Path/Motion Planning

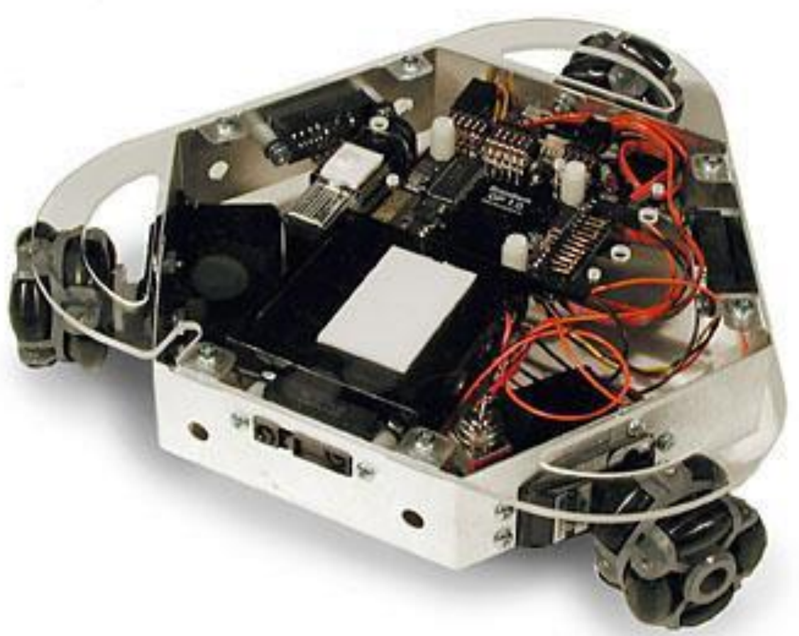
Jonas Kvarnström
Automated Planning and Diagnosis Group
Department of Computer and Information Science
Linköping University

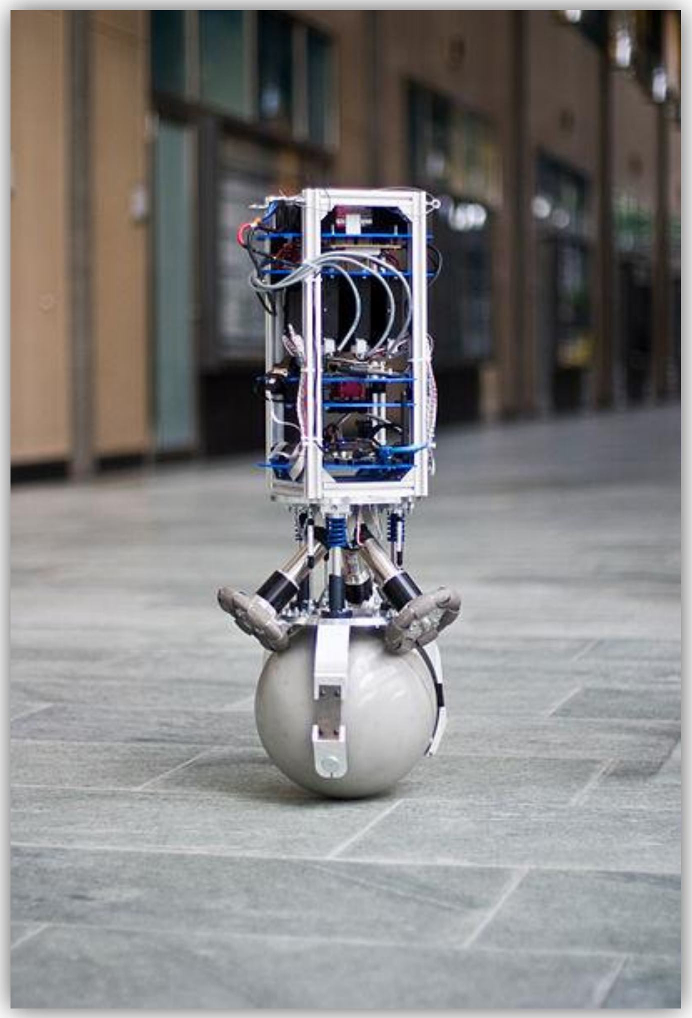
Path/Motion Planning (1)

- The *easiest* form of **path planning** / **motion planning**:
  - (1) A robot should move in **two dimensions** between start and goal
    - Avoiding known obstacles – or it would be *too* easy...

Path/Motion Planning (2)

- The *easiest* form of **path planning** / **motion planning**:
  - (2) The robot is **holonomic**
    - Informally: Can move in any direction (possibly by first rotating, then moving)





Path/Motion Planning (3)

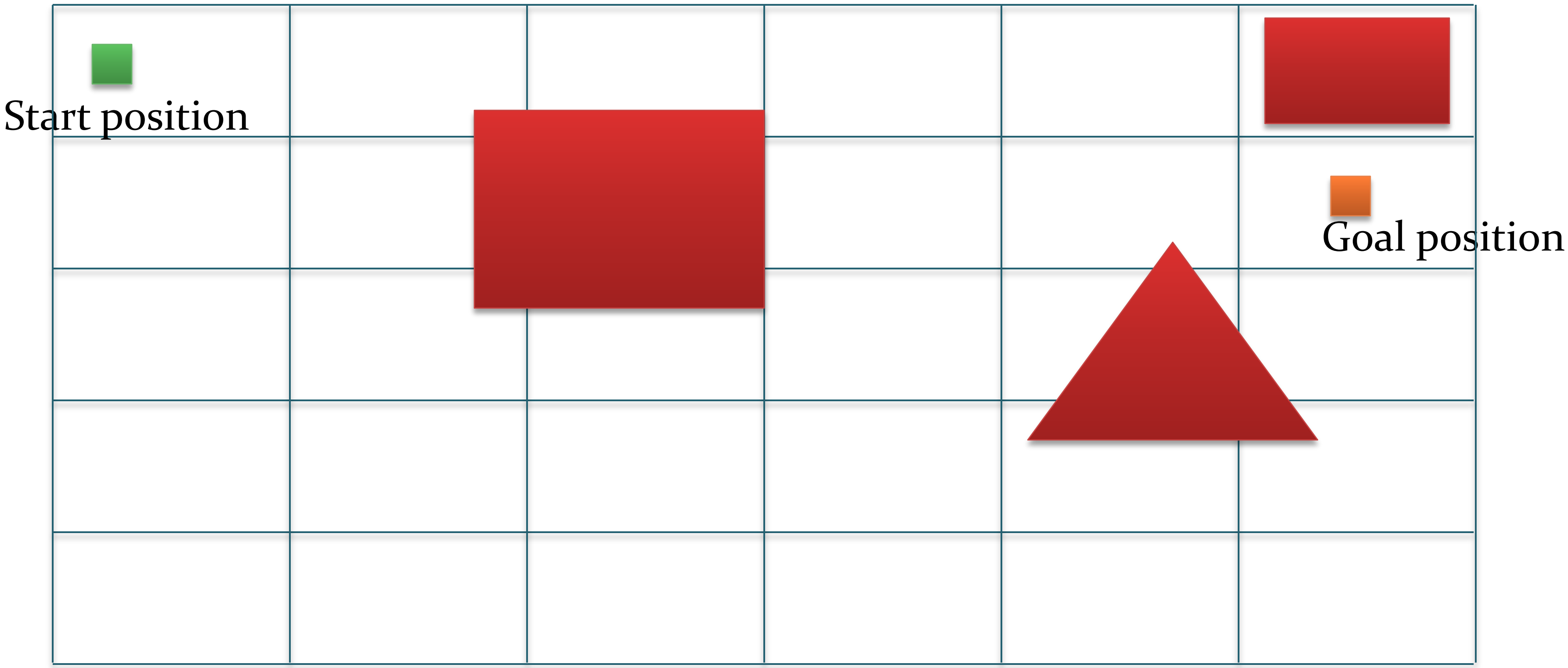
- Problem: Generating an **optimal continuous path** is hard!
  - Common solution: Divide and conquer
    - **Discretize**: Choose a finite number of **potential waypoints** in the map
    - Assume there exists a robot-specific **local planner** to determine whether one can move **between** two such waypoints (and how)
    - Use **search algorithms** to decide which waypoints to use

Remaining task: **choosing potential waypoints** + **finding a path** using them

Choosing Potential Waypoints: Grid-Based Methods

Regular 2D Grid

- The simplest type of discretization: A **<u>regular grid</u>**
  - A robot **<u>moves</u>** only *north, east, south* or *west*
    - Details are left to the local planner

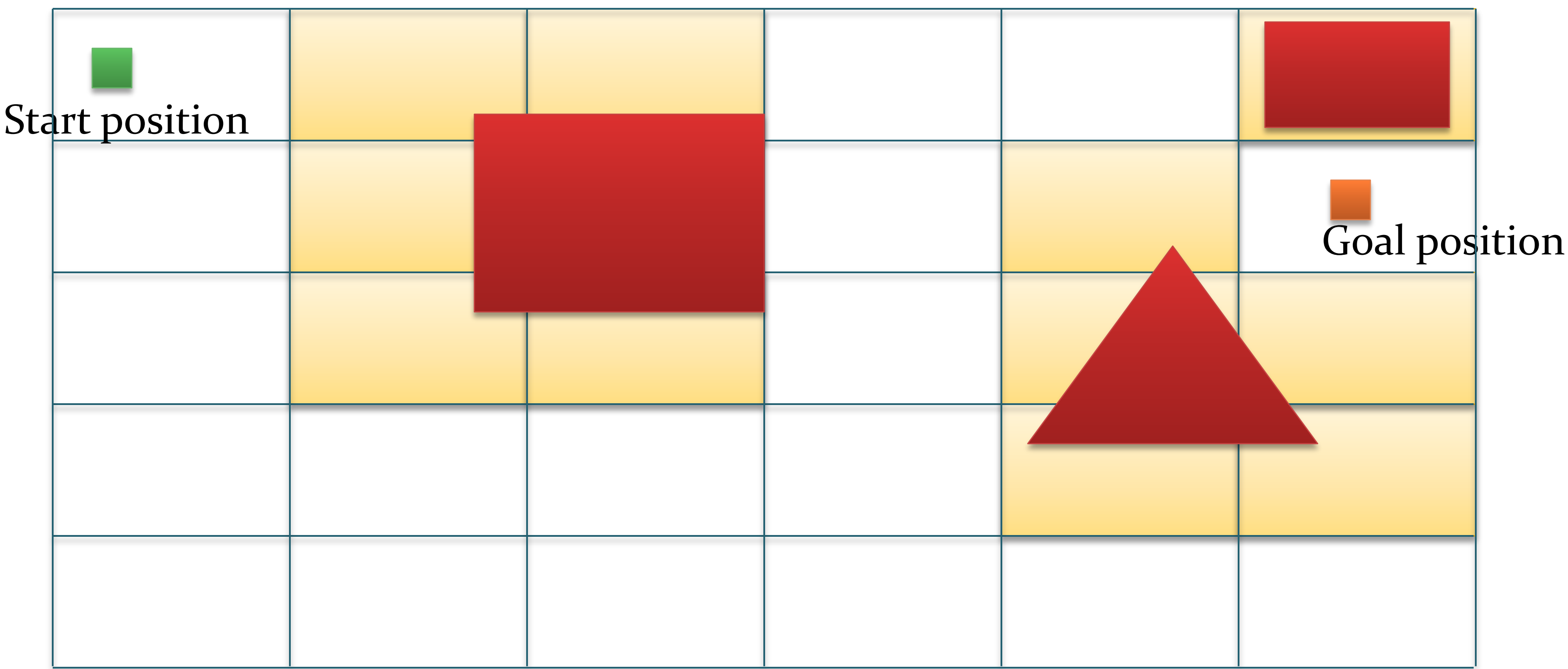


Regular 2D Grid: Real Obstacles

- Real obstacles do not correspond to square / rectangular cells…
  - But we can *cover* them with cells

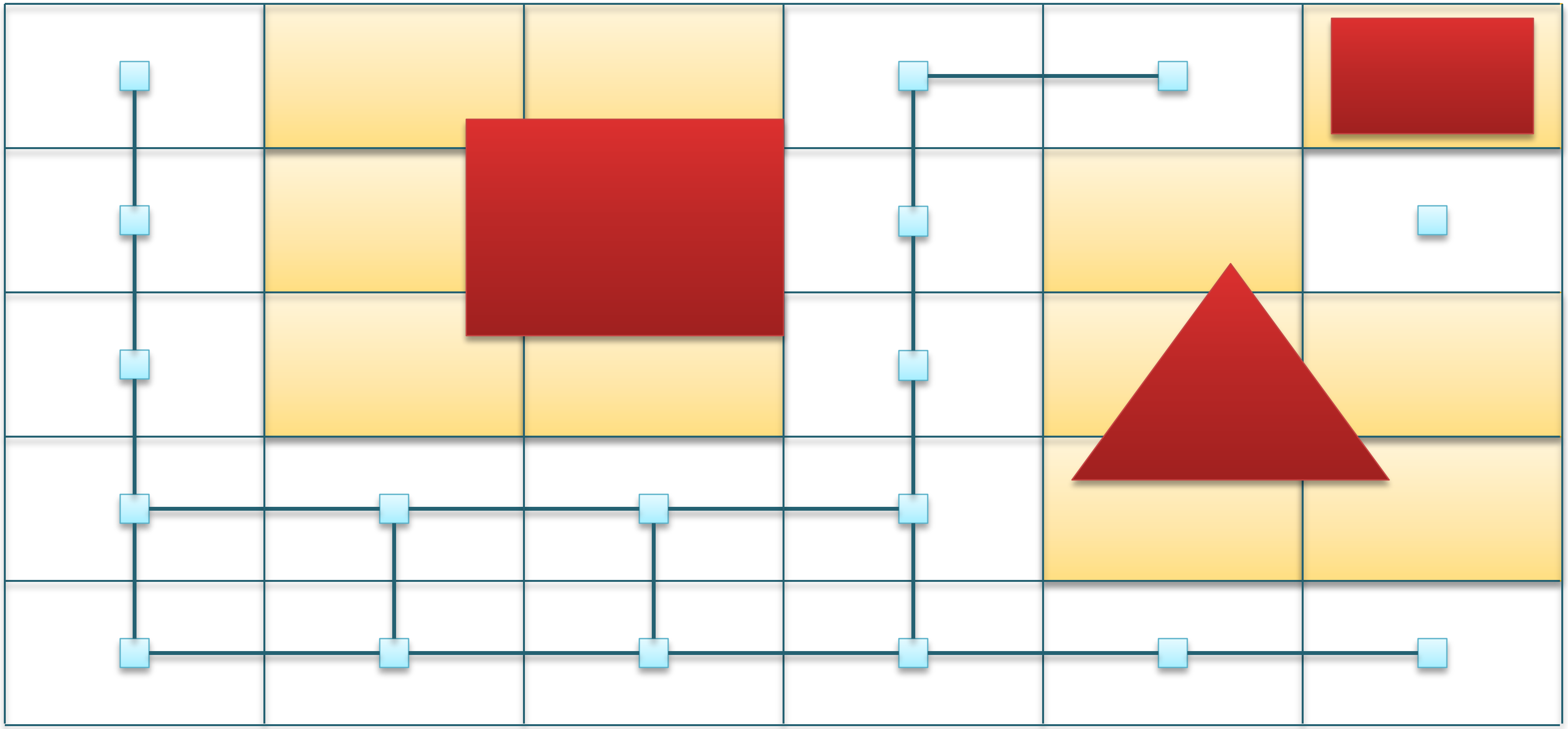
Partially covered – can't be used

Obstacle



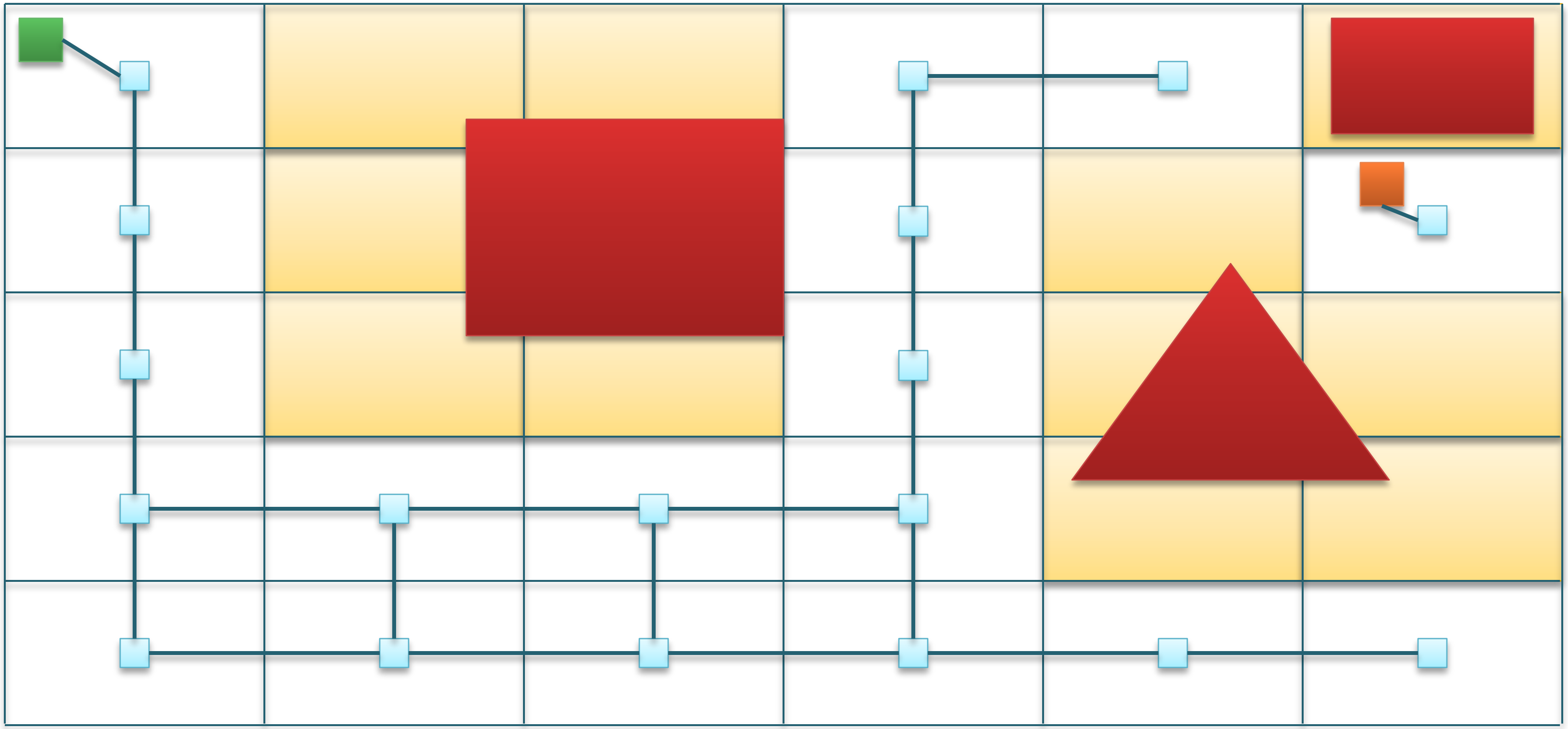
Regular 2D Grid: Discrete Graph

- View the grid **<u>implicitly</u>** as a **<u>discrete graph</u>**
  - Assume the local path planner can take us between any neighboring cells
    - Between blue nodes
    - No obstacles in the way
    - Sufficient free space to deal with non-holonomic constraints



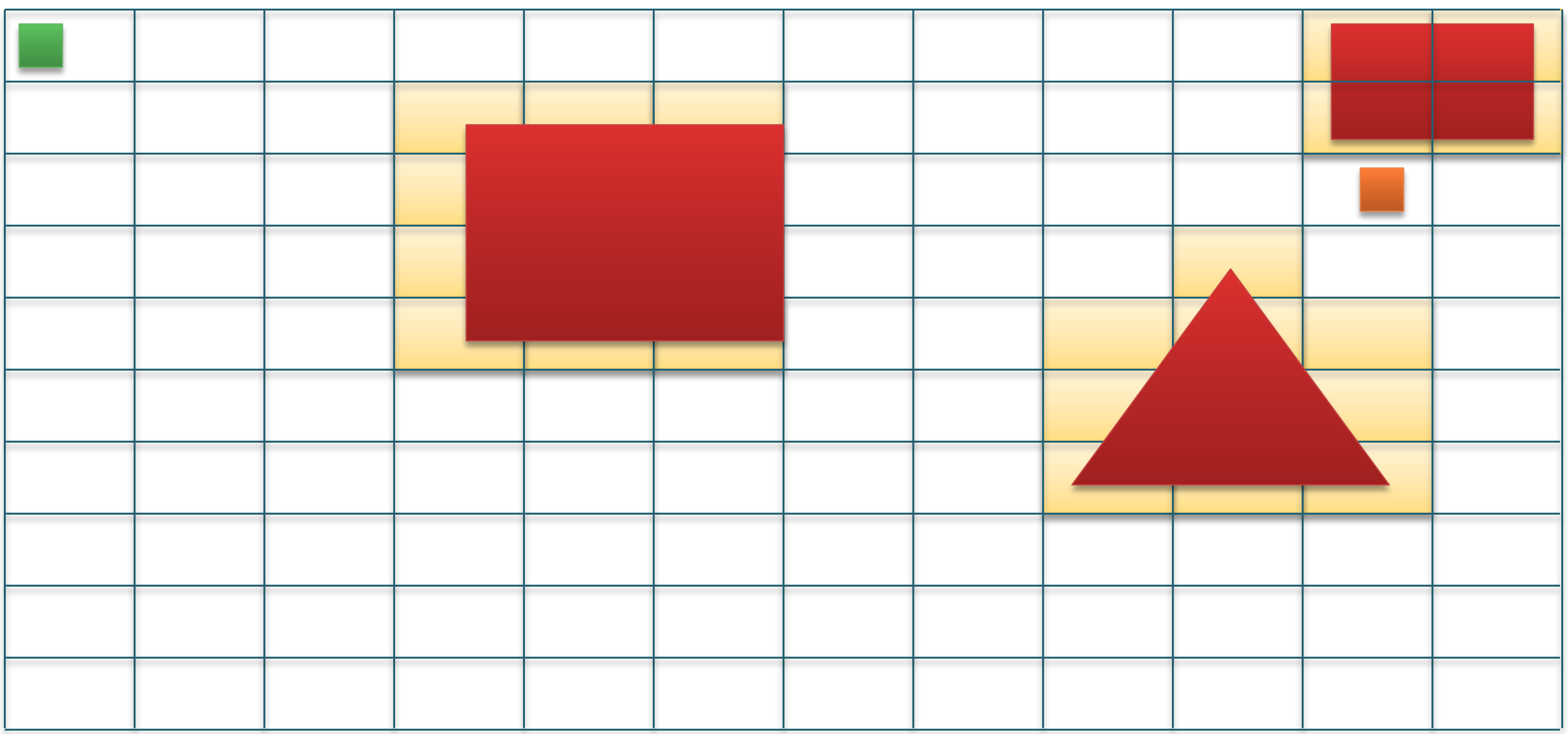
Regular 2D Grid: Discrete Graph (2)

- Connect start/goal configurations to the nodes in their cells
  - Within a cell ➔ no obstacles ➔ can plan a path using local planner
  - Here, the goal is unreachable…



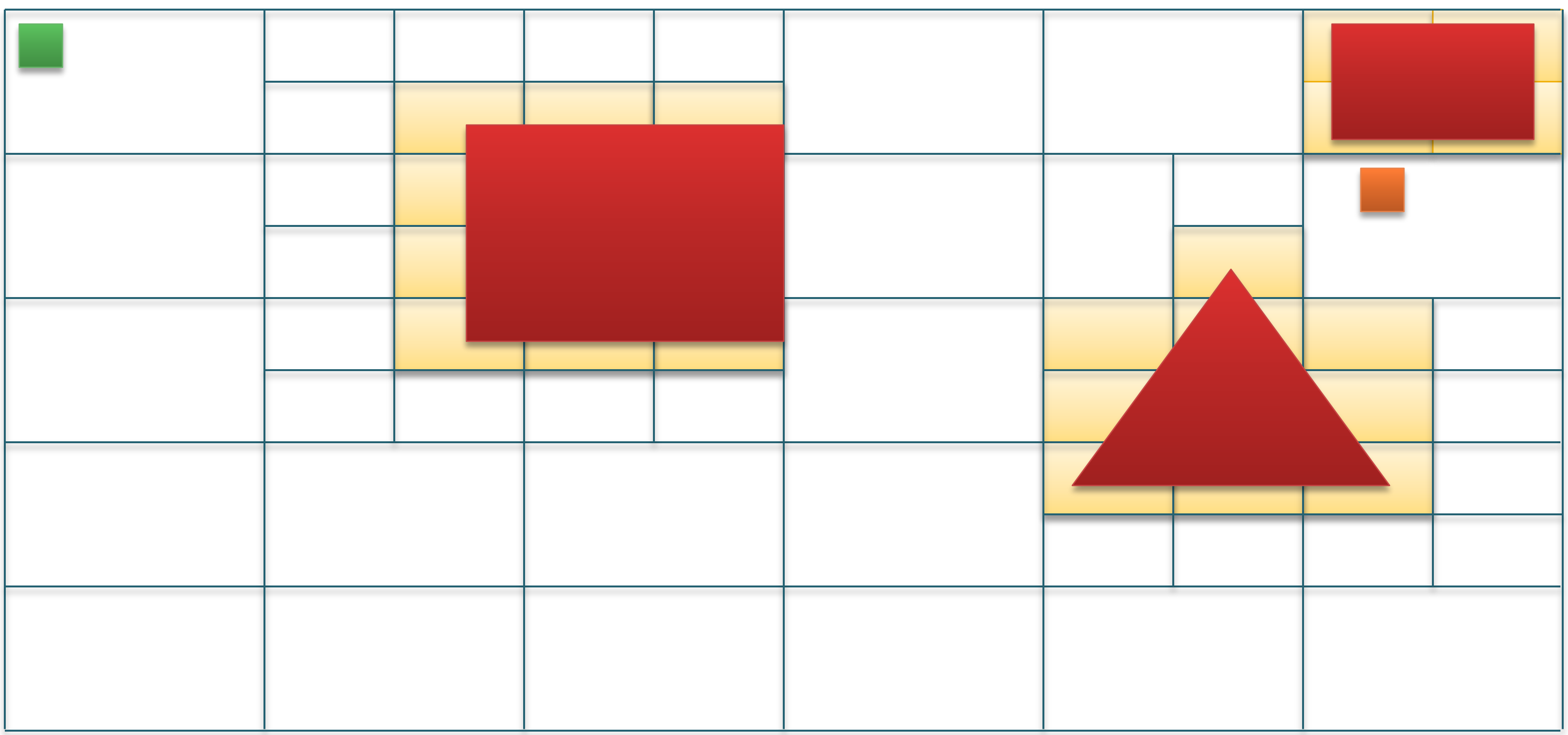
Regular 2D Grid: Grid Density

- **<u>Grid density</u>** matters!
  - Here: 4 times as many grid cells
  - Better approximation of the true obstacles, but many more nodes to search



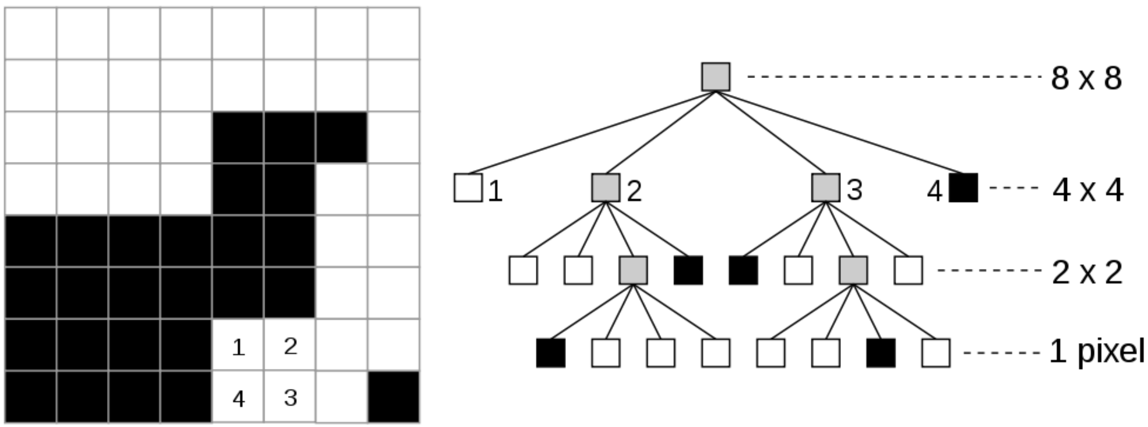
Non-Regular Grids

- Alternative: Use **non-regular** grids
  - For example, denser around obstacles
  - (Or even non-rectangular cells)



Grid Representations

- Space-efficient data structure: **quadtree**
  - Each node keeps track of:
    - Whether it is *completely* covered, *partially* covered or *non-covered*
  - Each non-leaf node has exactly four children



  - Can be generalized to 3D (octree), …

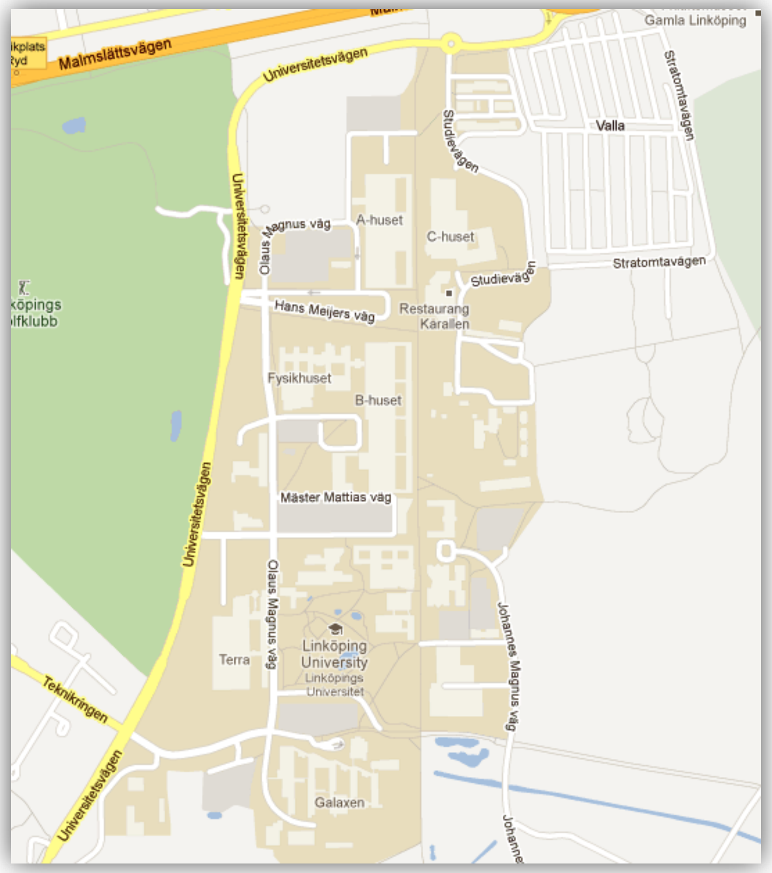
Choosing Potential Waypoints: Geometry-Based Methods

Regular 2D Grid: Grid Density

- Grid-based methods can result in many nodes
  - Even with efficient representation, *searching* the graph takes time
  - Alternative idea: Place nodes depending on obstacles

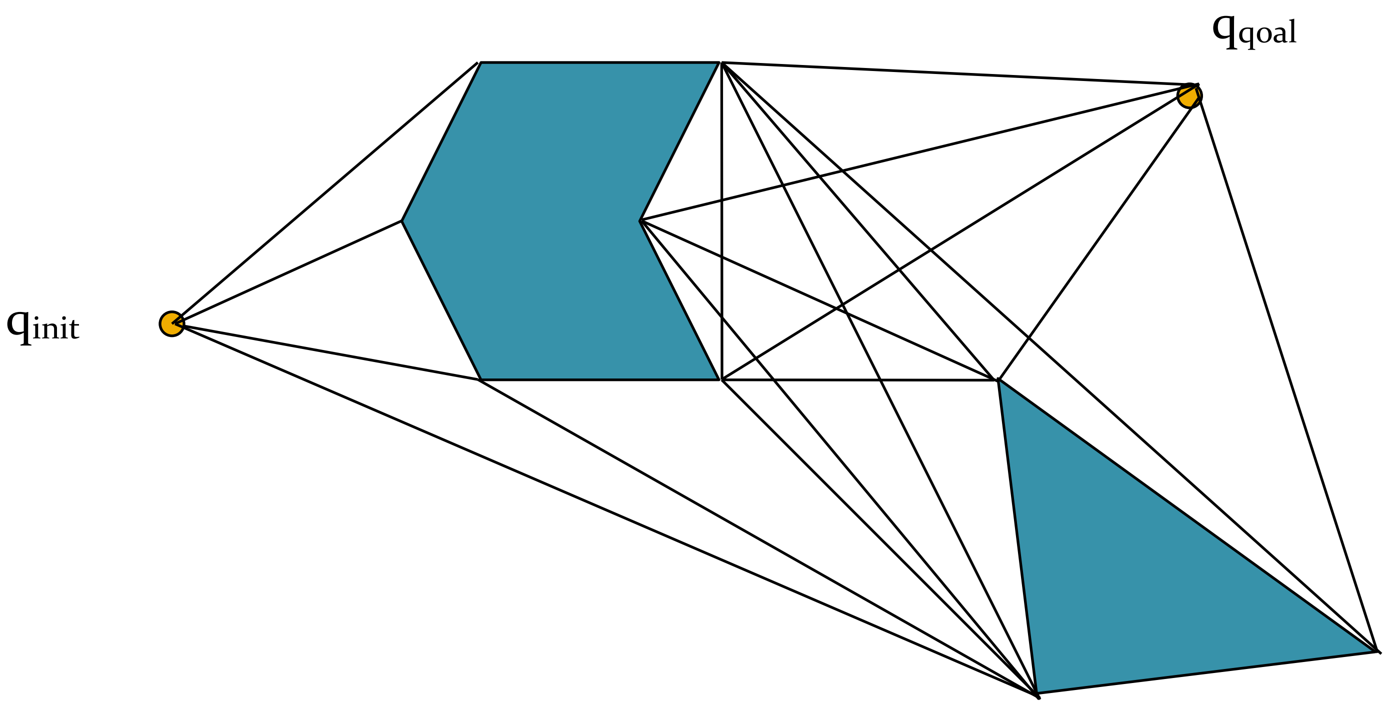
- Simple case: Known road map
  - Model all non-road areas as obstacles, then add a dense grid?

  - Or place a node in each intersection?



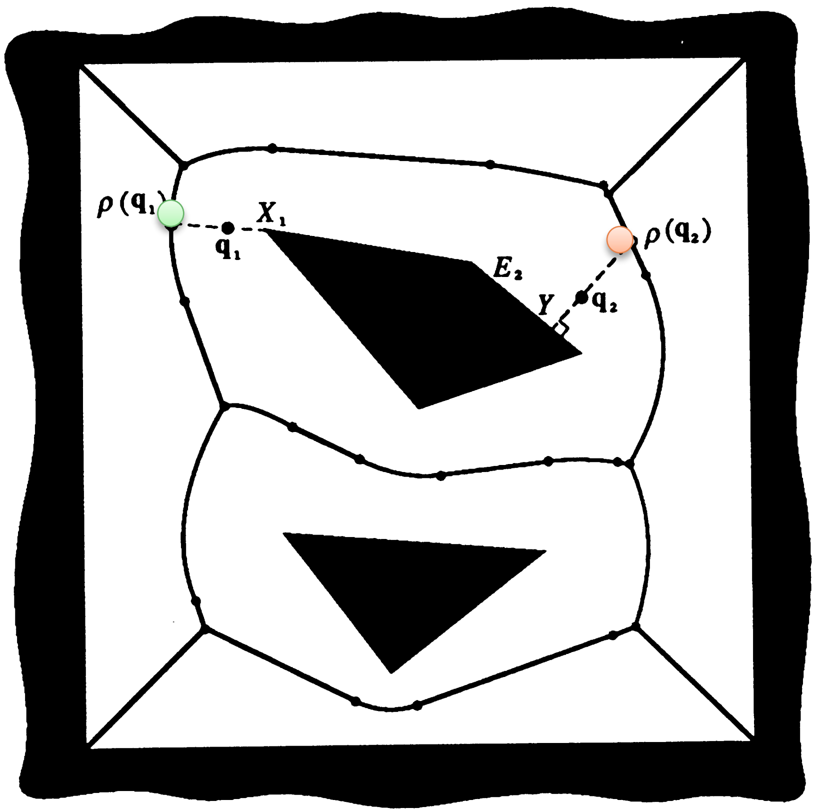
Visibility Graphs

- **Visibility graphs**
  - Applicable to simple polygons
    - Nodes at all polygon corners
    - Edges wherever a pair of nodes can be connected using the local planner
  - Mainly interesting in 2D
    - Optimal in 2D, not in 3D



Voronoi Diagrams

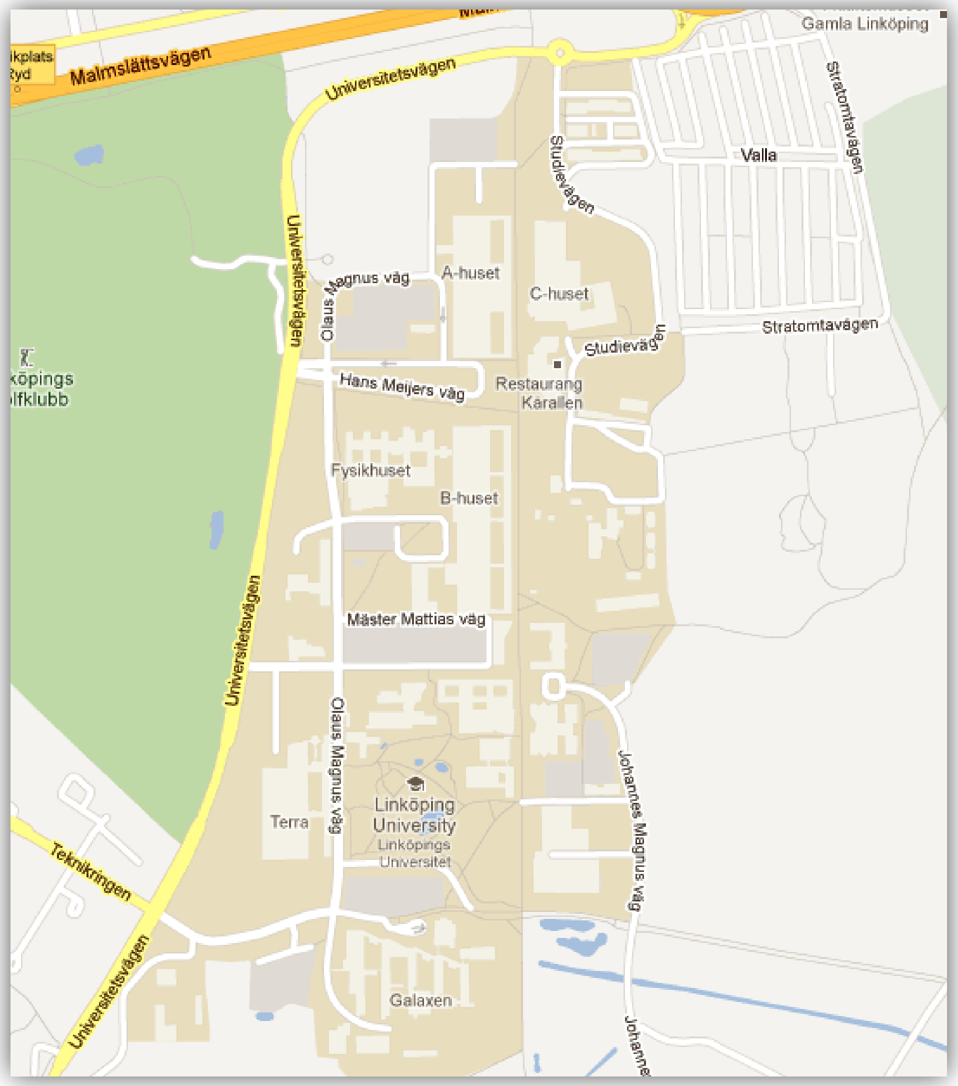
- **<u>Voronoi</u>** diagrams
  - Find all points that have the same distance to two or more obstacles
    - Maximizes clearance (free distance to the nearest obstacle)
  - Creates unnecessary detours
  - Mainly interesting in 2D – does not scale well



Complex Motion Planning Problems

Work Space

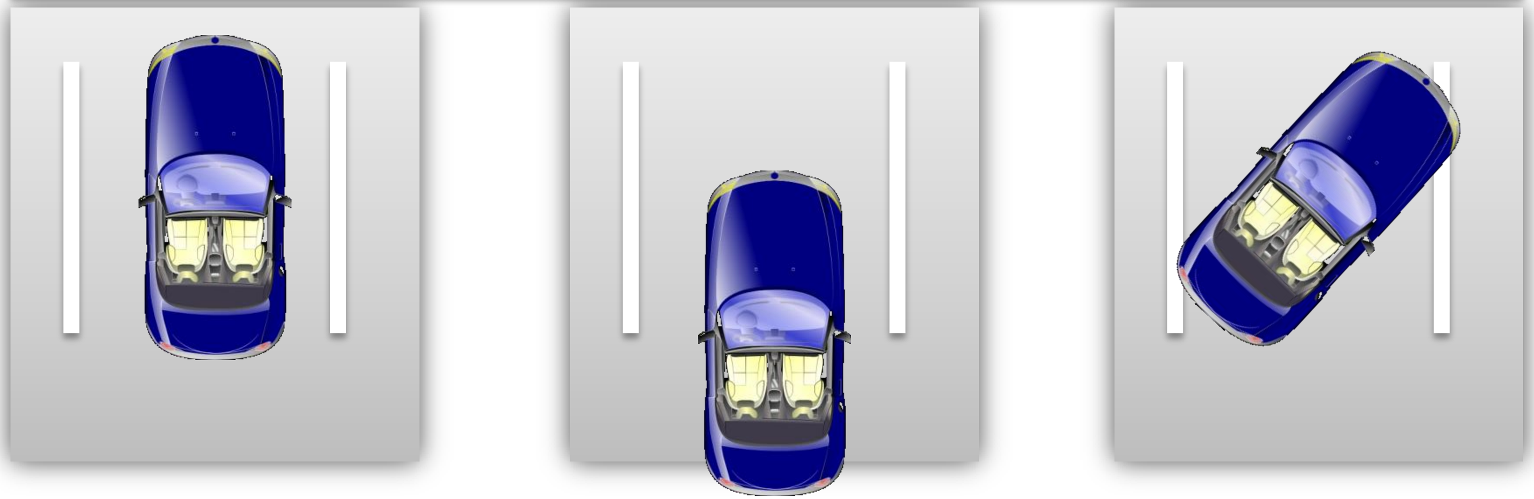
- A car moves in a *2-dimensional* plane
  - The **workspace** of the car
- Many robots have a *3-dimensional* workspace



Configuration Space

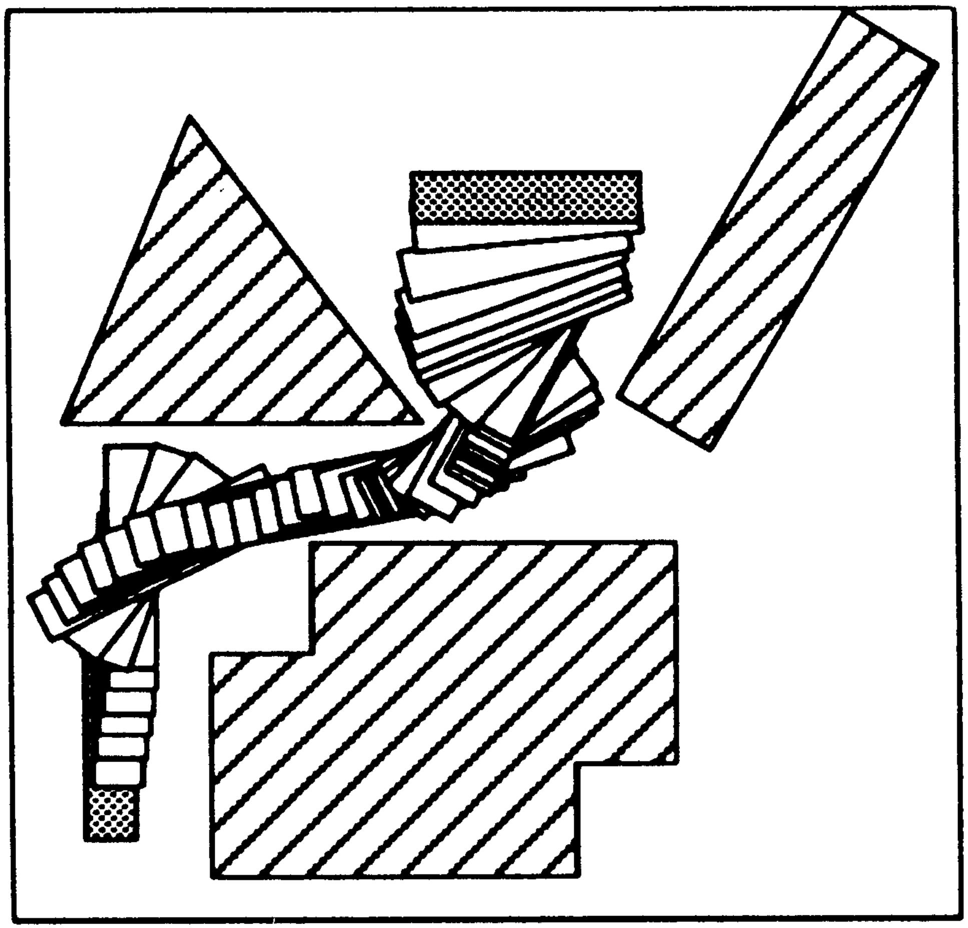
- Even a car has 3 **physical degrees of freedom** (DOF)!
  - The **configuration space** of the car
    - **Location** in the plane (*x*/*y*),
    - **Angle** (θ)
  - Each DOF is essential!
    - As part of the *goal* – park at the correct angle
    - As part of the *solution* – must turn the car to get through narrow passages

Motion planning takes place in configuration space:
How do I get from (200, 200, 12°) to (800, 400, 90°)?



The Ladder Problem

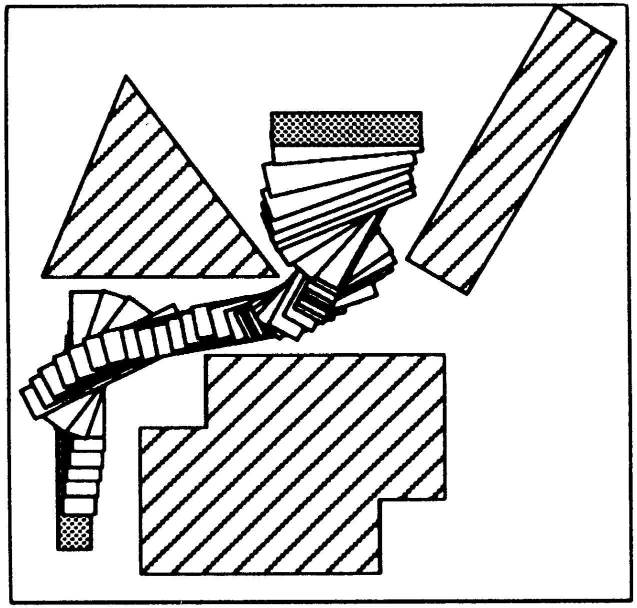
- The **ladder problem** is similar
  - Move a ladder in a 2D workspace , with 3 **physical** DOF
  - Configuration:
    - **Location** in the plane (x/y),
    - **Angle** (θ)



- Again, each DOF is essential:
  - As part of the *goal*
    - We want the ladder to end up at a specific angle
  - As part of the *solution*
    - We need to turn the ladder to get it past the obstacles

The Ladder Problem: Controllable DOF

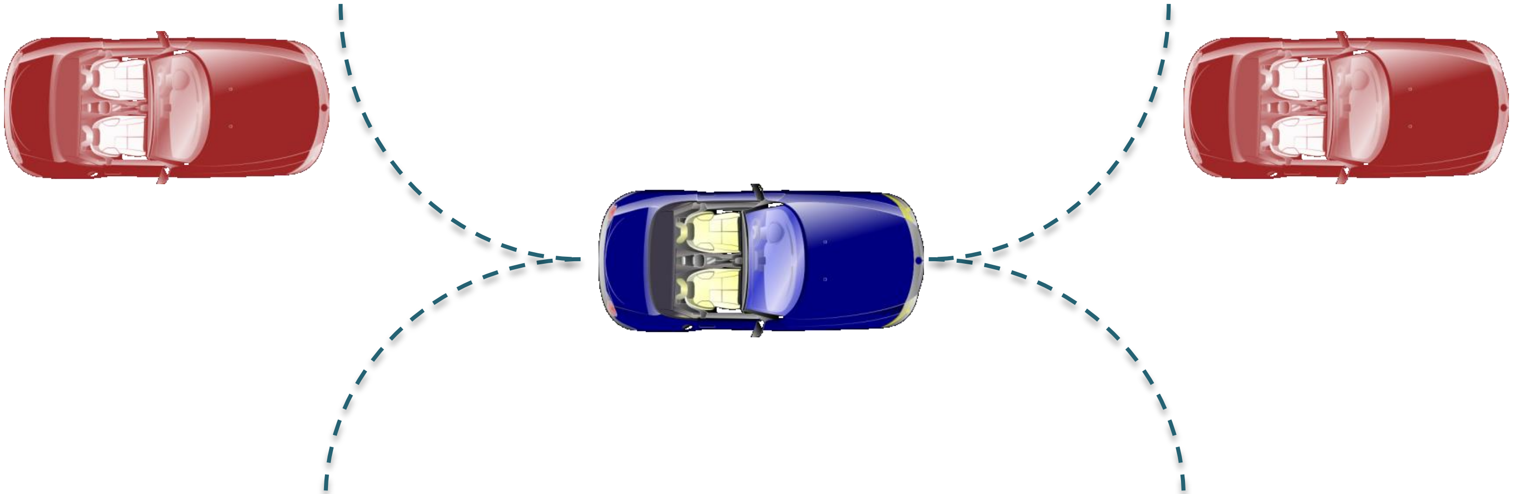
- For ladders, each physical DOF is directly controllable!
  - You can:
    - Change x (translate sideways)
    - Change y (translate up/down)
    - Change angle (rotate in place)
  - Therefore:
    - If you want to get from (200, 200, 12°) to (800, 400, 90°), any path connecting these 3D points and going through free configuration space is sufficient



  - The ladder is **<u>holonomic</u>**!
    - Controllable DOF >= physical DOF

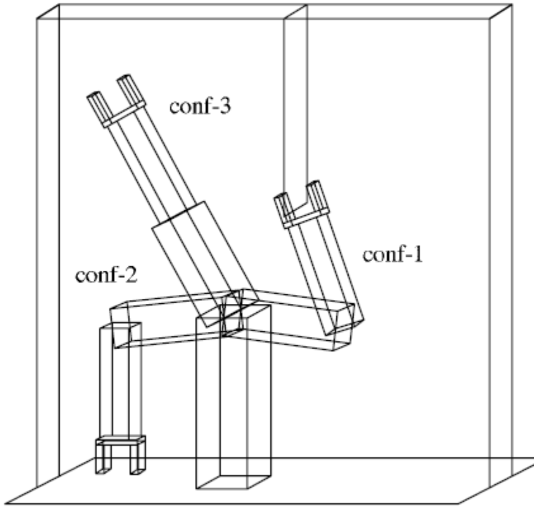
Controllable Degrees of Freedom

- For cars, we can control two DOF:
  - Acceleration/breaking
  - Turning (limited)
- In this parallel parking example:
  - There **<u>is</u>** free space between current and desired configurations
    - But we can't slide in sideways!
  - Fewer controllable DOF than physical DOF ➔ non-holonomic
    - *Limits possible curves in 3D configuration space!*

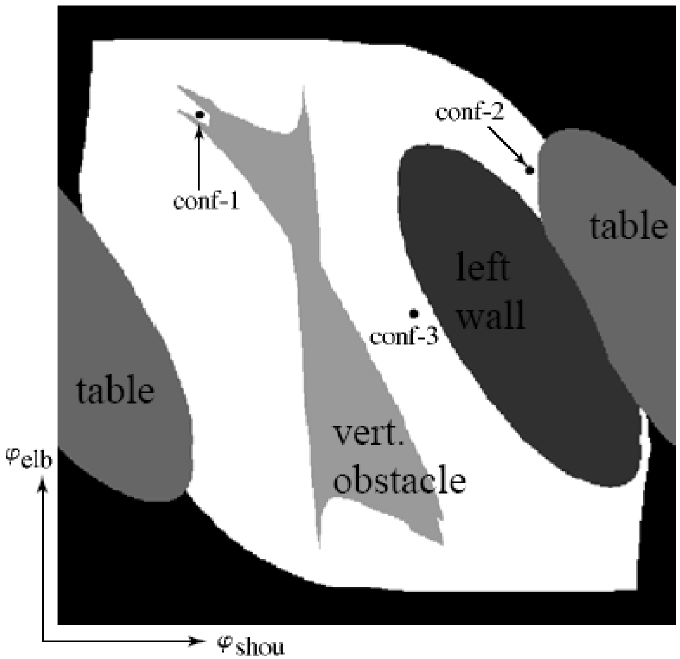


Work Space, Configuration Space

- Summary of important concepts:
  - **Work space**: The physical space in which you move
    - 3-dimensional for this robot arm

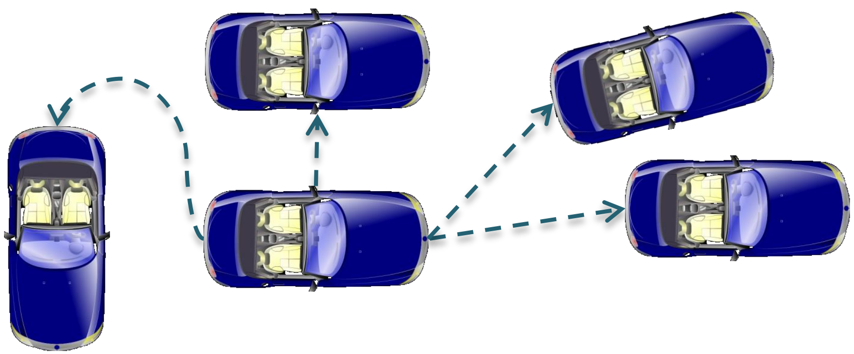


  - **Configuration space**:
    The set of possible configurations of the robot
    - Usually **continuous**
    - Often **many-dimensional**
      (one dimension per physical DOF)
    - Will often be **visualized** in 2D for clarity

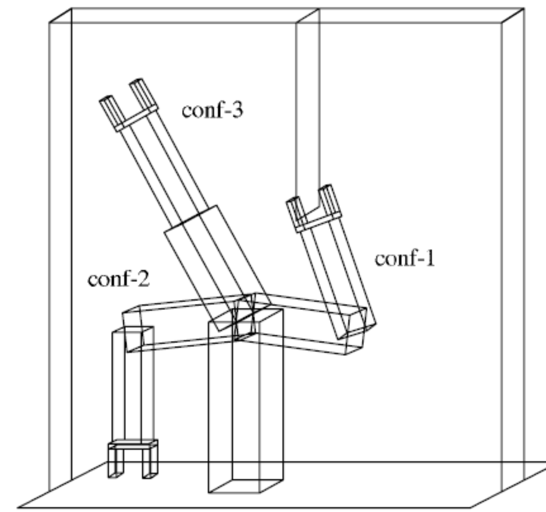


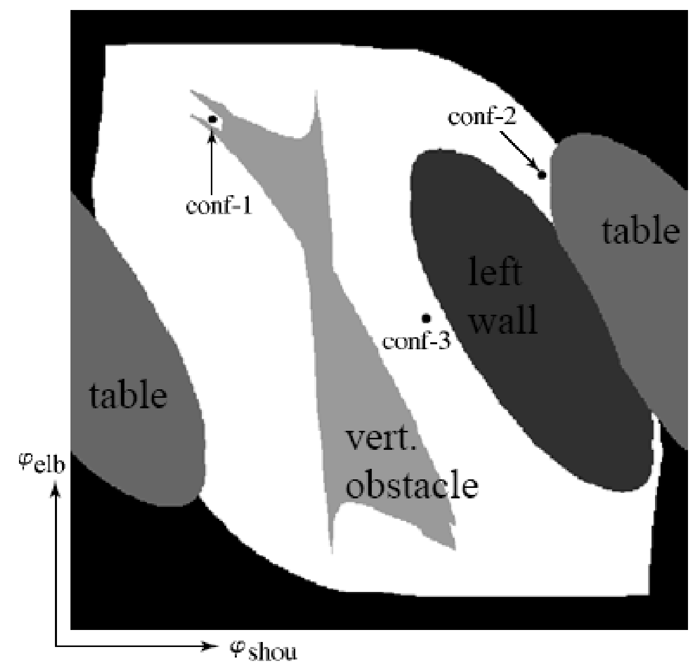
Work Space, Configuration Space (2)

- We have to search in the **configuration space**!
  - Local path planner
    - Determines whether two configurations can be connected with a path, and how
    - Considers vehicle-specific constraints



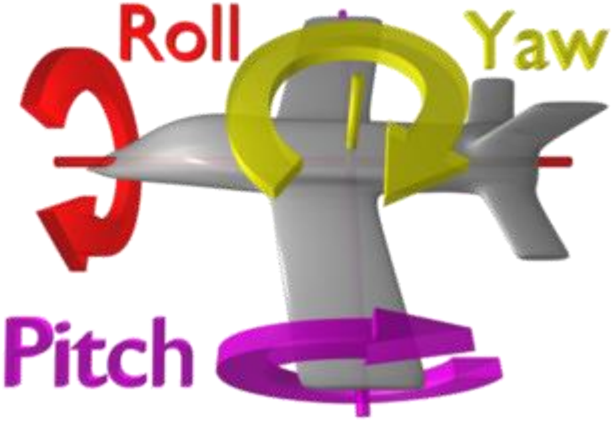
  - High-level path planner
    - Uses plug-in local planner to generate connected waypoints
    - For each specific problem, uses search to determine which waypoints to use





High-Dimensional Problems

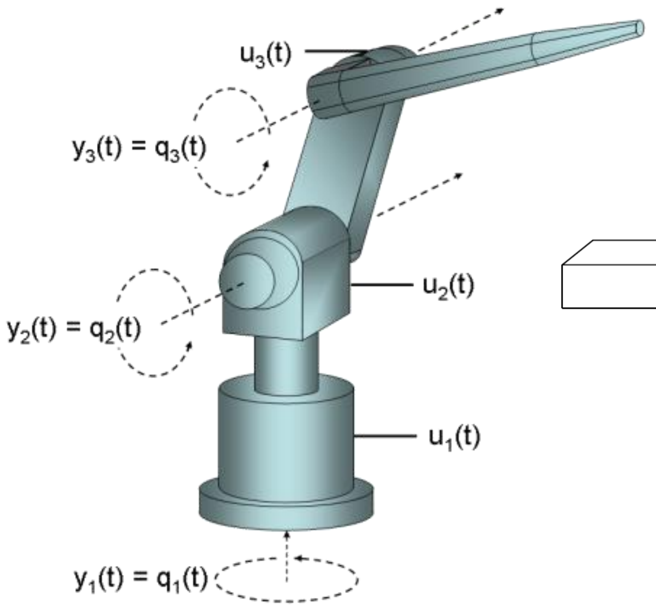
- For an **aircraft**, a configuration could consist of:
  - **location** in 3D space (x/y/z)
  - **pitch angle**
  - **yaw angle**
  - **roll angle**

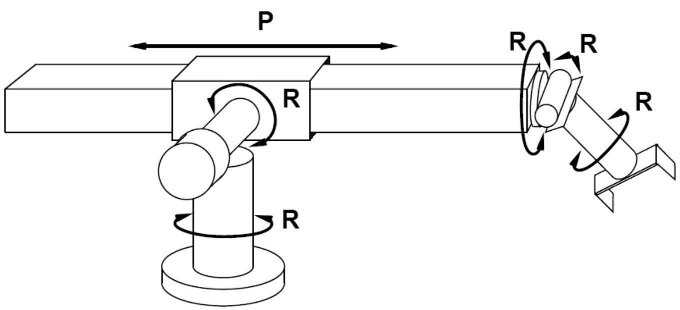


- A **path** is:
  - a continuous **curve** in 6-dimensional configuration space
    avoiding **obstacles**
    and obeying **constraints** on how the aircraft can turn
    - Can make tighter turns at low speed
    - Can't fly at arbitrary pitch angles
    - …

High-Dimensional Problems (2)

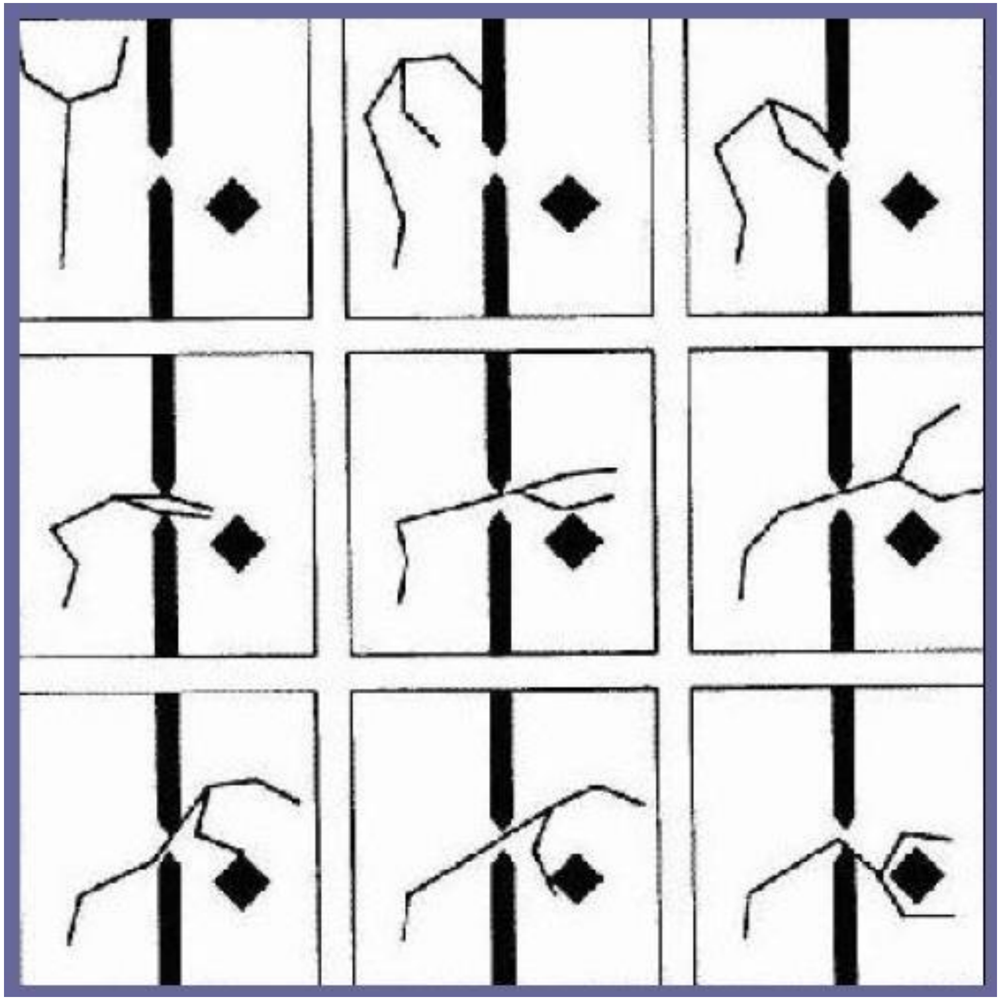
- For a **robot arm**, a configuration could consist of:
  - The position / angle of each joint
- A **path** is a continuous **curve** in n-dimensional configuration space (all joints move continuously to new positions, without “jumping”), avoiding **obstacles** and obeying **constraints** on joint endpoints etc.
- Typical goal: Reach inside the car you are painting / welding, without colliding with the car itself





High-Dimensional Problems (3)

- Moving in tight spaces, again…

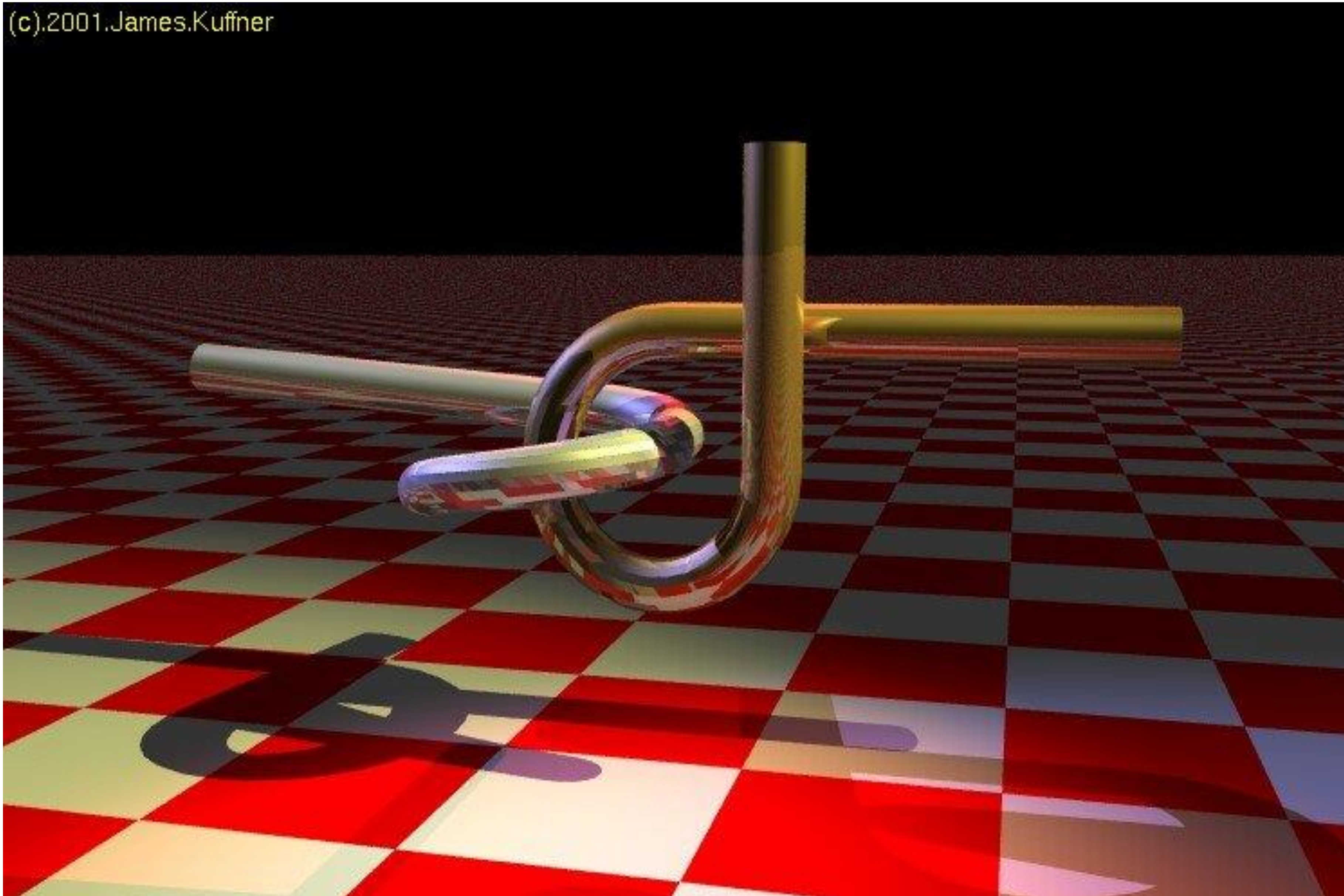


High-Dimensional Problems (4)

- For a **humanoid robot**, a configuration could consist of:
  - Position in x/y space
  - The position of each joint

- The Nao robot:
  - 14, 21 or 25 degrees of freedom depending on model
  - Up to 25-dimensional motion planning!
- Grid methods generally do not scale
  - 25-dimensional configuration space, with 1000 cells in each direction: 10^{75} cells…

Alpha Puzzle: Narrow Passages



Choosing Potential Waypoints: Probabilistic Methods

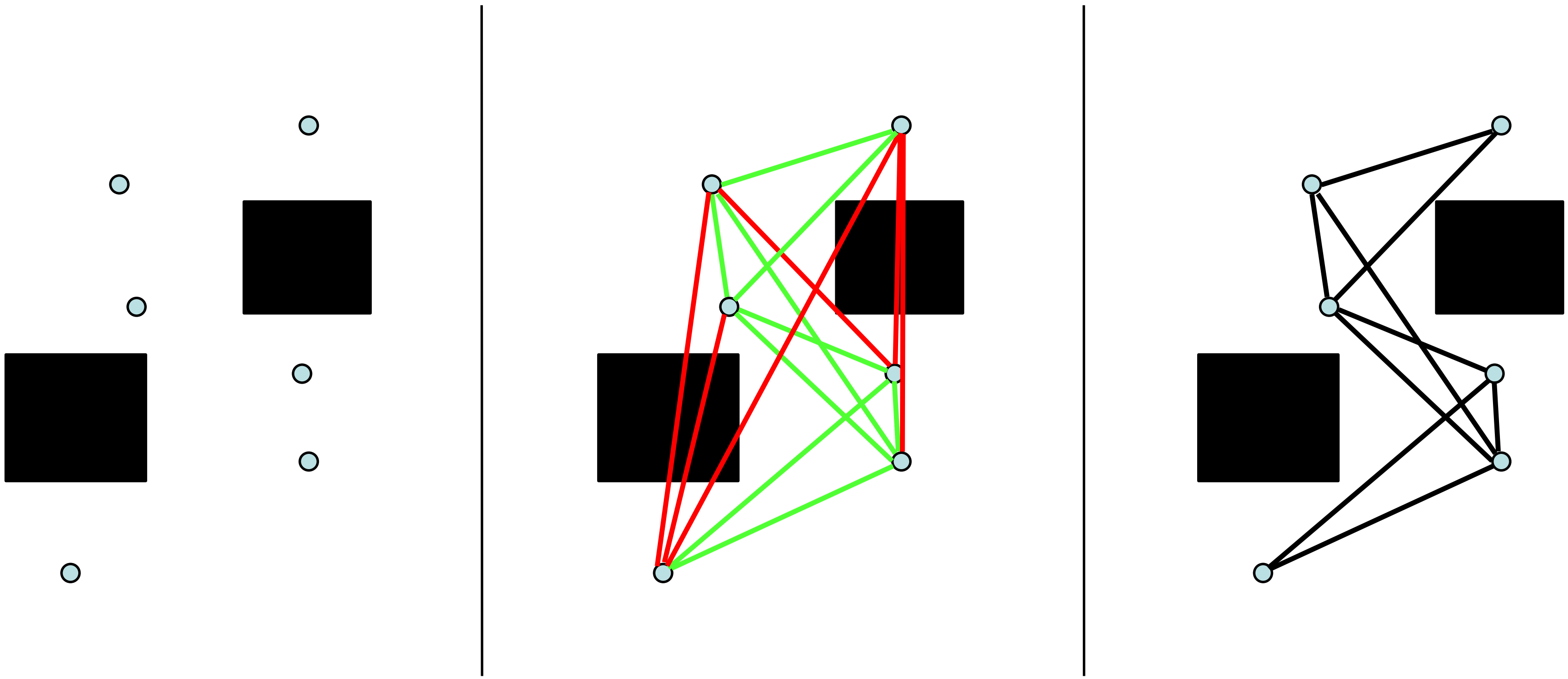
Probabilistic Roadmaps

- **Probabilistic roadmaps** (PRM):
  - Construction phase
    - **Randomly** generates a large number of configurations in free space
    - Builds a graph
  - Query phase
    - Searches the graph

- **Properties**:
  - Scales better to higher dimensions
  - Deterministically incomplete, **probabilistically complete**
    - The more configurations you create,
      the greater the probability that a path can be found if possible
      (approaching 1.0)

PRM: Construction phase

- Visualization in 2D…

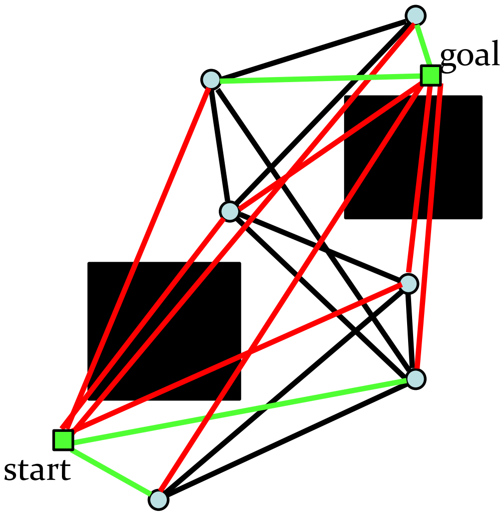


Generate random configurations

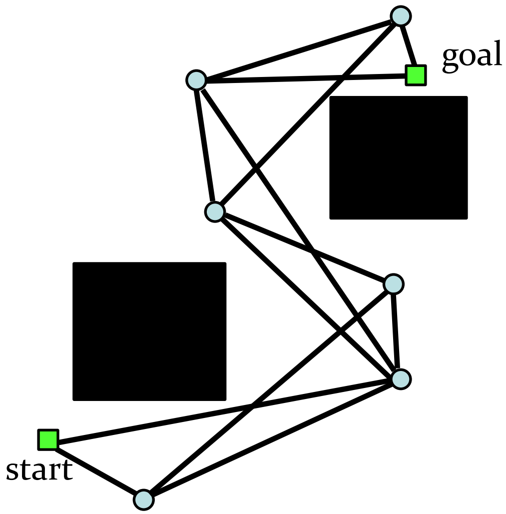
Make connections

Many methods for node placement, emphasizing narrow passages, …

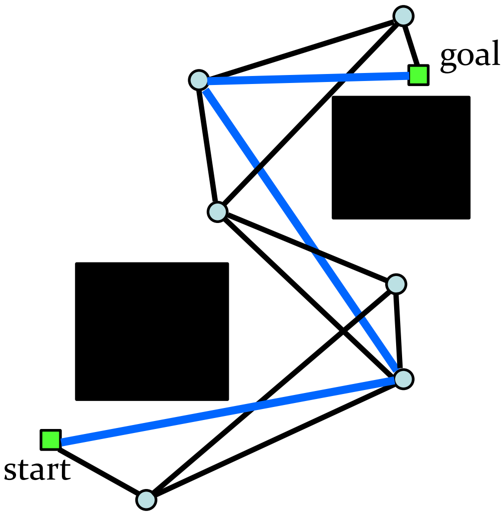
PRM: Query Phase



Add start and
goal configurations
to the roadmap

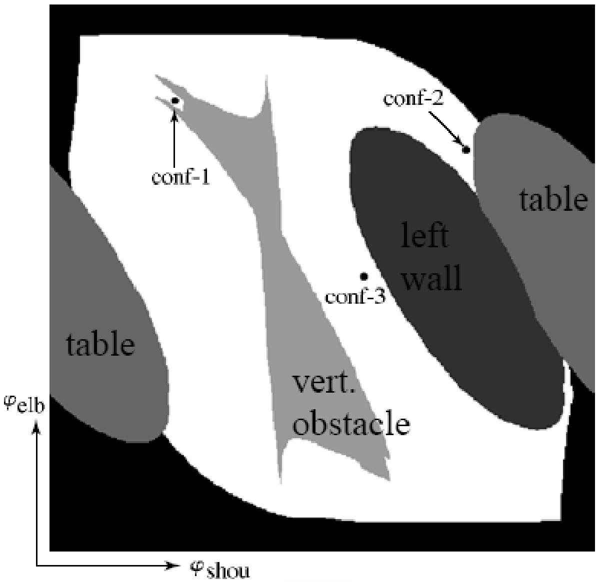


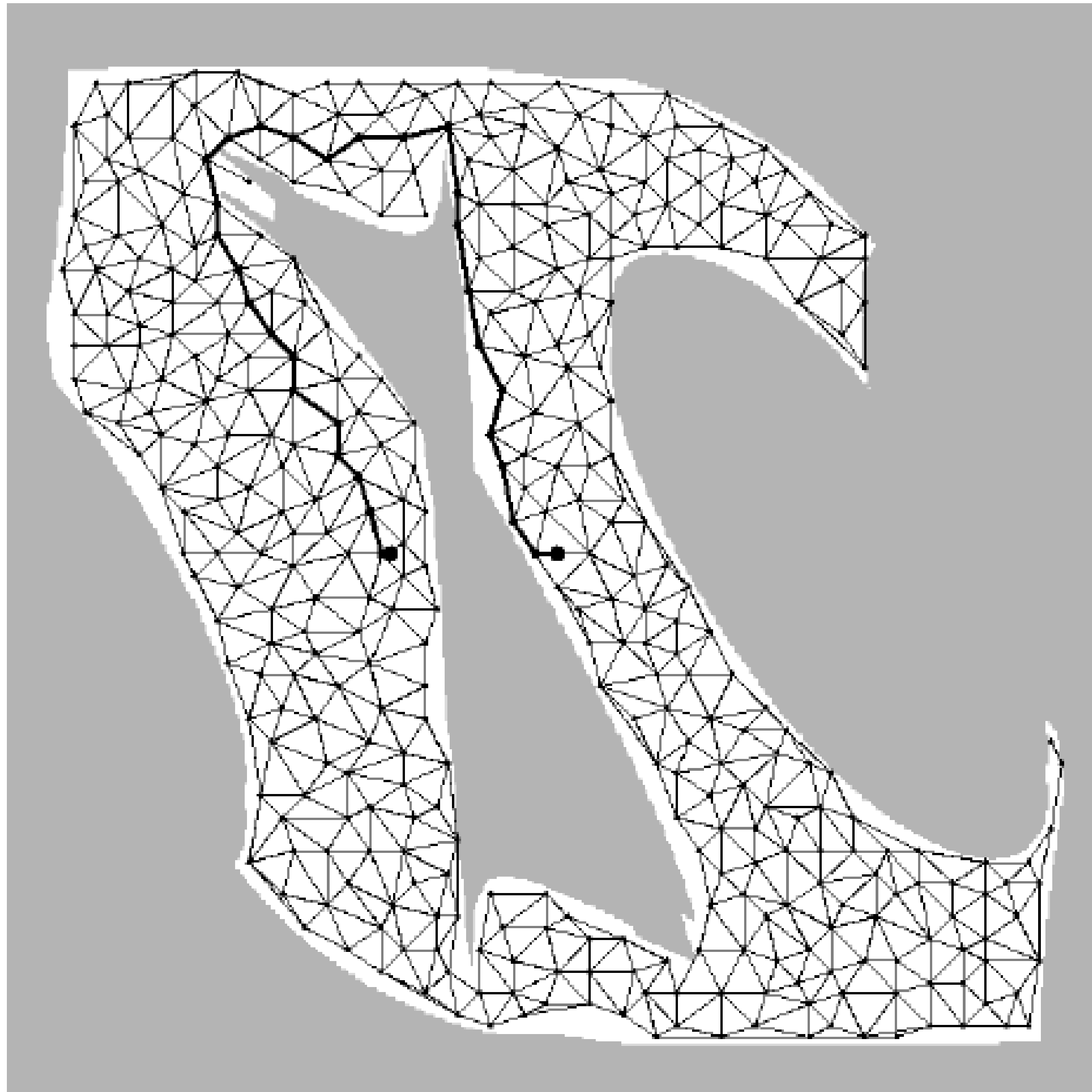
A* search



Curve Replacement &
Smoothing

PRM: Result



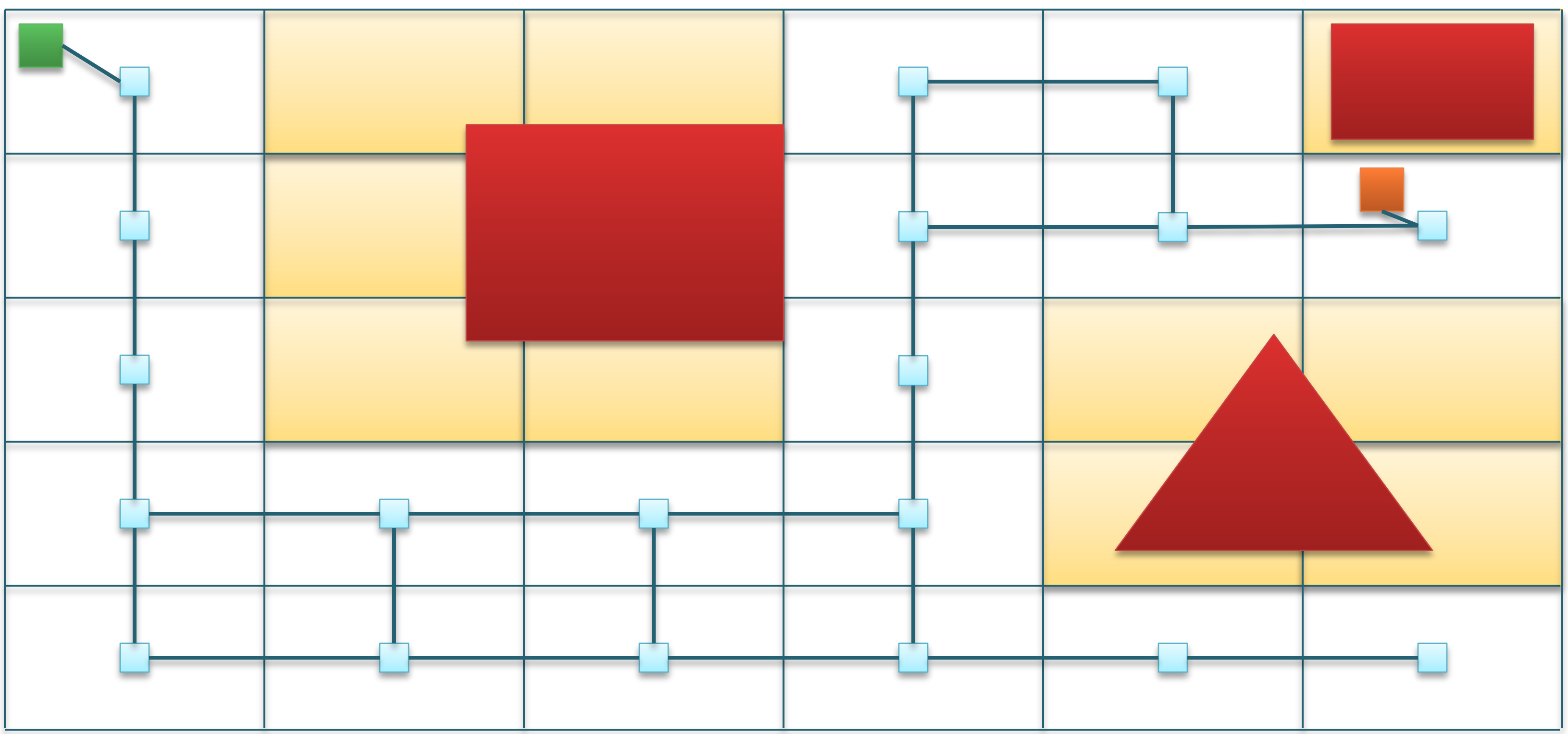


Visualized i 2D
Could be 25D

Graph Search

Graph Search (1)

- Given a discretization, how do we find a path?
  - One option: A*
    - Heuristics: Manhattan distance (moving in 4 directions), Chebyshev distance (moving in 8 directions), Euclidian distance (in general), …



Graph Search (2)

- Suppose **new obstacles** are detected during execution
  - A*: Update map and replan from scratch
    - Inefficient
  - D* (Dynamic A*): Informed **incremental** search
    - First, find a path using information about known obstacles
    - When new obstacles are detected:
      - Affected nodes are returned to the OPEN list, marked as RAISE: More expensive than before
      - Incrementally updates only those nodes whose cost change due to the new obstacles
  - Focused D*:
    - Focuses propagation towards the robot – additional speedup
  - …

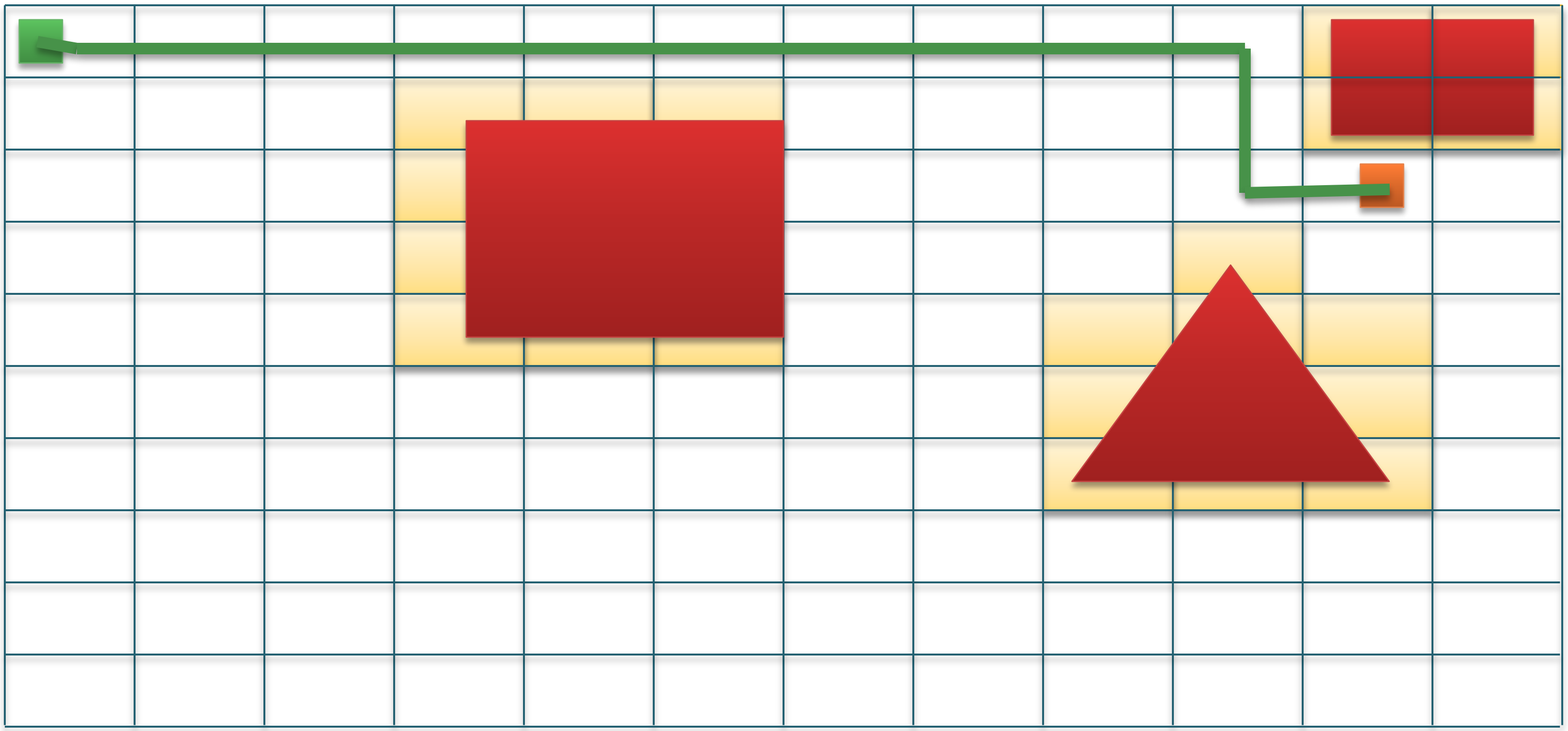
Graph Search (3)

- Anytime algorithms:
  - Return some path quickly, then incrementally improve it
  - ”Repeated weighted A*”
    - Run A* with $f(n) = g(n) + W * h(n)$, where $W > 1$: Faster but suboptimal
    - Decrease W and repeat
    - Has to redo search from scratch in each run!
  - Anytime Repairing A*
    - Like ”repeated weighted A*”, but reuses search results from earlier iterations
  - Anytime Dynamic A* (AD*)
    - **Both** replanning when problems change and anytime planning
  - …

Path Smoothing

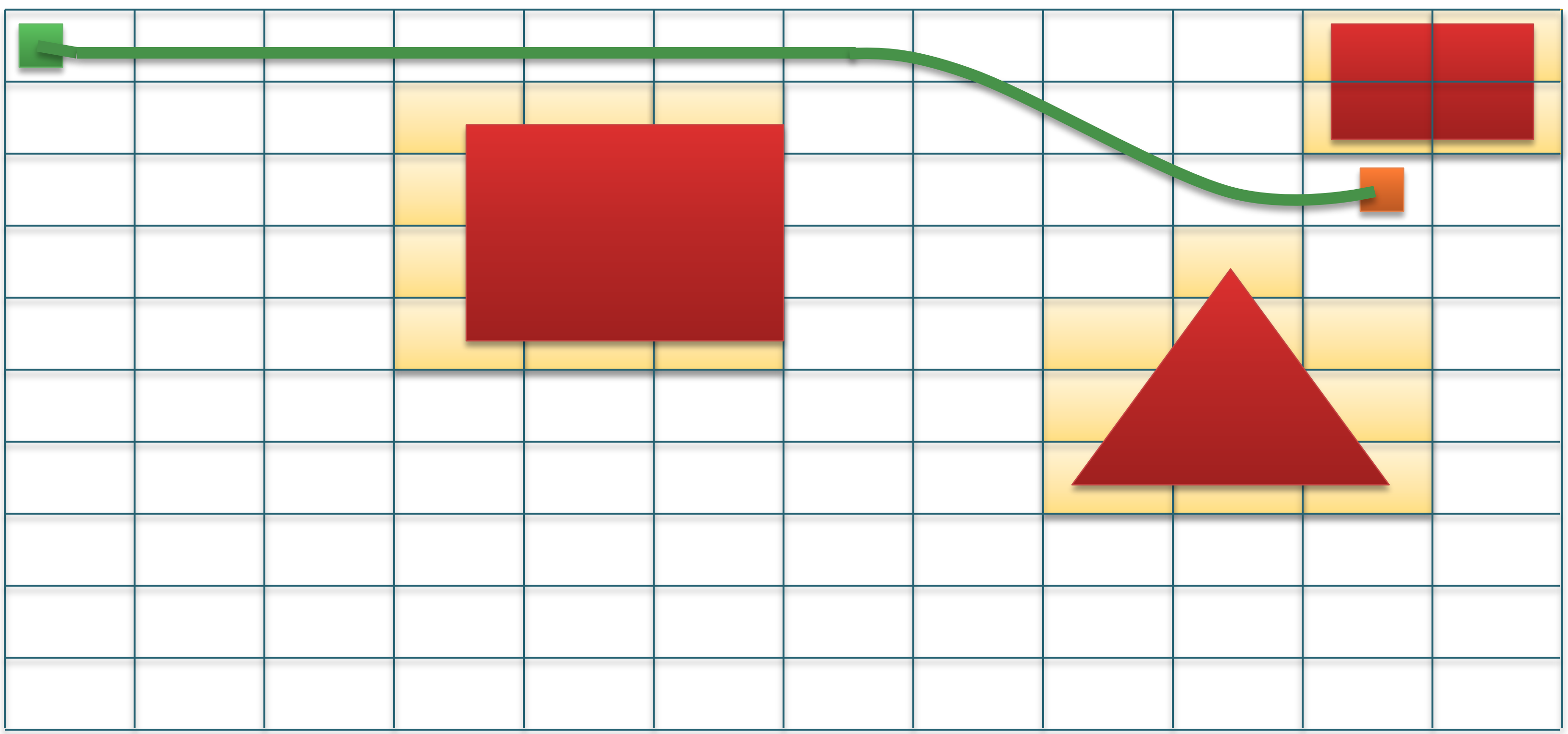
Suboptimal Paths

- Paths are often **suboptimal** in the continuous space
  - Only the chosen points in the cells are used
  - In this example: The midpoints



Smoothing

- Paths can be improved through <u>**smoothing**</u> after generation
  - Still generally does not lead to optimal paths
  - This is just a simple example, where smoothing is easy



Open Motion Planning Library

- Want to experiment?
  - Open Motion Planning Library
  - http://ompl.kavrakilab.org/index.html