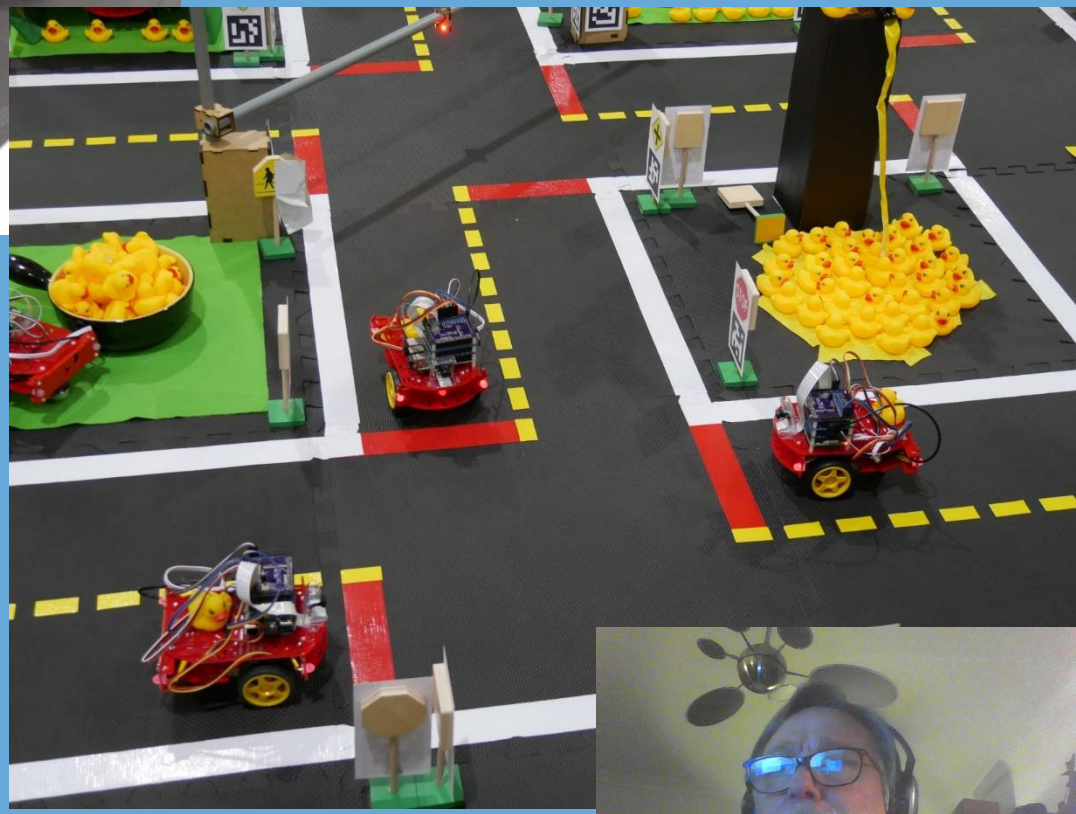
CS 3630
Motion Planning for Cars



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

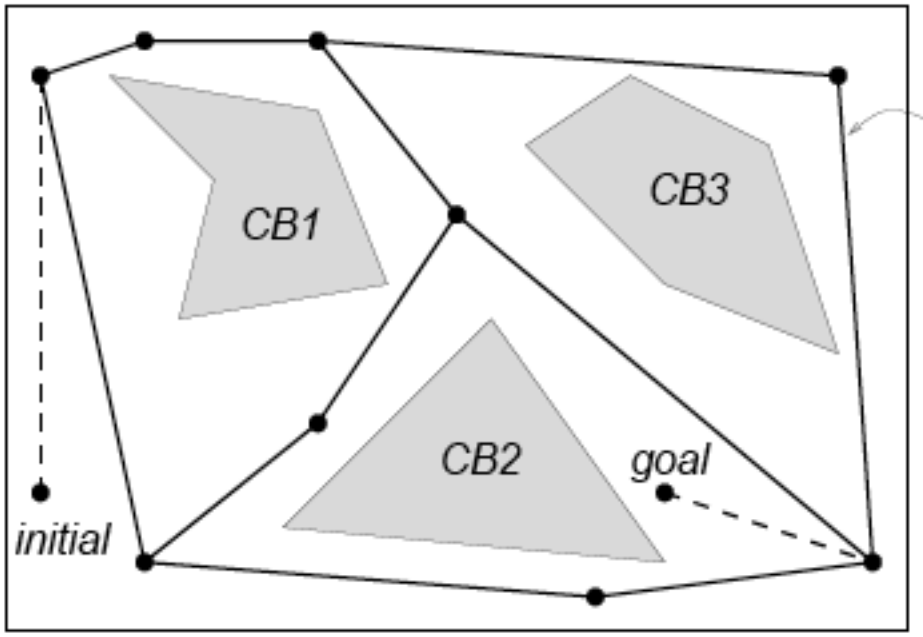
- In robotics, the path planning problem consists of finding a collision-free path from a start configuration to a goal configuration.
- The best algorithms known for this problem have time complexity that increases exponentially in the dimension of the configuration space.
  - For robots with low-dimensional configuration spaces, we can often find exact solutions.
  - For robots with high-dimensional configuration spaces, we generally settle for good approximate solutions, or notions like *probabilistic completeness*.
- The canonical path planning problem deals only with geometry; dynamics (i.e., properties such as force or momentum) are not considered. Considering dynamics can significantly increase the complexity of planning algorithms.
- 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

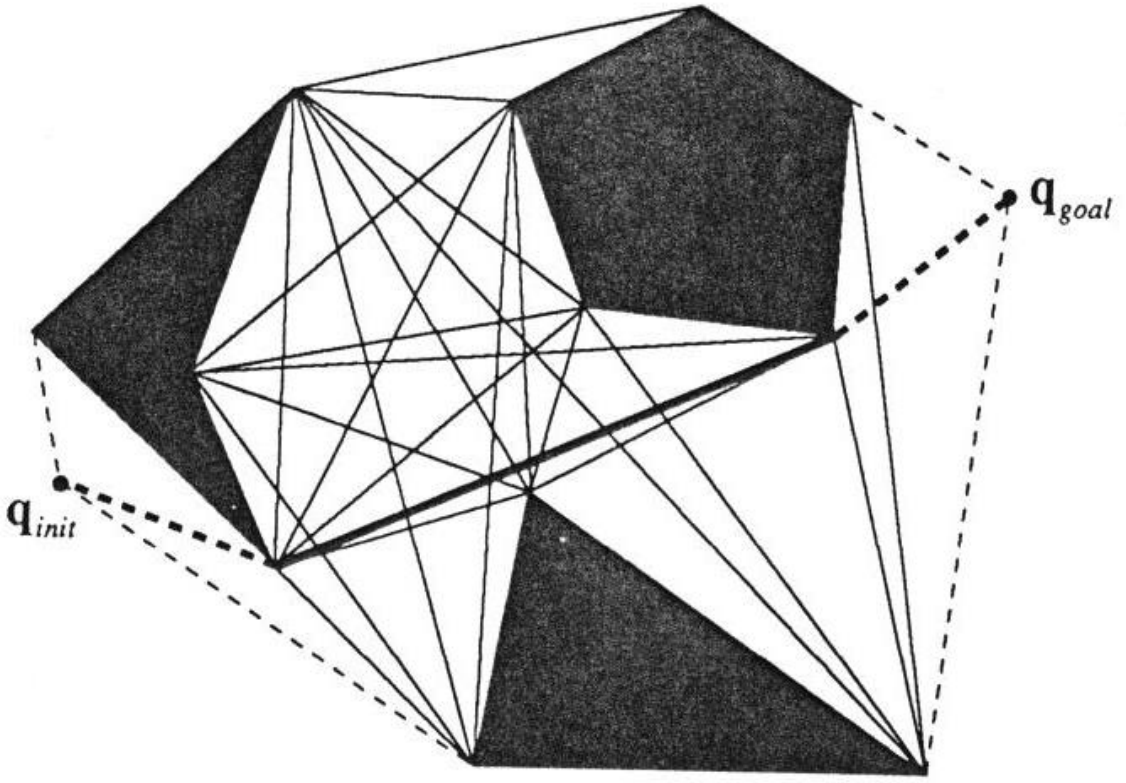
Shortest, But Possibly Dangerous Paths

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

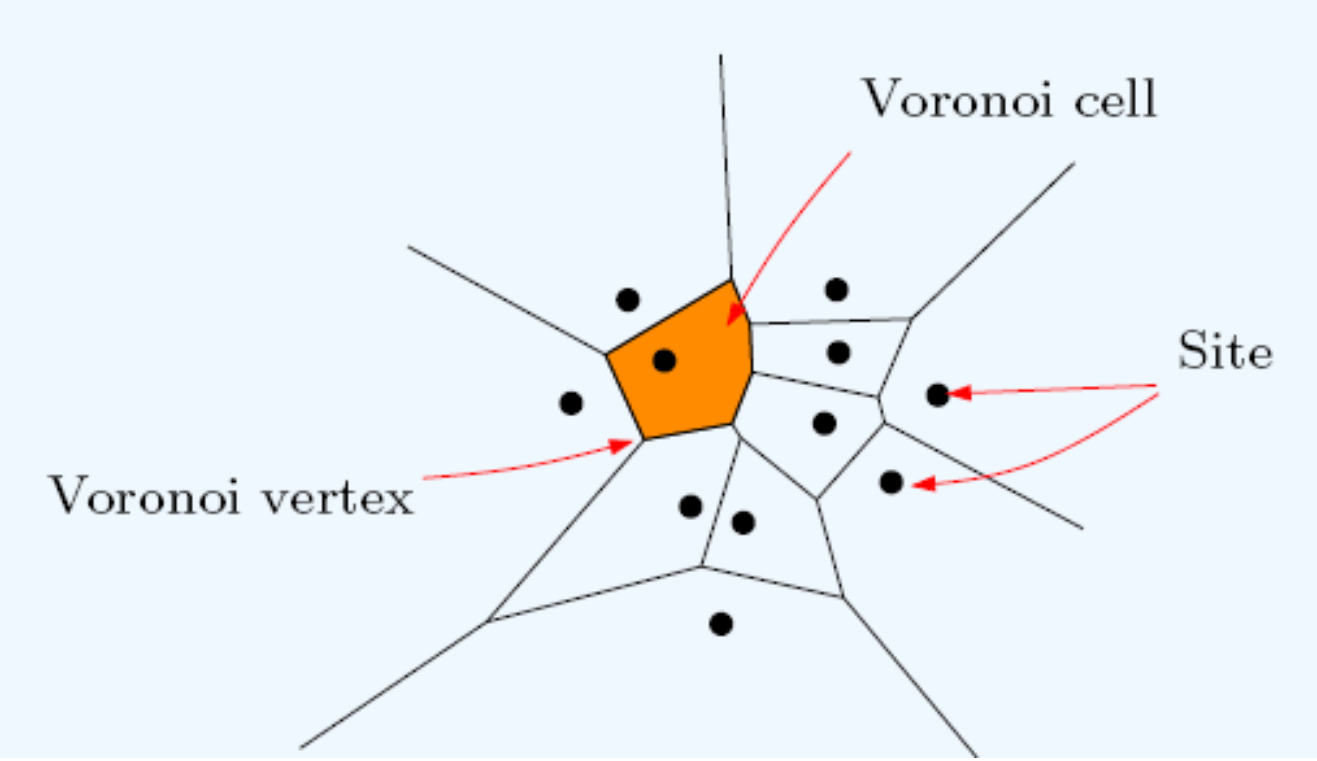


- If there is there a path, then the *shortest* path is in the visibility graph
- If we include the start and goal nodes, they are automatically connected
- Algorithms for constructing them can be efficient
  - $O(n^3)$ brute force (i.e., naïve)
  - $O(n^2 \log n)$ if clever

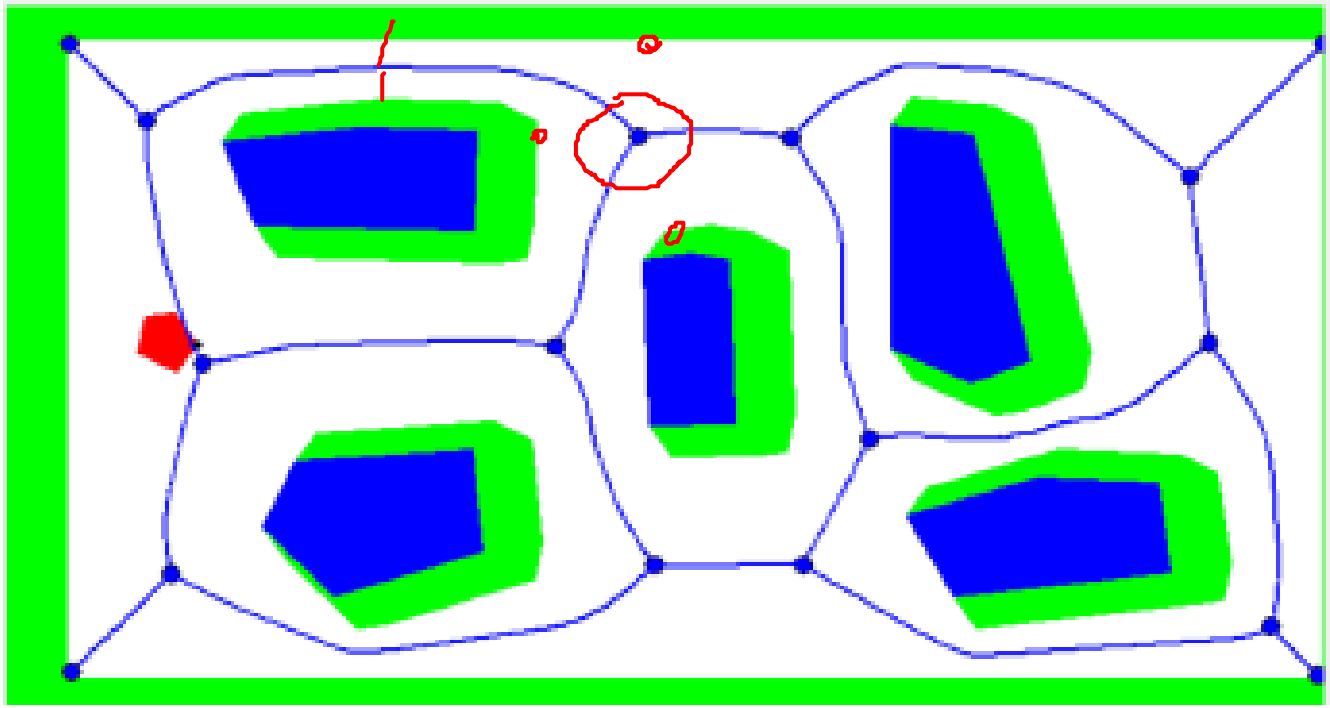
Safe Paths that Have Large Clearance to Obstacles

The Generalized Voronoi Diagram

Voronoi Diagrams

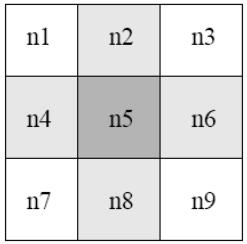


Generalized Voronoi Diagrams

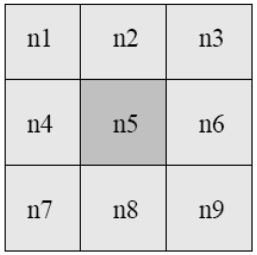


A Discrete Version of the Generalized Voronoi Diagram

- 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

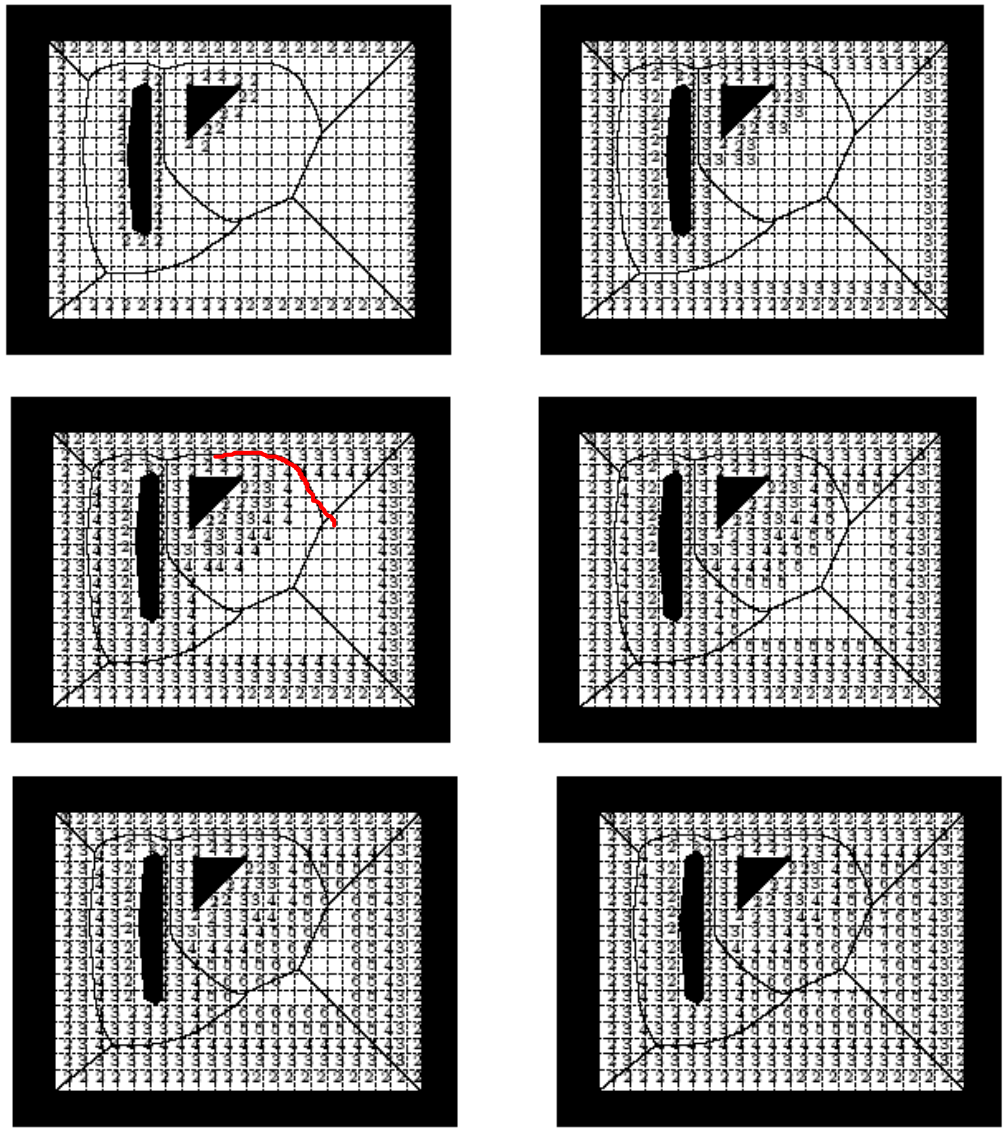
- Initially: create a queue L of pixels on the boundary of all obstacles, set $d(t) = 0$ for each non-boundary grid cell t

- 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…

Path Planning for Large Empty Spaces

Cell Decomposition

Cell Decomposition

- Don’t explicitly build a 1-D Roadmap.
- The “Roadmap” corresponds to the adjacency graph of the cellular decomposition.
- Nodes in the adjacency graph correspond to free cells.
- Arcs in the adjacency graph connect nodes that correspond to adjacent cells.

Definition

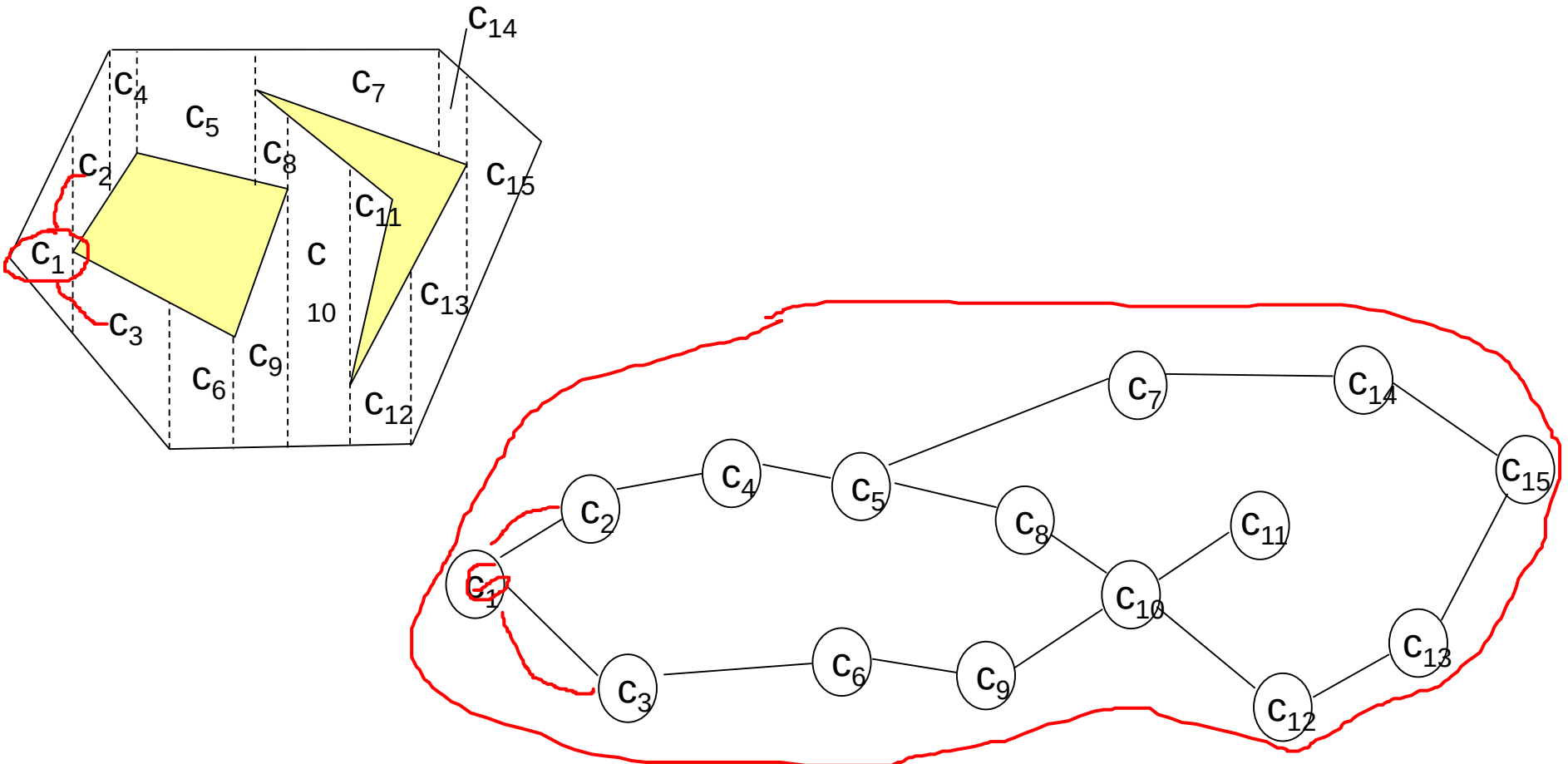
Exact Cellular Decomposition

- ν_i is a cell
- $\text{int}(\nu_i) \cap \text{int}(\nu_j) = \emptyset$ if and only if $i \neq j$
- $Q_{free} \cap (\text{cl}(\nu_i) \cap \text{cl}(\nu_j)) \neq \emptyset$ if ν_i and ν_j are adjacent cells
- $Q_{free} = \cup_i (\nu_i)$

Adjacency Graph

- Node correspond to a cell
- Edge connects nodes of adjacent cells

- Two cells are *adjacent* if they share a common boundary



Path Planning

- Path Planning in two steps:
  - Planner determines cells that contain the start and goal
  - Planner searches for a path within adjacency graph

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 are generally not useful when planning motions for cars.

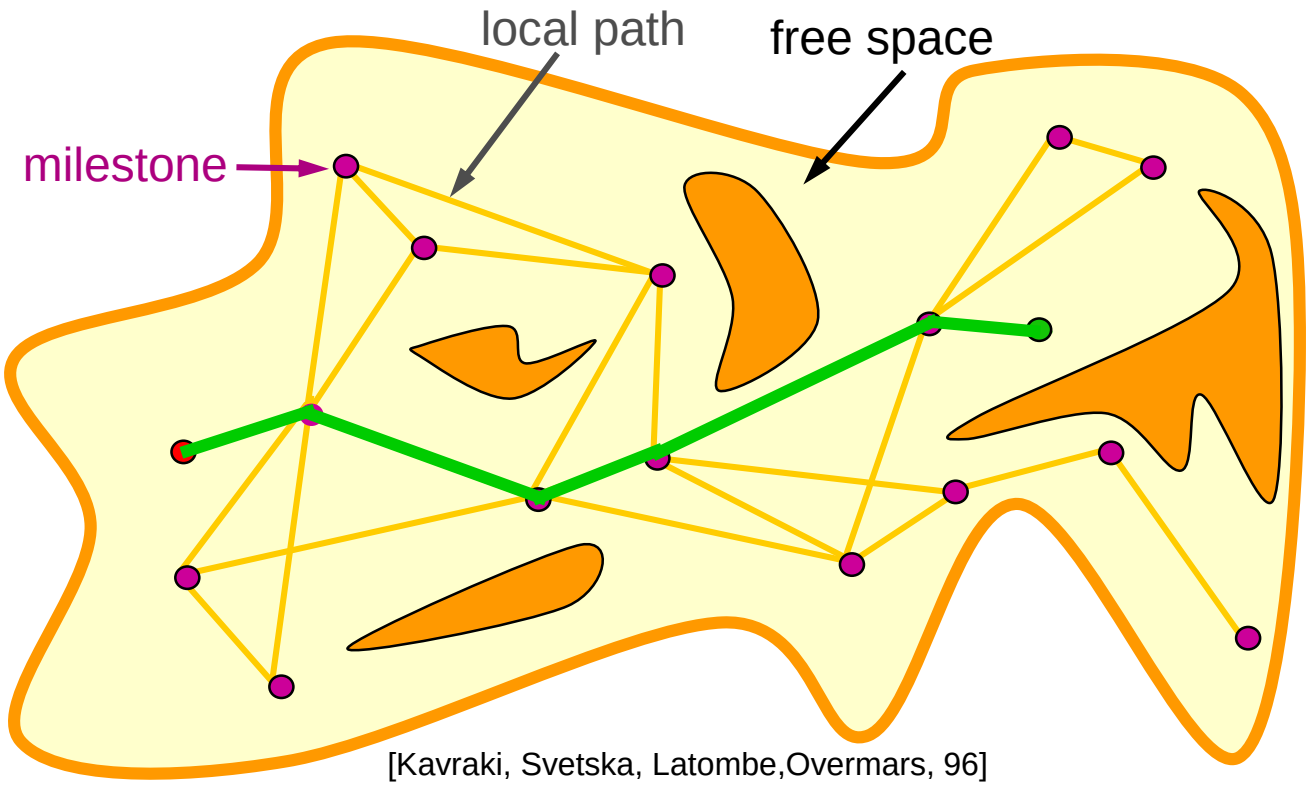
Sampling-Based Methods for Path Planning

Completeness

- Complete algorithm → Slow
  - A **complete** algorithm finds a path if one exists and reports no otherwise in finite time.
  - Example: visibility graph for 2D problems (translation in the plane) and polygonal robot and obstacles
- Heuristic algorithm → Unreliable
  - Example: potential field (we'll see it soon)

- **Probabilistic completeness**
  - Intuition: If there is a solution path, the algorithm will find it with high probability.

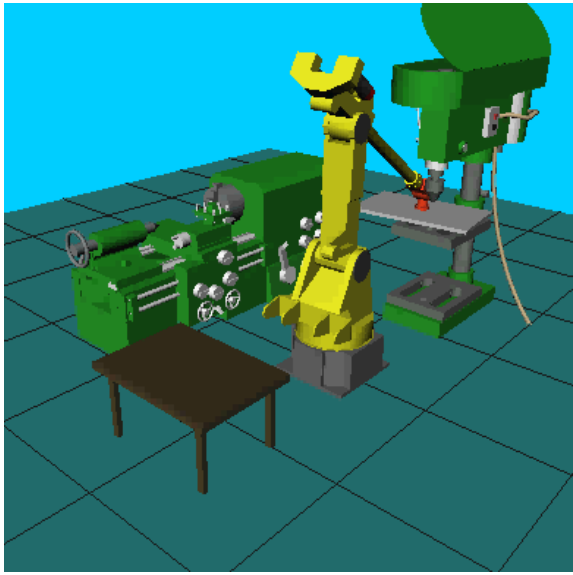
Probabilistic Roadmap (PRM): multiple queries



[Kavraki, Svetska, Latombe,Overmars, 96]

Assumptions

- Static obstacles
- Many queries to be processed in the same environment
- Examples
  - Navigation in static virtual environments
  - Robot manipulator arm in a workcell
- Advantages:
  - Amortize the cost of planning over many problems
  - Probabilistically complete



Overview

- Precomputation: roadmap construction
  - Uniform sampling
  - Resampling (expansion)
- Query processing

Terminology

- The graph G is called a **probabilistic roadmap**.
- The nodes in G are called **milestones**.

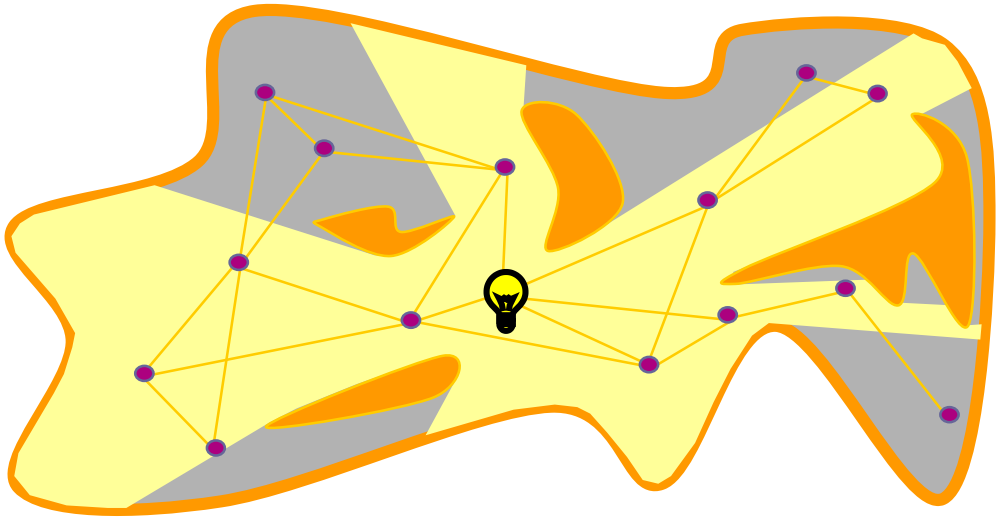
Uniform sampling

Input: geometry of the moving object & obstacles
Output: roadmap G = (V, E)

1: V ← ∅ and E ← ∅.
2: **repeat**
3: q ← a configuration sampled uniformly at random from C.
4: **if CLEAR**(q) **then**
5: Add q to V.
6: N_q ← a set of nodes in V that are close to q.
6: **for** each q′ ∈ N_q, in order of increasing d(q,q′)
7: **if LINK**(q′,q) **then**
8: Add an edge between q and q′ to E.

Why does it work? Intuition

- A small number of milestones **almost** “cover” the **entire** configuration space.

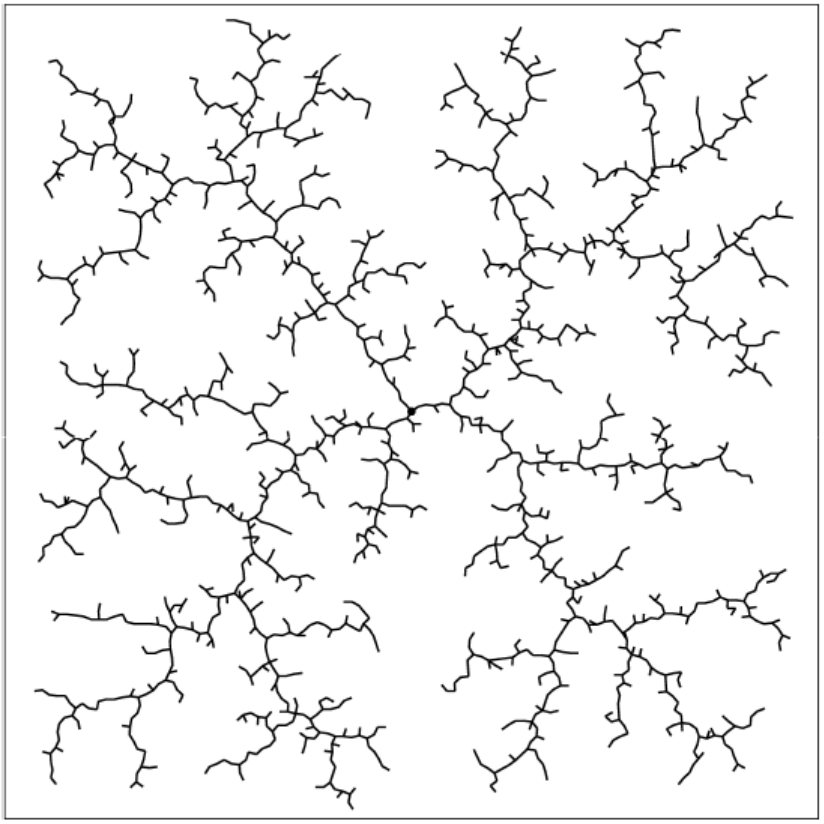


- Rigorous definitions and exist (of course!)

Rapidly-Exploring Random Tree (RRT)

- Searches for a path from the initial configuration to the goal configuration by expanding a search tree
- For each step,
  - The algorithm samples a target configuration and expands the tree towards it.
  - The sample can either be a random configuration or the goal configuration itself, depends on the probability value defined by the user.

Rapidly-Exploring Random Tree

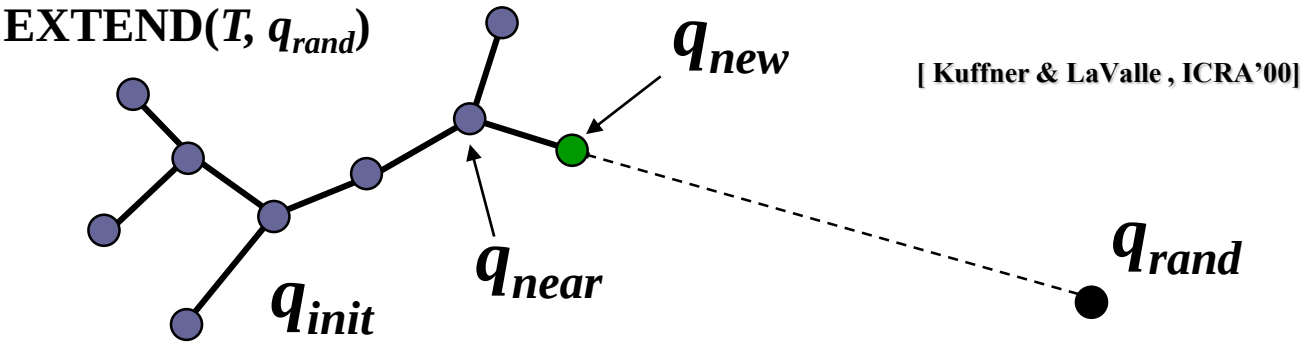


The Basic Idea: Iteratively expand the tree

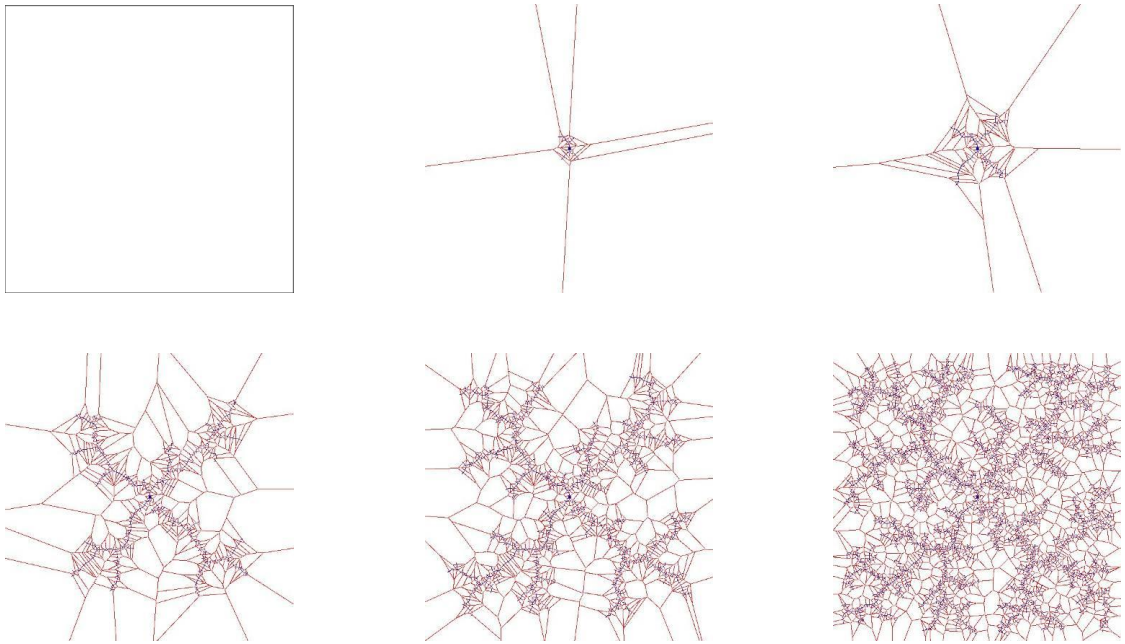
- Denote by T_k the tree at iteration k
- Randomly choose a configuration q_{rand}
- Choose $q_{near} = \arg\min_{q \in T_k} d(q, q_{rand})$
  - q_{near} is the nearest existing node in the tree to q_{rand}
- Create a new node, q_{new} by taking a small step from q_{near} toward q_{rand}

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)
}
```

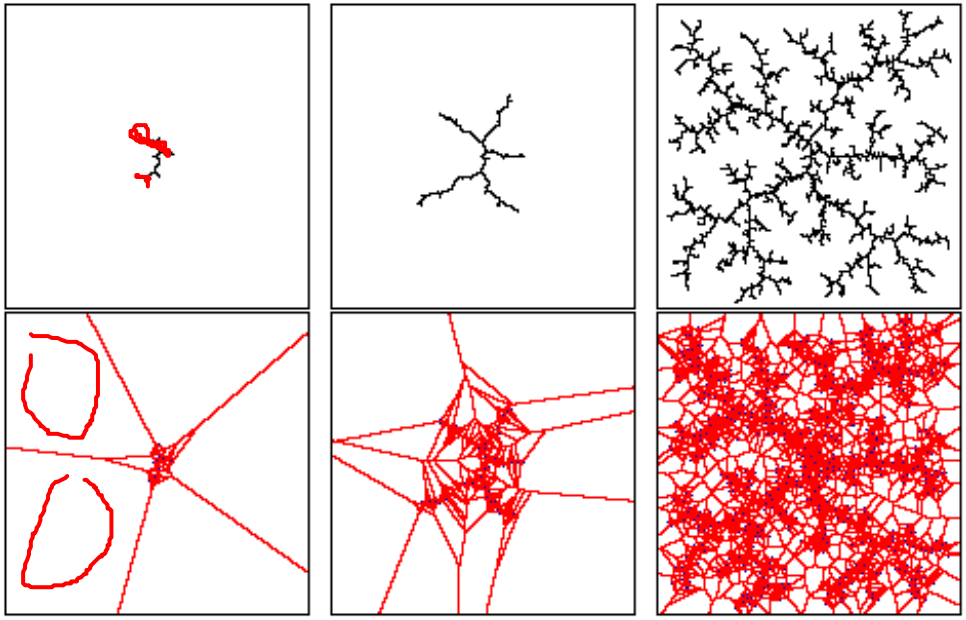


RRTs and
Bias toward large Voronoi regions



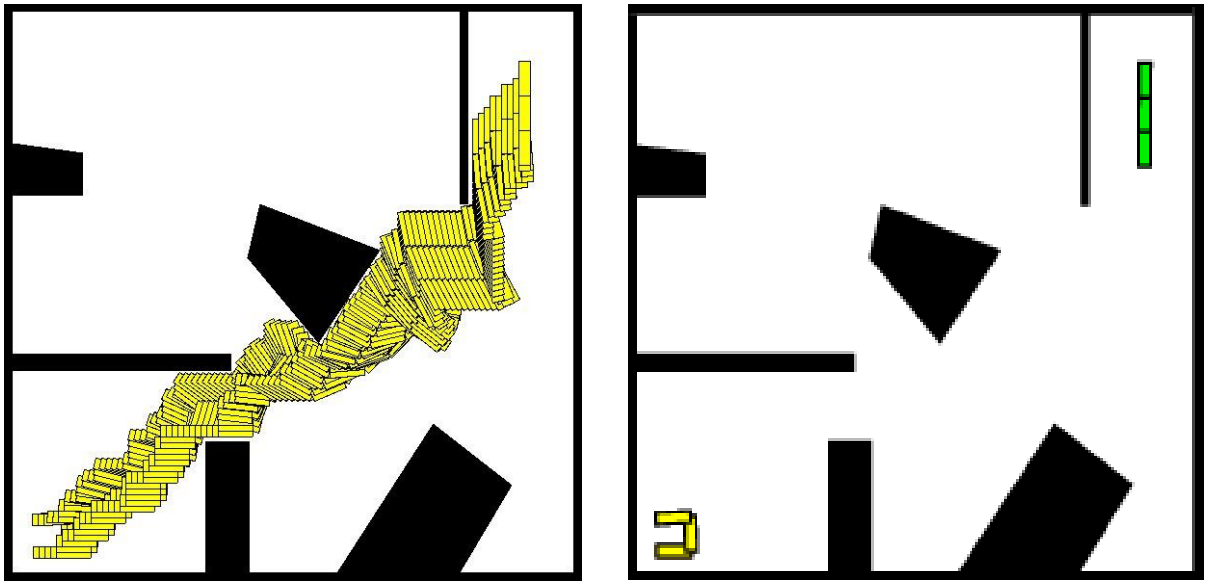
http://msl.cs.uiuc.edu/rrt/gallery.html

Why are RRT's rapidly exploring?

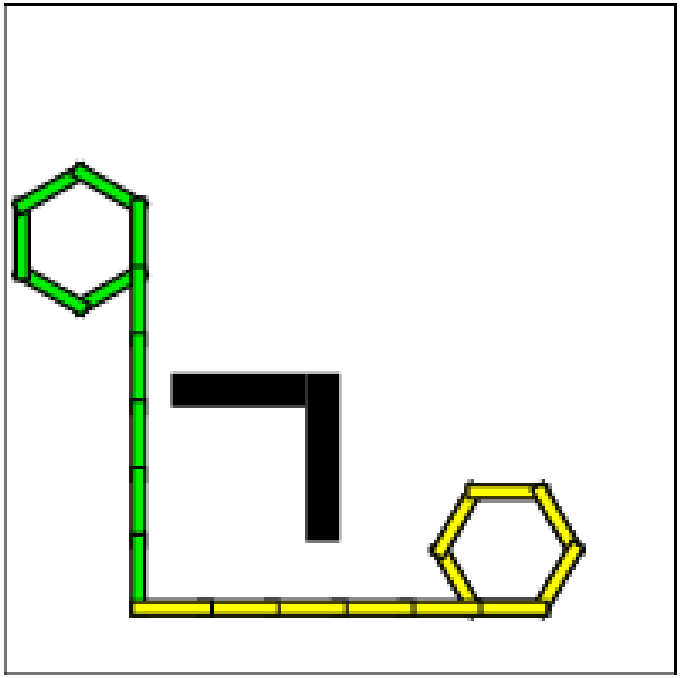


The probability of a node being selected for expansion (i.e. being a nearest neighbor to a new randomly picked point) is proportional to the area of its Voronoi region.

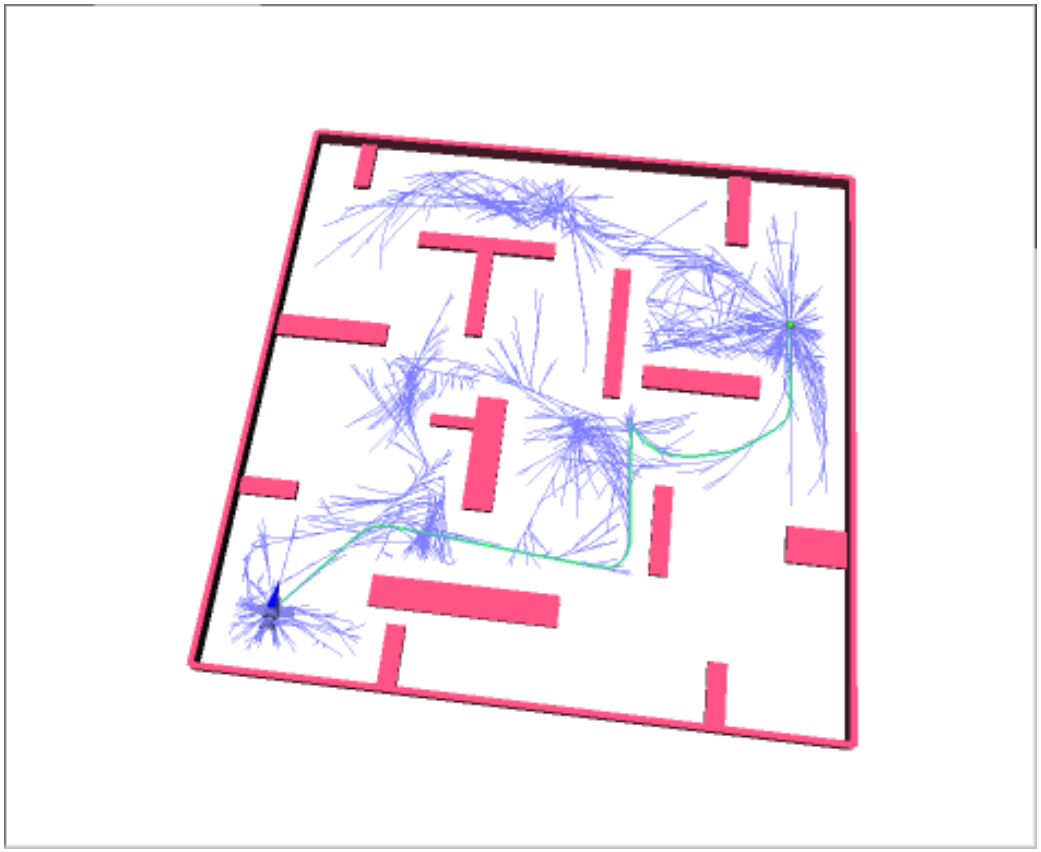
Articulated Robot



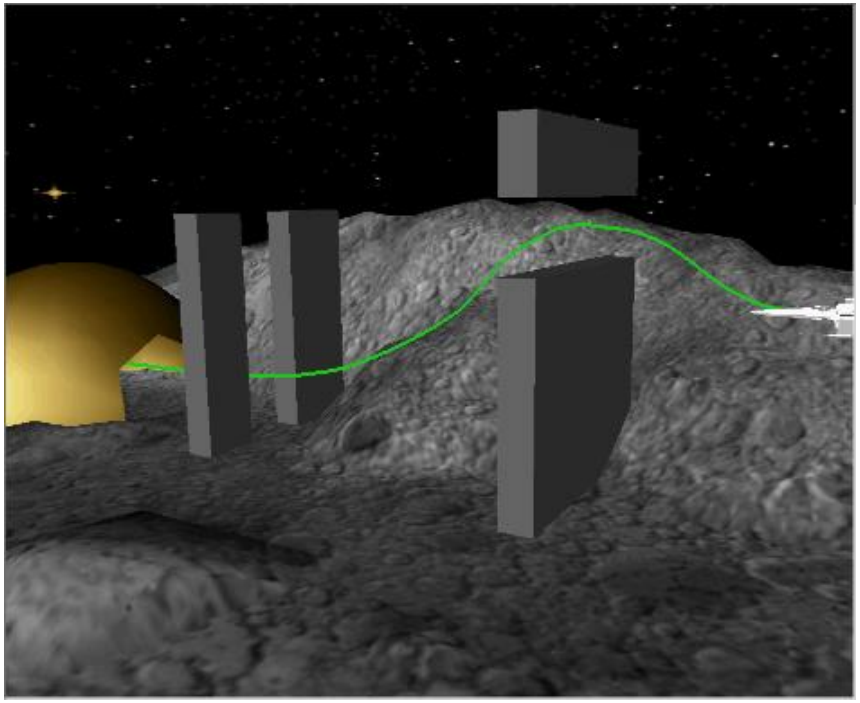
Highly Articulated Robot



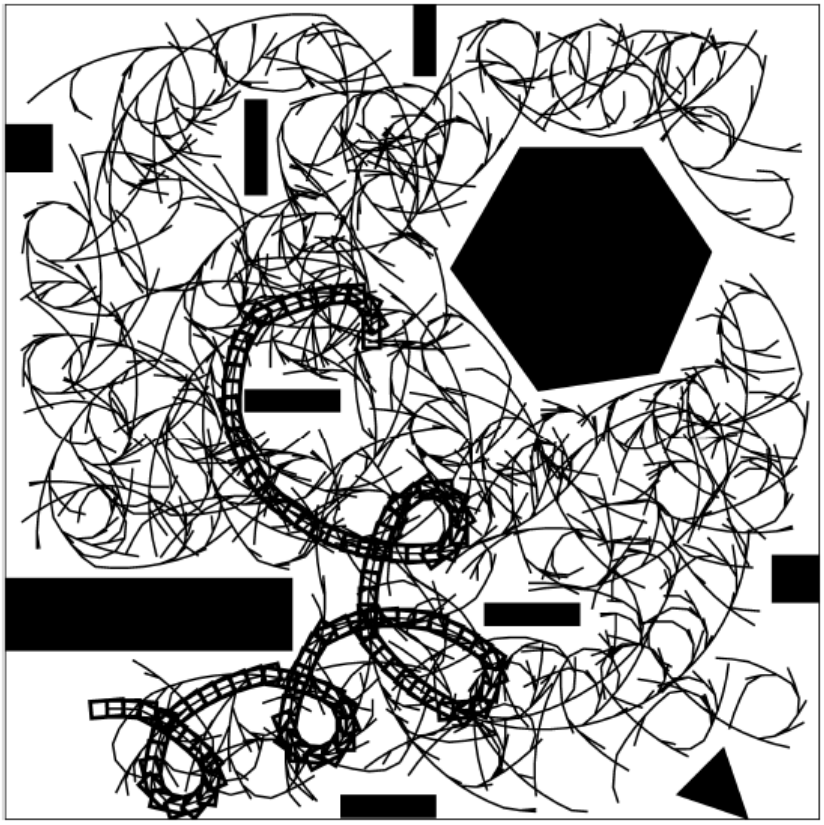
Hovercraft with 2 Thusters



Out of This World Demo



Left-turn only forward car

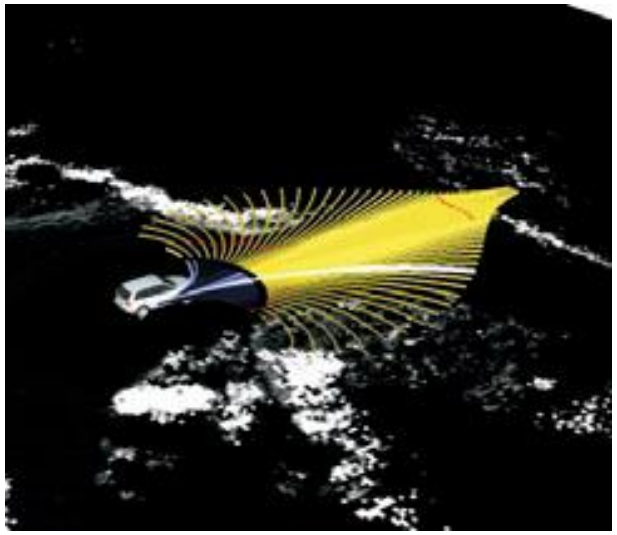


Application to Cars?

- These algorithms are great for complex planning problems, and can deal with
  - Complex dynamic models
  - Global planning problems
  - Complex environments
- This is actually 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 fairly 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!

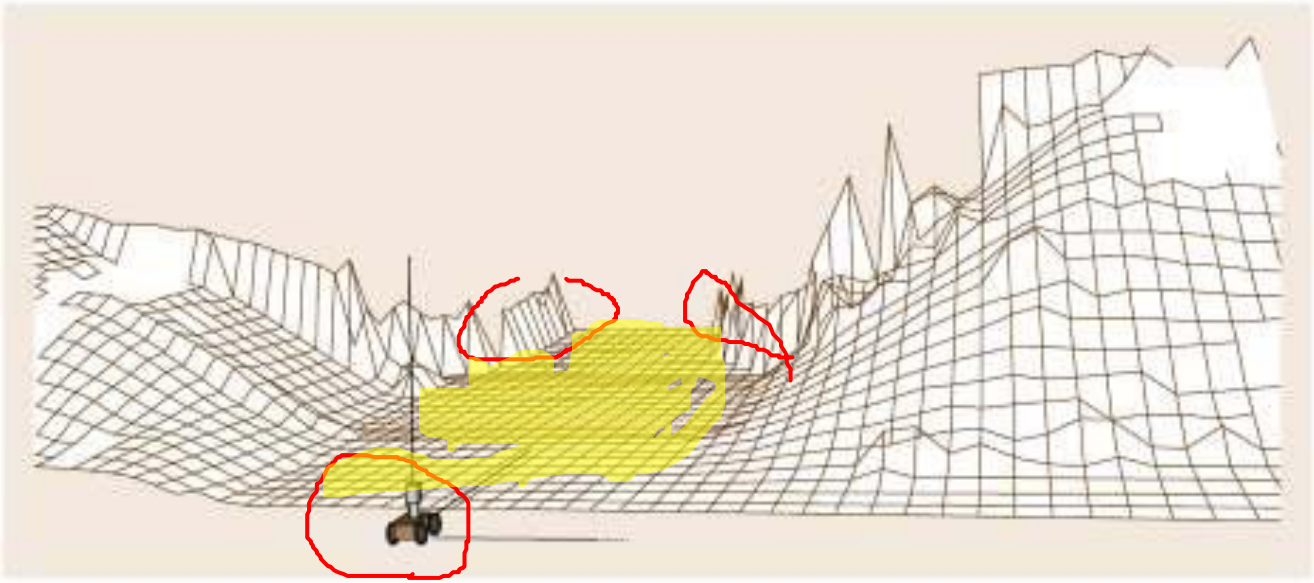
A modification to path planning…

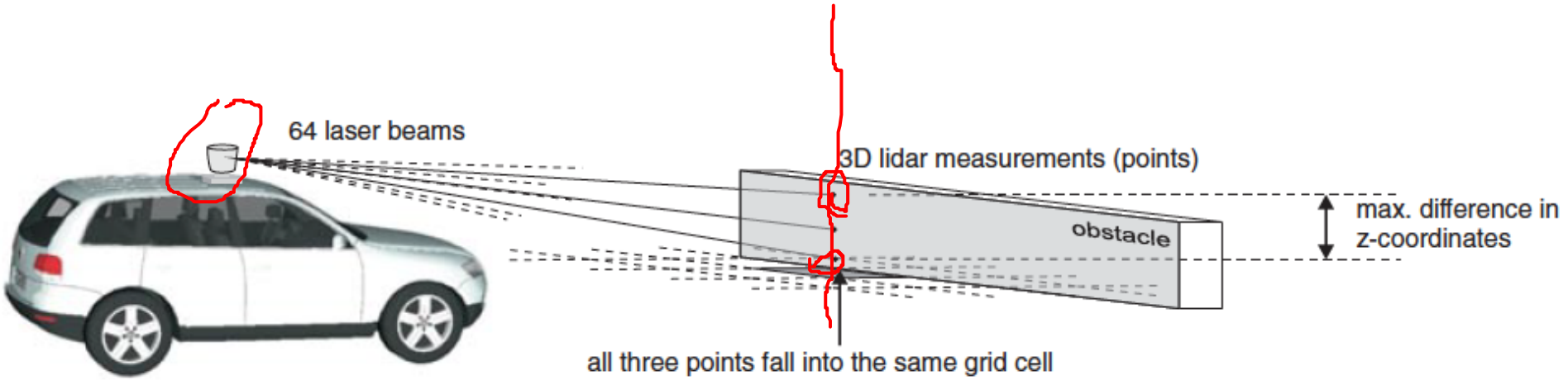
- Driving with tentacles

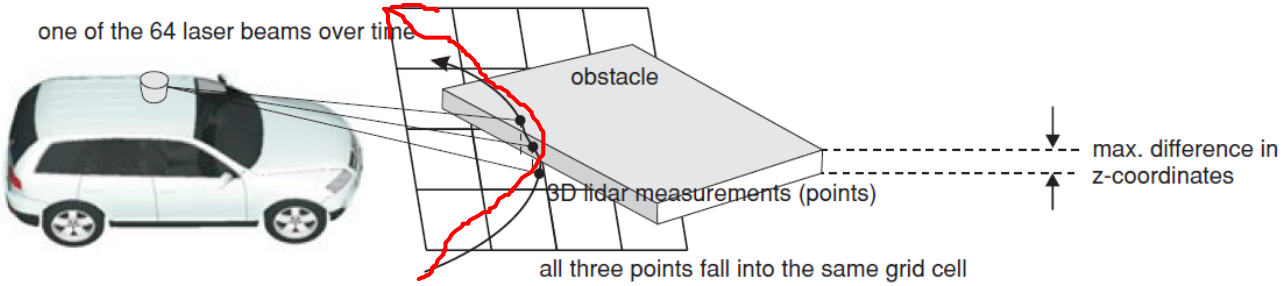


Felix von Hundelshausen, Michael Himmelsbach, Falk Hecker, Andre Mueller, and Hans-Joachim Wuensche, 2008.

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



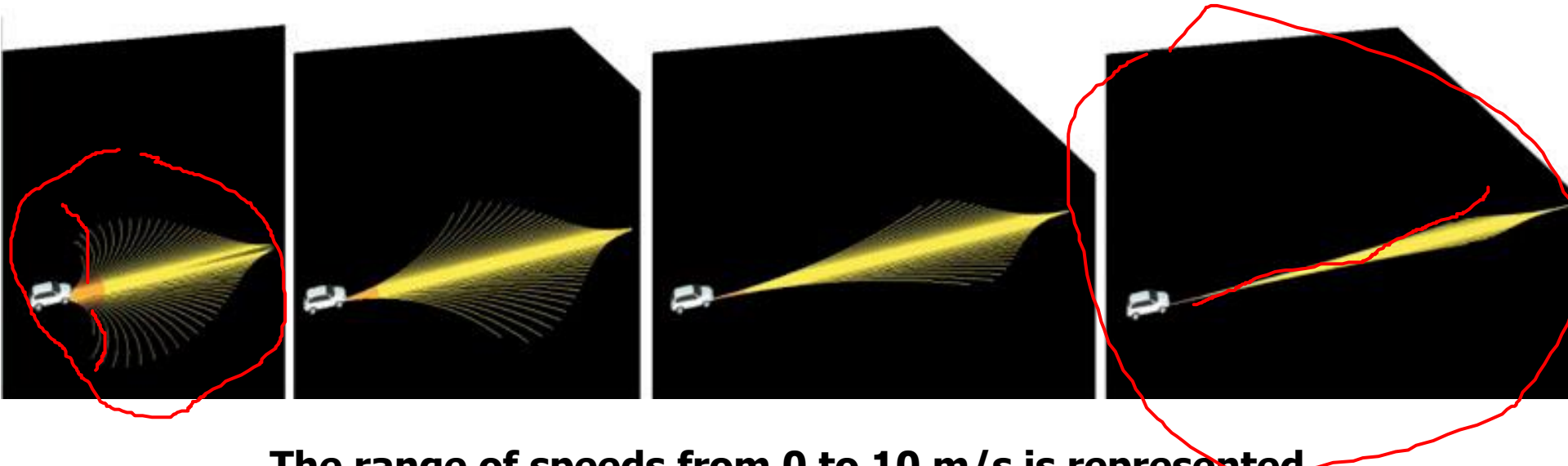




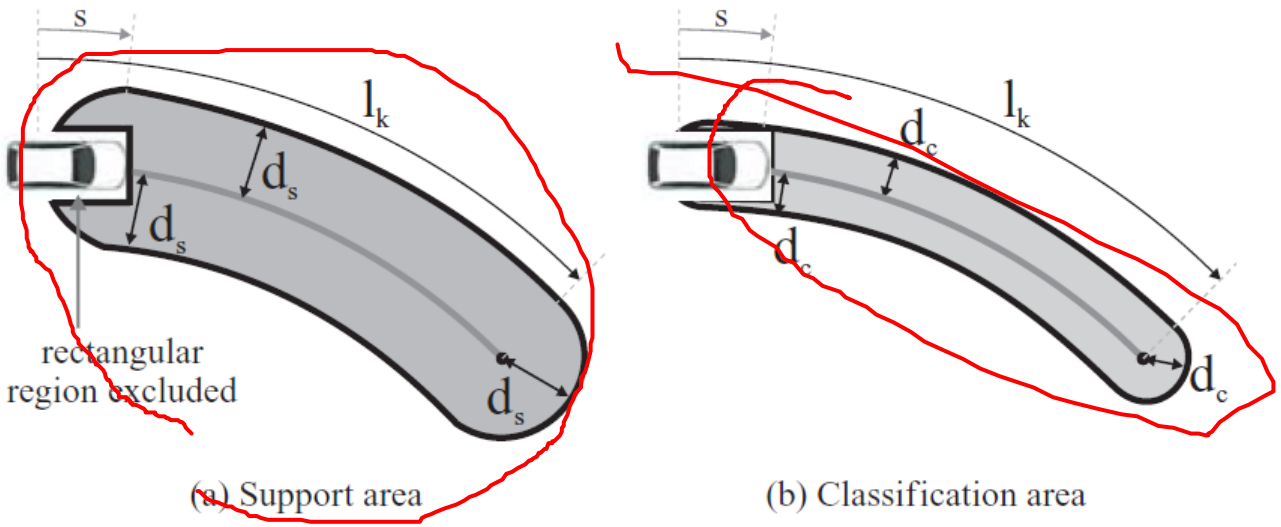
Occupancy grid value is computed to be the maximum difference of z coordinates of points in 3D space falling into that grid cell.

- 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.

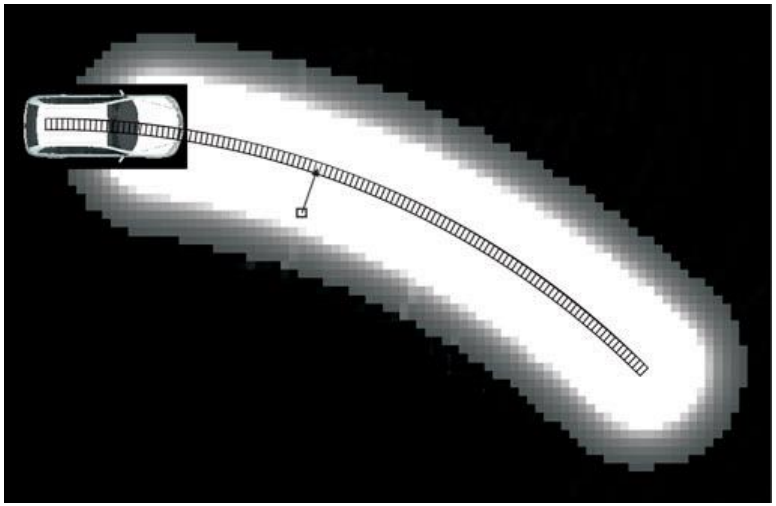


(a) The support area covers all cells within a distance d_s of the tentacle.

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

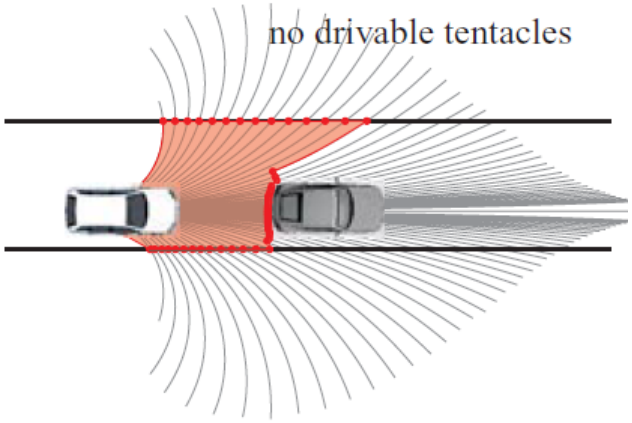
The classification area must be free for the tentacle to be driveable. The support area is preferred to be free.

Checking for Obstacles

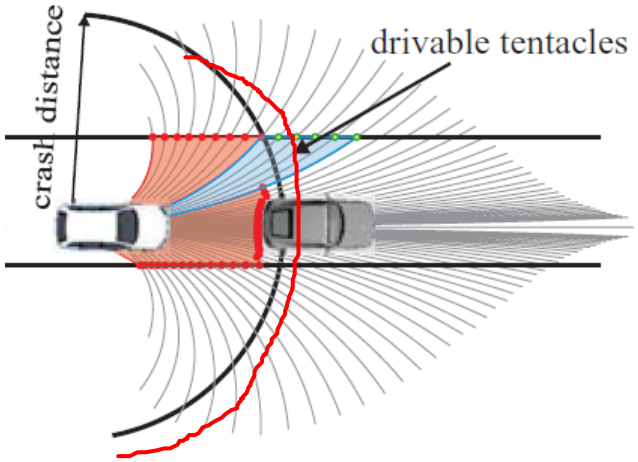


The algorithm looks at 5 consecutive cells at a time (a sliding window) and reports an obstacle if at least 2 of the cells are occupied in the occupancy grid.

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.

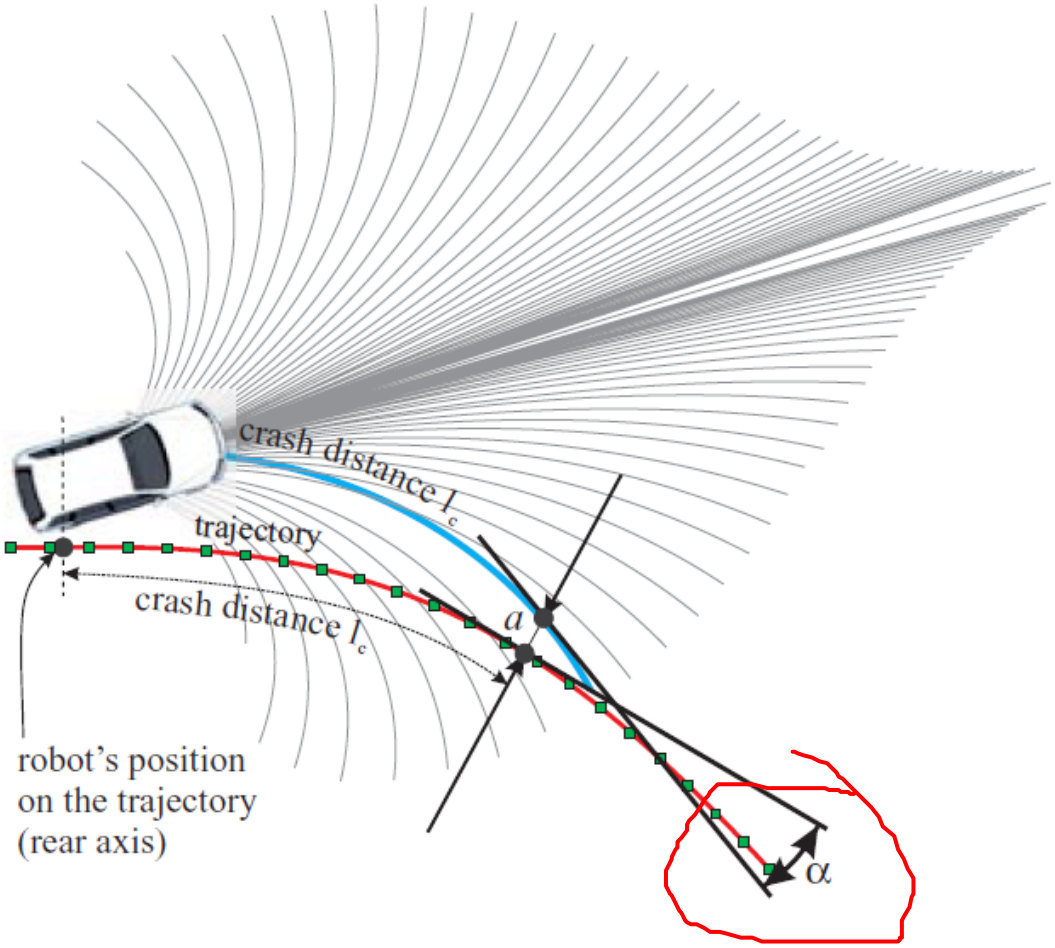


(a) By neglecting the distance to an obstacle, all tentacles would be classified undrivable.



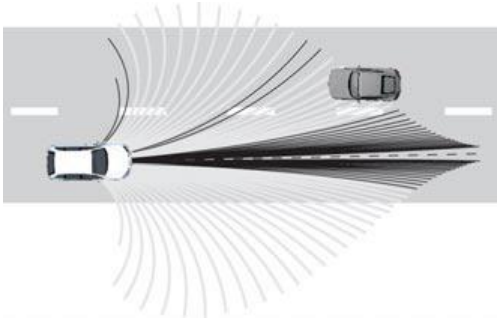
(b) shows the concept of classifying tentacles as undrivable only in case of being occupied within a speed-dependent crash distance. In this case, some drivable tentacles remain, allowing a pass of the car.

Using tentacles to follow a path

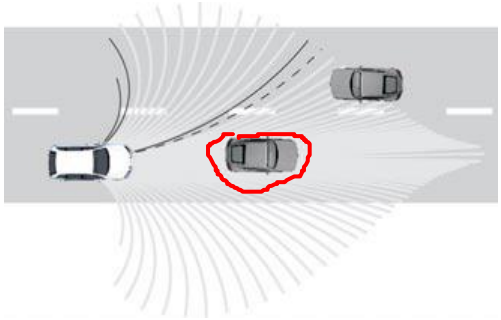


For each tentacle, a score value is computed by considering the distance and tangent orientations of two corresponding points, one on the tentacle and the other on the (GPS) trajectory to be followed.

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 out 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.