CS 3630, Fall 2026

Lecture 26

Planning and Learning



With lots of slides and ideas from:

Howie Choset, Steve Lavalle, Greg Hager, Zack Dodds, Nancy Amato, Sonia Chernova, James Kuffner

Outline

1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning



Path Planning – a quick review

- path planning = finding a **collision-free path** from a start to a goal.
- The best algorithms have time complexity that increases **exponentially** in the dimension of the configuration space.
  - low-dimensional configuration spaces: often find exact solutions.
  - high-dimensional configuration spaces: good approximate solutions, or notions like *probabilistic completeness*.
- Typically deal only with geometry: dynamics are not considered.
- Considering **dynamics** can significantly increase the complexity.
- Cars are a special case: low-dimensional configuration space, but dynamics matter.

Mobile Robots

- In general, motion planning is intractable.
- For certain special cases, efficient algorithms exist.
- Mobile robots that move in the plane are much simpler than robot arms, mobile manipulators, humanoid robots, etc.
- The main simplifying property is that we can often treat path planning as a two-dimensional problem for a point moving in the plane, $x \in \Re^2$.

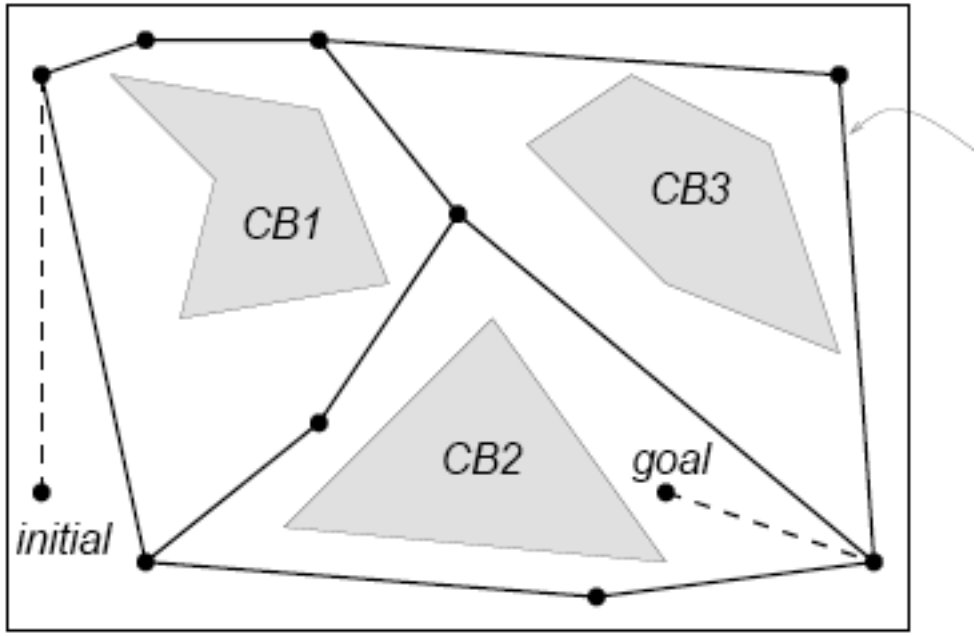
Outline

1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning



Roadmap methods

Capture the connectivity of the free space by a graph or network of paths.





Roadmaps

A roadmap, RM, is the union of one-dimensional curves such that for all x_{start} and x_{goal} that can be connected by a collision- free path:

- **Accessibility**: There is a collision-free path connecting x_{start} to some point $x_1 \in RM$.
- **Departability**: There is a collision-free path connecting x_{goal} to some point $x_2 \in RM$.
- **Connectivity**: There is a path in RM connecting x_1 and x_2.

If such a roadmap exists, then a free path from x_{start} to x_{goal} can be constructed from these three sub-paths, and the path planning problem can be reduced to finding the three sub-paths.

RoadMap Path Planning

1. Build the roadmap
   a) nodes are points in the free space or its boundary
   b) two nodes are connected by an edge if there is a free path between them

2. Connect start end goal points to the road map
   at point x_1 and x_2 , respectively

3. Find a path on the roadmap between x_1 and x_2

The result is a path from start to goal

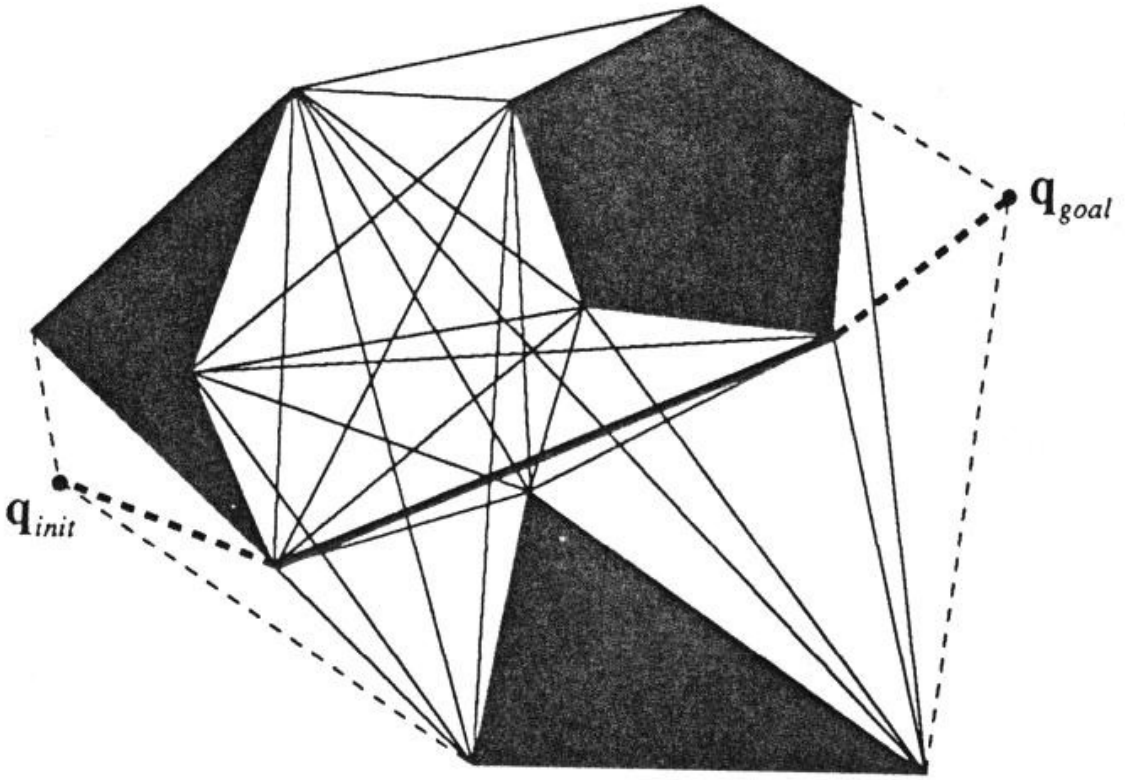
Shortest, But Possibly Dangerous Paths

The Visibility Graph

Visibility Graph methods

- Defined for polygonal obstacles
- Nodes correspond to vertices of obstacles
- Nodes are connected if
  - they are connected by an edge on an obstacle

  OR
  - the line segment joining them is in free space



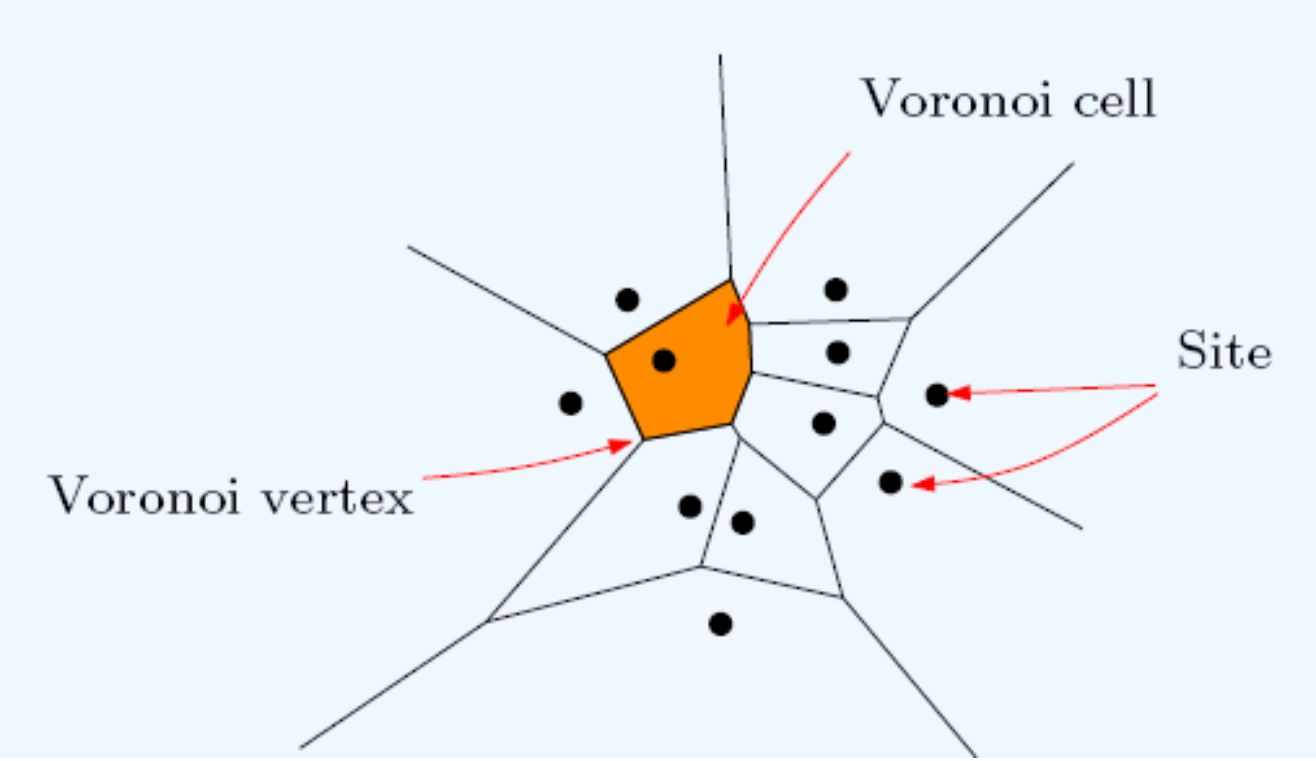
- If there is there a path, then the *shortest* path is in the visibility graph
- If we include the start and goal nodes, they are automatically connected
- Algorithms for constructing them can be efficient
  - $O(n^3)$ brute force (i.e., naïve)
  - $O(n^2 \log n)$ if clever

Outline

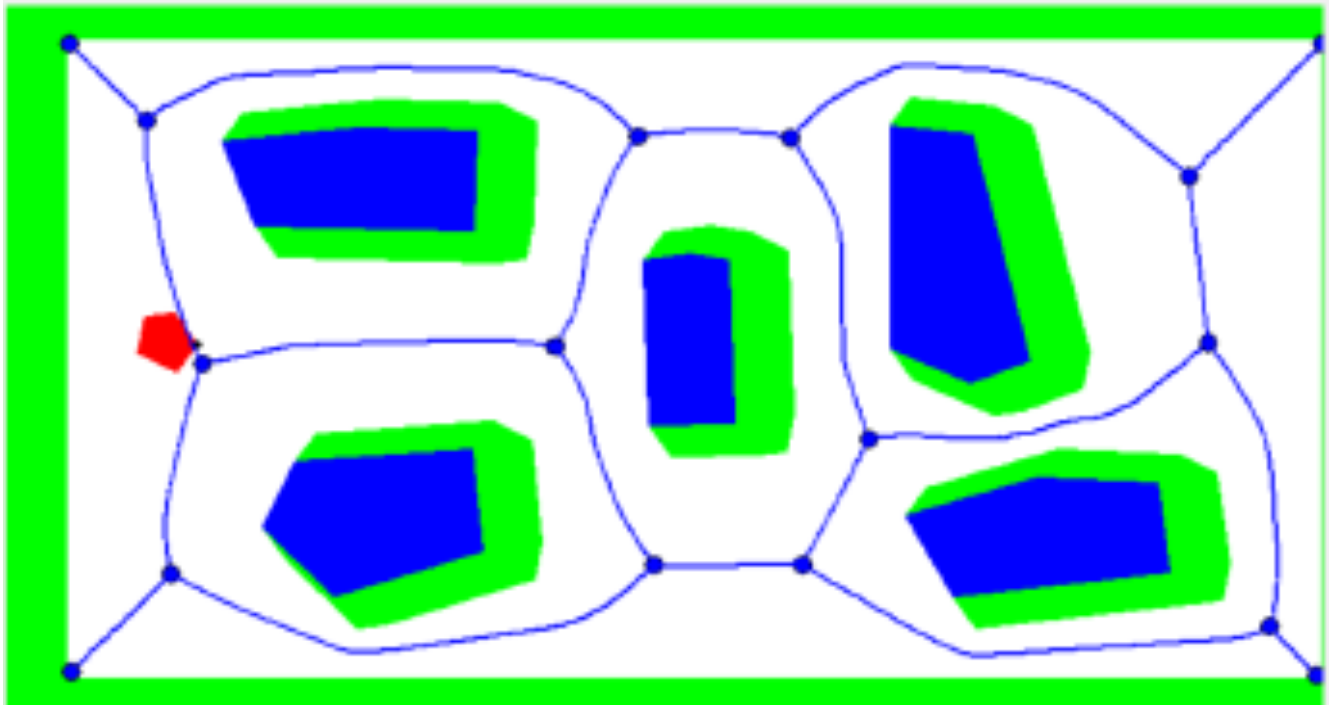
1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning



Voronoi Diagrams



Generalized Voronoi Diagrams



A Discrete Version of the Generalized Voronoi Diagram

- use a discrete version of space and work from there
  - The Brushfire algorithm is one way to do this
    - need to define a grid on space
    - need to define connectivity (4/8)
    - obstacles start with a 1 in grid; free space is zero

n1	n2	n3
n4	n5	n6
n7	n8	n9

4

n1	n2	n3
n4	n5	n6
n7	n8	n9

8

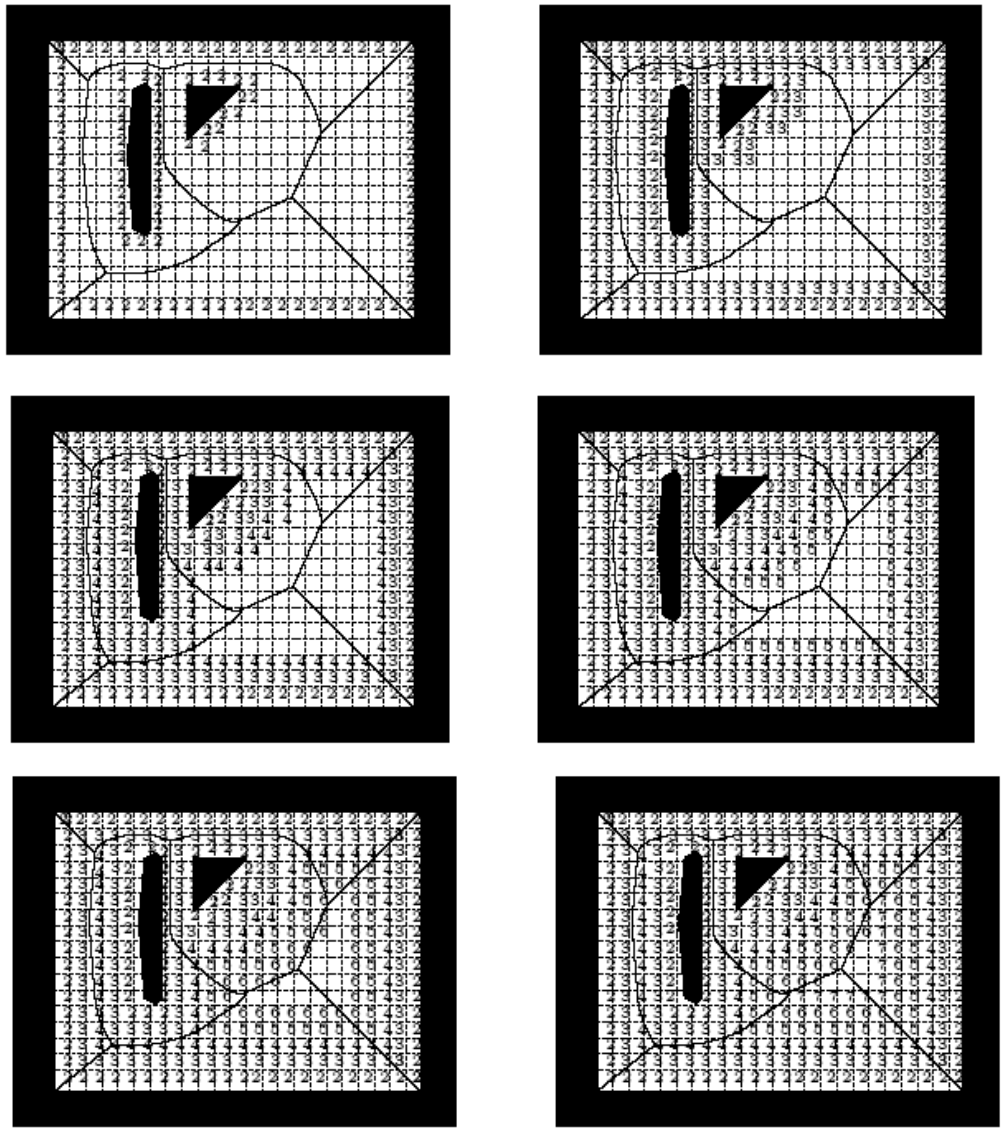
Brushfire Algorithm

- Initially: create a queue L of pixels on the boundary of all obstacles, set $d(t) = 0$ for each non-boundary grid cell t
- While $L \neq \emptyset$
  - pop the top element t of L
  - if $d(t) = 0$
    - $d(t) \leftarrow 1 + \min_{t' \in N(t), d(t') \neq 0} d(t')$
    - $L \leftarrow L \cup \{t' \in N(t) \mid d(t) = 0\}$ /* add unvisited neighbors to L

The result is a distance map d where each cell holds the minimum distance to an obstacle.

Local maxima of d define the cells at which “wave fronts” cross, and these lie on the discrete Generalized Voronoi Diagram.

Brushfire example



Note that the curves here are not at all perfect...

Application to Cars?

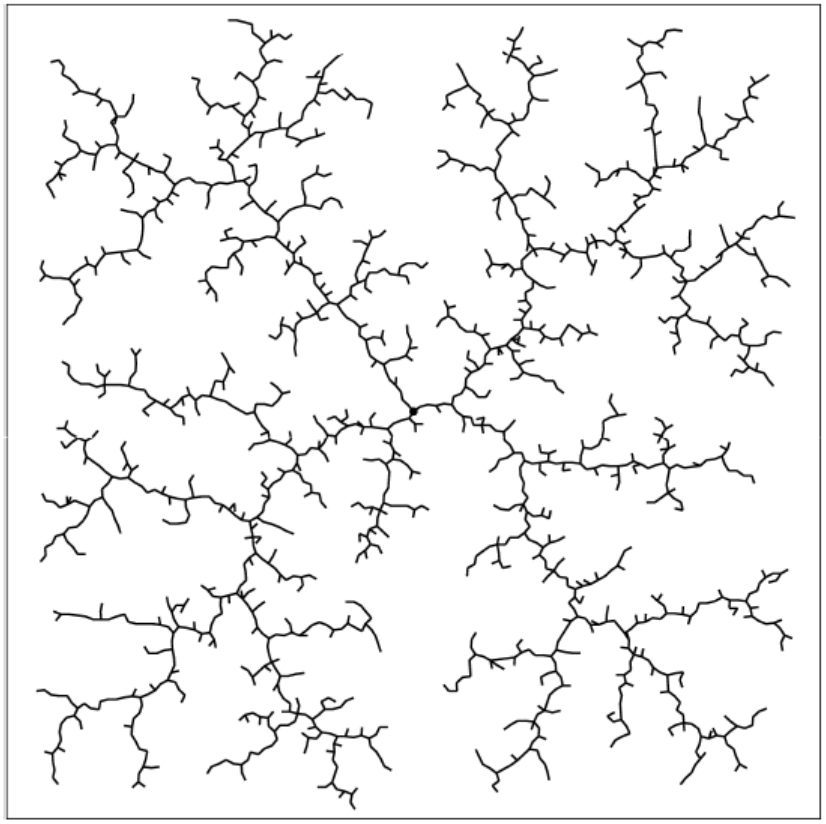
- These algorithms are great for wheeled mobile robots. Typically, these robots
  - Have dynamics that aren't significant
  - Can rotate in place, and therefore can arbitrarily change direction
  - Inhabit buildings, college compasses, other environments where free movement (i.e., not constrained to stay in a lane of a highway) is allowed.
- Cars don't have these properties, so these algorithms can be less useful when planning motions for cars.

Outline

1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning

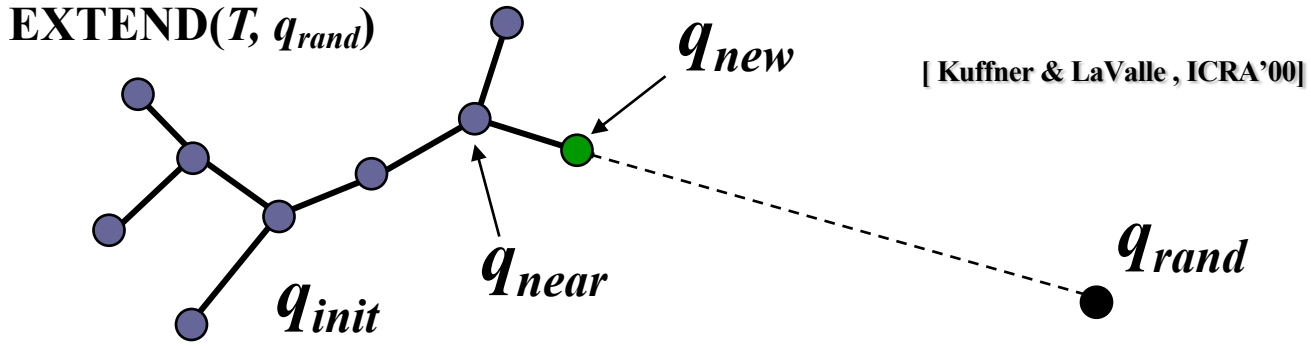


Rapidly-Exploring Random Tree

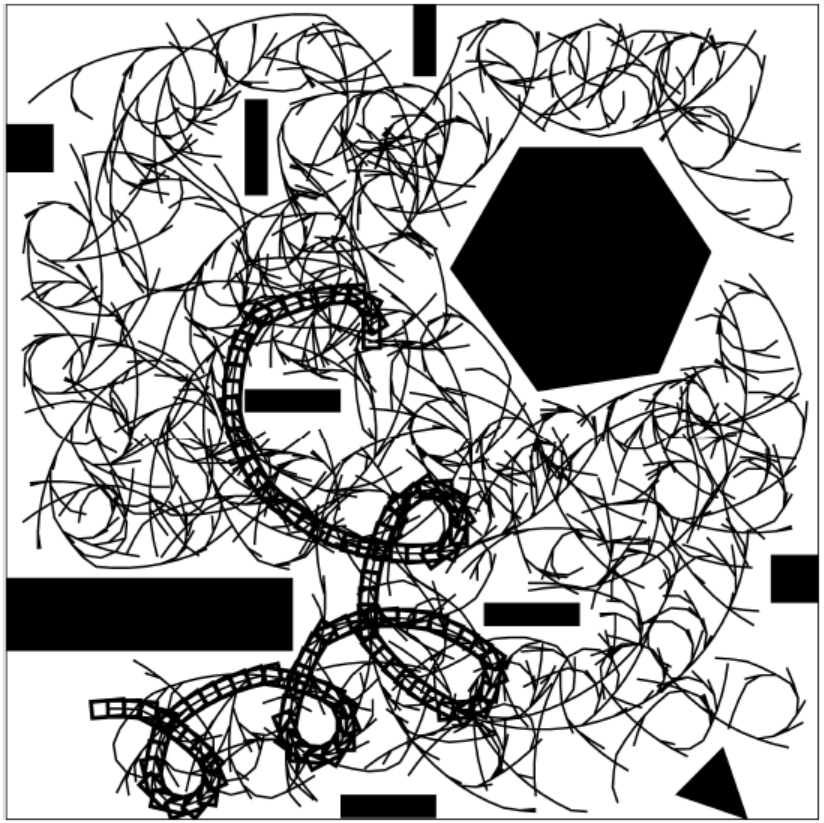


Path Planning with RRTs

```
BUILD_RRT (q_init)  {
   T.init(q_init);
   for k =  1 to K do
      q_rand = RANDOM_CONFIG();
      EXTEND(T, q_rand)
}
```



Left-turn only forward car



Outline

1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning



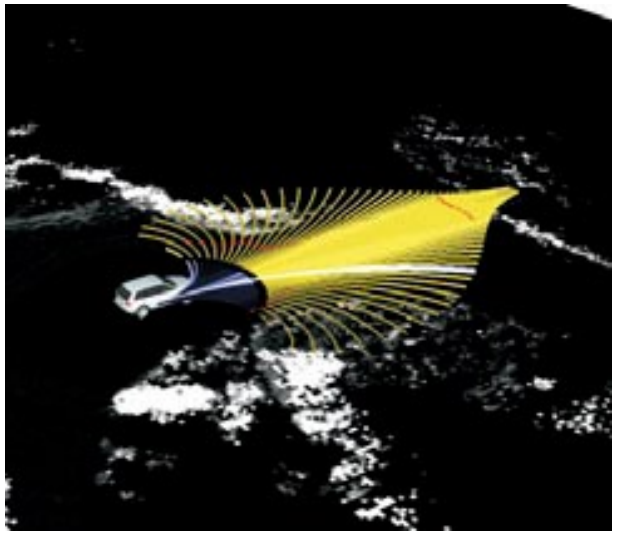
Application to Cars?

- These algorithms are great for complex planning problems, and can deal with
  - Complex dynamic models
  - Global planning problems
  - Complex environments
- This is a more complicated scenario than a car typically faces.
  For cars:
  - Local motion planning is enough (don't hit anything, stay on the road)
  - Dynamics are simple, and can often be modeled using purely geometric approaches
  - For a car, the search space for possible paths is highly constrained by the nonholonomic wheel constraints (i.e., cars can't move sideways).
  - Path Planning should be very fast!

A modification to path planning…

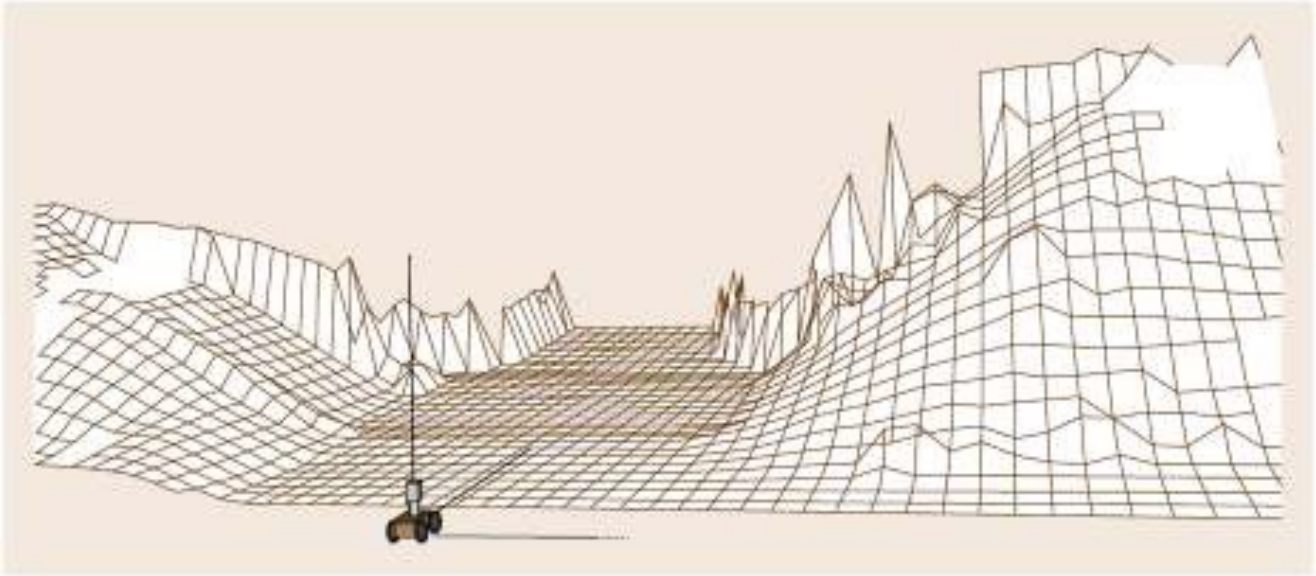
Driving with tentacles

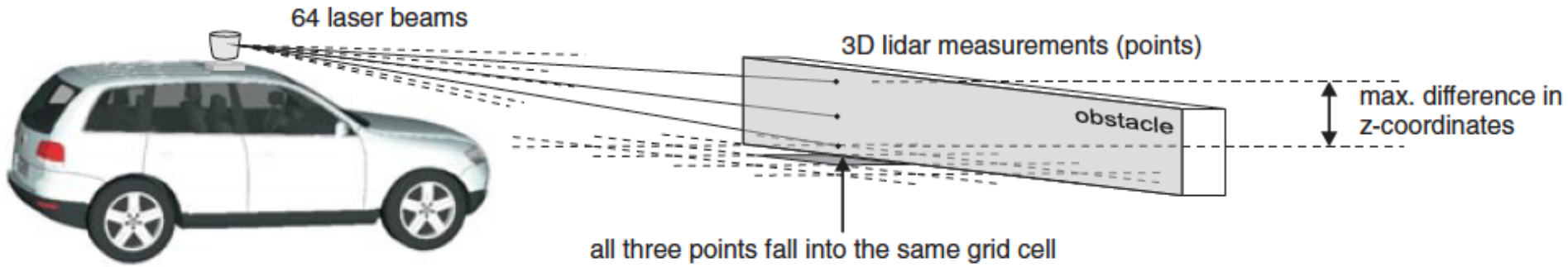


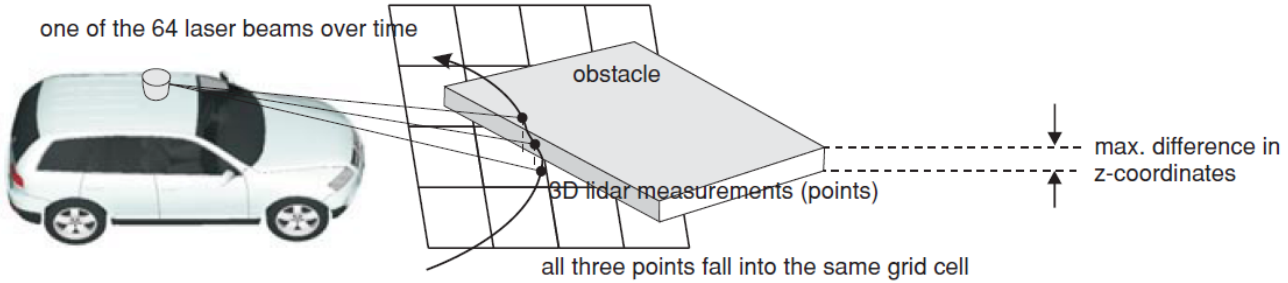


Felix von Hundelshausen, Michael Himmelsbach, Falk Hecker, Andre Mueller, and Hans-Joachim Wuensche, 2008.

$2\frac{1}{2}$ D Occupancy Grid (Elevation Map)



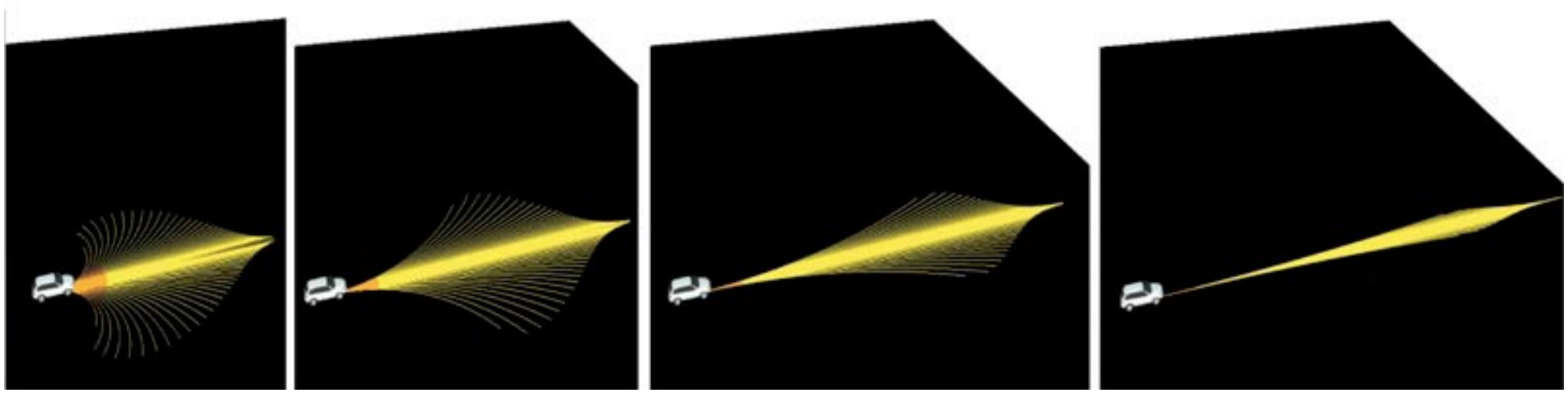




Occupancy grid value is computed to be the maximum difference of z coordinates of points in 3D space falling into that grid cell.

- Laser running at 10Hz, 100,000 3D measurements per cycle.
- No history is used in the occupancy grid data

Tentacles

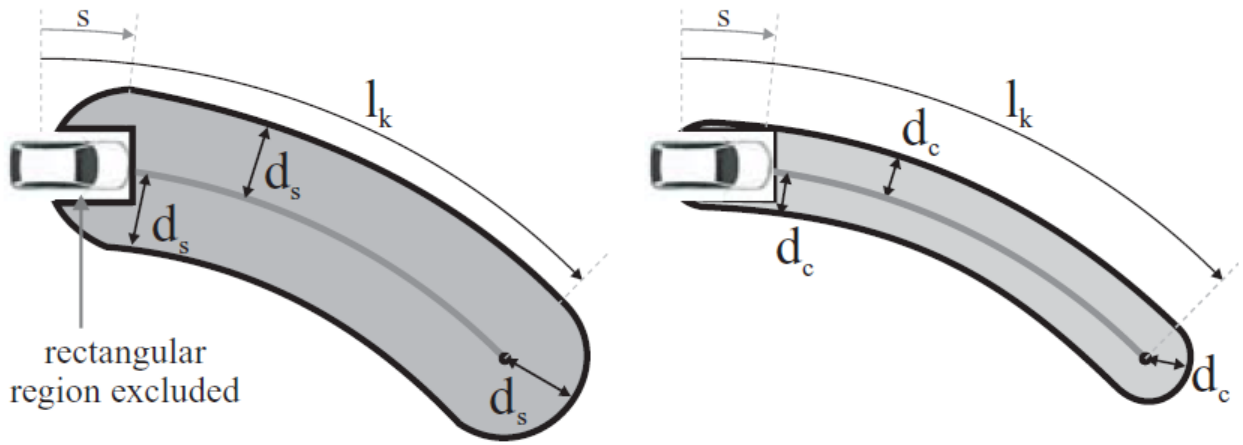


The range of speeds from 0 to 10 m/s is represented by 16 speed sets, each containing 81 tentacles. Only four of these sets are shown here. The tentacles are circular arcs and start at the center of gravity of the vehicle.

Outline

1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning





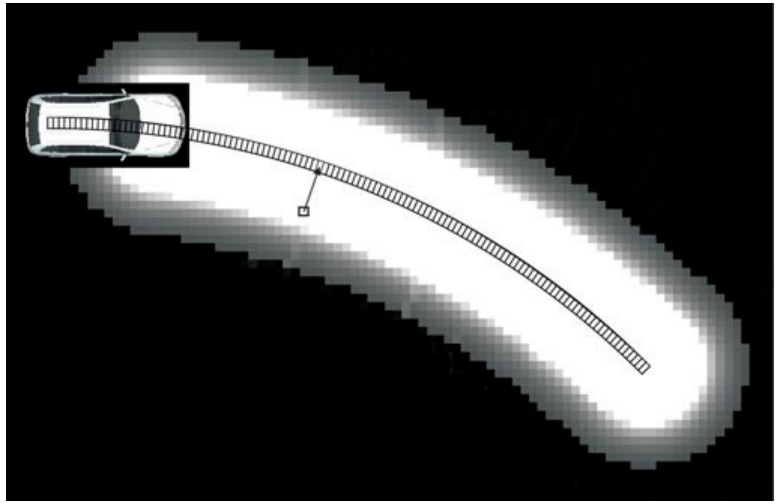
(a) Support area (b) Classification area

(a) The support area covers all cells within a distance d_s of the tentacle.

(b) The classification area is a subset of the support area covering all cells within a distance $d_c < d_s$ of the tentacle.

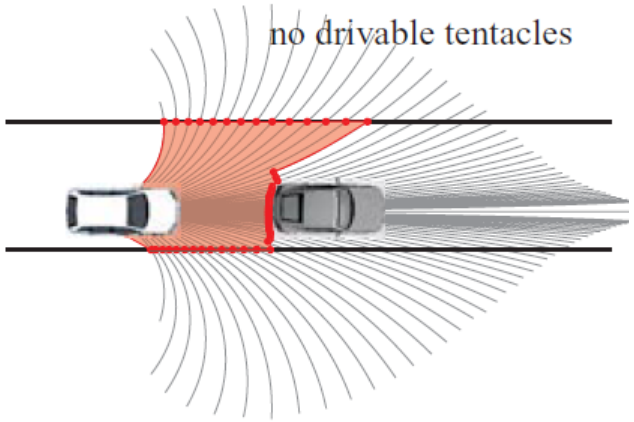
The classification area must be free for the tentacle to be drive-able. The support area is preferred to be free.

Checking for Obstacles

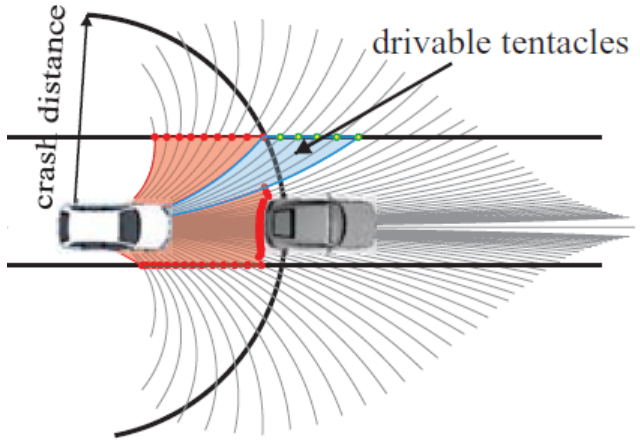


The algorithm looks at 5 consecutive cells at a time (a sliding window) and reports an obstacle if at least 2 of the cells are occupied in the occupancy grid.

The red points mark the locations along the tentacles where the vehicle would hit either the car or the road border. As can be seen, no tentacle is free of obstacles.

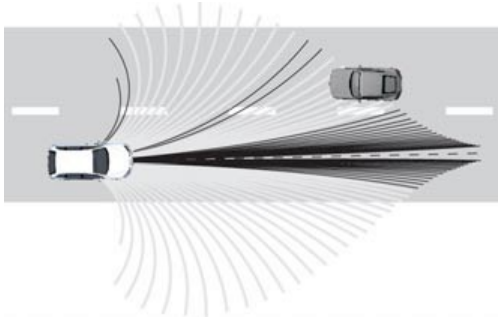


(a) By neglecting the distance to an obstacle, all tentacles would be classified non-drivable.

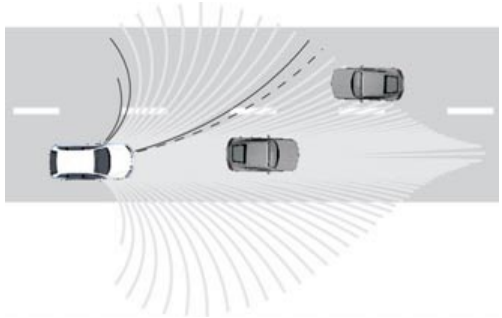


(b) shows the concept of classifying tentacles as non-drivable only in case of being occupied within a speed-dependent crash distance. In this case, some drivable tentacles remain, allowing a pass of the car.

Traffic situations



Three possible ways to go. Handled well by a heuristic that prefers tentacles that are more similar to current direction of motion.



Relying on the tentacle algorithm alone would take the car into the opposing traffic lane. Needs to be combined with additional safeguards.

Motion Planning for Cars

- The motion planning problem for cars is more complex than planning paths for wheeled mobile robots, but less complex than the general motion planning problem.
  - Local planning will do the job.
  - Car dynamics can be modeled well using geometric curves (i.e., no explicit computations of force momentum).
  - Collision checking is reduced to finding intersection of curves with obstacles.
  - A simple (2.5 D) map is a sufficiently rich representation of the environment.
  - No need for global maps, and no need for persistent representations.

Outline

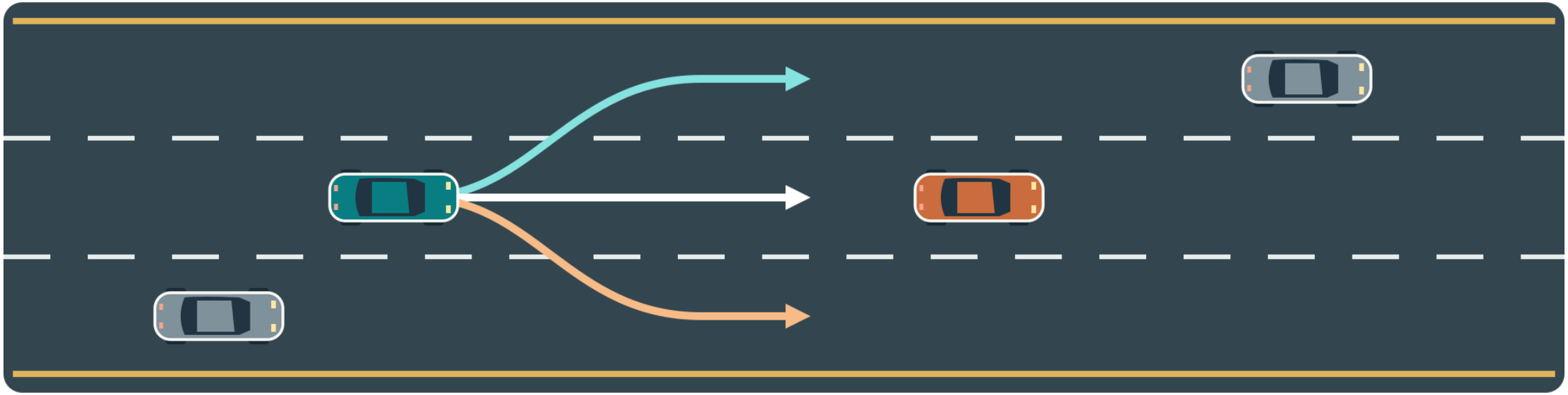
1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning

Lane choice, DQN, and REINFORCE



Which lane should the car choose?

A slower car is ahead. Another car is coming up on the right.



Deep RL learns a lane-choice policy.

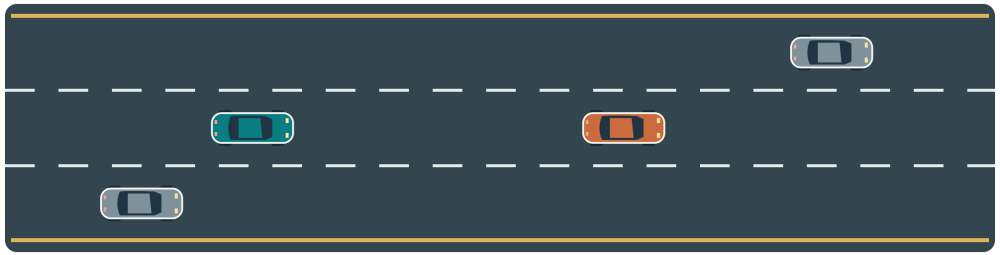
Left, stay, or right.

The local planner executes the maneuver.

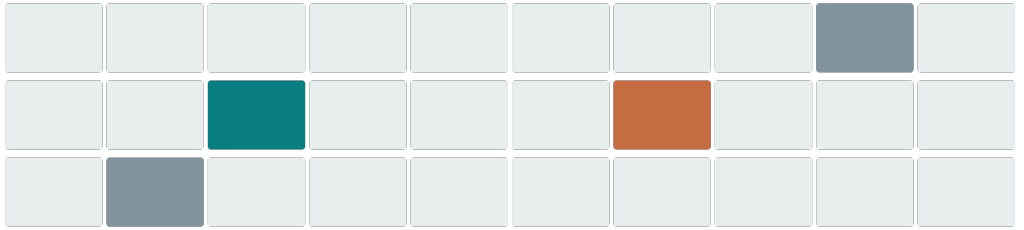
Cruise control handles speed.

How can a network see traffic?

Traffic around the ego car



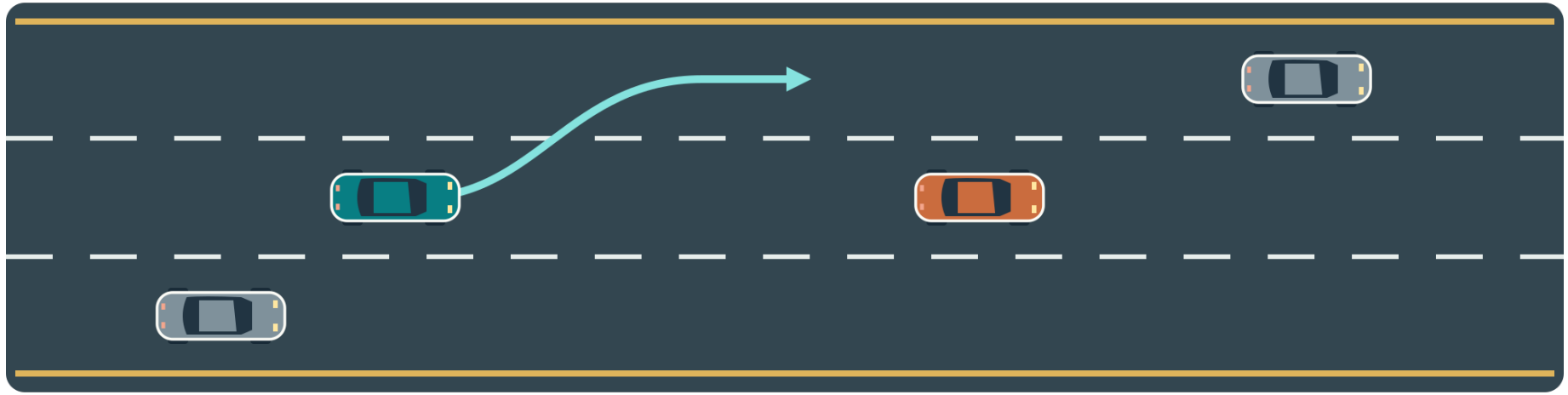
A three-lane occupancy grid



Each column covers a fixed distance along the road.

Two successive grids can reveal which gaps are closing.

What makes a lane change worthwhile?



Forward progress

Earn points for distance traveled.

A lane change

Pay a small penalty for switching lanes.

An unsafe gap

Pay a large penalty for cutting in.

A switch can cost points now and still pay off over the next few decisions.

Two ways to learn a lane choice

Deep Q-network (DQN)

What return can each maneuver earn?

Choose the maneuver with the highest predicted return.

Learn from stored transitions and the best predicted next value.

REINFORCE

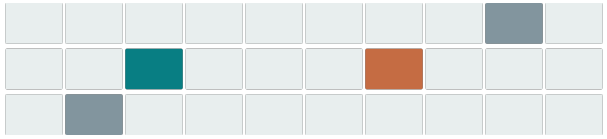
How likely is each maneuver?

Sample left, stay, or right from the policy.

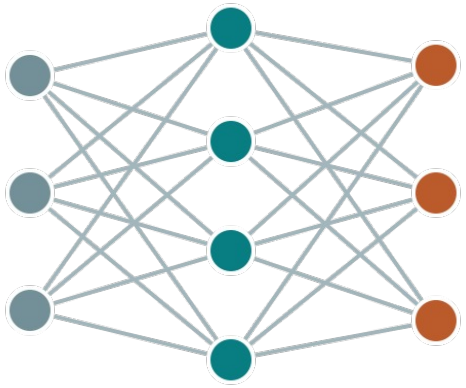
Learn from the rewards that follow each sampled action.

DQN predicts a return for each maneuver

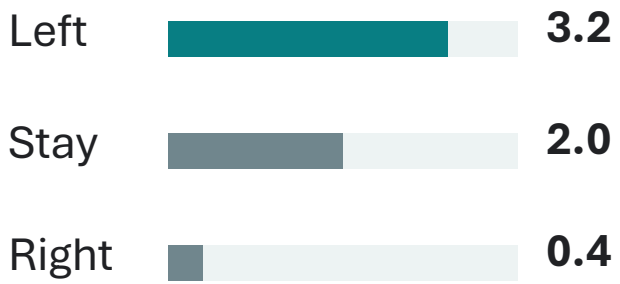
Traffic state x



Q-network



Predicted return



$$\pi^*(x) = \arg\max_a Q^*(x, a)$$

RoboticsBook · Eq. 6.68

Using the network's estimates, the greedy driver chooses left (3.2).

One transition gives DQN a training target

At the lane change

Q(x, left) = 3.2

The car moves left and earns r = 1.

One decision later

Best next Q-value = 4.0

A frozen copy of the network scores the new state.

$$y = R(x, a, x') + \gamma \max_{a'} \hat{Q}(x', a'; \theta^{\text{old}})$$

1 + 0.9 × 4.0 = 4.6

RoboticsBook · Eq. 6.72, with y for the target

Train the network to move its prediction for left from 3.2 toward 4.6.

DQN reuses past drives

Replay buffer



Each entry stores (state, action, reward, next state).

A random mini-batch



Fit the online Q-network on shuffled samples.

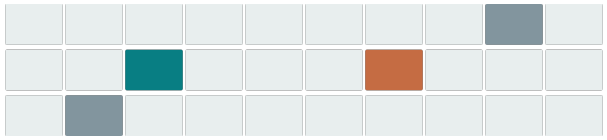
$$\mathcal{L}_{\text{DQN}}(\theta; D) = \sum_{(x, a, r, x') \in D} [y - Q(x, a; \theta)]^2$$

RoboticsBook · Eq. 6.73

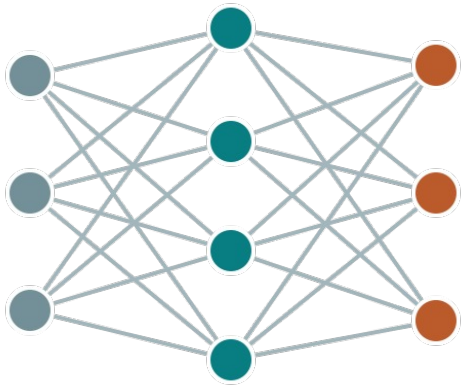
For the lane change, the squared error is $(4.6 - 3.2)^2 = 1.96$. Update the online network to reduce it.

REINFORCE learns a probability for each maneuver

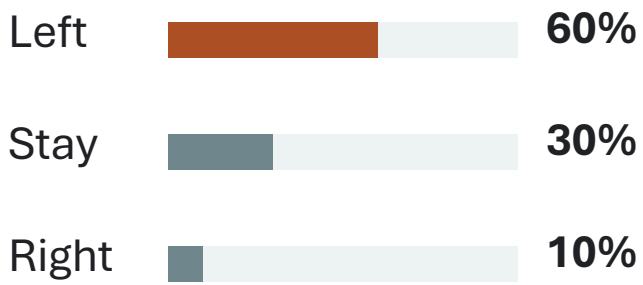
Same traffic state



Policy network



Action probability



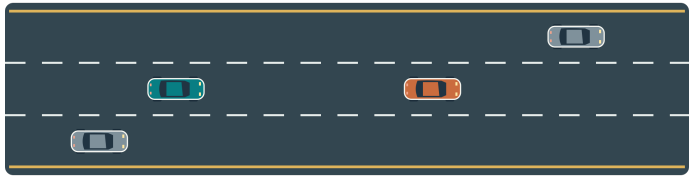
$$\pi(a \mid x; \theta) = \frac{e^{z_a}}{\sum_j e^{z_j}}$$

Softmax converts the logits z into probabilities. In this state, the policy samples left 60% of the time.

RoboticsBook · Eqs. 6.74 and 6.81

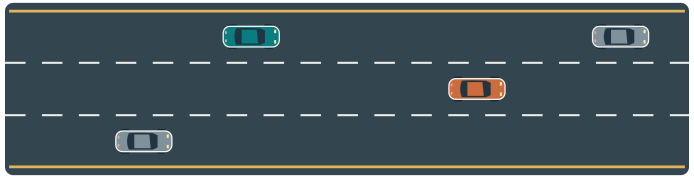
What return did the lane change earn?

1. Change left



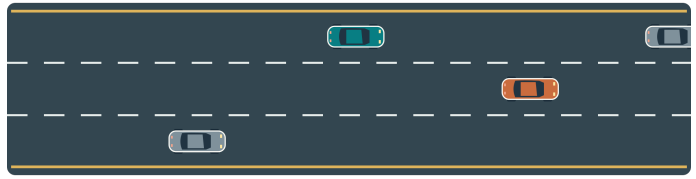
Reward +1

2. Stay left



Reward +1

3. Stay left



Reward +3

$$\hat{Q}(x_{it}, a_{it}) = \sum_{k=t}^{H} \gamma^{k-t} r_{ik}$$

$1 + 0.9 \times 1 + 0.9^2 \times 3 = 4.33$

RoboticsBook · Eq. 6.78, with r for the observed reward

Rollout i supplies a sample of the return from decision t onward.

The return weights the action's log probability

Policy loss across rollouts

$$\mathcal{L}(\theta) = -\sum_{i}\sum_{t=1}^{H} \hat{Q}(x_{it}, a_{it}) \log \pi(a_{it} \mid x_{it}; \theta)$$

RoboticsBook · Eq. 6.79

For the lane change, the loss includes −4.33 log π(left | x; θ).

Policy gradient

$$\nabla_\theta \mathcal{L}(\theta) = -\sum_{i}\sum_{t=1}^{H} \hat{Q}(x_{it}, a_{it}) \nabla_\theta \log \pi(a_{it} \mid x_{it}; \theta)$$

RoboticsBook · Eq. 6.80

Take a gradient descent step, then collect new drives with the updated policy.

A baseline tells REINFORCE how well an action did

Two rollouts start in the same traffic state. The baseline return is 3.00.

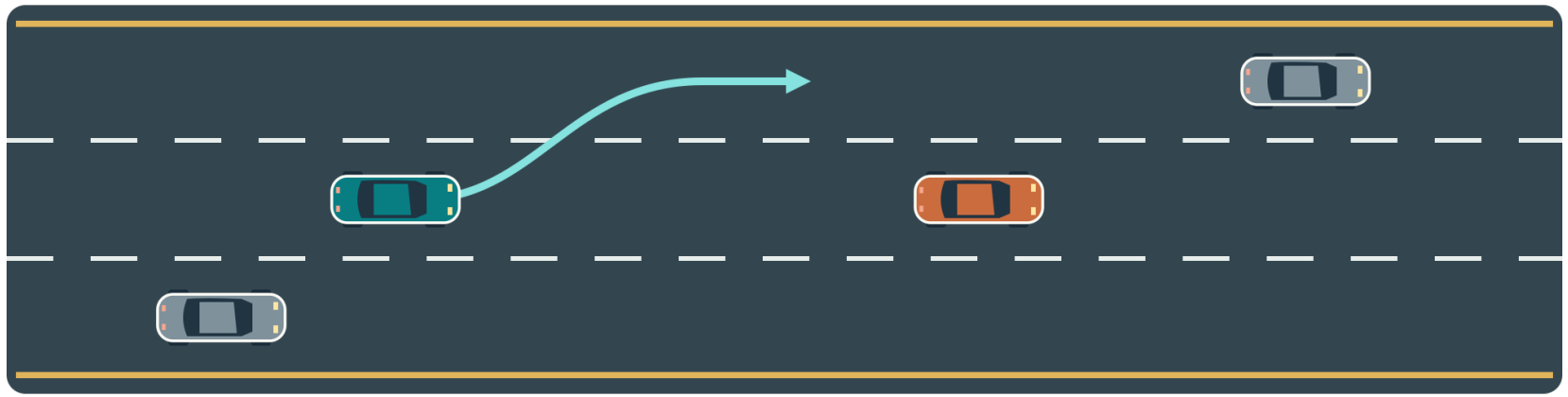
	Advantage	
Return 4.33	**+1.33**	Increase the probability of the sampled action.
Return 2.00	**−1.00**	Decrease the probability of the sampled action.

$$\Delta\theta \propto (G_t - b(x_t)) \nabla_\theta \log \pi_\theta(a_t \mid x_t)$$

RoboticsBook · Eq. 6.80, adapted with a state baseline

G_t is the sampled return and $b(x_t)$ is the expected return from that state.

How do planning and learning share the wheel?



Policy

Which lane looks best over time?

Trajectory planner

Which steering inputs reach that lane?

Collision checker

Will the path keep enough clearance?

If the left lane scores highest but the gap closes, what should the car do?

Summary

1. Planning for Mobile Robots
2. Roadmaps and Visibility Graphs
3. Voronoi Diagrams and Brushfire
4. Sampling-Based Planning
5. Driving with Tentacles
6. Collision Checking and Traffic
7. Deep Reinforcement Learning

