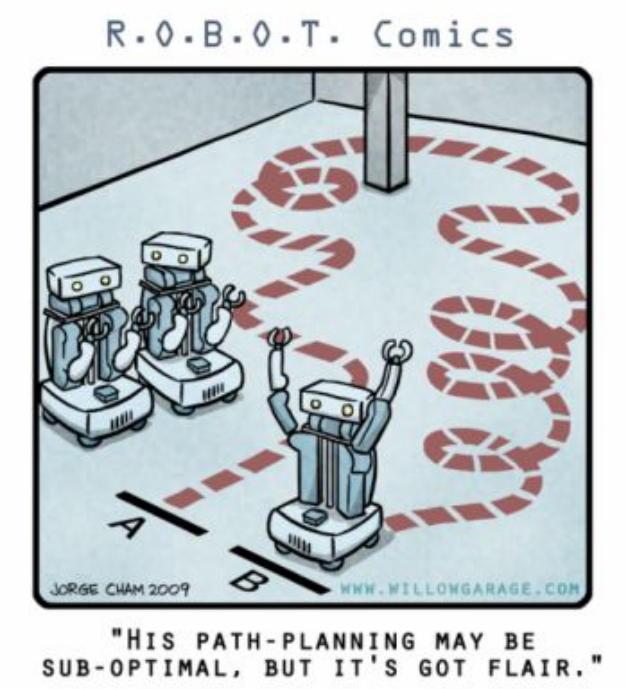
Sampling-based Motion Planning

Hongrui Zheng (hongruiz@seas.upenn.edu)

Penn Engineering

Lesson Plan

- Local planners
- Using RRT as a local planner
- Clothoids
- State Lattice Planner
- Graph-based Planner

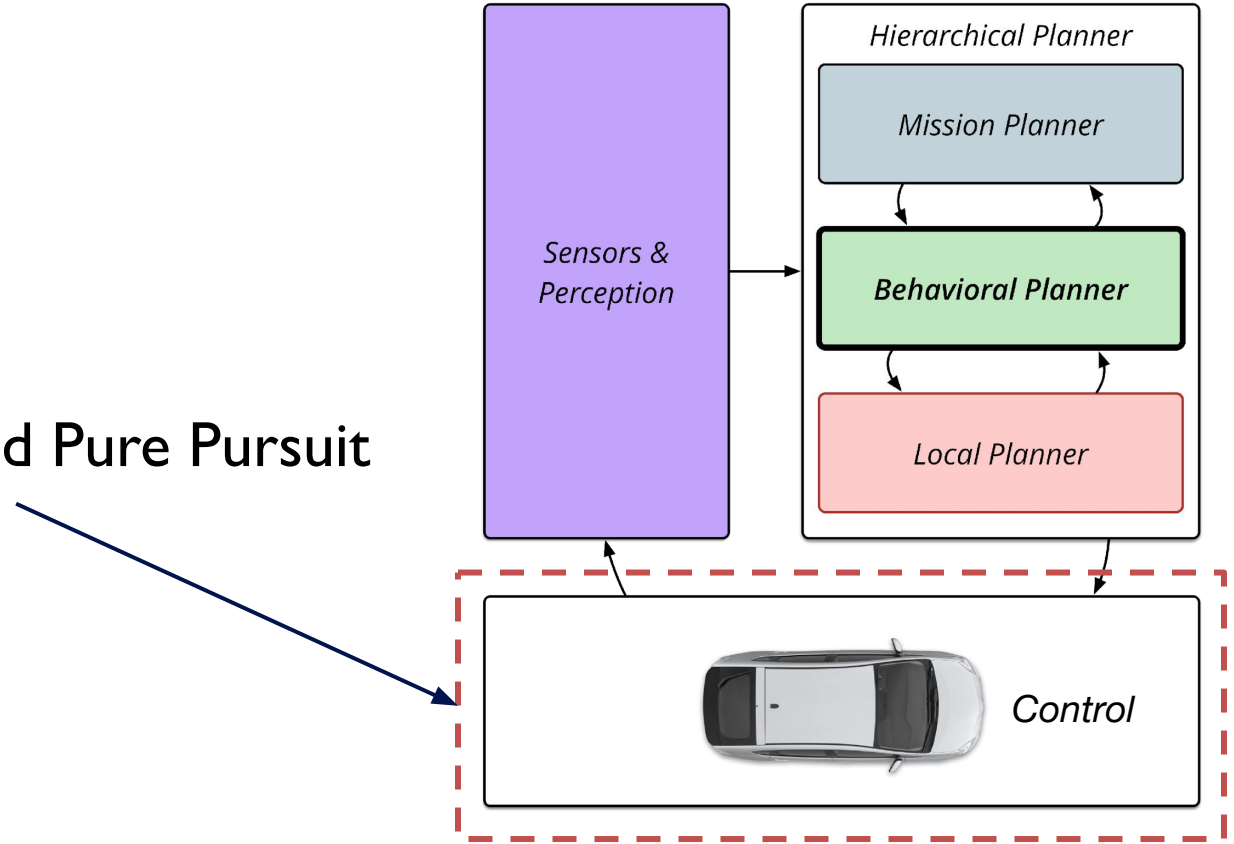


http://www.willowgarage.com/pages/research/motion-planning

Previously

Path tracking

We've covered Pure Pursuit

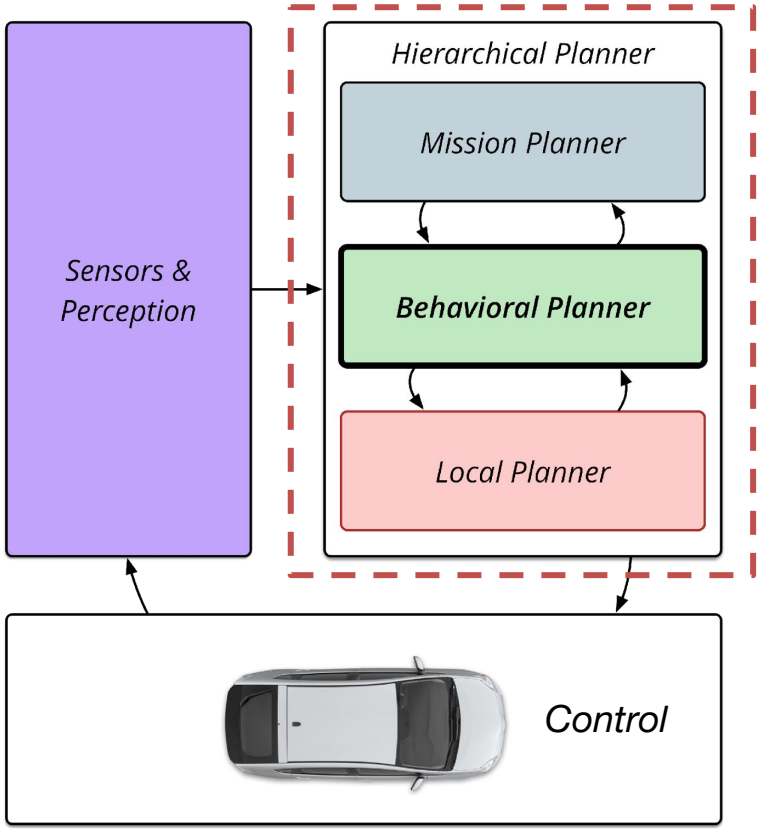


The Planning module

Mission/Global Planner:
What is the overall goal of the vehicle?

Behavioral Planner:
What rules should the vehicle follow in different situations?

Local Planner:
What are viable trajectories from position to goals?

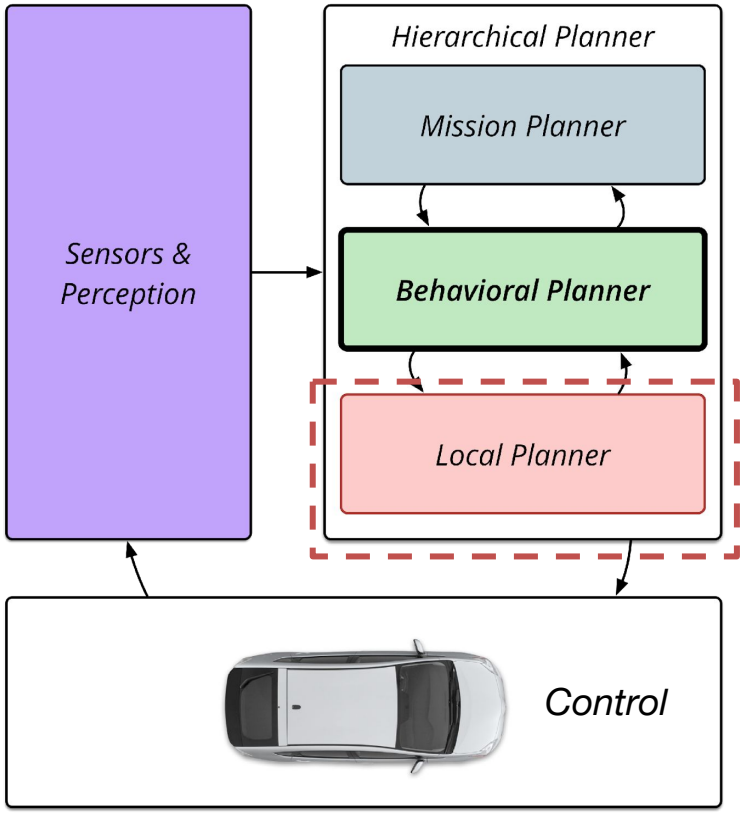


The Planning module

Mission/Global Planner:
What is the overall goal of the vehicle?

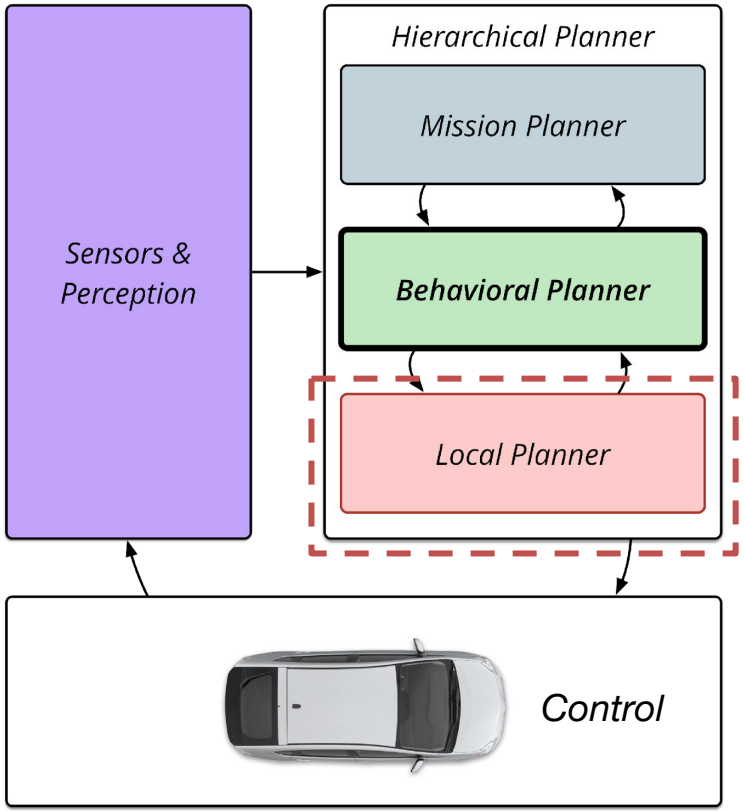
Behavioral Planner:
What rules should the vehicle follow in different situations?

Local Planner:
What are viable trajectories from position to goals?



The Local Planner

- How do we differentiate free space and occupied space?
- How do we generate a trajectory that spans the free space?
- How do we efficiently generate these trajectories for online planning?



The Motion Planning Problem

Given:

1. A robot with configuration space: C
2. The set of obstacles: C_{obs}
3. An initial configuration: q_{init}
4. A goal configuration: q_{goal}
5. (Possibly a cost function)

The Motion Planning Problem

Goal:

Find a path x (continuous function): $[0, 1] \rightarrow C$ such that the path:

1. Starts from the initial configuration $x(0) = q_{init}$
2. Reaches the goal configuration $x(1) = q_{goal}$
3. Avoids collision with obstacles $x(s) \notin C_{obs}$ for all $s \in [0, 1]$
4. (Possibly is the minimum cost path)

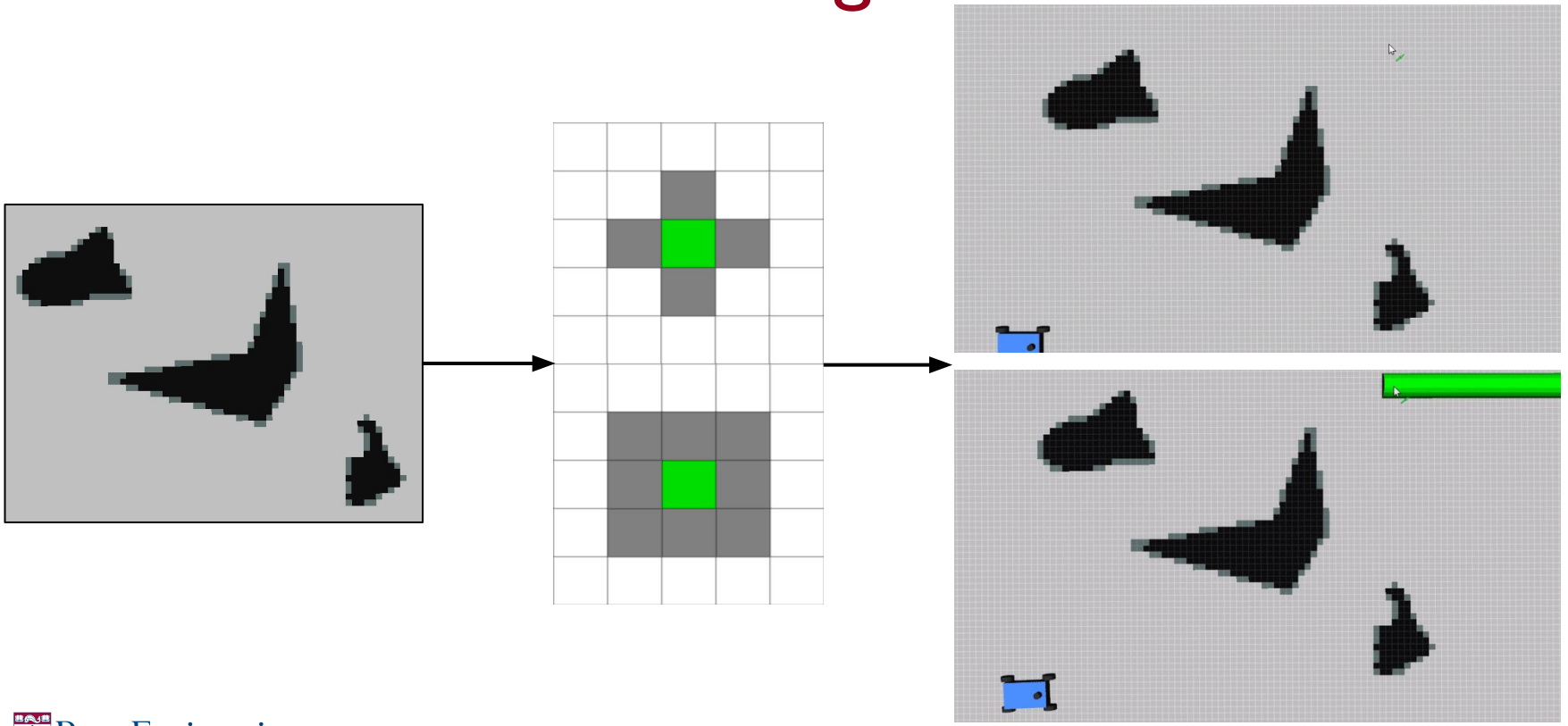
Complete and Optimal Motion Planning

- An algorithm is said to be **complete** if
  - It terminates in finite time and
  - It returns a solution when one exists and it returns failure otherwise
- An algorithm is said to be **optimal** if
  - It returns the minimum cost solution when a solution exists
- There are complete algorithms for the motion planning.
  - All of these algorithms require run time that increase exponentially fast with increasing number of degrees of the robot
  - Unfortunately, most of these algorithms are not practical.
- All practical algorithms relax completeness guarantees one way or another.

Motion Planning Representations

- The **core** of motion planning algorithms:
  - Converting the formal problem into a model that allows **search**
- Search: (usually) the final step of motion planning
- We'll go over multiple ways to construct different representations:
  - Trees, grids, and graphs

Discrete Motion Planning



Continuous Motion Planning

Goal: Plan collision-free trajectories through cluttered space.

C-space vs. workspace

Configuration space

- **Configuration:** a complete specification of the position of every point in the system.
- The space of **all** configurations is the **configuration space** or **c-space**

Workspace

- The coordinates of the robot are specified in the same coordinate system of the world it's manipulating.
- Even without obstacles, not all workspace coordinates are attainable.

C-space vs. workspace in the race car context

Configuration space

- The dimension of the c-space of the race car depends on how we define the full state.
- For example, the c-space could have: (x, y, z, heading angle, linear velocity, angular velocity, motor RPM, …)

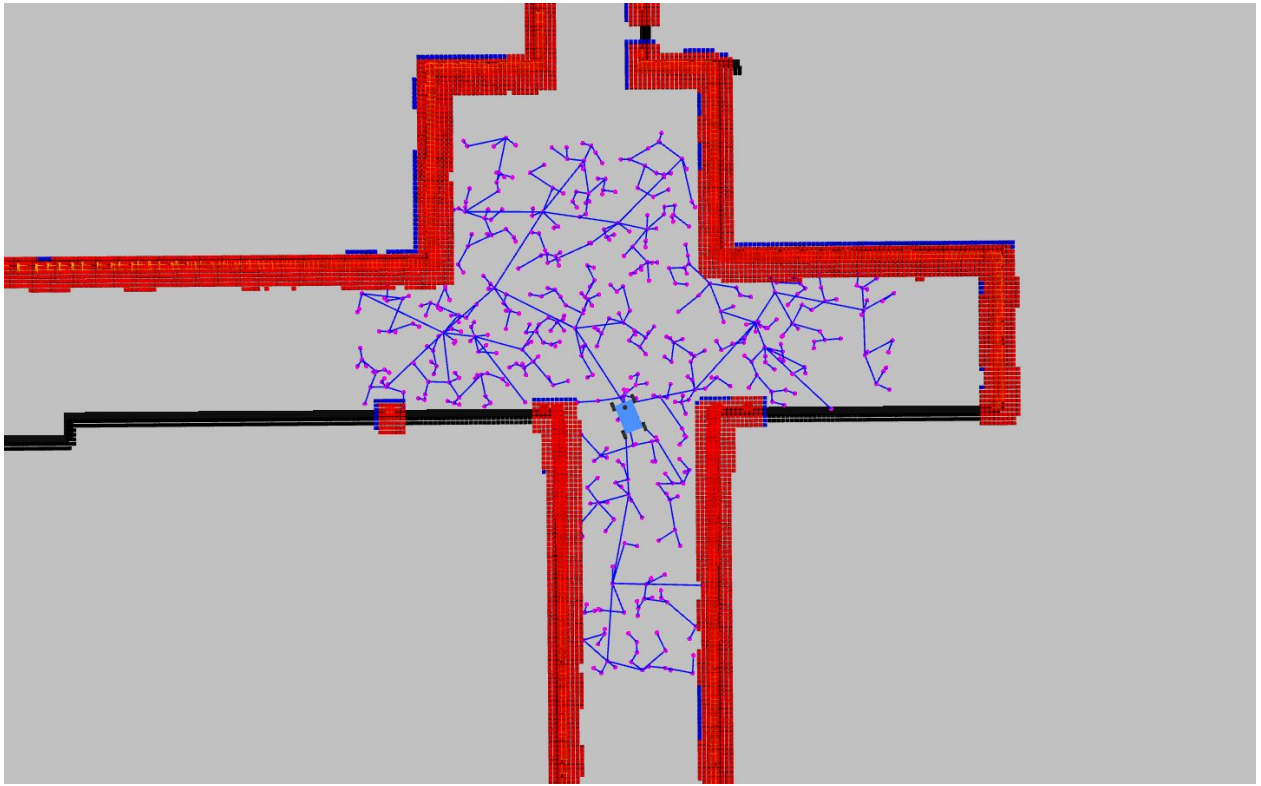
Workspace

- The car's pose (x, y, z, θ) in real world coordinates.

Motion Planning Challenges for the Race Car

- **Dimensionality**: as the state of the robot we define encodes more information, the higher the dimensionality of the planning space
- **Responsiveness**: the planning algorithm needs to react fast when the car is travelling at high speeds
- **Obstacles**: potentially dynamic and moving obstacles (head to head racing)
- **Motion constraints**: kinematic or dynamic
- ...

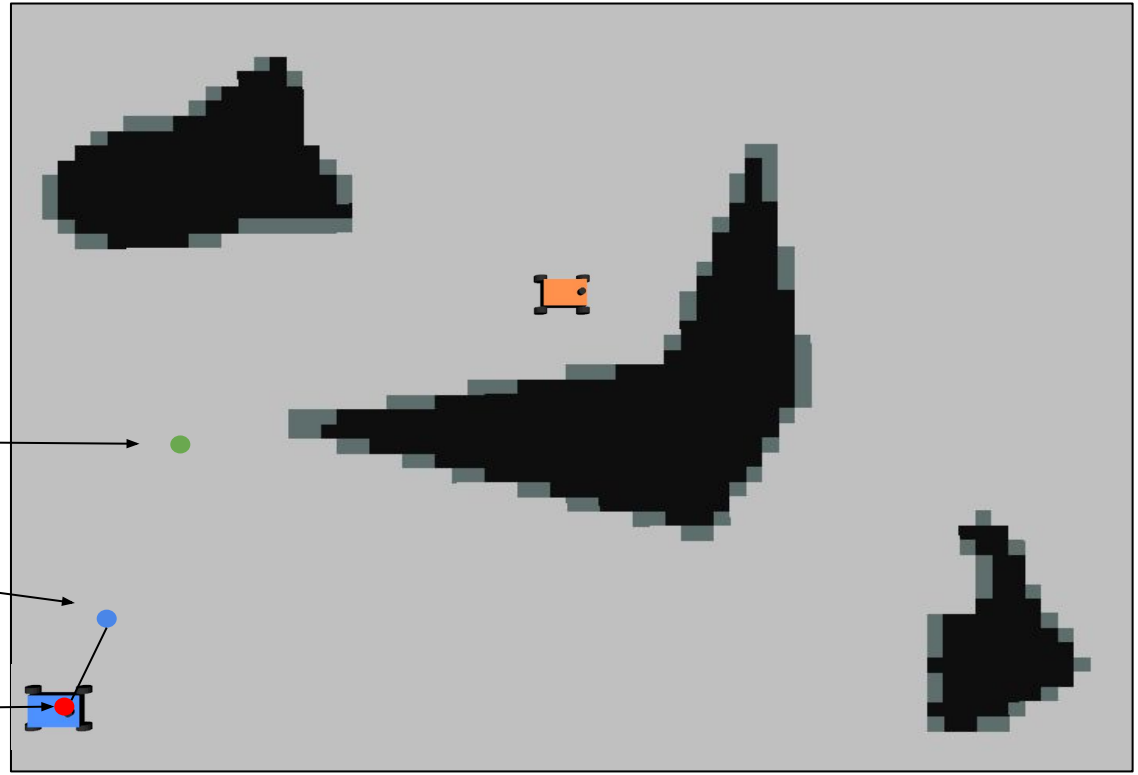
RRT (Rapidly-exploring Random Trees)



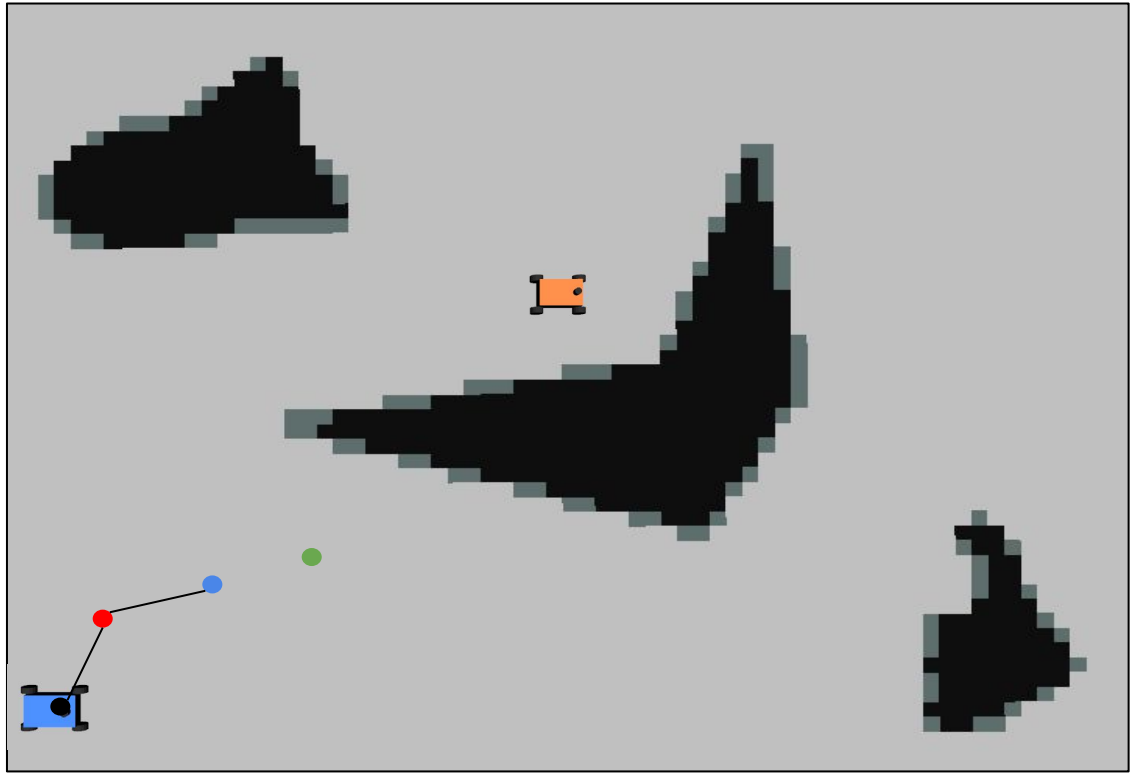
RRT

```
function rrt(x_init, max_iter):
    V ← {x_init}                                   // vertices
    E ← ∅                                          // edges

    for i = 1:max_iter:                            // maximum number of expansions
        x_rand ← SampleFree()                      // collision free random configuration
        x_nearest ← Nearest(G=(V,E), x_rand)       // closest neighbor in the tree
        x_new ← Steer(x_nearest, x_rand)           // expanding the tree

        if ObstacleFree(x_nearest, x_new):         // the edge is collision-free
            V ← V ∪ {x_new}
            E ← E ∪ {(x_nearest, x_new)}

    return G=(V,E)
```

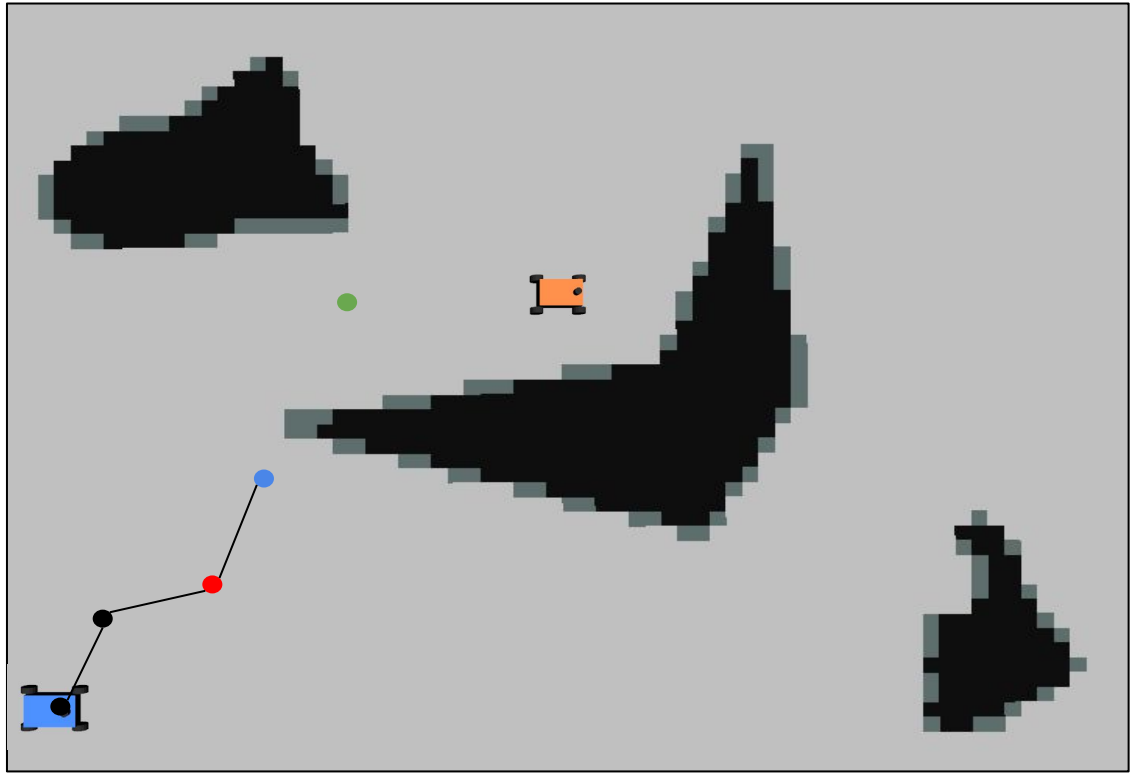
RRT


Random sample in free space
New node from steering
Nearest node on the tree

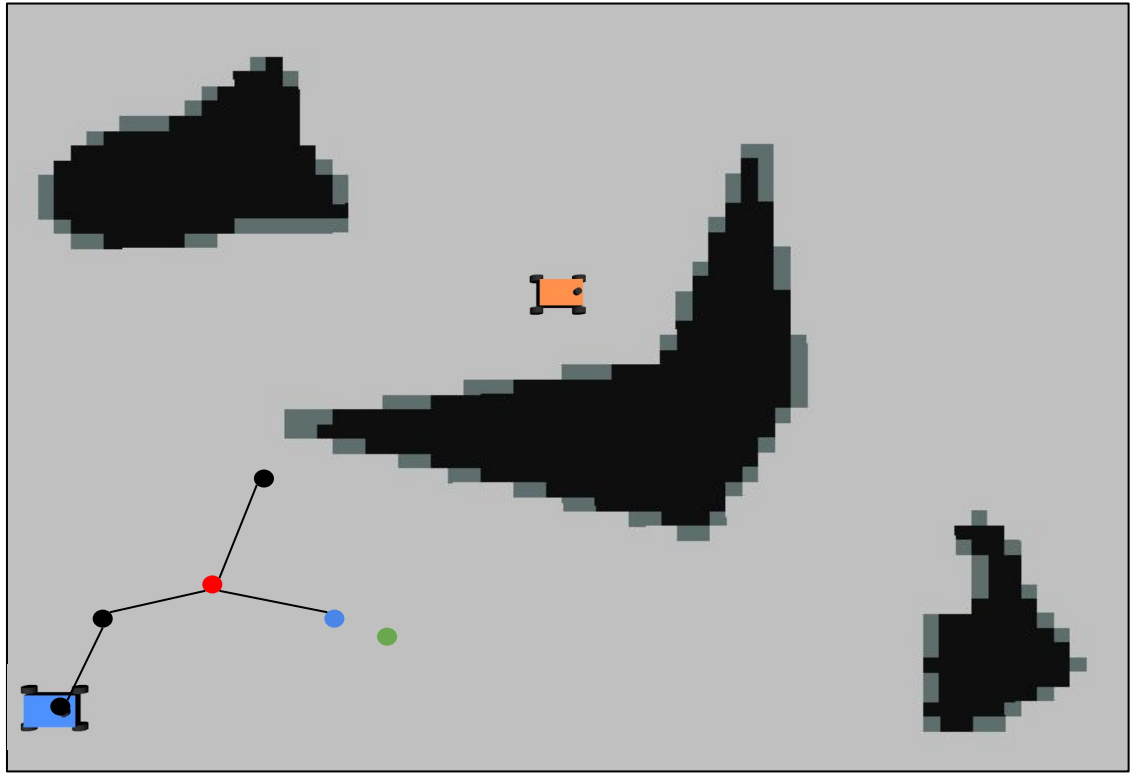
RRT



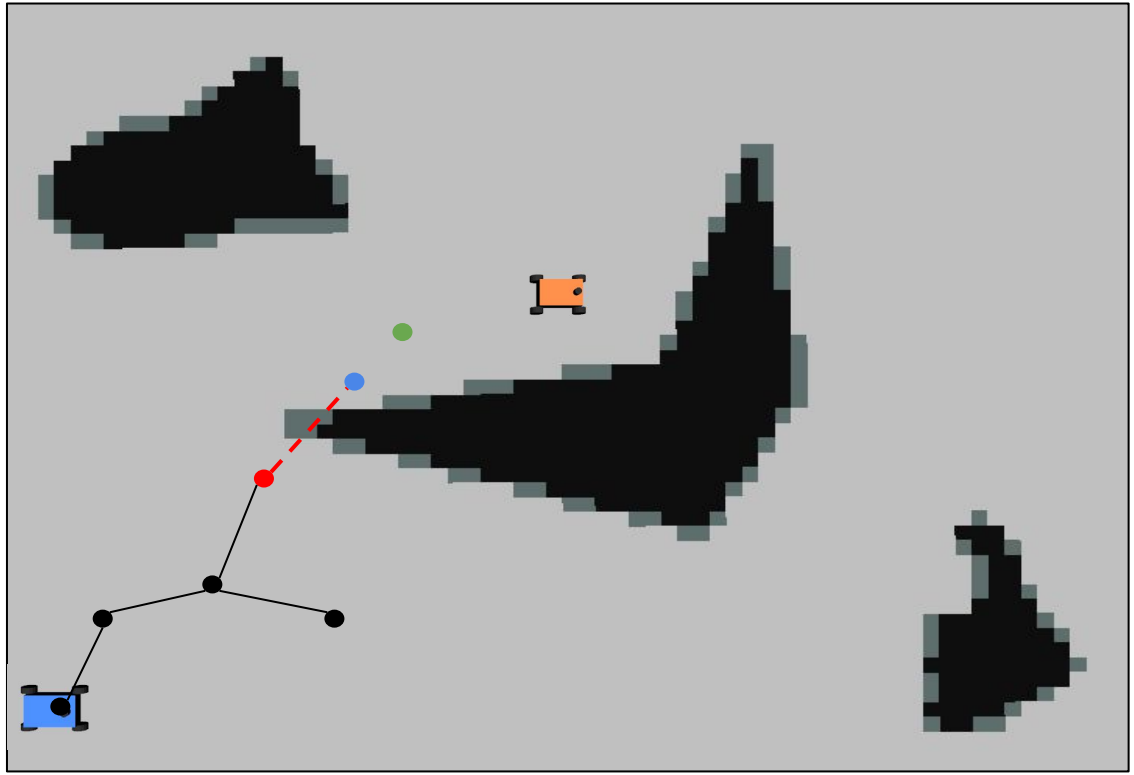
RRT



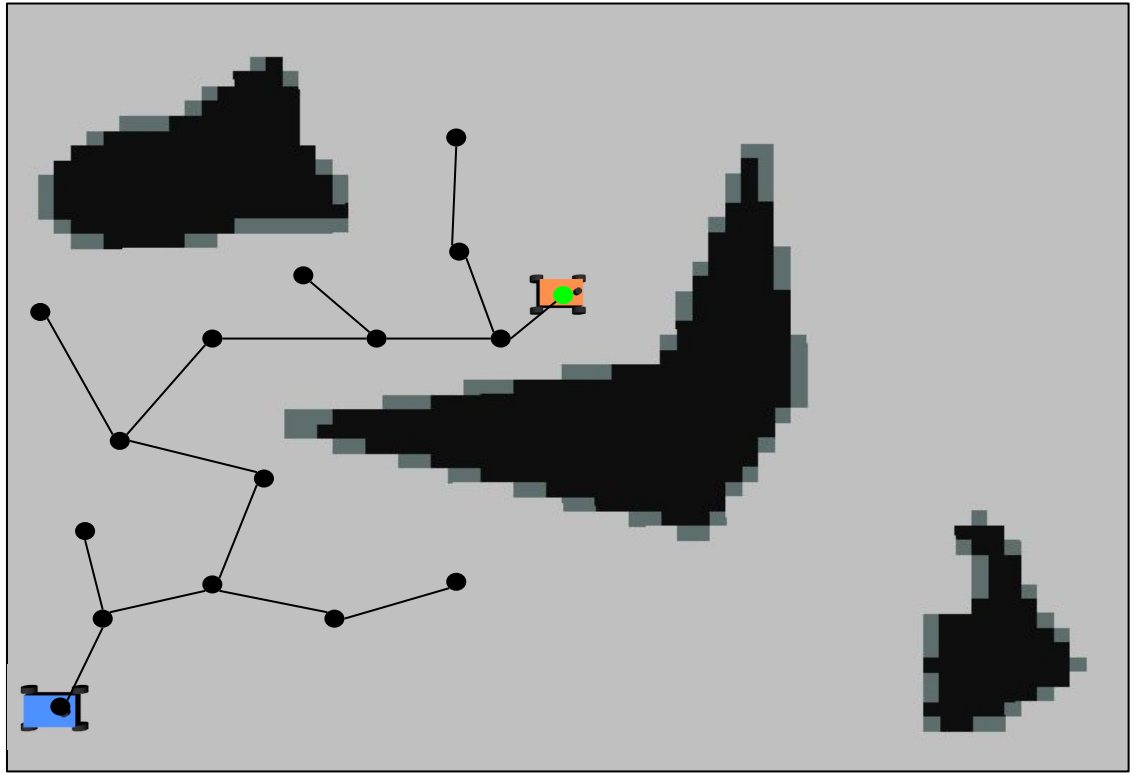
RRT



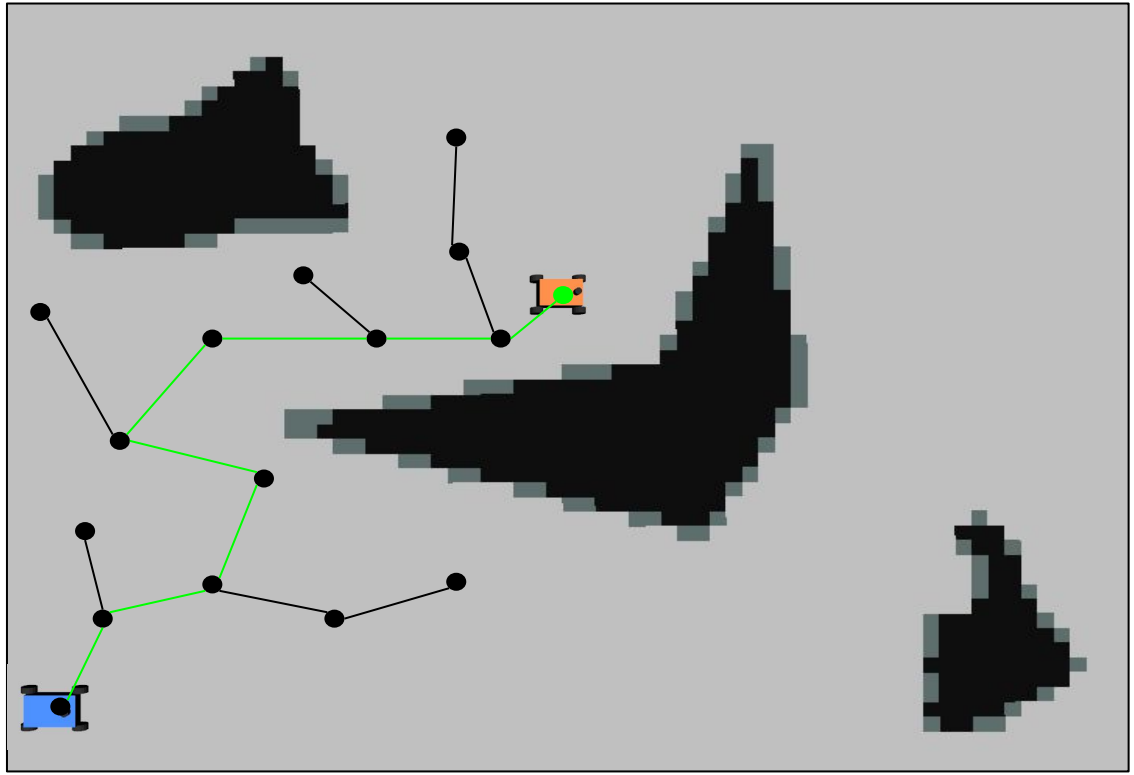
RRT



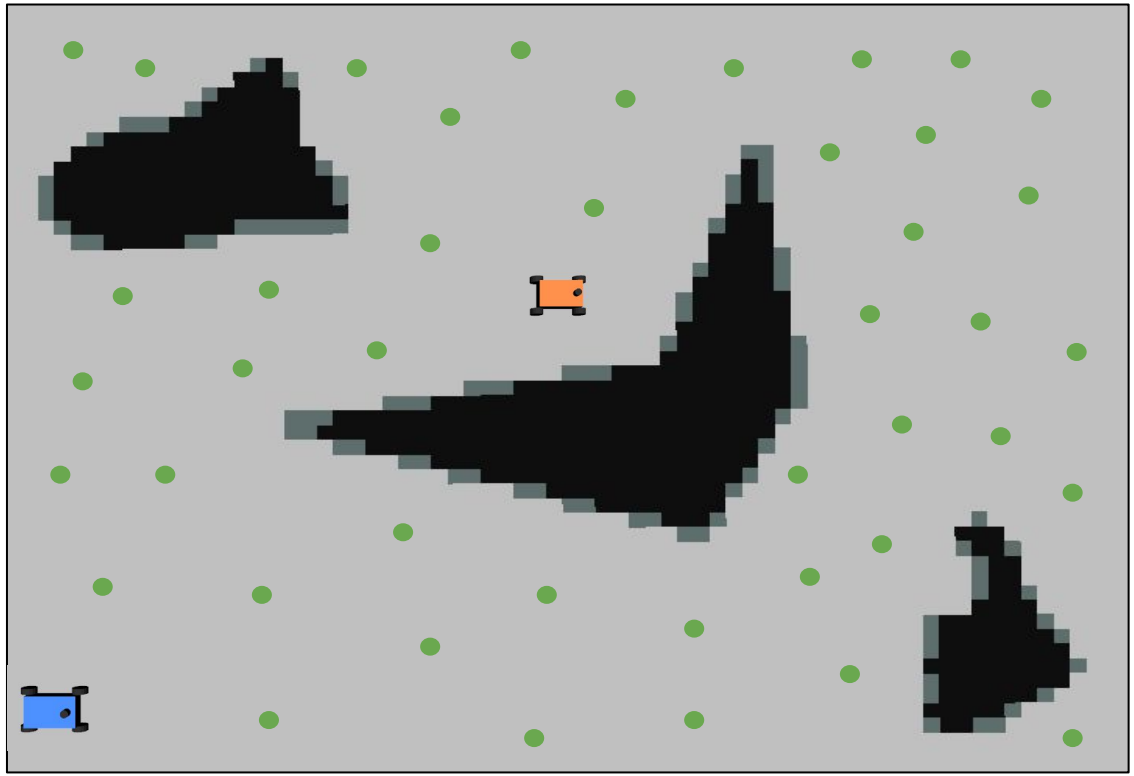
RRT



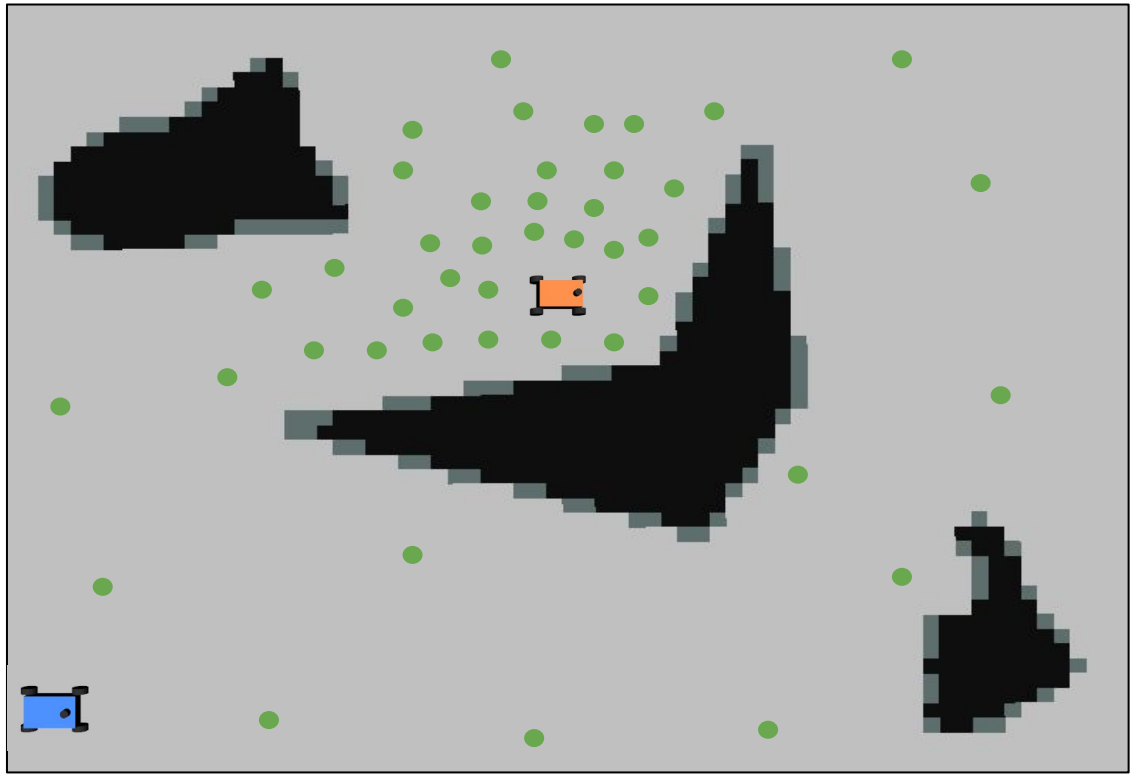
RRT



Sampling Strategy: Uniform vs. Biased



Sampling Strategy: Uniform vs. Biased



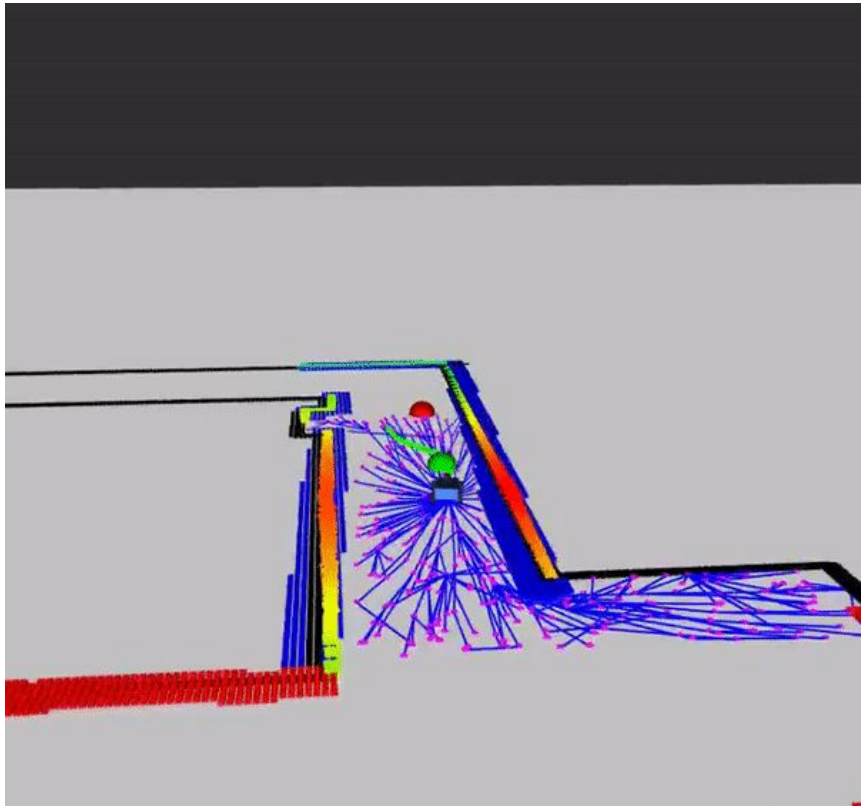
RRT: Completeness and Optimality

- Probabilistically complete:
  - The probability that the algorithm returns a solution, when one exists, approaches to 1 as the number of samples increases.

- Not optimal:
  - Due to random sampling.
  - There are variants of RRT that provides solutions with cost considered.

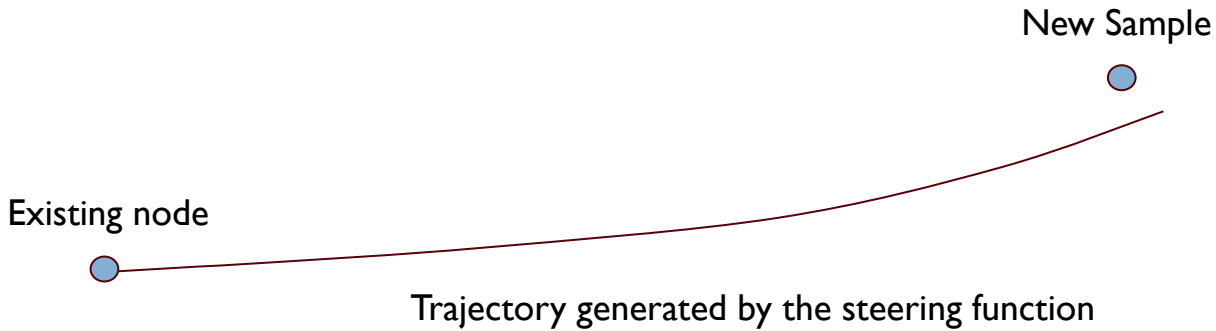
RRT as a Local Planner

1. Create local occupancy grid from scans.
2. Choose a local goal from reference path (similar to in pure pursuit).
3. Expand a tree to find a path in free space.
4. Track the current path with a tracker.
5. Get updates from scan and choose new local goal, repeat.



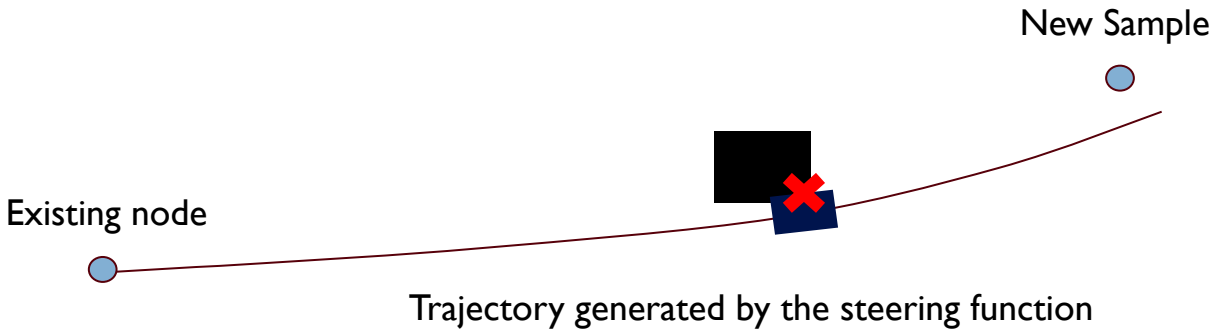
RRT for Motion Planning

- Need to design a "steering function" for your system based on its constraints.



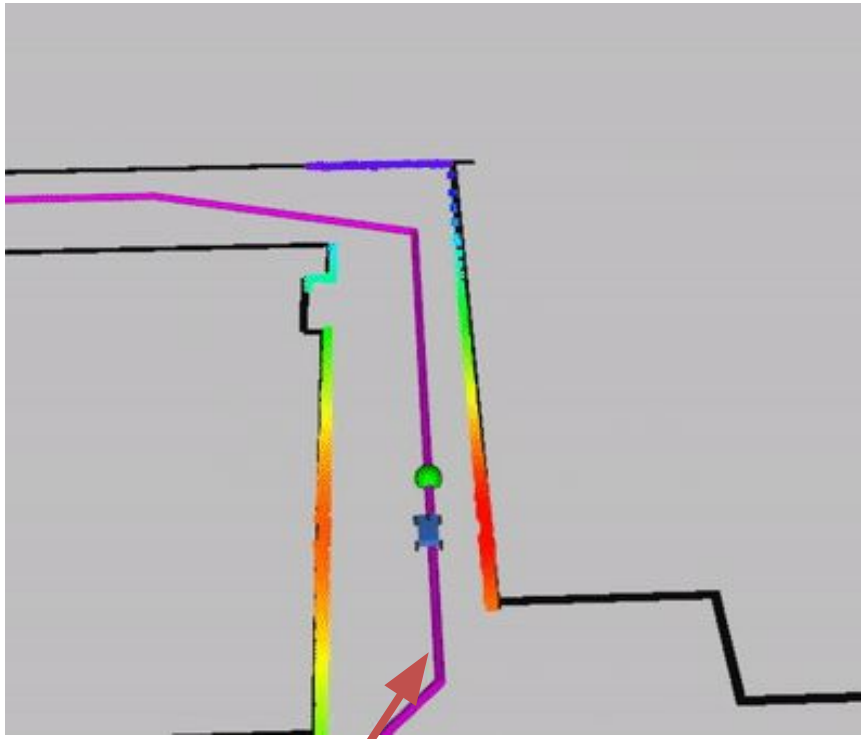
RRT for Motion Planning

- Need to design a collision checking function for your system.



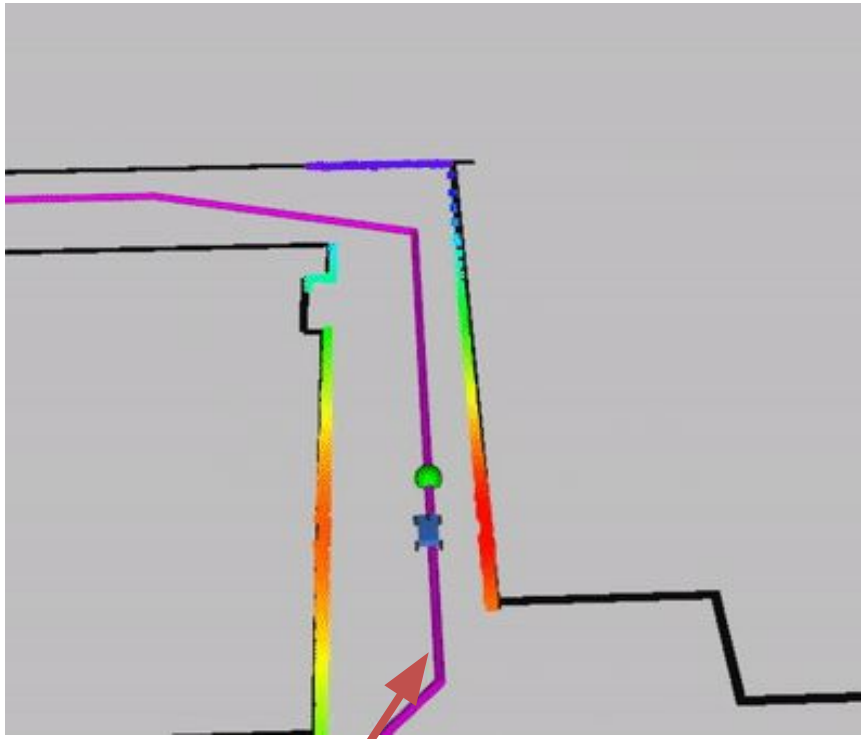
Paths

- Recall from pure pursuit, the behavior of the vehicle can be defined by the waypoints/the path.
- Prior to this we've been using a waypoint logger and some smoothing function to create a path.
- Is there a better solution?



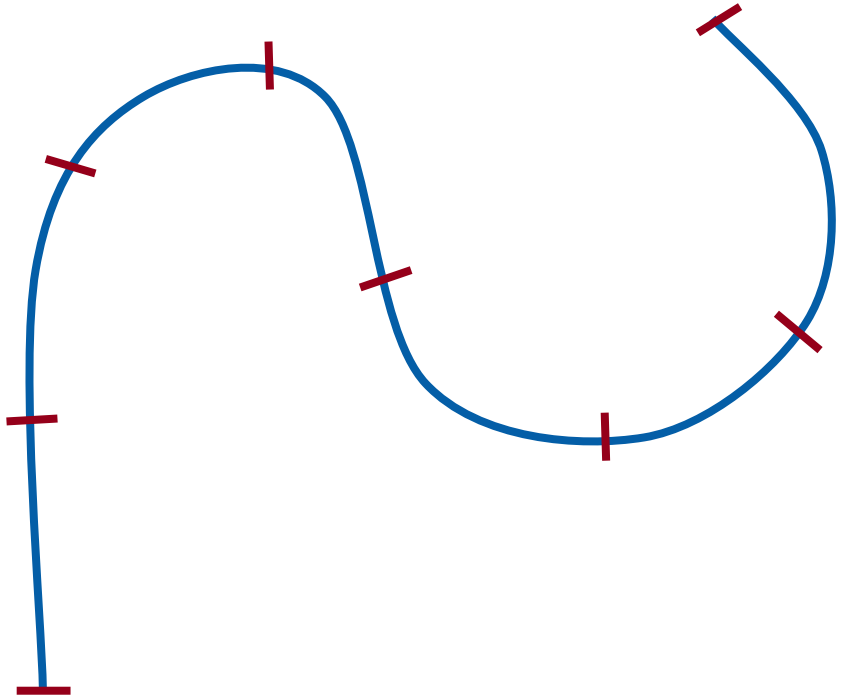
Paths

- Parameterize the path as a Spline.
  - The shape of the path can be changed by changing the parameters of the spline
  - Easier to sample the discrete points on the spline to define the path as (x, y) tuples.



Splines: Piecewise Polynomials

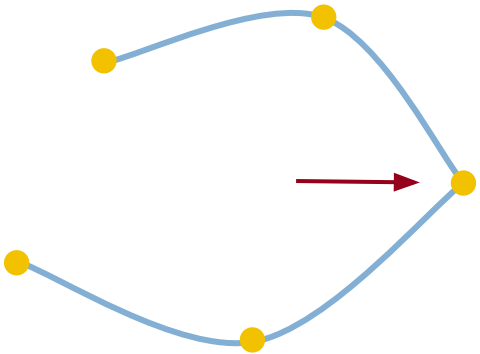
- A spline is a piecewise polynomial
  - Curve is broken into consecutive segments, each of which is a low-degree polynomial interpolating the control points.



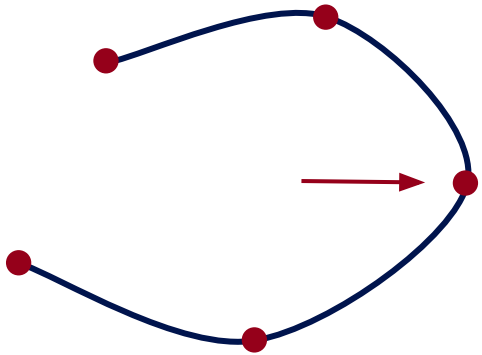
Splines: Piecewise Polynomials

How do we make sure all the segments fit together nicely?

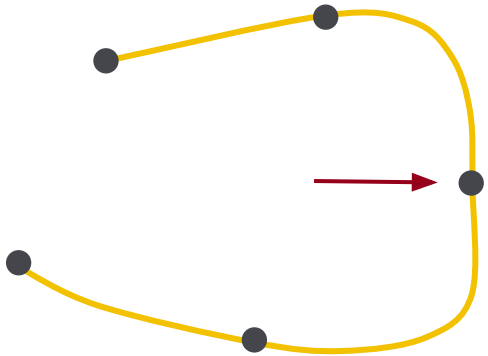
C_0 Continuity:
continuous in position



C_0 and C_1 Continuity:
continuous in position
and tangent line



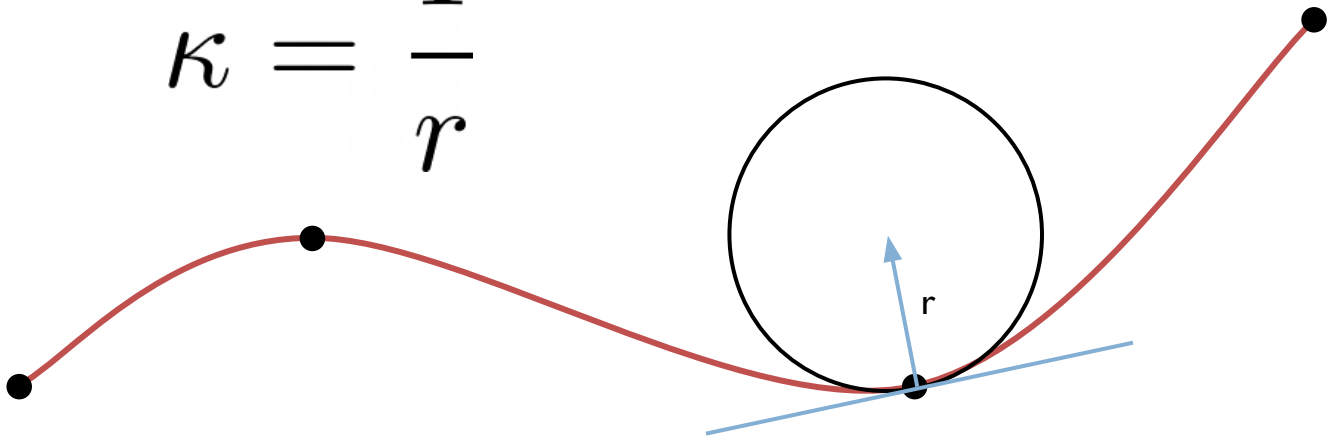
C_0, C_1 and C_2 Continuity:
continuous in position,
tangent line, and curvature



Curvature

The curvature of a differentiable curve is defined through the *osculating circle*, which is the circle that best approximates the curve at a point. And the curvature at that point is calculated by:

$$\kappa = \frac{1}{r}$$

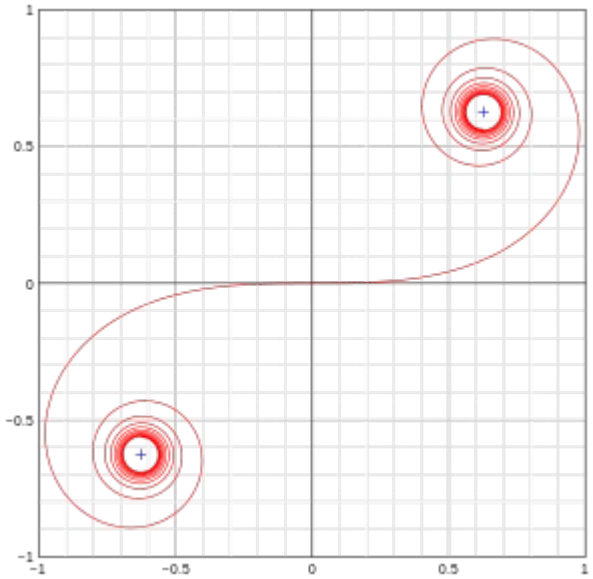


Different Types of Splines

- Cubic Splines
- Bezier Splines
- B-Splines
- NURBS
- ...

Clothoids

Clothoids



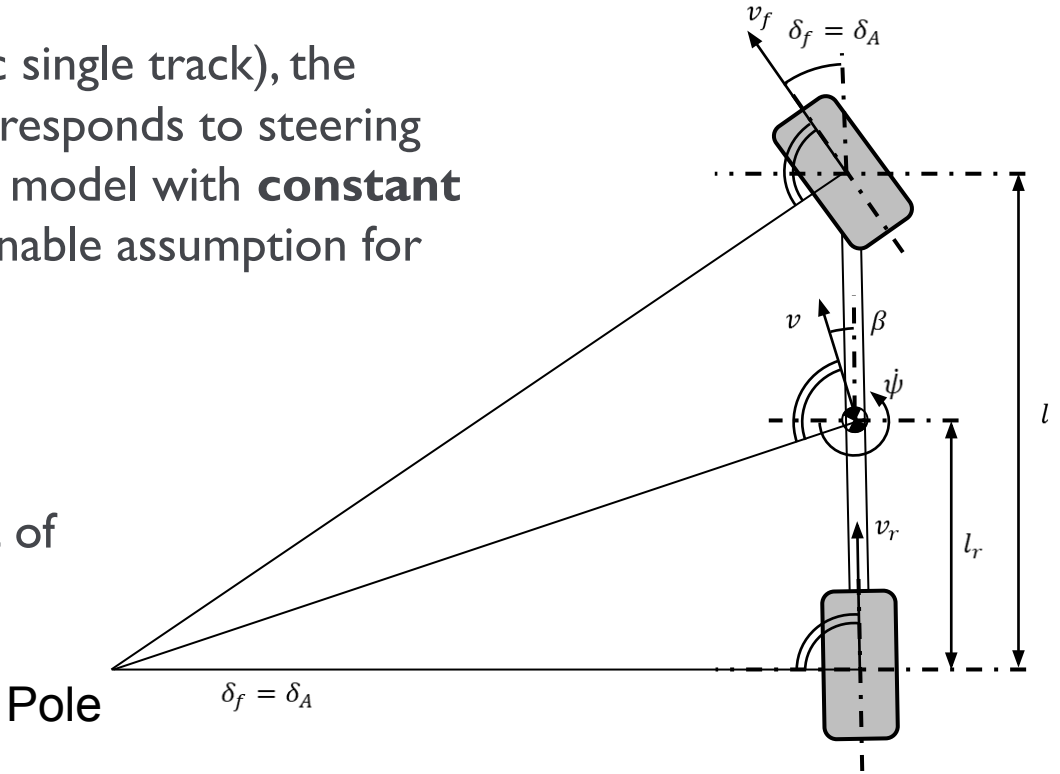
- Posture-continuous, (x, y, θ, κ)
- Distinct in that the **curvature varies linearly** with the length of the curve:
  - $\kappa(s) = a + bs$
  - κ is the curvature, and s is the arc length of the curve.
  - A generalized clothoid's curvature can also vary as a polynomial.
- Can also be parameterized as a polynomial.
  - This means we're solving for the parameters of the polynomial.

Revisiting Vehicle Dynamics

With the bicycle model (kinematic single track), the linear curvature characteristic corresponds to steering the front wheel of our ackermann model with **constant steering velocity**. This is a reasonable assumption for our vehicle.

$$\kappa = \frac{\tan\delta}{l} \approx \frac{\delta}{l}$$

We also add the curvature as part of our state vector: $[\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\kappa}]$



Vehicle Dynamics and Clothoids

$$x = [x, y, \theta, \kappa] \longrightarrow \text{Our state vector}$$

$$u = [v, \dot{\delta}] \longrightarrow \text{Our input vector}$$

$$\begin{aligned} \dot{x}(t) &= v(t)\cos\theta(t) \\ \dot{y}(t) &= v(t)\sin\theta(t) \\ \dot{\theta}(t) &= \kappa(t)v(t) \\ \dot{\kappa}(t) &= \dot{\delta}(t)/L \end{aligned} \longrightarrow \text{Our dynamics}$$

Vehicle Dynamics and Clothoids

First of all, we ignore velocity constraints in this discussion since we're using kinematic dynamics.

Next, since we're solving for a parameters of a polynomial, let's look at what constraints we have.

$$x(s_0) = [x_0, y_0, \theta_0, \kappa_0]$$
$$x(s_f) = [x_f, y_f, \theta_f, \kappa_f]$$

→ Initial position and heading is trivial. Thus this leaves us with 5 constraints: initial curvature (because we can steer the wheel without moving), and the end state.

$$\kappa(s) = a + bs$$

→ The original formulation for the curvature doesn't have enough parameters.

$$\kappa(s) = a + bs + cs^2 + ds^3$$

→ We instead use a cubic (or higher, usually quintic) polynomial to formulate the curvature.

$$q = [a, b, c, d, s_f]$$

→ Parameter vector of polynomial

Vehicle Dynamics and Clothoids

Since we included the curvature to be part of the input we can omit the last dynamics equation. Then the rest of our equations become homogeneous to the first degree in linear velocity v. We can then divide the rest of the equations by v in the form of ds/dt, and have a change of variable:

$$\frac{d}{ds}x(s) = \cos\theta(s)$$

$$\frac{d}{ds}y(s) = \sin\theta(s)$$

$$\frac{d}{ds}\theta(s) = \kappa(s)$$

Our states are now all functions of s and the parameter vector.

$$\kappa(s) = a + bs + cs^2 + ds^3$$

$$\theta(s) = as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4}$$

$$x(s) = \int_0^s \cos\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4}\right] ds$$

$$y(s) = \int_0^s \sin\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4}\right] ds$$

These integrals are not as nice though…

Easy to fix with Forward Euler Discretization with ds.

Optimizing for a solution

Now that we have our parameterization for the polynomial that follows a kinematic dynamics, how do we solve for the parameter vector for the polynomial?

We derive an optimization problem that follow the constraints.

Objective function

$$\text{minimize} \quad |x(s_f) - x_f|^2 + |y(s_f) - y_f|^2 + |\theta(s_f) - \theta_f|^2$$

Constraints

$$\kappa(s_f) = a + bs_f + cs_f^2 + ds_f^3$$

$$\theta(s_f) = as_f + \frac{bs_f^2}{2} + \frac{cs_f^3}{3} + \frac{ds_f^4}{4}$$

$$x(s_f) = \int_0^{s_f} \cos\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4}\right] ds$$

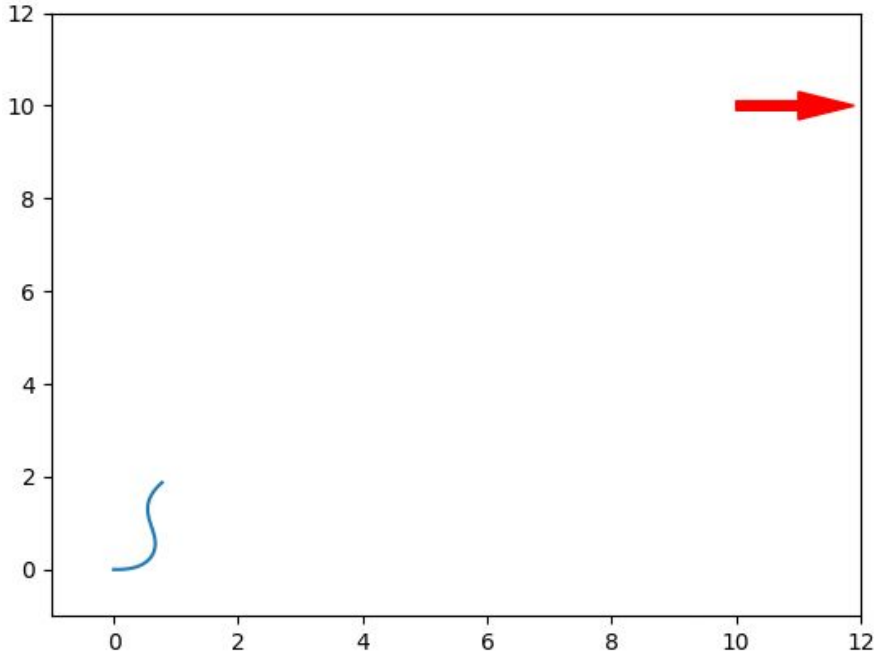
$$y(s_f) = \int_0^{s_f} \sin\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4}\right] ds$$

Optimization variable

$$q = [a, b, c, d, s_f]$$

This could be optimized with Newton's method because we can derive the Jacobian (omitted here, too long). Could also use something like `scipy.optimize.minimize`.

Optimization Progress



The animation on the left shows an example progress of optimizing for such a spiral. The goal pose is `[10, 10, 0, 0]`.

Is a spline enough?

- What else is needed to define a racing strategy for a given race car for a given track?
- Is the shape of the path alone enough to define a good racing strategy?

Spline with a velocity profile

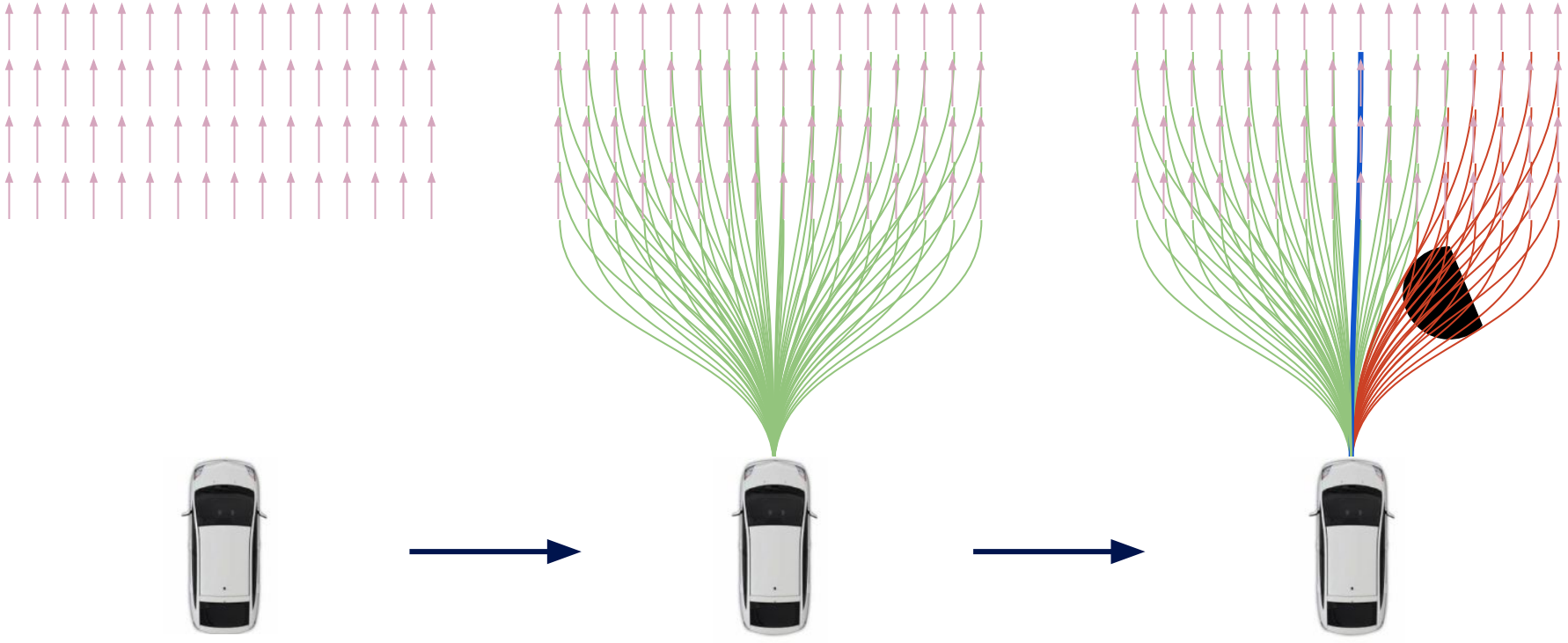
- The top speed of the car at certain points on the path is constraint by the curvature of the spline at those points.
- We'll denote a path defined by a spline, along with the velocity profile on the path as **trajectories**.

Penn Engineering

Notes on using clothoids

- To decrease the amount of computations, we always plan in the local frame of the vehicle. i.e. the initial states has all 0s.
- Although you'll end up having a trajectory of full states after integration, we still track it with something like pure pursuit since it's in a feedback loop and corrects for tracking errors.
- Many methods for creating a velocity profile. Could either create velocity profile for a clothoid based on the curvature, or just track the closest velocity you've found on a global reference.

As a Sampling-based Planner

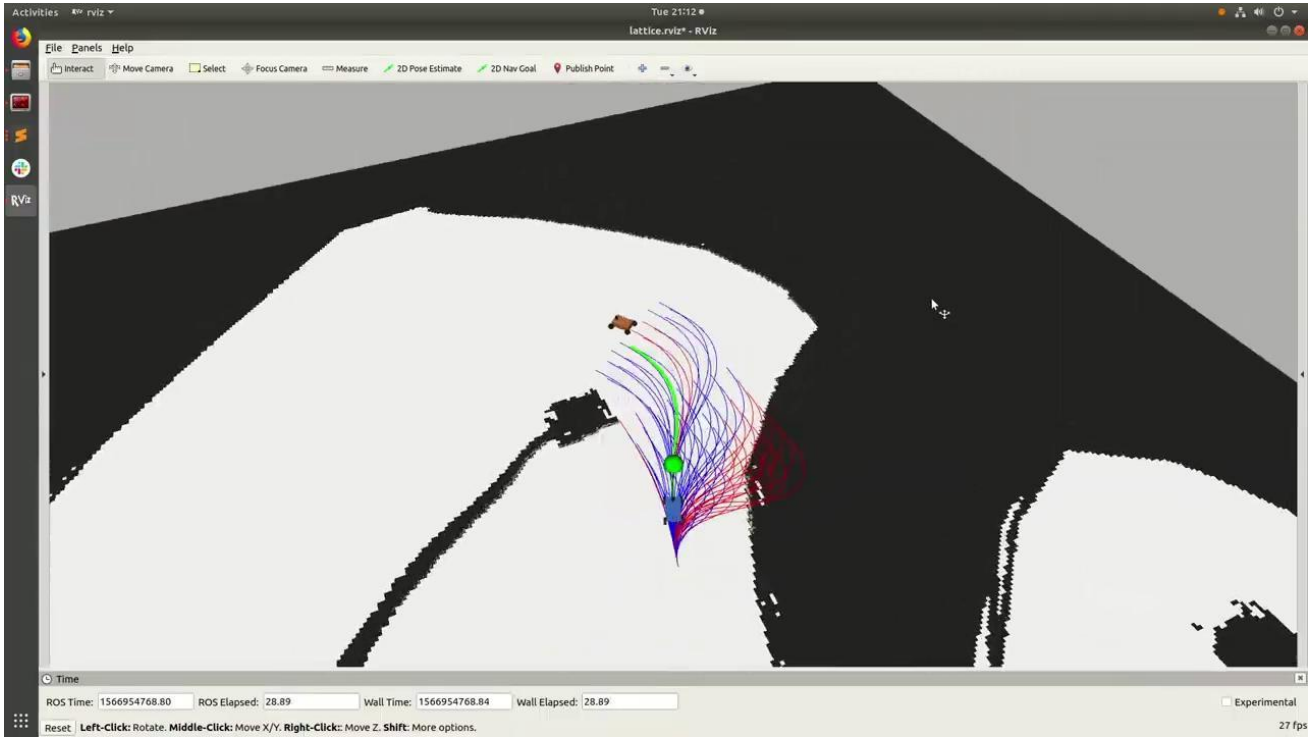


Lookup Table

	Lower bound	Upper bound	resolution
x	0.5m	10m	0.1m
y	-10m	10m	0.1m
θ	-pi/2 rad	pi/2 rad	0.01 rad
κ	-1.0	1.0	0.01

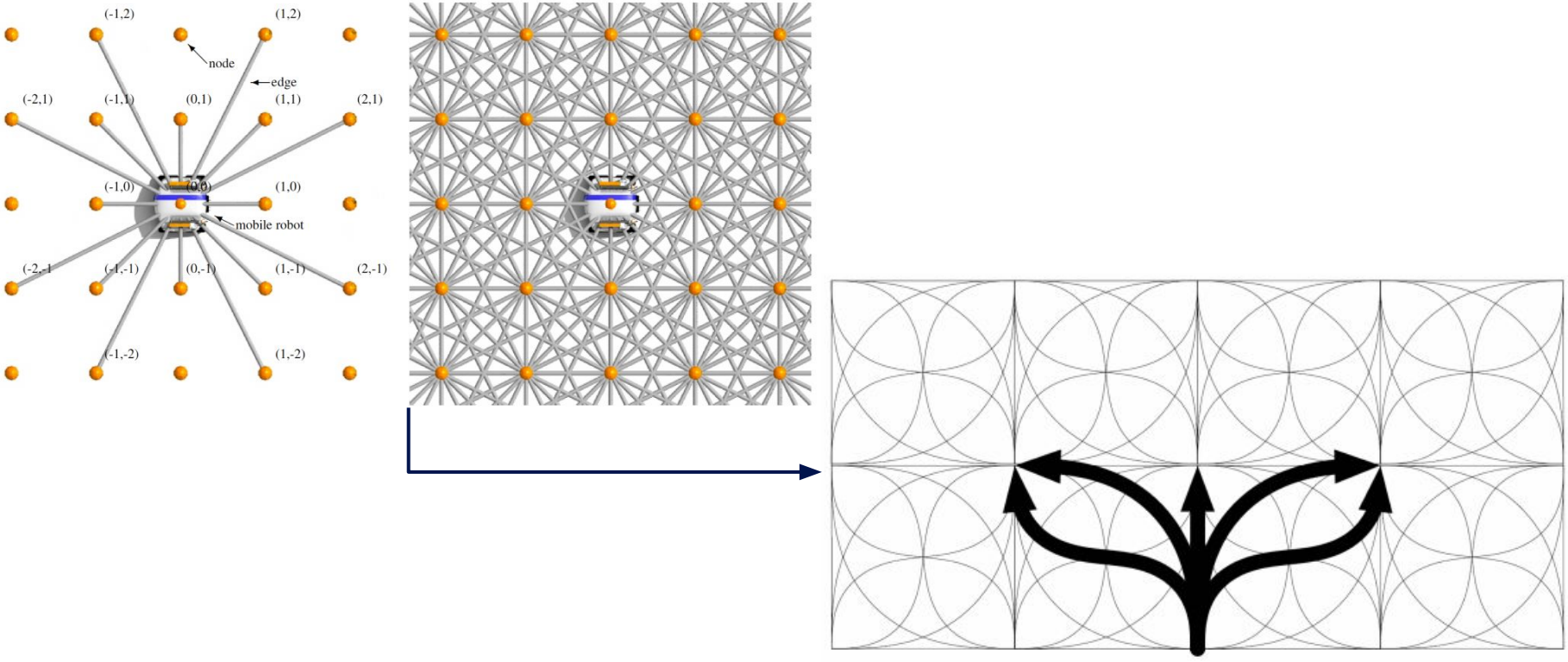
- Computation time increases as the number of primitives increase.
- Can't optimize online efficiently.
- Since our clothoids can be recovered from 5 numbers, we can create a lookup table.
- The lookup dimensions are the position, heading, and curvature of the final state, and the data stored are the 5 parameters for the corresponding clothoid solution.
- Lookup is easier to speed up than running an optimization problem.

Planner in action



Penn Engineering

State Lattice

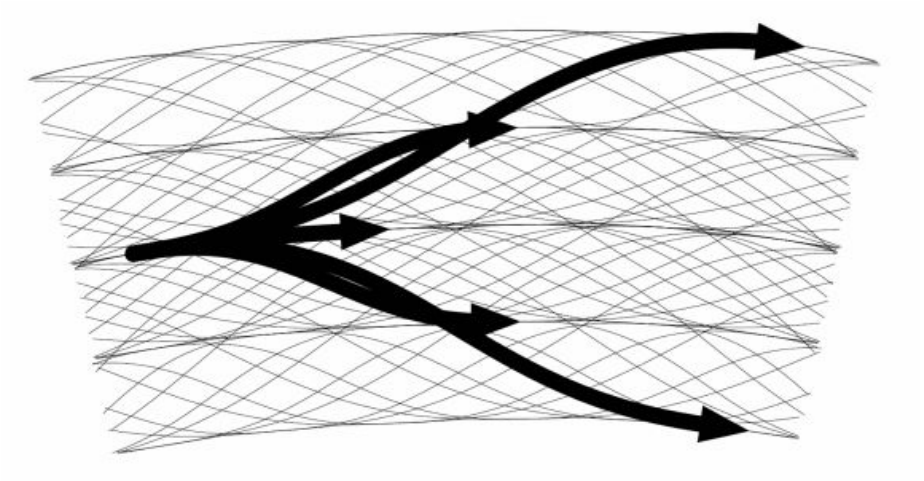


Penn Engineering

Image: Thomas Howard

Conformal State Lattice

- Ackermann, non-holonomic robots.
- Environment can be more structured.
- Higher density for nodes.



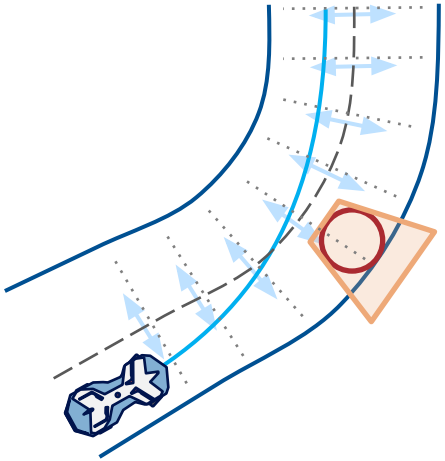
State Lattice - Literature

- Original paper using clothoids for planning: Shin and Singh, 1990, Path Generation for Robot Vehicles Using Composite Clothoid Segments
  - https://www.ri.cmu.edu/pub_files/pub4/shin_dong_hun_1990_1/shin_dong_hun_1990_1.pdf
- Papers from Kelly, Howard, and McNaughton:
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.637.356&rep=rep1&type=pdf
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.8908&rep=rep1&type=pdf
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.375.9234&rep=rep1&type=pdf
  - https://www.ri.cmu.edu/pub_files/pub4/howard_thomas_2008_1/howard_thomas_2008_1.pdf
  - https://www.ri.cmu.edu/pub_files/2011/7/mcnaughton-thesis.pdf
  - https://courses.cs.washington.edu/courses/cse571/16au/slides/Ferguson_et_al-2008-Journal_of_Field_Robotics.pdf

Advanced Local Path Planners - Overview

Optimization

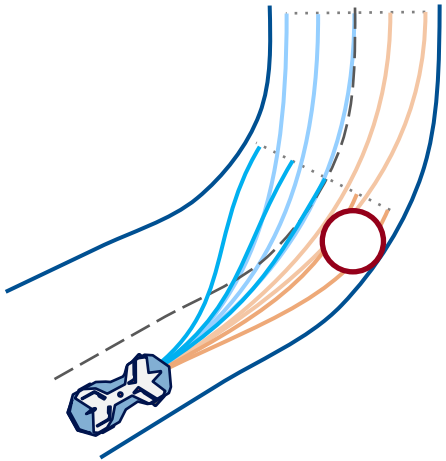
[Ziegler2014, Williams2016, …]



- Requires convex problem description
- Computationally expensive
- Smooth path with no discretization issues

1-Layer Graph-Search

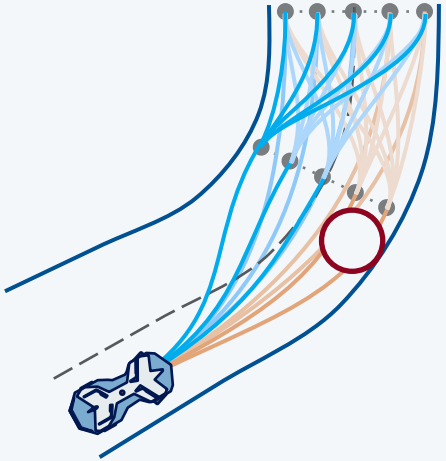
[Werling2010, Hu2018, …]



- No complex maneuvers (suitable for motorway only)
- Robust and fast

N-Layer Graph-Search

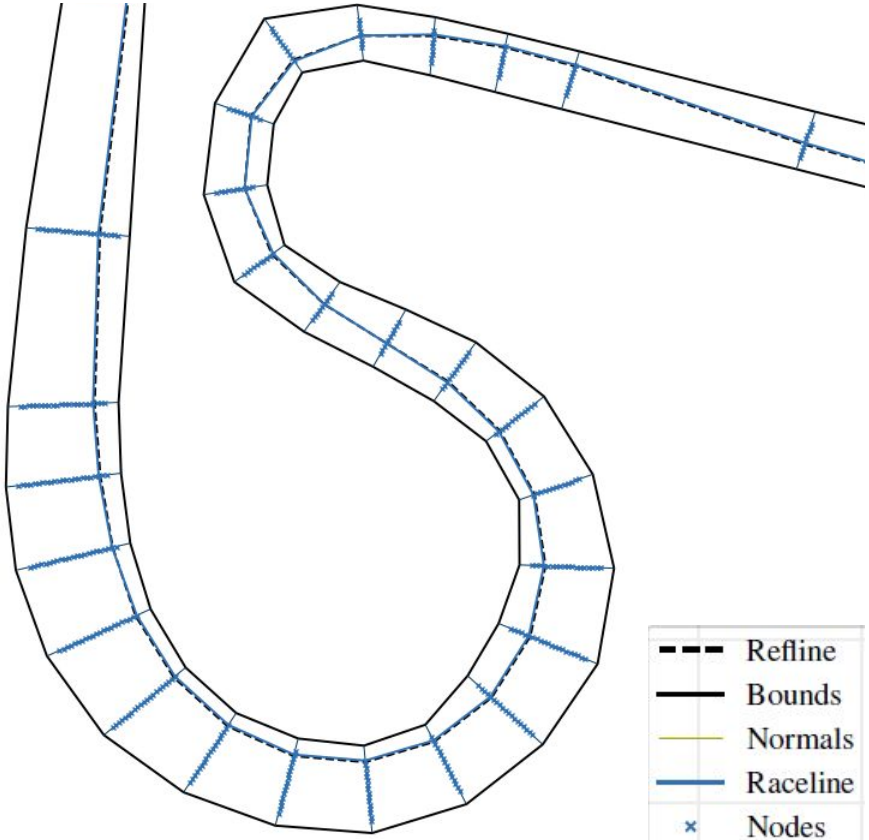
[McNaughton2011, Gu2013, …]



- Quality conflict: calculation time vs. discretization
- Complex maneuvers possible
- Possible to pre-compute parts

Penn Engineering

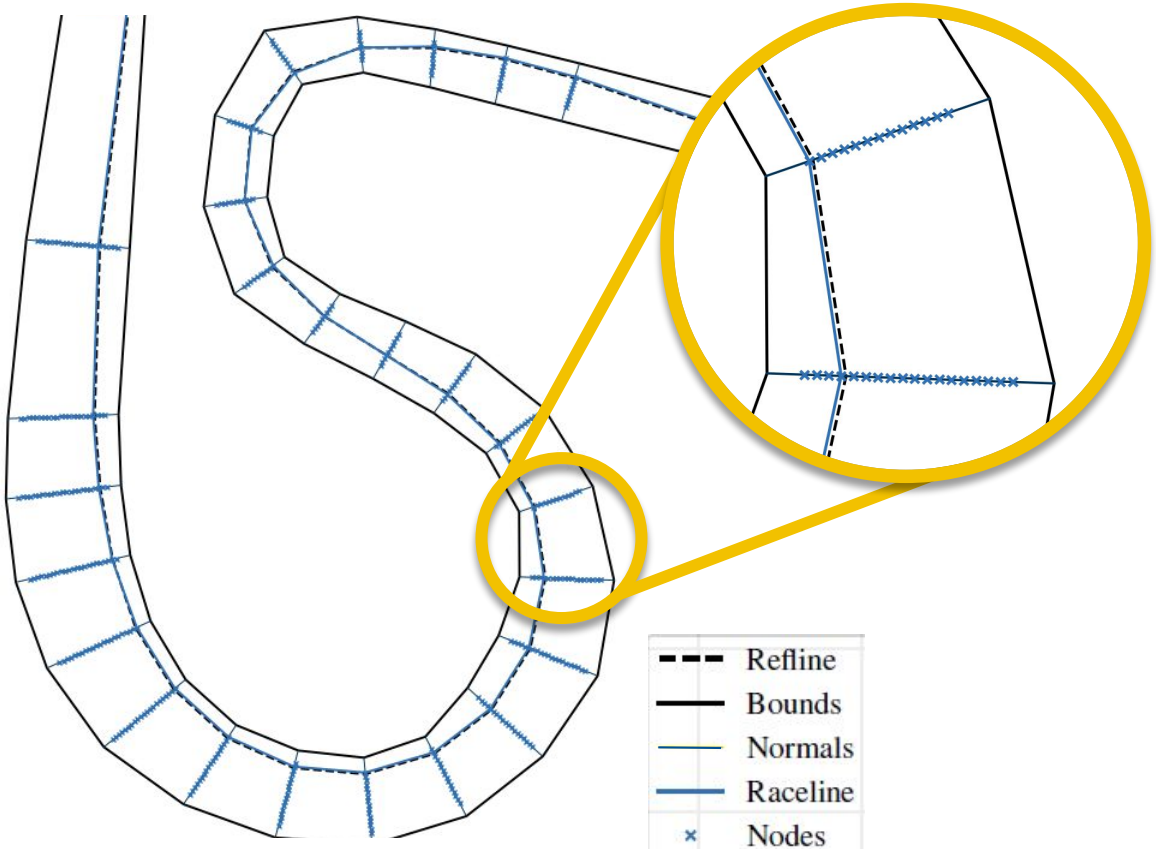
Graph Based Planner - Offline

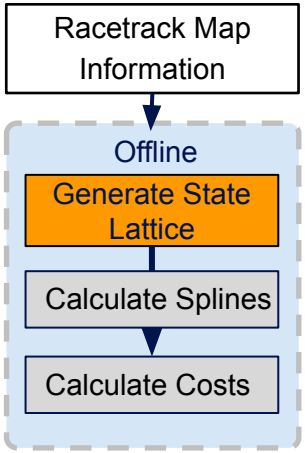


Racetrack Map Information

T. Stahl, et al. “Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios“ in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

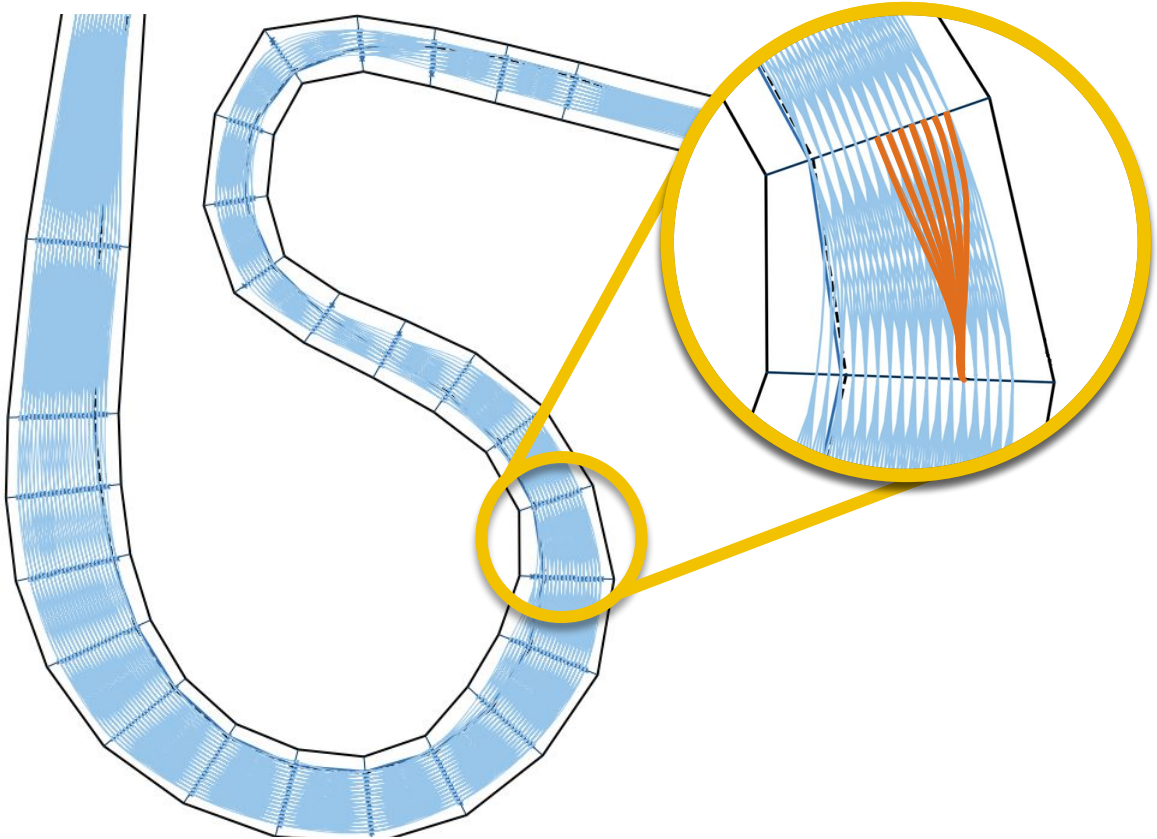
Graph Based Planner - Offline

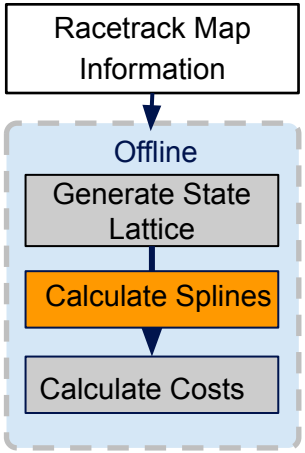




T. Stahl, et al. "Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios" in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

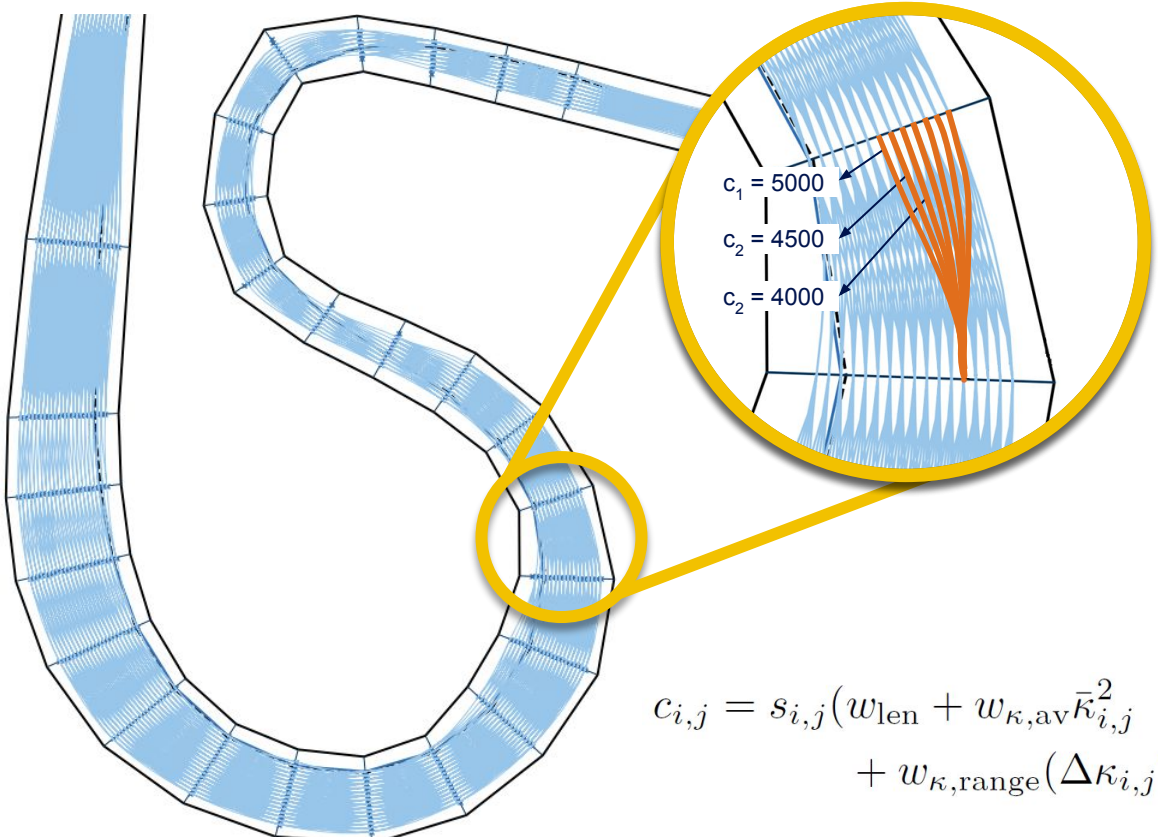
Graph Based Planner - Offline

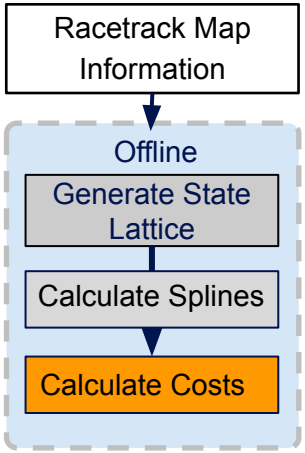




T. Stahl, et al. “Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios“ in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

Graph Based Planner - Offline

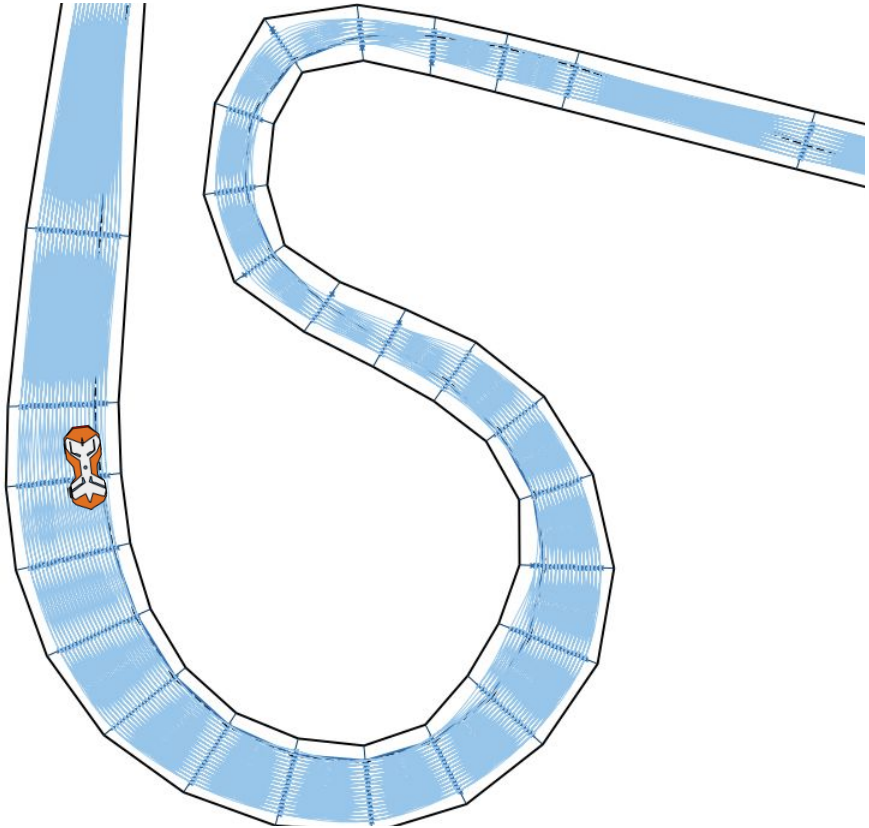


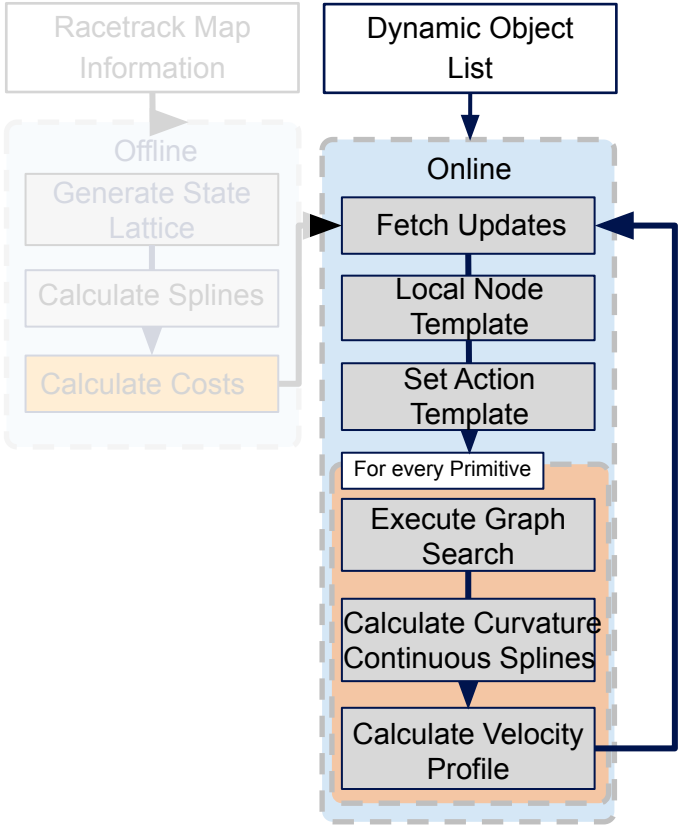


$$c_{i,j} = s_{i,j}(w_{\text{len}} + w_{\kappa,\text{av}}\bar{\kappa}_{i,j}^2 + w_{\kappa,\text{range}}(\Delta\kappa_{i,j})^2 + w_{\text{rl}}d_{\text{lat},j})$$

T. Stahl, et al. “Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios“ in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

Graph Based Planner - Online



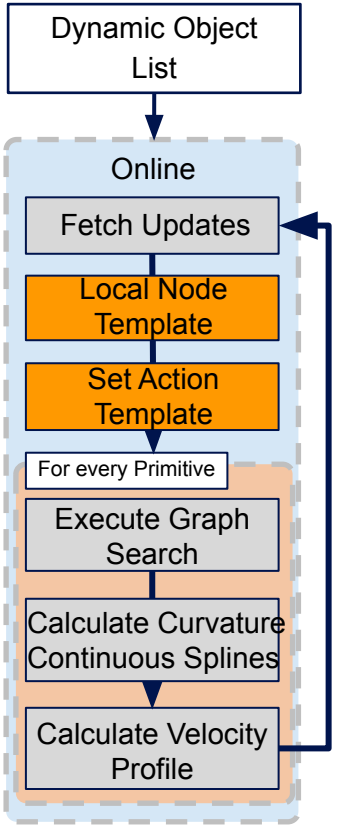


T. Stahl, et al. “Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios“ in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

Graph Based Planner - Online

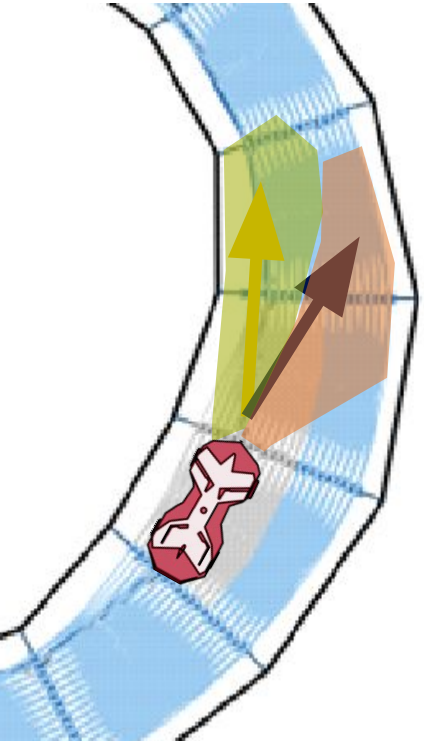
Setup Local Node Template by filtering the edges of the full graph based on:

- Planning horizon
- Action templates
  - Driving straight, overtake left / right
  - Selected afterwards in Behavioral Planner to dismiss invalid trajectories



T. Stahl, et al. “Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios“ in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

Graph Based Planner - Online

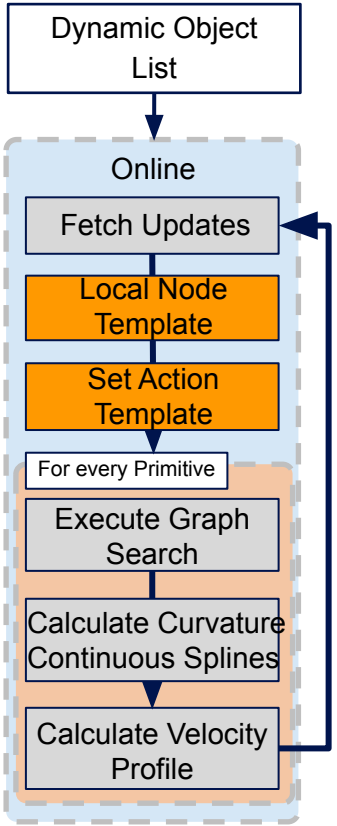


Opponent Prediction:

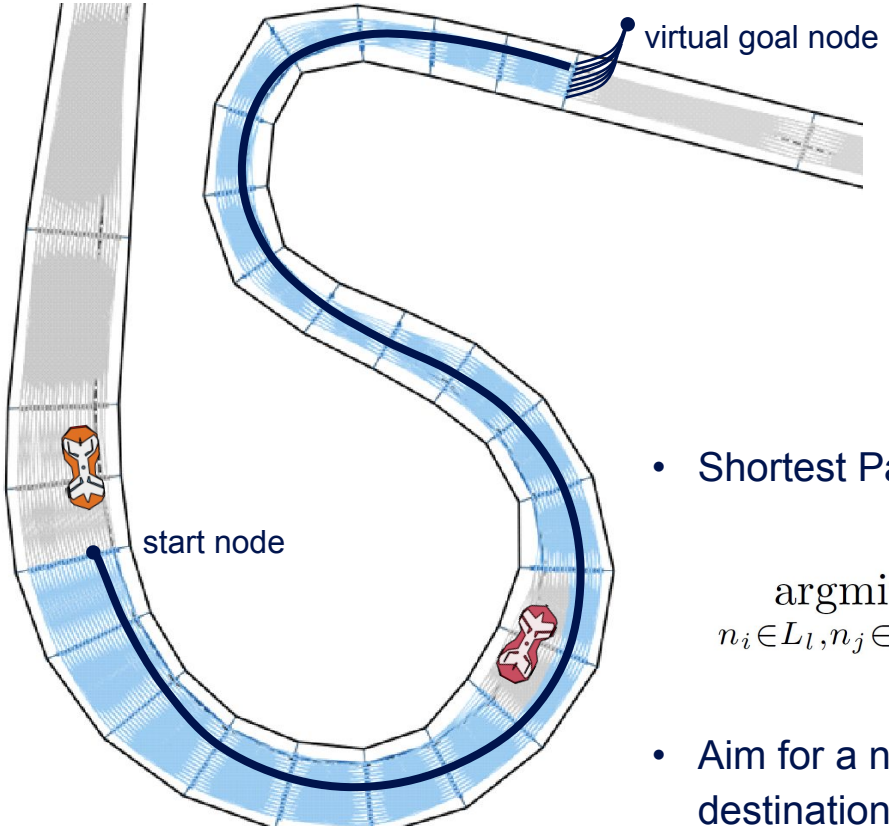
- Influences quality of local node template
- The better the prediction –> The better the path planning

Approach:

- Data based approach
- LSTM + CNN
- Online Learning and Adaptation



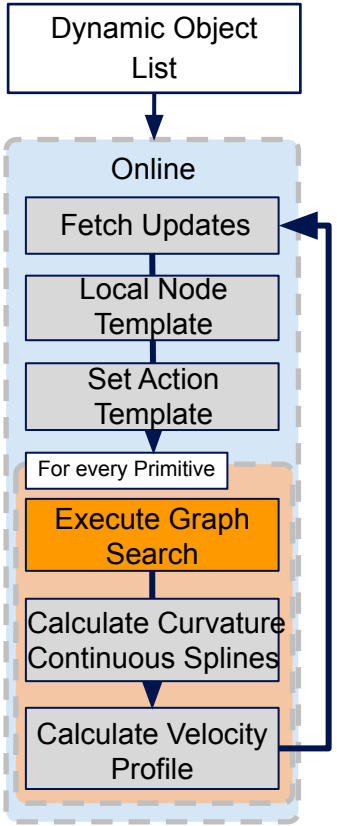
Graph Based Planner - Online



- Shortest Path Problem: Dijkstra

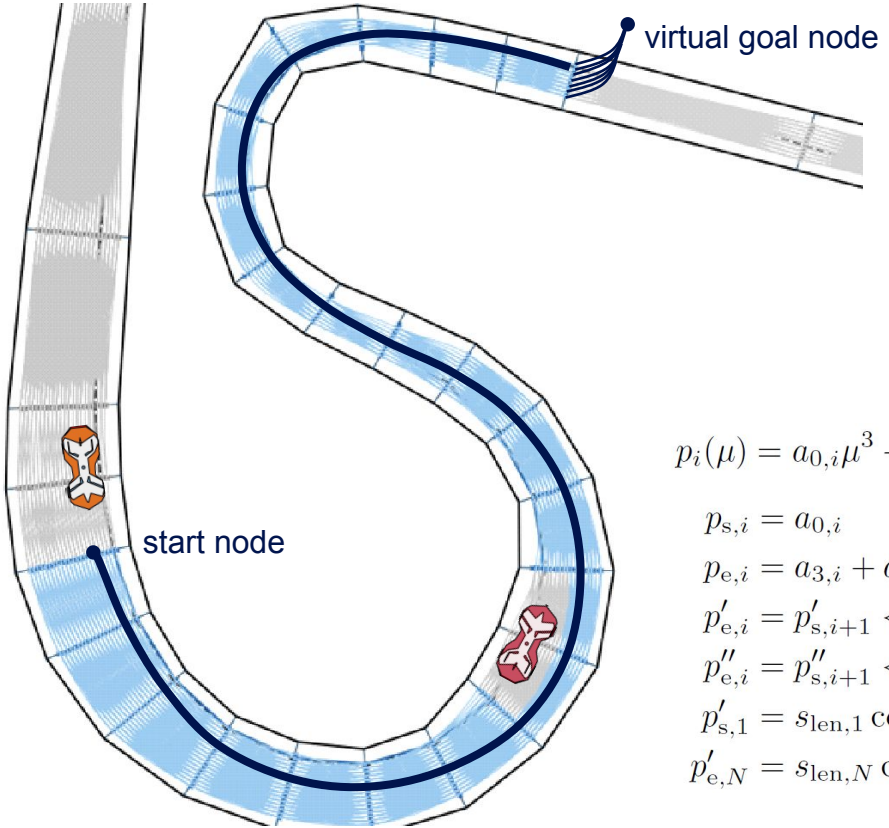
$$\underset{n_i \in L_l, n_j \in L_{l+1}}{\operatorname{argmin}} \sum_{l=l_s}^{l_e-1} c_{i,j}$$

- Aim for a node on the race line in the destination layer



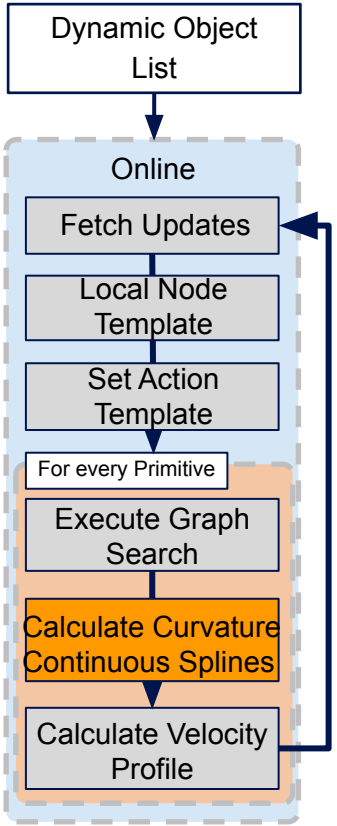
T. Stahl, et al. "Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios" in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

Graph Based Planner - Online



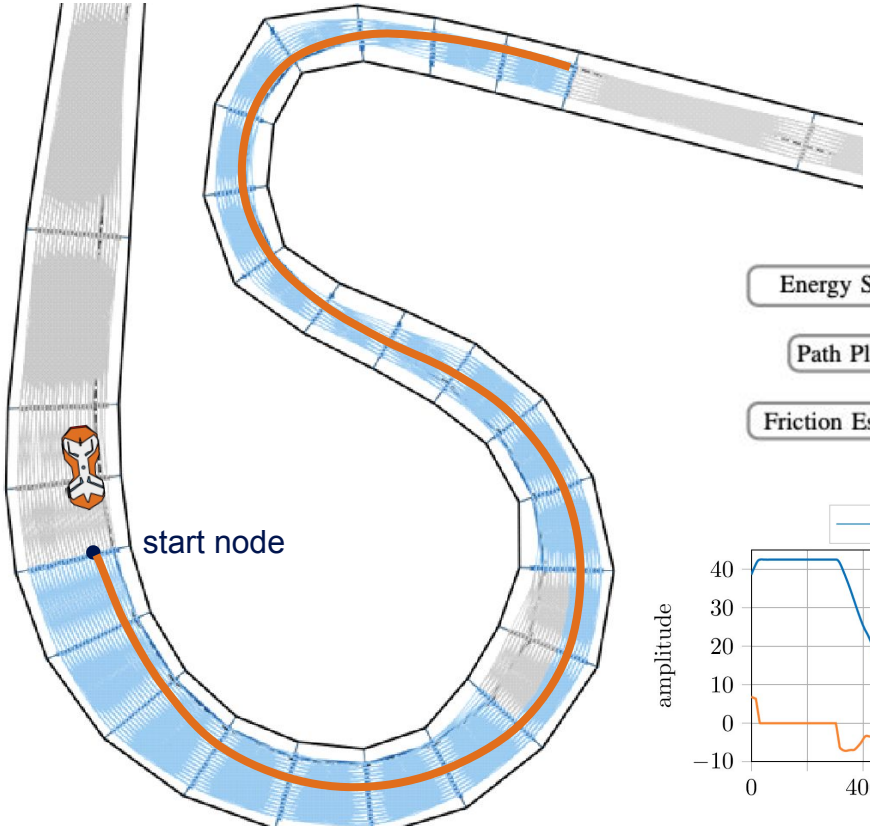
$$p_i(\mu) = a_{0,i}\mu^3 + a_{0,i}\mu^2 + a_{0,i}\mu + a_{0,i}$$

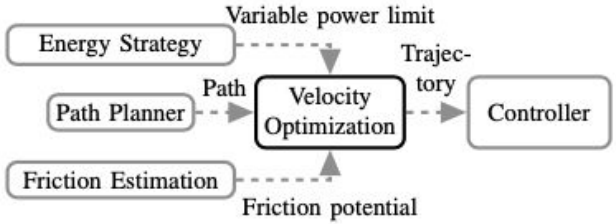
$$\begin{aligned}
p_{\mathrm{s},i} &= a_{0,i} \\
p_{\mathrm{e},i} &= a_{3,i} + a_{2,i} + a_{1,i} + a_{0,i} \\
p'_{\mathrm{e},i} &= p'_{\mathrm{s},i+1} \iff 3a_{3,i} + 2a_{2,i} + a_{1,i} = a_{1,i+1} \\
p''_{\mathrm{e},i} &= p''_{\mathrm{s},i+1} \iff 6a_{3,i} + 2a_{2,i} = a_{2,i+1} \\
p'_{\mathrm{s},1} &= s_{\mathrm{len},1}\cos(\theta_{\mathrm{s},1}) = a_{1,1} \\
p'_{\mathrm{e},N} &= s_{\mathrm{len},N}\cos(\theta_{\mathrm{e},N}) = 3a_{3,N} + 2a_{2,N} + a_{1,N}
\end{aligned}$$

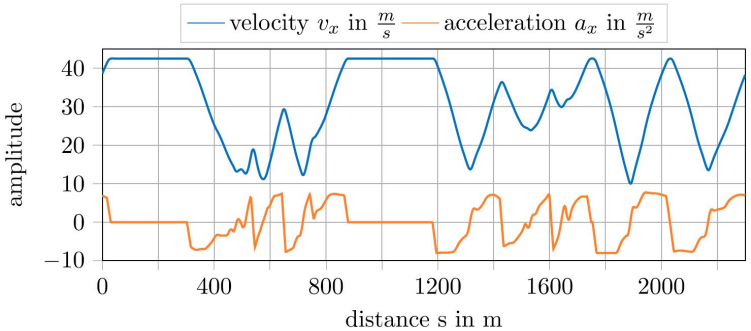


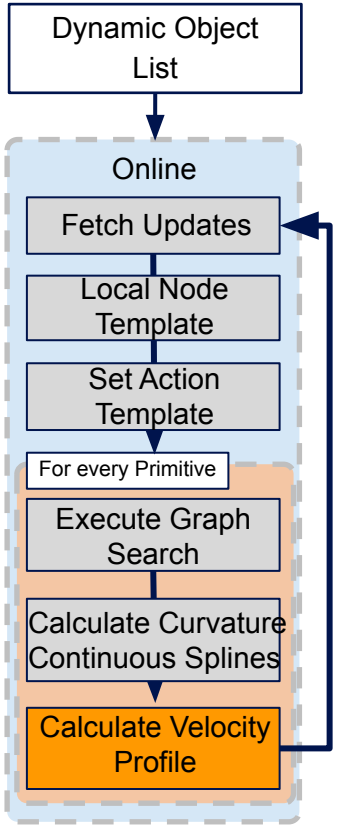
T. Stahl, et al. “Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios“ in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,

Graph Based Planner - Online









T. Stahl, et al. "Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios" in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019,