3 – Path planning

Advanced Methods for Mapping and Self-localization in Robotics (MPC-MAP)

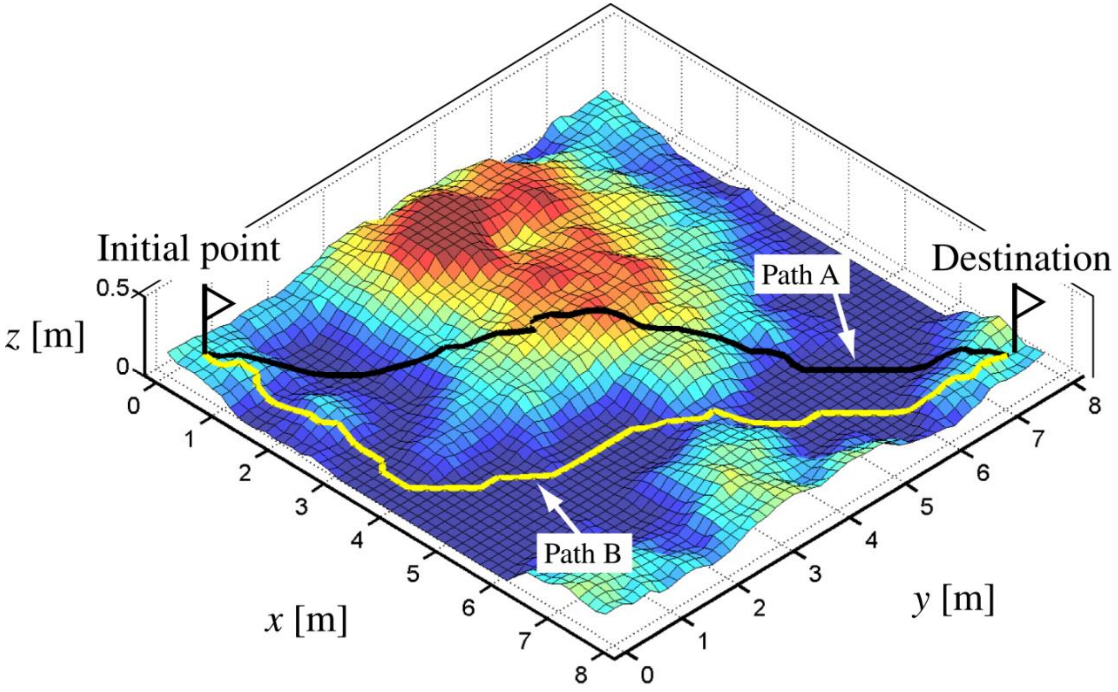
Tomas Lazna

Brno University of Technology
2022

What is path planning for?

- Map, obstacles
- Self-localization
- Where to go?
- How to get there?
- What is the best way?
- What robot do I have?

Path planning = motion planning



[1]

[1] DAHDOUH, Andrew. Graph-Based Path Planning: Dijkstra's Algorithm. In: *Reality Bytes* [online] 2017 [cit. 2021-02-26]. Available at: https://realitybytes.blog/2017/07/11/graph-based-path-planning-dijkstras-algorithm/

Configuration

- A complete specification of the position of every point in the system

Cofiguration space = C-space

- Space of all possible configurations
- A manifold in higher-dimension Cartesian space

C-obstacle

- Set of configurations where the robot collides with a workspace obstacle or with itself

Free configuration

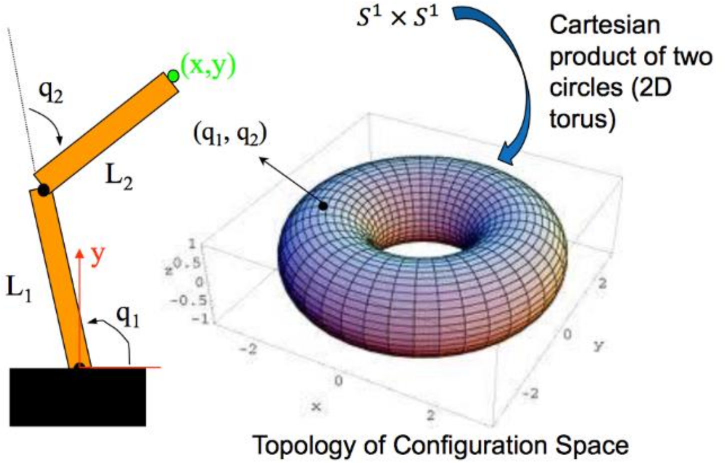
- Null intersection of the configuration with workspace obstacles
- Semi-free = config. touches an obstacle

Inputs

- Initial and goal configurations
- Workspace representations and obstacles
- Allowed actions with associated costs
- (Geometric) Description of robot

Outputs

- Collistion-free sequence of configurations (the shortest / the fastest)

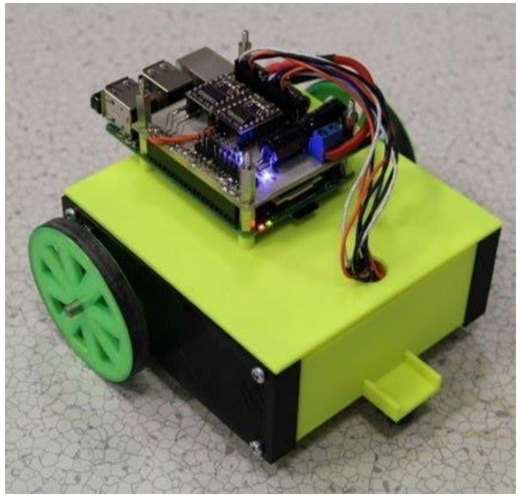


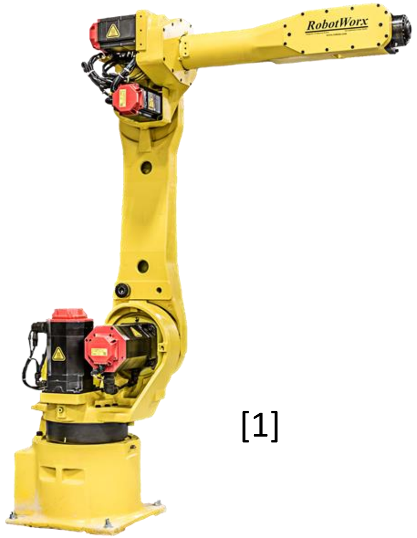
[1]

[1] ATANASOV, Nikolay. Lecture 5: Configuration space [online]. UC San Diego, 2018 [cit. 2021-02-28]. Available at: https://natanaso.github.io/ece276b2018/ref/ECE276B_5_ConfigurationSpace.pdf

Rigid vs. articulated robots

- Rigid: all parts of a robot have constant mutual positions
    - Mobile robots with differential drive
- Articulated: A robot consist of parts connected by rotary joints
    - Industrial manipulators



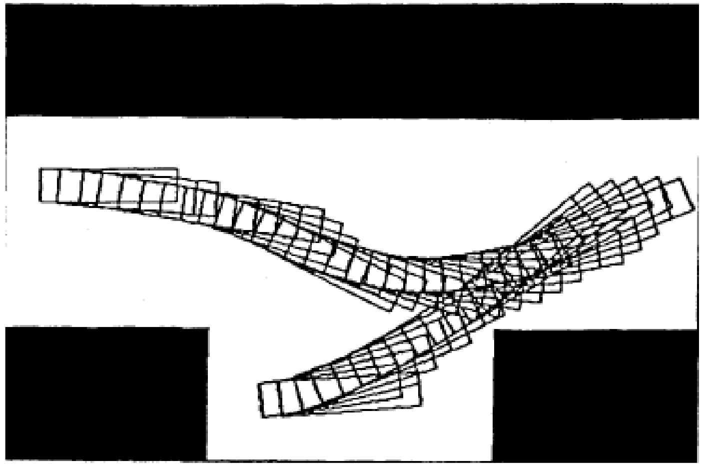


[1]

Holonomic vs. nonholonomic robots

- Holonomic: Has the same or higher number of controllable DOFs than their total number in configuration space
    - Robots with omnidirectional drive
- Nonholonomic: lower number of controllable DOFs
    - Cars (Ackermann drive)

[2]



[3]

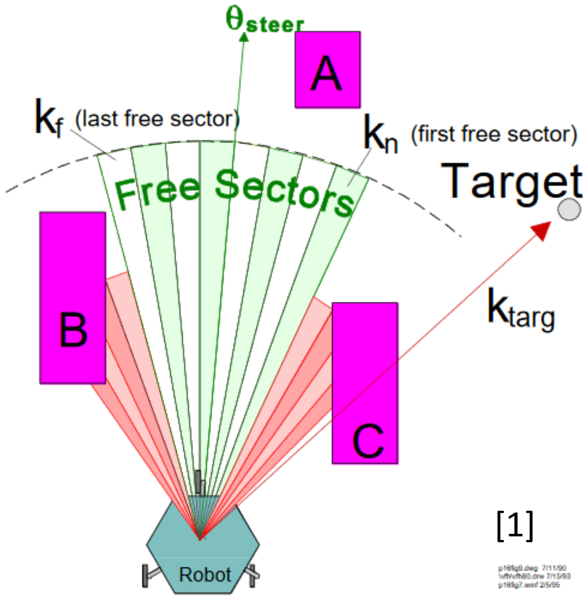
[1] What is a robot manipulator? [online]. RobotWorx, 2021 [cit. 2021-02-28]. Available at: https://www.robots.com/faq/what-is-a-robot-manipulator
[2] Robotics Lab [online]. Nairi-Tech, 2014 [cit. 2021-02-28]. Available at: http://nairi-tech.com/products/view/robotics-lab.html
[3] LATOMBE, Jean-Claude. Class #12: Nonholonomic planning [online]. Stanford University, 2009 [cit. 2021-02-28]. Available at: http://robotics.stanford.edu/~latombe/cs326/2009/class12/class12.htm

Types of planning

Local planning
- Is limited by range of sensors
- Provides a *direction* to move in
- Does not guarantee finding the goal

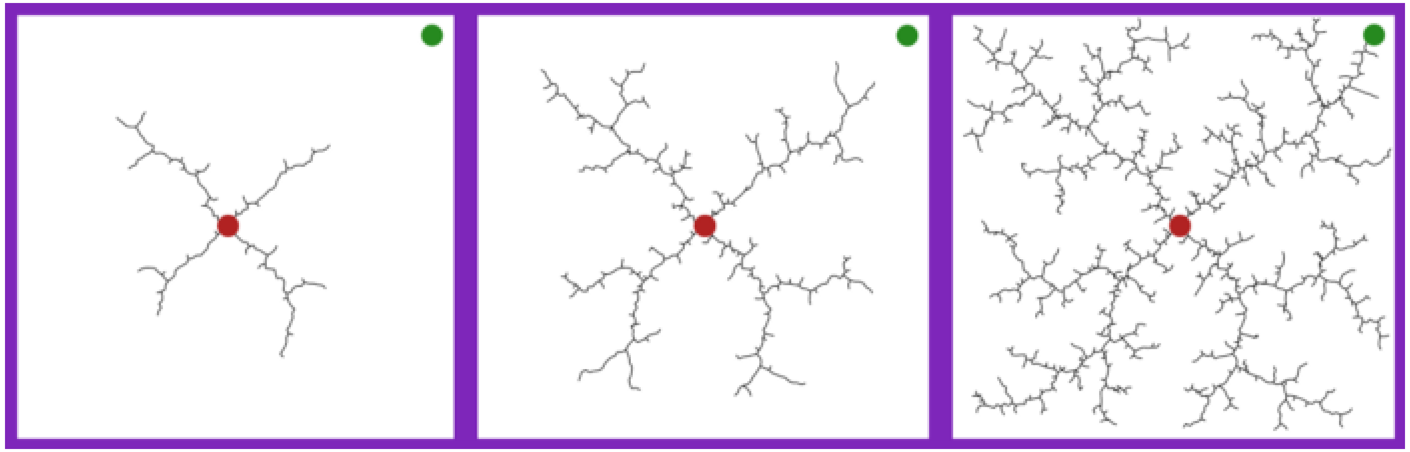
Global planning
- Work with the whole configuration space
- Provides complete path to the goal
- Finds an *optimal* solution
  - Shortest
  - Lowest cost

Stochastic planning
- Monte Carlo methods
- Fast (especially in higher dimensions)
- Not suitable for all scenarios
- Does not find an optimal path



[1]

[2]



[1] BORENSTEIN, J. and Y. KOREN. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation* [online]. **7**(3), [cit. 2021-02-28]. DOI: 10.1109/70.88137
[2] SVEGLIATO, Justin. How does a robot plan a path using RRT? In: *Towards Data Science* [online]. 2020 [cit. 2021-02-28]. Available at: https://towardsdatascience.com/how-does-a-robot-plan-a-path-in-its-environment-b8e9519c738b

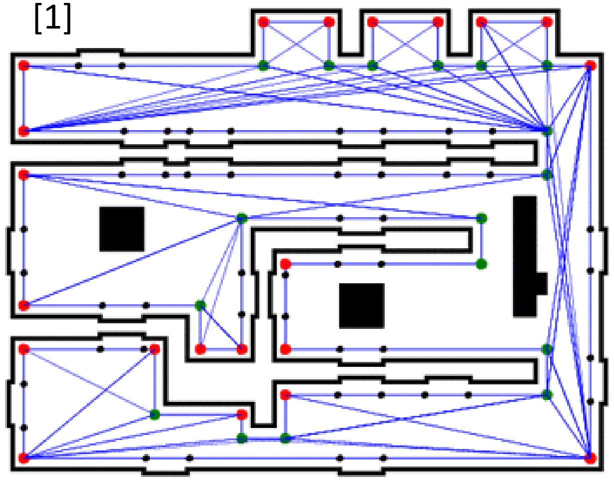
Types of map

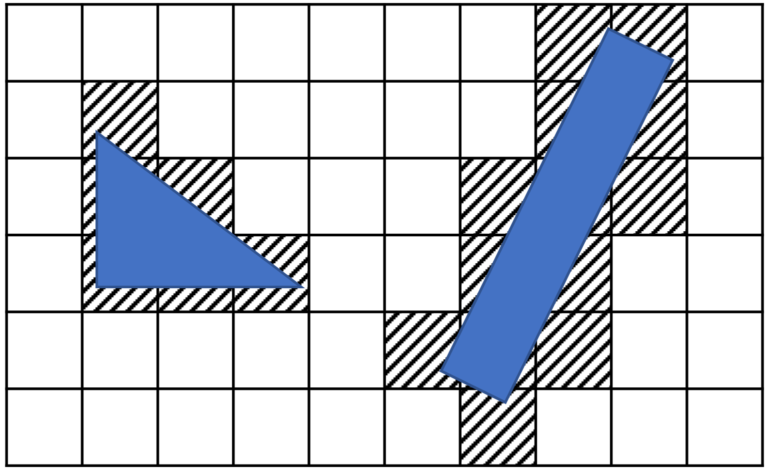
- Continuous vs. discrete
- Metric vs. topological
- Planning algorithm are designed to search in a graph (topological map)

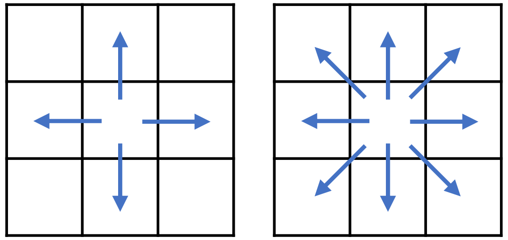
Converting maps to graphs

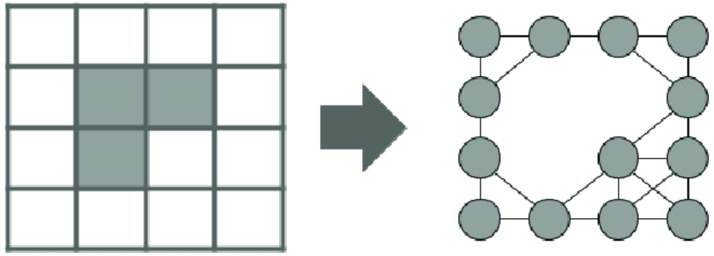
- Continuous → discrete
    - Choose a cell size
    - Cells with not-null intersection with an obstacle are occupied and vice versa
    - Occupancy grid is created
- Discrete → graph
    - Cell adjacency graph
    - 4-connected or 8-connected
- Continuous → graph
    - Visibility graph
    - Voronoi diagrams

[1]

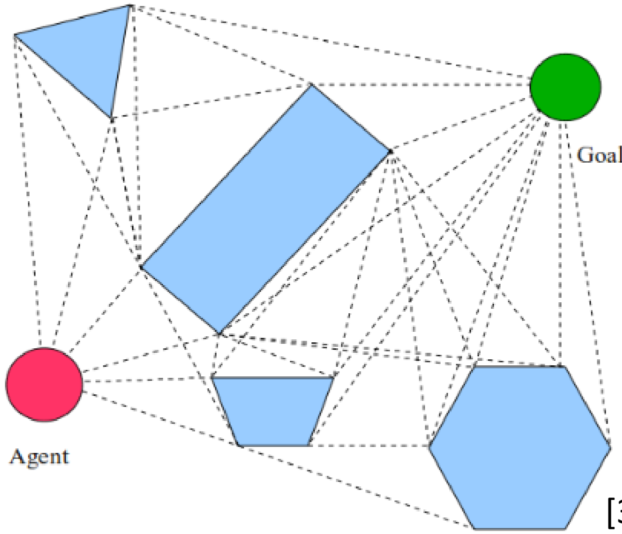








[2]



[3]

[1] BEEVERS, Kris and Wesley H. HUANG. Topological mapping with sensing-limited robots. [online]. Rensselaer Polytechnic Institute, 2005 [cit. 2021-02-28]. Available at: http://cs.krisbeevers.com/mapping.html
[2] DE OLIVEIRA, Guilherme, Kevin DE CARVALHO and Alexandre BRANDÃO. A Hybrid Path-Planning Strategy for Mobile Robots with Limited Sensor Capabilities. *Sensors* [online]. 2019, **19**(5) [cit. 2021-02-28]. DOI: 10.3390/s19051049
[3] KONERU, Haarika. Visibility graphs [online]. Kent State University, 2016 [cit. 2021-02-28]. Available at: http://www.cs.kent.edu/~dragan/ST-Spring2016/visibility%20graphs.pdf

Attributes of planners

Measuring the performance of search algorithms

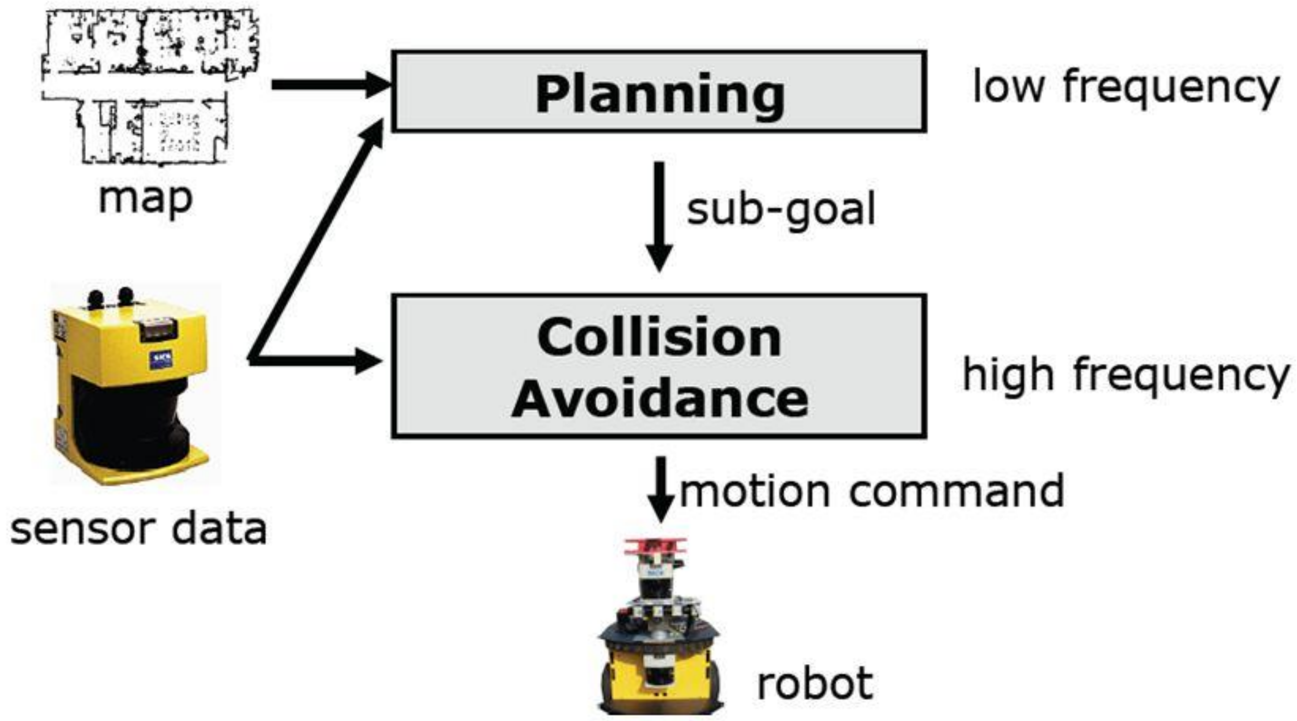
- Completeness: If there is a solution, it is found
- Optimality: The provided solution is the best in terms of some criterion
- Time complexity: Time required to find the solution
- Space complexity: Memory and/or other assets required to find the solution

What other attributes do we require?

- Take uncertainties into account
- React to dynamic obstacles quickly
- Guarantee safety

Layered architecture

Classic Two-Layered Architecture for Mobile Robots



[1]

[1] BURGARD, Wolfram et al. Path Planning and Collistion Avoidance [online]. Uni Freiburg, 2010 [cit. 2021-02-27]. Available at: http://ais.informatik.uni-freiburg.de/teaching/ss10/robotics/slides/16-pathplanning.pdf

Graph search – general algorithm

1	$Q.Insert(x_0)$
2	$x_0.visited = \text{true}$
3	**while** length(Q) > 0 **do**
4	$x = Q.GetFirst()$
5	**if** $x \in X_G$
6	**return** SUCCESS
7	**foreach** $u \in U(x)$
8	$x' = f(x,u)$
9	**if** $!\, x'.visited$
10	$x'.visited = \text{true}$
11	$Q.Insert(x')$
12	**else**
13	Resolve duplicity of x'
14	**return** FAILURE

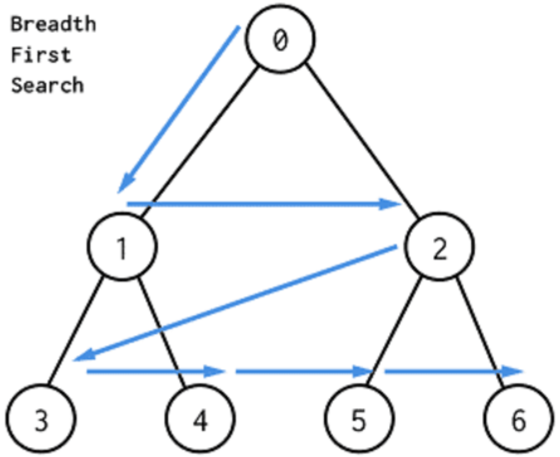
Q	Priority queue
x_0	Initial state
X_G	Set of goal states
x, x'	States
u	State transition
$U(x)$	Set of all transitions in state x

Various planners differ especially in a type of employed priority queue Q and a mechanism for resolving state conflicts (line 13).

- No information on goal is provided
- Configuration space is systematically explored until a goal is encountered

Breadth-first search

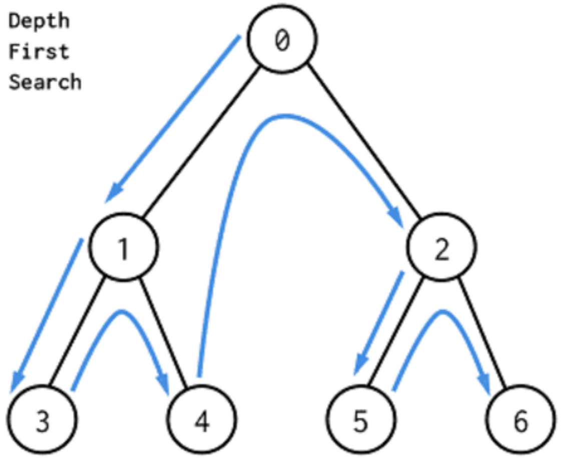
- A FIFO type priority queue → all states in depth k are explored before those in depth $k+1$
- Complete
- Optimal in case of equal transition costs



[1]

Depth-first search

- A LIFO type priority queue → whole branch is explored to the end of graph before a new branch is opened
- Not complete in infinite C-spaces
- Not optimal

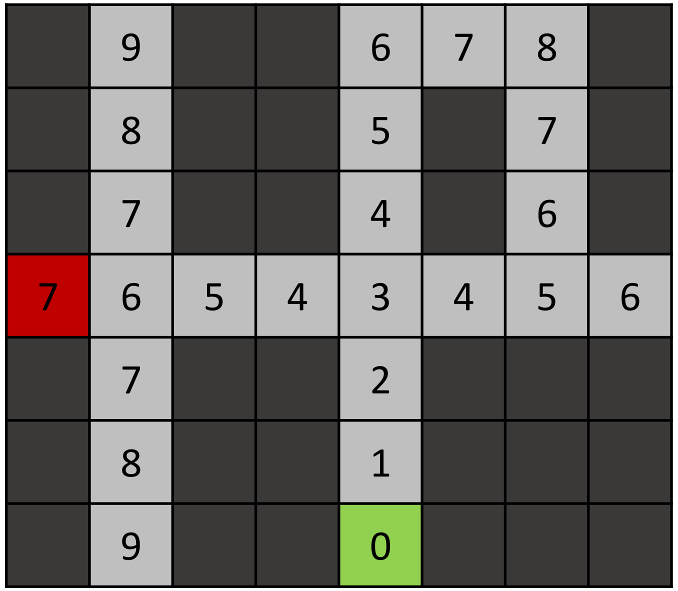


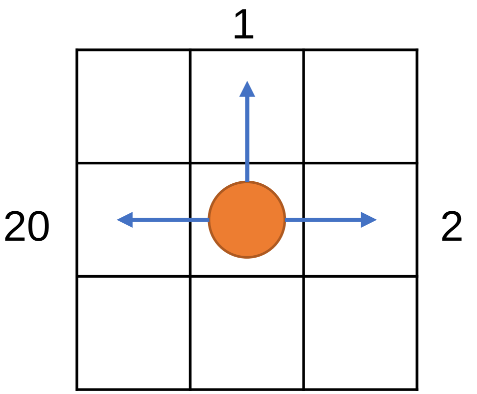
[1]

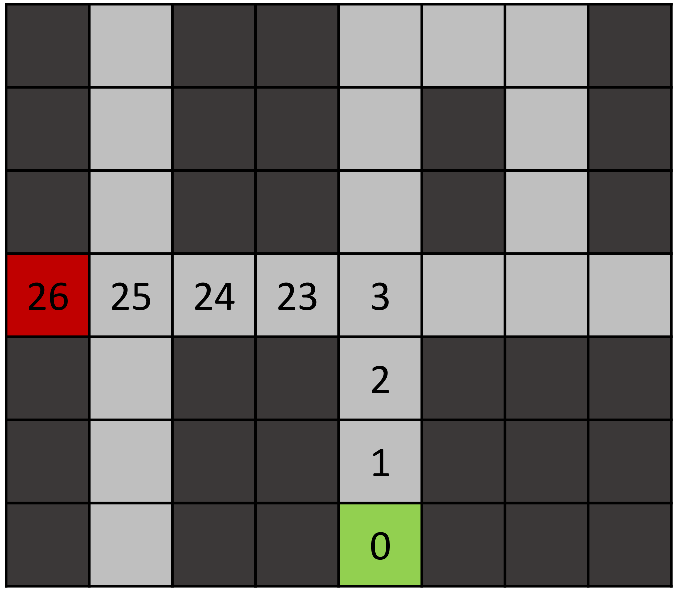
[1] ZALTSMAN, Daniel. Difference between Depth First Search and Breadth First Search. In: *DEV Community* [online]. 2020 [cit. 2021-02-27]. Available at: https://dev.to/danimal92/difference-between-depth-first-search-and-breadth-first-search-6om

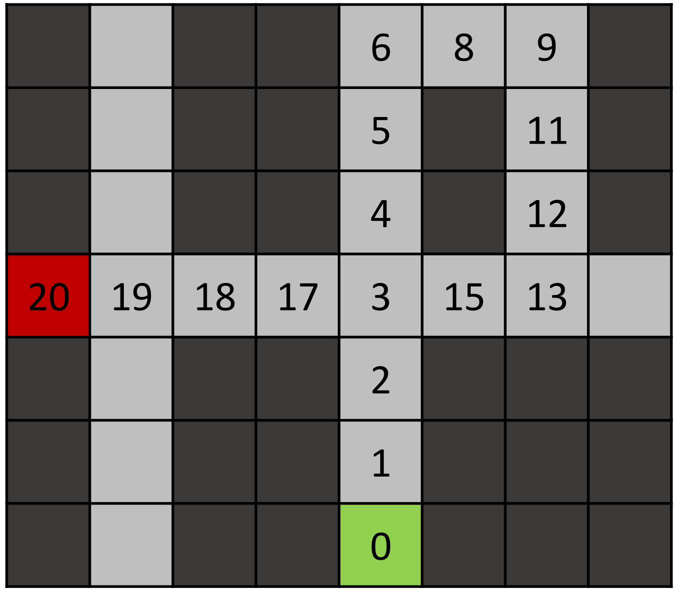
Assigning costs to transitions

- Some actions can be more „expansive“ than others in terms of complexity
- Examples:
    - It takes more energy to go up a steep slope
    - It is possible to go faster on a road than on an agricultural field
    - Turning right takes less time than turing left in city traffic
- To represent such circumstences, we can assign various costs to:
    - Graph edges (what is the cost of going to adjacent cell)
    - Actions (what is the cost of particular operation performer by a robot)







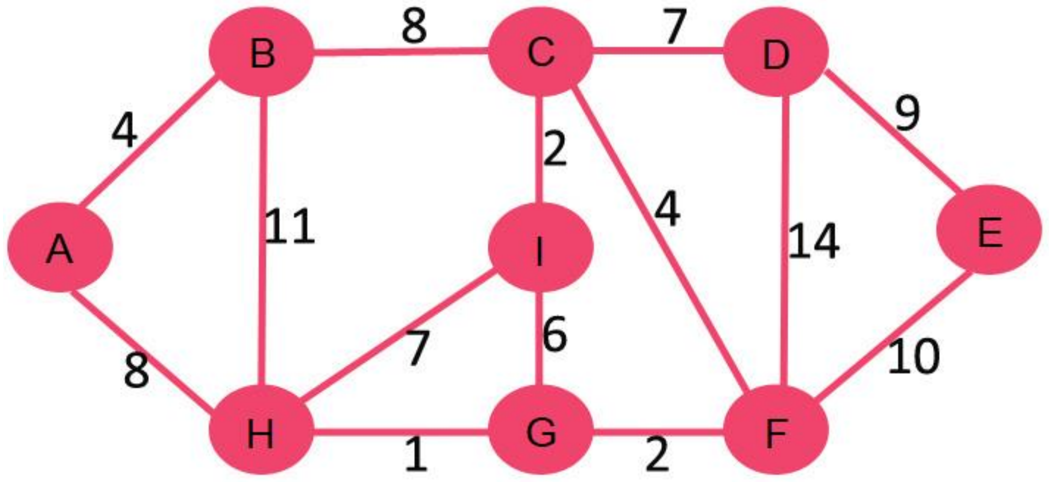


Greedy search and Dijkstra‘s algorithm

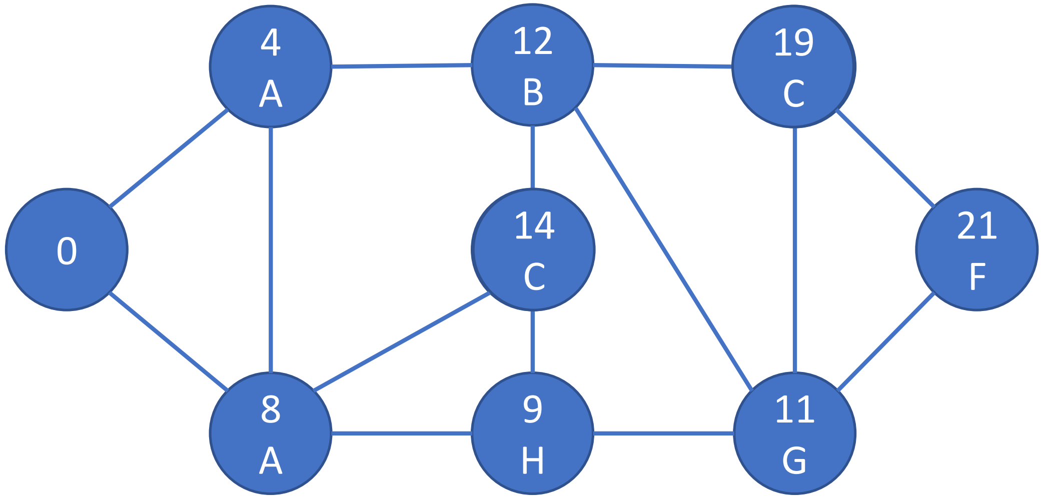
- The priority queue is ordered by the cost function $g(x)$ (aka cost-to-come)
- Cost function of a state is equal to sum of costs of all transitions and actions necessary to reach the state from the initial configuration

$$g(x_n) = \sum_{i=0}^{n-1} C(x_i \rightarrow x_{i+1})$$

- Each explored state is assigned value of cost function and respective action the state has been reached
- If state is revisited more efficiently, the cost function and action are updated
- Dijkstra‘s algorithm = full graph is expanded → optimal path from one state to each other



[1]



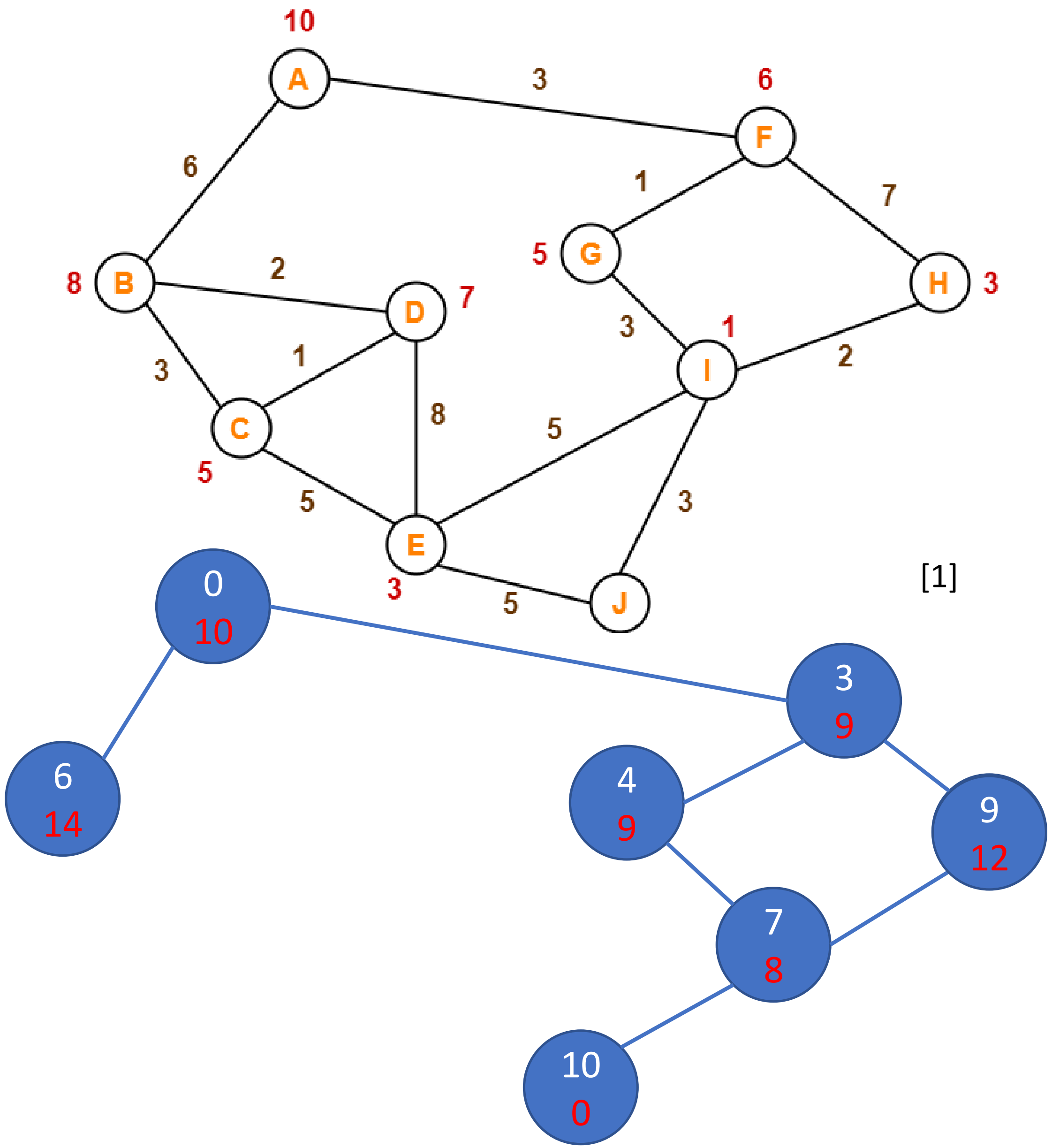
[1] Dijkstra’s shortest path algorithm | Greedy Algo-7. In: *GeeksforGeeks* [online]. 2020 [cit. 2021-02-28]. Available at: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Informed search – A* algorithm

- A heuristic function $h(x)$ is introduced (aka cost-to-go) – underestimated guess of cost necessary to reach the goal, e.g., Euclidean distance

  $h(x_n) = \|x_G - x_n\|$

- The priority queue is ordered by function $f(x) = g(x) + h(x)$
- Revisited states: Same as Dijkstra
- States closer to the goal are preferred in the exploration → smaller part of the graph is explored compared to the greedy search
- Critical condition to ensure optimality: $\forall x: h(x) \leq h^*(x)$ where $h^*(x)$ is the actual cost from x to the goal
- Works only for non-negative edges



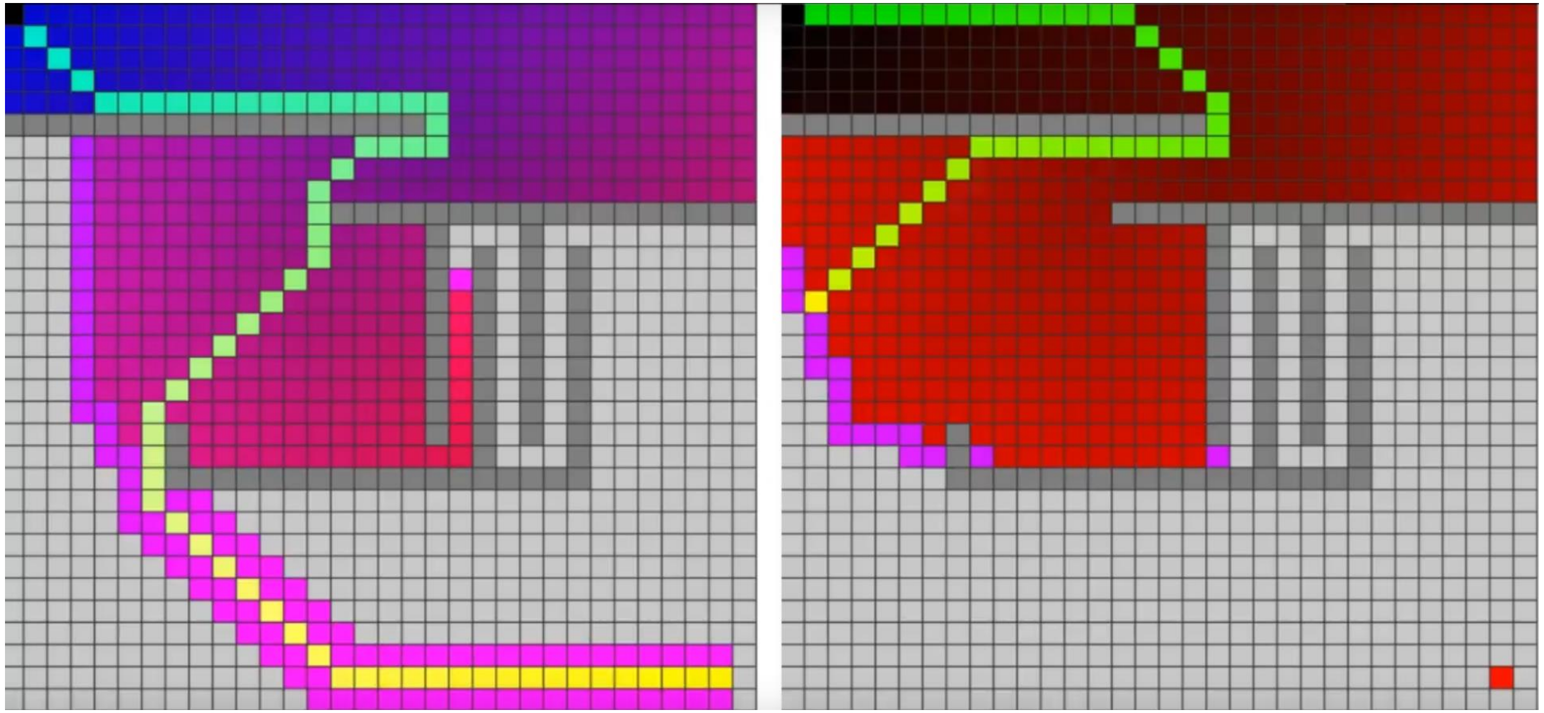
[1]

[1] SINGHAL, Akshay. A* Algorithm | A* Algorithm Example in AI. In: Gate Vidyalay [online]. 2020 [cit. 2021-02-28]. Available at: https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/

Dijkstra vs. A* – comparison

A*
- Informed search – use of heuristics
- Finds optimal sequence between two configurations
- What is the best heuristics?

Dijkstra
- Uniformed search
- Find optimal sequence between one and all other configurations
- The most accurate estimate for the A*



[1]

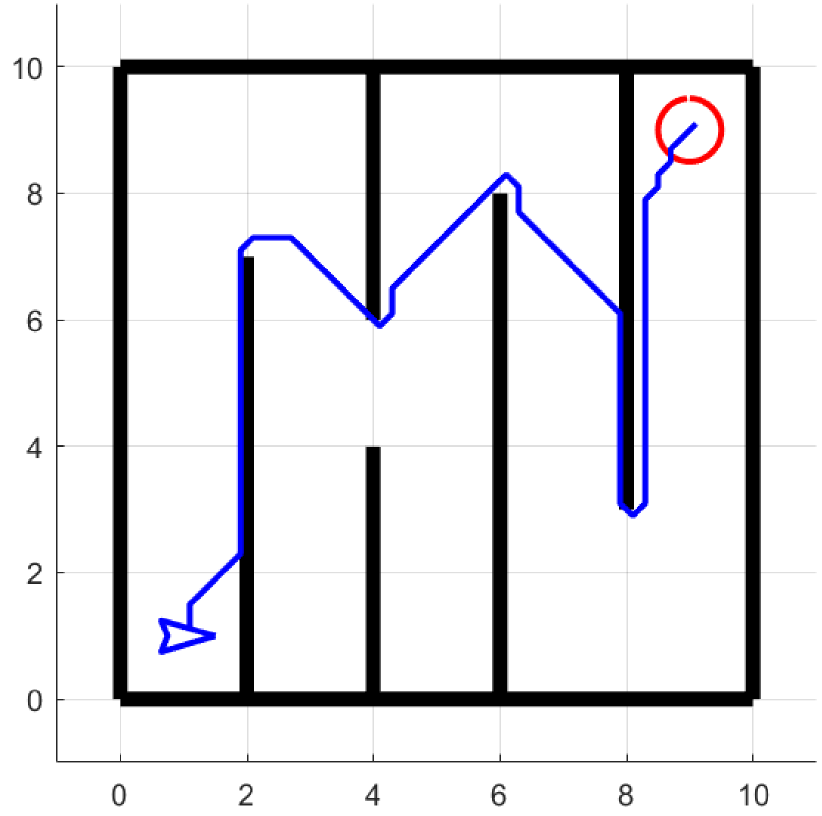
[1] A* (A star) vs Dijkstra's algorithm pathfinding grid visualization - JavaScript. In: *YouTube* [online]. 2018 [cit. 2021-02-27]. Available at: https://youtu.be/9REexHx0hDY

Possible problems

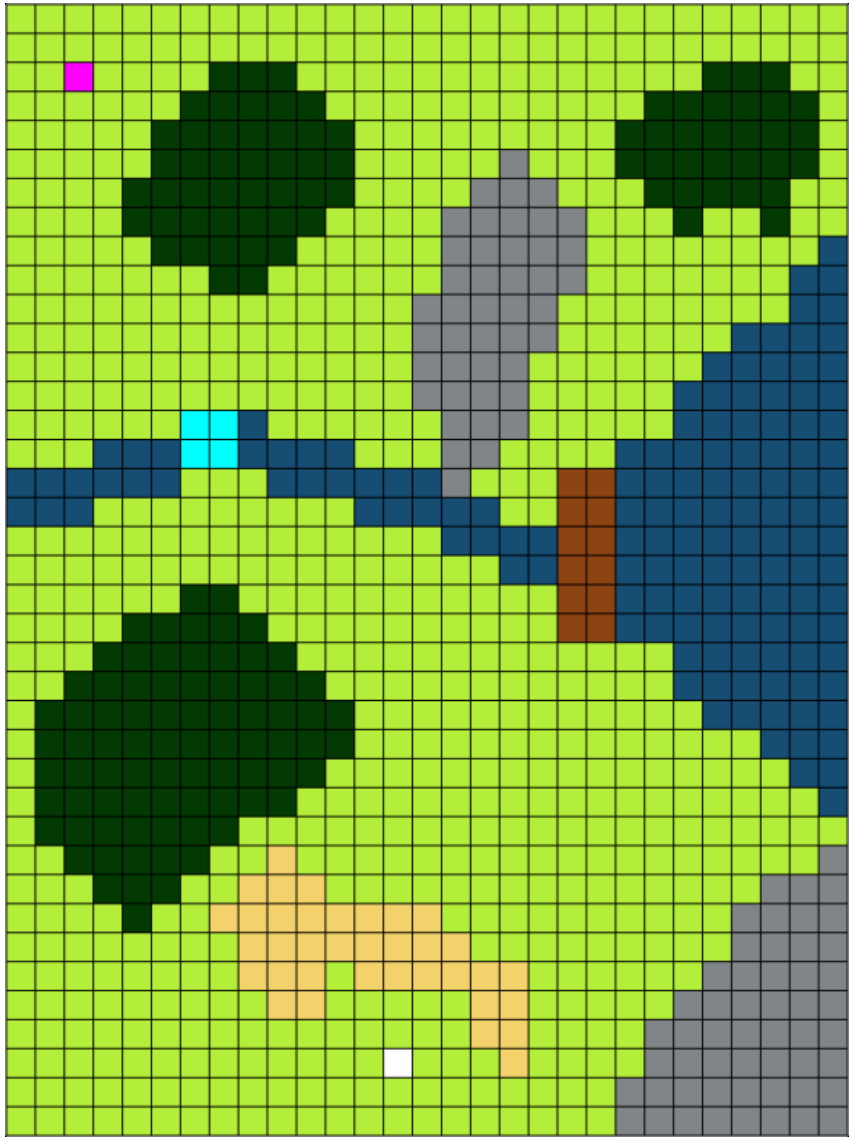
- Robot is not localized correctly
- The shortest path is usually found in the vicinity of obstacles
- Trajectory is aligned with the grid structure

Best-first (greedy) search

- Uses only the heuristics (cost-to-go) to order the priority queue
- Does not guarantee optimal solution
- Can be faster than A* but provides generally worse resulsts – one has to pay for beeing too greedy



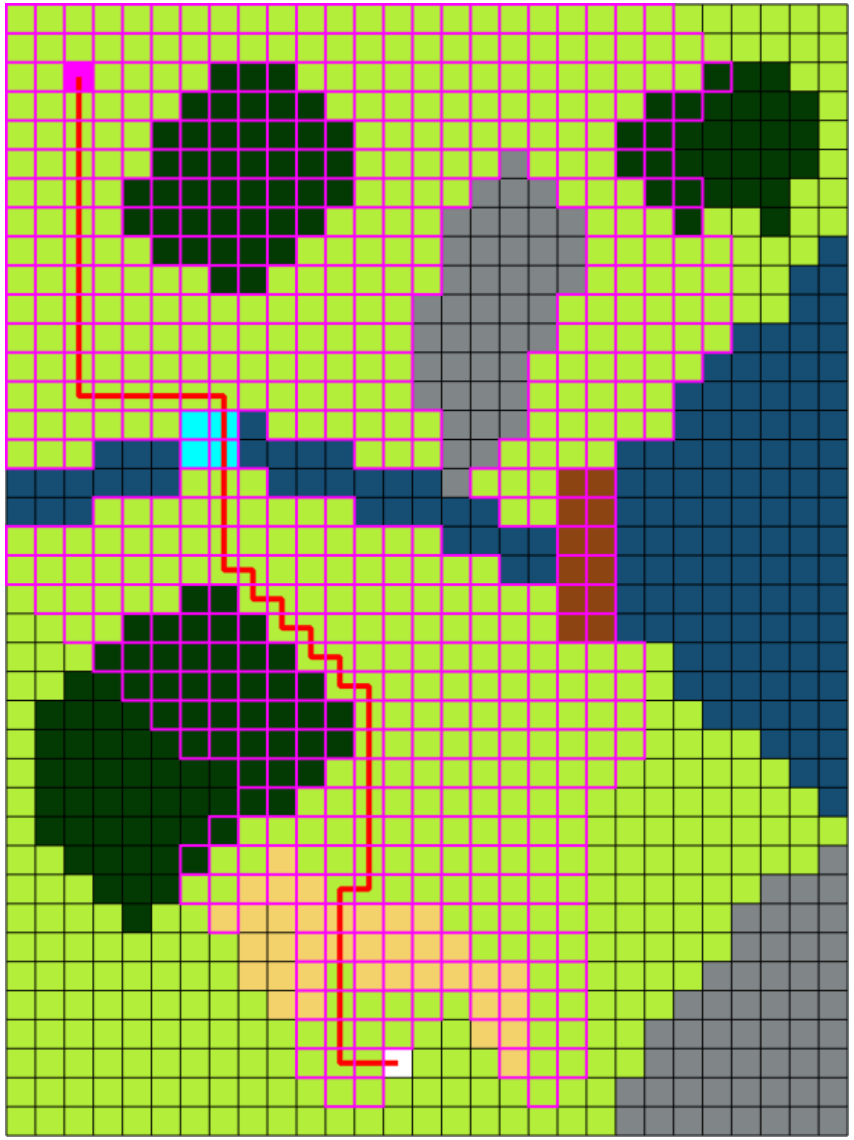
A* – examples



Grass
Forest
Sand
Rocks
Water
Crossing
Bridge

Start

Goal



Grass = 1
Forest = 3
Sand = 5
Rocks = Inf
Water = Inf
Crossing = 6
Bridge = 2

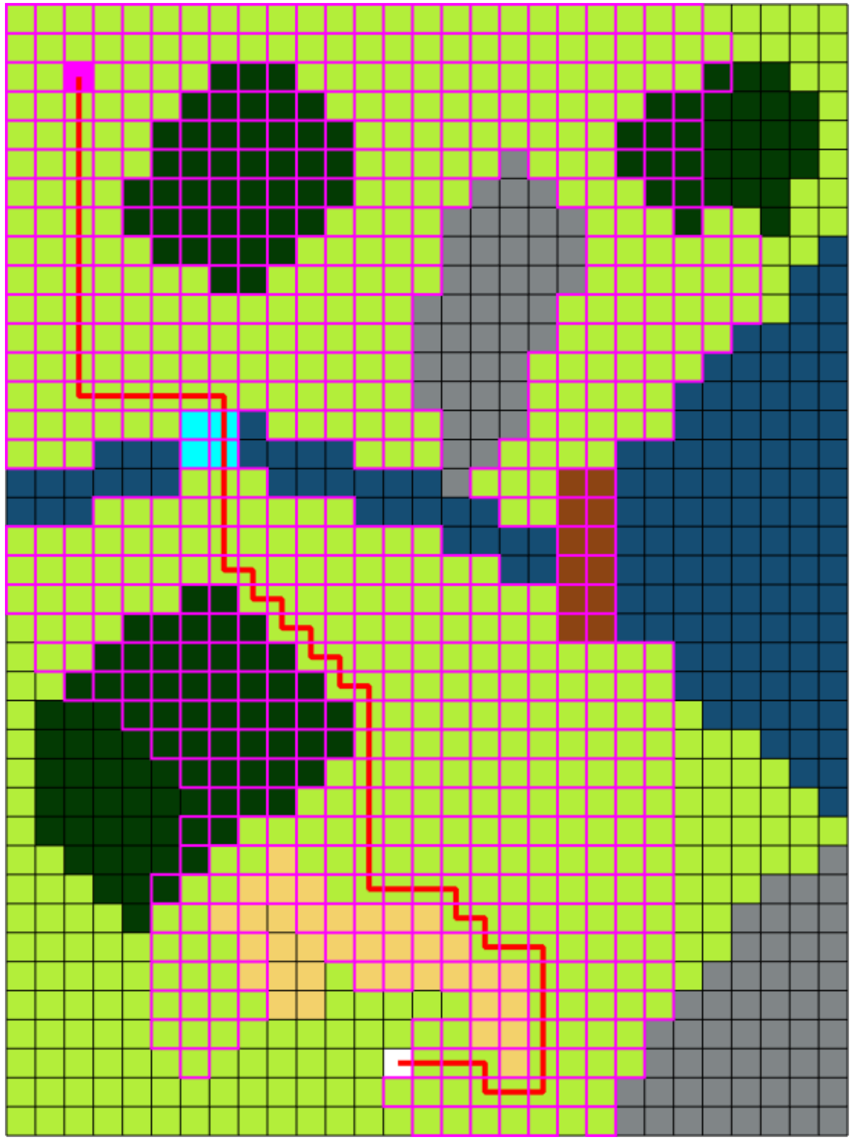
Start

Goal

Visited

Path

A* – examples

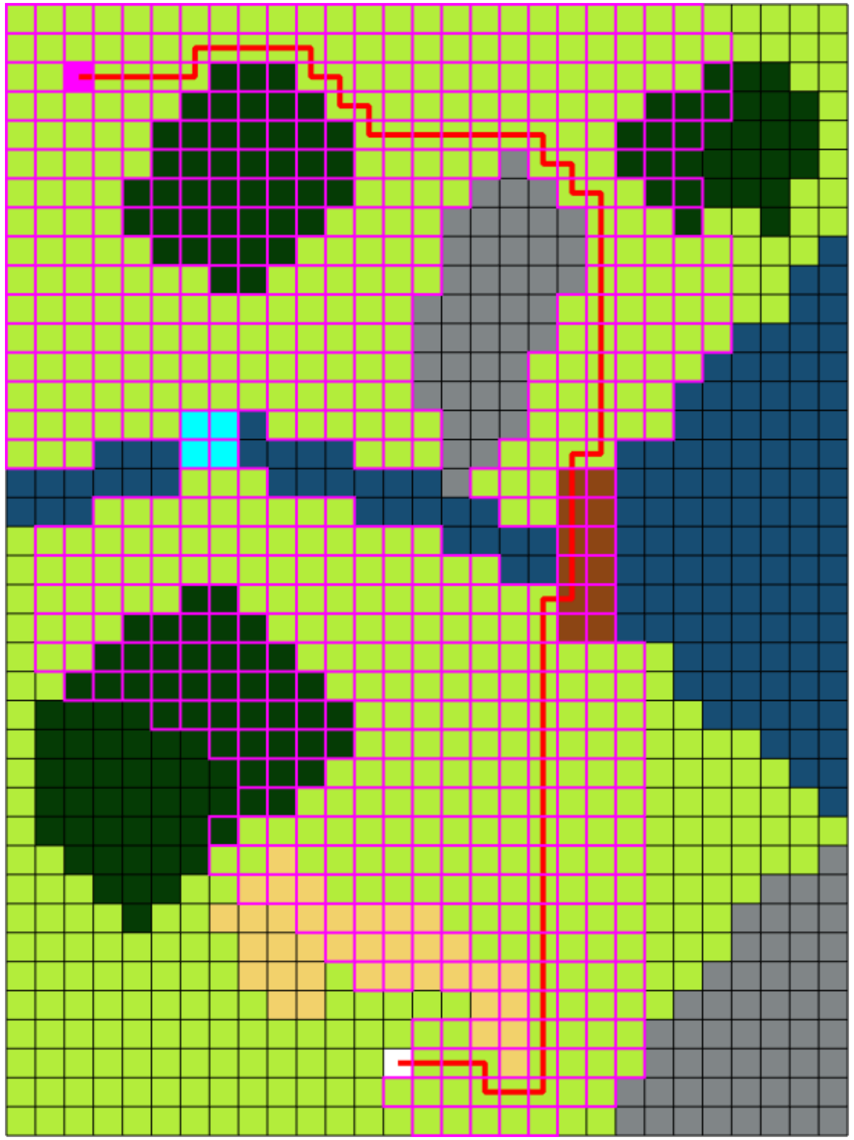


Grass = 1
Forest = 3
Sand = 7
Rocks = Inf
Water = Inf
Crossing = 6
Bridge = 2

Start
Goal

Visited

Path



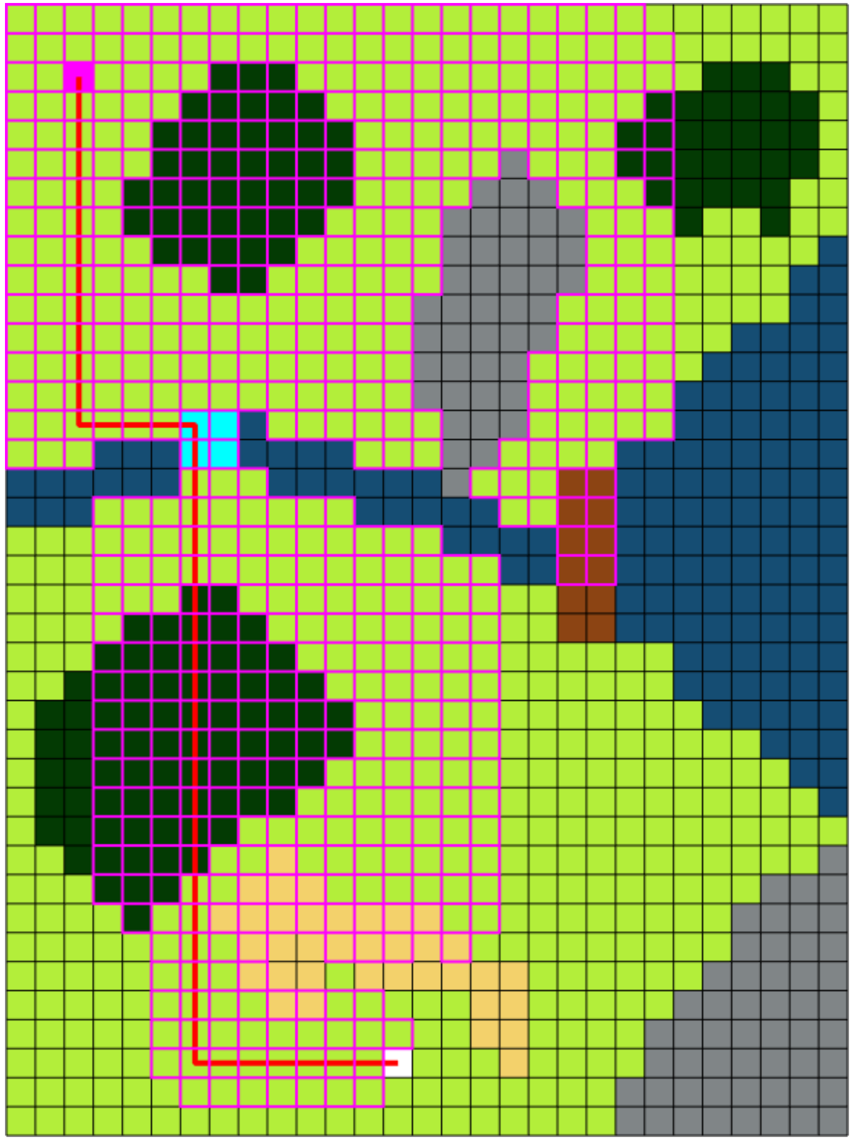
Grass = 1
Forest = 3
Sand = 5
Rocks = Inf
Water = Inf
Crossing = 8
Bridge = 2

Start
Goal

Visited

Path

A* – examples

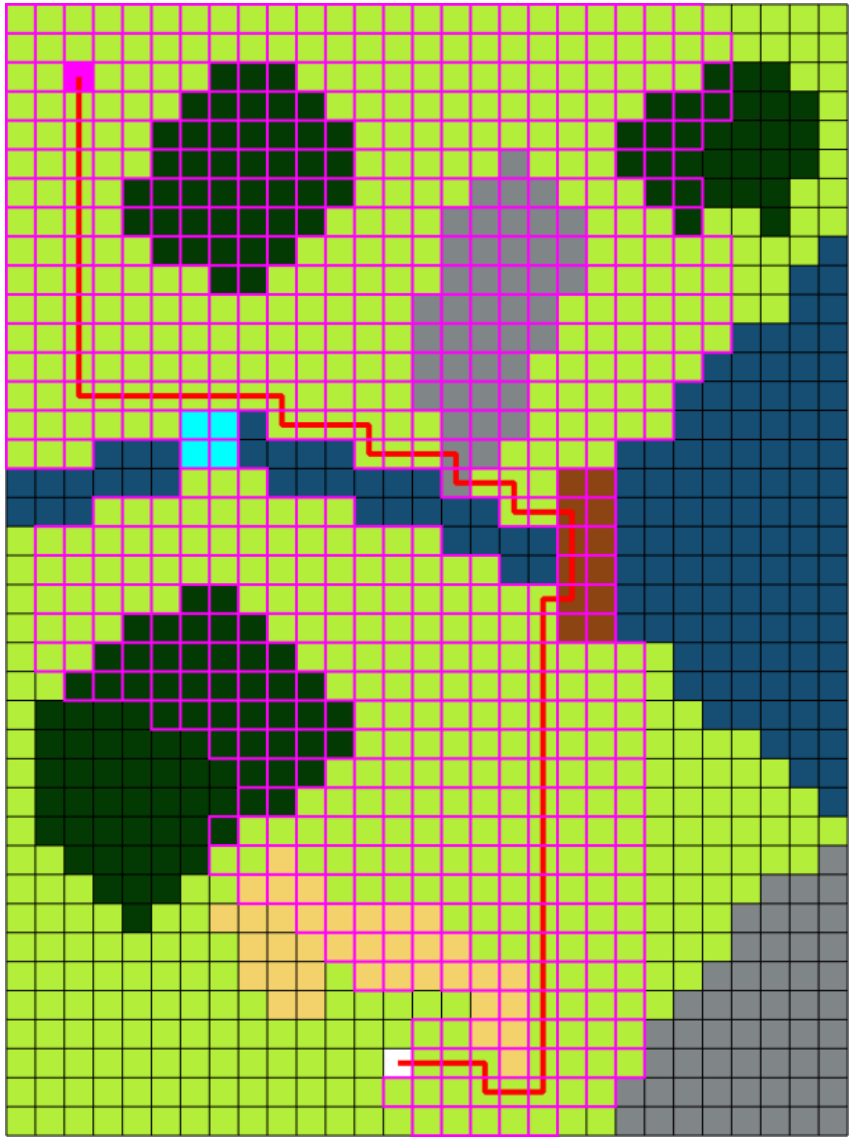


Grass = 1
Forest = 1.5
Sand = 5
Rocks = Inf
Water = Inf
Crossing = 8
Bridge = 2

Start
Goal

Visited

Path



Grass = 1
Forest = 3
Sand = 5
Rocks = 3
Water = Inf
Crossing = 8
Bridge = 2

Start
Goal

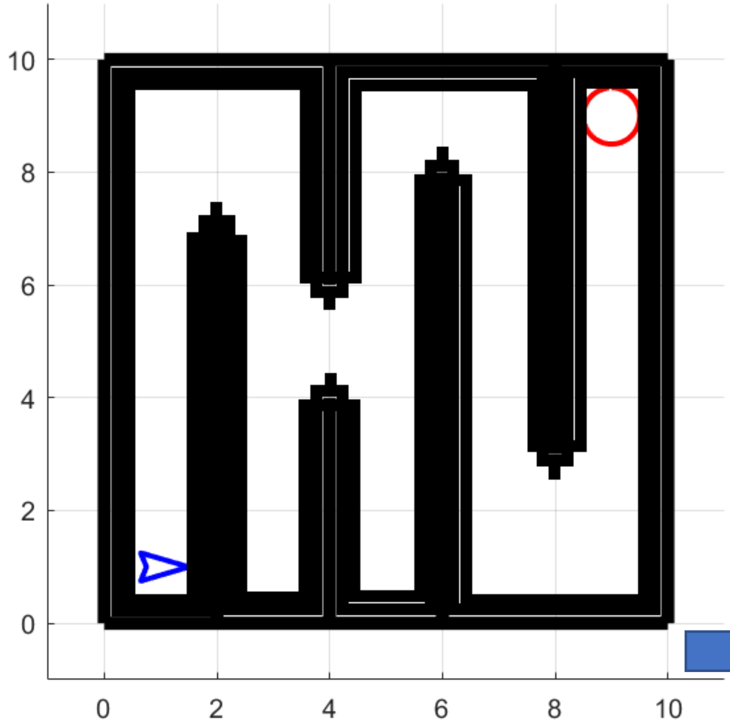
Visited

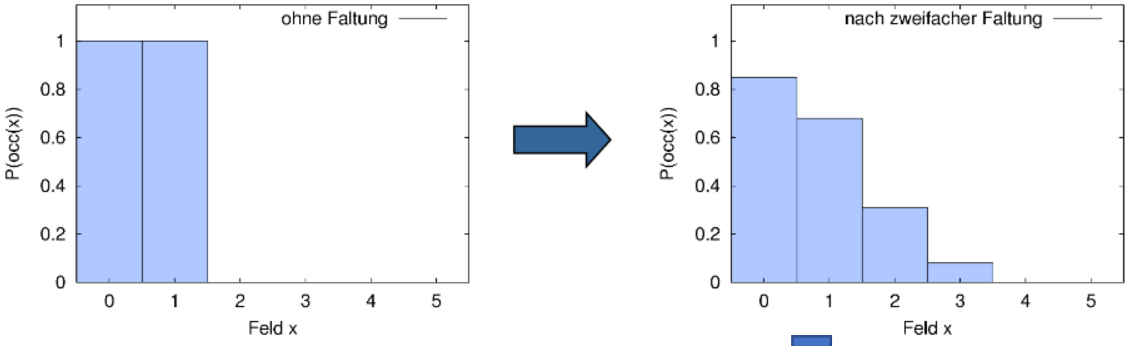
Path

Keeping obstacle clearance

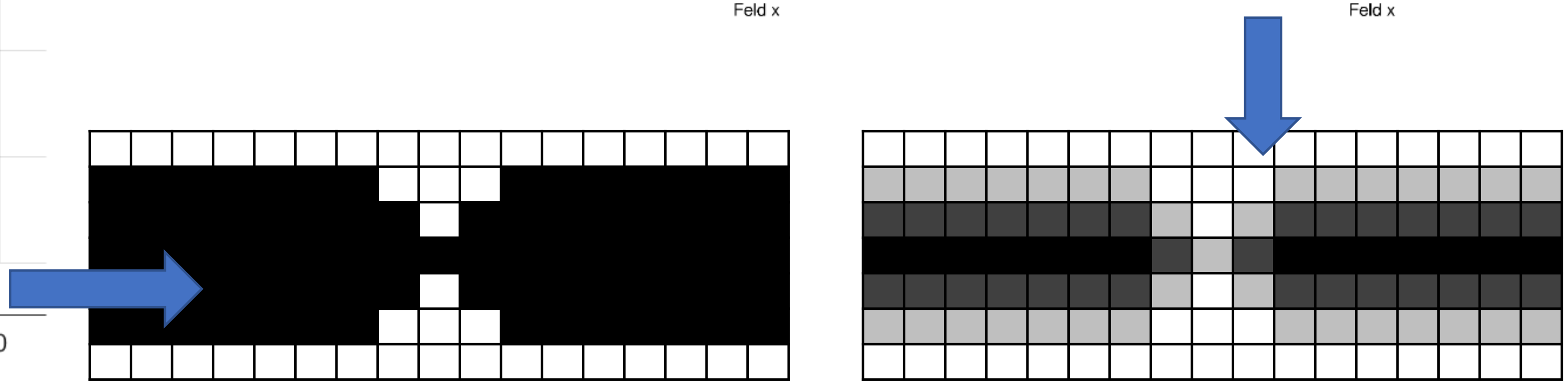
Dilating obstacles
- Works well in open spaces
- Inconvenient for narrow passages

Applying convolution
- Keeps passages open
- Cells closer to obstacles have higher cost
- Disk-shaped or diamond-shaped kernel





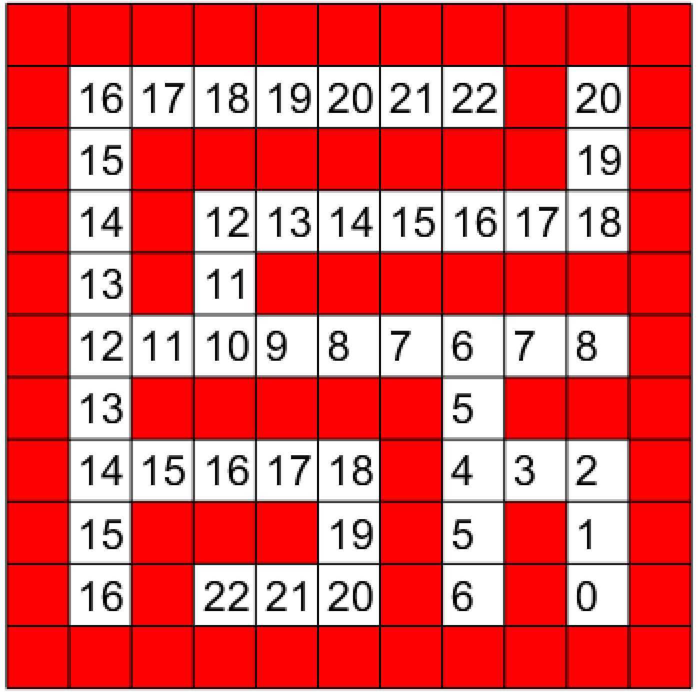
[1]



[1] BURGARD, Wolfram et al. Path Planning and Collistion Avoidance [online]. Uni Freiburg, 2010 [cit. 2021-02-27]. Available at: http://ais.informatik.uni-freiburg.de/teaching/ss10/robotics/slides/16-pathplanning.pdf

Dynamic programming

- A way to deal with stochastic environments
- For each cell a *value* is computed and optimal policy is set
- Iterative process



Given: map, value (>> 0), policy (= null), actions, goal, cost_step

```
value[goal[1]][goal[2]] = 0
change = true
while change do
    change = false
    foreach (x, y) st. map[x][y] is empty
        foreach a in actions
            x2 = x + a[1]
            y2 = y + a[2]
            if (x2, y2) is in range && map[x2][y2] is empty
                v2 = value[x2][y2] + cost_step
                if v2 < value[x][y]
                    change = true
                    value[x][y] = v2
                    policy[x][y] = a
```

Dynamic programming

- A way to deal with stochastic environments
- For each cell a *value* is computed and optimal policy is set
- Iterative process

Given: map, value (>> 0), policy (= null), actions, goal, cost_step	
1	*value*[*goal*[1]][*goal*[2]] = 0
2	*change* = true
3	**while** *change* **do**
4	*change* = false
5	**foreach** (*x*, *y*) st. *map*[*x*][*y*] is empty
	[*x2*][*y2*] is empty
	_*step*

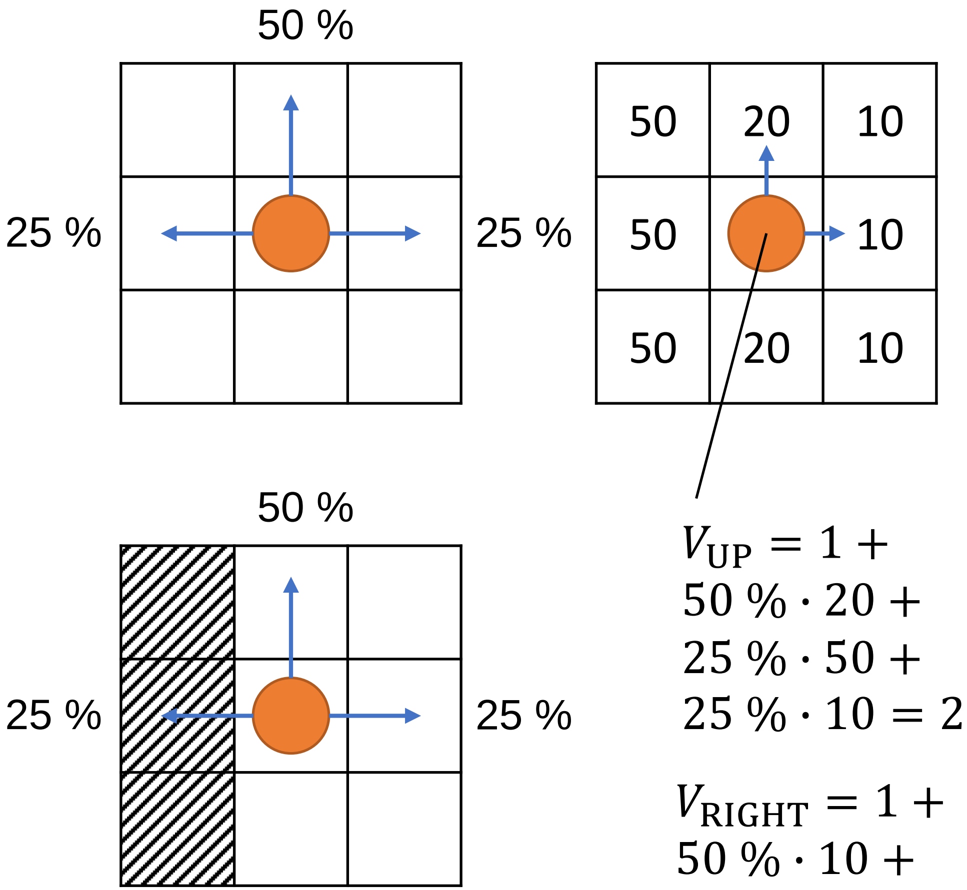
16	17	18	19	20	21	22		20
15								19
14		12	13	14	15	16	17	18
13		11						
12	11	10	9	8	7	6	7	8
13						5		
14	15	16	17	18		4	3	2
15				19		5		1
16		22	21	20		6		0

v	<	<	<	<	<	<		v
v								v
v		v	<	<	<	<	<	<
v		v						
>	>	>	>	>	>	v	<	<
^						v		
^	<	<	<	<		>	>	v
^				^		^		v
^		>	>	^		^		*

Dynamic programming

- Why not use Dijktra to compute the values?
- Iterative approach can be extended to take the uncertainty of motion into account (stochastic actions)
- DP with stochastic actions keeps obstacle clearance

99	99	99	99	99	99	99	99	99	99	99	99	99
99	91	89	91	99	67	57	61	99	56	48	60	99
99	87	85	87	99	54	47	43	39	37	37	43	99
99	83	80	82	99	49	44	41	37	33	32	36	99
99	79	76	78	99	49	45	42	37	30	28	30	99
99	75	73	73	99	51	49	49	99	26	24	26	99
99	72	69	66	62	55	53	54	99	22	20	22	99
99	71	68	64	61	58	56	59	99	19	16	19	99
99	73	69	66	63	61	61	65	99	18	10	18	99
99	80	73	68	66	64	66	75	99	30	0	30	99
99	99	99	99	99	99	99	99	99	99	99	99	99

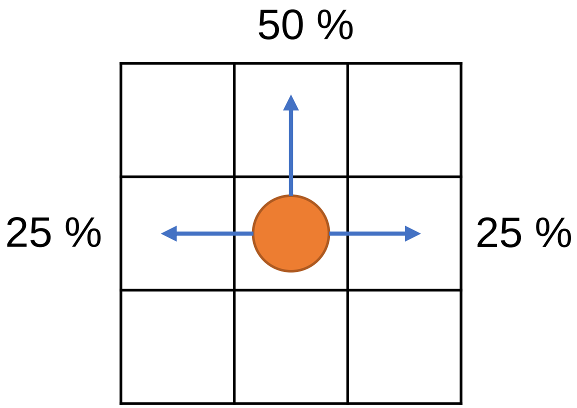


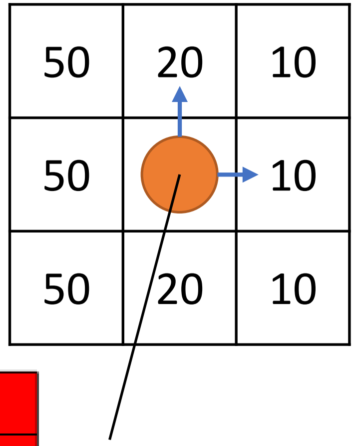
$$V_{\mathrm{UP}} = 1 + 50\,\% \cdot 20 + 25\,\% \cdot 50 + 25\,\% \cdot 10 = 26$$

$$V_{\mathrm{RIGHT}} = 1 + 50\,\% \cdot 10 + 25\,\% \cdot 20 + 25\,\% \cdot 20 = 14$$

Dynamic programming

- Why not use Dijktra to compute the values?
- Iterative approach can be extended to take the uncertainty of motion into account (stochastic actions)
- DP with stochastic actions keeps obstacle clearance



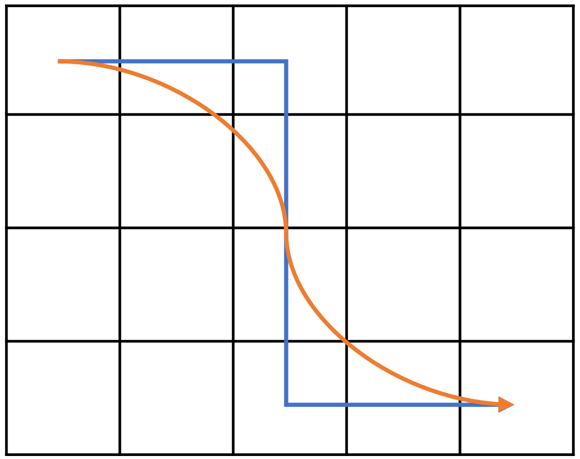


99	99	99	99	99	99	99	99	99	99	99	99	99
99	91	89	91	99	67	57	61	99	56	48	60	99
99	87	85	87	99	54	47	43	39	37	37	43	99
99	83	80	82	99	49	44	41	37	33	32	36	99
99	79	76	78	99	49	45	42	37	30	28	30	99
99	75	73	73	99	51	49	49	99	26	24	26	99
99	72	69	66	62	55	53	54	99	22	20	22	99
99	71	68	64	61	58	56	59	99	19	16	19	99
99	73	69	66	63	61	61	65	99	18	10	18	99
99	80	73	68	66	64	66	75	99	30	0	30	99
99	99	99	99	99	99	99	99	99	99	99	99	99

	v	v	v		v	v	v		v	v	v	
	>	v	<		>	v	v	v	v	v	<	
	>	v	<		>	>	>	>	v	v	<	
	>	v	<		>	>	^	^	>	v	<	
	>	v	<		>	^	<		>	v	<	
	>	>	v	v	>	^	<		>	v	<	
	>	>	>	>	^	^	<		>	v	<	
	>	>	>	>	^	^	<		>	v	<	
	>	^	^	^	^	^	^		>	*	<	

$$V_{\mathrm{UP}} = 1 + 50\,\% \cdot 20 + 25\,\% \cdot 50 + 25\,\% \cdot 10 = 26$$

$$V_{\mathrm{RIGHT}} = 1 + 50\,\% \cdot 10 + 25\,\% \cdot 20 + 25\,\% \cdot 20 = 14$$



- Grid-based planners yield paths that are aligned with the grid and do not consider kinematic contraints of the robot
- In actual scenario, we would like the path to be *smooth*, i.e., it does not contain sharp turns

Smoothing algorithm:

1. Let us have path $X = \{x_i\}_{i=1}^{N}$
2. Create copy of the original path $Y = X$
3. Optimize:

$$(x_i - y_i)^2 \rightarrow \min \qquad\qquad y_i = y_i + \alpha(x_i - y_i)$$

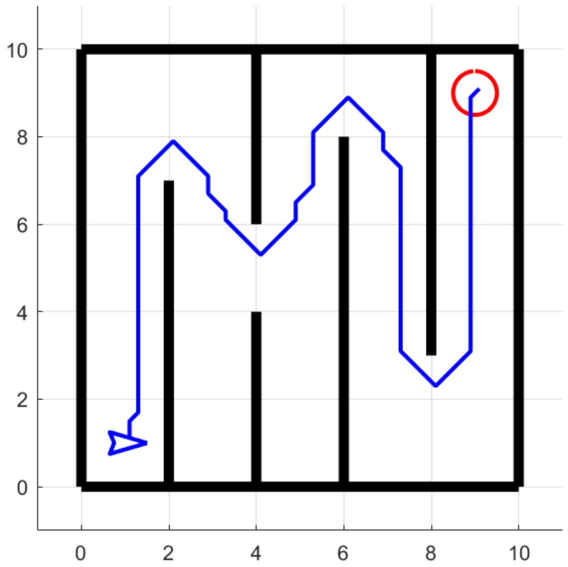
$$(y_i - y_{i+1})^2 \rightarrow \min \qquad\qquad y_i = y_i + \beta(y_i - y_{i+1})$$

Iterative method:

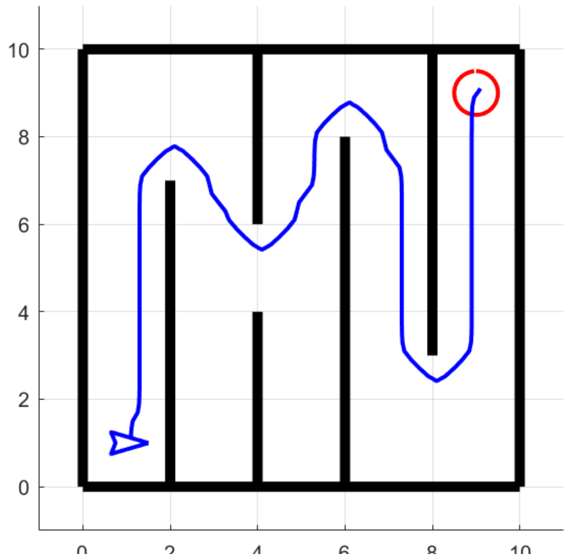
$$y_i^{(k+1)} = y_i + \alpha(x_i - y_i) + \beta(y_{i-1} + y_{i+1} - 2y_i), \qquad i = 2 \ldots N - 1$$

Path smoothing – weights

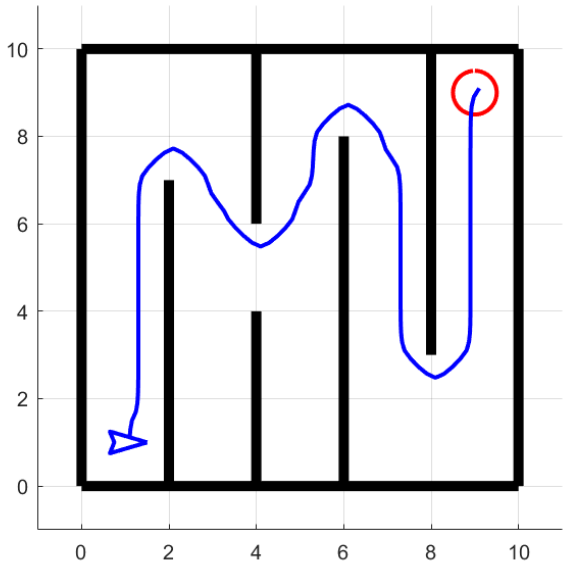
$\alpha = 0, \beta = 0$



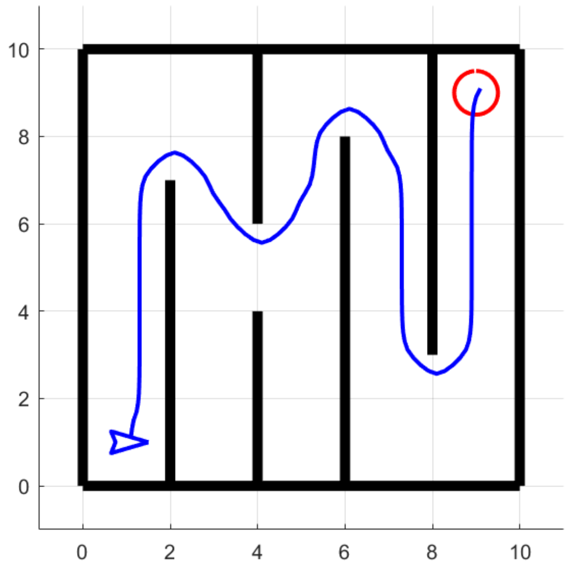
$\alpha = 0.5, \beta = 0.25$



$\alpha = 0.5, \beta = 0.5$



$\alpha = 0.25, \beta = 0.5$

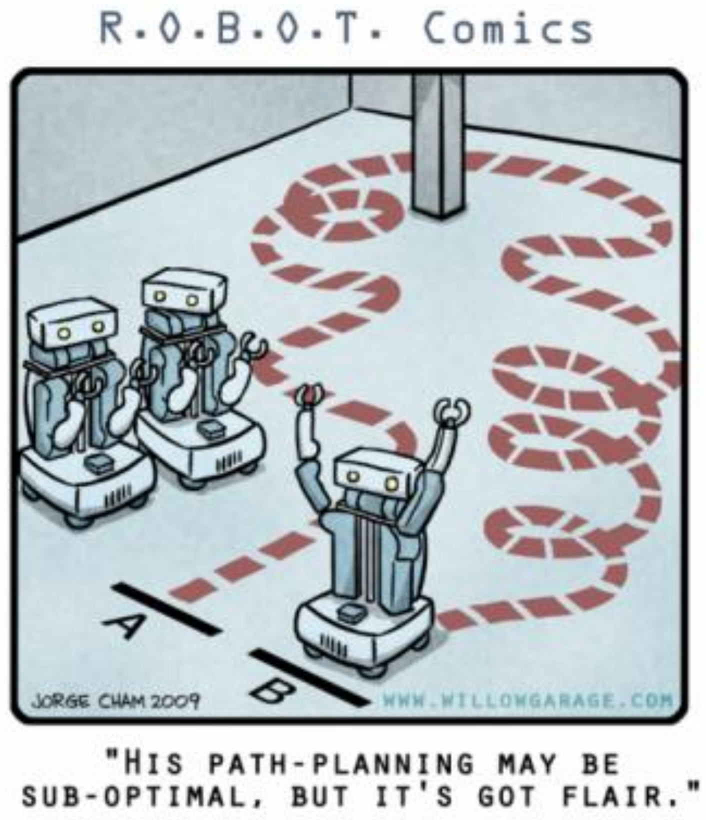


Path (motion) planning

- Given map of the workspace and description of the robot, find collistion-free sequence of steps (configurations) from the initial to the goal point.
- Completeness and optimality
- Assign costs to configuration transitions
- Usually reduced to search in a graph

Planning based on A*

- Convert map to occupancy grid and connect adjacent cells by graph edges
- Sort open nodes in queue by sum of cost-to-come and cost-to-go (heuristic) function
- Faster than uninformed search
- Finds optimal path between two points



[1]

[1] CHAM, Jorge. R.O.B.O.T. Comics: Path Planning In: *Willow Garage* [online]. 2009 [cit. 2021-02-28]. Available at: http://www.willowgarage.com/blog/2009/09/04/robot-comics-path-planning

Profile

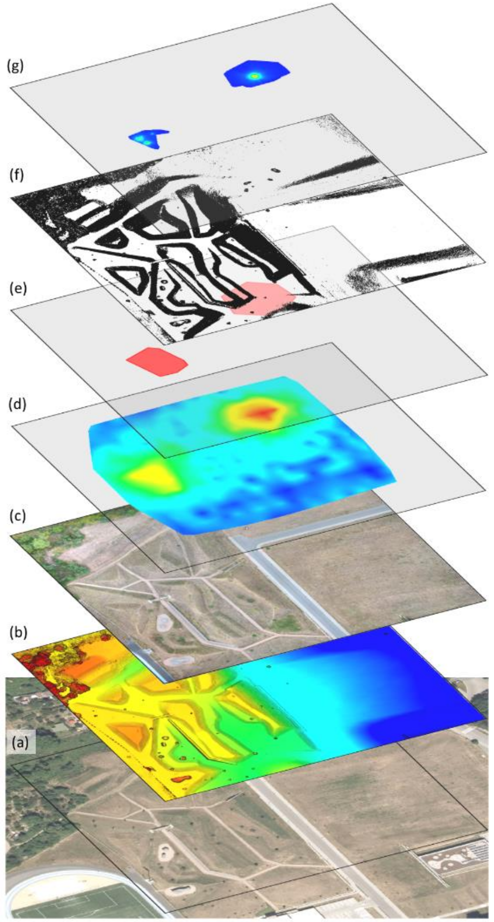
Tomas Lazna

Position: Ph.D. Student @ FEEC,
Junior Researcher @CEITEC

Research Topic: Radiation mapping via
robotic platforms

Room: SE1.102

Contact: Chat @ MS Teams,
tomas.lazna@ceitec.vutbr.cz



Background:
- Motion planning in mobile robotics
- Radiation data processing
- Cooperation of UASs and UGVs
- Estimation problems

Hobbies and interests:
- Star Wars & science fiction
- LEGO
- Politics

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Tomas Lazna
tomas.lazna@ceitec.vutbr.cz

Brno University of Technology
Faculty of Electrical Engineering and Communication
Department of Control and Instrumentation

Robotics and AI Research Group