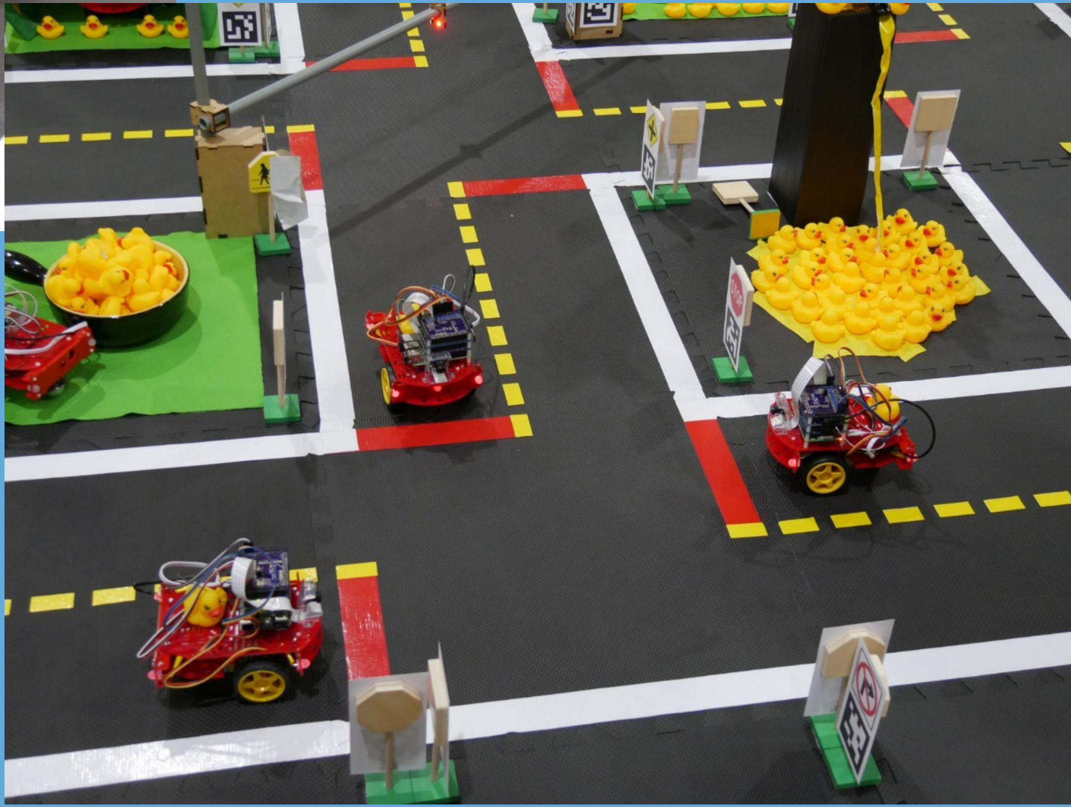
CS 3630
L25: Motion Planning for Cars



With lots of slides and ideas from:

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

Path Planning – a quick review

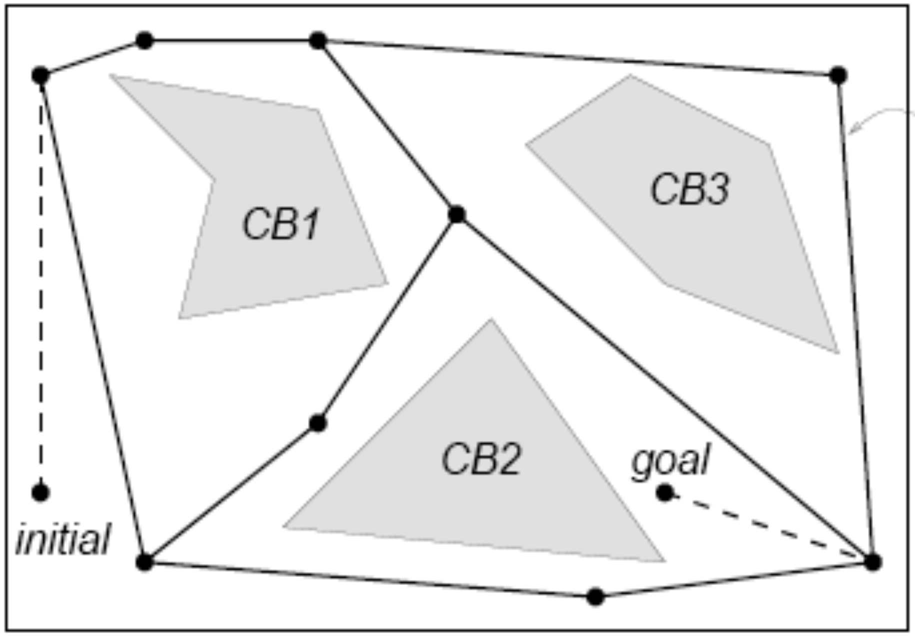
- path planning = finding a **collision-free path** from a start to a goal.
- The best algorithms have time complexity that increases **exponentially** in the dimension of the configuration space.
    - low-dimensional configuration spaces: often find exact solutions.
    - high-dimensional configuration 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, $x \in \Re^2$.

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

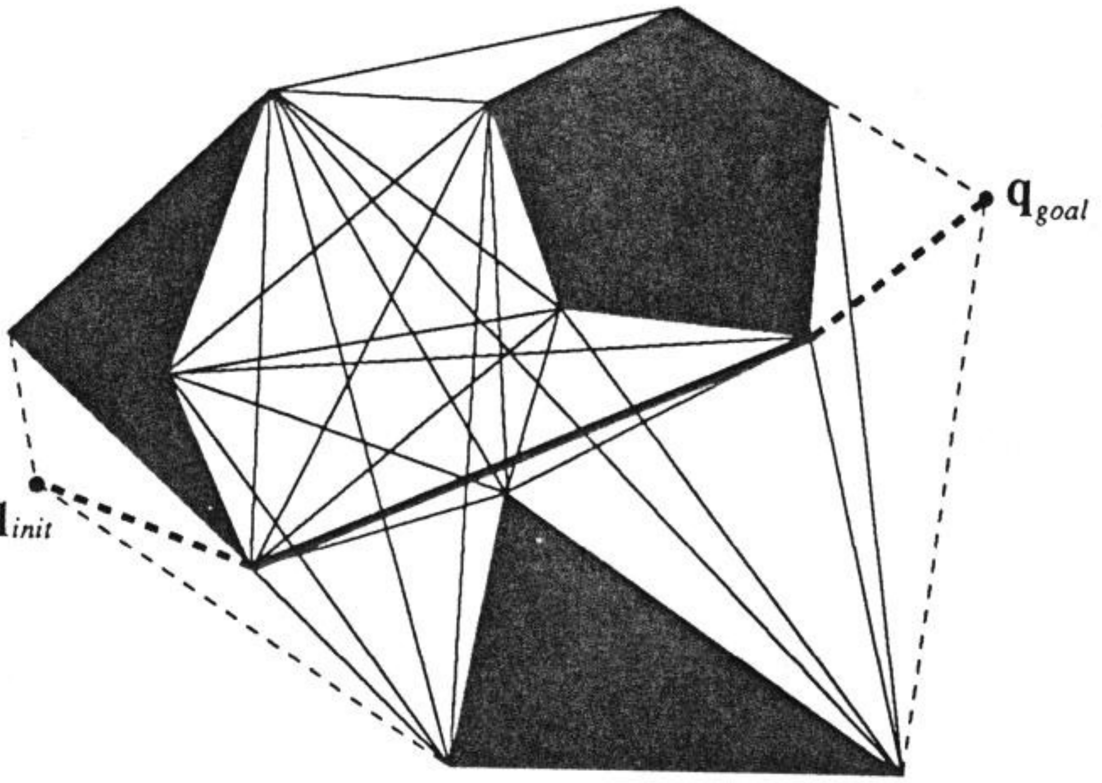
The Visibility Graph

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

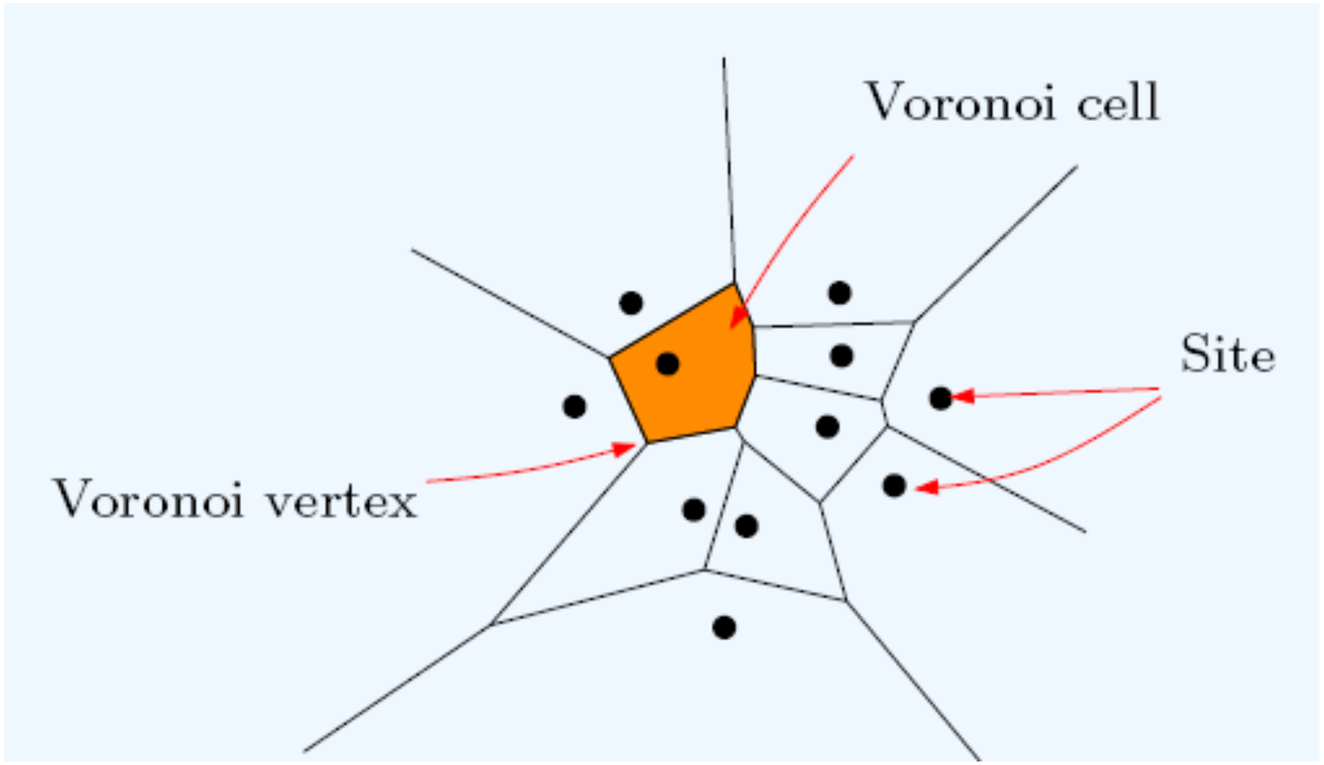


- If there is there a path, then the *shortest* path is in the visibility graph
- If we include the start and goal nodes, they are automatically connected
- Algorithms for constructing them can be efficient
  - ➢ $O(n^3)$ brute force (i.e., naïve)
  - ➢ $O(n^2 \log n)$ if clever

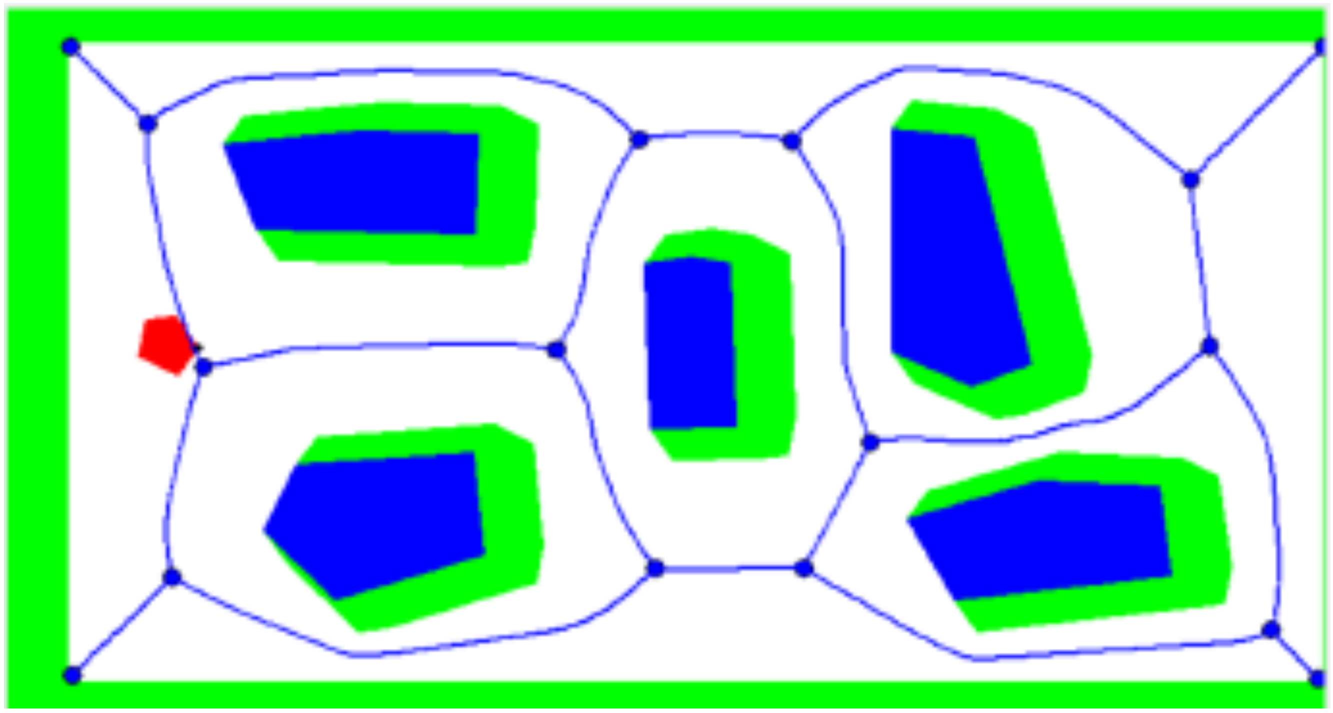
Safe Paths that Have Large Clearance to Obstacles

The Generalized Voronoi Diagram

Voronoi Diagrams



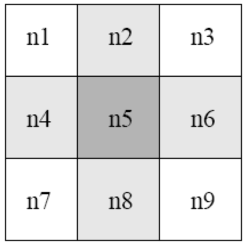
Generalized Voronoi Diagrams



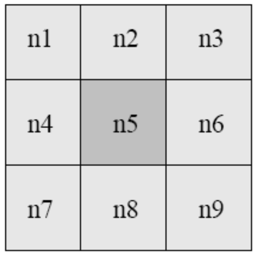
A Discrete Version of the Generalized Voronoi Diagram

- 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



4



8

Brushfire Algorithm

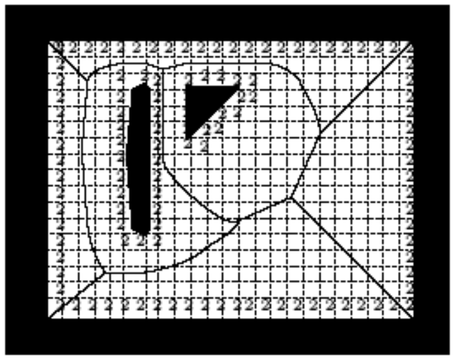
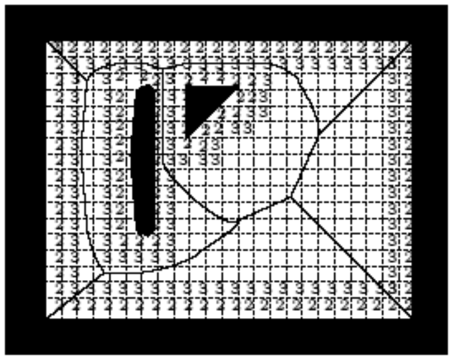
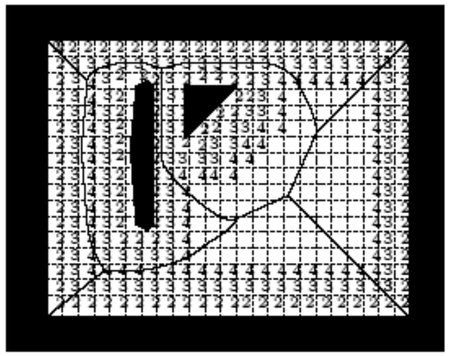
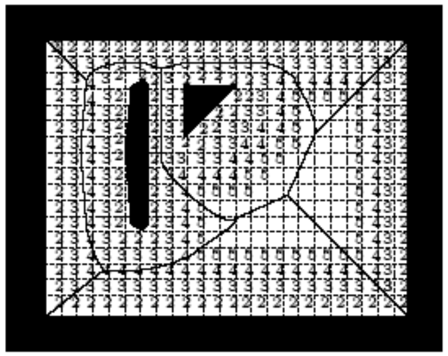
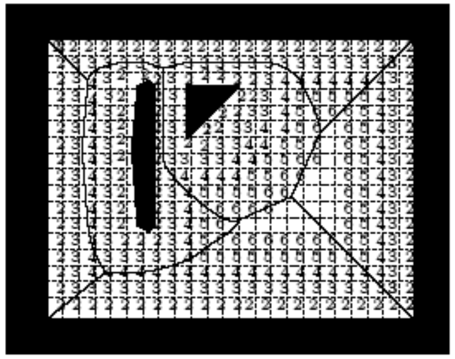
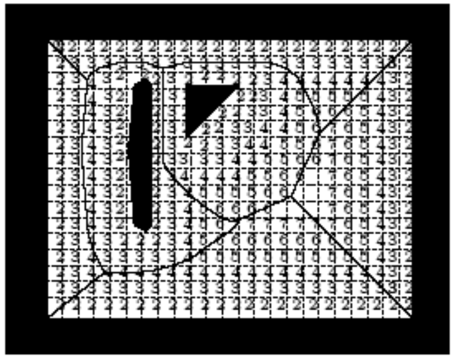
- Initially: create a queue L of pixels on the boundary of all obstacles, set $d(t) = 0$ for each non-boundary grid cell t

- 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

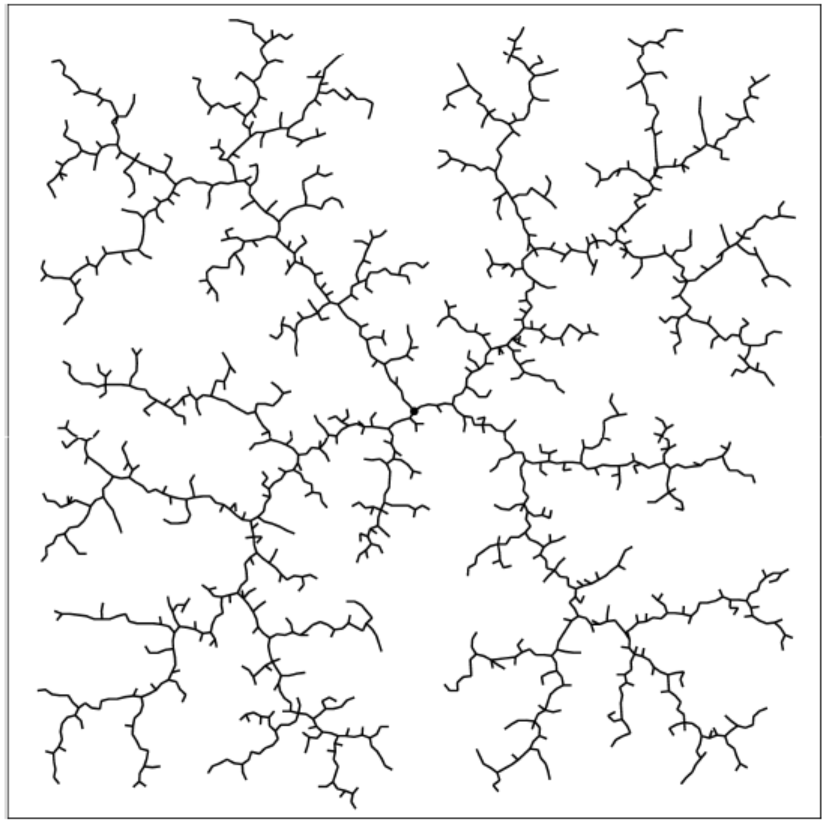
Note that the curves here are not at all perfect…

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.

Sampling-Based Methods for Path Planning

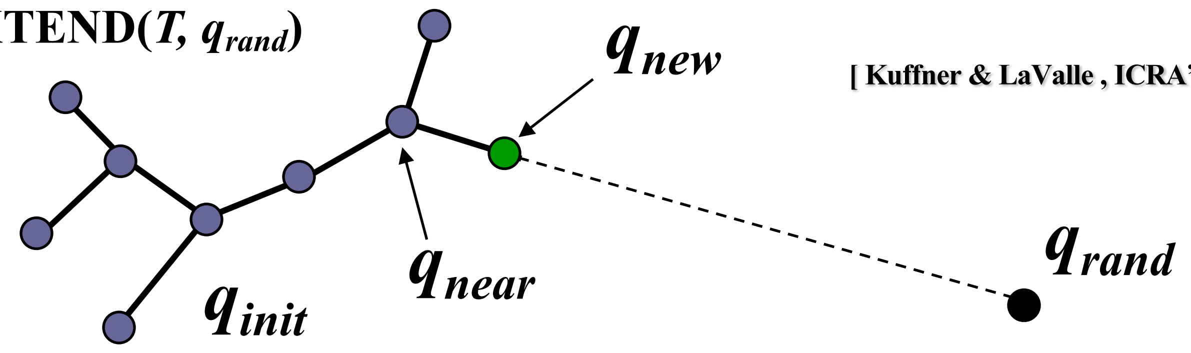
Rapidly-Exploring Random Tree



Path Planning with RRTs

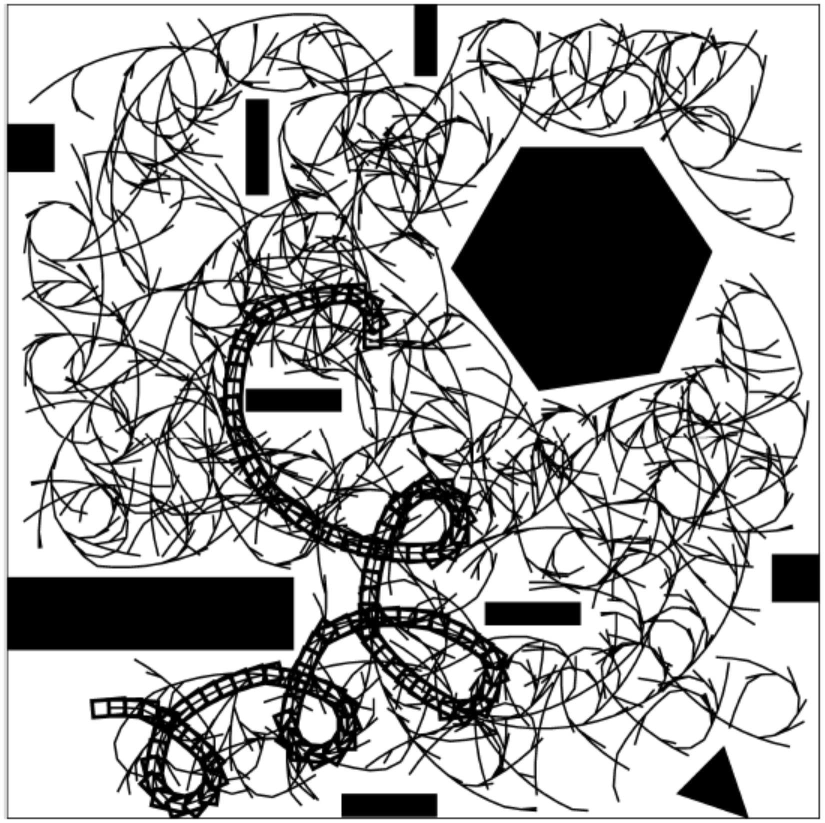
```
BUILD_RRT ($q_{init}$)  {
   $T.init(q_{init})$;
   for $k$ =  1 to K do
      $q_{rand}$ = RANDOM_CONFIG();
      EXTEND($T$, $q_{rand}$)
}
```

EXTEND(T, q_{rand})



[Kuffner & LaValle , ICRA'00]

Left-turn only forward car

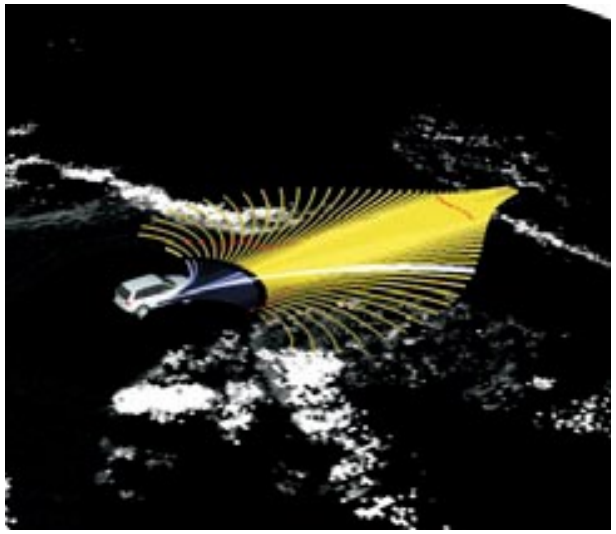


Application to Cars?

- 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!

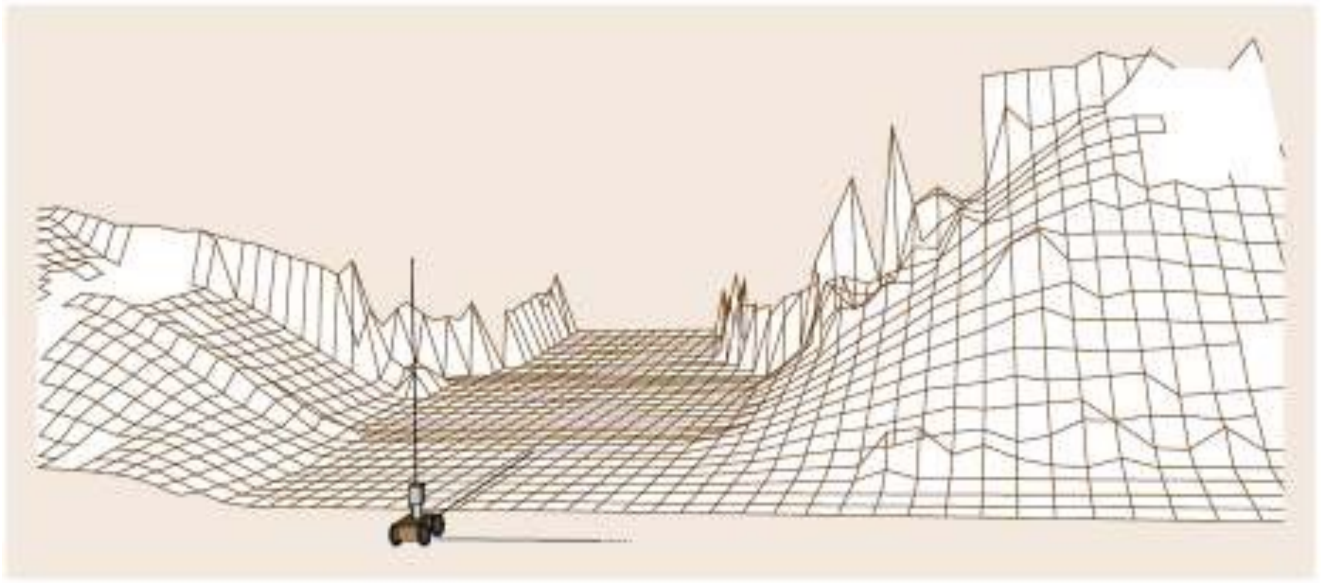
A modification to path planning…

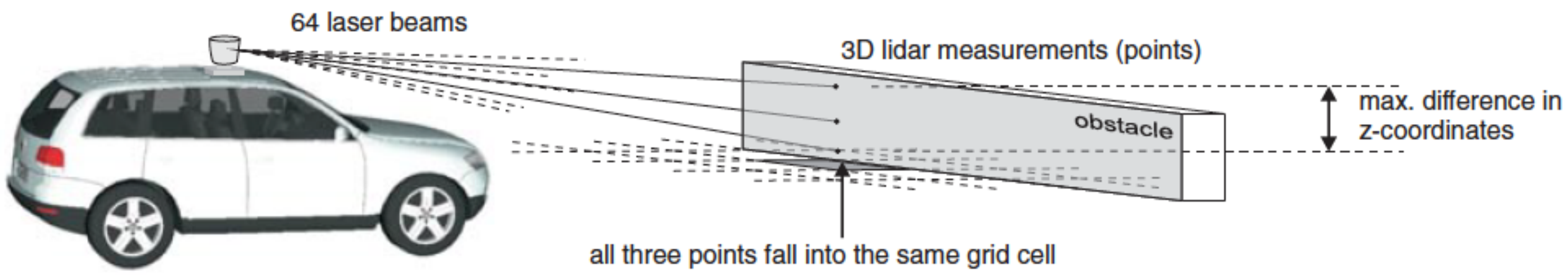
- Driving with tentacles

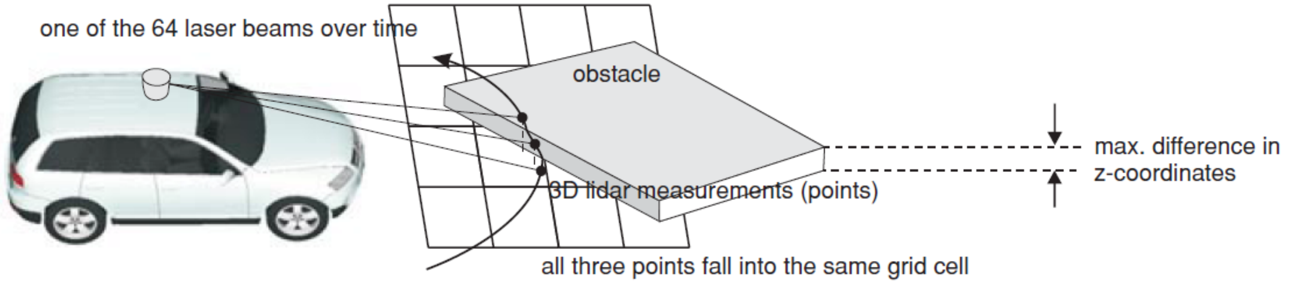


Felix von Hundelshausen, Michael Himmelsbach, Falk Hecker, Andre Mueller, and Hans-Joachim Wuensche, 2008.

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



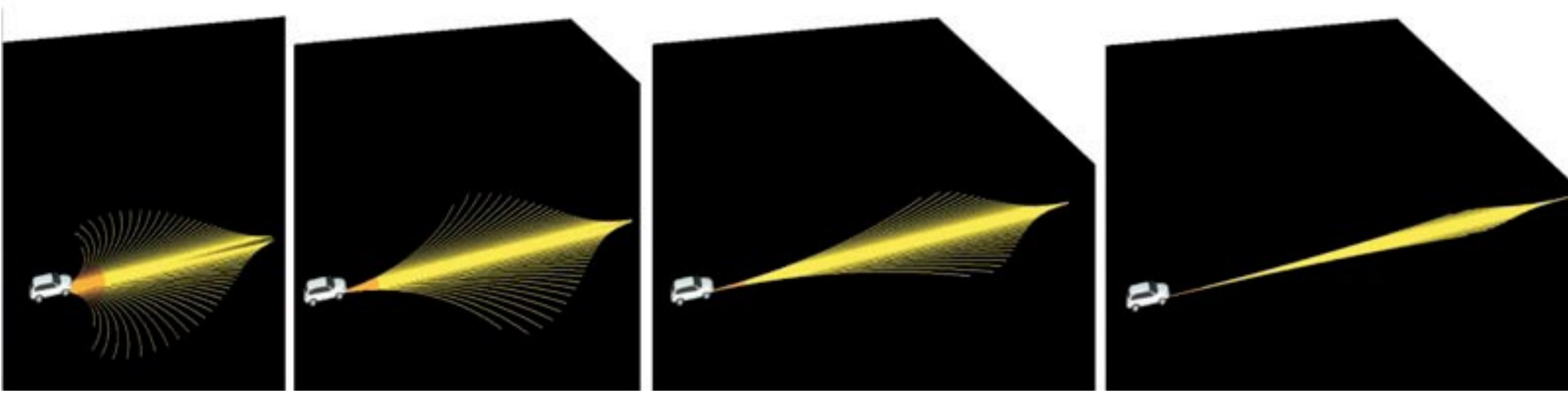

64 laser beams
3D lidar measurements (points)
obstacle
max. difference in z-coordinates
all three points fall into the same grid cell


one of the 64 laser beams over time
obstacle
max. difference in z-coordinates
3D lidar measurements (points)
all three points fall into the same grid cell

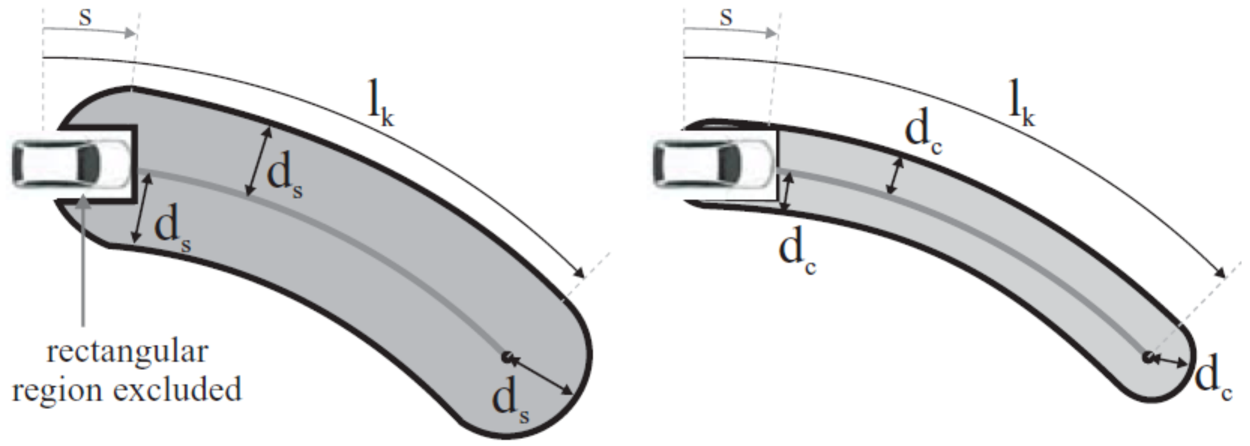
Occupancy grid value is computed to be the maximum difference of z coordinates of points in 3D space falling into that grid cell.

- 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.



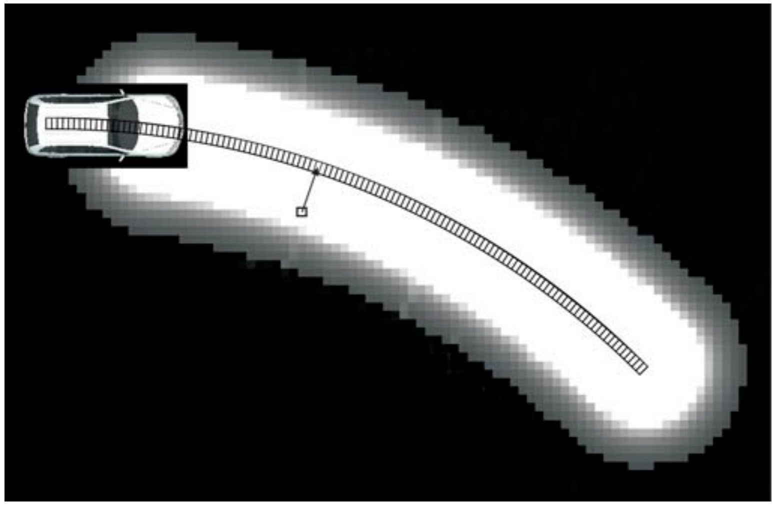
(a) Support area (b) Classification area

(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.

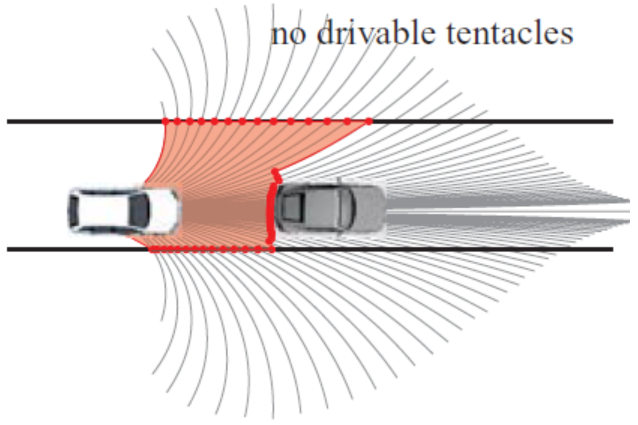
The classification area must be free for the tentacle to be drive-able. The support area is preferred to be free.

Checking for Obstacles

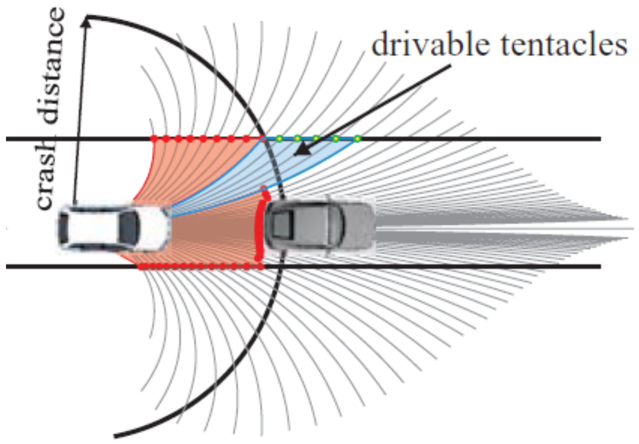


The algorithm looks at 5 consecutive cells at a time (a sliding window) and reports an obstacle if at least 2 of the cells are occupied in the occupancy grid.

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.

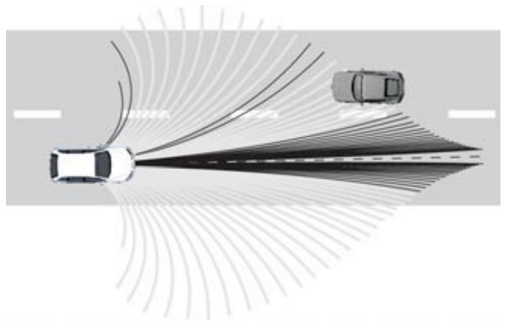


(a) By neglecting the distance to an obstacle, all tentacles would be classified non-drivable.

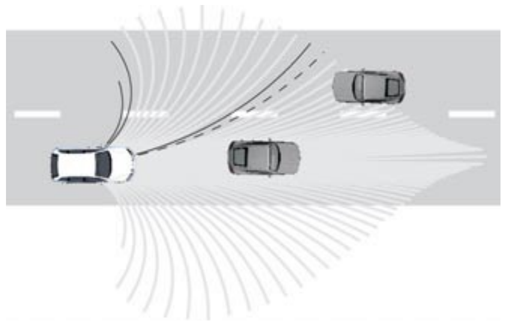


(b) shows the concept of classifying tentacles as non-drivable only in case of being occupied within a speed-dependent crash distance. In this case, some drivable tentacles remain, allowing a pass of the car.

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.