DTU

Rasmus Andersen

34761 – Robot Autonomy

Exploration

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

Software frameworks

Robot autonomy stack

Project work + guest lectures

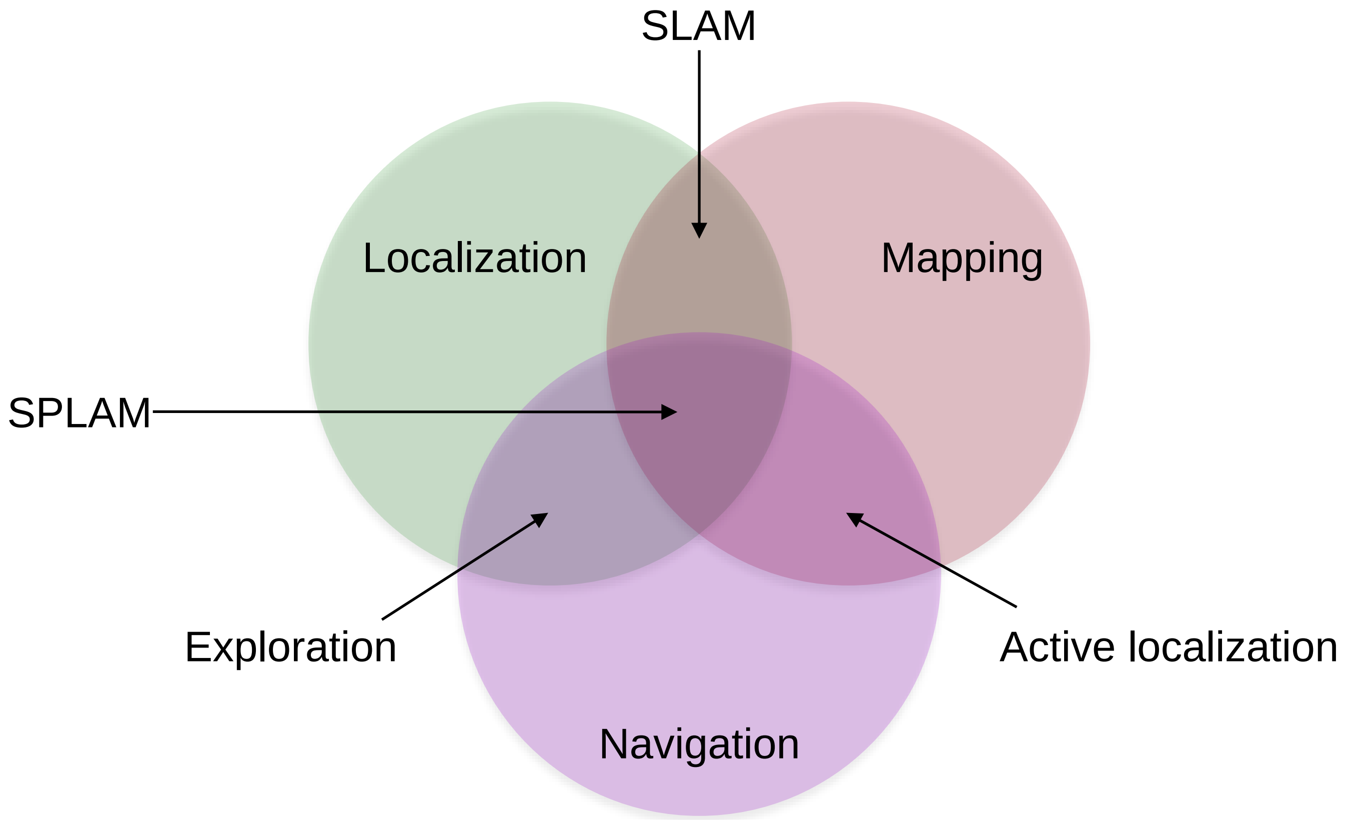
30th Jan	6th Feb	13th Feb	20th Feb	27th Feb	6th Mar	13th Mar	20th Mar	27th Mar	3rd Apr	10th Apr	17th Apr	24th Apr	1st May
Software architecture	Intro to ROS2	More on ROS2	Localization	Mapping	Localization wrt. maps	SLAM	Navigation	Easter	Exploration	Behavior Trees	Black Box Approaches	Group work	

Today

Outline for the next 7 weeks

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

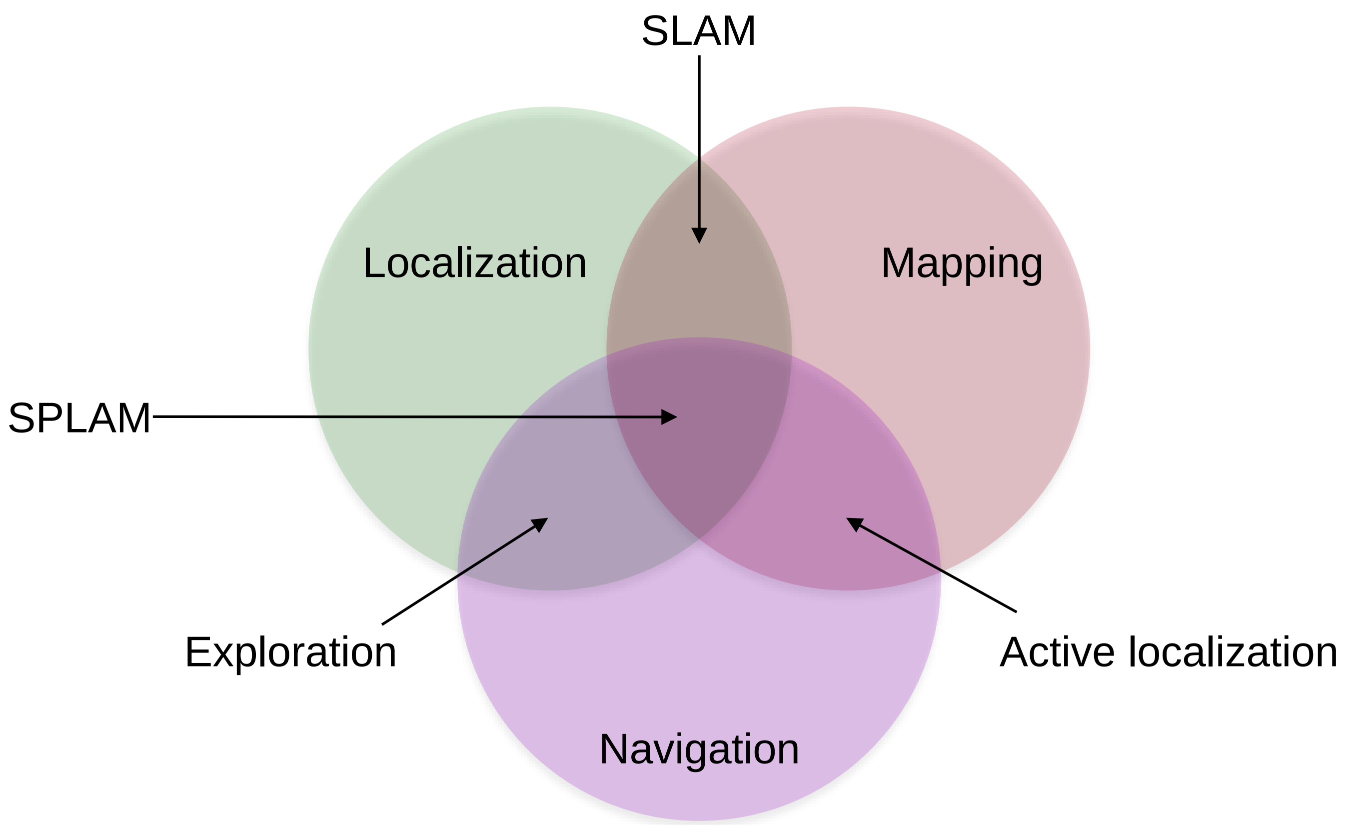
Topic of today



Outline for the next 7 weeks

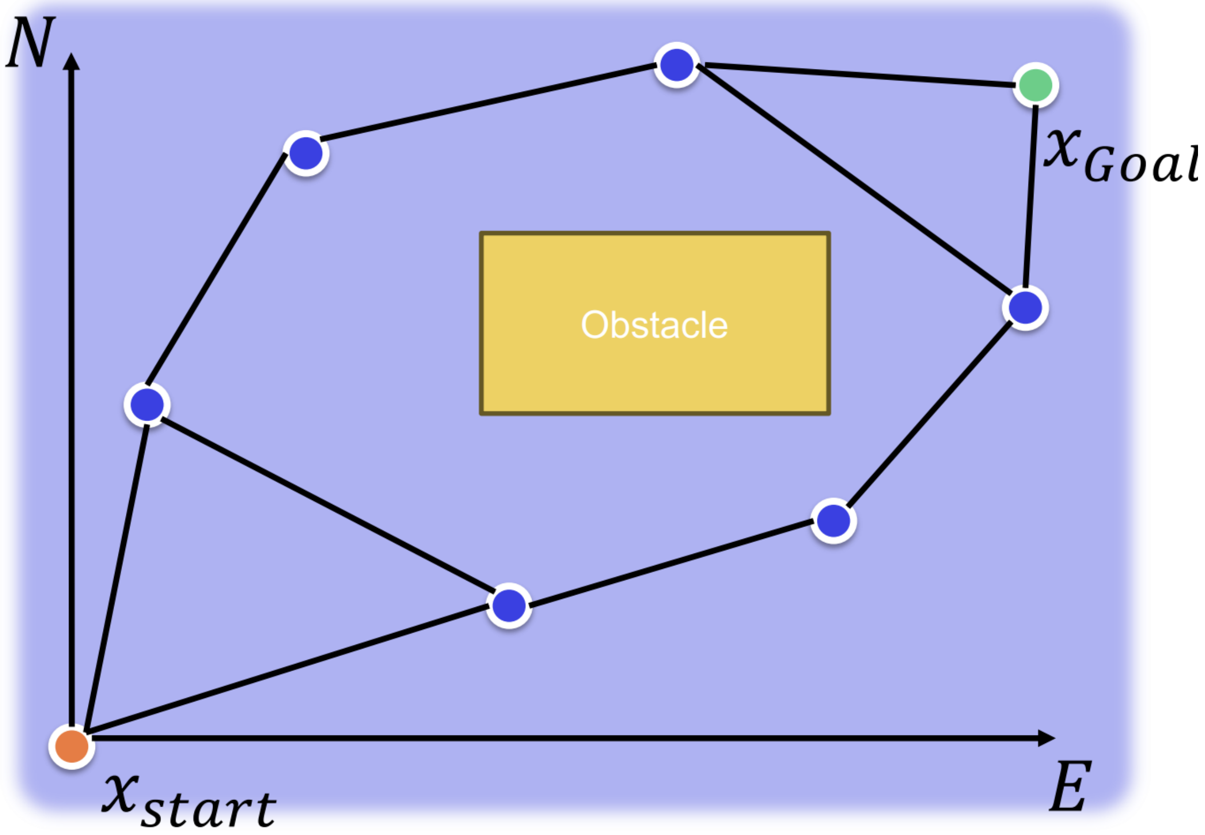
- Our own autonomy stack:
  1. Localization
  2. Mapping
  3. Navigation
  4. Exploration
     1. Depth-first search
     2. Breadth-first search
     3. Frontier-based exploration
     4. Next-best-view exploration
  5. Behaviour trees

Topic of today

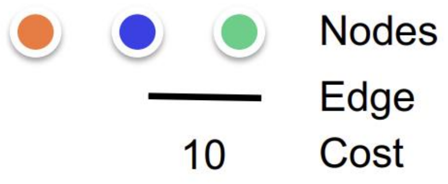


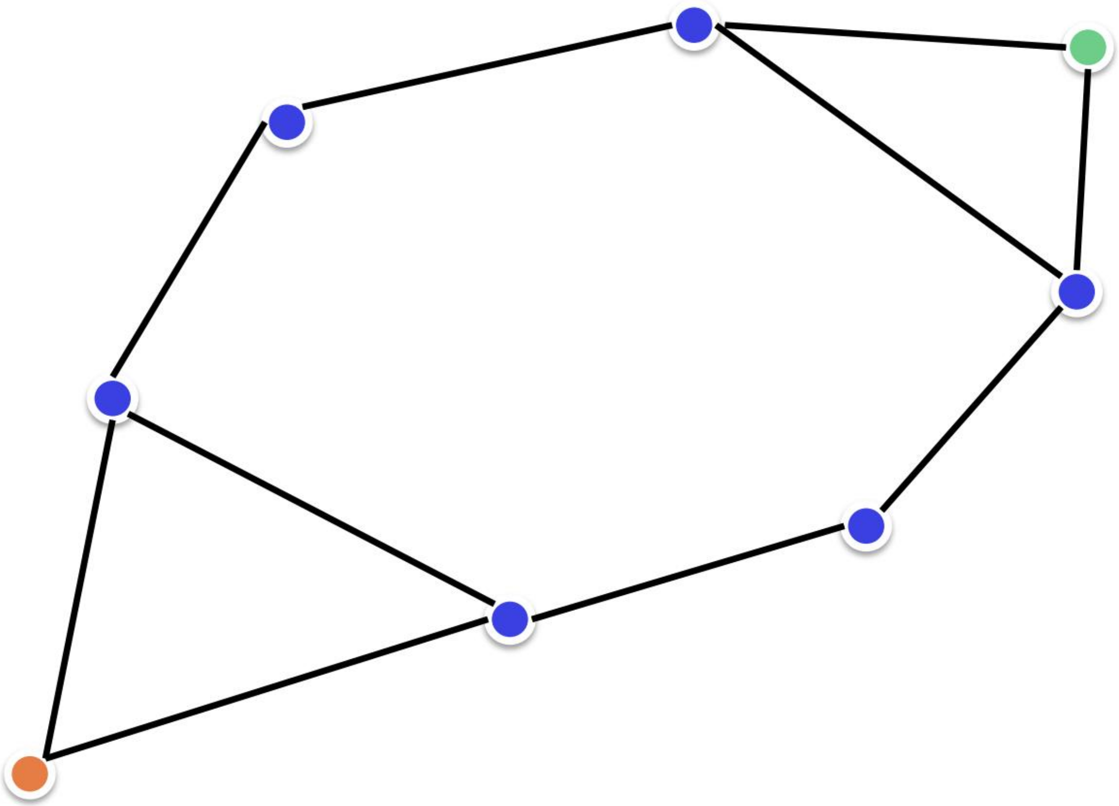
Recall from last lecture – discretization of configuration space

- 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

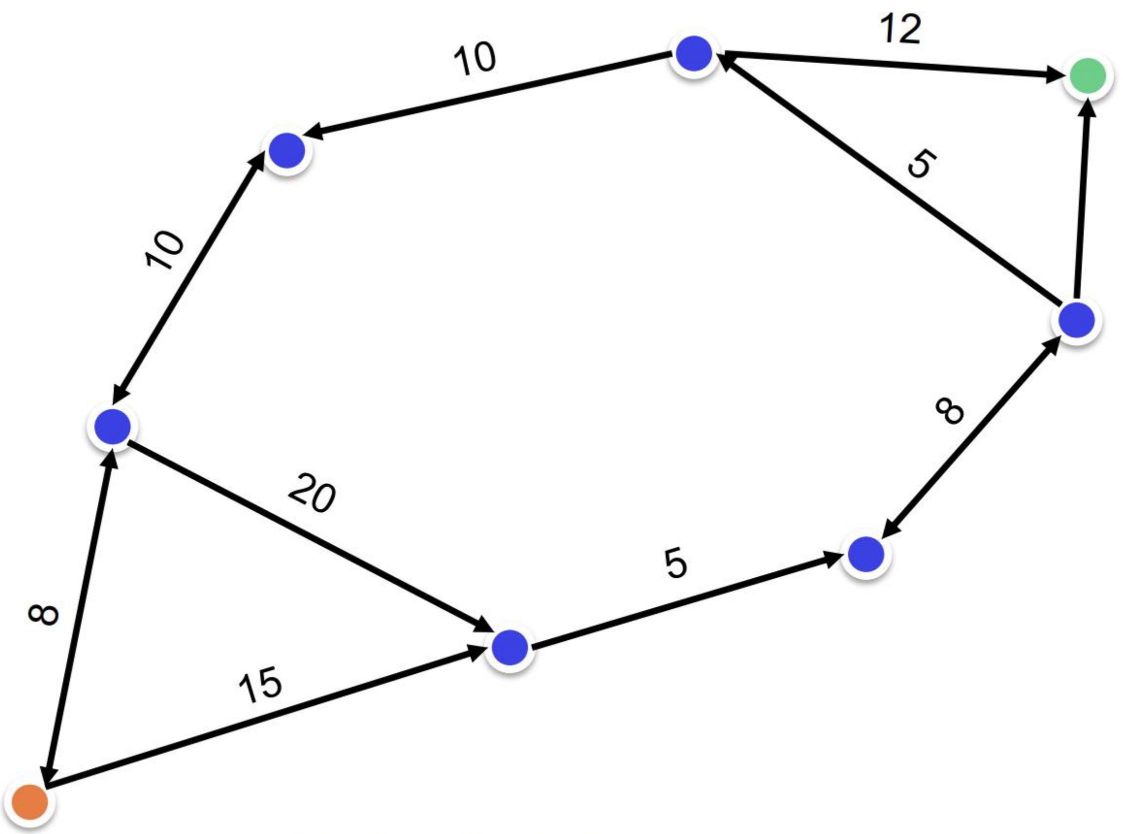


Graphs





Unweighted undirected graph



Weighted directed graph

Recall from last lecture – map-based planning algorithms

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

How is exploration different than SLAM?

In SLAM, we try to localize the robot while building a map

- i.e. there is nothing to dictate where or how the robot should move
- Purely passive, and can be done in post-processing

In exploration we try to maximize the map-building autonomously

- Ideally by quantifying what the robot “learns” by navigating to a new pose
- Has to be performed online, the algorithms dictate where the robot navigate to

NB: exploration is also different than *coverage*

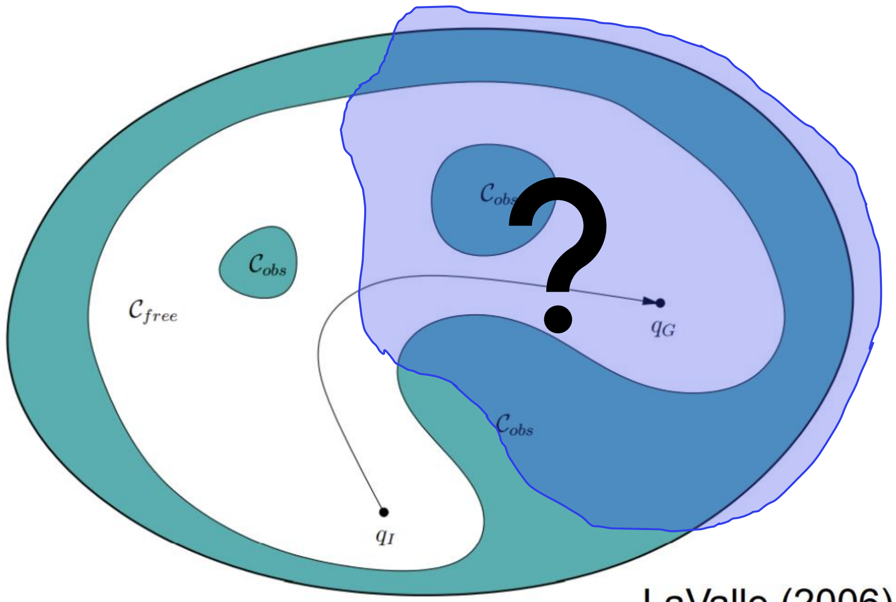
- Coverage problems assume a map and that we want the robot to optimally cover the map (e.g. vacuum cleaner or lawn mover)

Exploration applications

- Exploration is central to a range of indoor, outdoor, in-air and underwater applications for autonomous vehicles.
- At home: vacuum cleaner, lawn mower
- Air: surveillance with unmanned air vehicles
- Underwater: reef monitoring
- Underground: exploration of mines
- Space: terrain mapping for localization

The exploration problem

- We don't have a full map of the environment
  - C and therefore C_{free}, C_{obs} are only partially known
- The goal is to 'maximize' C, and less to navigate from point A to point B
  - i.e. produce a map configuration we can use for SLAM and navigation



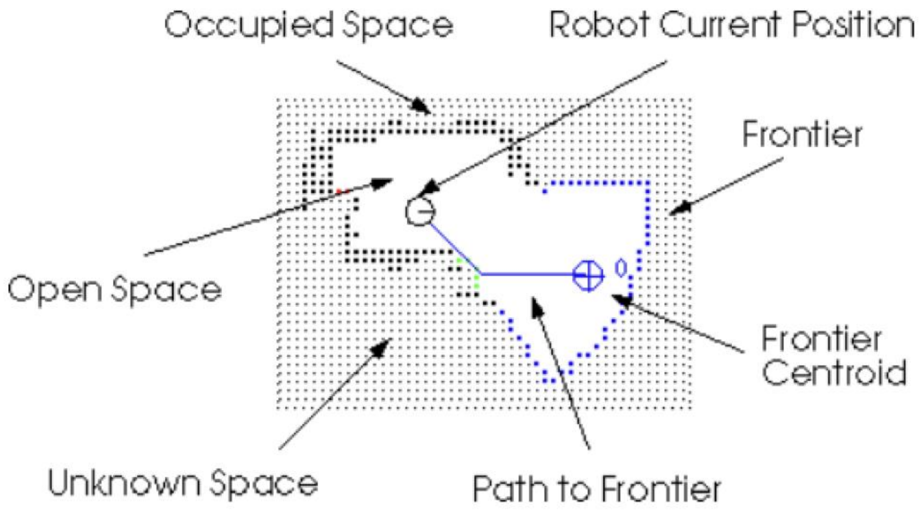
LaValle (2006)

The exploration problem

- How do we quantify exploration?

- How do we maximize the exploration?

- Terminology
  – Frontier
    - The border between our open space and unknown space
  – Frontier centroid
    - The center of the frontier border
  – Information gain
    - A metric for how much the robot learns from a given goal pose



The exploration problem

Problem definition

The exploration path planning problem consists in exploring a bounded 2D space $V \subset R^2$. This is to determine which parts of the initially unmapped space $V_{unm} = V$ are free $V_{free} \subset V$ or occupied $V_{occ} \subset V$. The operation is subject to vehicle kinematic and dynamic constraints, localization uncertainty and limitations of the employed sensor system with which the space is explored.

- As for most sensors the perception stops at surfaces, hollow spaces or narrow pockets can sometimes not be explored with a given setup. This residual space is denoted as V_{res}. The problem is considered to be fully solved when $V_{free} \cup V_{occ} = V / V_{res}$.

- Due to the nature of the problem, a suitable path has to be computed online and in real-time, as free space to navigate is not known prior to its exploration.

Exploration approaches for today

- Depth first search & Breadth first search
  - Finding the frontier
  - Exploring the frontier
- Next-best view exploration
  - Quantifying the exploration approach

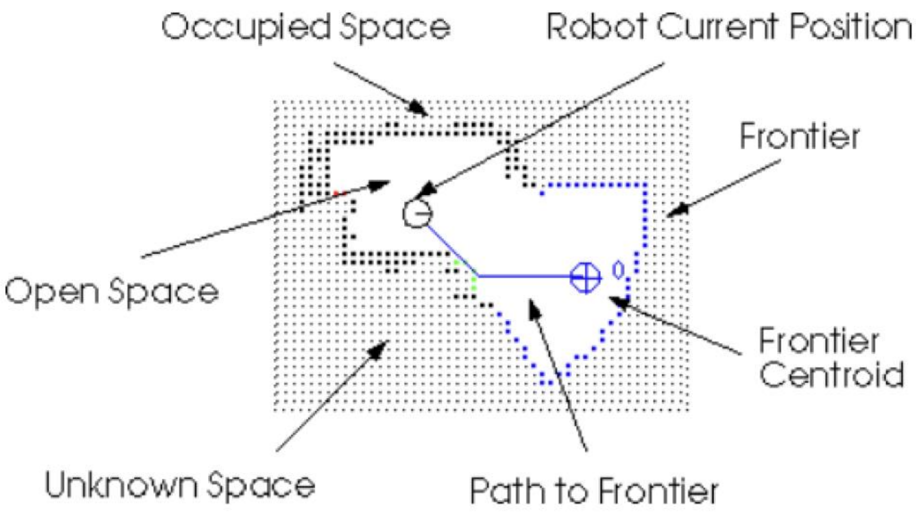
Frontier exploration

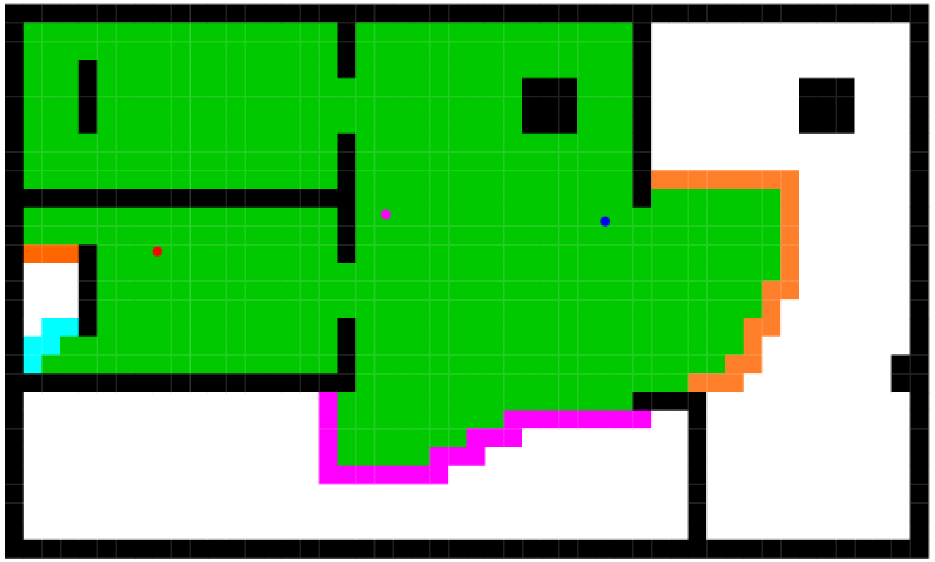
- The scenario: we have a partial map
- If we can identify the border of where we have been before, we can move towards the unknown

Frontier exploration

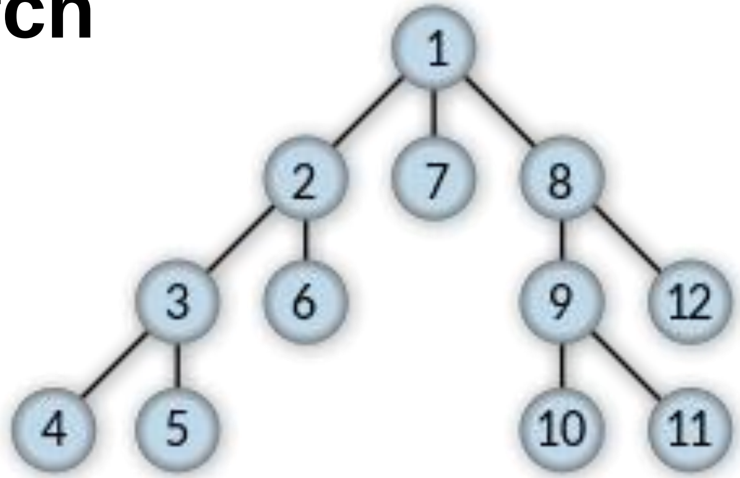
- The scenario: we have a partial map

- If we can identify the border of where we have been before, we can move towards the unknown

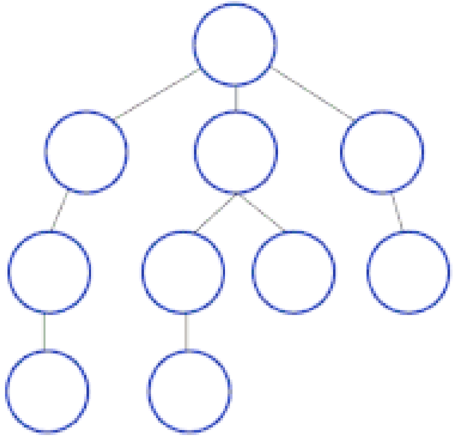


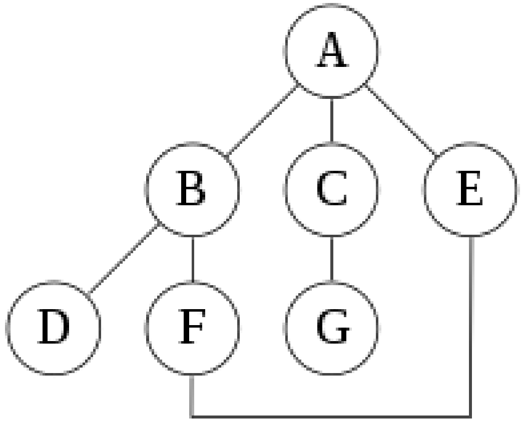


Finding the frontier - Depth first search



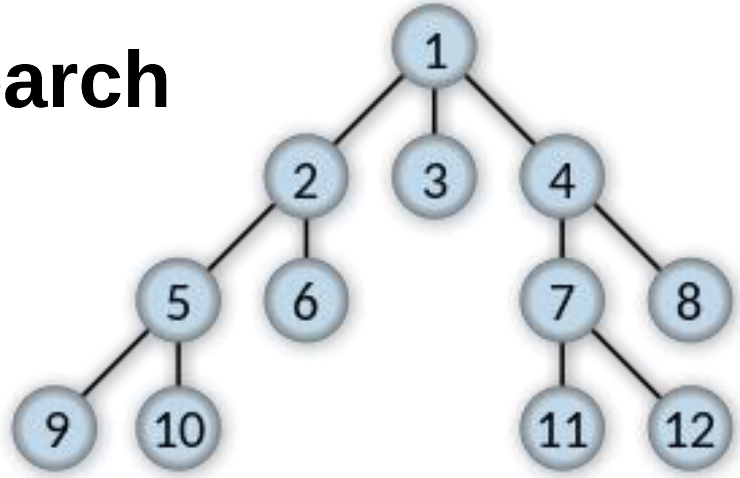
- Create a graph of all the cells in your map
- Traverse the branches prioritizing depth
  - NB, beware of loops (non-termination) – keep track of already visited nodes!
  - Sometimes we set a max depth to avoid graphs with loops or infinite depths
- This approach can set high memory requirements due to the tracking of the visited states

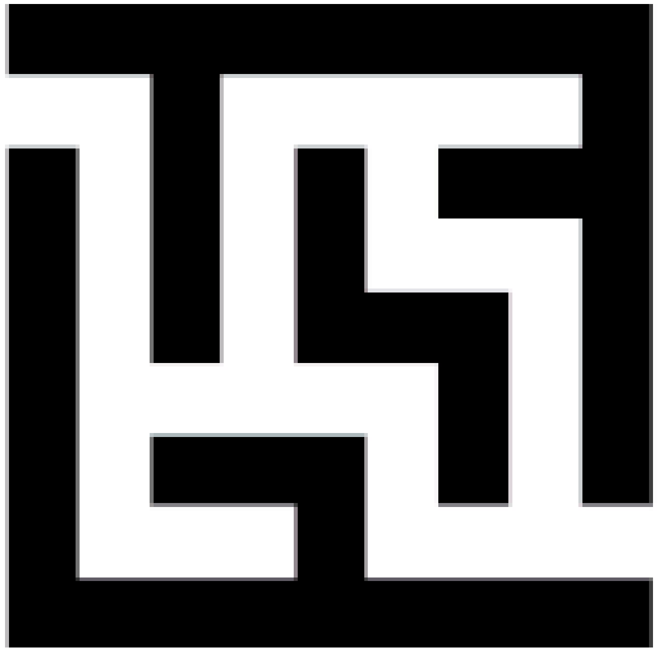


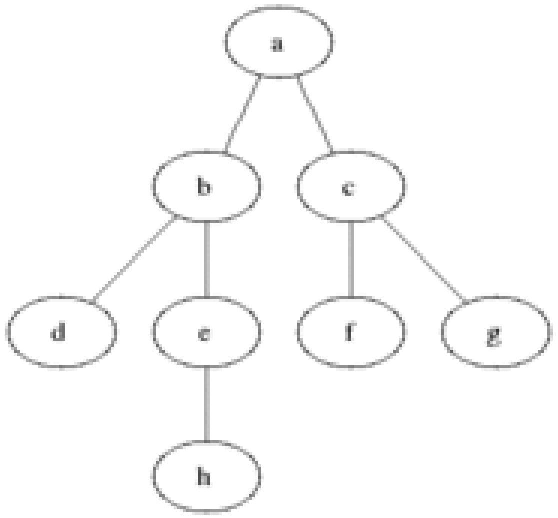


Finding the frontier - Breadth first search

- Explore horizontally instead of vertically in the graph
- Needs to keep track of child nodes that have been queued but not yet visited



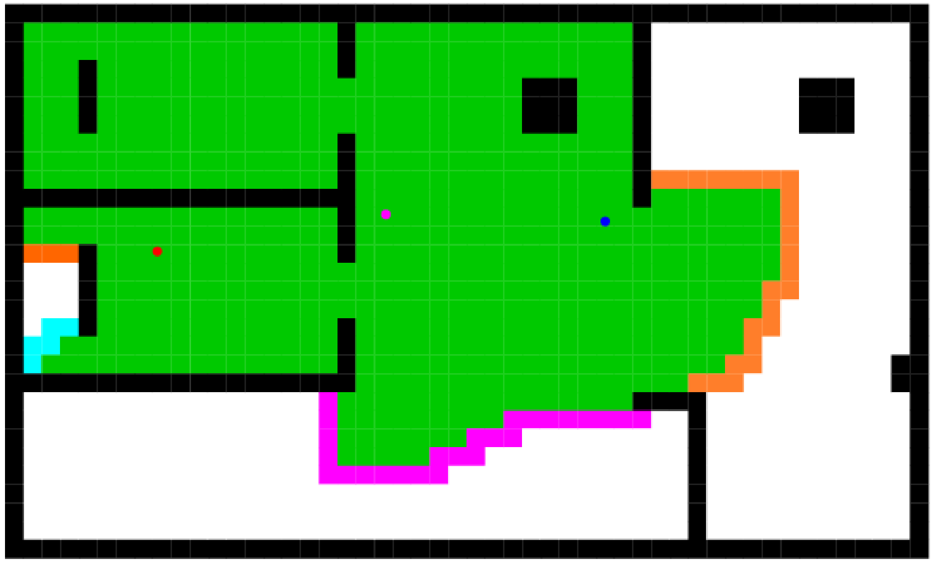




Depth- and Breadth-first search

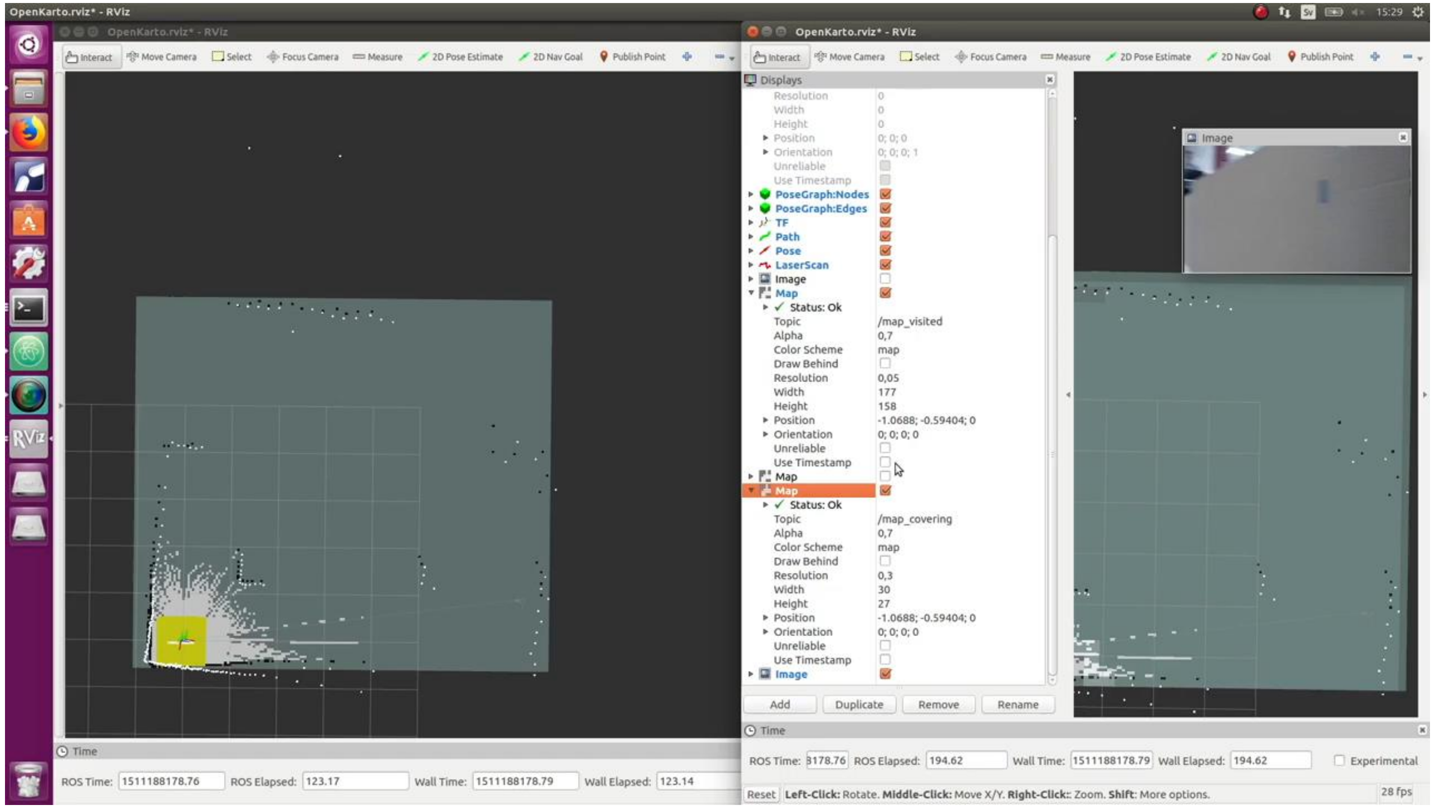
- Can be used to find the border between the observed and unknown space
  - i.e. every time we visit an open-space node that is connected to a unknown node, mark it as a frontier

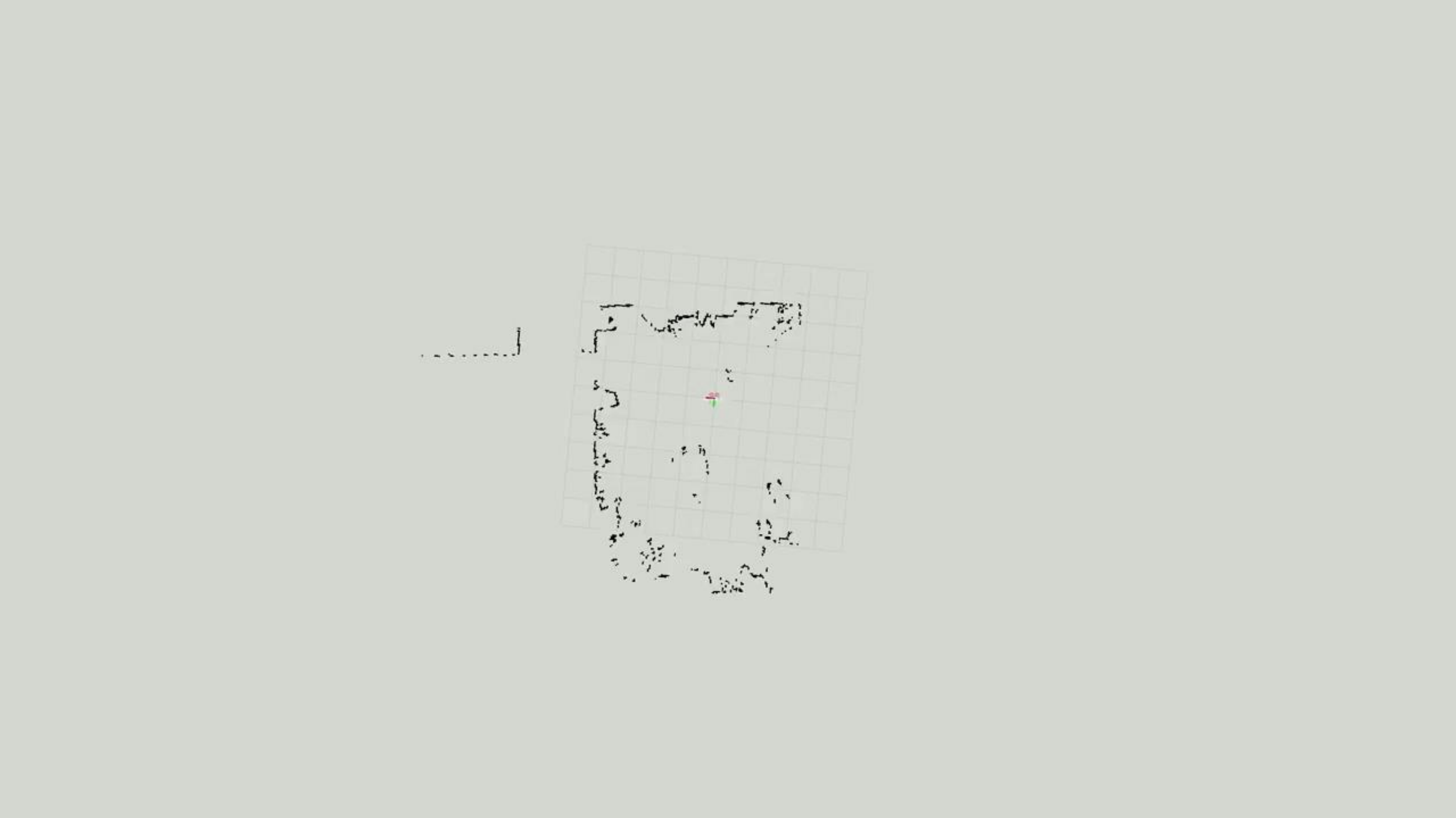
  - In a second run of either depth or breadth-first search, only add connected frontier nodes to the graph – i.e. separate frontiers



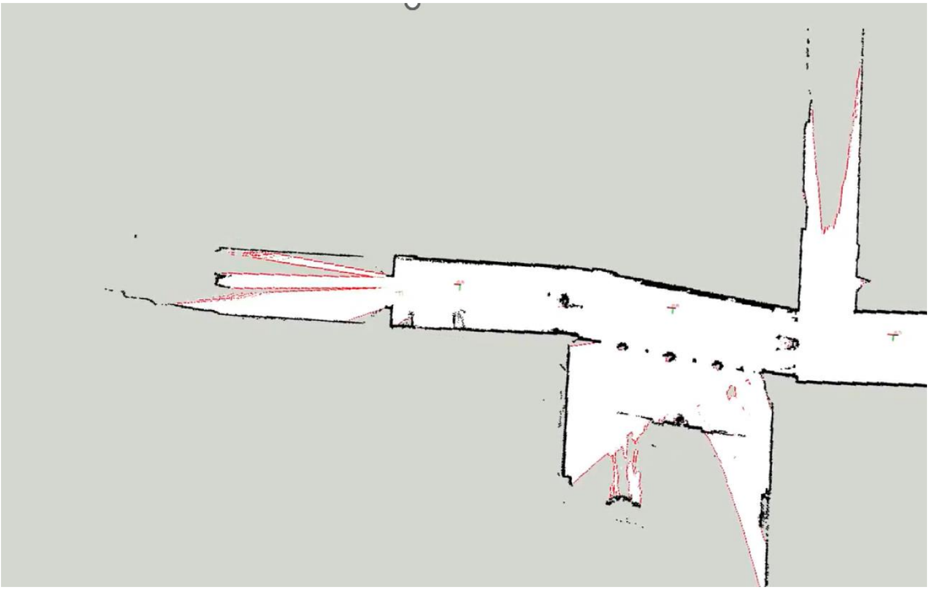
Depth- and Breadth-first search

- Can be used to find the border between the observed and unknown space
  - i.e. every time we visit an open-space node that is connected to a unknown node, mark it as a frontier





Exploring the frontier

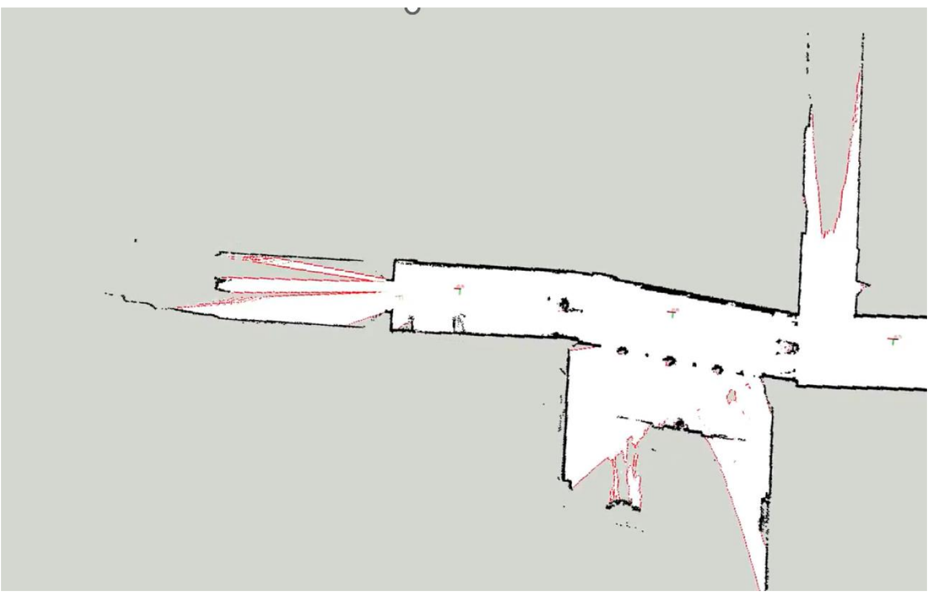


- To keep track of multiple frontiers, we perform a second search only on the identified frontier points

- To explore, navigate the robot towards the closest frontier

- Update the map after reaching the frontier, to get new frontiers

Remarks on exploring the frontier



- A very minimalistic approach
  - See a frontier and move towards it
- Doesn’t quantify our goal of exploration
  - Why is this frontier better than all other frontiers?
- Requires us to perform a depth- or breadth-first search after each navigation
  - The map is updated which produces new frontiers than needs to be found
- Information maximization

Next-best view exploration

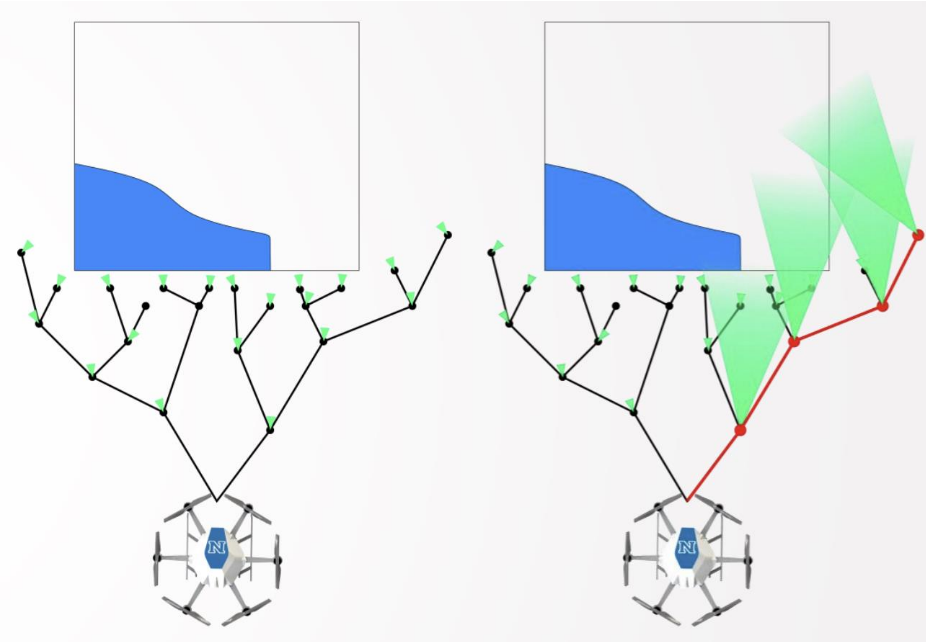
- An algorithm for fast exploration of unknown environments
- Defines a sequences of states/nodes from a graph (e.g. build using PRM or RRT)
- Select the sequence of states/nodes with the best sequence of viewpoints
- Execute only the first step of the best sequence (i.e. the path producing the best views of unknown areas)
- Repeat the hole process in a receding horizon fashion until the entire environment has been explored

Next-best view functional principle

- Tree-based exploration: At every iteration, NBVP spans a random tree of finite depth. Each vertex of the tree is annotated regarding the collected Information Gain – a metric of how much new space is going to be explored.

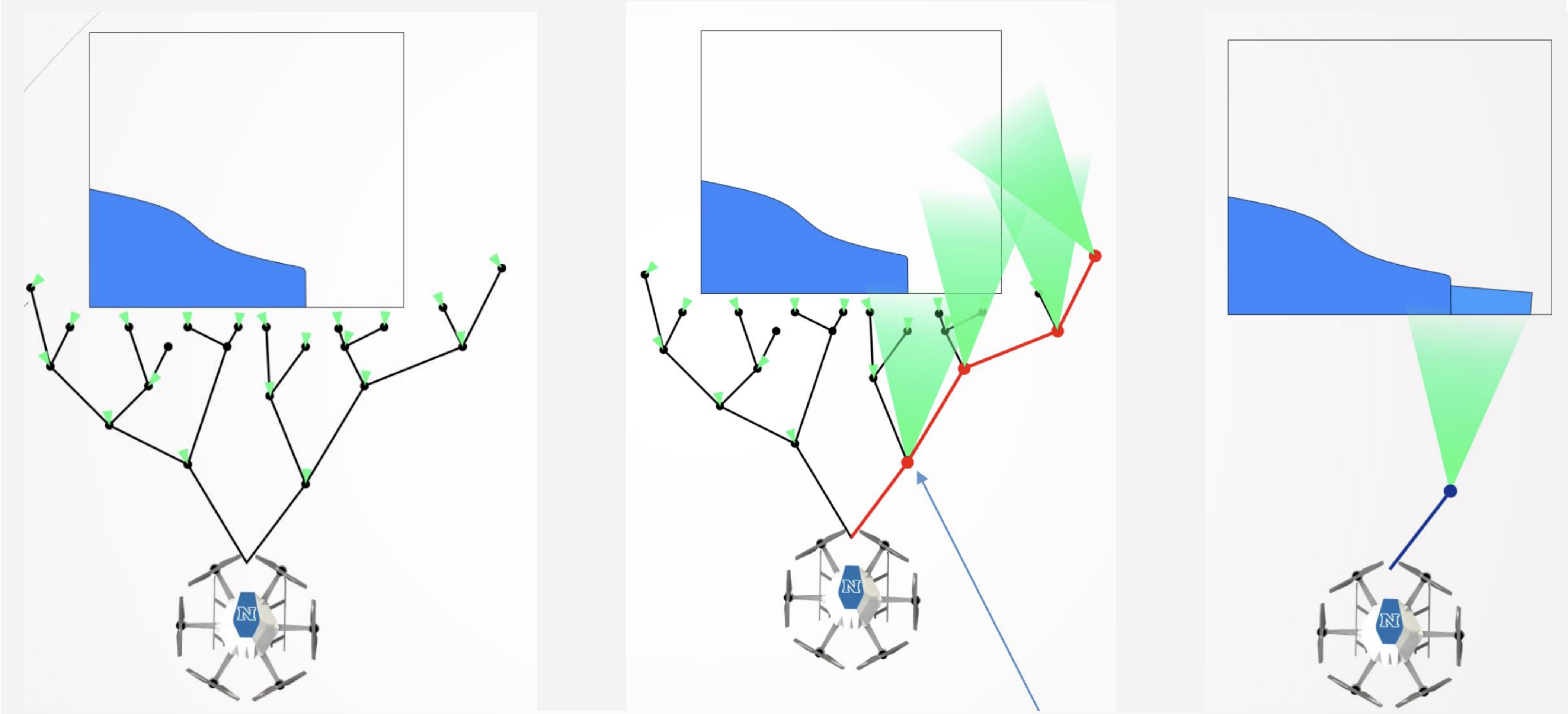
$$\mathrm{Gain}(n_k) = \mathrm{Gain}(n_{k-1}) + \mathrm{Visible}(\mathrm{M}, \xi_k)\mathrm{e}^{-\lambda c\left(\sigma_{k-1}^{k}\right)}$$

- Within the sampled tree, evaluation regarding the path that overall leads to the highest information gain is conducted. This corresponds to the best path for the given iteration. It is a sequence of next-best-views as sampled based on the vertices of the spanned random tree.



Kostas Alexis, Robotics Short Seminars, Feb 11 2016

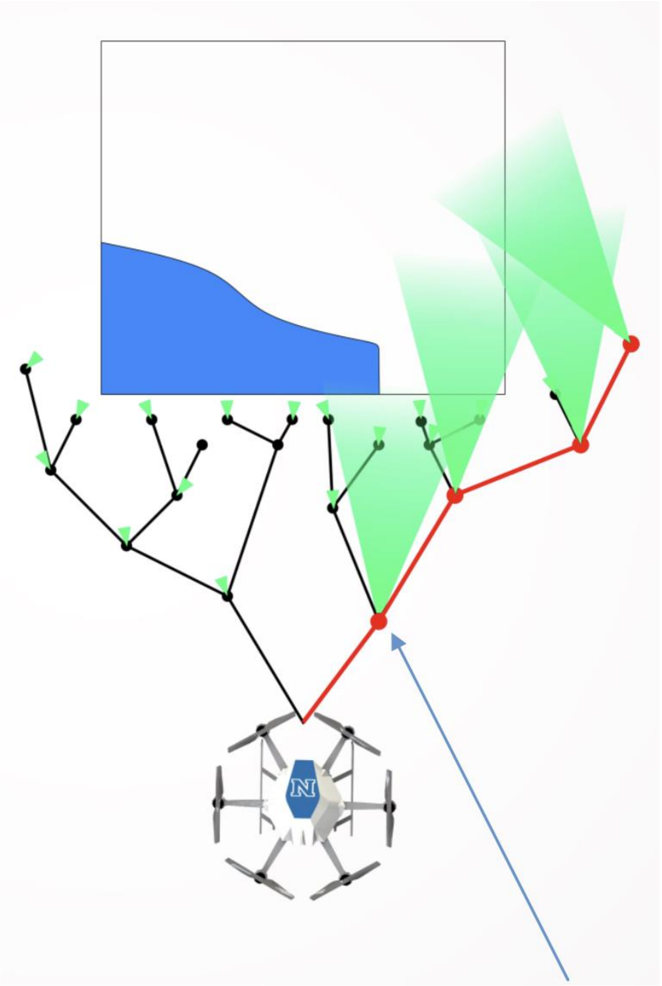
Next-best view functional principle



$$\mathbf{Gain}(n_k) = \mathbf{Gain}(n_{k-1}) + \mathbf{Visible}(\mathcal{M}, \xi_k)e^{-\lambda c(\sigma_{k-1}^k)}$$

Next-best view functional principle

- $\mathbf{Gain}(\boldsymbol{n_k}) = \mathbf{Gain}(\boldsymbol{n_{k-1}}) + \mathbf{Visible}(\mathbf{M}, \boldsymbol{\xi_k})\mathbf{e}^{-\boldsymbol{\lambda c}\left(\boldsymbol{\sigma_{k-1}^{k}}\right)}$
  - $\mathrm{Gain}(n_{k-1})$ is the information gain for all previous nodes up to the current node
  - $\mathrm{Visible}(\mathrm{M}, \xi_k)$ is the number of cells in the map M than can be seen from the robot configuration ξ_k
  - $\mathrm{e}^{\lambda c\left(\sigma_{k-1}^{k}\right)}$ is a discounting factor to limit the horizon of the planner (i.e. receding horizon). This ensure immediate nodes have a bigger influence on the selected path than nodes deep into the node tree
    - $c\left(\sigma_{k-1}^{k}\right)$ is the cost of moving from node n_{k-1} to n_k along path σ_{k-1}^{k} (this could be the Euclidian distance if the path is just a straight line)
    - λ is a tuning factor

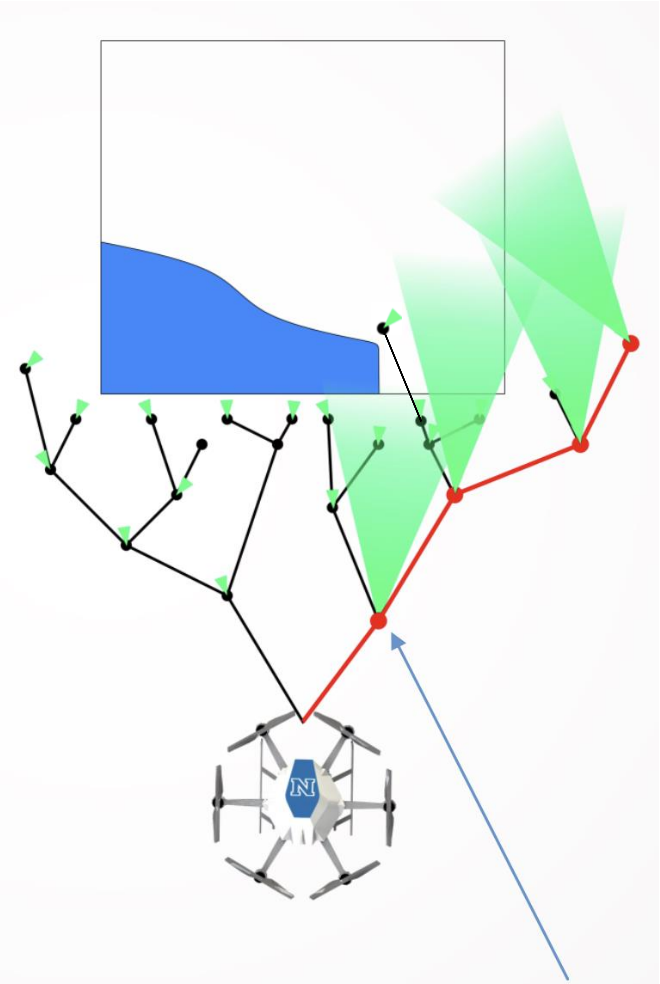


Next-best view functional principle

- Environment representation: Occupancy Map dividing space cells that can be marked either as free, occupied or unmapped.

- Only sample robot configurations within the free space our map to avoid planning trajectories into areas we do not know
  – These configurations will generally also have very high gains

- At each viewpoint/configuration of the environment ξ, the amount of space that is visible is computed as $\mathrm{Visible}(M, \xi)$
  – This means we need efficient ray-casting (e.g. using a camera matrix for depth cameras or projection for LiDARs)

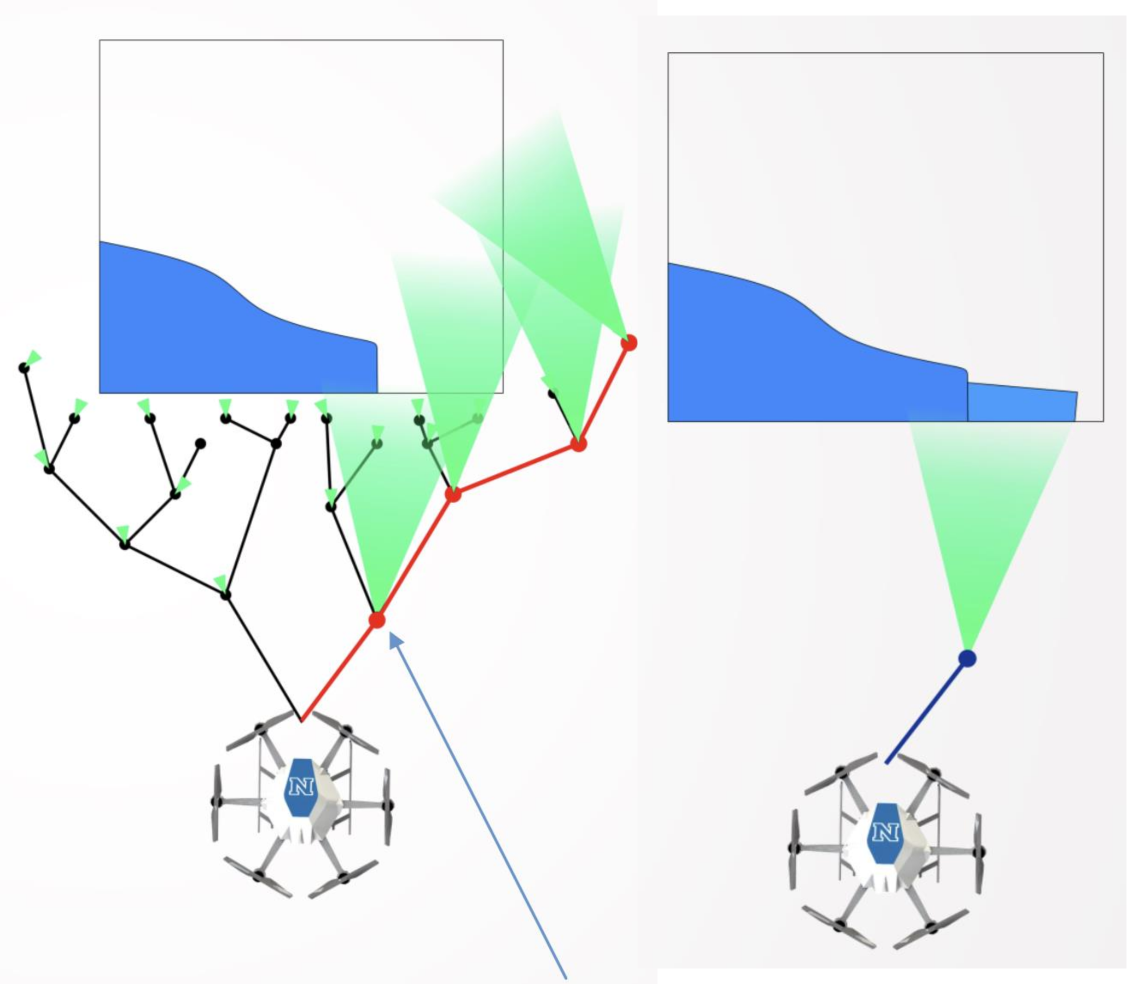


Next-best view functional principle

- Receding Horizon: For the extracted best path of viewpoints, only the first viewpoint is actually executed.

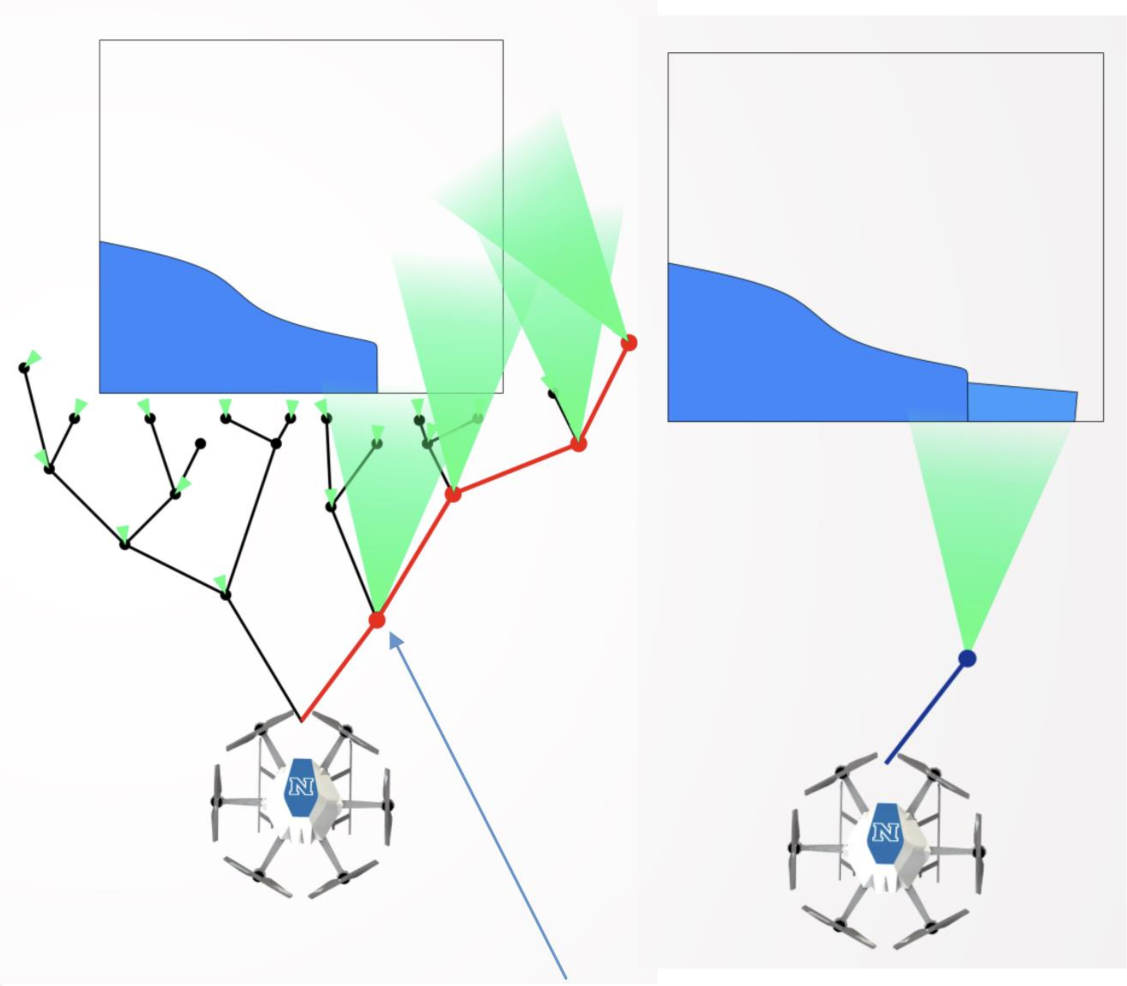
- The system moves to the first viewpoint of the path of best viewpoints.

- Subsequently, the whole process is repeated within the next iteration. This gives rise to a receding horizon operation.

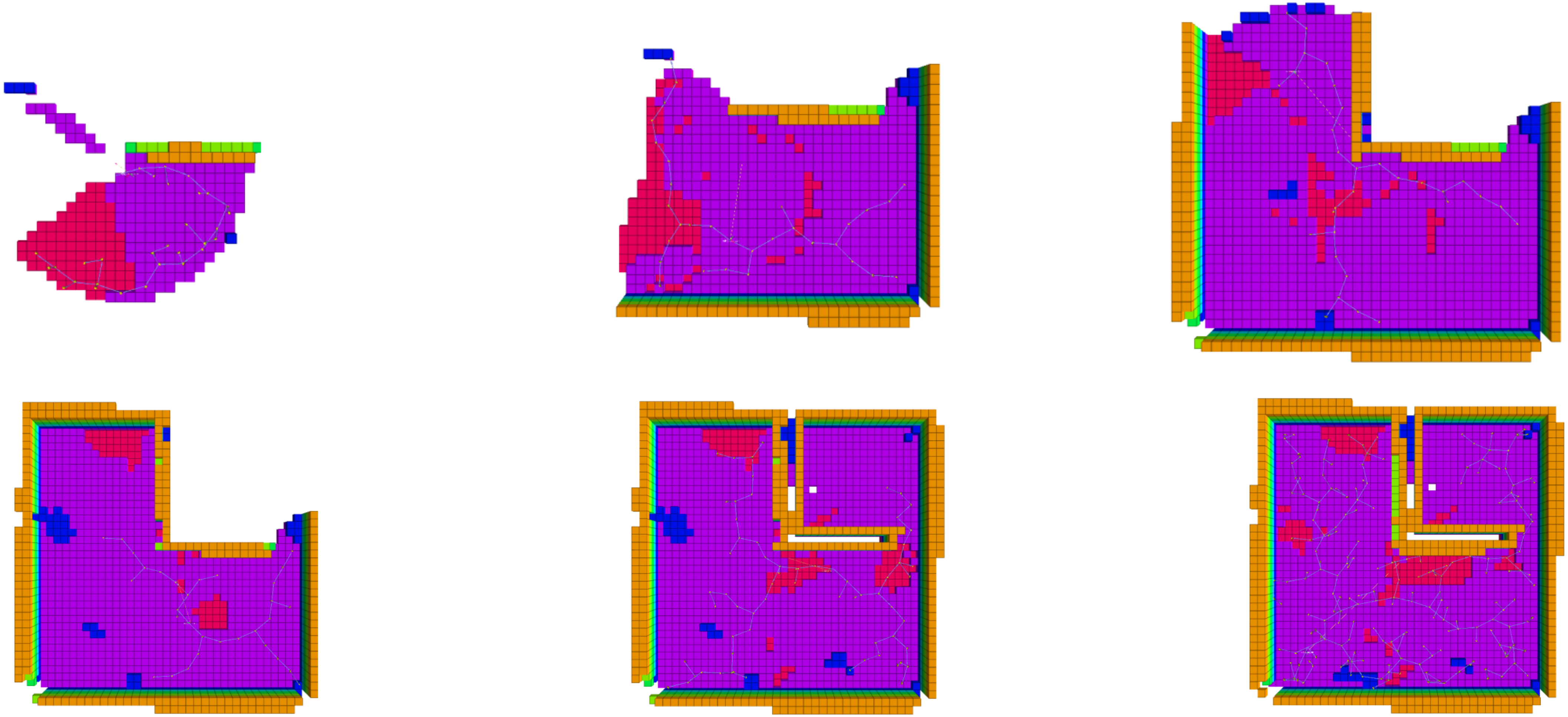


Next-best view algorithm

- ξ_0 ←current vehicle configuration
- Initialize $\boldsymbol{T}$ with ξ_0 and, unless first planner call, also previous best branch
- $g_{best} \leftarrow 0$ // Set best gain to zero
- $n_{best} \leftarrow n_0(\xi_0)$ // Set best node to root
- N_T ←Number of nodes in $\boldsymbol{T}$
- **while** $N_T < N_{max}$ or $g_{best} == 0$ **do**
  - Incrementally build T by adding $n_{new}(\xi_{new})$
  - $N_T \leftarrow N_T + 1$
  - **if** $Gain(n_{new}) > g_{best}$ **then**
    - $n_{best} \leftarrow n_{new}$
    - $g_{best} \leftarrow Gain(n_{new})$
  - **if** $N_T > N_{TOT}$ **then**
    - Terminate exploration
- $\sigma \leftarrow \boldsymbol{ExtractBestPathSegment}(n_{best})$
- Delete $\boldsymbol{T}$
- **return** σ



Examples of a Next-best view planner



Example of a Next-best view planner

Remarks on next-best view planning

- Can be inherently collision free as long as the path σ_{k-1}^{k} is collision free
  - This means you can use any of the planners from last week (PRM, RRT) and if they produce collision free navigation, the next-best view algorithm will also be collision free

- Most of the computational costs are from the collision checking, however, the information gain computation increases with the resolution of the map
  - Remember, we have to compute the number of visible cells in each robot configuration

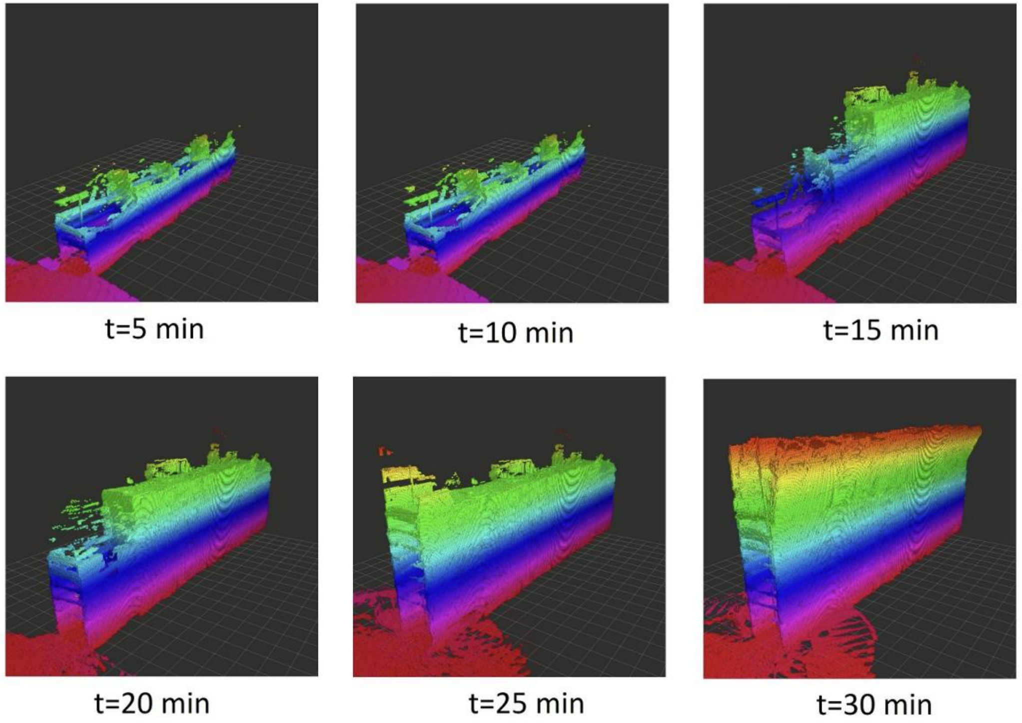
- The Receding Horizon Next-Best-View Exploration Planner relies on the real-time update of the 3D map of the environment.
  - We need to update the map, so we can update the gain computation as we explore the map

Remarks on next-best view planning

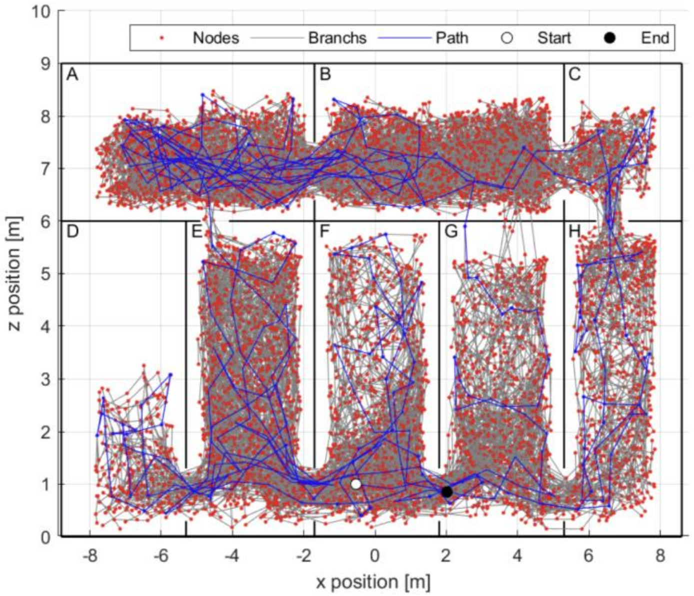
- The metric we use doesn't have to be purely unoccupied space – we can add multiple types of information to the information gain

$$Q(n_k) = Q(n_{k-1}) + \mu E(M, \xi) \cdot e^{-\lambda c(\sigma_{k-1}^{k})}$$

$$\mu = \sum_{i \in I} \frac{1}{d_i}$$



t=5 min
t=10 min
t=15 min
t=20 min
t=25 min
t=30 min



Nodes
Branchs
Path
Start
End
A
B
C
D
E
F
G
H
z position [m]
x position [m]
10
9
8
7
6
5
4
3
2
1
0
-8
-6
-4
-2
0
2
4
6
8

Exercises

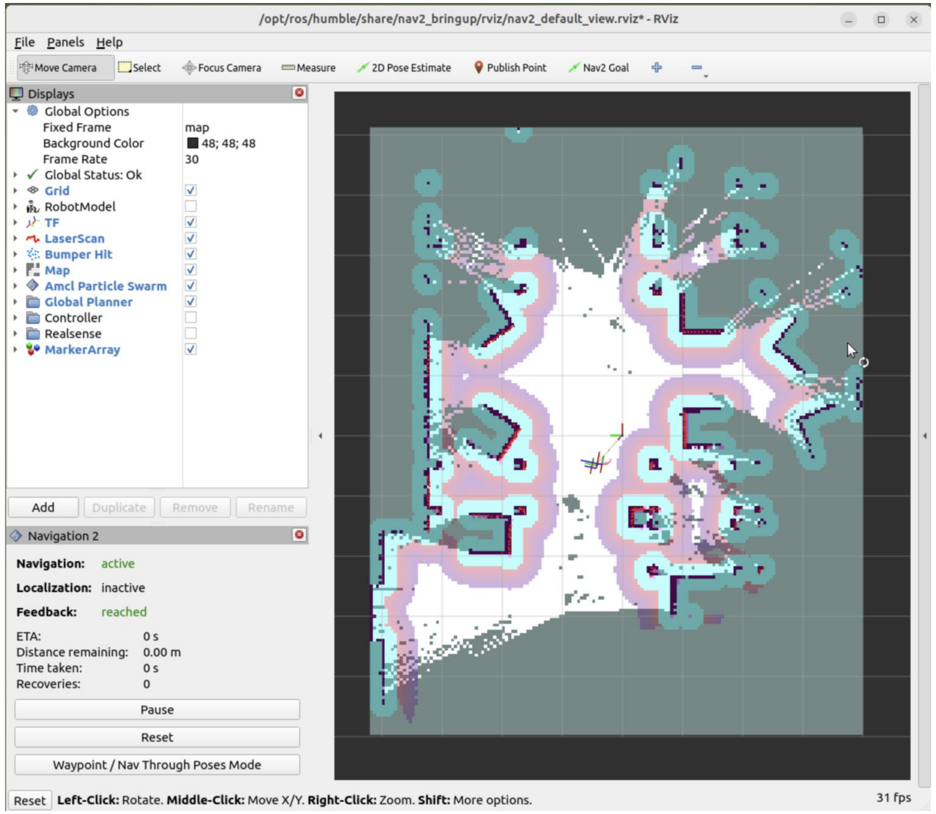
- Create a new map of the environment through rviz, *but only map it partially*

- Build a probabilistic roadmap of the partial map you just created
  - to reduce computational load, you can reject nodes sampled in unknown space
  - Keep it simple initially with just a few random nodes, and visualize them in rviz2

- For each node in the roadmap, compute the information gain according to

$$\mathrm{Gain}(n_k) = \mathrm{Gain}(n_{k-1}) + \mathrm{Visible}(\mathrm{M}, \xi_k)\mathrm{e}^{-\lambda c\left(\sigma_{k-1}^{k}\right)}$$

  - The visible cells has to be computed with ray-casting
  - Set $\lambda = 1$
  - You can assume a straight path between the current robot location and the next node and use a Euclidian cost function, i.e. $c\left(\sigma_{k-1}^{k}\right) = \left|\left|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right|\right|_2$



Exercises

- Execute the path with highest information gain
  - The action interface */navigate_to_pose* will drive the robot to the specified location
  - Example using the terminal:

```
ros2 action send_goal /navigate_to_pose
nav2_msgs/action/NavigateToPose '{pose: {header:
{frame_id: "map"}, pose: {position: {x: 0.0, y: 0.0, z:
0.0}, orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}}}}'
```