Aalto University
School of Electrical
Engineering

Graph-based Path Planning

Graphs to the rescue!!

Dr. Kshitij Tiwari

Dept. of Electrical Engineering and Automation
kshitij.tiwari@aalto.fi

March 26, 2019

Overview

Biography

- ***Bachelors:*** Electronics & Comm. Engg. (HKU,2013)
- ***Masters:*** Artificial Intelligence (UoE,2014)
- ***Ph.D.:*** Robotics (JAIST,2018)
- ***Postdoc:*** Brain-inspired Robotics (Aalto,Present)

Let's get to business!!!

Motivation

- Definition of a robot ✓
- Robot positioning (Localization) ✓
- Path planning . . .

Motivation (cont.)

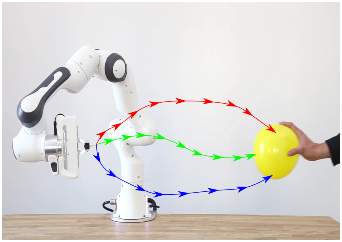


Figure: Robotic arm.

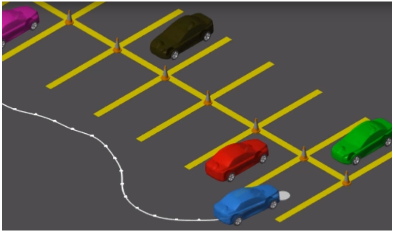


Figure: Autonomous cars.

All ***mobile*** robots need to plan paths.

About

- Introductory lecture for graph-based path planning approaches.
- Includes several visual props.
- ***Pop Quizzes!!*** to be discussed on *MyCourses*.
  - For self-learning and are ***NOT*** graded.
  - ***DL:*** Lecture 12, April 3, 2019 @ 1700 Hrs.

Graph-based Path Planners

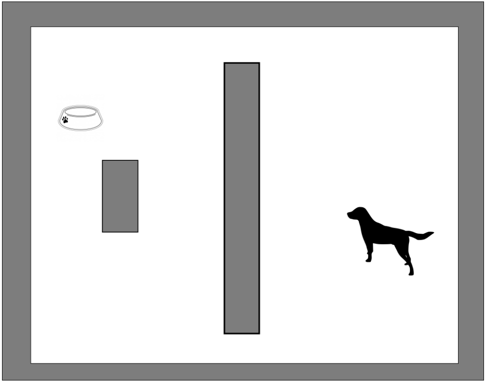


Figure: Dog and the food scenario.

Graph-based Path Planners (cont.)

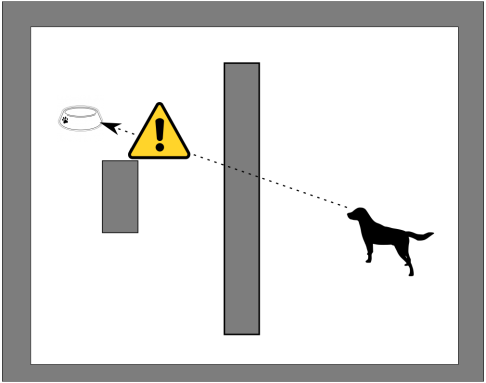


Figure: Obstructions to direct path to food.

Graph-based Path Planners (cont.)

- Direct path blocked by obstacles.
- Need to *find* path to goal.

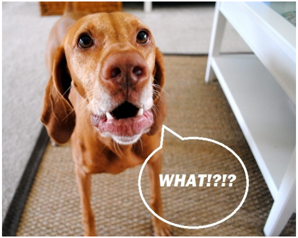


Figure: Grrrr . . . I will heck you up hooman.

Need a *managable* representation of search area !!!

Graph-based Path Planners (cont.)

- Discretizing the search space.
  - Pixelation using pixels.
  - Tesselation using **Square**/Hexagon/Triangles/Circles. ✓
- Representing grid as a graph.
- Finding paths on graphs.

Graph-based Path Planners (cont.)

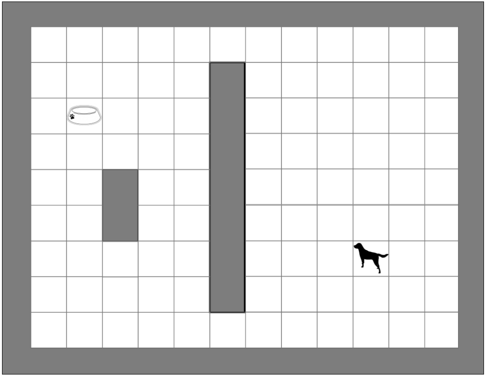


Figure: Simplifying search area ⇒ Grid World.

Graph-based Path Planners (cont.)

Graph descriptors:

- **Nodes:** The locations represented by circles.
- **Edges:** The lines connecting the nodes.
- **Path:** The collection of edges and nodes which define paths from start to goal.

Let's look at basic types of graphs.

Graph-based Path Planners (cont.)

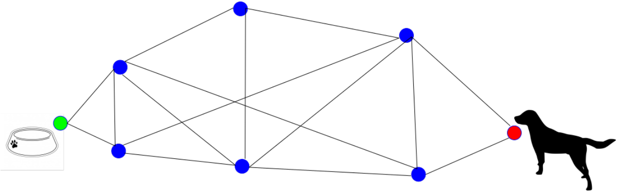


Figure: Fully connected graphical representation with bi-directional edges.

Graph-based Path Planners (cont.)

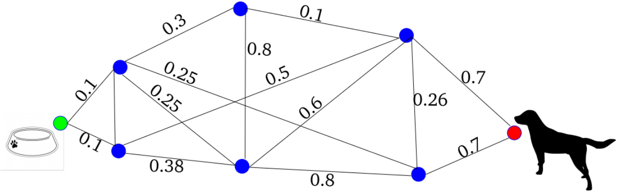


Figure: Weighted graphical representation.

Graph-based Path Planners (cont.)

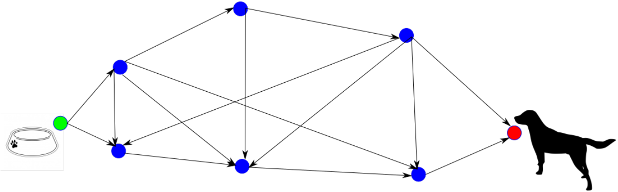


Figure: Directed (Acyclic) graphical representation.

Graph-based Path Planners (cont.)

Remarks:

- Graph-based approaches deal with **graphs** only.
- They do not care about:
  - Indoor/outdoor
  - Walls/doors/windows
- They only care about:
  - Obstacles
  - Free space
- $Graph \rightarrow Path$. Process graph to find **path**
- **Edges** are abstract concepts.
  - Explain which nodes are reachable but NOT how.

Graph-based Path Planners (cont.)

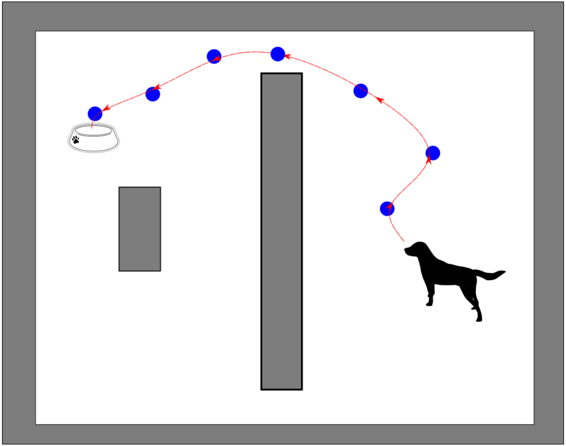


Figure: Graphical representations of scenario.

Graph-based Path Planners (cont.)

Path finding in graphs can be done via:

1. **Best-first Search Methods**
   - Dijkstra:
     - prioritizes exploration over low-cost paths.
   - A*
     - bias search towards goal using heuristics.
   - D*
     - using heuristics and tackle dynamic obstacles.
2. **Sampling Based Methods**
   - RRT
     - Single-query Sampling based search.
   - PRM
     - Multi-query Sampling based search.

Let's dive deeper!!

Best-first Search Methods

Dijkstra Algorithm

- Calculates ***shortest*** path.
  - Chosen root node to every other node in graph.
- Needs weighted graphs.

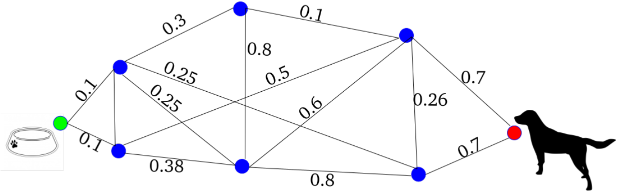


Figure: Recap: Weighted graphical representation.

Best-first Search Methods (cont.)

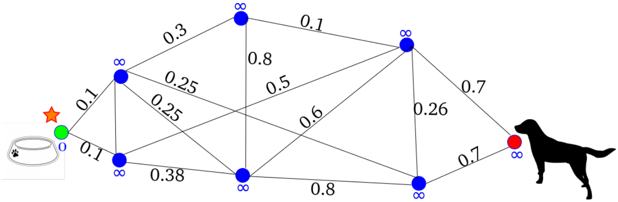


Figure: Step 1. Select a node and mark with star symbol. Mark costs of selected node as 0 and others as ∞.

Best-first Search Methods (cont.)

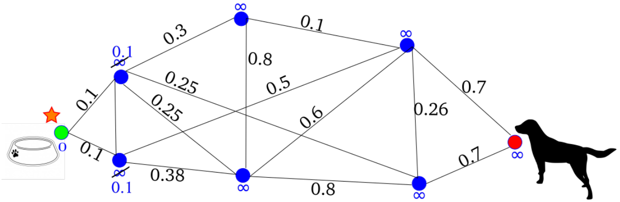


Figure: Step 2. Associate cost for neighbors of selected node.

Best-first Search Methods (cont.)

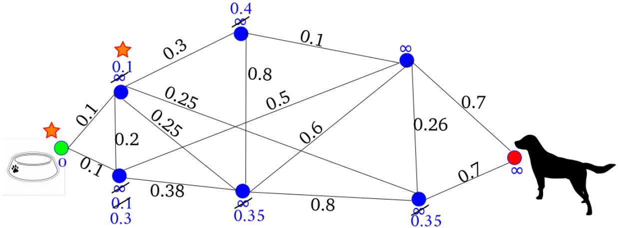


Figure: Step 3. Choose the cheapest node and repeat while ignoring visited nodes. Pick any in case of identical cost.

Best-first Search Methods (cont.)

Pop-Quiz (PQ1)!!

- What is the **shortest** path for doggo based on this graph?
- What is the net cost?

Best-first Search Methods (cont.)

$\boldsymbol{Dijkstra}(StartNode, G)$

1: **Input**:

- $StartNode$: node representing start position
- G : entire graph of state space

2: **Output**:

- $ShortestPath$: list of node that form shortest path from start to goal

Best-first Search Methods (cont.)

$Visited \leftarrow StartNode; UnVisited \leftarrow G\ StartNode$
do
  $CurrNode \leftarrow \arg\min_{\forall u \in UnVisited} \texttt{DIST}(Visited[-1], u)$ ▷ Cheapest node as current node
  $NN \leftarrow \texttt{Neighbors}(CurrNode, G)$ ▷ Extract 1-hop neighbors
  $\forall n \in NN \leftarrow min\texttt{Cost}(n), \texttt{Cost}(CurrNode, n)$ ▷ Update Costs
  $Visited \leftarrow \cup\ CurrNode$ ▷ Mark current node as visited
while $UnVisited! = NULL$ ▷ Repeat until all nodes visited

Best-first Search Methods (cont.)

Figure: Showcasing the progress of Dijkstra. **N.B.:** Lot of wasteful steps. Image ©: Subhrajit Bhattachary

Best-first Search Methods

A Algorithm*

Overview:

- Dijkstra wastes a lot of steps
  - Expansions not always towards the goal
- Some method to bias the expansion of needs required
  - Use ***heuristics*** to bias the expansion

This heuristic expansion is then called A^* algorithm.

Best-first Search Methods (cont.)

Components for A* based planning:

- Definition of valid **actions**.
- Definition of *grid resolution*.
- Definition of **cost** associated with movement.

Best-first Search Methods (cont.)

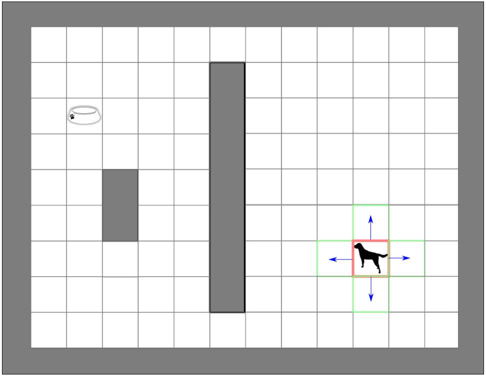


Figure: Valid actions from a grid cell. No diagonal movements for simplicity.

Best-first Search Methods (cont.)

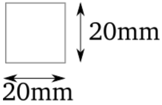


Figure: Arbitrary grid resolution.

Best-first Search Methods (cont.)

The *path cost* encompasses two kinds of costs:

1. **Transition Cost:**
   The cost incurred when moving from one grid to its immediate neighbor.
2. **Net Estimated Cost:**
   This cost is an estimate (heuristic) of the overall path cost to be incurred from start to goal.

Best-first Search Methods (cont.)

Let us look at a worked example: Helping the dog get to its food.

Notational Convention:

- **Bottom Left:** Transition cost from start to current grid
- **Bottom Right:** Heuristic cost from current node to goal using *Manhattan distance*

Best-first Search Methods (cont.)

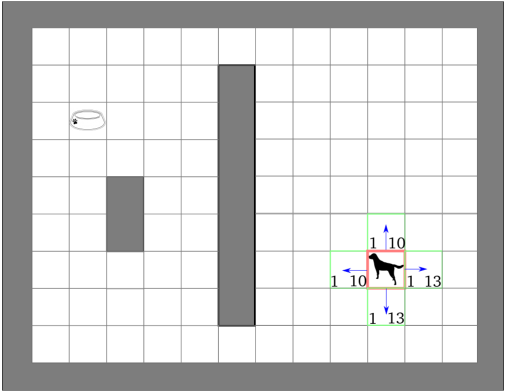


Figure: Step 1. 2 cells have same cost. Pick *LEFT*.

Best-first Search Methods (cont.)

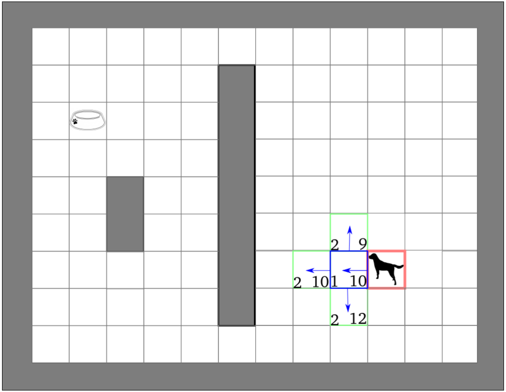


Figure: Step 2. Action *UP* has lowest cost.

Best-first Search Methods (cont.)

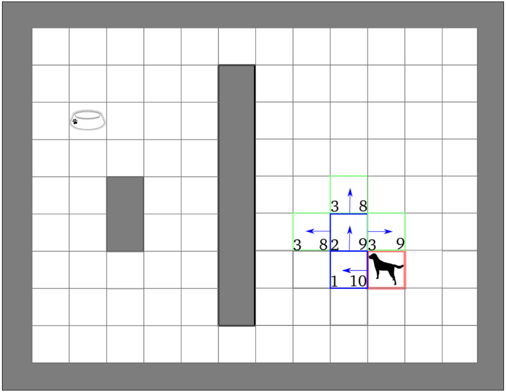


Figure: Step 3. Tied low-cost so, continue with previous action:*UP*

Best-first Search Methods (cont.)

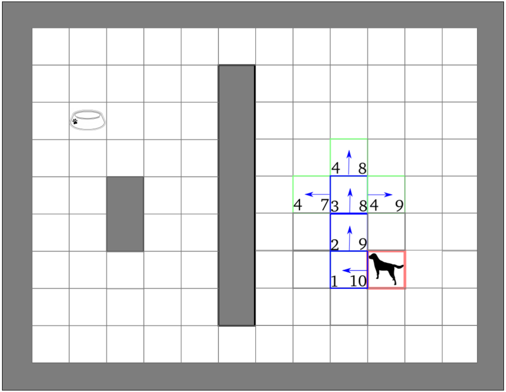


Figure: Step 4. Action:*LEFT*

Best-first Search Methods (cont.)

Pop-Quiz (PQ2)!!

- How many steps finally?
- Final cost?
- Final path?
- Type of graph?

Best-first Search Methods (cont.)

Need to find the *shortest* path to goal autonomously !!!

- Use A* algorithm.
- Additionally, define 2 kinds of nodes:
  - **Open Node:**
    consists on nodes that have been visited but not expanded (meaning that sucessors have not been explored yet). This is the list of pending tasks.
  - **Close Node:**
    consists on nodes that have been visited and expanded (sucessors have been explored already and included in the open list, if this was the case).

Best-first Search Methods (cont.)

A*$(StartNode, GoalNode)$

Input:
- $StartNode$: node representing start position
- $GoalNode$: node representing goal position

Output:
- $ShortestPath$: list of node that form shortest path from start to goal

$OpenList = []; ClosedList = []; ChildNodes = []$

$OpenList \leftarrow StartNode$ ▷ Store Start Loc

do

  $CurrNode \leftarrow \arg\min_{\forall n \in OpenList} \texttt{Cost}(StartNode, n)$

  $OpenList \leftarrow CurrNode$ ▷ Remove from OpenList

  $ClosedList \leftarrow CurrNode$

Best-first Search Methods (cont.)

if $CurrNode == GoalNode$ **then**
    `print`("Arrived at Goal")
    `break` ▷ Terminate
end if
$ChildNodes \leftarrow$ `GenChild`$(CurrNode)$
for $\forall c \in ChildNodes$ **do**
    if $c \in ClosedList$ **then**
        `pass`
    end if
    $c.f, c.t, c.e \leftarrow Cost(StartNode, c, GoalNode)$
    if $c \in OpenList$ **then**
        if $c.t > on.t \forall on \in OpenList$ **then**
            `pass`
        end if

Best-first Search Methods (cont.)

end if

$OpenList \leftarrow c$ ▷ Store child

end for

while $OpenList! = NULL$

Return `ShortestPath`$(StartNode, GoalNode, ClosedList)$

Best-first Search Methods (cont.)

$\boldsymbol{Cost}(StartNode, CurrNode, GoalNode, StepCost = 1)$

1: **Input**:

- $StartNode$: node representing start position
- $CurrNode$: node representing current position
- $GoalNode$: node representing goal position
- $StepCost$: fixed cost of transitioning to immediate neighbors (children)

2: **Output**:

- f : sum of transition and net estimated costs from current node
- t : transition cost
- e : net estimation cost

Best-first Search Methods (cont.)

$t = \texttt{DIST}(CurrNode, StartNode) + StepCost$

$e = \texttt{DIST}(CurrNode, GoalNode)$

$f = t + e$

Return $\{f, t, e\}$

Best-first Search Methods (cont.)

GenChild$(CurrNode)$

1: **Input**:

- $CurrNode$: node representing current position

2: **Output**:

- $ChildNodes$: all adjacent (valid) nodes of current node

3: $ChildNodes \leftarrow$ `isNeighbor`$(CurrNode)$ ▷ Get all children

Best-first Search Methods (cont.)

Figure: Showcasing the progress of A*. Image ©: Subhrajit Bhattachary

Best-first Search Methods (cont.)

Pop-Quiz (PQ3)!!

In Romeo and Juliet, Shakespeare said “What’s in a name?”. If there is nothing, is A* just an arbitrary name?

Best-first Search Methods

D Algorithm*

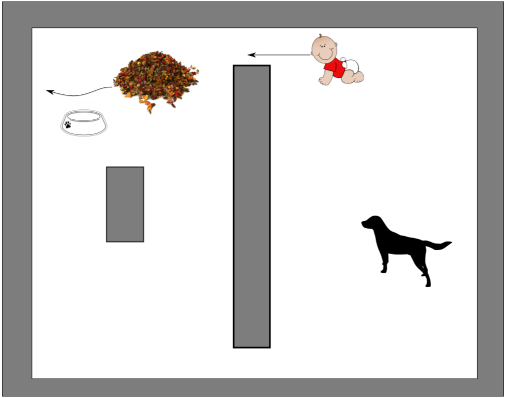


Figure: Dog and the food but with *dynamic* obstacles also present.

Best-first Search Methods (cont.)

- Presence of *dynamic* obstacles makes planning challenging.
- A* re-planning is costly.
  - **Solution:** D* Search Algorithm

Best-first Search Methods (cont.)

D*$(StartNode, GoalNode)$

Input:

- $StartNode$: node representing start position
- $GoalNode$: node representing goal position

Output:

- $ShortestPath$: list of node that form shortest path from start to goal

$OpenList = []; ClosedList = []; ChildNodes = []$

$OpenList \leftarrow StartNode$ ▷ Store Start Loc

do

    $CurrNode \leftarrow \arg\min_{\forall n \in OpenList} \texttt{Cost}(StartNode, n)$

    $OpenList\ CurrNode$ ▷ Remove from OpenList

    $ClosedList \leftarrow CurrNode$

Best-first Search Methods (cont.)

```
    if CurrNode == GoalNode then
        print("Arrived at Goal")
        break                                        ▷ Terminate
    end if
    ChildNodes ← GenChild(CurrNode)
    for ∀c ∈ ChildNodes do
        ...                                          ▷ Same as A*
    end for
    if EnvChange() then                              ▷ If environment changed
        ChildNodes ← GenChild(CurrNode)              ▷ Revise nodes
        UpdateCost(ChildNodes)                       ▷ Update affected children
    end if
while OpenList! = NULL
Return ShortestPath(StartNode, GoalNode, ClosedList)
```

Best-first Search Methods (cont.)

Remarks:

- D* is quite similar to A* but . . .
  - Allows changes in environments
  - Recalculation only for affected nodes

Best-first Search Methods (cont.)

Pop-Quiz (PQ4)!!

Is D* just an arbitrary name?

Best-first Search Methods

Pop-Quiz (PQ5)!!

- If you lookup the literature, you will find the mention of ***Configuration Spaces*** and ***Work Spaces***. Can you explain the difference(s) between them?
- How do they relate to the doggo example?

Sampling Based Methods

RRT

- Search-space size is a bottleneck for A* and D*.
- *E.g.:*
  - Dog navigation from 2D to UAV navigation in 3D.
  - 7DOF robot arm navigation in ND.
- *Solution:*
  - Uniform Grid $\rightarrow$ random sampling
  - Rapidly-exploring Random Trees a.k.a. *RRT*

Sampling Based Methods (cont.)

RRT Jargon:

- **Root:** Just like the root of tree (start node).
- **Goal:** Goal node which must be reached eventually.
- **Rand:** Randomly generated/sampled node in space to expand tree.
- **Next:** Closest node from tree to the **Rand** node in its direction.

Sampling Based Methods (cont.)

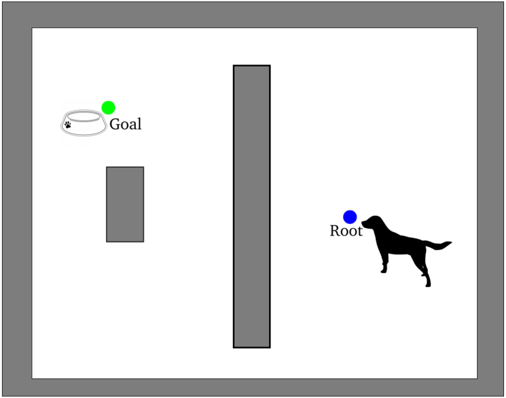


Figure: RRT setup for dog-food scenario. Root node shown in blue and Goal node shown in green.

Sampling Based Methods (cont.)

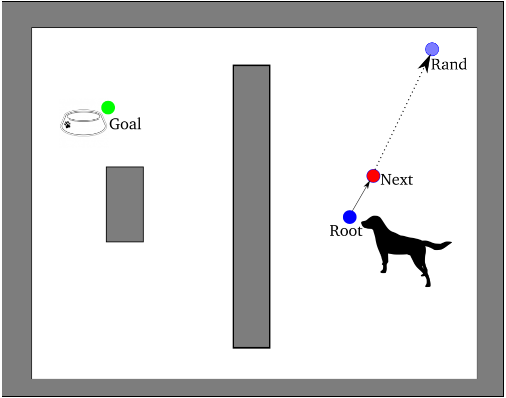


Figure: Step 1. Generate random sample and find the next/nearest node from free.

Sampling Based Methods (cont.)

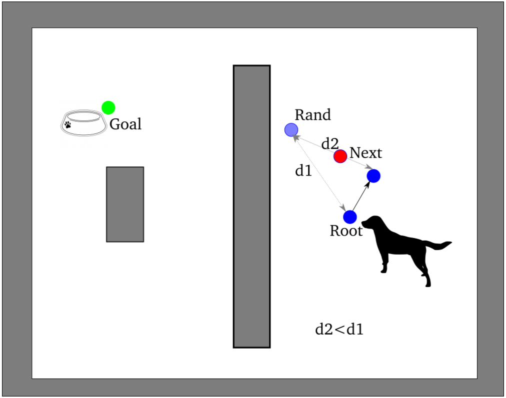


Figure: Step 2. Generate new random sample and repeat Step 1 with nearest neighbor.

Sampling Based Methods (cont.)

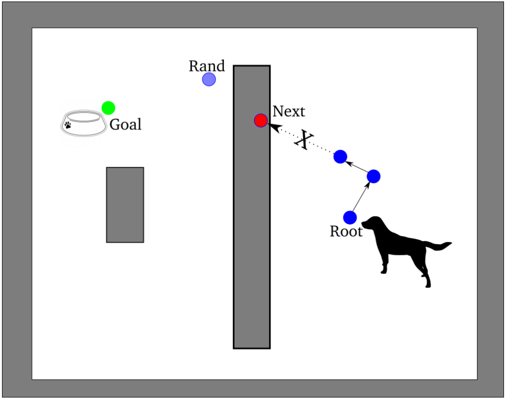


Figure: Step 3. Generate new random sample and ignore if obstructed.

Sampling Based Methods (cont.)

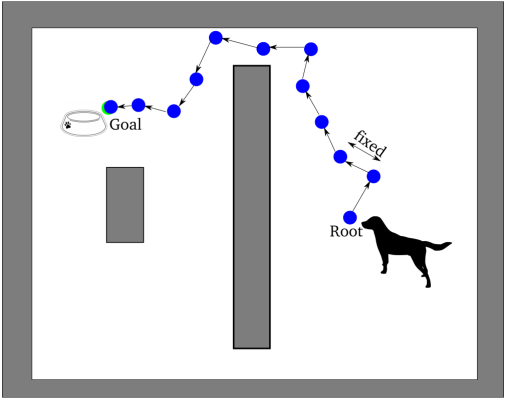


Figure: Step N. Final tree (arrival at Goal).

Sampling Based Methods (cont.)

$\boldsymbol{RRT}(StartNode, GoalNode, NumVert, StepSize)$

1: **Input**:

- $StartNode$: node representing start position (root)
- $GoalNode$: node representing goal position
- $NumVert$: number of vertices in tree
- $StepSize$: incremental distance for tree expansion

2: **Output**:

- G : RRT Graph

Sampling Based Methods (cont.)

$G \leftarrow StartNode$
for $v = 1, \ldots, NumVert$ **do**
    $Rand \leftarrow$ `GetRandVert()` ▷ Get random config
    $Next \leftarrow$ `GetNearestVert(Rand,G,StepSize)` ▷ Get Nearest vertex
    if $Next \in$ `FreeSpace` **then**
        $G \leftarrow Next$ ▷ Move if free
    end if
end for

Sampling Based Methods (cont.)

Figure: Showcasing the progress of RRT. Image ©: Steven LaValle

Sampling Based Methods (cont.)

Remarks:

- RRT is *single-query*
  - Rooted at start location of dog.
  - Dog moves $\rightarrow$ renew tree from new root.
  - Goal moves $\rightarrow$ renew tree.
  - Multiple goals $\rightarrow$ handle one start-goal configuration at a time.
  - Valid only for one-time querying.
  - Static obstacles only.

Sampling Based Methods

PRM
Overview:

- Stands for *Probabilistic Roadmaps*.
- are *multi-query* in nature.
  - Roadmaps “rooted” to environment configurations.
  - Independent of location of dog.
- Randomly samples points in state space and connects peers.
- Static environments only.

Sampling Based Methods (cont.)

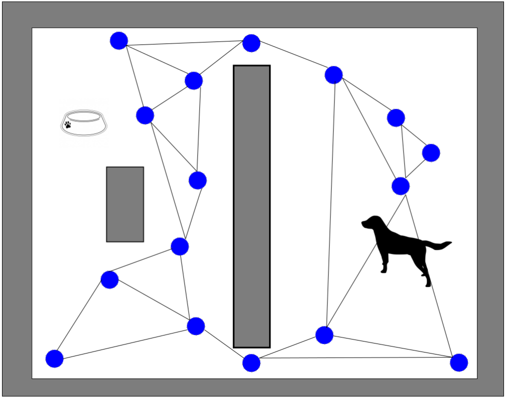


Figure: PRM setup. Roadmap around static obstacles irrespective of dog and food location.

Sampling Based Methods (cont.)

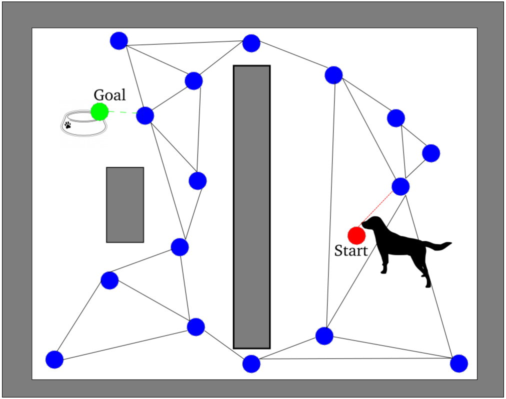


Figure: Connect start and goal nodes to roadmap.

Sampling Based Methods (cont.)

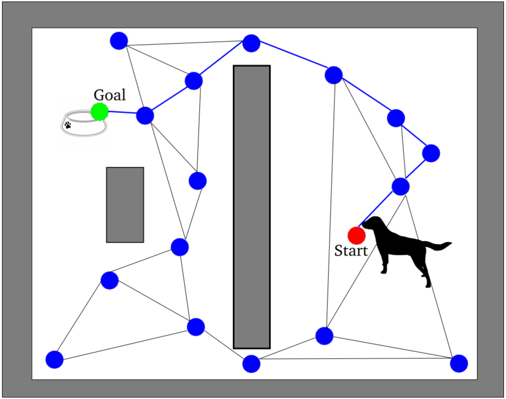


Figure: Find shortest, obstacle-free to food.

Sampling Based Methods (cont.)

PRM$(StartNode, GoalNode, k, NumVert)$

1: **Input**:

- $StartNode$: node representing start position (root)
- $GoalNode$: node representing goal position
- k : number of nearest neighbors
- $NumVert$: number of vertices in tree

2: **Output**:

- R : Roadmap

Sampling Based Methods (cont.)

$R \leftarrow StartNode$
while $|R| < NumVert$ **do** ▷ Comp. constraint
  $Rand \leftarrow$ `GetRandVert()` ▷ Get random config
  if $Rand \in$ `FreeSpace` **then**
    $R \leftarrow Rand$ ▷ Store if free
  end if
end while
for $v = 1, \ldots, |R|$ **do**
  $kNN \leftarrow$ `GetNeighbors`$(k, R[v])$ ▷ Get k-neighbors
  for $\forall nn \in kNN$ **do**
    if $(v, nn) \nexists E$ **and** $\Delta(v, nn) \neq NULL$ **then**
      $R \leftarrow nn$ ▷ Store if valid and novel
    end if
  end for

Sampling Based Methods (cont.)

17: **end for**

Pop-Quiz (PQ6)!!

What is “probabilistic” about PRMs?

Summary

- Introduction to Graph-search
- Exposure to Best-first search approaches
  - Dijkstra
  - A*
  - D*
- Exposure to Sampling-based approaches
  - RRT
  - PRM
- Worked examples and pseudo-codes.
- Animations as visual props to ease understanding.
- Several other variants not covered herewith.
- Several pop quizzes for open discussion via *MyCourses*

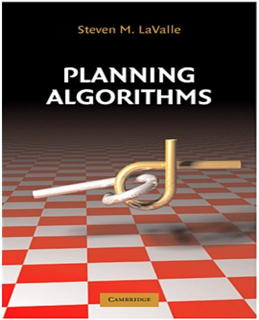
Cliff Hanger

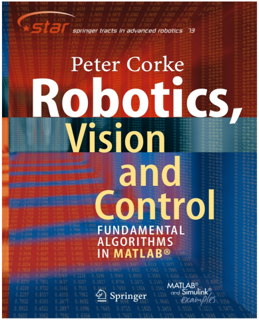
Ever seen a dog perform graph-search?

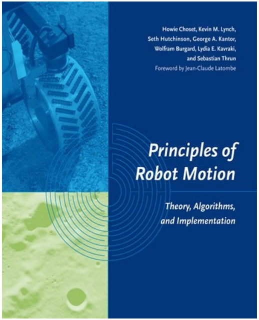
Pop-Quiz (PQ7)!!

How does one represent the dog-sniffing behavior?

Books


Steven M. LaValle
PLANNING ALGORITHMS
CAMBRIDGE


star springer tracts in advanced robotics 73
Peter Corke
Robotics, Vision and Control
FUNDAMENTAL ALGORITHMS IN MATLAB®
Springer
MATLAB® and Simulink® examples


Howie Choset, Kevin M. Lynch,
Seth Hutchinson, George A. Kantor,
Wolfram Burgard, Lydia E. Kavraki,
and Sebastian Thrun
Foreword by Jean-Claude Latombe
Principles of Robot Motion
Theory, Algorithms, and Implementation

Thank You!!