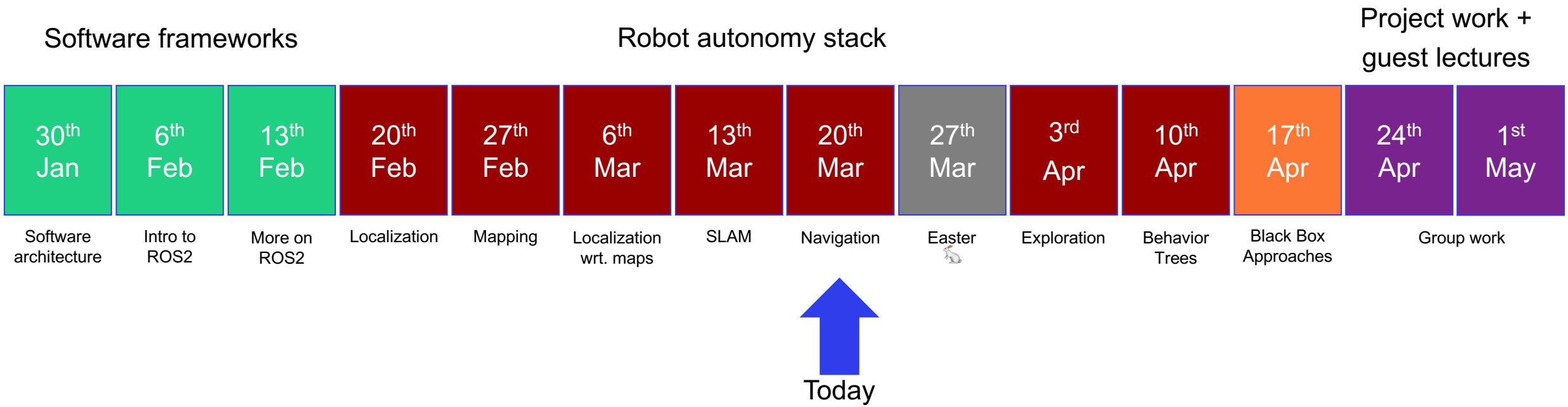
Nicholas Hansen

34761 – Robot Autonomy

Navigation

Overview of 34761 – Robot Autonomy

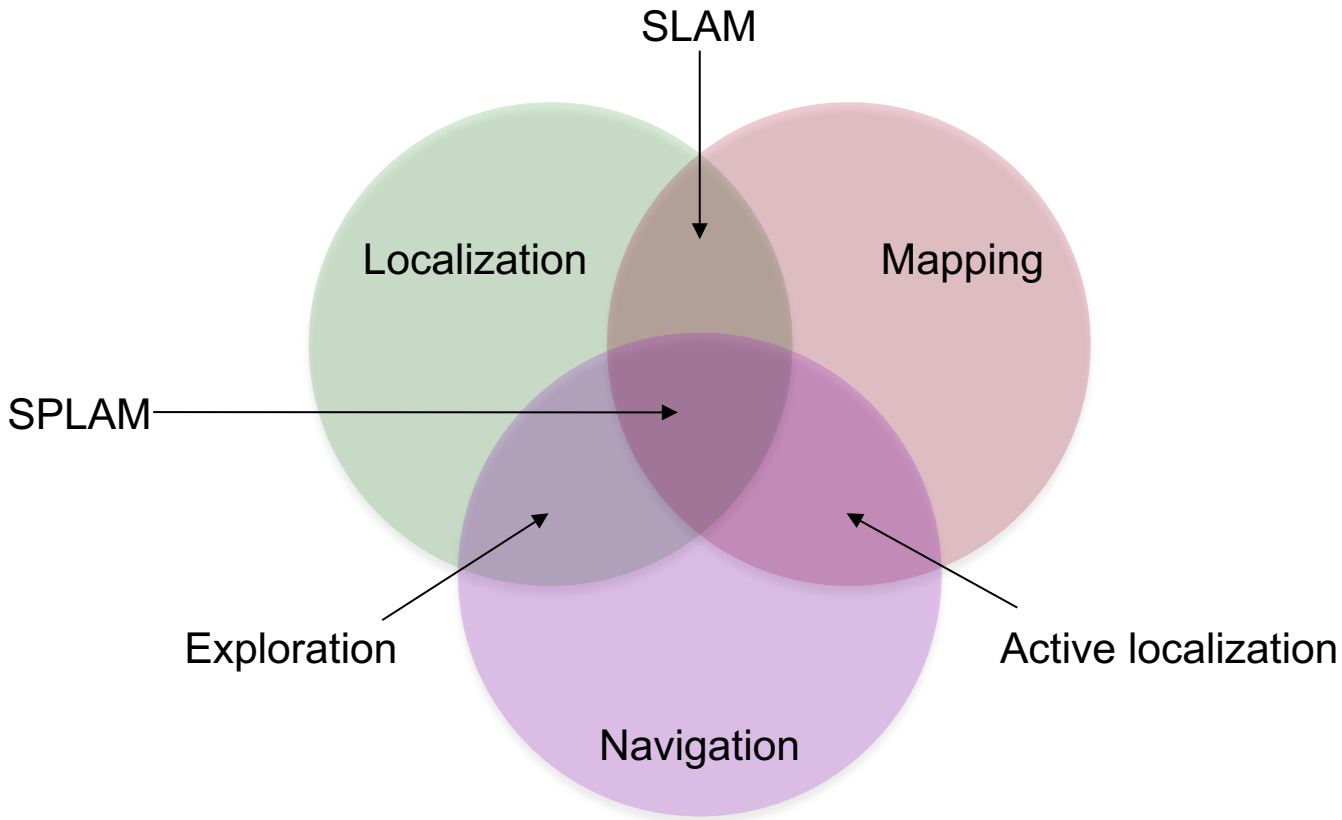
- 3 lectures on software frameworks
- 7 lectures on building your own autonomy stack for a mobile robot
- 1 lecture on DL/RL – an overview of black-box approaches to what you have done
- 2 lectures of project work before hand in + guest lectures



Outline for the next 7 weeks

- Our own autonomy stack:
  1. Localization
  2. Mapping
  3. Navigation
  4. Exploration
  5. Behaviour trees

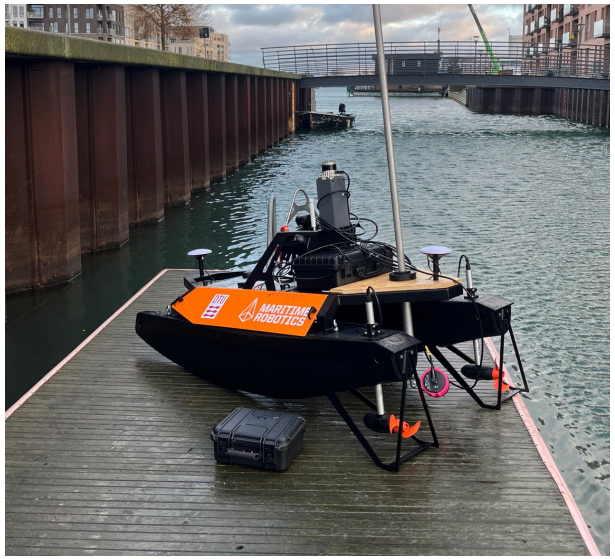
Topic of today

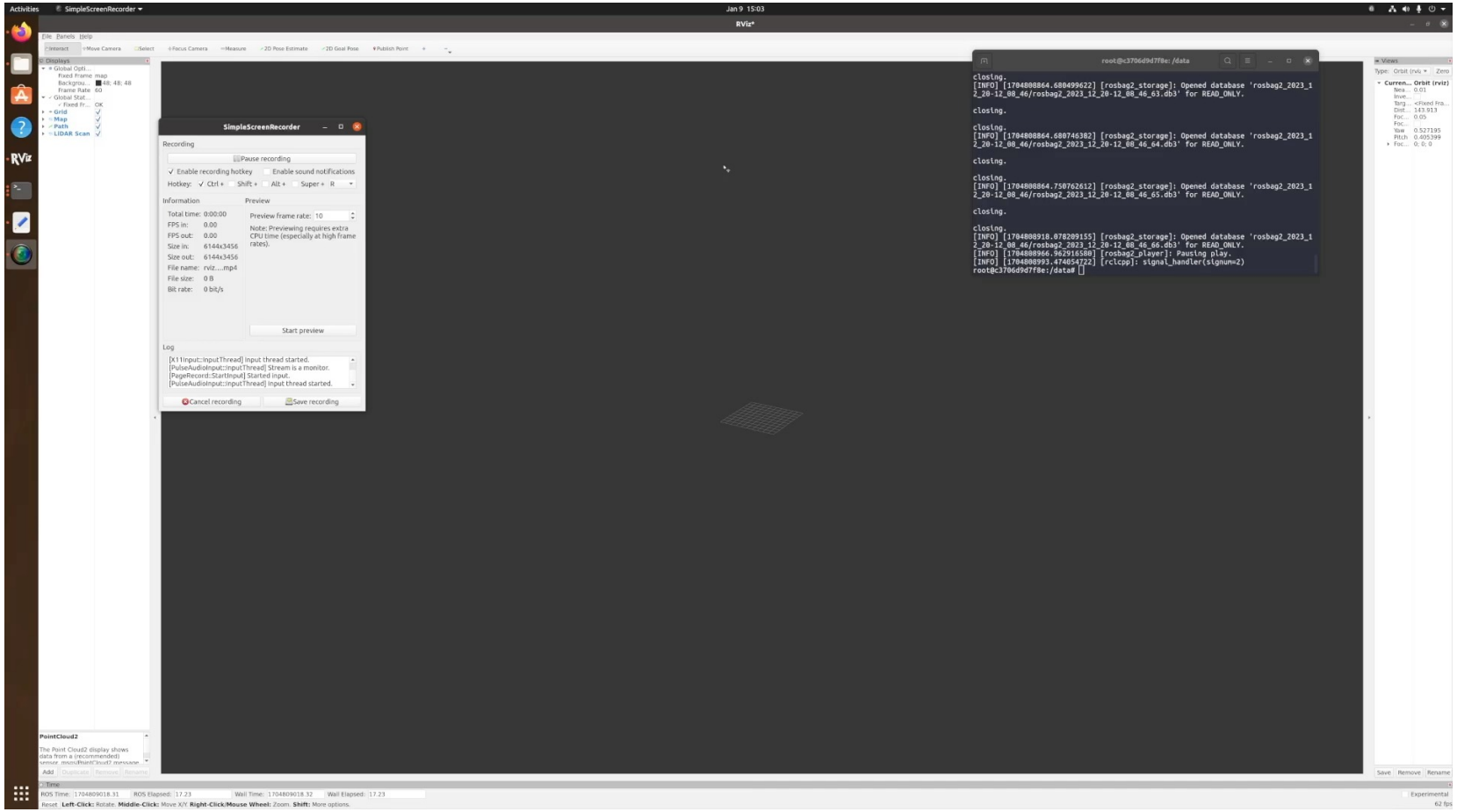


Recall from last lecture

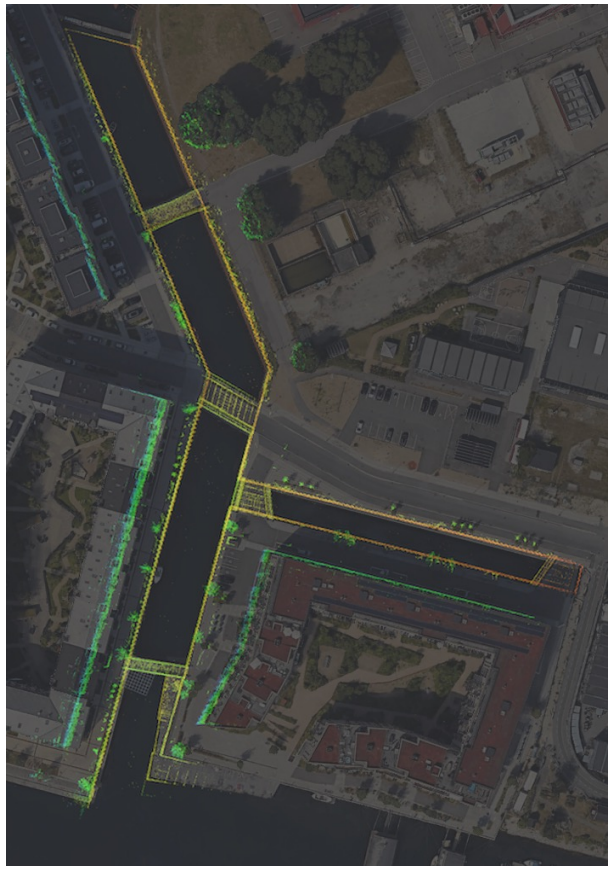
SLAM I guess?

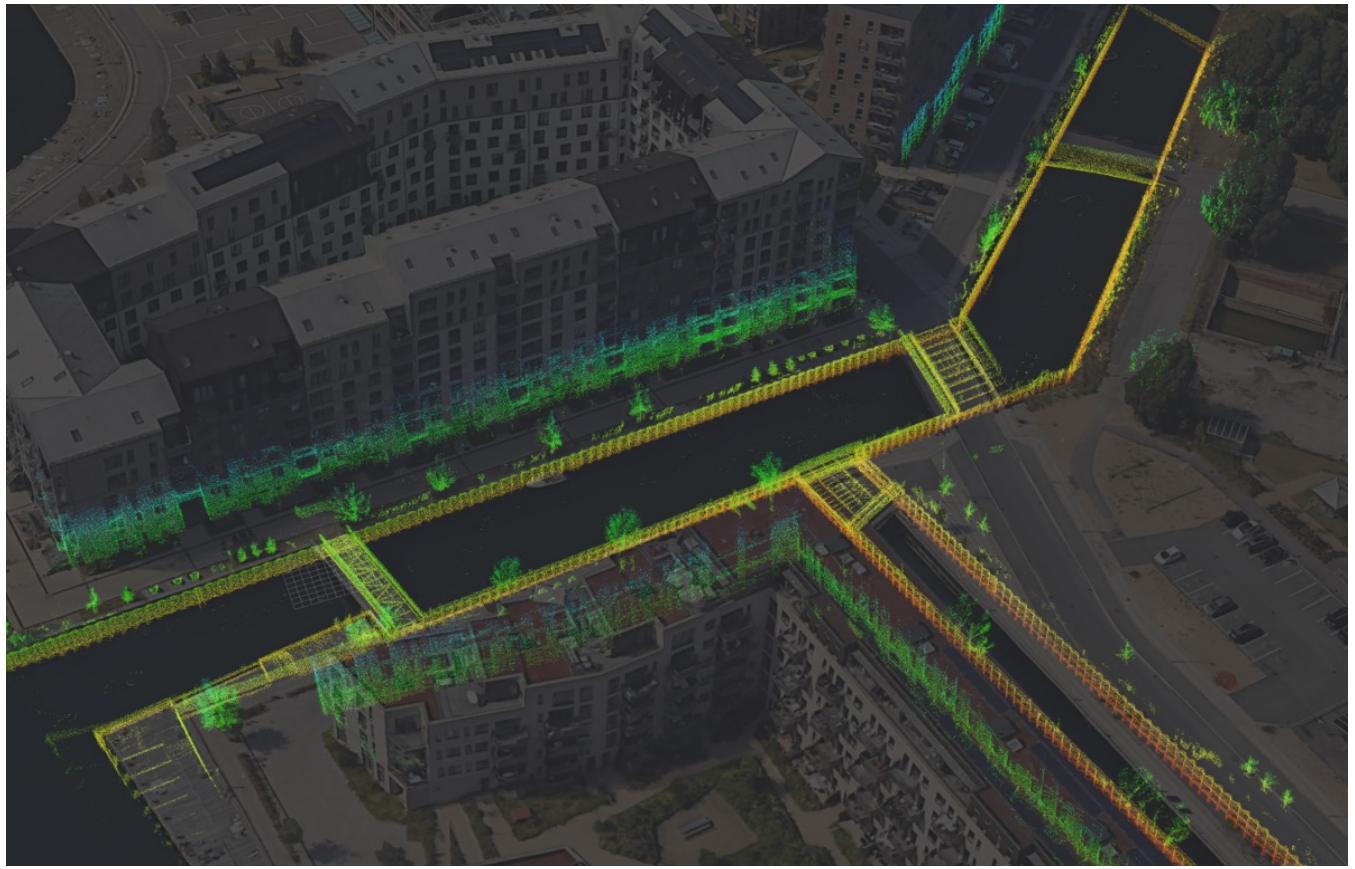
Examples of SLAM – Mapping Sydhavn (CPH)



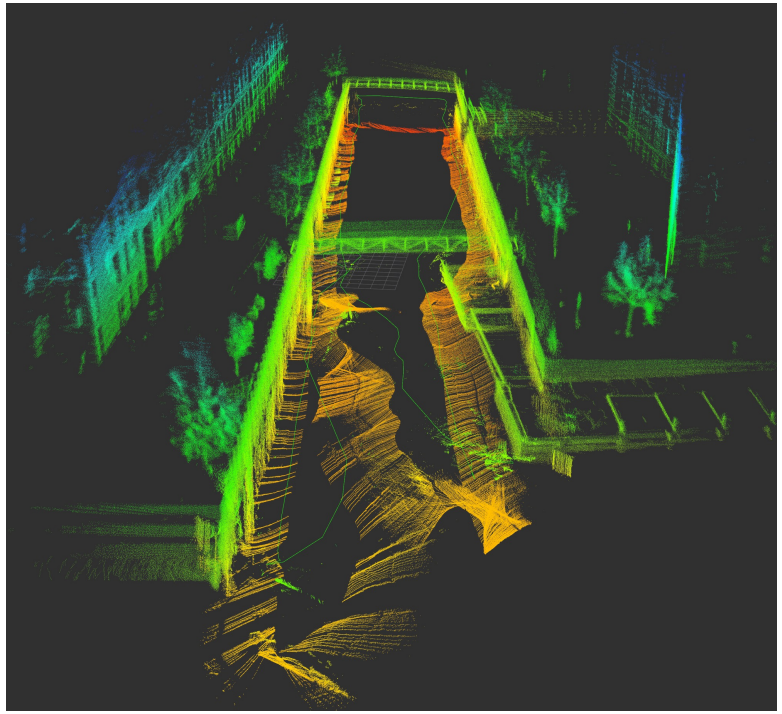


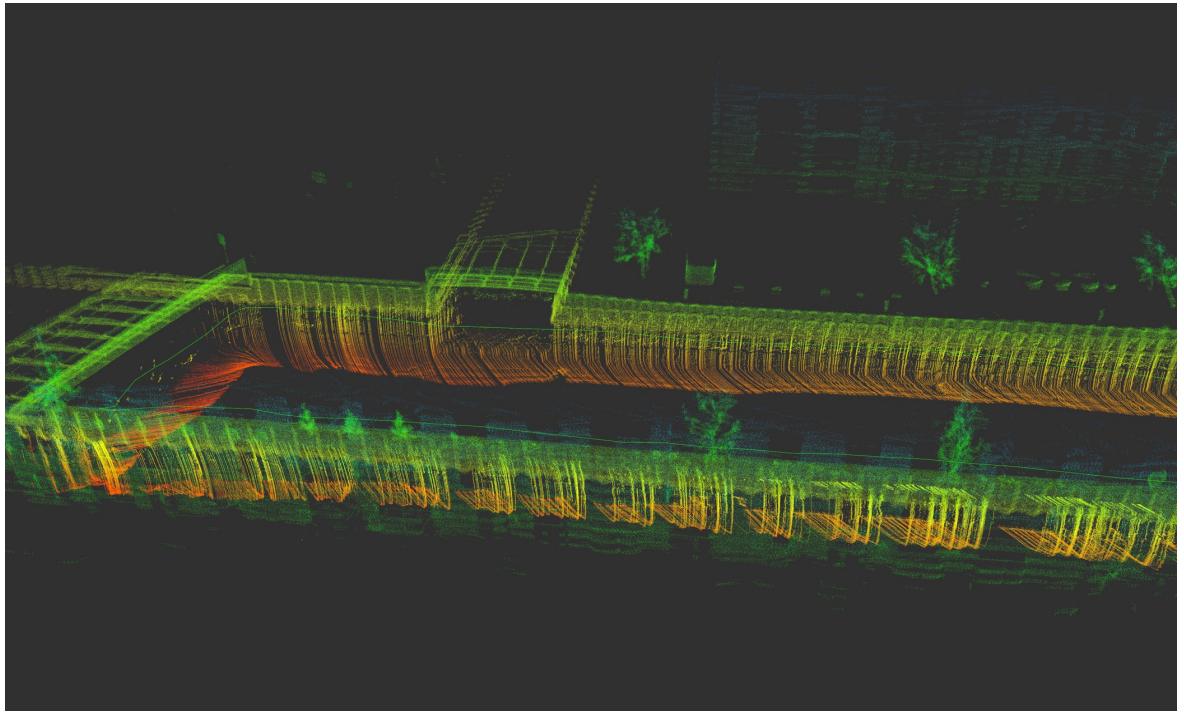
Examples of SLAM – Mapping Sydhavn (CPH)





Examples of SLAM – Mapping Sydhavn (CPH)





What is navigation?

- The process or activity of accurately ascertaining one's position, planning and following a route

How do we do navigation?

1. Generate a plan/map that can be used to query paths from A to B
2. Use your position to move relative to landmarks towards a goal

What do we need for navigation?

- Our position, landmarks we can track our position with, a goal position,
- If we don’t have a map: **reactive navigation**
- If we have a map: **map-based navigation**

Terminology

- Planning vs Navigation
  - **Planning**: Use a map to generate a high-level path of how to reach the goal (draw the path on a road map)
  - **Navigation**: Executing the planned path (using the map with the drawn path while driving)

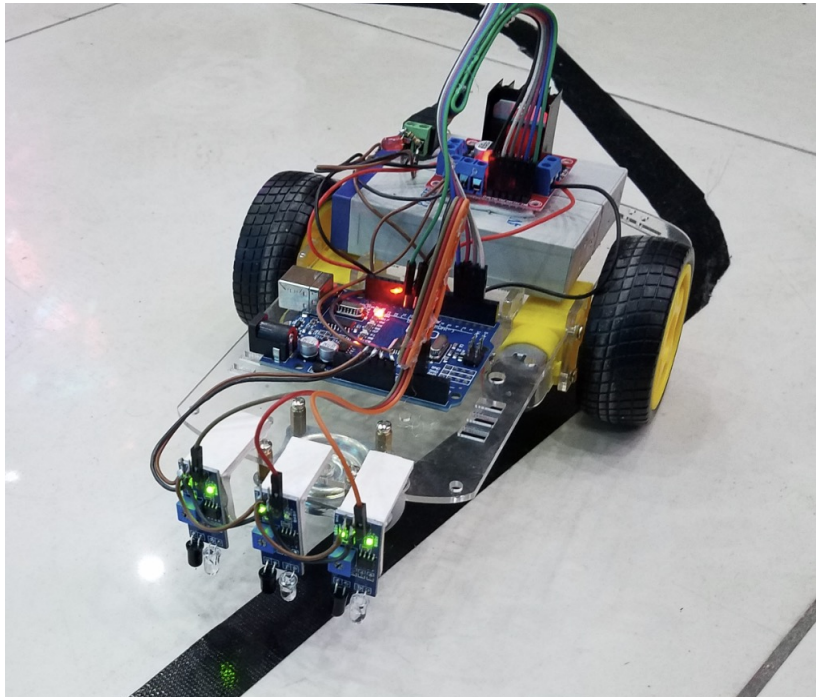
- Path vs Trajectory
  - **Path**: A series of points without considering dynamics (e.g. time)
  - **Trajectory**: A series of points including dynamic information (e.g. at what time should the robot reach a specific point in the trajectory)

- **Preprocessing phase**: creation of a plan
- **Query phase**: using the plan to find a path

Reactive navigation

- The robot is acting directly based on observations of the environment
  - The robot that can sense, ~~plan~~, and act

- Examples of reactive navigation:
  - Heading towards light
  - Following a white line on the floor
  - Moving through a maze by following a wall
  - (older) vacuum cleaners that moves around randomly until they hit something



The reactive BUG algorithm(s)

- The simplest bugs can only sense their immediate proximity and know their own location in the map

- **BUG2:** Move in a straight line towards the goal
  - When an object is hit follow the perimeter of the obstacle
  - When the robot again encounters the straight line between its initial position and the goal, continue towards the goal

- There is many bug-type algorithms (also more efficient ones than this example)
  - The lack of a map representation means they can only take paths that are locally optimal
  - They rely on knowing their position in the map, which is non-trivial to get without knowing the map, in which case more sophisticated methods might as well be used

Map-based navigation

- Produce the *optimal* trajectory between two points and then execute it

- But what is optimal?
  - Must be associated with a *cost*

- *Optimal* usually refers to shortest (collision free) path or distance, but could also incorporate other factors:
  - Terrain traversal difficulties (e.g. avoid driving over soft sand, or narrow passages)
  - Maximize previously unseen environment
  - Kinematics or dynamic constraints of the system (e.g. how maneuverable is the system)

Planning Fundamentals

- State/Configuration space

$$X, \quad X_{free}, \quad X_{obs} = X \backslash X_{free}$$

$$\mathcal{C}, \quad \mathcal{C}_{free}, \quad \mathcal{C}_{obs} = \mathcal{C} \backslash \mathcal{C}_{free}$$

- Initial and goal states/configurations

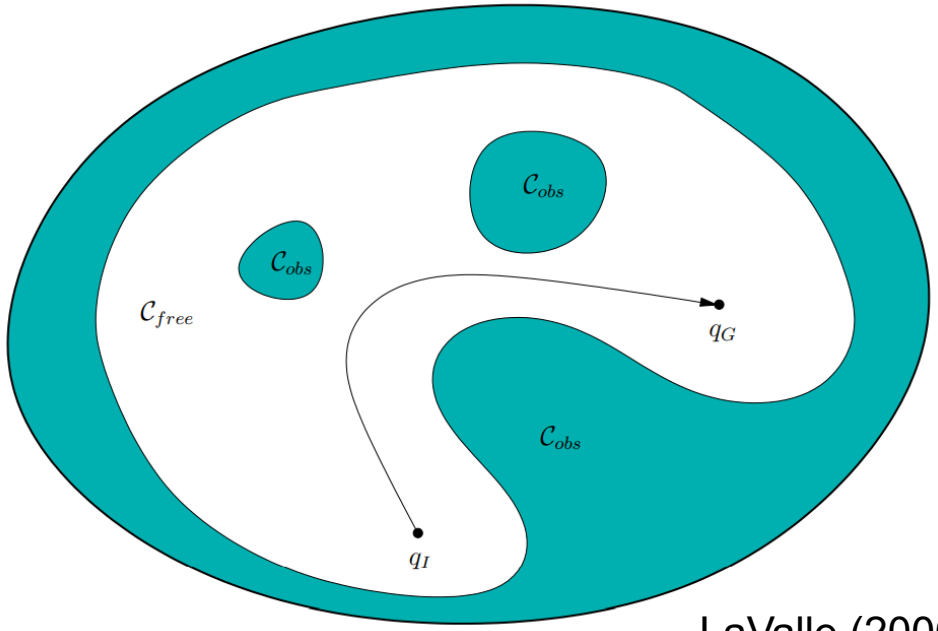
$$x_{start} \in X_{free}, \quad X_{goal} \subset X_{free}$$

$$q_{start} \in \mathcal{C}_{free}, \quad \mathcal{C}_{goal} \subset \mathcal{C}_{free}$$

- State transitions and actions

$$x' = f(x, u) \qquad u \in U(x), \quad x \in X$$

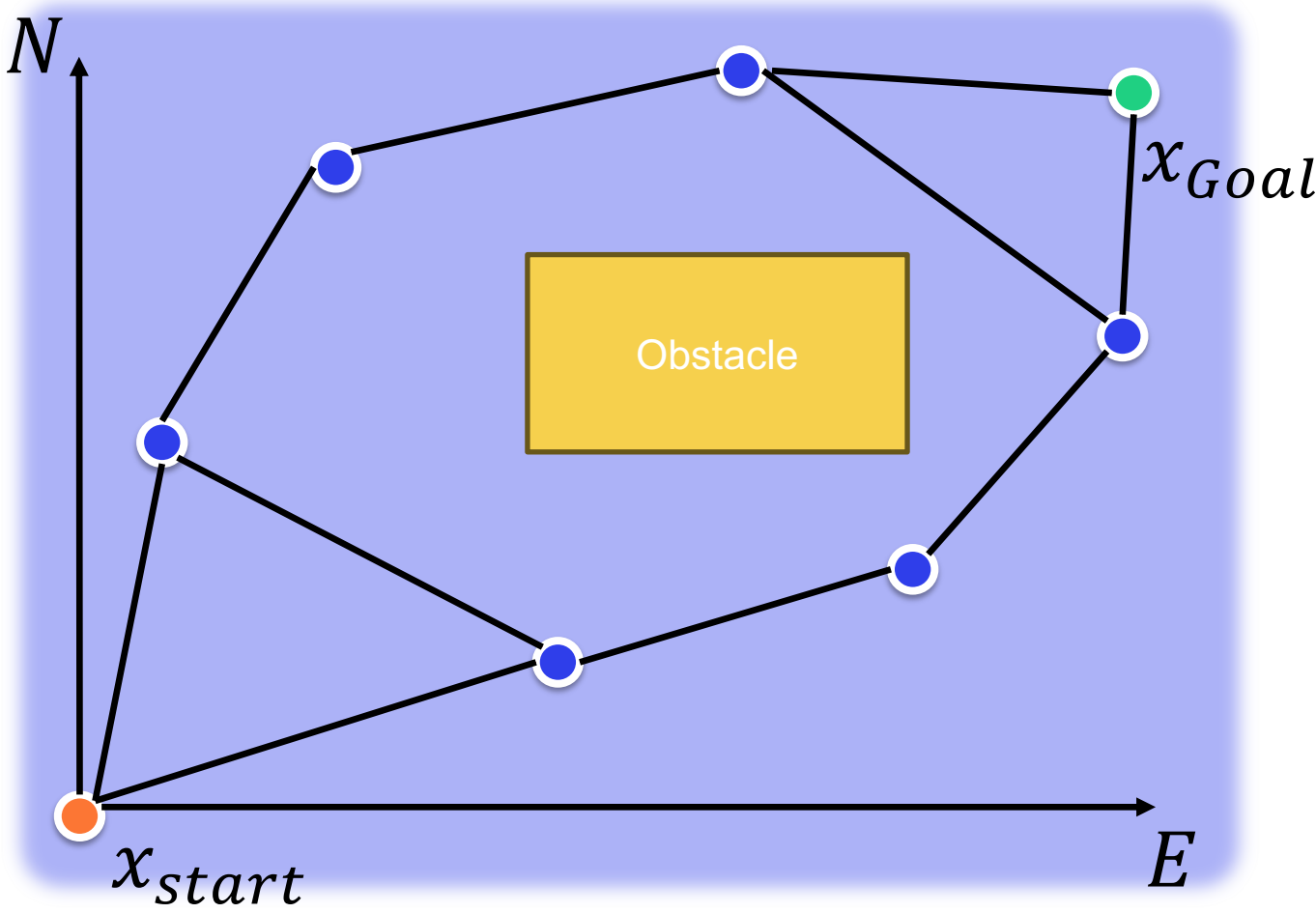
$$q' = f(q, u) \qquad u \in U(q), \quad q \in \mathcal{C}$$



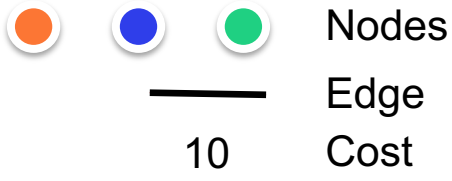
LaValle (2006)

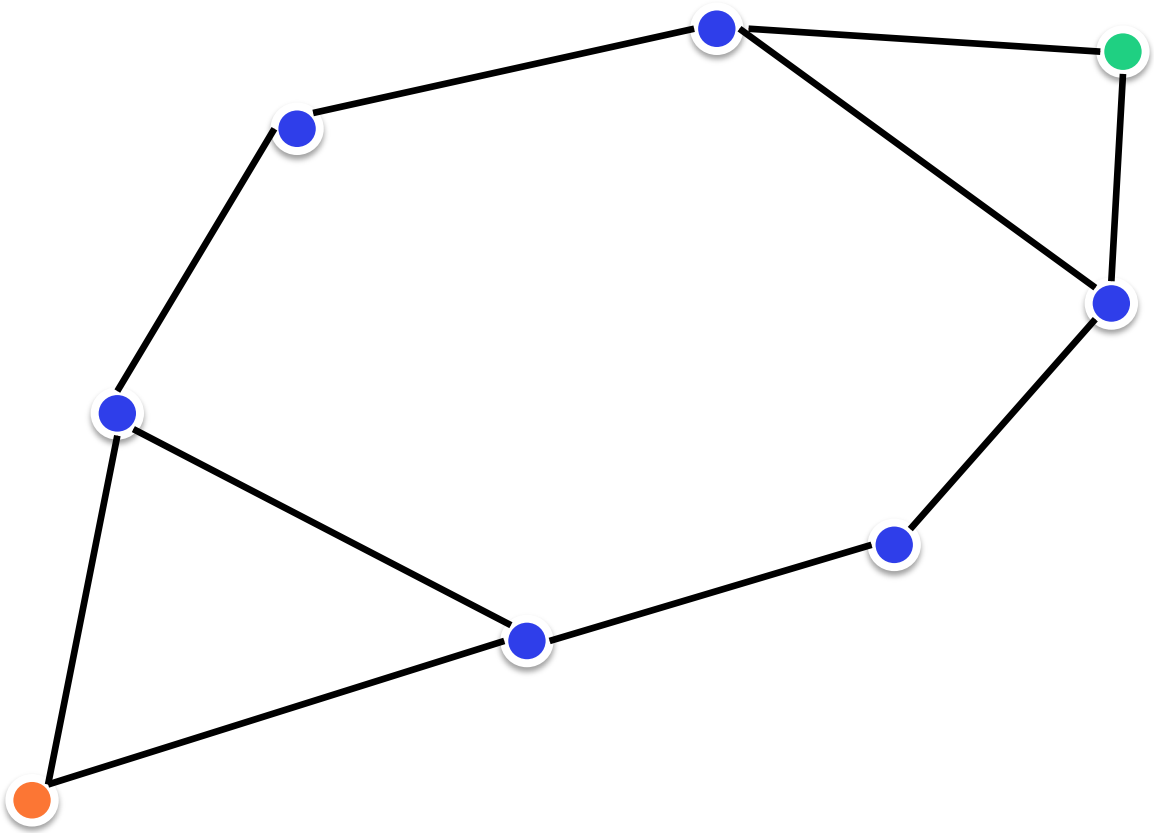
Discretization of configuration spaces

- Configuration spaces are, in general, continuous spaces (e.g., R^2 in the example)
- We simplify the path planning problem by discretizing the space, e.g.,
  - Gridding
  - Random sampling free configurations
- Graphs become powerful computational tools for representing the configuration space

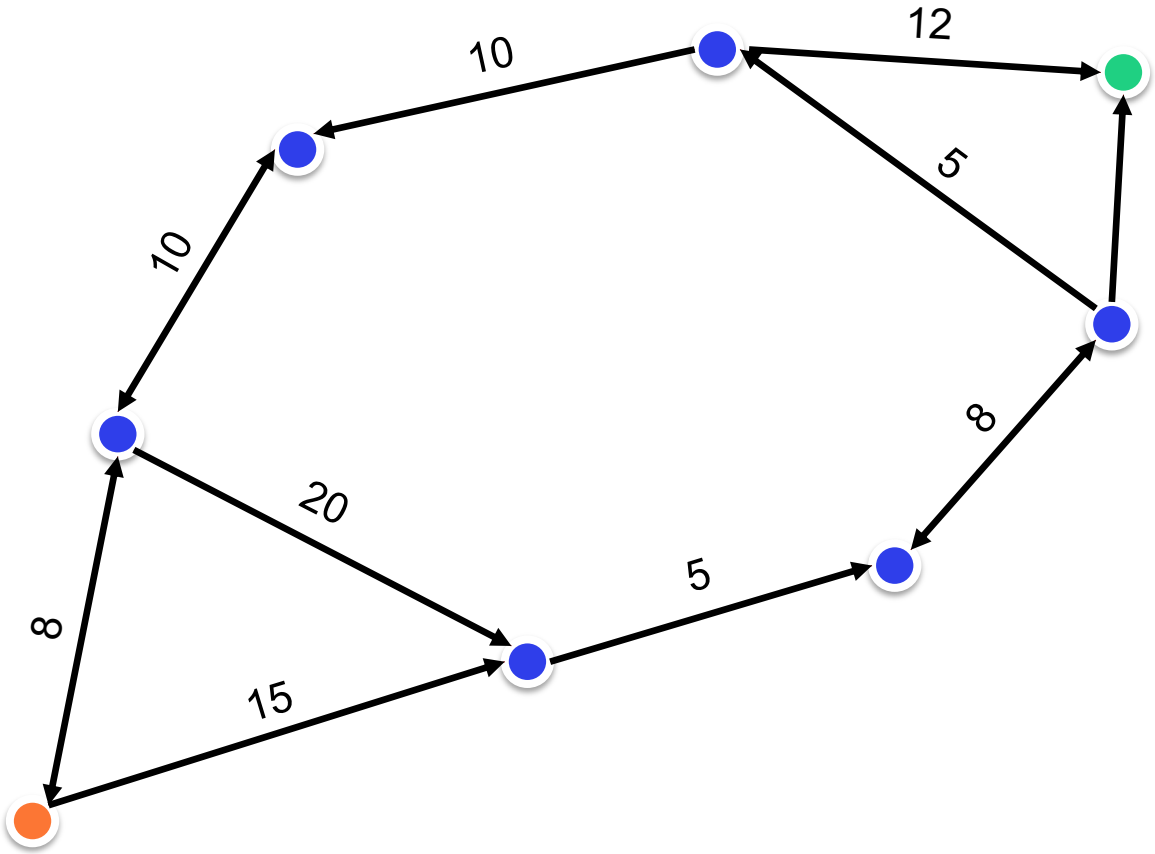


Graphs





Unweighted undirected graph



Weighted directed graph

Map-based planning algorithms

- Distance transform
- Voronoi roadmap method
- Probabilistic roadmap method
- Dijkstras Algorithm
- Rapidly-exploring random tree (RRT)

Map-based planning algorithms

- **Distance transform**
- Voronoi roadmap method
- Probabilistic roadmap method
- Dijkstras Algorithm
- Rapidly-exploring random tree (RRT)

Distance transform

- Represent the environment as a zero-matrix with a single goal with a value 1
- The distance transform is a matrix of equal size, but each element denotes the distance to the goal

0	0	0
0	1	0
0	0	0

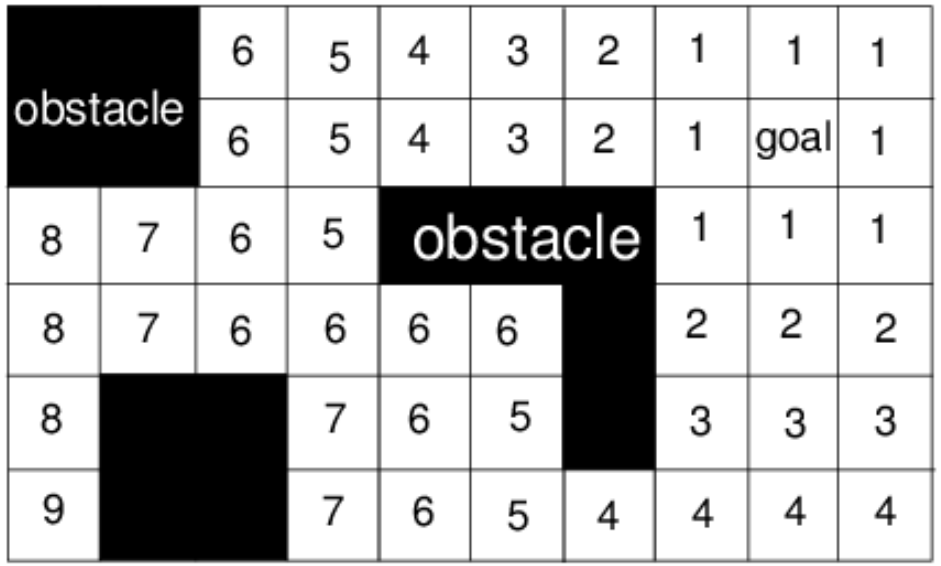
Environment

1.41	1	1.41
1	0	1
1.41	1	1.41

Distance transform

Distance transform

- Still works for environments with obstacles
- Just account for the obstacle when computing the distance
- Wherever the robot is placed in the map, find the goal by moving to the neighboring cell with the lowest value

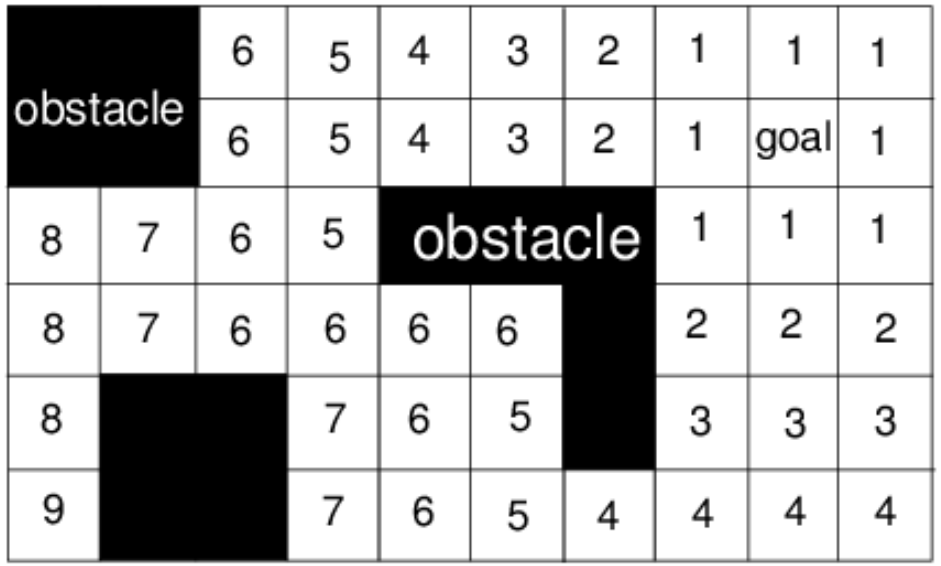


Note: Uniform cost (no diagonal cost)

Distance transform

- Effectively, the planning is a ball rolling down hill with the valley at the goal

- Constructing a distance transform is computationally expensive
  – More so if the environment is dynamic with moving obstacles



Note: Uniform cost
(no diagonal cost)

Distance Transform Recap

- Benefits
  - Lightweight in the query phase (”Roll the ball”)

- Downsides:
  - Computationally heavy in the preprocessing/planning phase
  - Changing the goal, requires complete replanning

Map-based planning algorithms

- Distance transform
- **Voronoi roadmap method**
- Probabilistic roadmap method
- Dijkstras Algorithm
- Rapidly-exploring random tree (RRT)

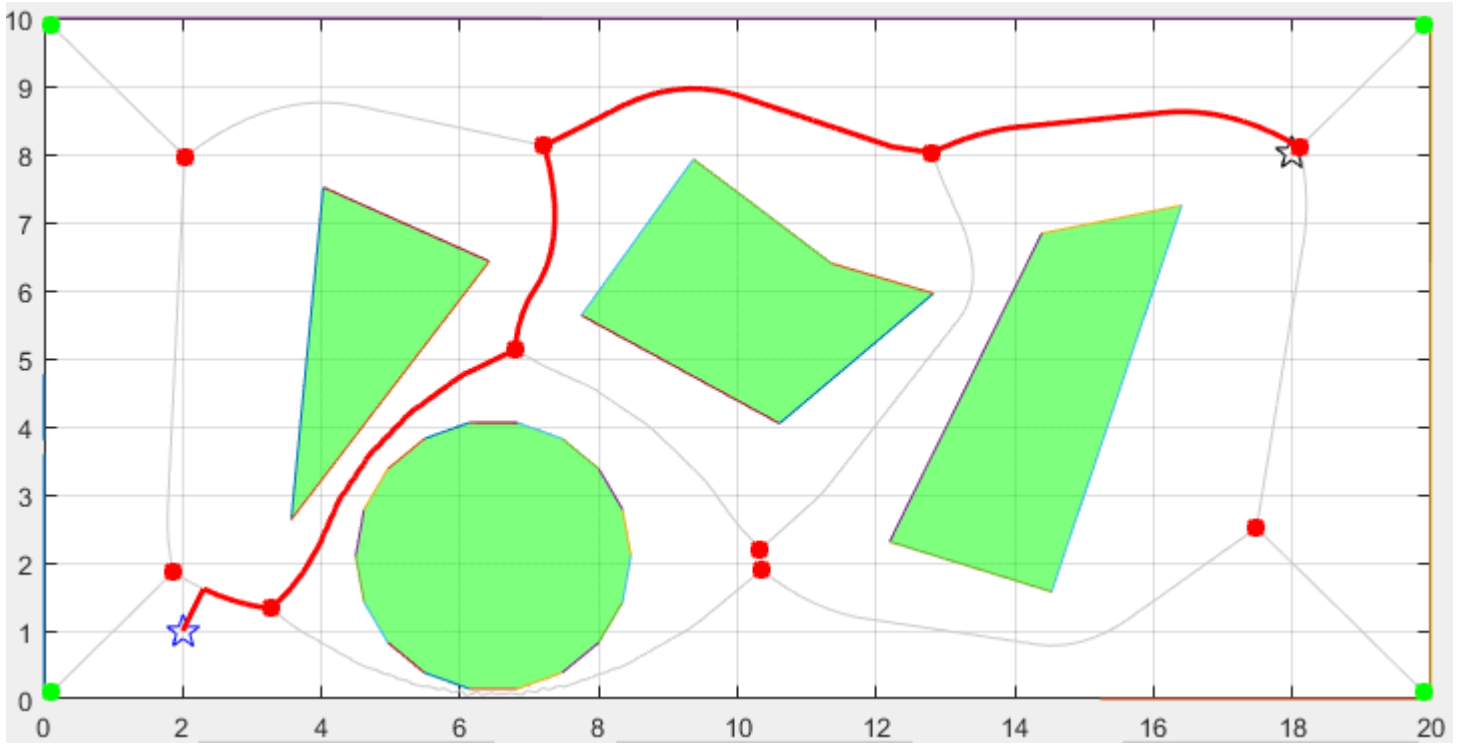
Voronoi roadmap method

- The heavy planning phase has led to the development of roadmaps that support moving goals

- Create a rough plan from a starting position to the goal
  - If the start position changes, only update the local map from your start position to the start position of the rough plan
  - Similarly with the goal – only update the local map from the rough plan to the goal position

Voronoi roadmap method

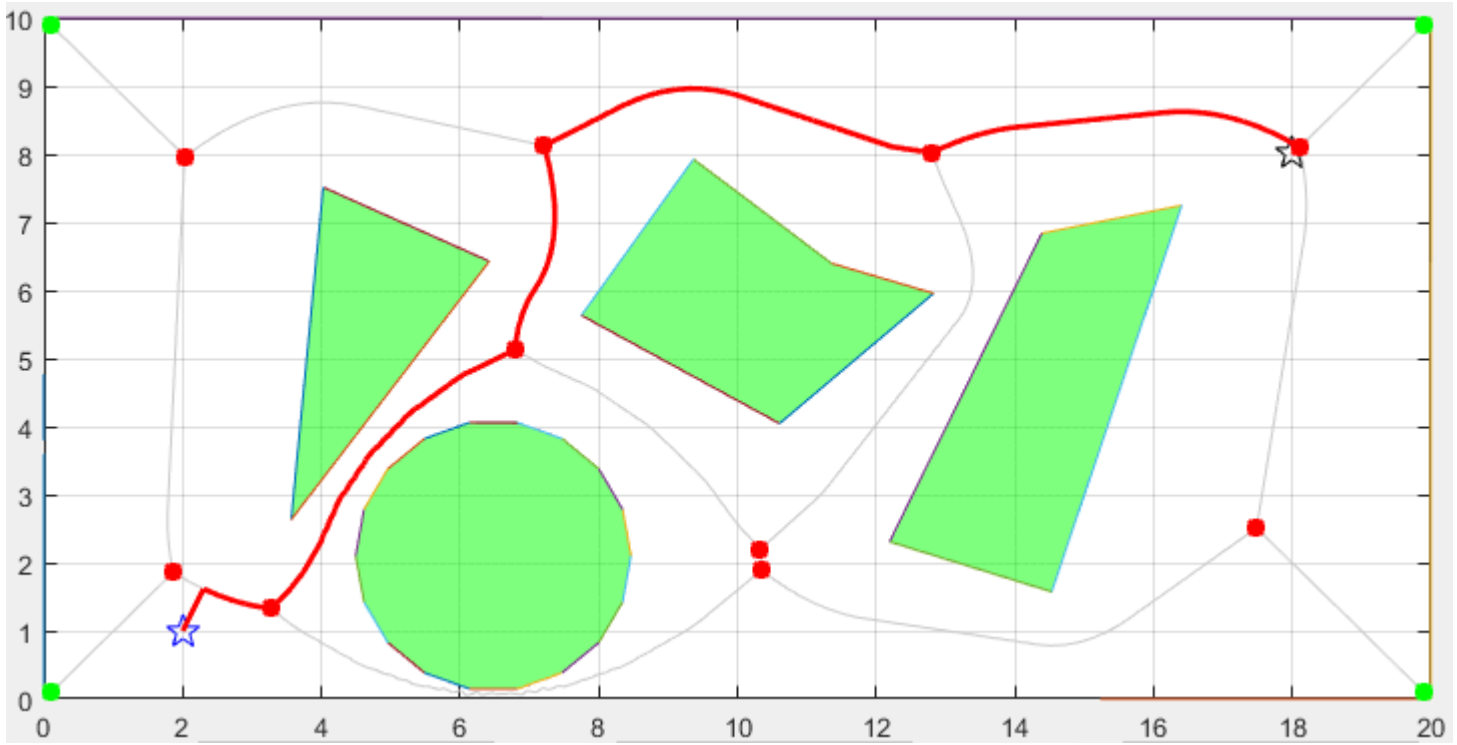
- To achieve full map coverage, simply create a large number of pre-planned roadmaps
- Obstacle-free path planning then becomes navigating all the roadmaps



Voronoi roadmap method

- One way to generate the roadmaps is with a Voronoi diagram

- Each node in the Voronoi becomes a “train station”, and you only need to find your way to and from the nearest train station



Voronoi Illustrative Example

Navigating via the Copenhagen Metro:

- The “map” of train stations is given
  - Locally plan to the nearest train station
  - Take train to station nearest your goal
  - Locally plan from the station to the goal

Map-based planning algorithms

- Distance transform
- Voronoi roadmap method
- **Probabilistic roadmap method**
- Dijkstras Algorithm
- Rapidly-exploring random tree (RRT)

Probabilistic roadmap method (PRM)

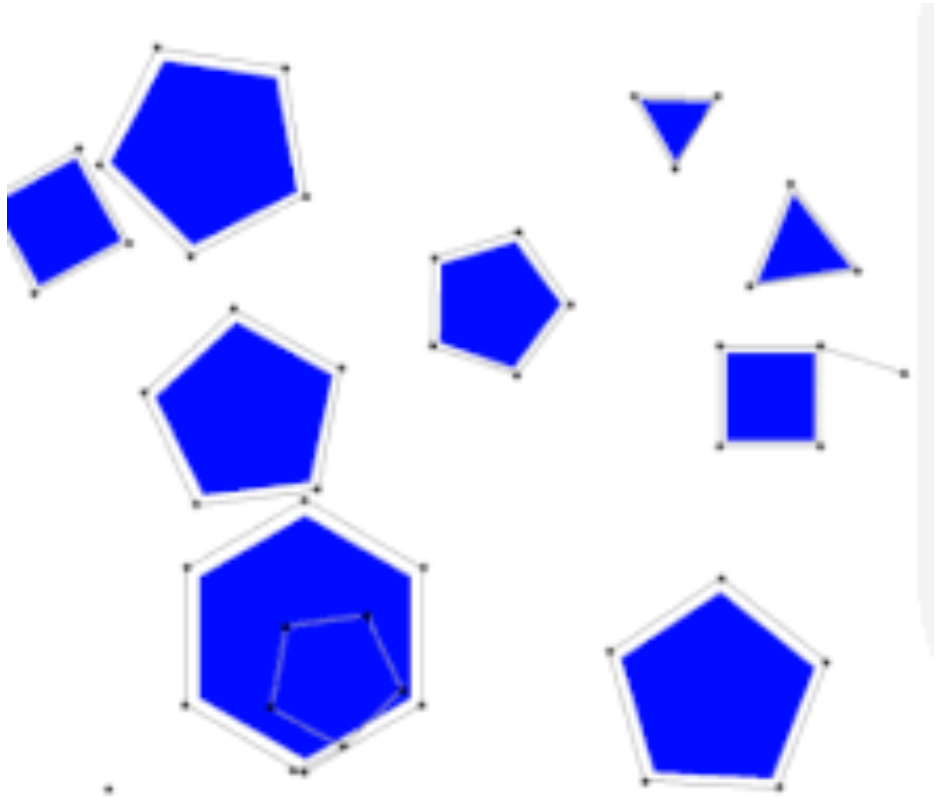
- Instead of representing the environment on a grid, randomly (and sparsely) sample positions on the map (making sure the positions are in free space)
  - Check that none of the connected nodes are located on opposite sides of an obstacle

- Connect all nodes/positions with their respective nearest neighbors
  - The vertices connecting the neighboring nodes are weighted according to how close they are to the goal

- PRMs works in continuous environments as we just randomly sample positions (like we did with the particle filter)

Probabilistic roadmap method (PRM)

- The more nodes we sample in the map, the higher number of edges we can traverse
- To traverse the roadmap, select the node with the lowest weight, i.e. the neighboring node closest to the goal
- You can either manually add nodes to represent the start/goal positions, or you can treat the roadmap like the Voronoi case, and compute a local map to the start and goal position



A PRM with increasing number of nodes

Demo Time

- Probabilistic Roadmap Method.cdf
- Probabilistic Roadmap Method For Robot Arm.cdf
- Probabilistic Roadmap Method In 3D.cdf
- PRM With Seven Link Articulated Robot.cdf

Probabilistic roadmap method (PRM)

Benefits:

- We only need to execute the planning phase *once*, any replanning can be done by querying the graph of nodes
- The start/goal position agnostic
- Works in continuous space (not constrained to a grid)

Drawbacks

- Due to sampling: Optimal in graph space, *not* in state/configuration space
- Re-planning results in a new graph – i.e. we won't get the same solution
- Not guaranteed a solution – especially if the path goes through a narrow passage
  - The same principle also means long narrow corridors are unlikely to be explored

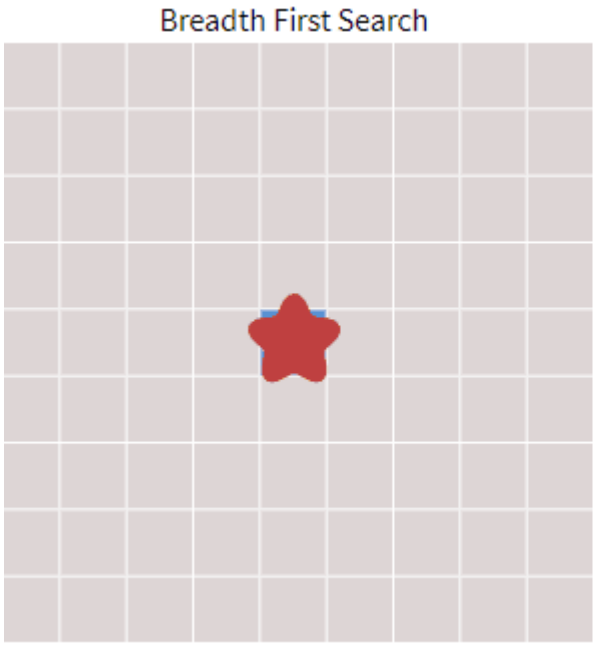
We have a graph; how do we find the optimal path?

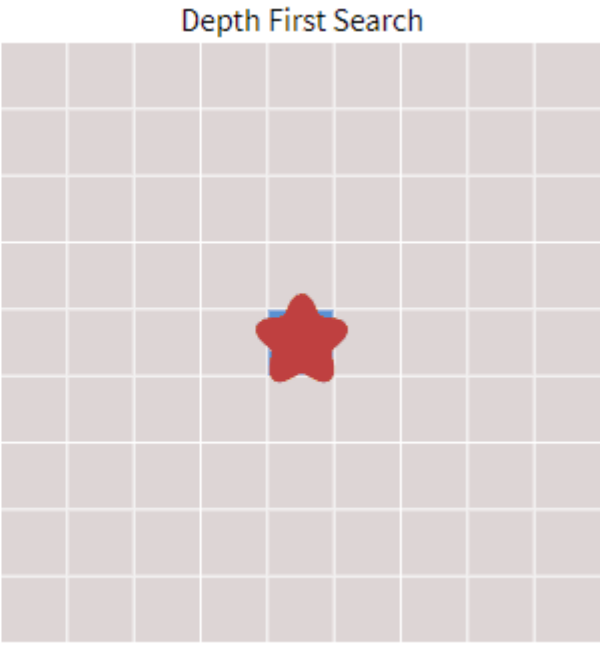
Searching Graphs

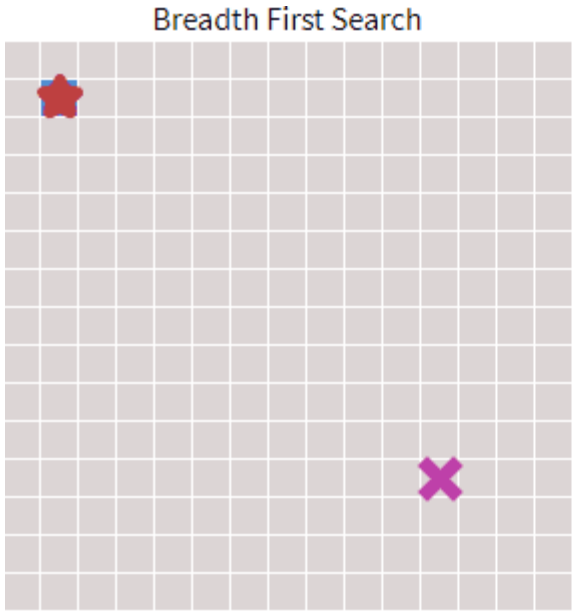
Many common graph search methods are able to solve these planning problems.

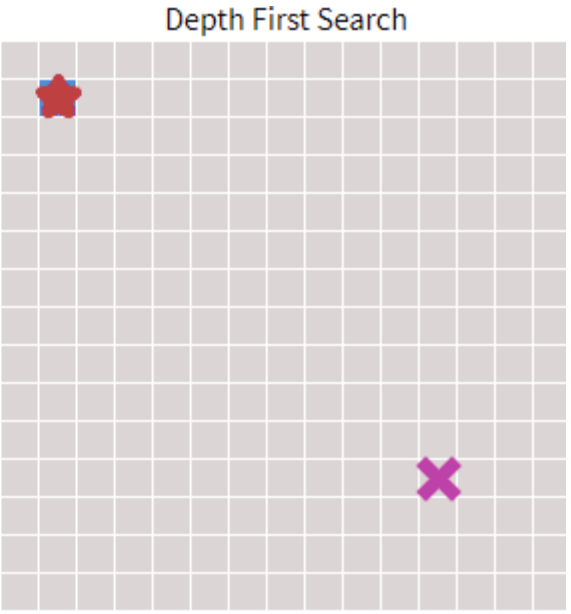
Such as:

- Breadth First (FIFO)
- Depth First (LIFO)





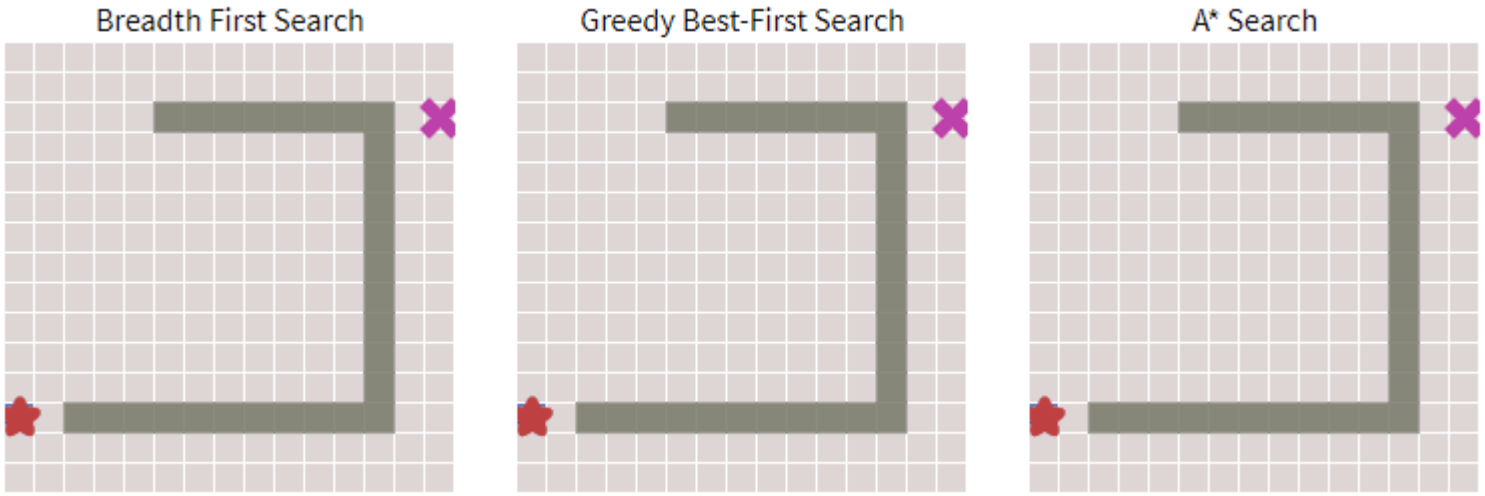




https://cs.stanford.edu/people/abisee/tutorial/

Search algorithms

- Improvements using heuristics
  - Dijkstra's algorithm (special case of Breadth First)
  - A* (A-star)
  - Best First (Greedy search)



- Improvements using various search directions:
  - Forward search, Backward search, Bi-directional search

https://cs.stanford.edu/people/abisee/tutorial/

Map-based planning algorithms

- Distance transform
- Voronoi roadmap method
- Probabilistic roadmap method
- **Dijkstras Algorithm**
- Rapidly-exploring random tree (RRT)

Dijkstra's Algorithm

- A solution to the single-source shortest path problem in graph theory.
- Works on both directed and undirected graphs. However, all edges must have nonnegative weights.
- Approach: Greedy
- Input: Weighted graph $G = \{E, V\}$ and source vertex $v \in V$, such that all edge weights are nonnegative
- Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

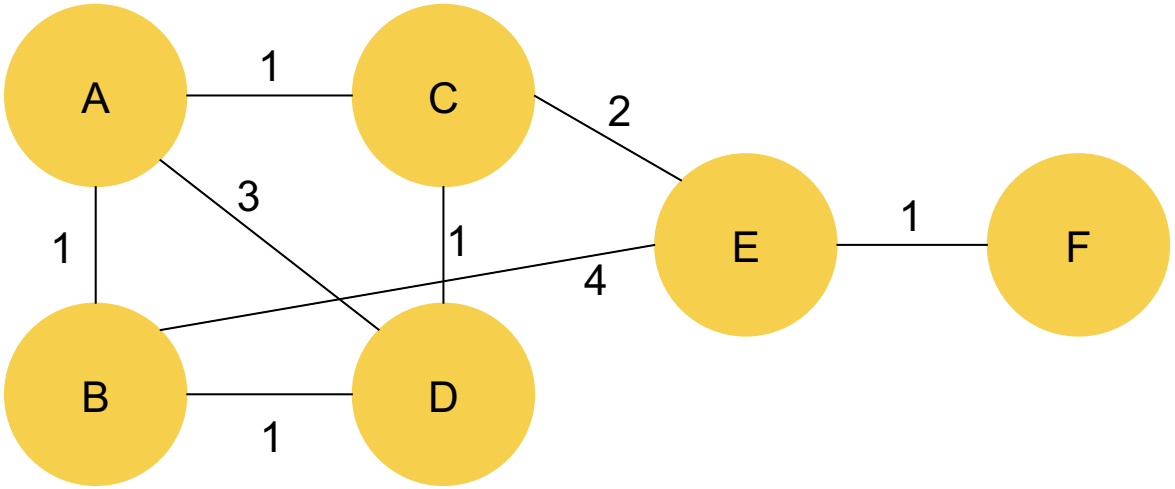
Dijkstra's algorithm

- Generate a table of shortest path between a set of nodes

- The idea is to explore outwards from your starting position

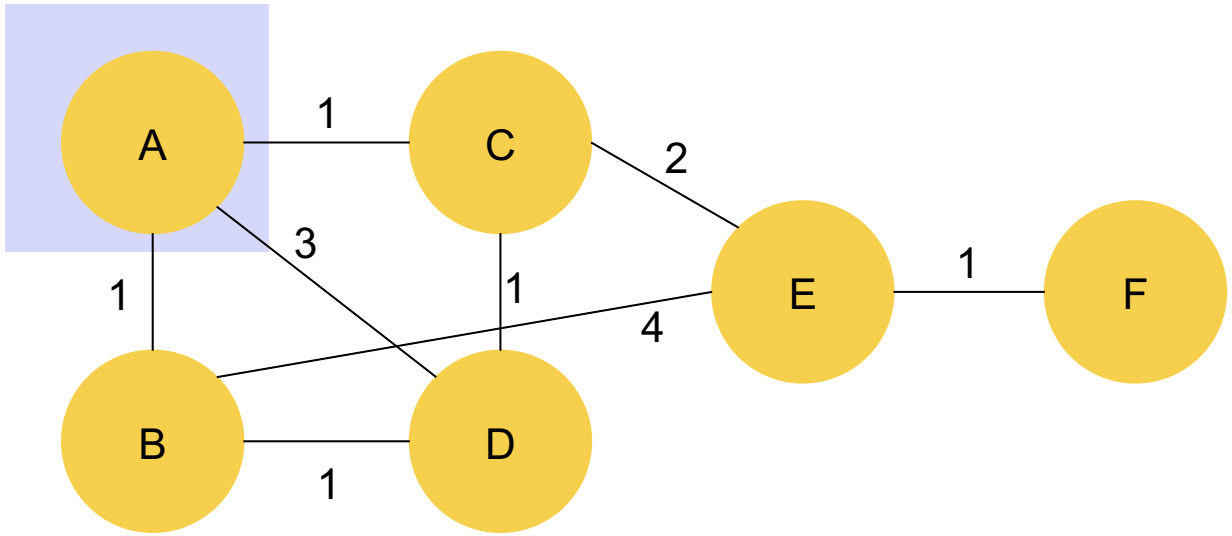
```
FORWARD_LABEL_CORRECTING(x_G)
 1   Set C(x) = ∞ for all x ≠ x_I, and set C(x_I) = 0
 2   Q.Insert(x_I)
 3   while Q not empty do
 4       x ← Q.GetFirst()
 5       forall u ∈ U(x)
 6           x' ← f(x, u)
 7           if C(x) + l(x, u) < min{C(x'), C(x_G)} then
 8               C(x') ← C(x) + l(x, u)
 9               if x' ≠ x_G then
10                   Q.Insert(x')
```

Dijkstra's algorithm



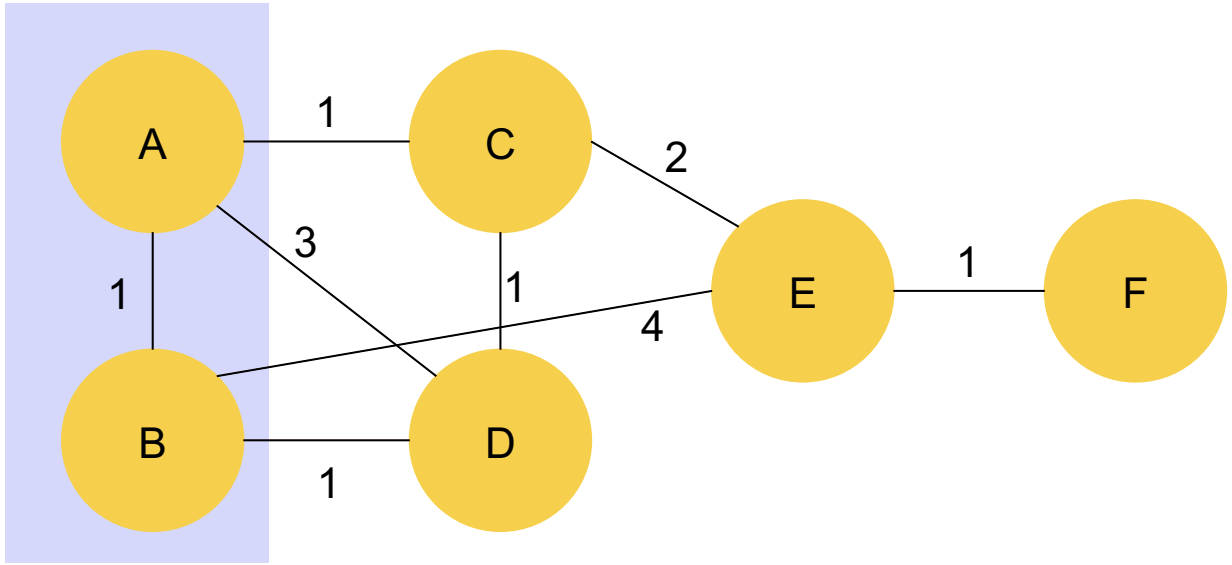
Node	D()

Dijkstra's algorithm



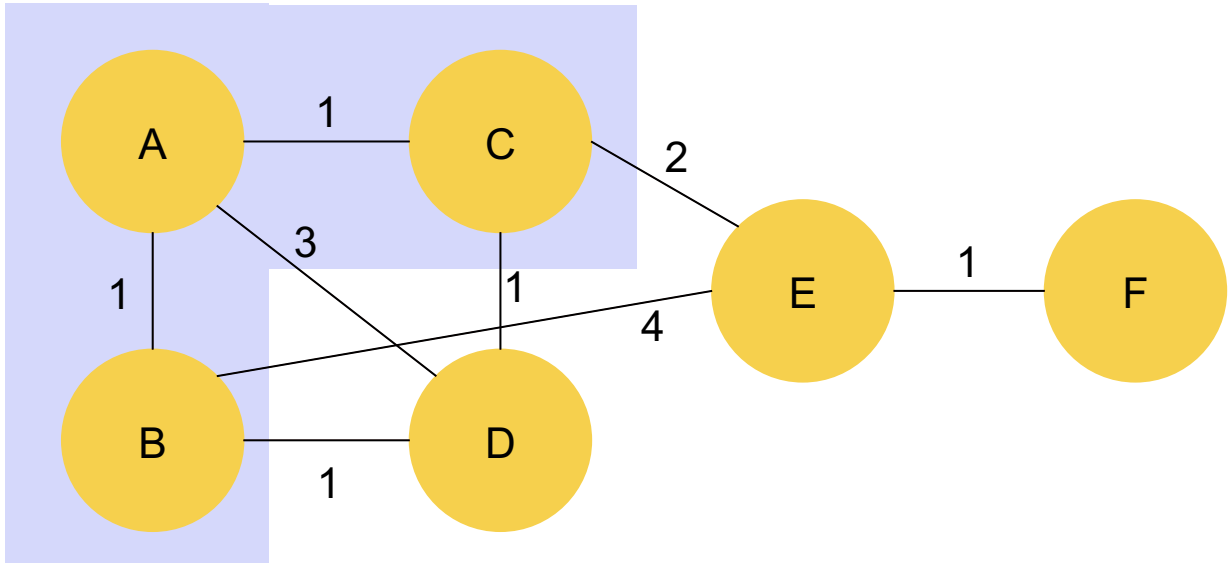
Node	D()
A	0
B	∞
C	∞
D	∞
E	∞
F	∞

Dijkstra's algorithm



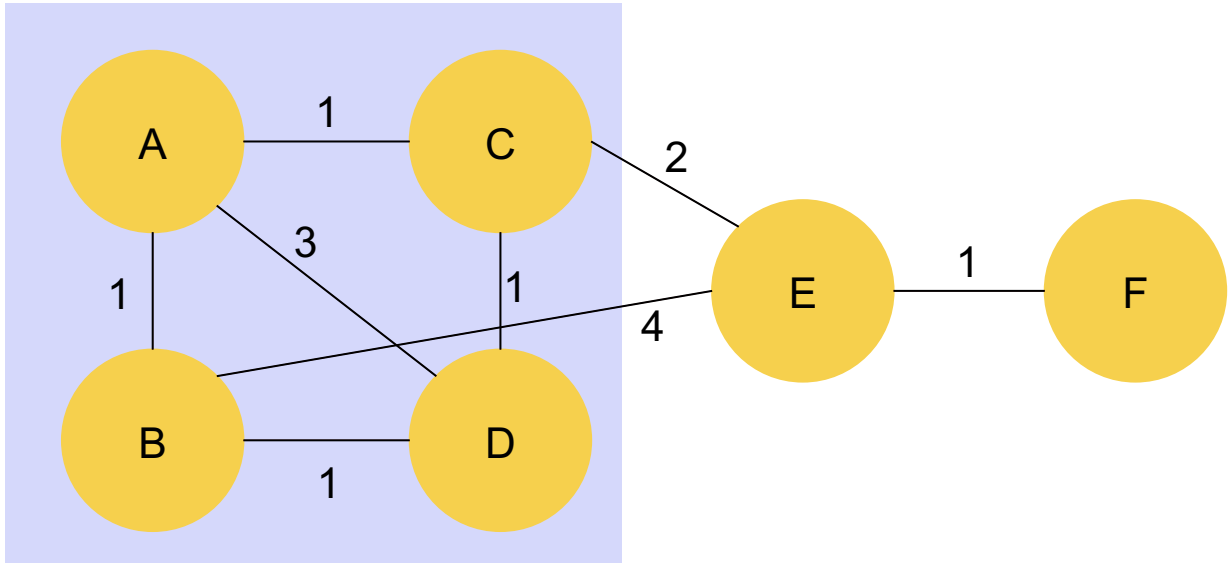
Node	D()
A	0
B	1
C	∞
D	∞
E	∞
F	∞

Dijkstra's algorithm



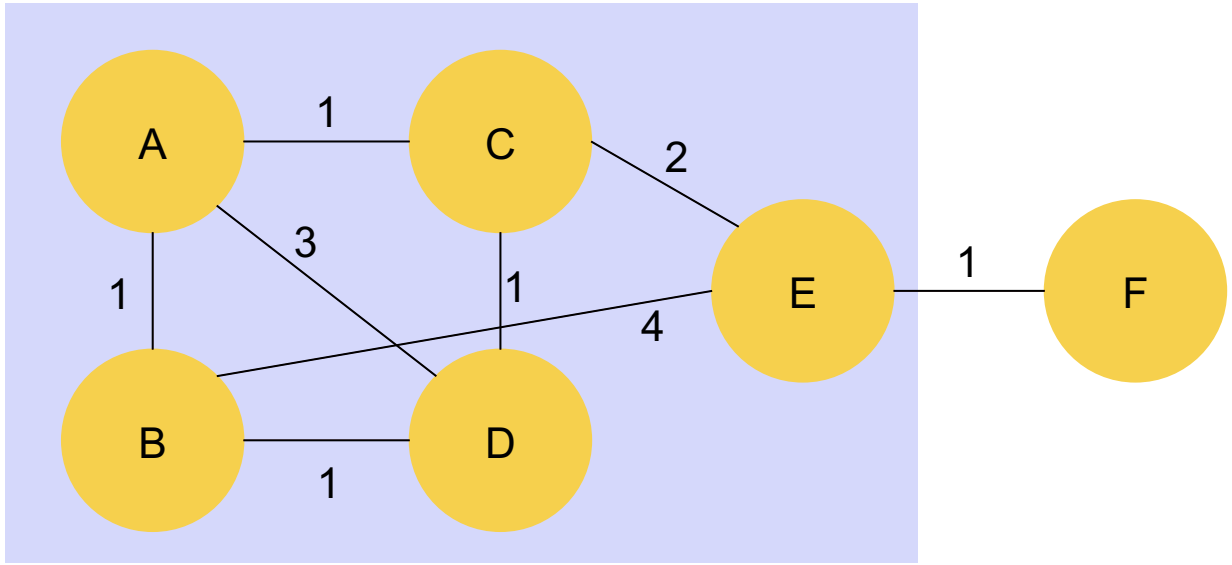
Node	D()
A	0
B	1
C	1
D	∞
E	∞
F	∞

Dijkstra's algorithm



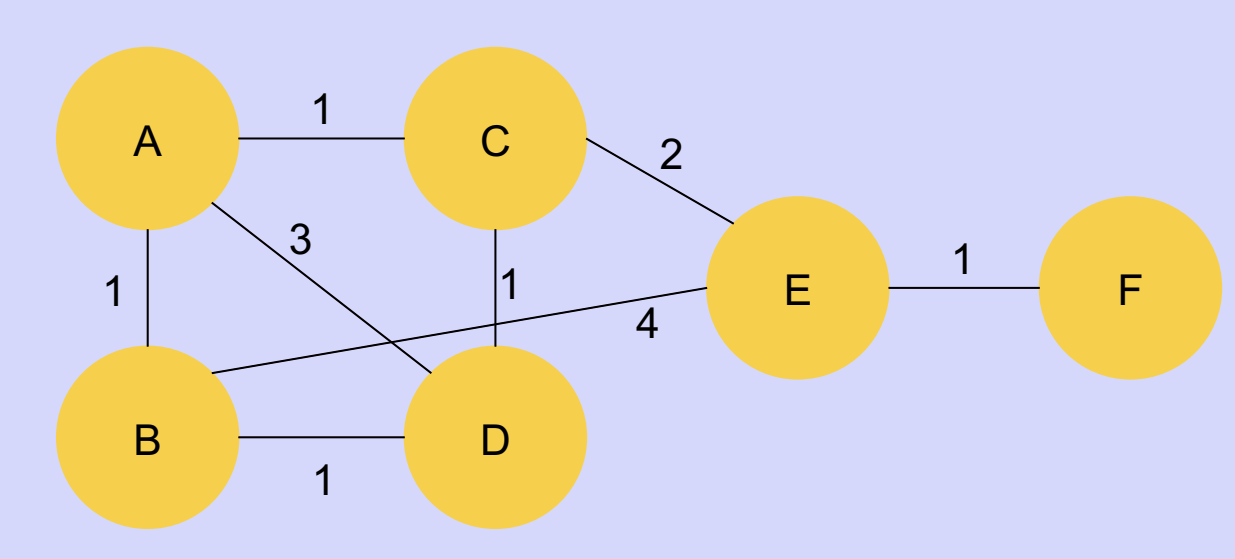
Node	D()
A	0
B	1
C	1
D	2
E	∞
F	∞

Dijkstras algorithm



Node	D()
A	0
B	1
C	1
D	2
E	3
F	∞

Dijkstra's algorithm



Node	D()
A	0
B	1
C	1
D	2
E	3
F	4

Dijkstra's algorithm

- The nodes are just abstract representations of positions on the map or cells in a grid
- Similarly to the previous algorithms, the cost of moving along a vertice can be chosen to optimize some behavior (time, shortest path, etc.)

Demo Time

- Dijkstras Demo

Quick sum up

- So far we have introduced a couple of ways to path plan with decreasing planning costs
- The Distance Transform is confined to a grid representation of the environment
  – This can be difficult to scale to large maps with high resolution (many cells)
- The Voronoi method still requires replanning of local maps
- Probabilistic Roadmaps are useful high dimensional state/configuration spaces
- Djikstras algorithm yields the shortest path in a graph

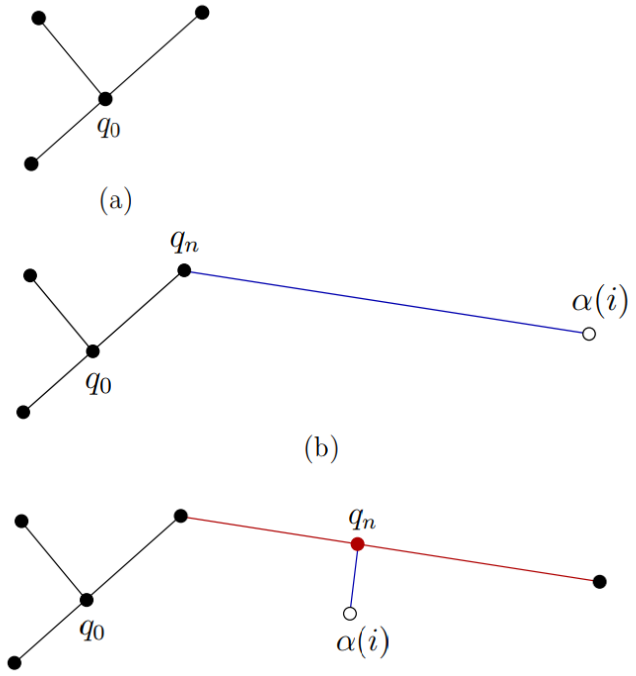
Map-based planning algorithms

- Distance transform
- Voronoi roadmap method
- Dijkstras Algorithm
- Probabilistic roadmap method
- **Rapidly-exploring random tree (RRT)**

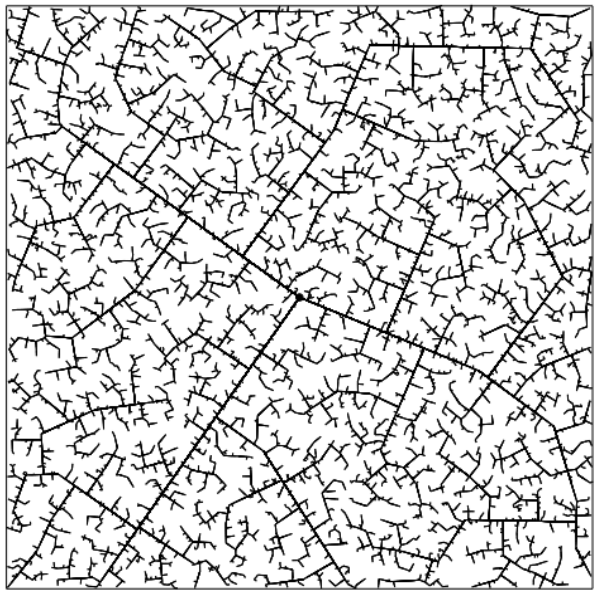
Sneak peak: Rapidly-exploring Random Tree (RRT)

- “Rapidly-exploring random trees: A new tool for path planning”, LaValle (1998)

- Step 1
- Step 2
- Step 3



45 iterations



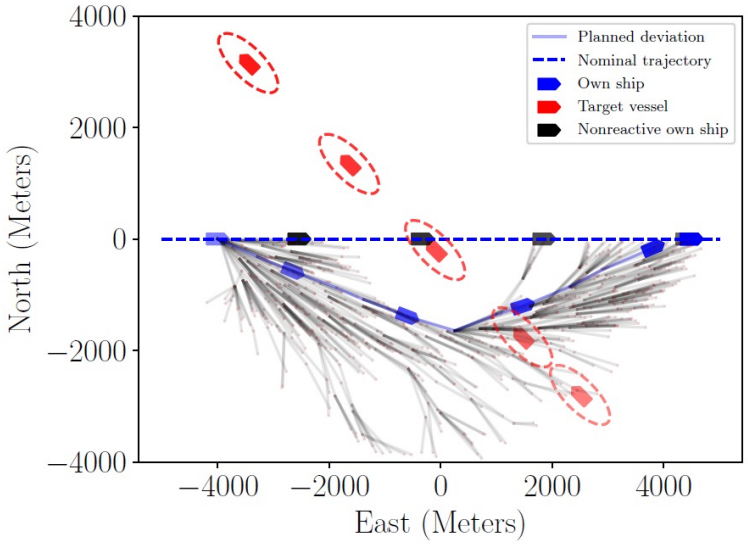
2345 iterations

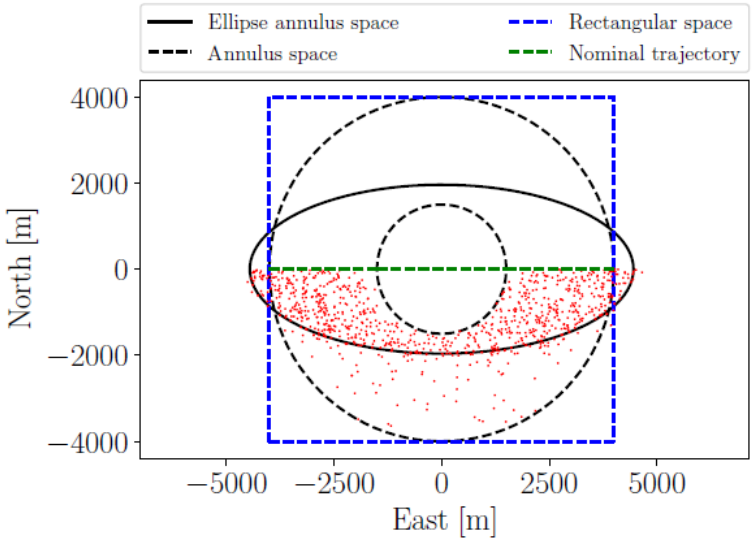
Demo Time

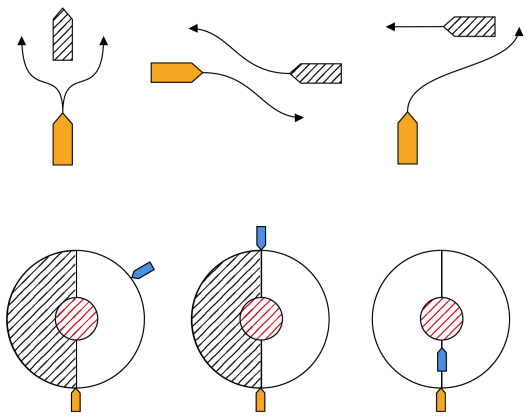
- RRT Demo

Marine Applications of Path Planning

- *Enevoldsen, T.T., Reinartz, C. and Galeazzi, R., 2021, May. COLREGs-Informed RRT* for collision avoidance of marine crafts. In 2021 IEEE International Conference on Robotics and Automation (ICRA) (pp. 8083-8089). IEEE.*
  - Encodes the marine “Rules of the road” into an informed sampling space.
  - Increased probability for a given sample of decreasing the found solution cost.







The Vast Landscape of Planning Algorithms

We have barely scratched the surface..

- A*, D*, theta*
- Artificial Potential Fields (APF)
- Various QPs, NLPs, Multi-objective Optimization routines..
- Particle Swarm Optimization (PSO)
- Genetic Algorithms
  – Ant Colony, Bee colony..
- RRT, RRT* and their variations
- PRM, PRM*
- FM, FMT*, …
- Reinforcement Learning (Learning feedback policies), etc.

Exercises

- Implement a probabilistic roadmap method (the algorithm can be found in *Planning Algorithms* that you read for today*).* To implement in ROS2, subscribe to the map topic and use that map for collision checks.

```
BUILD_ROADMAP
1  G.init(); i ← 0;
2  while i < N
3      if α(i) ∈ C_free then
4          G.add_vertex(α(i)); i ← i + 1;
5          for each q ∈ NEIGHBORHOOD(α(i),G)
6              if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
7                  G.add_edge(α(i), q);
```

- Where $\mathcal{G}$ is the graph of connected nodes, $\mathcal{C}_{free}$ denotes the free space on the map, $\alpha(i)$ are all the nodes to be added to the roadmap
- Assume a constant cost value for all nodes
- You can use any nearest neighbor implementation (your own or an off-the-shelf)
- **Hint**: create a function that takes as input a number of randomly sampled node positions and returns the graph $\mathcal{G}$

Exercises

- Implement a probabilis… n be found in *Planning Algorithms* that you rea… subscribe to the map topic and use that map for c…

```
BUILD_ROADMA
1  G.init(); i ← 0
2  while i < N
3      if α(i) ∈ C
4          G.add_
5          for ea
6              if (                    CT(α(i), q)) then
7
```

- Where $\mathcal{G}$ is the graph o… free space on the map, $\alpha(i)$ are all the nodes to be added to the roadmap
- Assume a constant cost value for all nodes
- You can use any nearest neighbor implementation (your own or an off-the-shelf)
- **Hint**: create a function that takes as input a number of randomly sampled node positions and returns the graph $\mathcal{G}$

Navigation Colab Notebook

Exercises

- ~~Add a start and goal position to the graph (effectively run the previous algorithm 2 more times) and ensure they are connected to the graph~~

- Query the plan from the previous exercise by searching for the list of vertices in the graph that connects the start and the goal

- **Hint:** create a function that takes as input a start, goal, and a graph, and returns a list of the nodes on the queried path

Exercises

- To execute the path:
  1. Turn the turtlebot to face the direction of the next node in the path
  2. Drive straight until the turtlebot reaches approximately the next node within some tolerance (depends on how fast you want to move the turtlebot)
- To visualize the computed path before you start to drive the turtlebot, you can use the Path ROS2 interface: https://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Path.html

Links to interactive demonstrations

- https://demonstrations.wolfram.com/ProbabilisticRoadmapMethod/
- https://demonstrations.wolfram.com/ProbabilisticRoadmapMethodForRobotArm/
- https://demonstrations.wolfram.com/ProbabilisticRoadmapMethodIn3D/
- https://demonstrations.wolfram.com/ProbabilisticRoadmapMethodWithSevenLinkArticulatedRobot/
- https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/