dot{s}(1-k_r l)]^2 + (\dot{s}l')^2} \\
a_x = \ddot{s}\overset{l}{}\dfrac{1-k_r l}{\cos(\theta_x-\theta_r)} + \dfrac{\dot{s}^2}{\cos(\theta_x-\theta_r)}\left[l'\overset{l}{}\left(k_x\dfrac{1-k_r l}{\cos(\theta_x-\theta_r)} - k_r\right) - (k_r' l + k_r l')\right] \\
k_x = \left(\left(l'' + (k_r' l + k_r l')\tan(\theta_x-\theta_r)\right)\dfrac{\cos^2(\theta_x-\theta_r)}{1-k_r l} + k_r\right)\dfrac{\cos(\theta_x-\theta_r)}{1-k_r l}
\end{cases}
$$

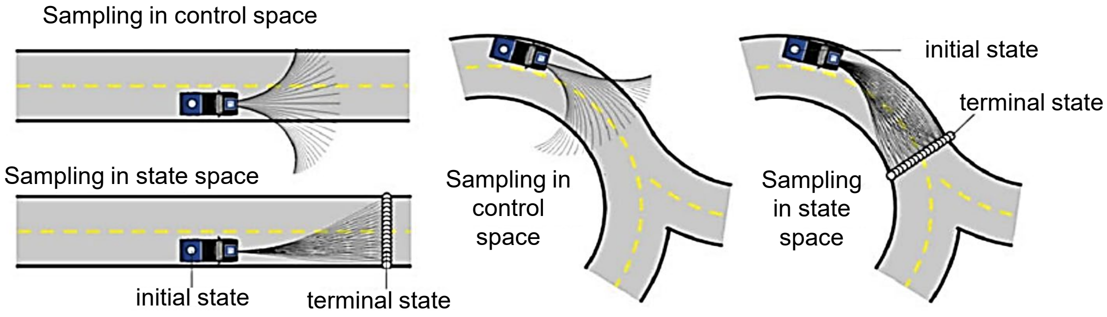
Lattice planner sampling

Lattice planner sampling includes:

- Lateral sampling
- Longitudinal sampling
- Time sampling

- State space: Those arising from the environment
- Control space: Those arising from vehicle mobility





State Space Sampling of Feasible Motions for High-Performance Mobile Robot Navigation in Complex Environments, Thomas M. Howard, Colin J. Green, and Alonzo Kelly

Lattice planner: speed planning

- Formulate lateral planning function $l(s)$ and longitudinal planning function $s(t)$ using polynomials based on the sampling.
- Typically, 4th or 5th order polynomials are used to ensure the smoothness of the path.

$\boxed{\text{In stop and go, or spacing control (5}^{\text{th}}\text{ order)}}$

$$s(t) = c_1t^5 + c_2t^4 + c_3t^3 + c_4t^2 + c_5t + c_6$$

$$v(t) = 5c_1t^4 + 4c_2t^3 + 3c_3t^2 + 2c_4t + c_5$$

$$a(t) = 20c_1t^3 + 12c_2t^2 + 6c_3t + 2c_4$$

$\boxed{\text{In cruse control (4}^{\text{th}}\text{ order)}}$

$$s(t) = b_1t^4 + b_2t^3 + b_3t^2 + b_4t + b_5$$

$$v(t) = 4b_1t^3 + 3b_2t^2 + 2b_3t + b_4$$

$$a(t) = 12b_1t^2 + 6b_2t + 2b_3$$

Lattice planner: speed planning

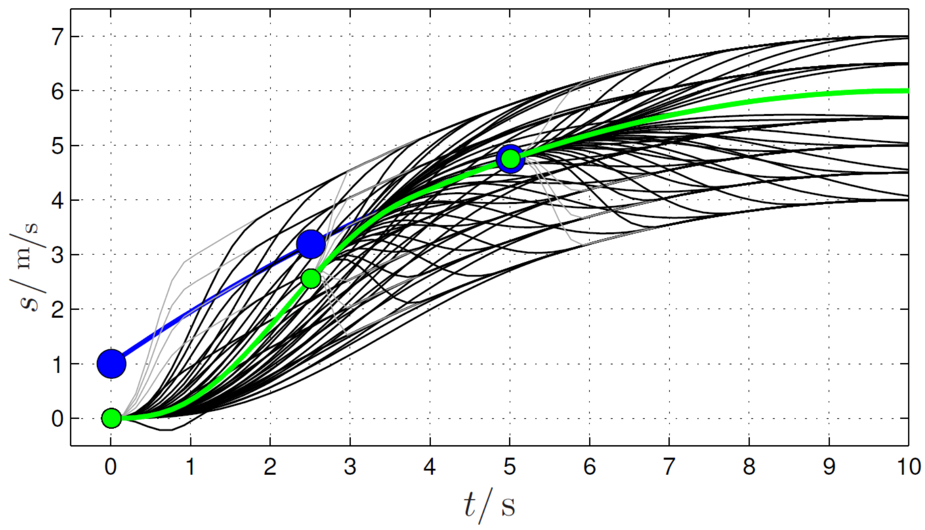
Longitudinal fitting polynomial solution:

In stop and go, or spacing control (5th order example)

$$s(t) = c_1 t^5 + c_2 t^4 + c_3 t^3 + c_4 t^2 + c_5 t + c_6$$

$$v(t) = 5c_1 t^4 + 4c_2 t^3 + 3c_3 t^2 + 2c_4 t + c_5$$

$$a(t) = 20c_1 t^3 + 12c_2 t^2 + 6c_3 t + 2c_4$$



Constraint functions:

$$s(t_0) = c_6 = s_0$$

$$v(t_0) = c_5 = ts_0$$

$$a(t_0) = 2c_4 = tts_0$$

$$s(t_1) = c_1 t_1^5 + c_2 t_1^4 + c_3 t_1^3 + c_4 t_1^2 + c_5 t_1 + c_6 = s_1$$

$$v(t_1) = 5c_1 t_1^4 + 4c_2 t_1^3 + 3c_3 t_1^2 + 2c_4 t_1 + c_5 = ts_1$$

$$a(t_1) = 20c_1 t_1^3 + 12c_2 t_1^2 + 6c_3 t_1 + 2c_4 = tts_1$$

Solving the equations to get coefficients

Lattice planner: speed planning

Longitudinal fitting polynomial solution:

In cruise control (4th order example)

$$s(t) = b_1 t^4 + b_2 t^3 + b_3 t^2 + b_4 t + b_5$$

$$v(t) = 4b_1 t^3 + 3b_2 t^2 + 2b_3 t + b_4$$

$$a(t) = 12b_1 t^2 + 6b_2 t + 2b_3$$

Constraint functions:

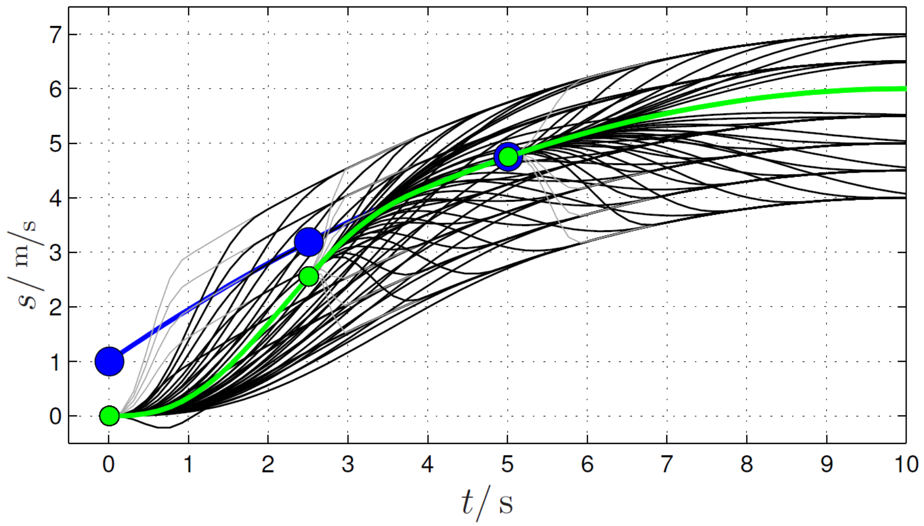
$$s(t_0) = b_5 = s_0$$

$$v(t_0) = b_4 = ts_0$$

$$a(t_0) = 2b_3 = tts_0$$

$$v(t_1) = 4b_1 t_1^3 + 3b_2 t_1^2 + 2b_3 t_1 + b_4 = ts_1$$

$$a(t_1) = 12b_1 t_1^2 + 6b_2 t_1 + 2b_3 = tts_1$$



Solving the equations to get coefficients

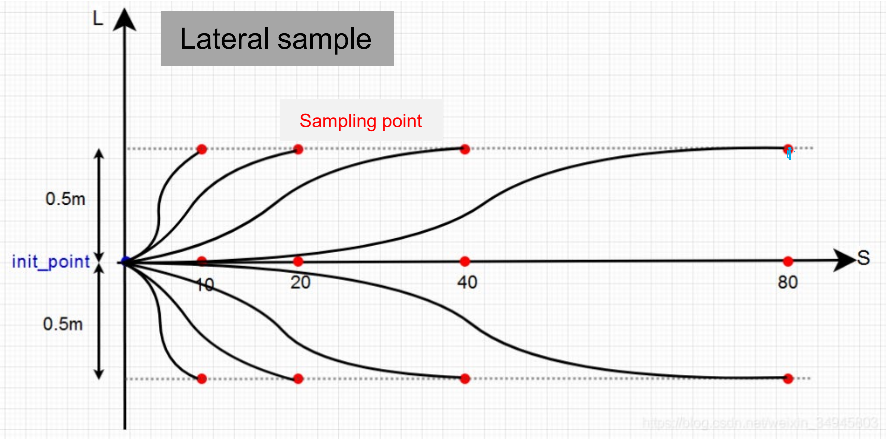
Lattice planner: lateral trajectory planning

Lateral fitting polynomial solution

$$d(s) = k_1 s^5 + k_2 s^4 + k_3 s^3 + k_4 s^2 + k_5 s + k_6$$

$$d_v(t) = 5k_1 s^4 + 4k_2 s^3 + 3k_3 s^2 + 2k_4 s + k_5$$

$$d_a(t) = 20k_1 s^3 + 12k_2 s^2 + 6k_3 s + 2k_4$$



Constraint function

$$d(s_0) = k_1 s_0^5 + k_2 s_0^4 + k_3 s_0^3 + k_4 s_0^2 + k_5 s_0 + k_6 = d_0$$

$$d_v(s_0) = 5k_1 s_0^4 + 4k_2 s_0^3 + 3k_3 s_0^2 + 2k_4 s_0 + k_5 = sd_0$$

$$d_a(s_0) = 20k_1 s_0^3 + 12k_2 s_0^2 + 6k_3 s_0 + 2k_4 = ssd_0$$

$$d(s_1) = k_1 s_1^5 + k_2 s_1^4 + k_3 s_1^3 + k_4 s_1^2 + k_5 s_1 + k_6 = d_1$$

$$d_v(s_1) = 5k_1 s_1^4 + 4k_2 s_1^3 + 3k_3 s_1^2 + 2k_4 s_1 + k_5 = sd_1$$

$$d_a(s_1) = 20k_1 s_1^3 + 12k_2 s_1^2 + 6k_3 s_1 + 2k_4 = ssd_1$$

Constraint variables:

d_0: initial lateral displacement

sd_0: initial lateral speed

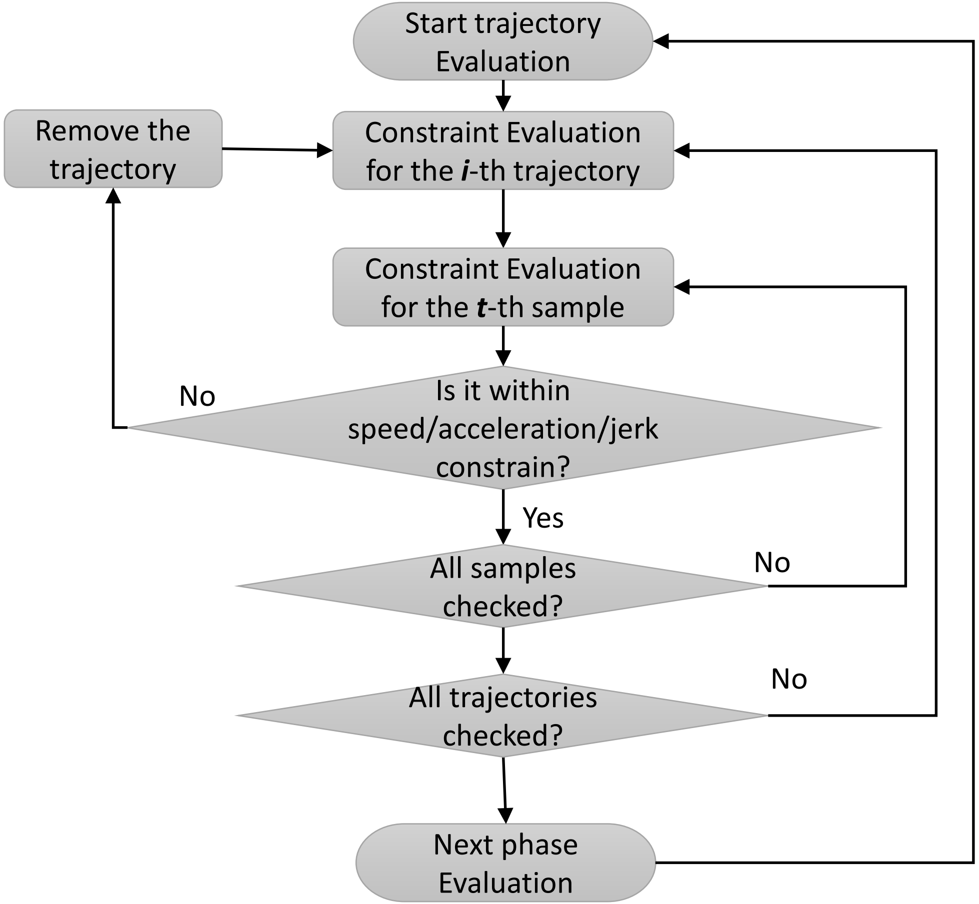
ssd_0: initial lateral acceleration

d_1: sampled lateral displacement

sd_1: sampled lateral speed

ssd_1: sampled lateral acceleration

Lattice planner: Constraint evaluation



For every trajectory generated, it needs to be evaluated to check if it violates any constraint and remove it if it does.

Lattice planner: trajectory cost function

Objective: to choose a feasible path that is nearest to the static reference path, and at the same time, avoid large speed change to ensure comfortability and stay away from obstacles.

$$J = k_{\mathrm{longi}} * J_{longi} + k_{comfort} * J_{comfort} + k_{collision} * J_{collision}$$

where

- k_{longi}: weight on longitudinal objective cost
- J_{longi}: tracking cost, considering speed error, distance to travel.
- $k_{comfort}$: weight on comfort cost
- $J_{comfort}$: comfort cost, considering longitudinal jerk
- $k_{collision}$: weight on collision cost.
- $J_{collision}$: cost of collision to the near objects.

Lattice planner: trajectory cost function

Longitudinal Objective achievement cost : to choose a feasible path that is nearest to the static reference path.

- Input: $Lon\_trajectory, planning\_target, reference\_s\_dot$
- $J_{speed} = \frac{\sum_{t=0}^{length} t^2 \cdot |V_{ref_t} - V_{evaluation_t}|}{\sum_{t=0}^{length} t^2}$
- $J_{dist} = \frac{1}{1+dist}$
- $J_{\mathrm{longi}} = \frac{W_{speed} Cost_{speed} + W_{dist} Cost_{dist}}{W_{speed} + W_{dist}}$

Lattice planner: trajectory cost function

Comfort Objective: to choose a feasible path that is having less *jerk*

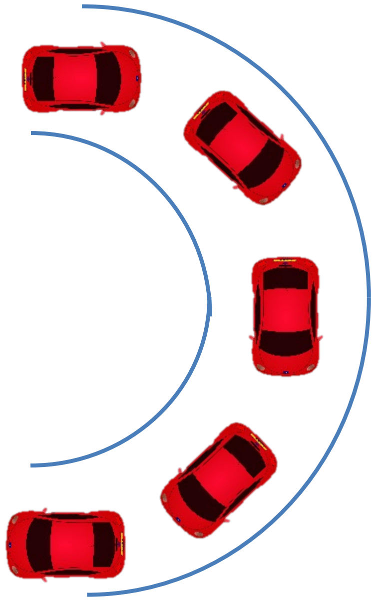
$$J_{comfort} = \frac{\sum_{t=0}^{length}(jerk_t)^2}{1 + \sum_{t=0}^{length}|jerk_t|}$$

$$J_{comfort} = \max(jerk)$$

Lattice planner: trajectory cost function

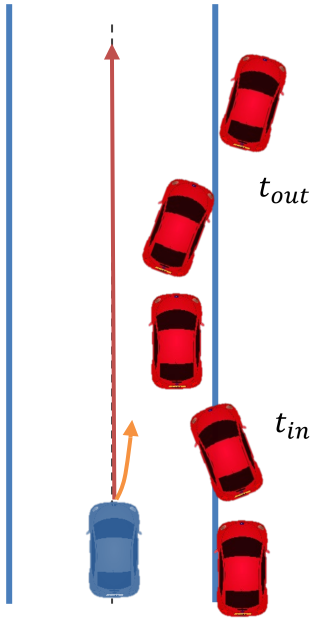
Centripetal Objective: to choose a feasible path that is having less *less centripetal accel jerk*.

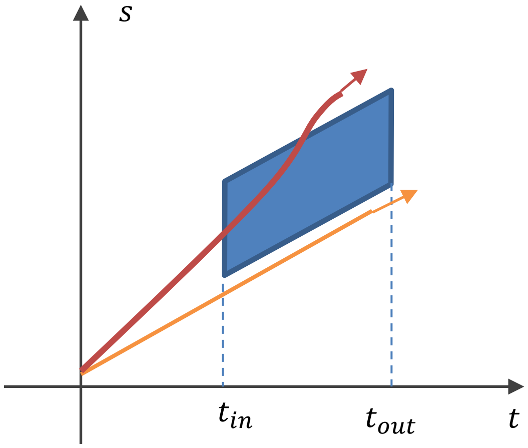
- $a_{centr\,t} = \frac{v_t^2}{R_t} = v_t^2\ k_t$
- $J_{CentriAcc} = \frac{\sum_{t=0}^{length} a_{centri_t}^2}{\sum_{t=0}^{length} |a_{centri_t}|}$



Lattice planner: trajectory cost function

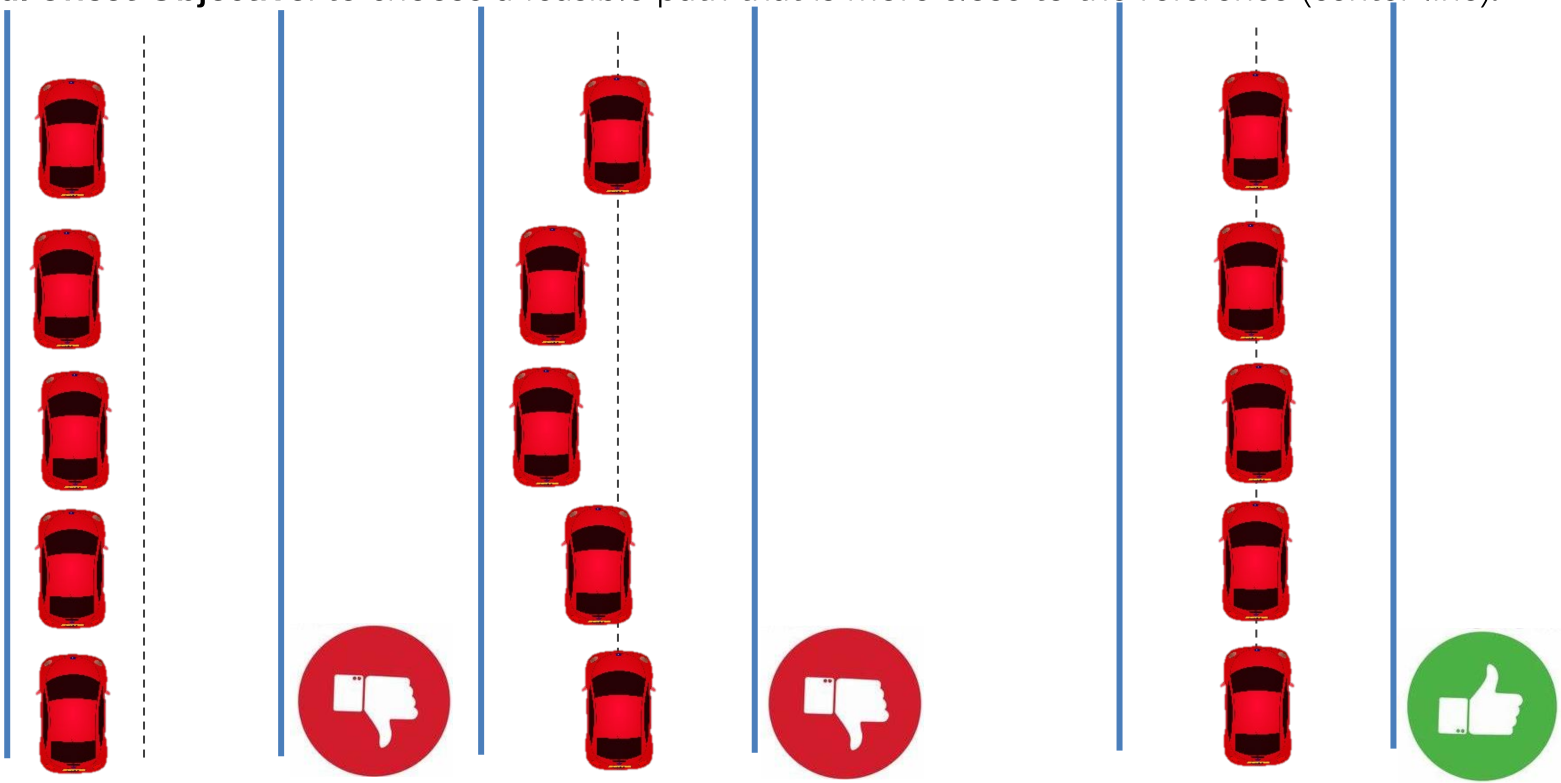
Collision Objective: to choose a path that is furthest from obstacles





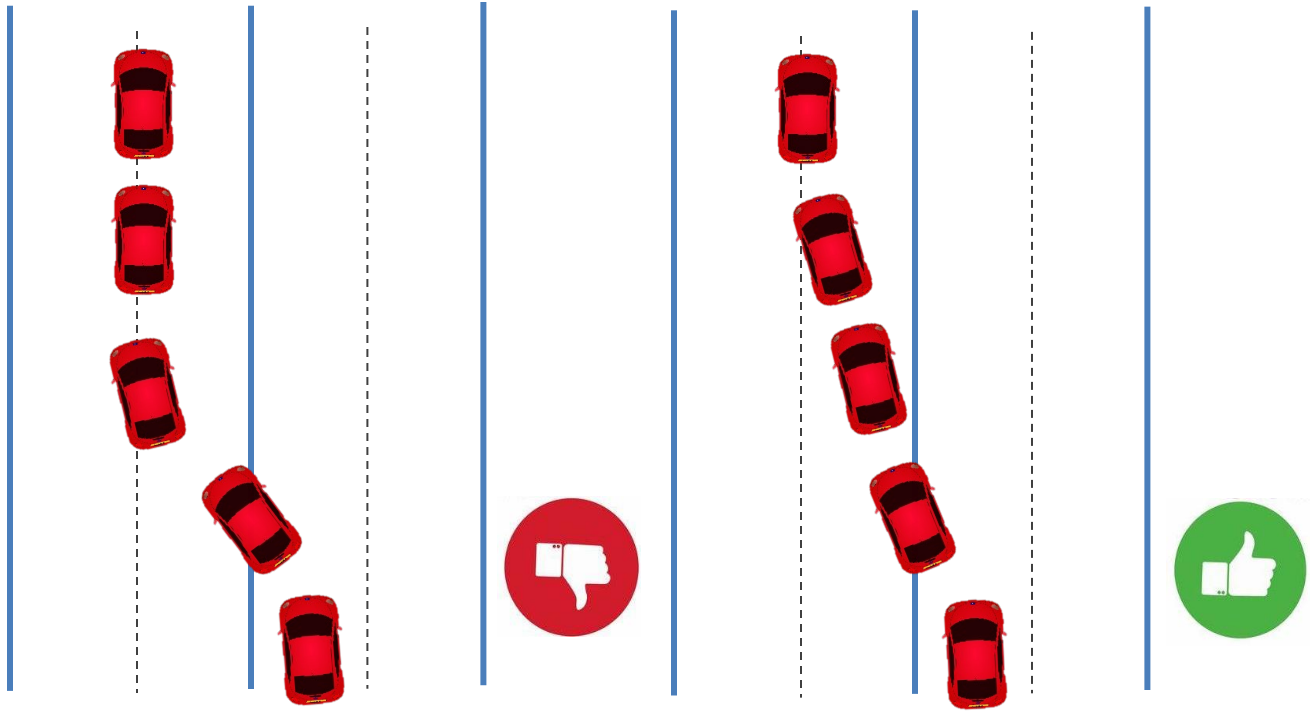
Lattice planner: trajectory cost function

Lateral offset Objective: to choose a feasible path that is more close to the reference (center line).



Lattice planner: trajectory cost function

Lateral Acceleration Objective: to choose a feasible path that is having smoother lane change



Summary

- Motion planner
  - Motion planner basic concept
  - Functionality and Common method
  - Fundamental Concepts and Key terminology
- Stochastic Sampling Methods
  - RRT
  - RRT*
  - Other improve methods
- Lattice planner
  - Frenet Coordinate
  - Speed planning
  - Trajectory planning

深蓝学院
shenlanxueyuan.com

Thanks for Listening !