Global and local planners

Real-Time Embedded System - The F1tenth autonomous racing

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

High Performance Real Time Lab

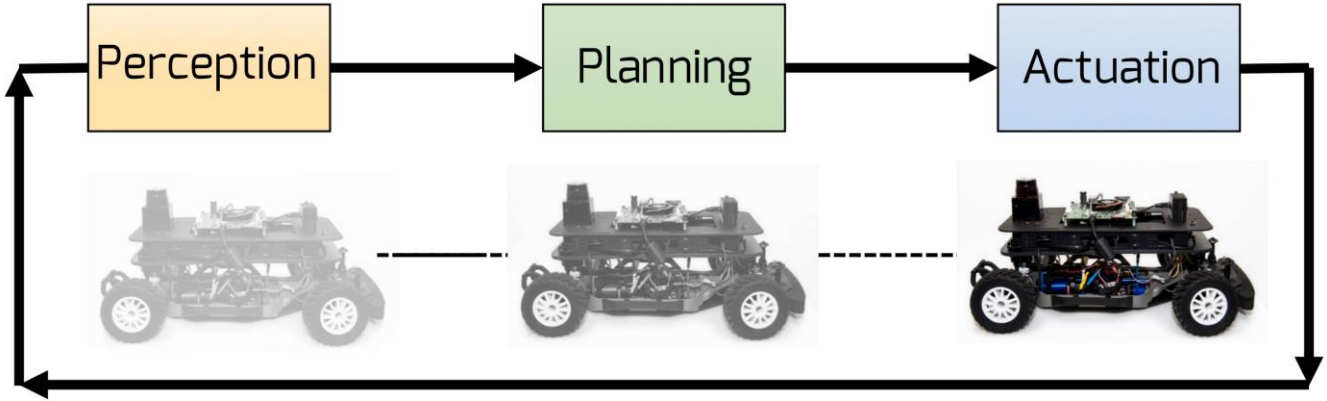
Course outline

- Intro course + basics of AD
- Hardware platform
- ROS2: Installation and profiling
  - Ex: ROS2 to HiL, open a bag
- Navigation: FTG, FTW, Pure pursuit
  - EX: navigation HiL
- Perception: scan matching, PF, LIO?
  - Ex: perception (PF with PThreads)
- Build the car

I do <u>not</u> cover all aspects of AD!!!

- Systems and control theory => Prof. Falcone
- Platforms and algorithms for autonomous systems => Prof. Sanudo & Prof. Falcone
- High-Performance Computing => Prof. Marongiu (FIM)
- Machine Learning => Cucchiara's

Our simplest stack



Urg → Localize (?) → Plan/Ctrl → Teensy

Global vs. local planners

Global planners (aka: static planners) work on the track map

- Pro: optimal trajectories and speed
- Pro: can pre-compute the optimal trajectory (don't care how long it takes, nor on which computer), and store in memory for simply retrieving it
- Con: need to build the map!
- Con: need to localize
- Con: cannot deal with dynamic obstacles

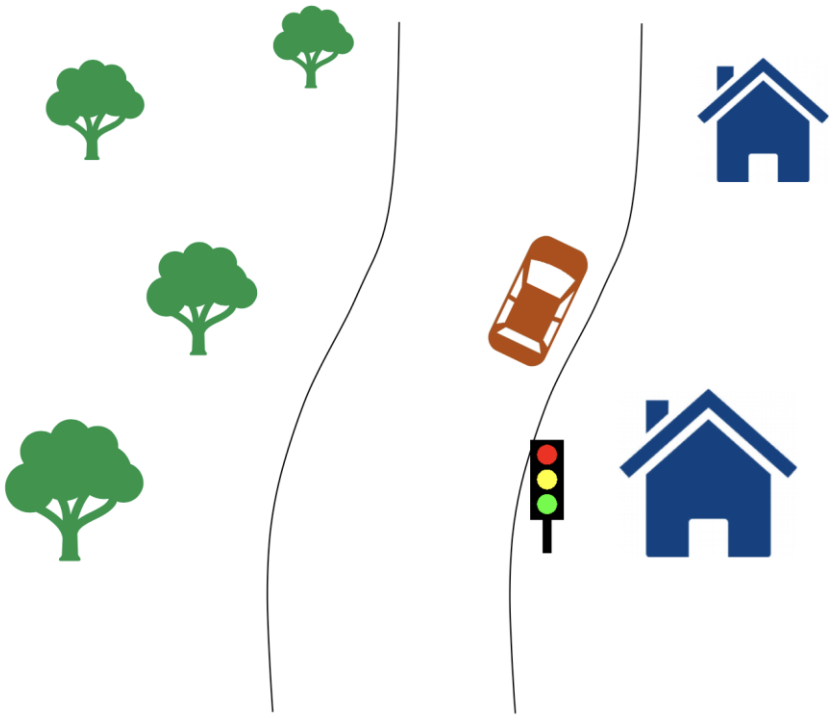
Local (aka: dynamic, reactive) planners

- Pro: no map is required, nor localization
- Pro: work with dynamic obstacles
- Con: must be fast!
- Con: ~~must~~ shall be on-board

Global approaches - Map representations

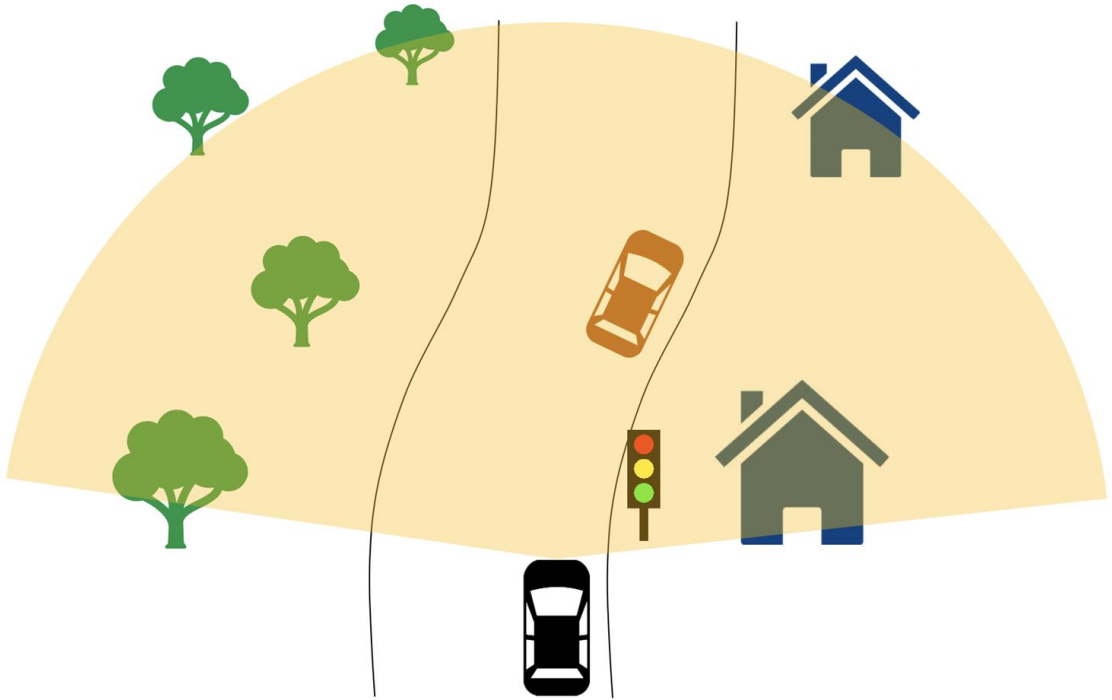
- The vehicle needs to know what kind of obstacle are around it
- The vehicle needs to know where these obstacle are
- The vehicle needs to represent its surroundings some how
- We can use different map representations:
  - Occupancy Grid Map ⟶ Focus for small-scale robots e.g. F1TENTH
  - Point Cloud Map ⟶ Focus for real-world passenger vehicles
  - Feature Map ⟶ Focus on localization
  - Semantic Maps ⟶Focus on object detection
  - HD Map ⟶ Focus for real-world passenger vehicles

Example map



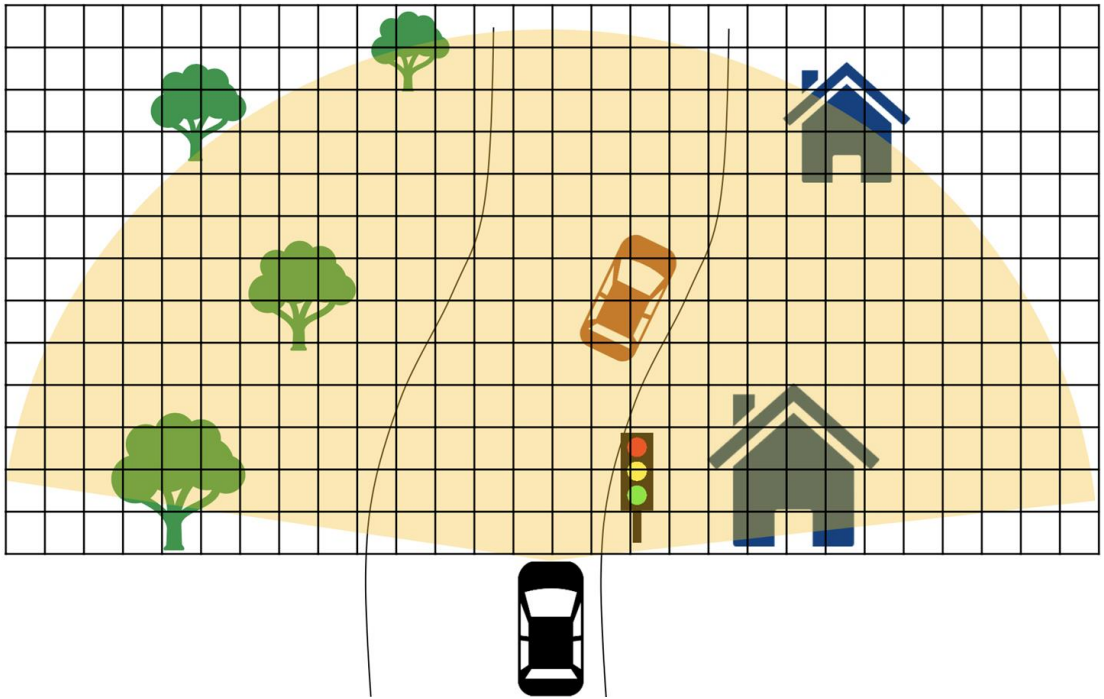
Example Environment with obstacles on one street and surroundings

Example map



The car perceives its environment with sensor: Lidar, Camera, Radar, GPSA

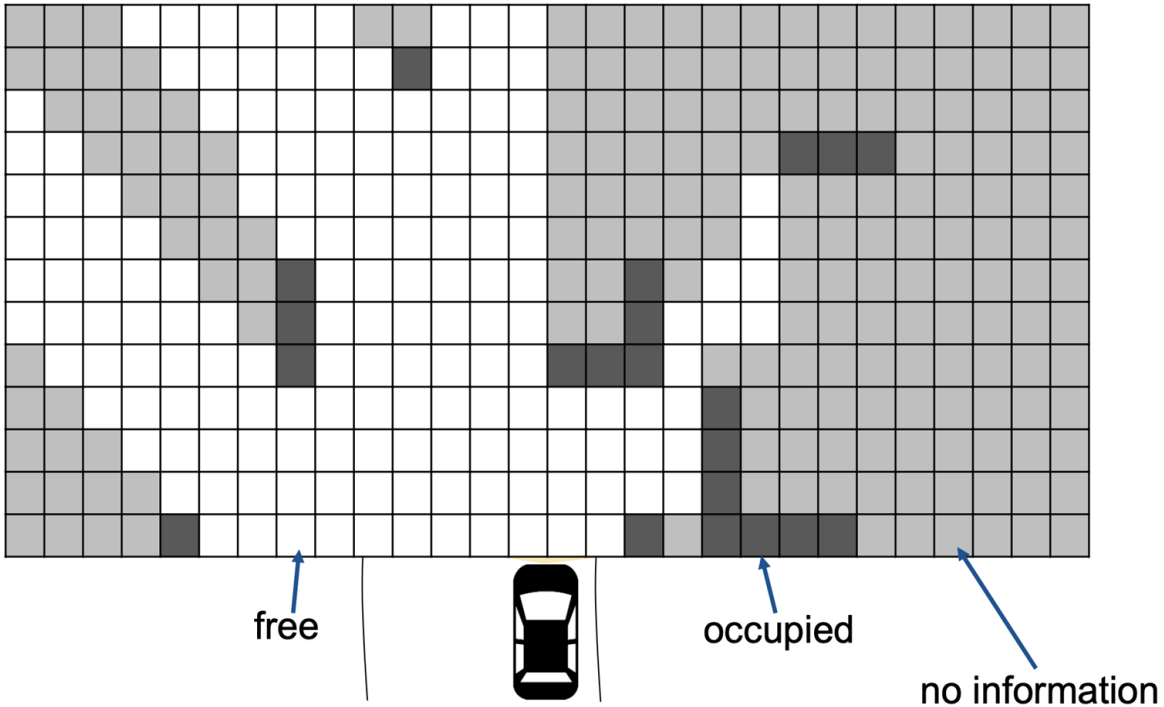
Occupancy Grid Map



- Discretization of the map
- Each cell can be occupied, free or no information
- Mainly used for 2D-mapping
- Mainly used with LiDAR-sensors

Create a 2D grid that is covering the whole sensor area (Its size matters!)

Occupancy Grid Map



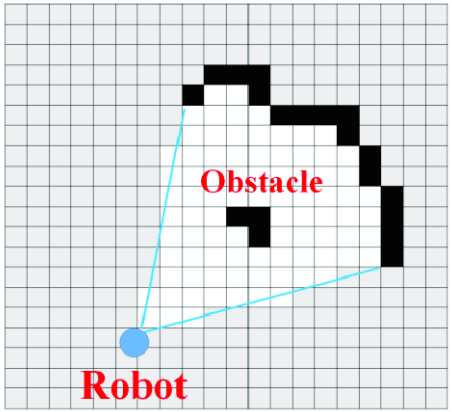
Final Occupancy Grid Map

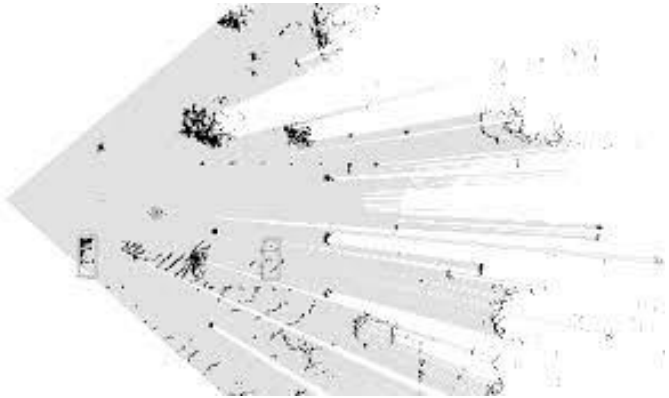
- Discretization of the map
- Each cell can be occupied, free or no information
- Mainly used for 2D-mapping
- Mainly used with LiDAR-sensors

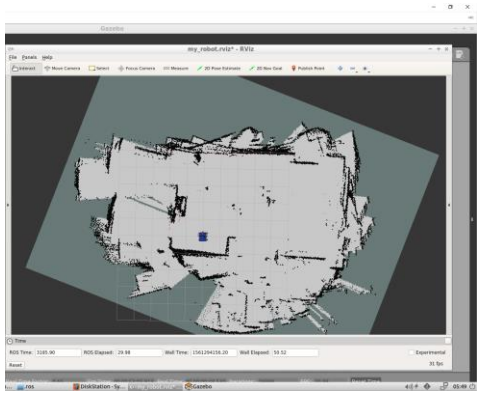
Occupancy Grid Map

- Fixed discretization as map resolution
- Overlay of laser measurements depending on ego-motion

Occupancy Grid Map

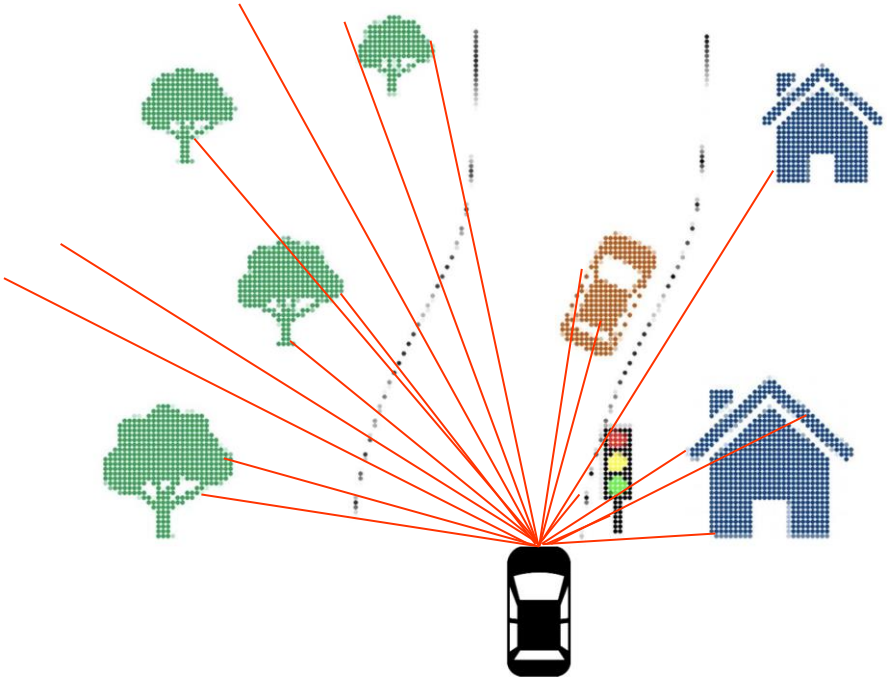






+ Easy readable	- Difficult to use in 3D
+ Directly usable for path planning	- Limited precision of localization
+ Relatively easy to create	- Fixed discretization

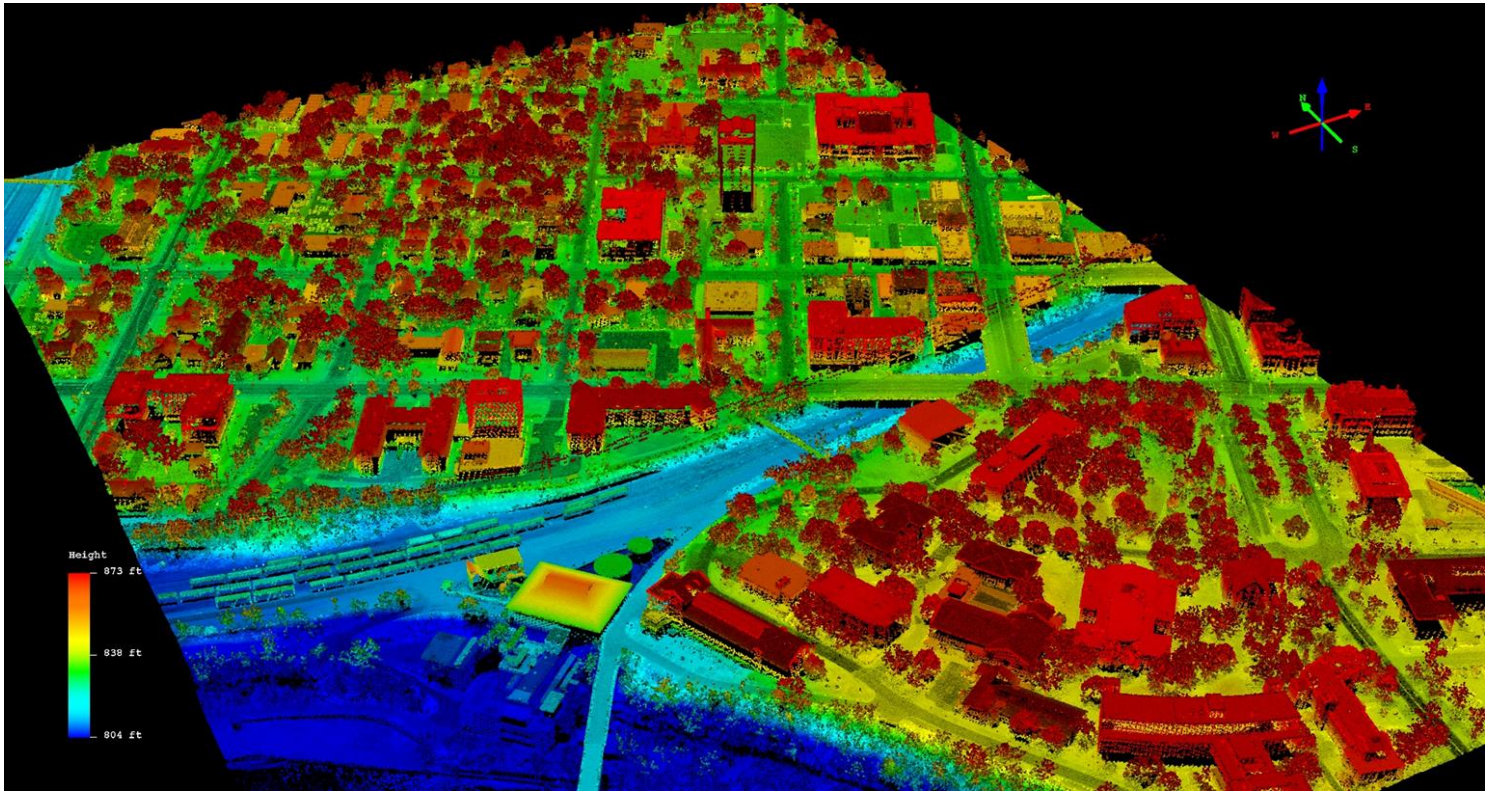
Point Cloud



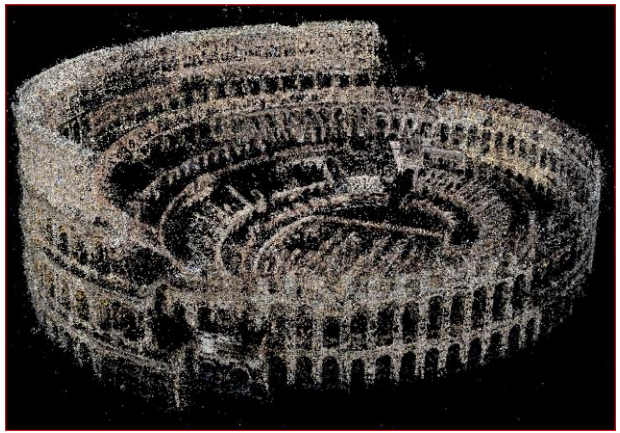
Create a 3D Point Cloud that is covering the whole sensor area

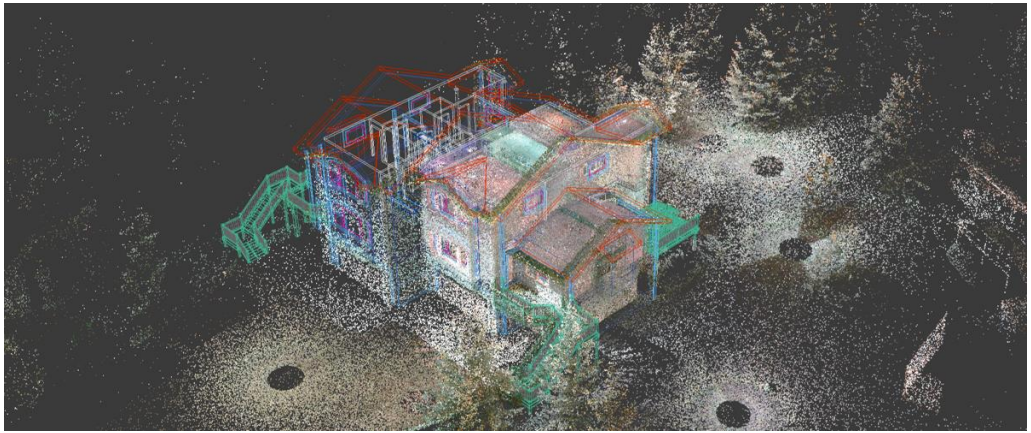
- Discretization of the map
- Lidar Sensor defines the area: Number of beams, number of layers
- Mainly used for 3D-mapping
- Mainly used with LiDAR-sensors

Point Cloud



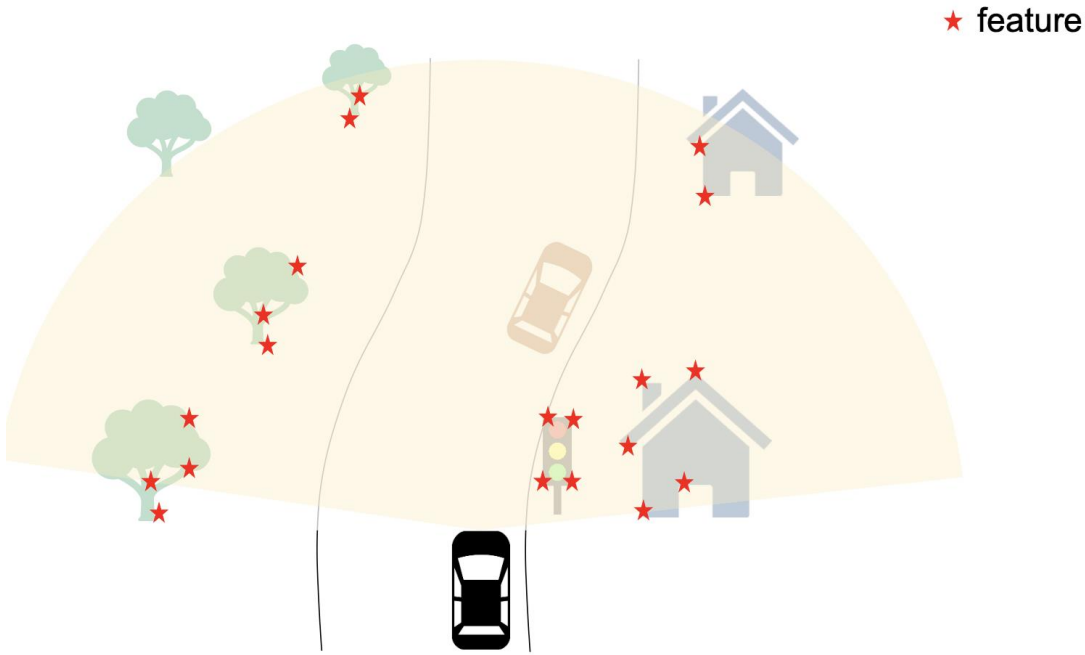
Point Cloud





+ Easy understandable	- Expensive 3D Lidar
+ Directly usable for path planning	- High amount of data
+ Relatively easy to create	- Inclusion of irrelevant information

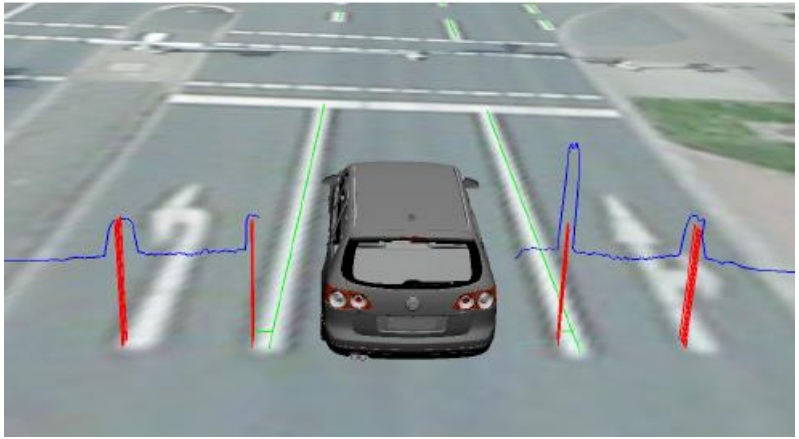
Feature Map

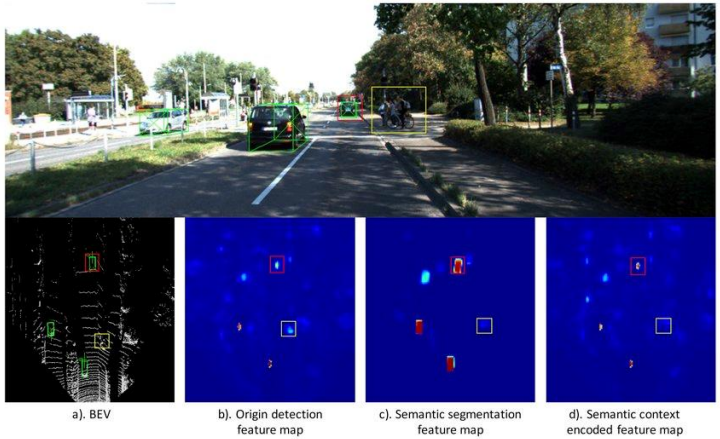


- Create a map based on camera (and LiDAR)
- Define **Features**: information about an object that is a prominent or distinctive part, quality, or characteristic
- Features **extracted** from environment as fix-points
- 3D representation possible

Define features for all objects that are around the vehicle

Feature Map

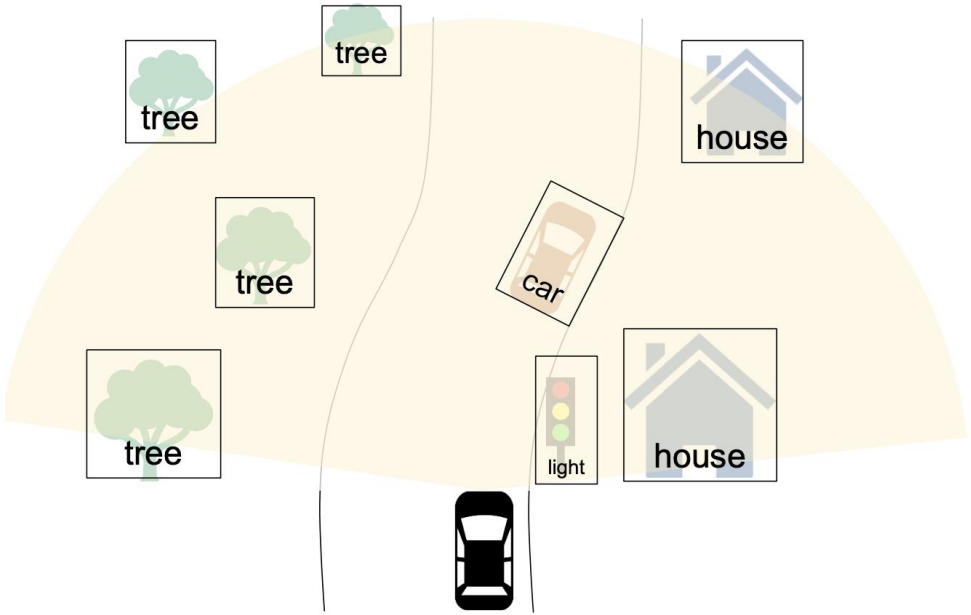




a). BEV | b). Origin detection feature map | c). Semantic segmentation feature map | d). Semantic context encoded feature map

+ Application in 2D and 3D	- No understandable for humans
+ No discretization needed	- No information on occupancy
+ Low memory consumption	

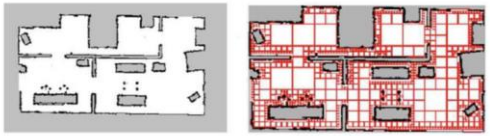
Semantic Map



- **Detection** and **position estimate** of objects
- Inclusion of semantic information
- Semantics = Meaning
- Usually combined with other map representation
- Can be done per pixel in the camera image

Define semantic meaning for all objects that are around the vehicle

Semantic Map

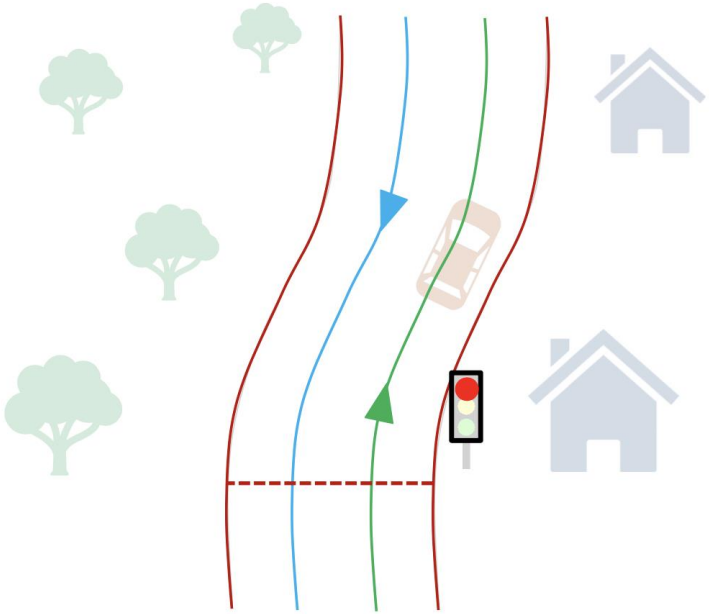


2D grid map

3D map with semantic information

+ Include semantic information	- Needs computational resources
+ Different objects and classes	- Dependent on object detection
+ Additional information for path planner	- Needs additional information on surroundings

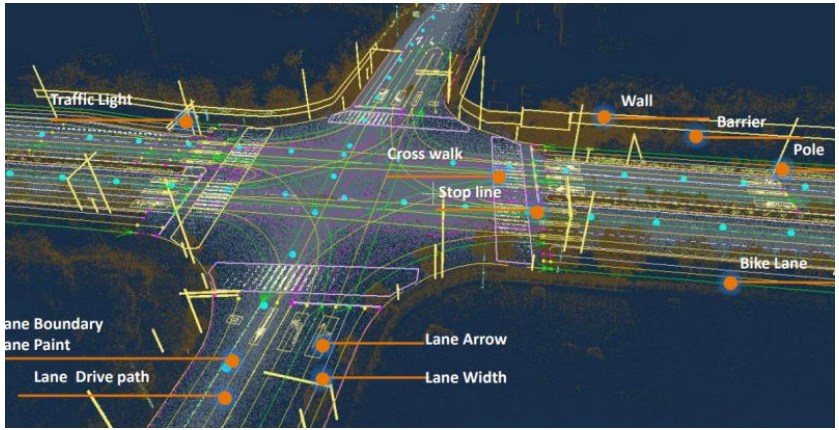
High Definition (HD) Map

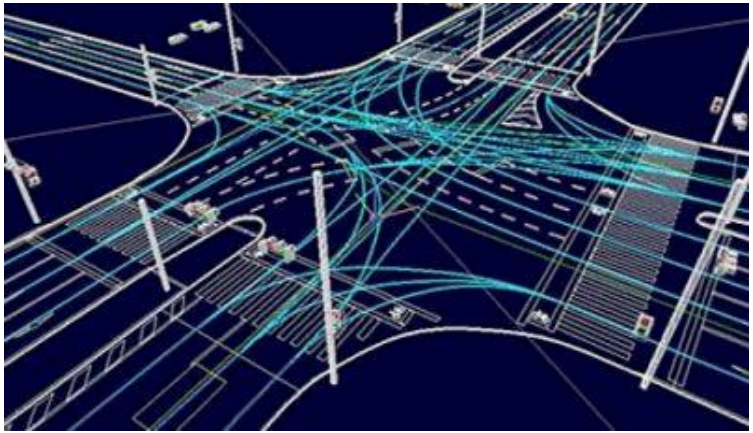


- Usage of all vehicle sensors: Lidar, Camera, Radar, GPS
- High-definition maps for self-driving cars usually include map elements such as road shape, road markings, traffic signs and barriers
- High precise lane information and semantics are available

Create a high information based version of the streets and objects that are around the vehicle

High Definition (HD) Map





+ High Information content	- Expensive to create
+ High Precision	- Needs to be updated
+ Fusion of many sensors	- High memory and bandwidth demand

Contol on a map - Pure pursuit

Given a map, and vehicle position…

› We'll soon see this

…and also the (optimal?) raceline..

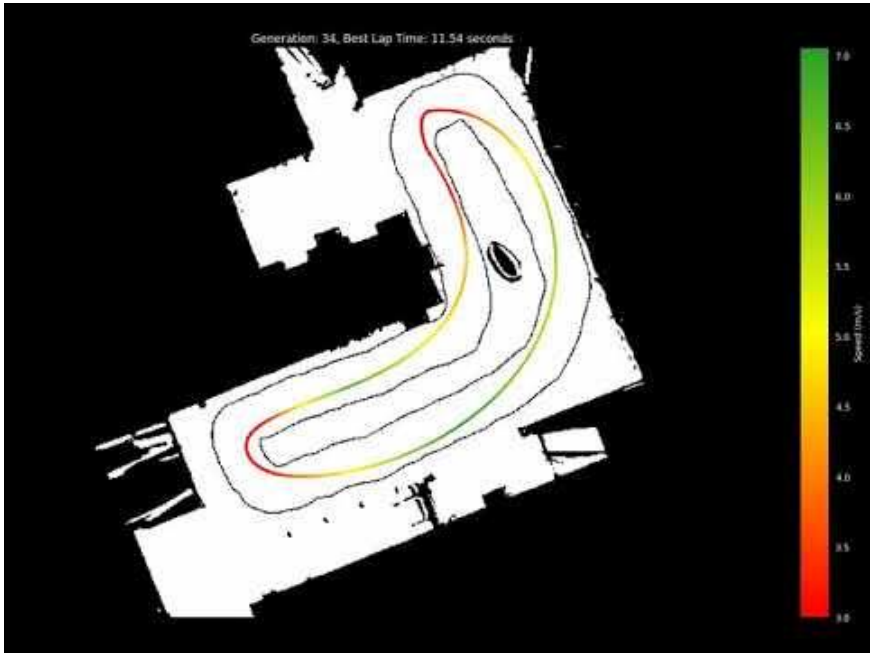
› Won't see this, in this course

..race, race, race!!

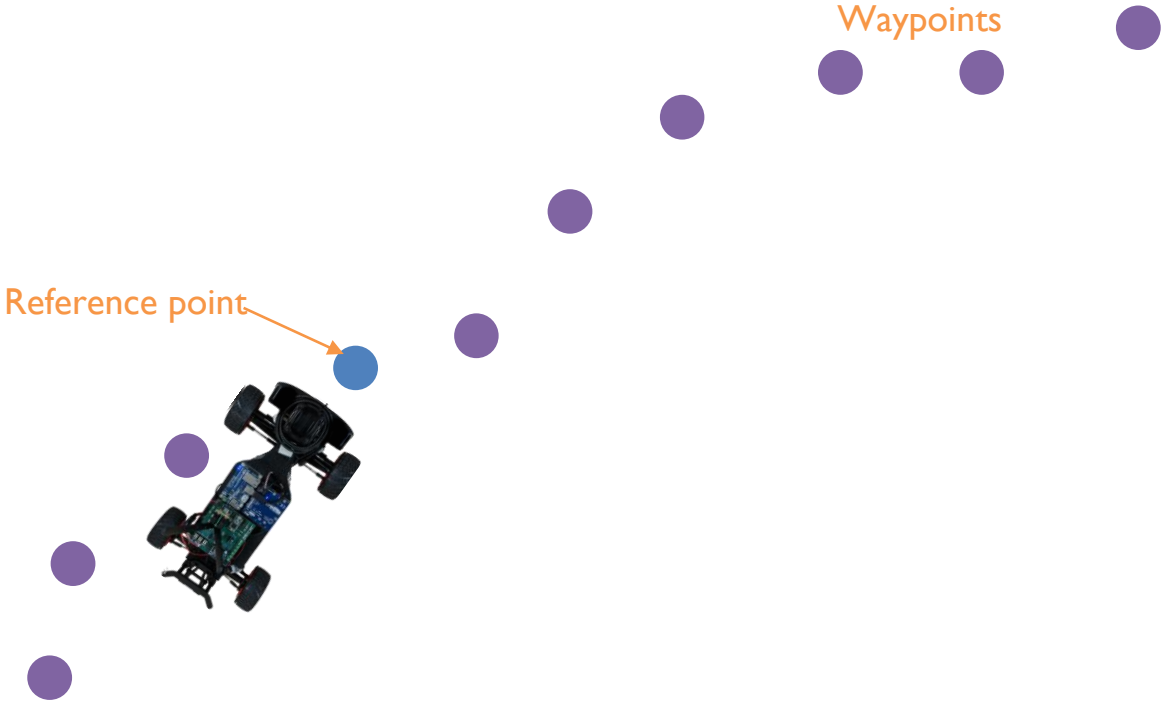
› How do we track a given trajectory?

› How do we correct for actuation errors?

› How do we drive as fast as possible?



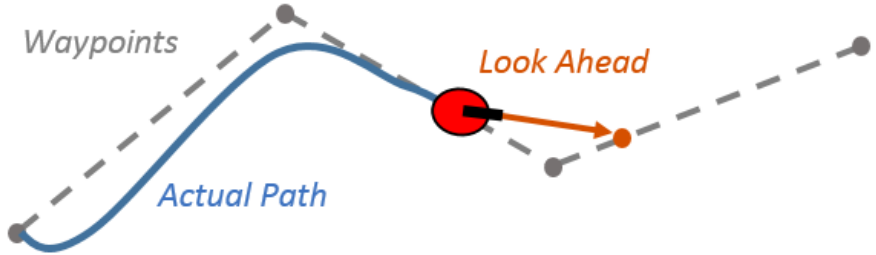
Pure Pursuit



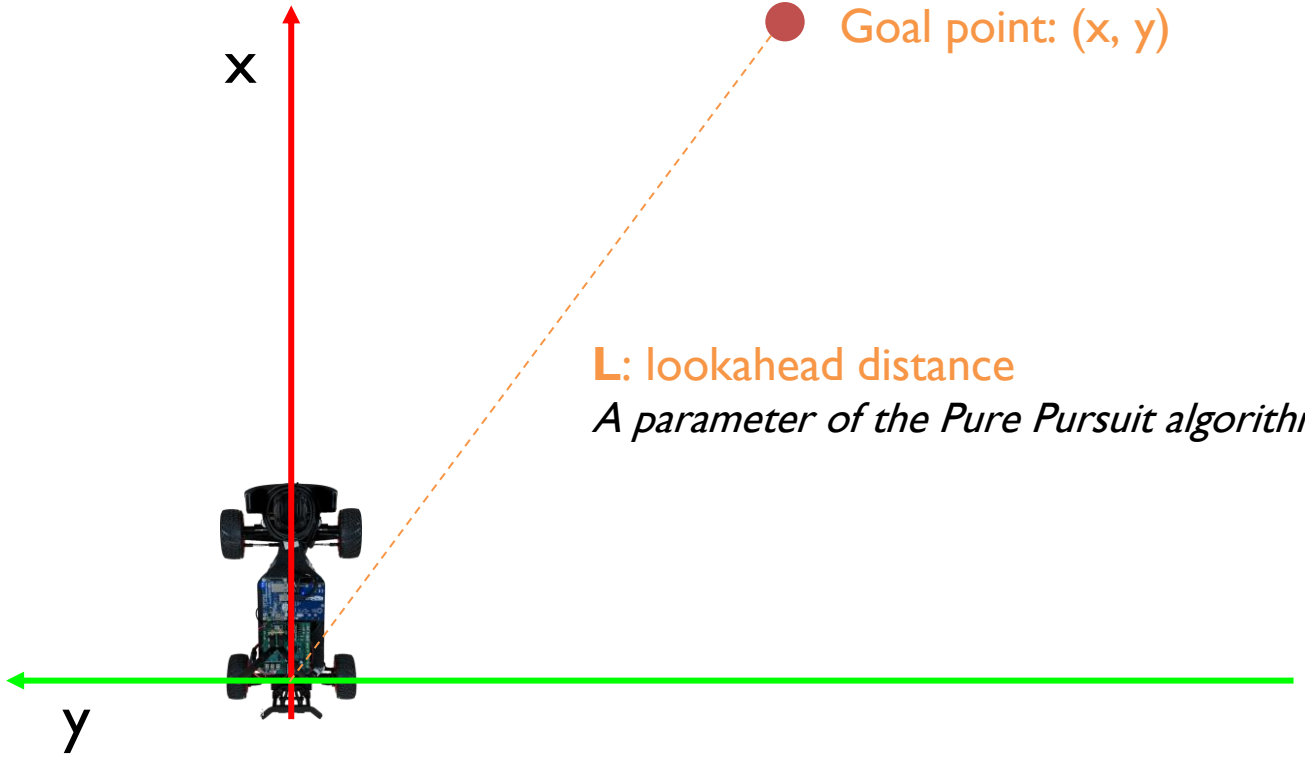
F1 TENTH

Assumptions

1. Vehicle **is given** a sequence of 2D positions, *i.e. waypoints,* to follow
   - Planning phase already done
2. Vehicle knows where the given waypoints are in the vehicle's frame of reference
   - Underlying assumptions that the vehicle can localize itself
3. Goal is to follow these waypoints



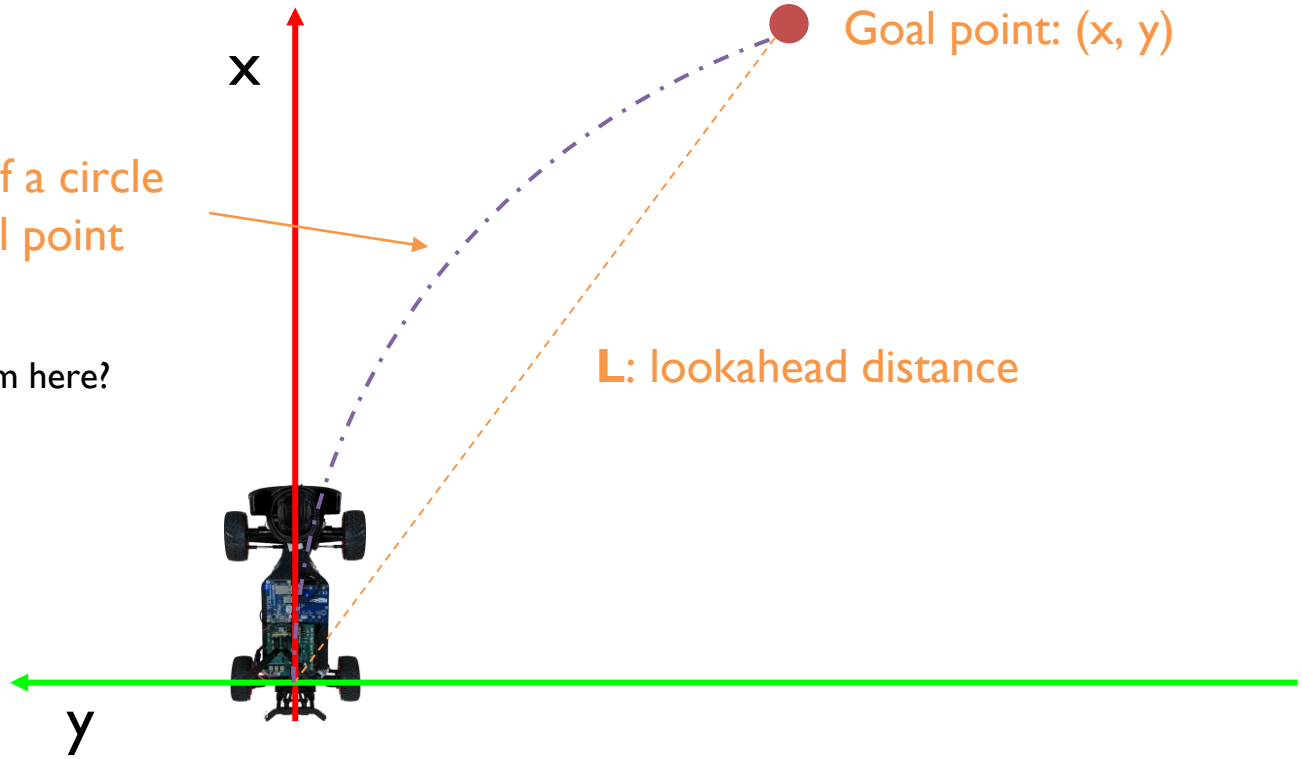
Geometric Interpretation



Geometric Interpretation

Follow the arc of a circle to reach the goal point

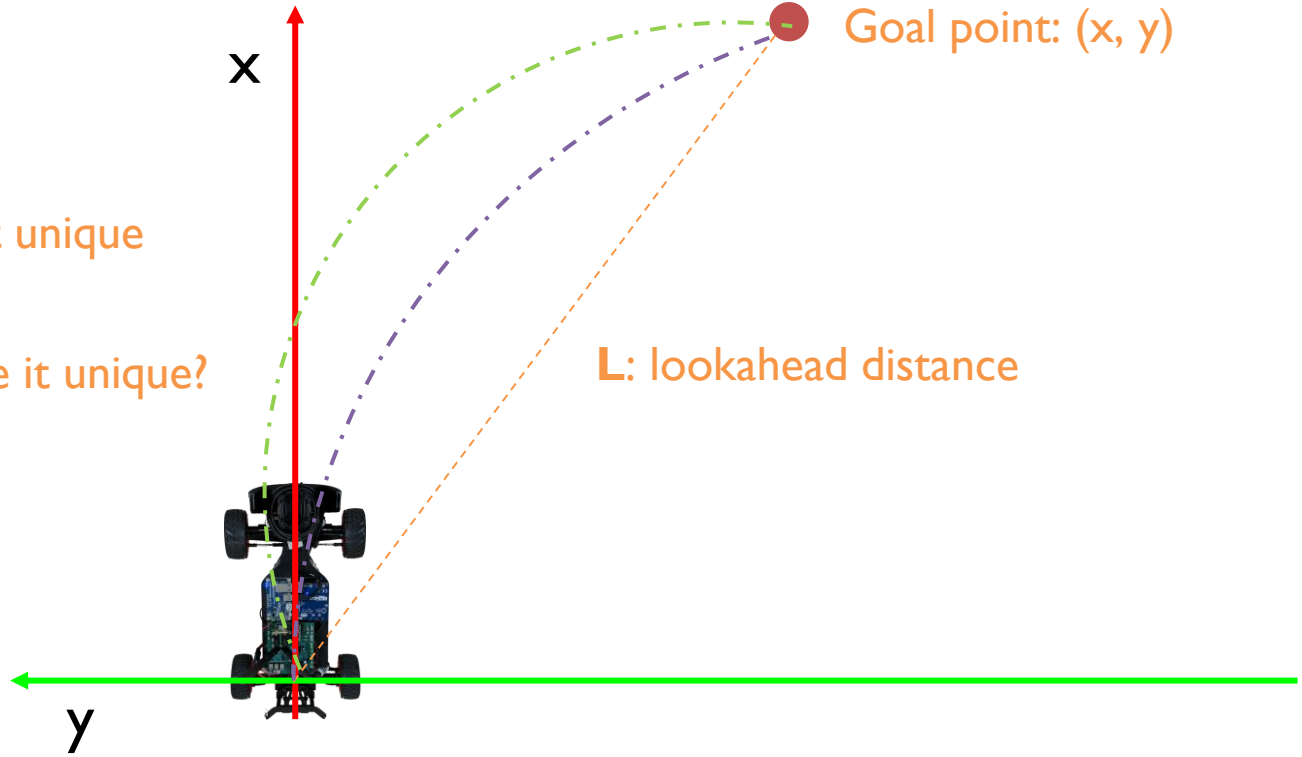
Do you see a problem here?



Geometric Interpretation

But the arc is not unique

How do we make it unique?



Finding the right arc to the next waypoint

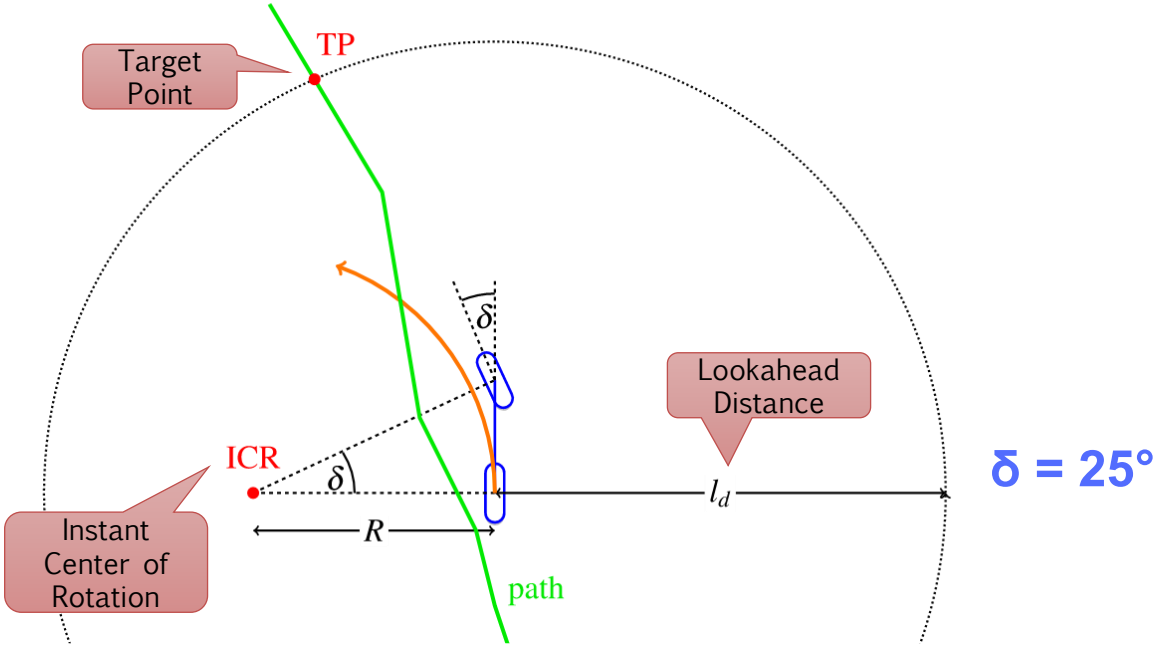
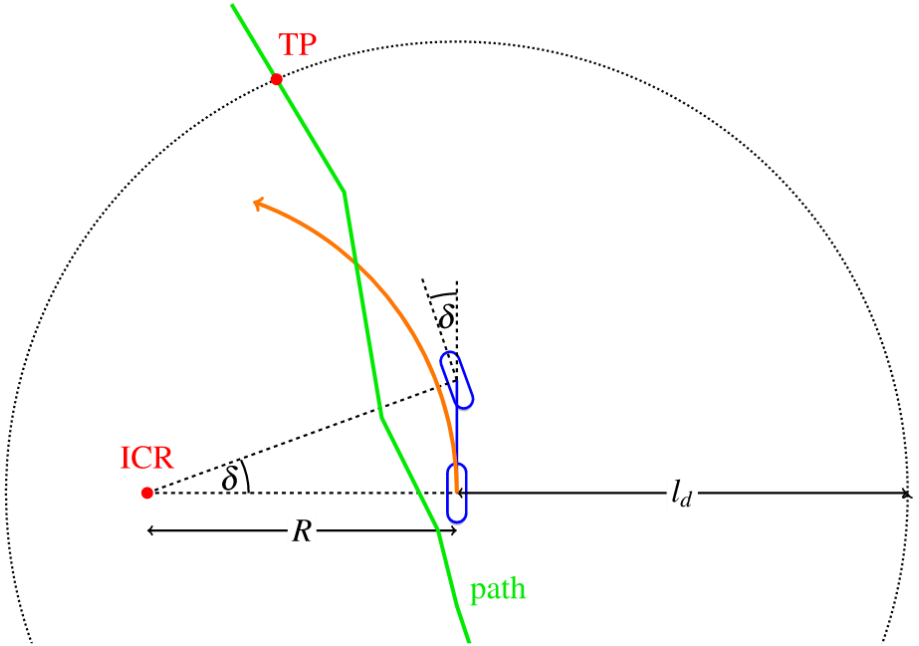


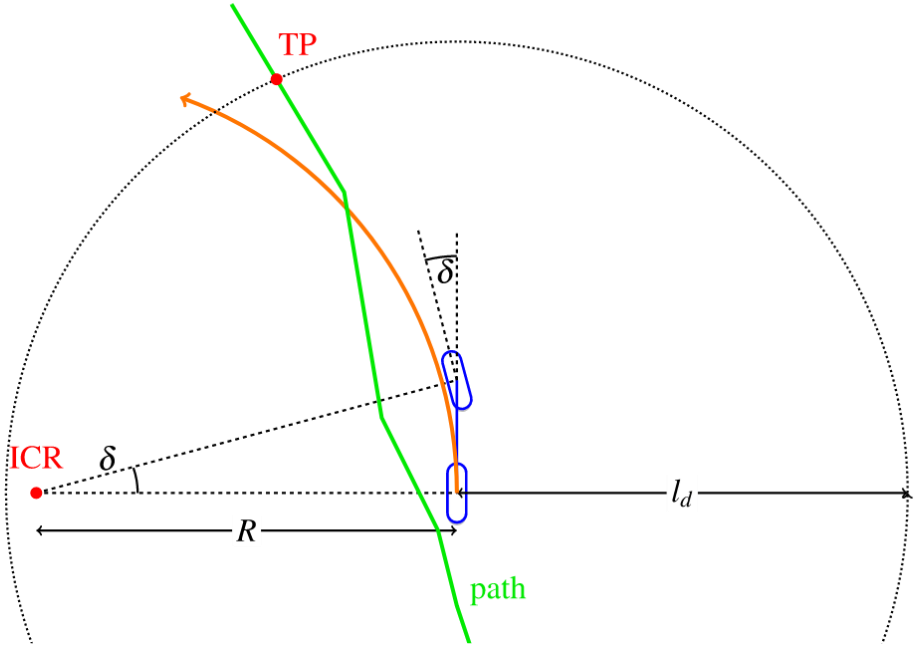
Image credit: https://github.com/thomasfermi/Algorithms-for-Automated-Driving

Finding the right arc to the next waypoint



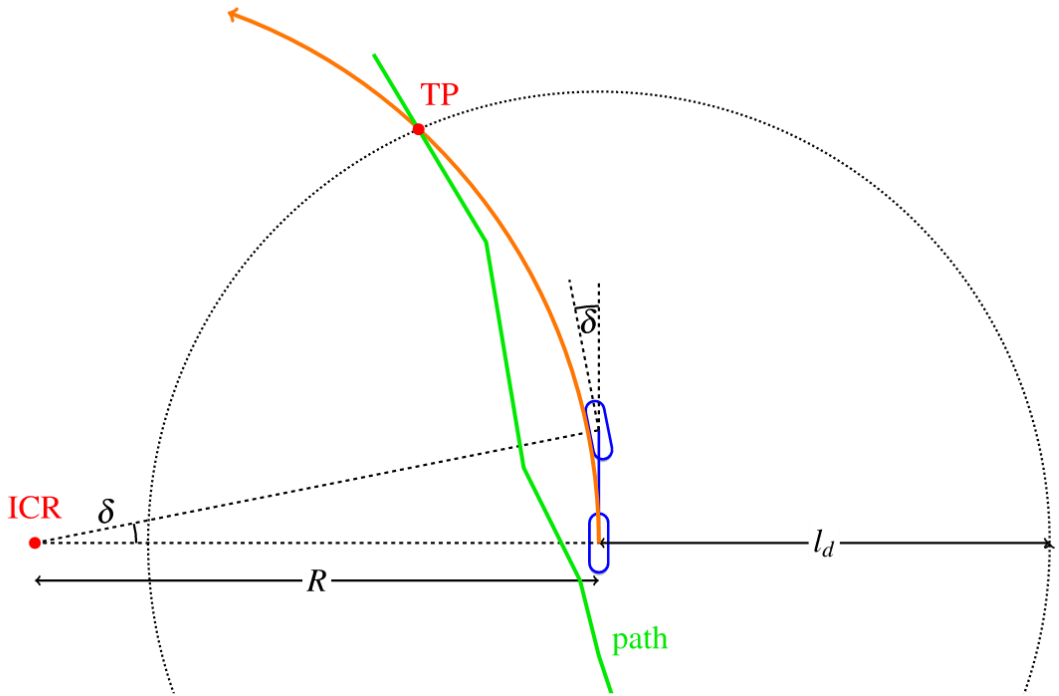
δ = 20°

Finding the right arc to the next waypoint



δ = 15°

Finding the right arc to the next waypoint

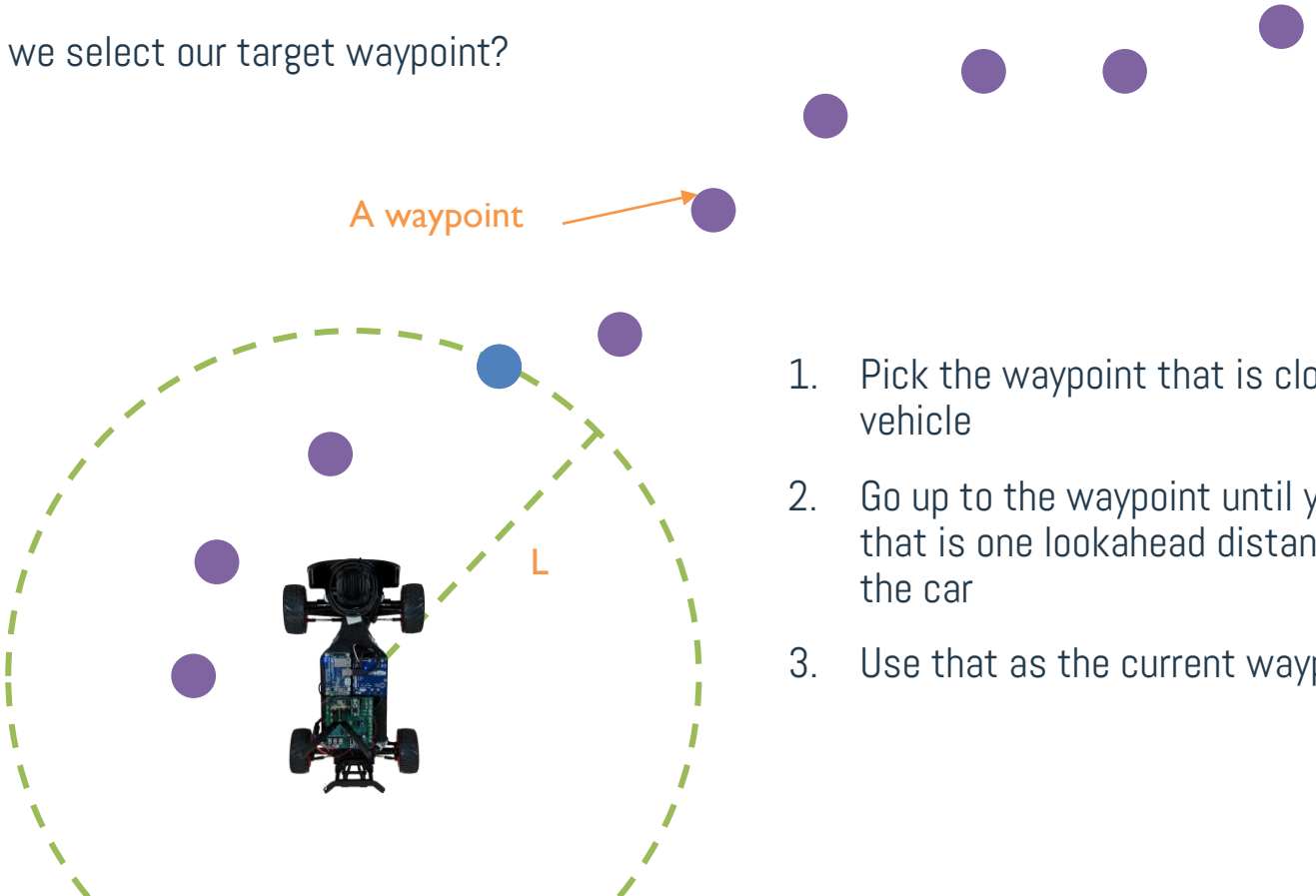


δ = 11.3°

The distance from ICR to TP is equal to *R*, since TP lies on the orange circle of radius *R* around ICR.

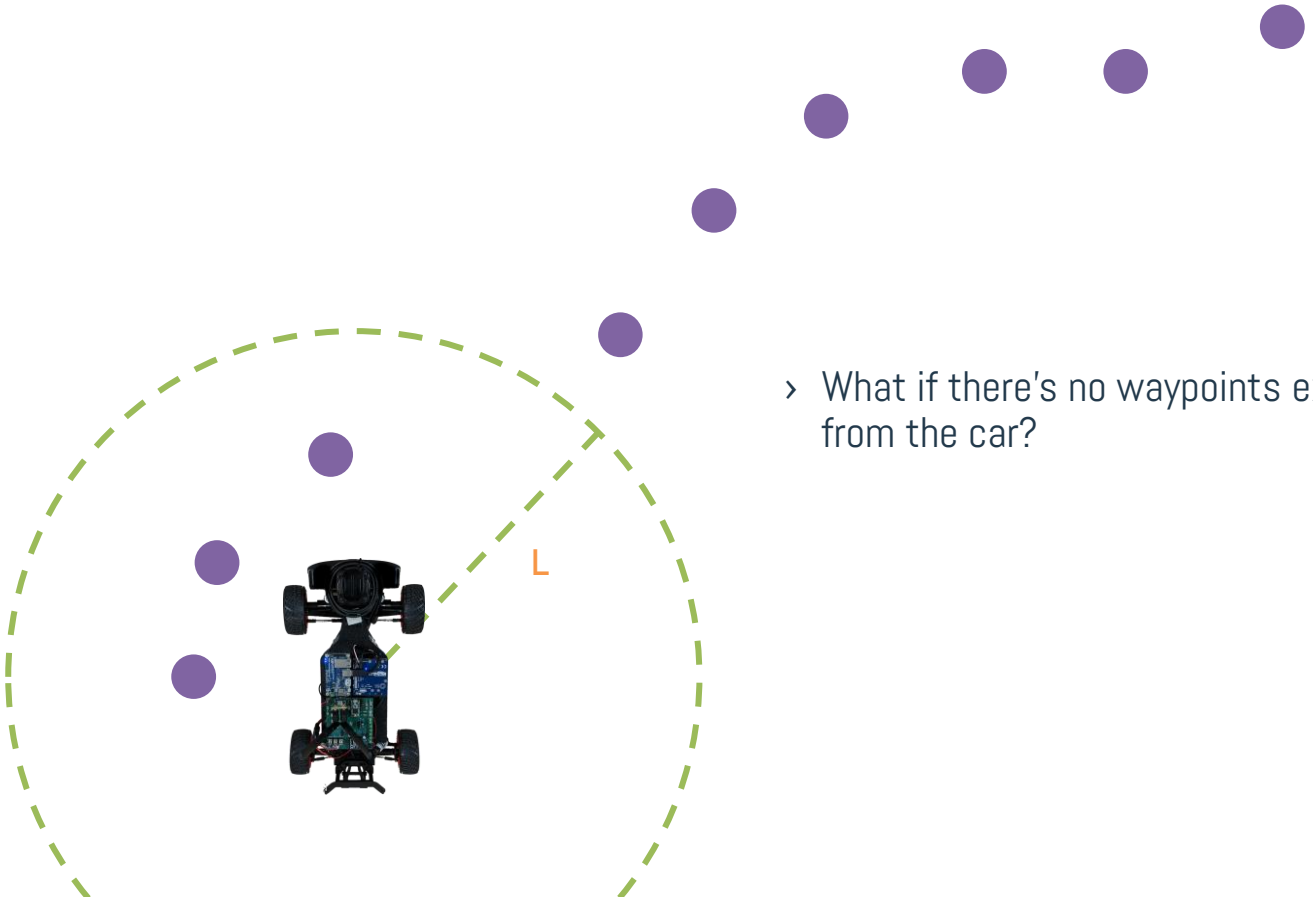
Picking a goal point

› How do we select our target waypoint?



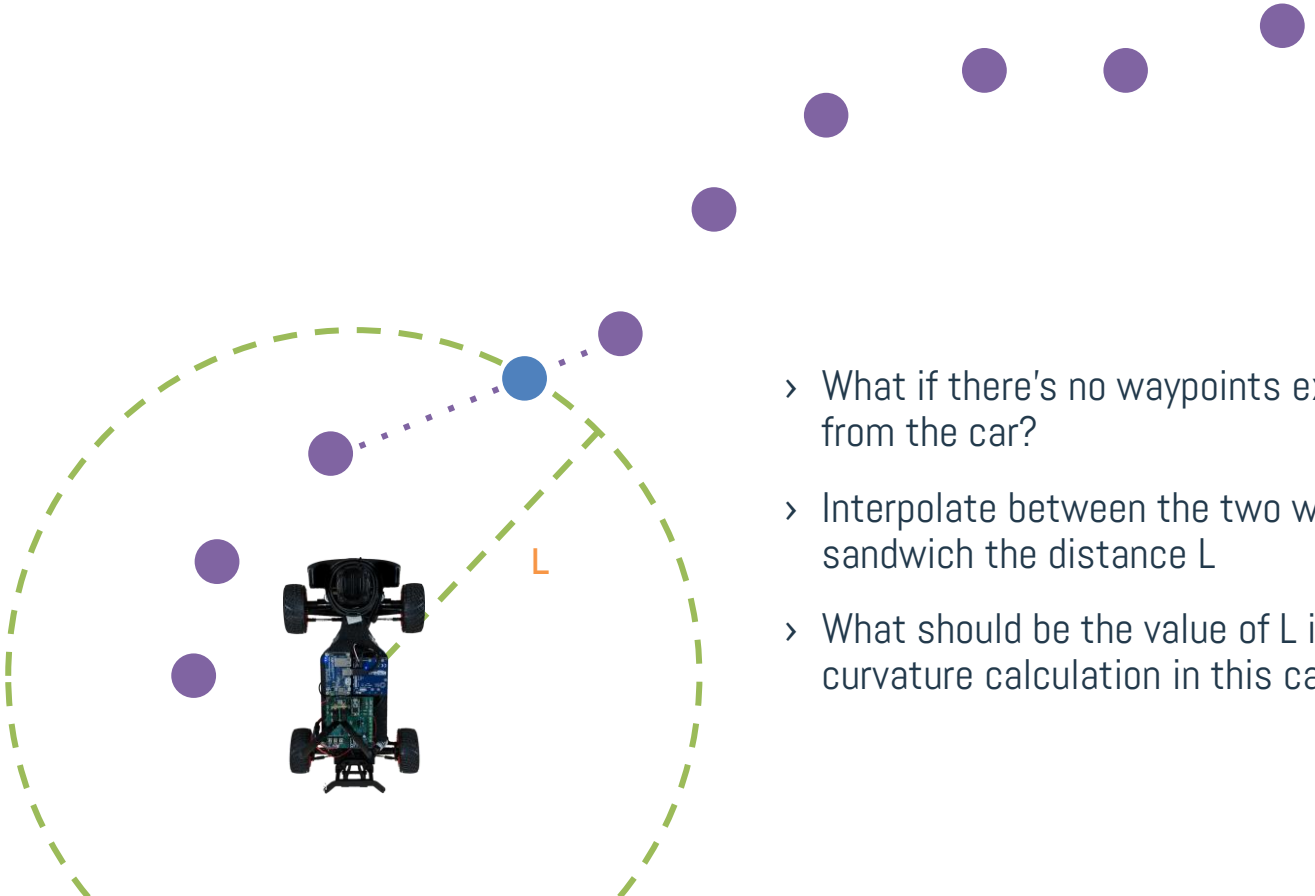
1. Pick the waypoint that is closest to the vehicle
2. Go up to the waypoint until you get to one that is one lookahead distance away from the car
3. Use that as the current waypoint

Picking a goal point



› What if there's no waypoints exactly L away from the car?

Picking a goal point



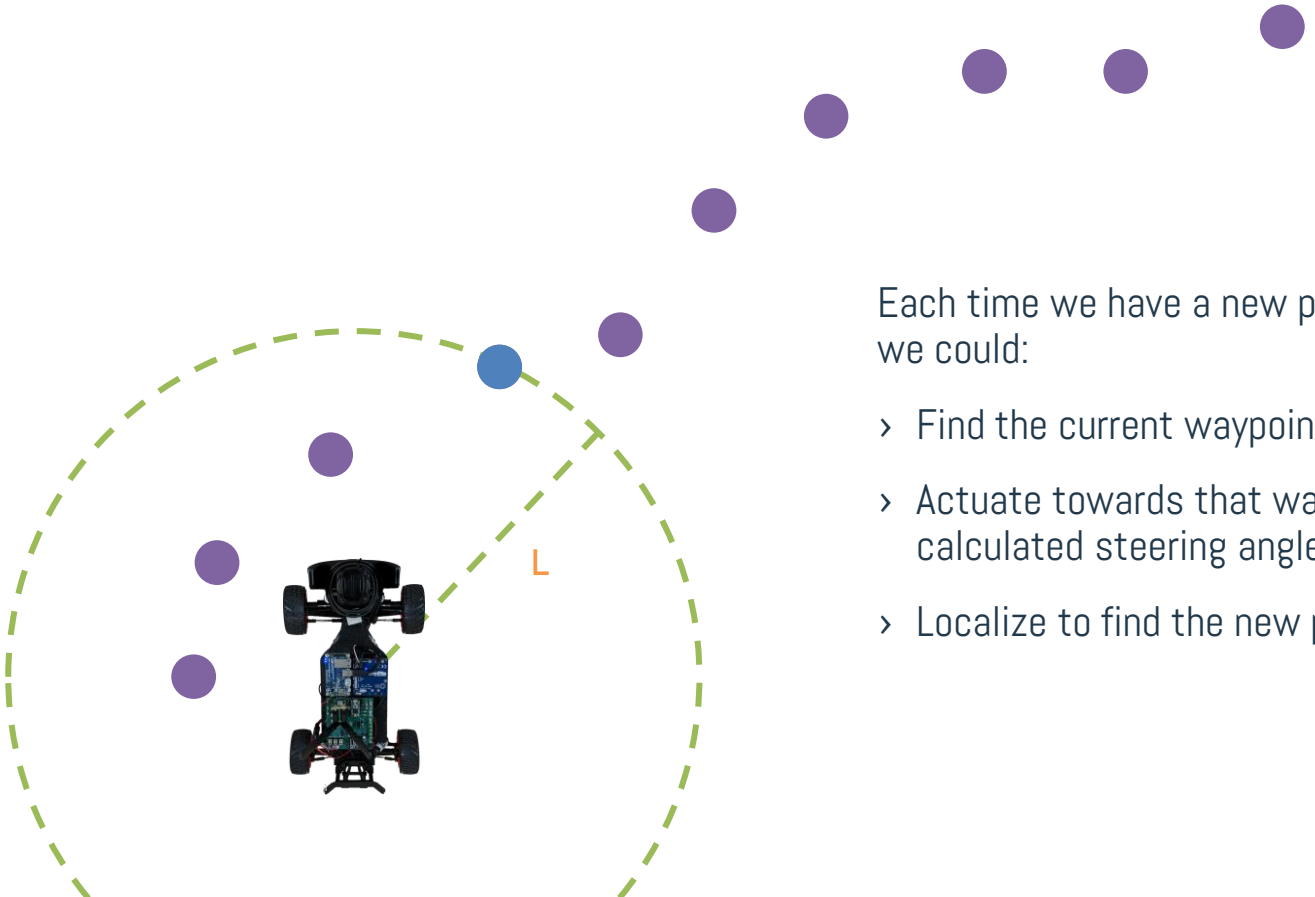
- What if there's no waypoints exactly L away from the car?
- Interpolate between the two waypoints that sandwich the distance L
- What should be the value of L in your curvature calculation in this case?

Updating the goal point (one way to do it)

Each time we have a new pose of the car, we could:

- Find the current waypoint
- Actuate towards that waypoint with calculated steering angle
- Localize to find the new pose, repeat

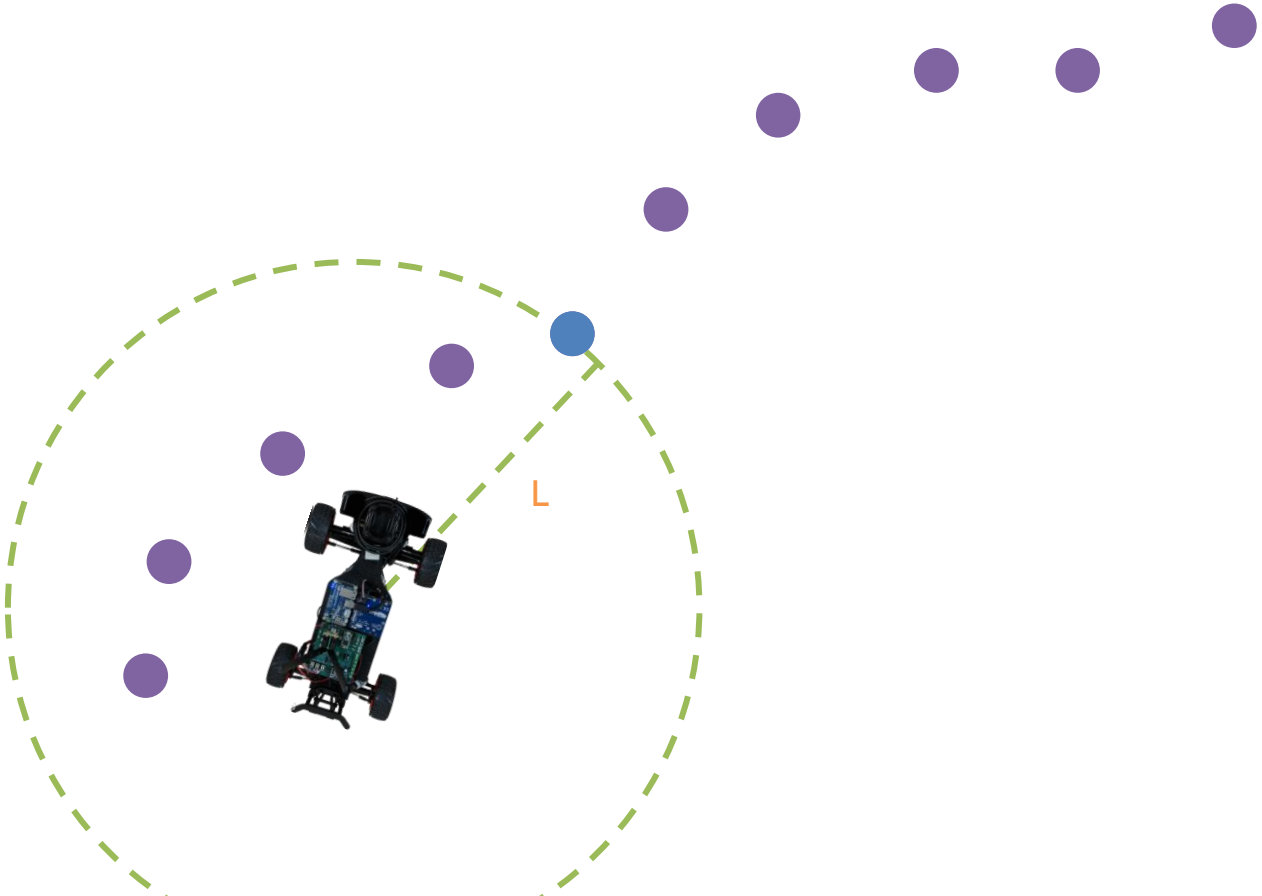
Updating the goal point (one way to do it)



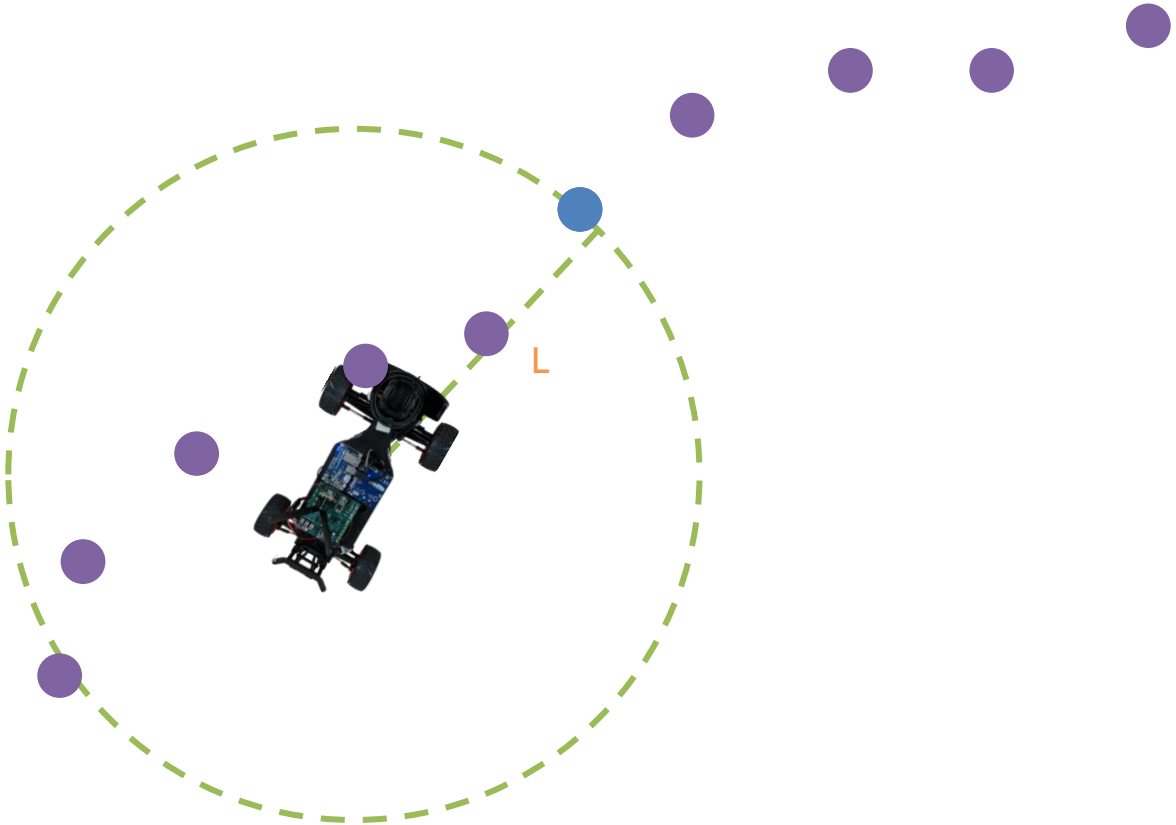
Each time we have a new pose of the car, we could:

- Find the current waypoint
- Actuate towards that waypoint with calculated steering angle
- Localize to find the new pose, repeat

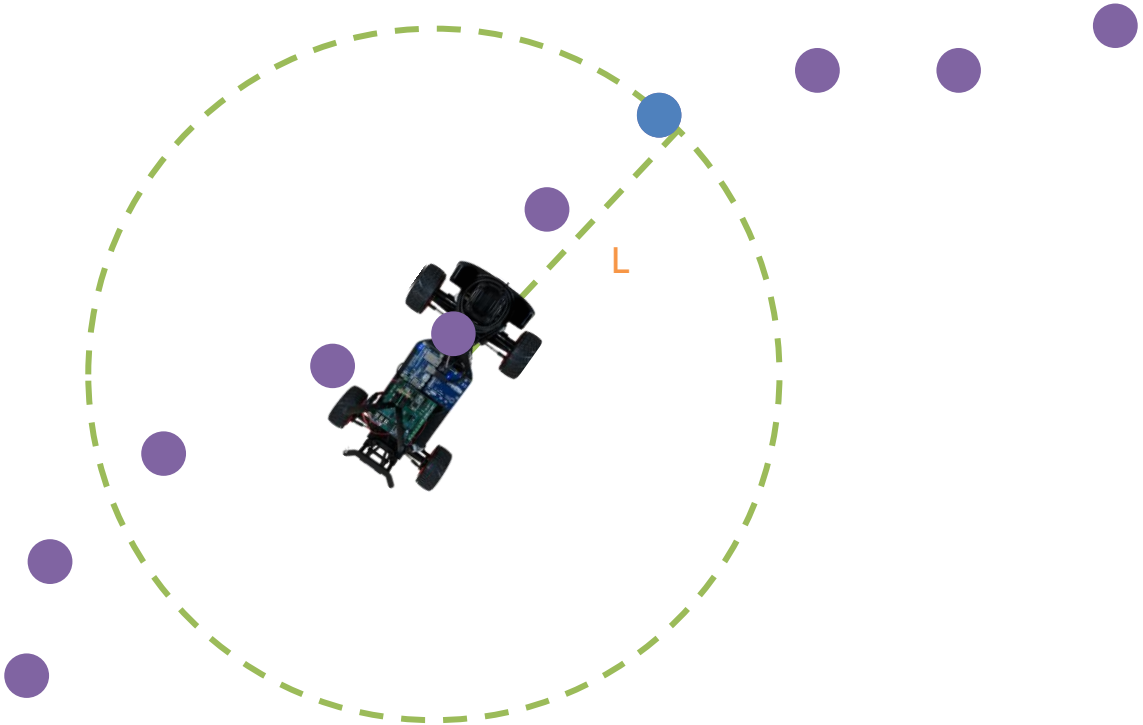
Updating the goal point (one way to do it)



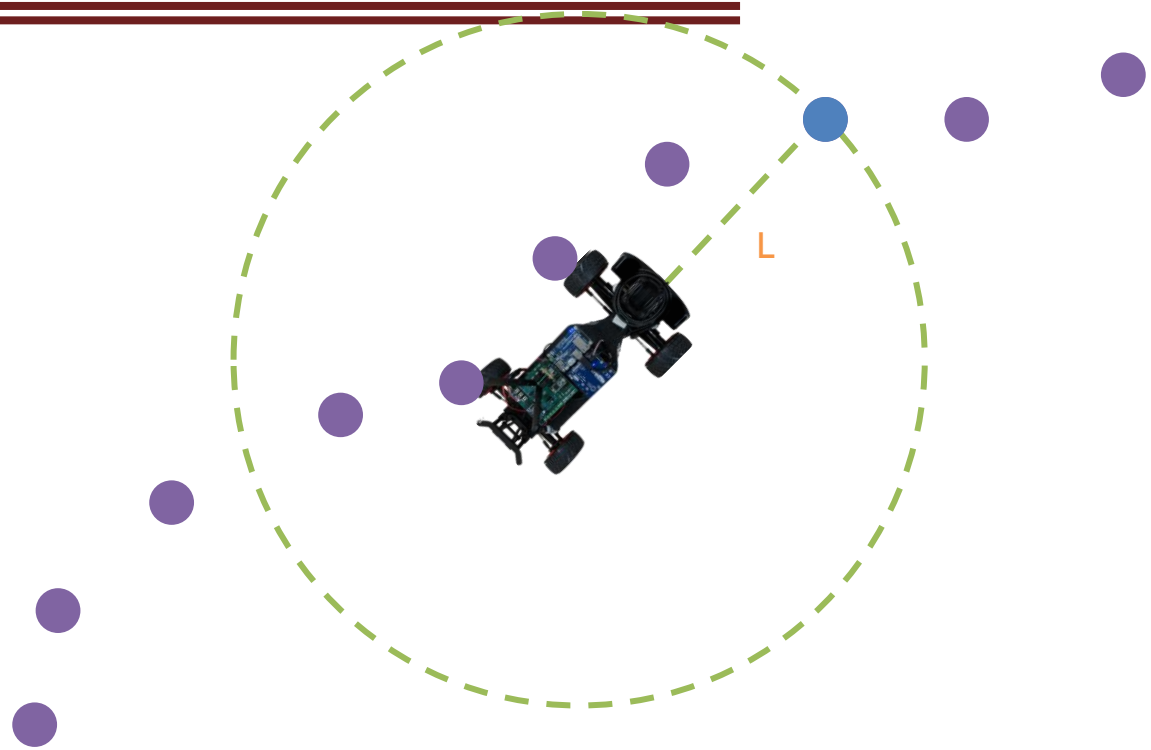
Updating the goal point (one way to do it)



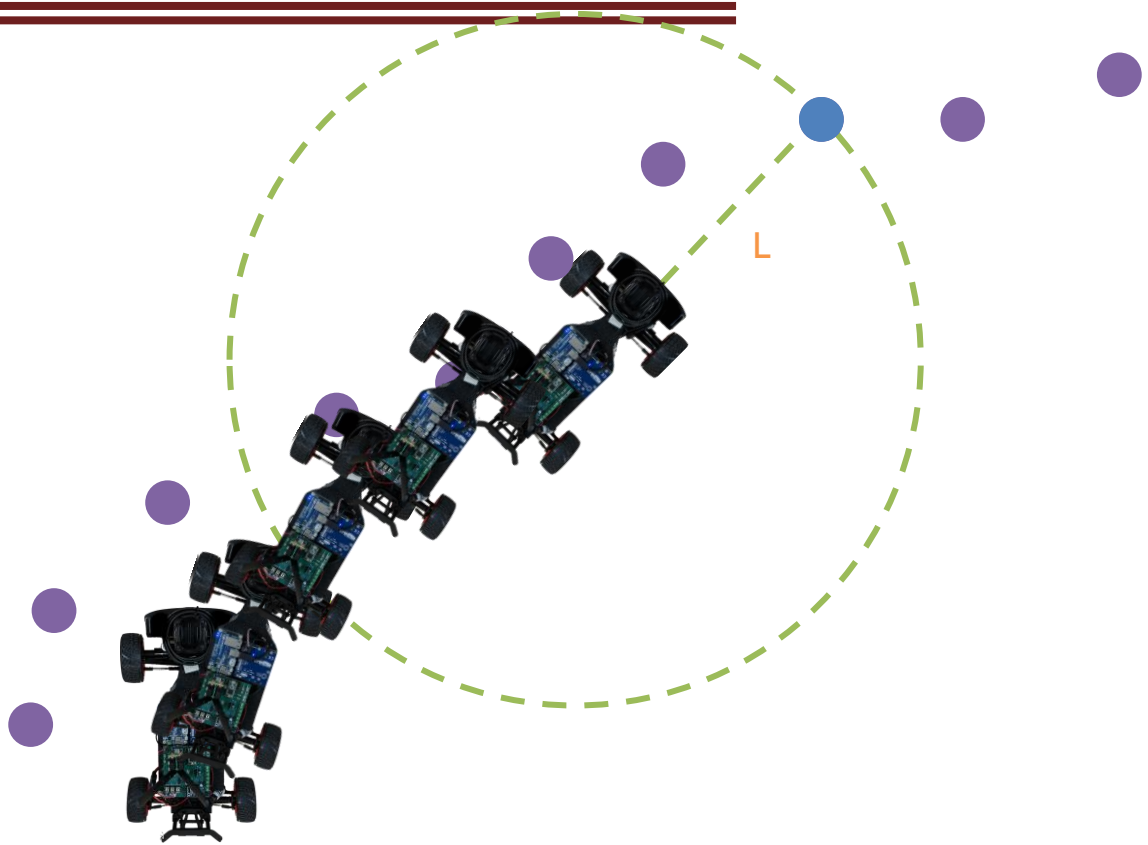
Updating the goal point (one way to do it)



Updating the goal point (one way to do it)

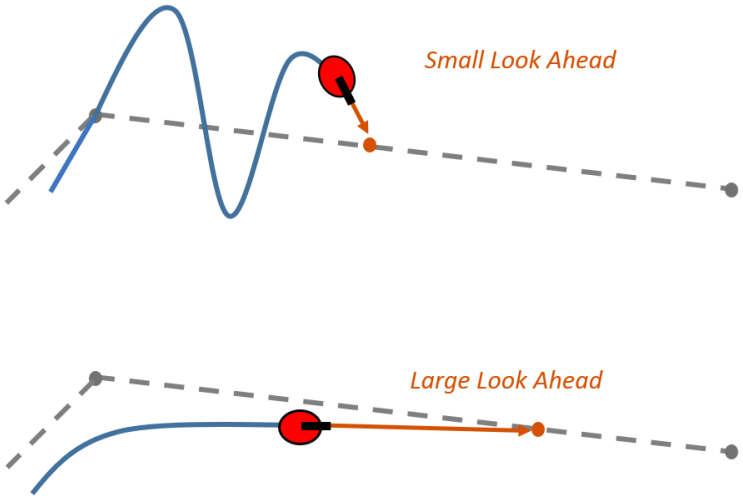


Updating the goal point (one way to do it)



Effects of Changing the Lookahead Distance

- The parameter L (lookahead distance) is a parameter of pure pursuit.
- Smaller L leads to more aggressive maneuvering to track tighter arc, and the tighter arcs might be against dynamical limits of the car.
- Larger L leads to smoother trajectory but larger tracking errors, might lead to close calls with obstacles.



Small Lookahead

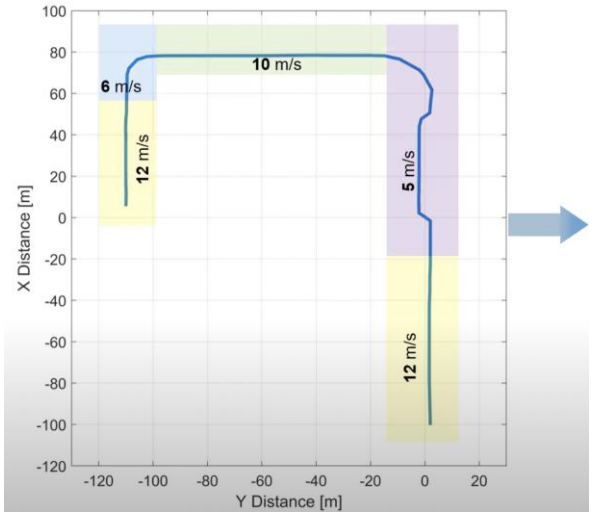
- Oscillatory path
- Accurate tracking
- Requires fast computation

Large Lookahead

- No oscillations
- Poor tracking

Observation

- Tuning **L** will change the behavior of pure pursuit the most.
- The waypoints are a sequence of pre-computed positions (planner)
- Could also have a velocity component at positions



		Velocity lookup table		

Global vs. local planners

Global planners (aka: static planners) work on the track map

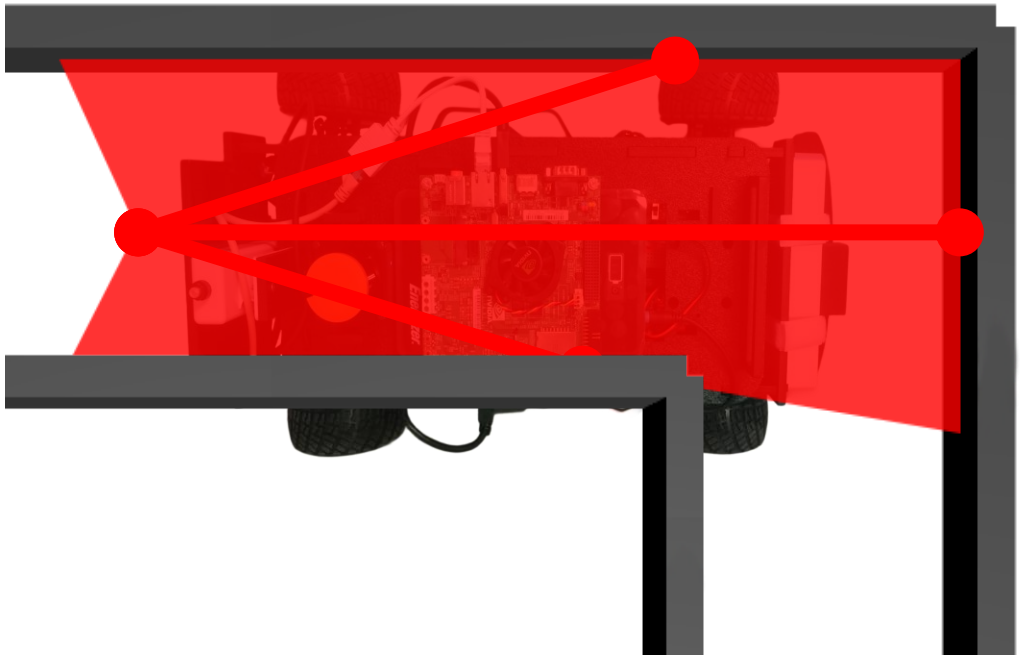
- Pro: optimal trajectories and speed
- Pro: can pre-compute the optimal trajectory (don't care how much it takes), and store in memory for simply retrieving it (fast)
- Con: need to build the map!
- Con: need to localize
- Con: cannot deal with dynamic obstacles

Local (aka: dynamic, reactive) planners

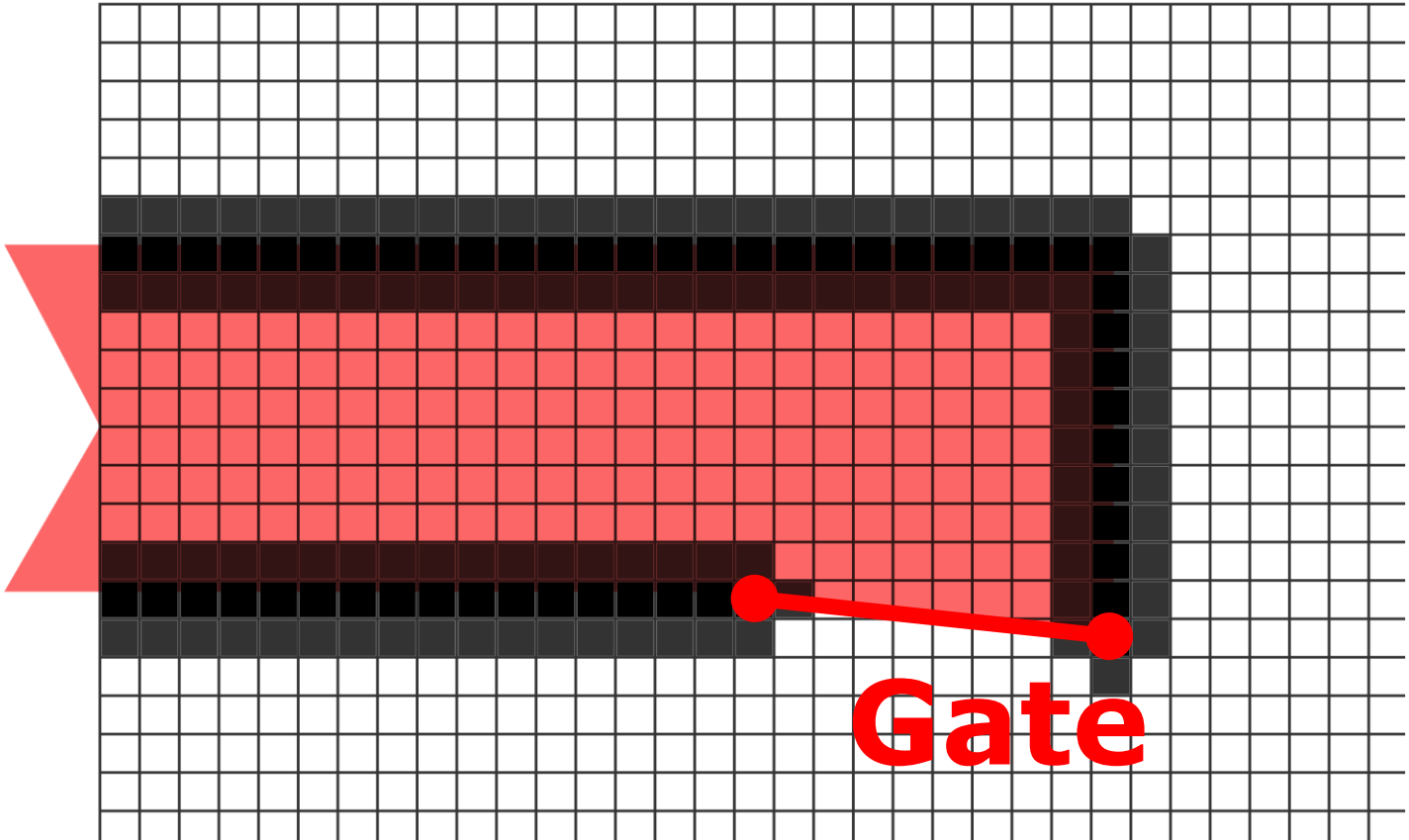
- Pro: no map is required, nor localization
- Pro: work with dynamic obstacles
- Con: must be fast!

Local planner – Follow-the-Gap

› Credits: bachelor thesis of Francesco G. (https://github.com/ceccocats)

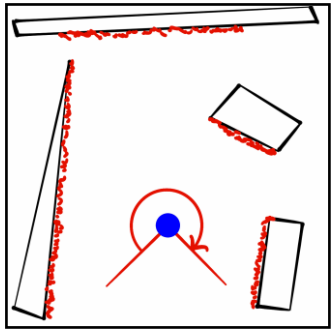


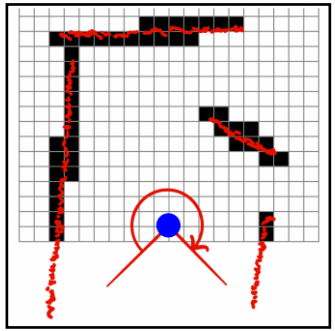
2d map



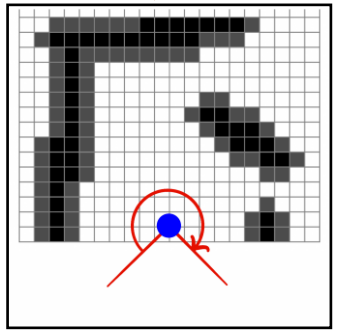
Process data to find the gates

Raw data (/map)

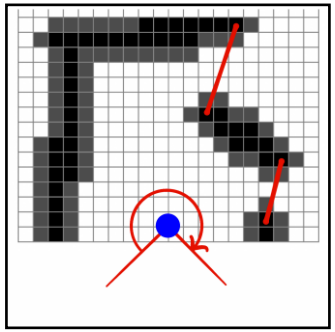




2d map w/occupancy



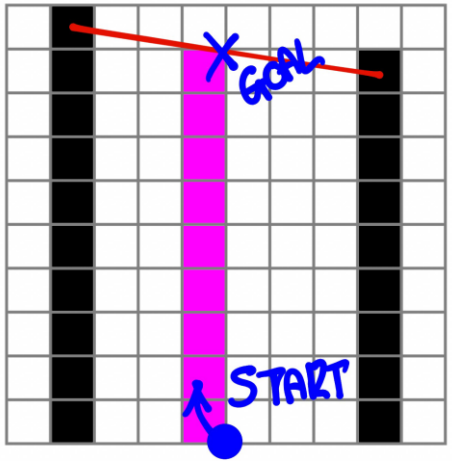
...a bit of processing...



Find the gates

Pathfinder: go to gate

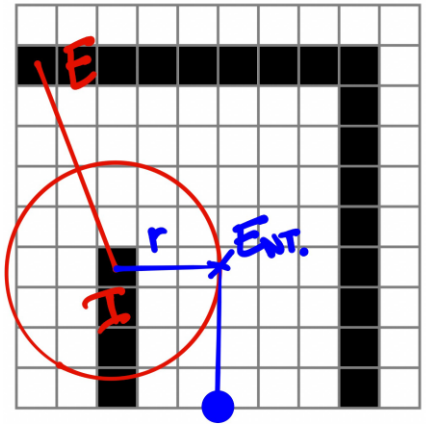
1. No curves: trivial

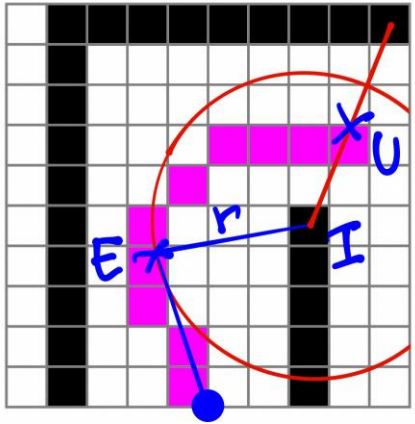


2. With curves: vehicle has a size! (safety circle of radius r)

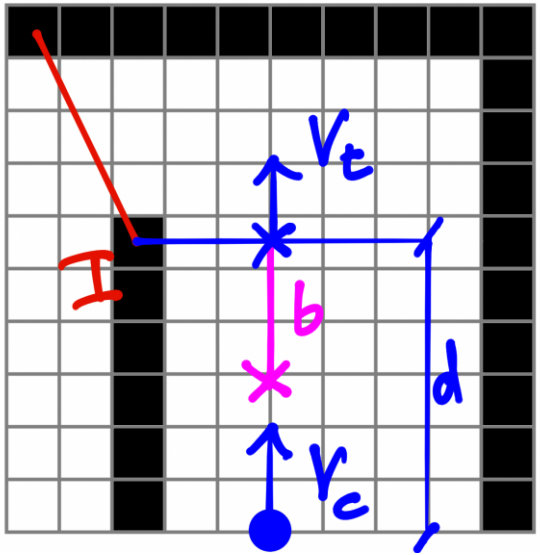
Three relevant points

- Internal
- External
- ENTry point





Throttling, aka: when shall I brake?



Known

- V_c: vehicle speed
- V_t: target speed
- *d*: distance to turn
- *dec*: max allowed deceleration

Compute

- *b*: minimal distance to brake

$$b = \frac{V_c^2 - V_t^2}{2 \cdot dec}$$

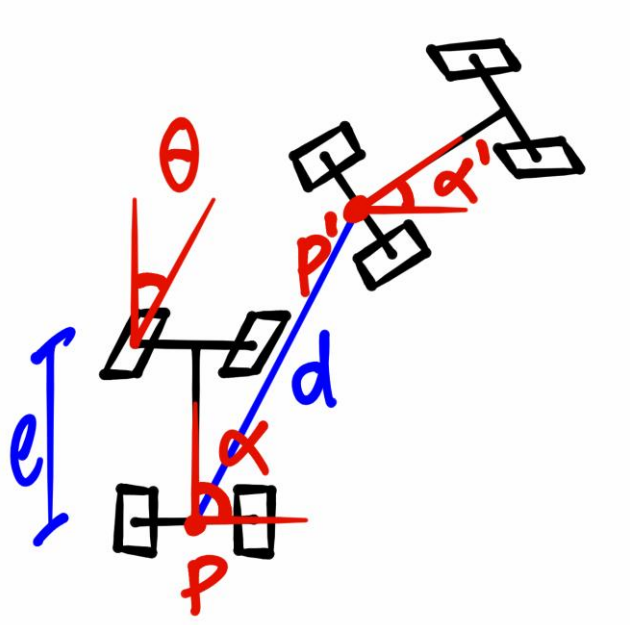
Steering: possible trajectories

Ackermann model. Given:

- Θ: max steer for front wheels
- l: inter-axes distance
- *steer*: signal [-1, +1]

➔find θ

$$\theta = steer * \Theta$$



Then, you can find compute the new position $p''$and angle α'

$$\begin{cases} px' = px + \cos(\alpha + \theta) \\ py' = py + \sin(\alpha + \theta) \\ \alpha' = \alpha + \frac{d}{l} + tan(\theta) \end{cases}$$

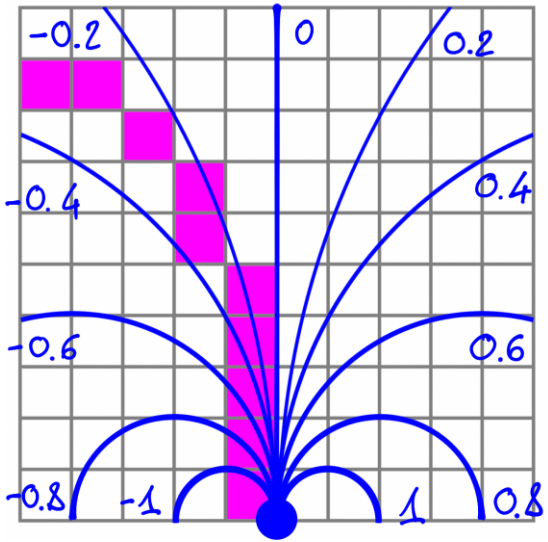
Steering: possible trajectories

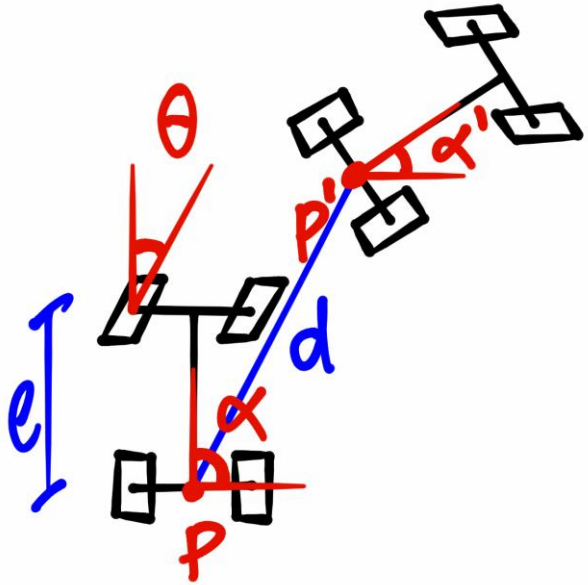
Iterate over multiple values of *steer*, with

$$d = \frac{cell\_len}{2}$$

Check if the computed trajectory matches with the path to follow

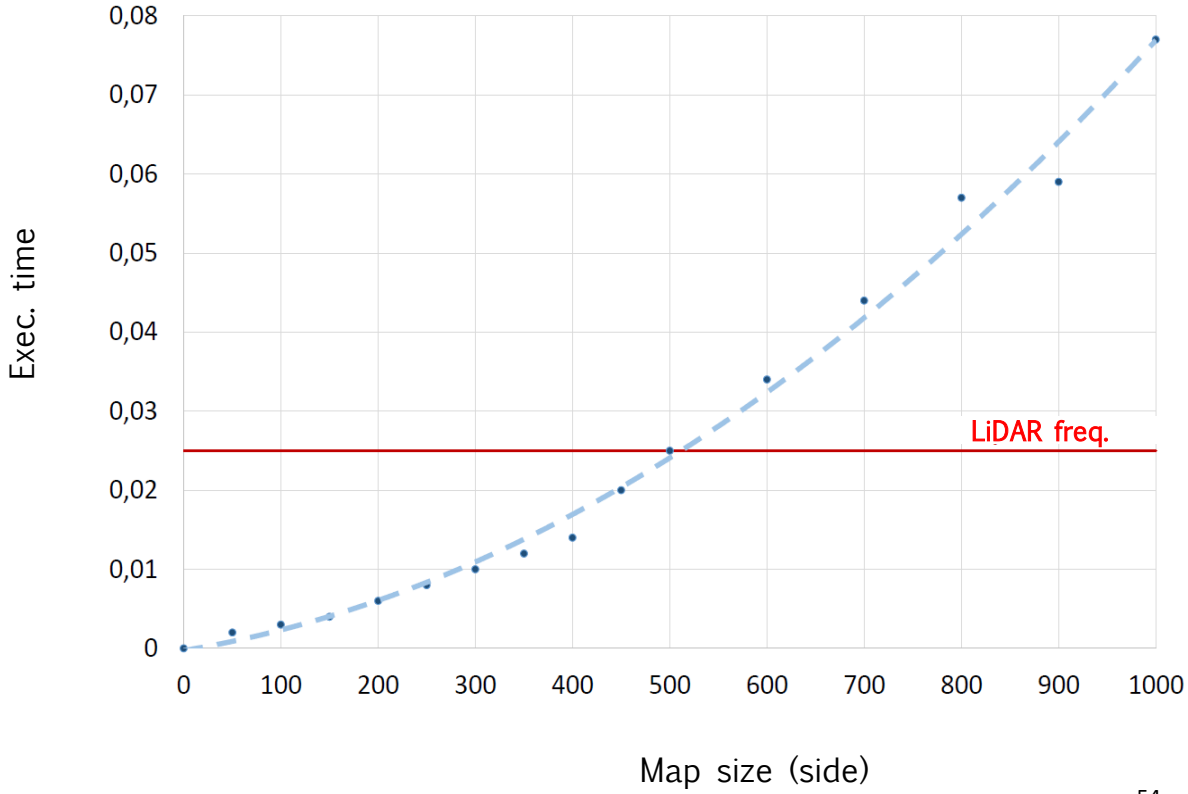
› *Compute match fn*





Timing requirements

- Bigger map: more precise, but more computation
- Remember: 25 ms!



References

Course website

- http://personale.unimore.it/rubrica/contenutiad/markober/2023/71846/N0/N0/10005
- https://github.com/HiPeRT/F1tenth-RTES
  - Online resources/preview

My contacts

- paolo.burgio@unimore.it
- http://hipert.mat.unimore.it/people/paolob/

Resources

- A "small blog"
  - http://www.google.com