Weston Robot

RSE2107A – Lecture 6

ROS Navigation Part 2


SMURF 20
Weston Robot
SINGAPORE

Weston Robot

Agenda

01
Global Planners

02
Local Planners

Global Planners

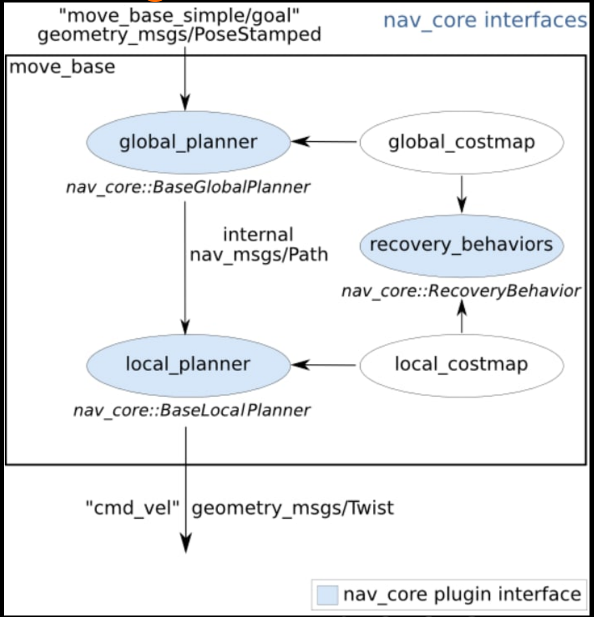
Global Planners

- The aim of a global planner is to find the shortest/most efficient and collision-free path to a given point from a start point.

- In the context of robotic autonomous navigation, this path is the path to a navigation goal that costs the least according to the global costmap.

Global Planners

- In the ROS navigation stack, all global planners are “plugins” for the move_base node, that share the same programming interface as the nav_core::BaseGlobalPlanner.
- Currently there are 3 such planner plugins:
  - global_planner < (Used by limo)
  - navfn
  - carrot_planner



Weston Robot

How does global_planner work?

- global_planner mainly use 2 algorithms commonly used to find paths
  1. Dijkstra’s
  2. A* (A-star)
- We will take a closer look into these 2 algorithms from 2 standpoints
  - How they work in (graph) theory?
  - How they are applied in the ROS navigation stack?

Weston Robot

In Graph theory

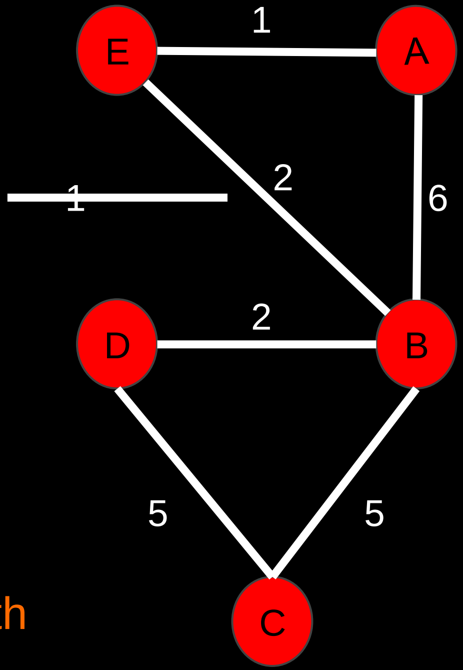
Graph Theory?!

- In mathematics, graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects.
- A graph in this context is made up of vertices (also called nodes or points) which are connected by edges (also called links or lines)

Weston Robot

Graphs in path finding

- In any path finding problem, the multitude of choices in traversing through a place/map can be challenging to visualize and analyse.
- To overcome this problem, the map can be simplified to a (weighted) graph where
  - Vertices/Nodes - Any place we can travel to
  - Links/Edges - Any possible paths between pairs of places
  - Numbers/Weights - Cost/Effort to travel along that path



Dijkstra's Algorithm

Dijkstra’s algorithm

- Published by computer scientist Edsger W.Dijkstra in 1959
- Used to find the shortest paths from a given start point to all other vertices/nodes in a given map or graph.
- This process results in a shortest path tree or table (spt) describing the shortest path to every other node from a specific starting node.

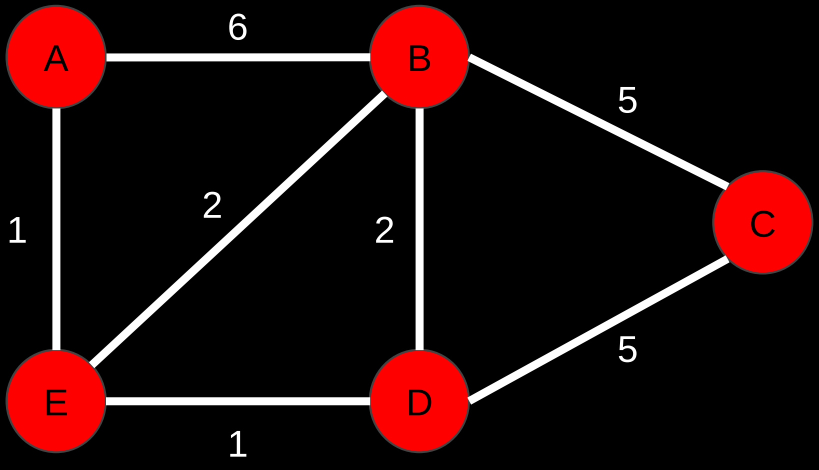
How does it work?

- Two lists are created, one to store the visited vertices and another to store the unvisited vertices.
- Set distance for the start vertex to 0.
- Set the distance of all the other vertices from start vertex to infinity.
- Visit the unvisited vertex with the smallest known distance from start.
- Let’s call this unvisited vertex, current vertex.

How does it work?

- For the current vertex, calculate the distance of each neighbouring vertex.
- If the calculated distance of a vertex is lesser than the known distance, update the shortest distance.
- Add the current vertex to list of visited vertices and repeat till all the vertices are visited.

Example

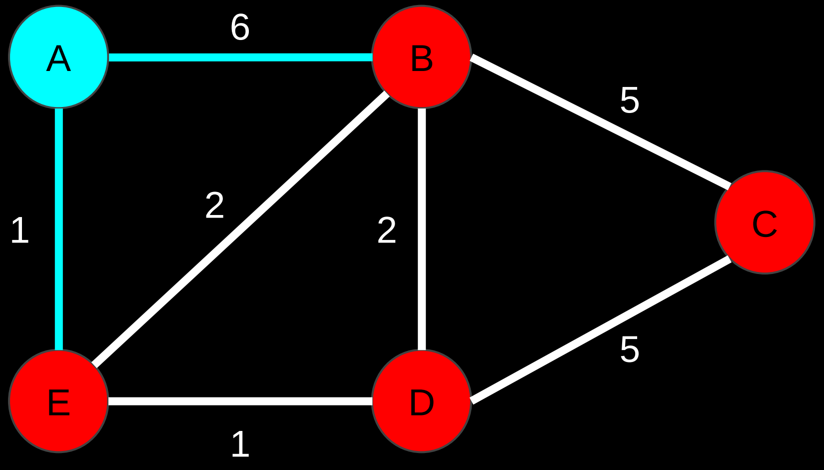


Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = []

Unvisited = [A, B, C, D, E]

Example

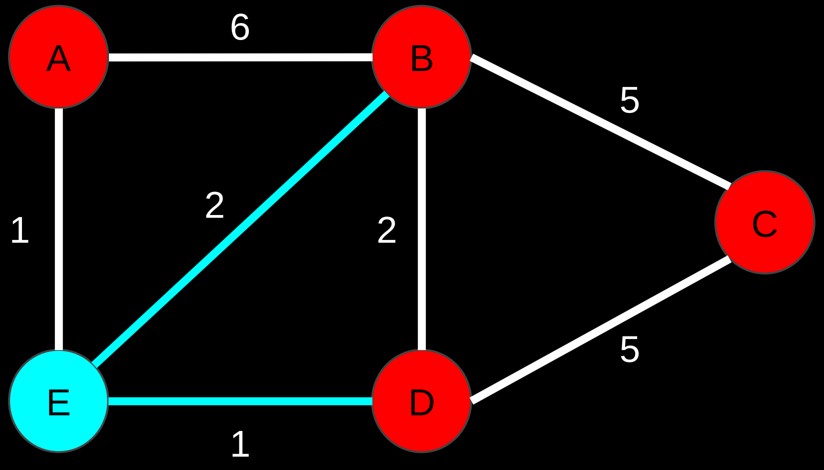


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	∞	
E	1	A

Visited = [A]

Unvisited = [B, C, D, E]

Example

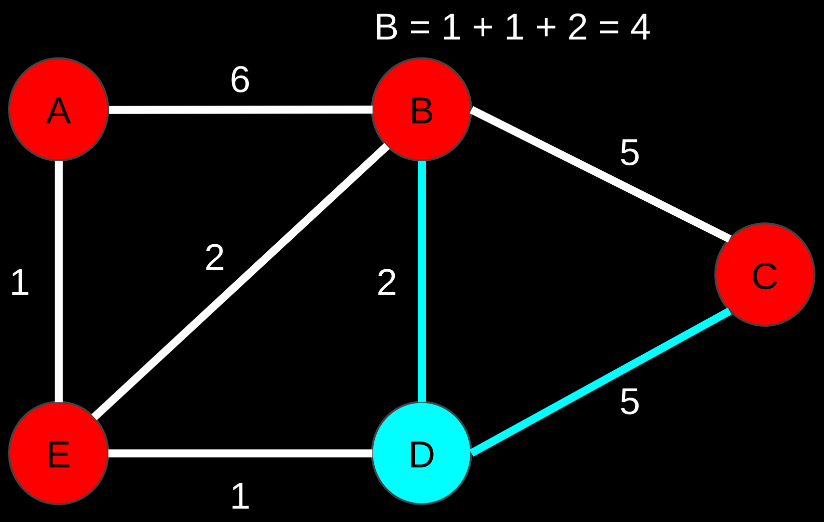


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	E
C	∞	
D	2	E
E	1	A

Visited = [A, E]

Unvisited = [B, C, D]

Example

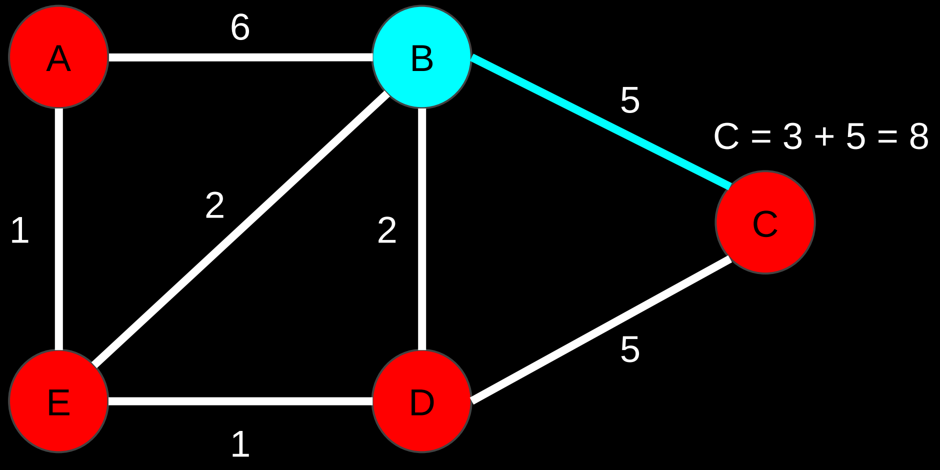


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	E
C	7	D
D	2	E
E	1	A

Visited = [A, E, D]

Unvisited = [B, C]

Example

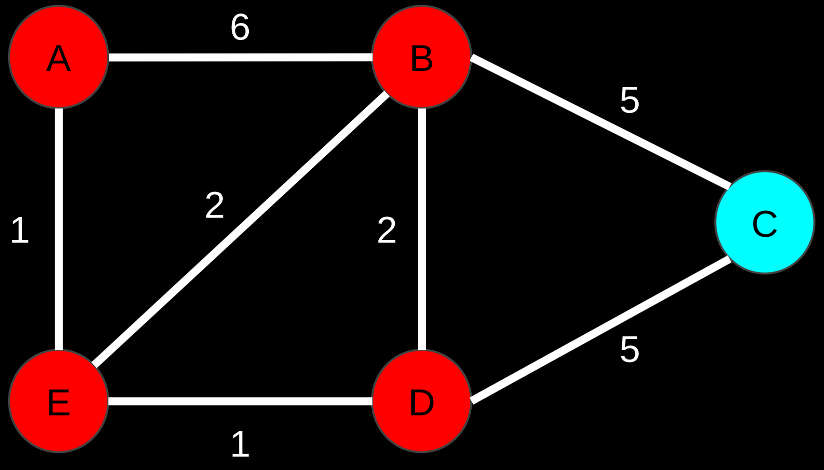


Visited = [A, E, D, B]

Unvisited = [C]

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	E
C	7	D
D	2	E
E	1	A

Example



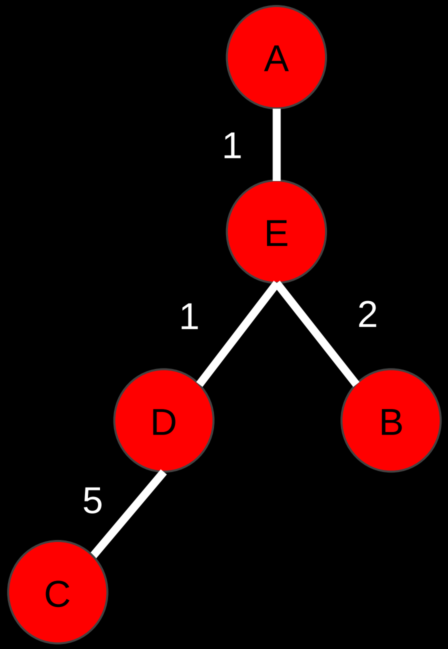
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	E
C	7	D
D	2	E
E	1	A

Visited = [A, E, D, B, C]

Unvisited = [ ]

Example

Tree



Table

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	E
C	7	D
D	2	E
E	1	A

Weston Robot

A* Algorithm

A* algorithm

- Published first in 1968 by Stanford Research Institute.
- Extension of Dijkstra’s algorithm. Achieves better performance by using heuristics to find the shortest path.
- Unlike Dijkstra’s algorithm, the A* algorithm only finds the shortest path from a specified source to a specified goal.
- Necessary trade-off for using a specific goal-directed heuristic.

A* algorithm

- The algorithm starts from the pre-defined start node and calculates the cost for all its surrounding nodes while searching for the shortest path.
- G cost [G(x)]: Cost to return to start node
- H cost [H(x)]: Cost to reach end node
    - H cost is estimated using heuristics
    - Eg, Manhattan, Euclidean
- F cost [F(x)]: Total cost = H(x) + G(x)

Heuristics

Weston Robot

- Denoted by h(x), where n represents the node
- The value of h(x) would ideally be equal to the cost of reaching the destination. However, this is not possible as we do not know the path to the destination.
- For a heuristic to be admissible, the estimated cost must be lower than or equal to the actual cost.
- For a value of h(x) that is greater than the actual cost, it will lead to a faster but less accurate search.

Weston Robot

How does it work?

- Given a map with a starting node, target node and obstacles according to the cost value F(x). At each step, the algorithm picks the node with the lowest F(x) and calculates the cost of surrounding nodes.
- When 2 nodes have the same cost value, the algorithm picks the node with the lower H(x) cost.
- Repeat till end node is reached.

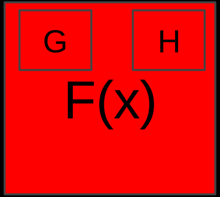
Weston Robot

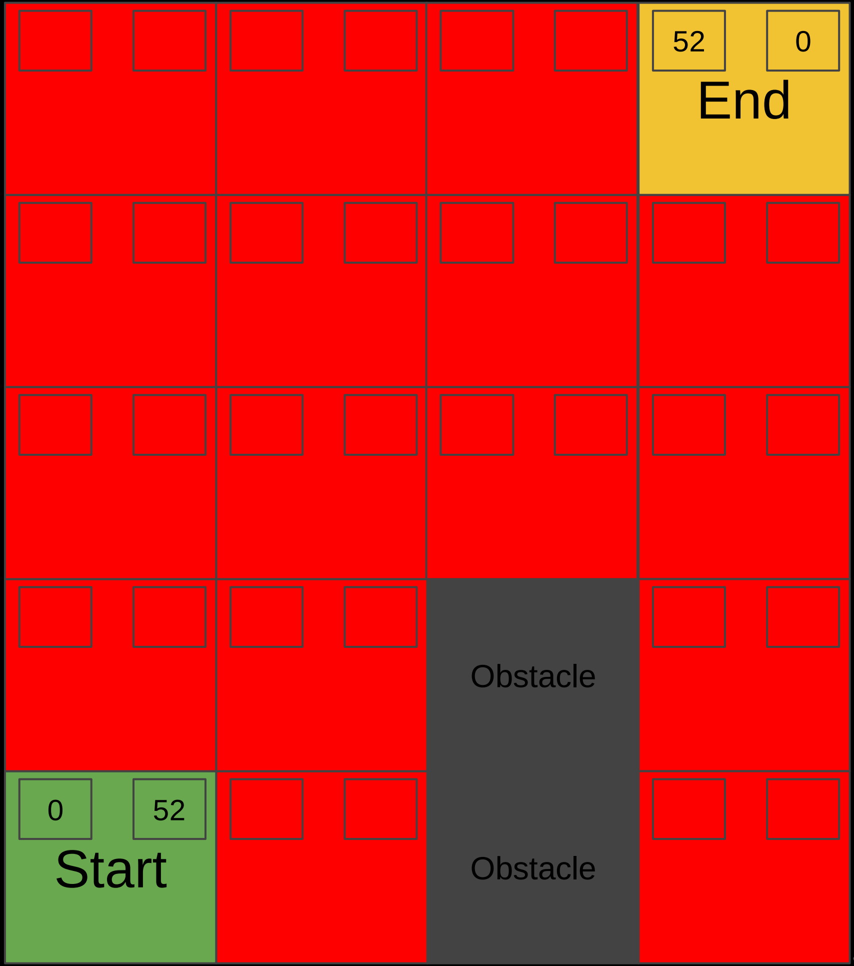
Example

- H(x): Euclidean
- H(x) = sqrt (

  $(current\_node.x - goal\_node.x)^2 +$

  $(current\_node.y - goal\_node.y)^2)$
- Each grid is 10 x 10
- Diagonal is ~14





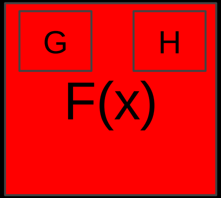
Weston Robot

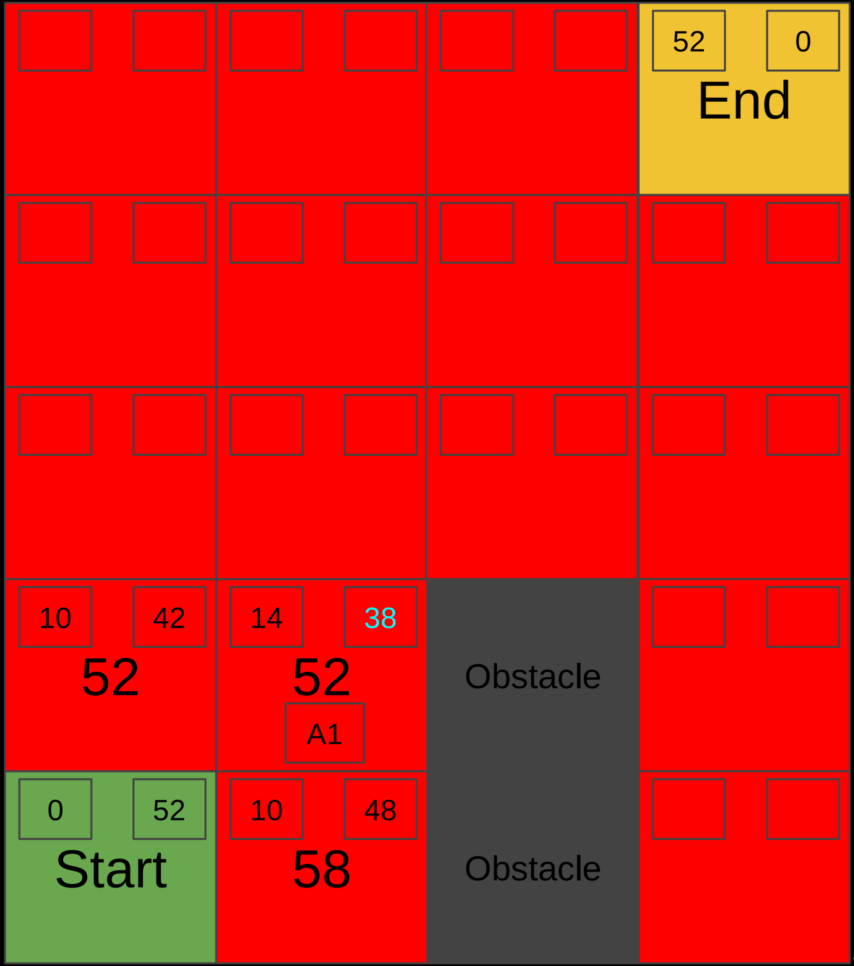
Example

- H(x): Euclidean
- H(x) = sqrt (

  (current_node.x - goal_node.x)^2 +

  (current_node.y - goal_node.y)^2)
- Each grid is 10 x 10
- Diagonal is ~14

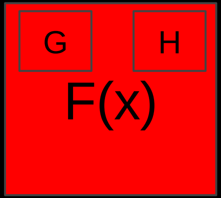


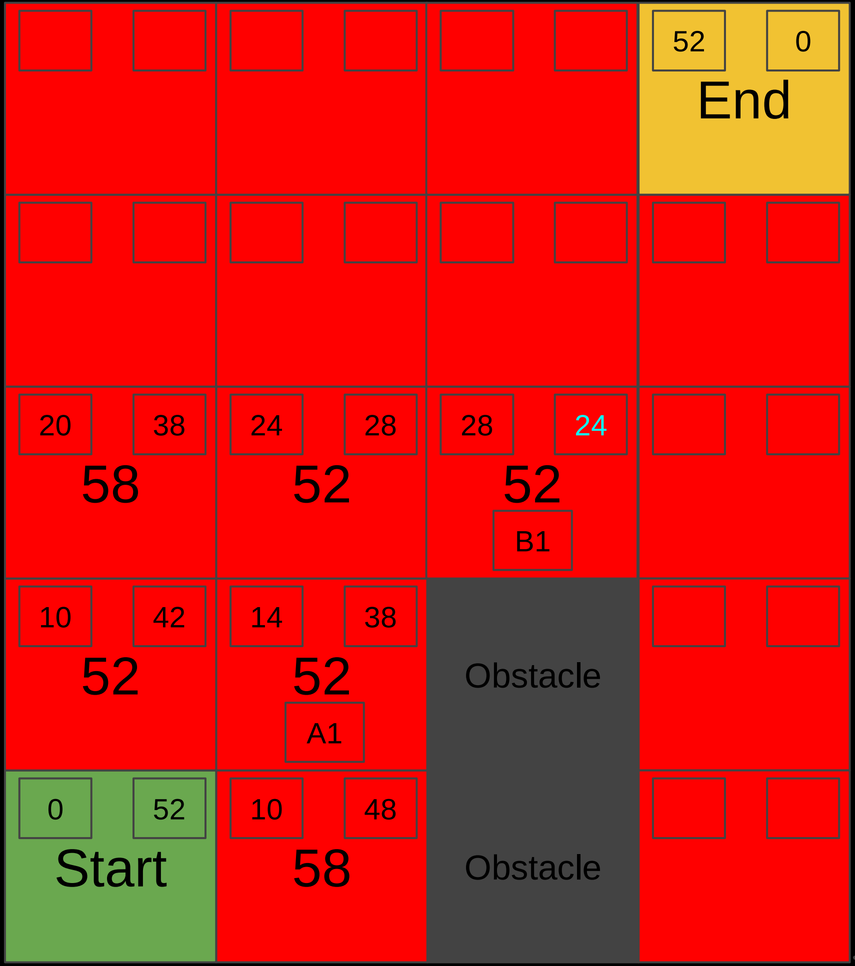


Weston Robot

Example

- H(x): Euclidean
- H(x) = sqrt (
  $(current\_node.x - goal\_node.x)^2 +$
  $(current\_node.y - goal\_node.y)^2$)
- Each grid is 10 x 10
- Diagonal is ~14



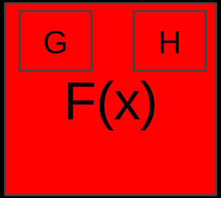


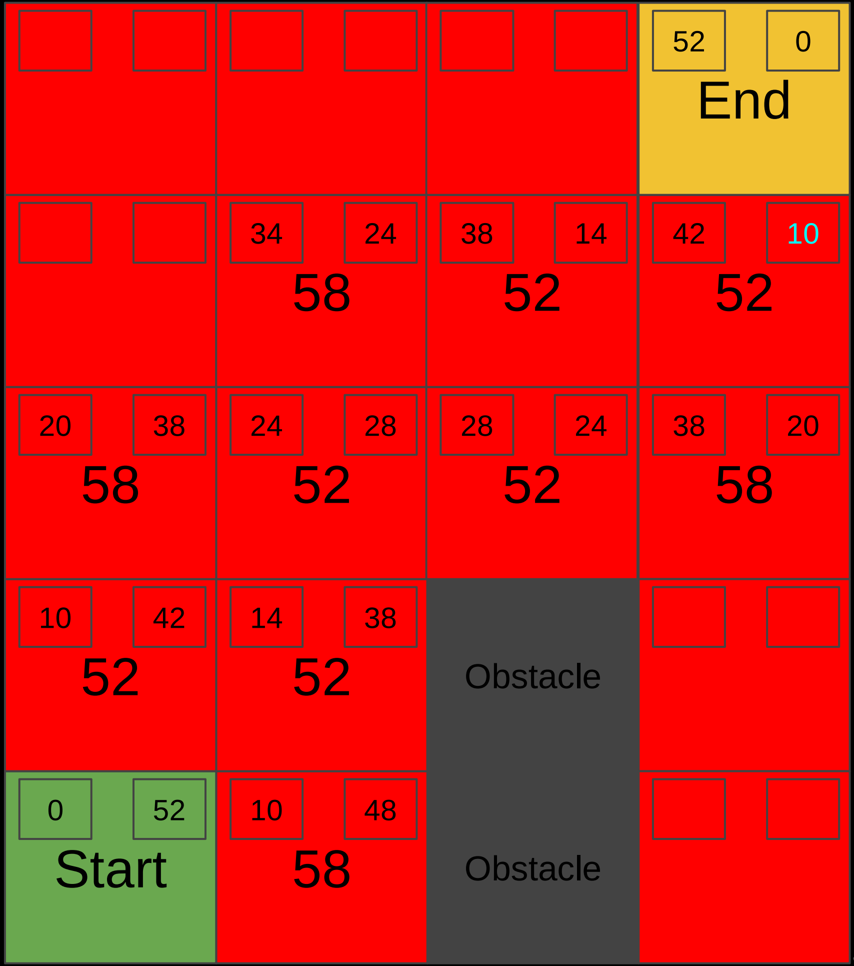
Example

- H(x): Euclidean
- H(x) = sqrt (

  $(current\_node.x - goal\_node.x)^2 +$

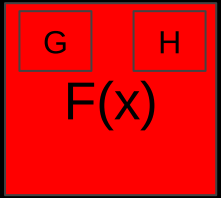
  $(current\_node.y - goal\_node.y)^2)$
- Each grid is 10 x 10
- Diagonal is ~14

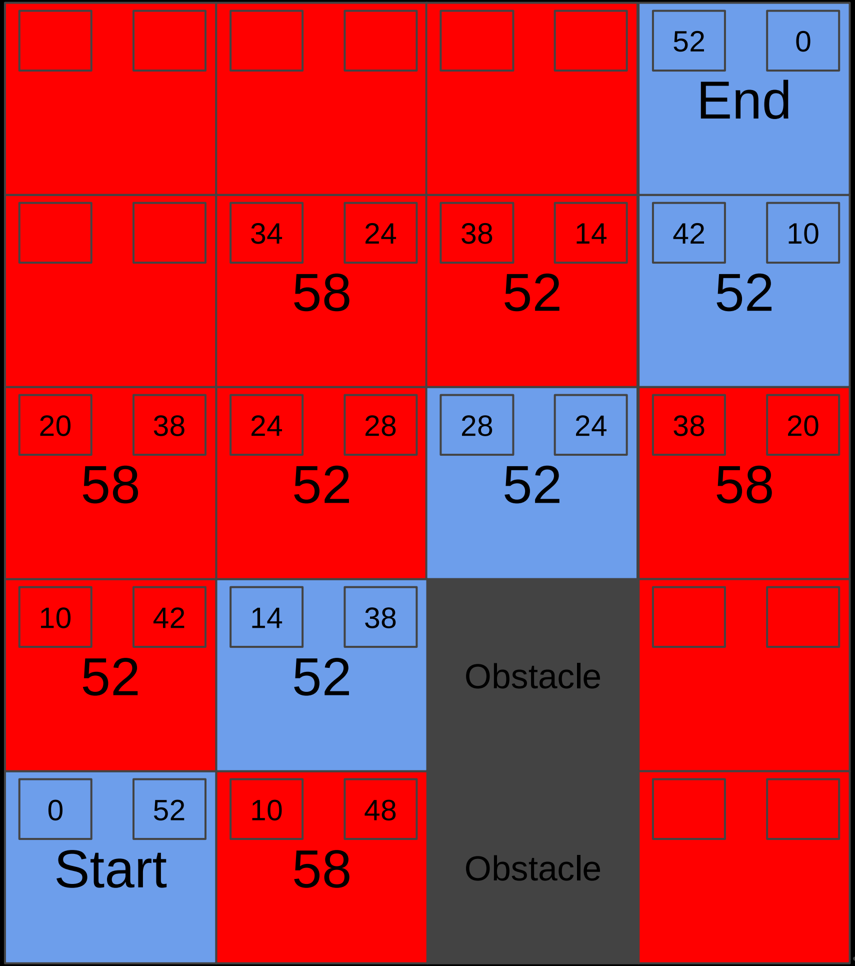




Example

- H(x): Euclidean
- H(x) = sqrt ((current_node.x - goal_node.x)^2 + (current_node.y - goal_node.y)^2)
- Each grid is 10 x 10
- Diagonal is ~14



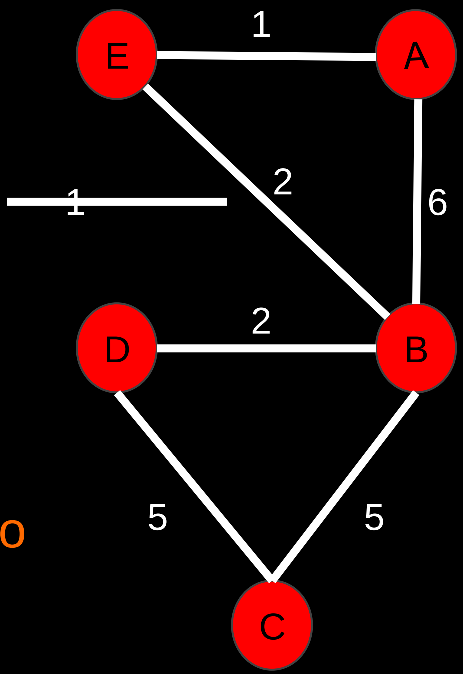


In ROS Navigation

Weston Robot

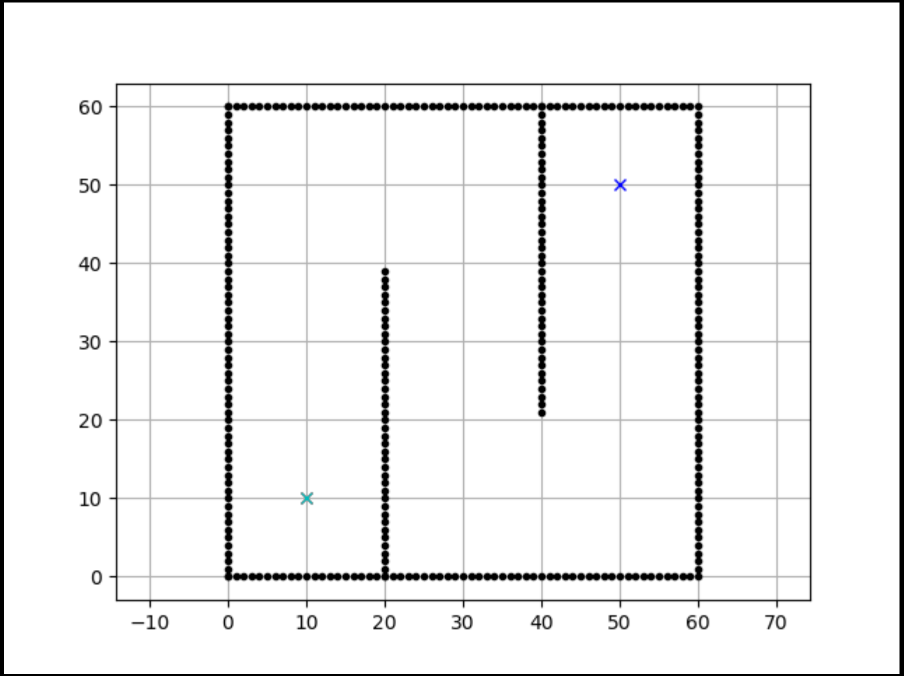
Graph <> Maps

- In the ROS Navigation stack, the graph represents
  - Nodes/Vertices - Points on the static map
  - Links/Edges - Possible paths between adjacent points.
  - Numbers/Weights - Cost of travel through that path (calculated from the global costmap)
- By doing so, we can find a path from the start point to the navigation goal

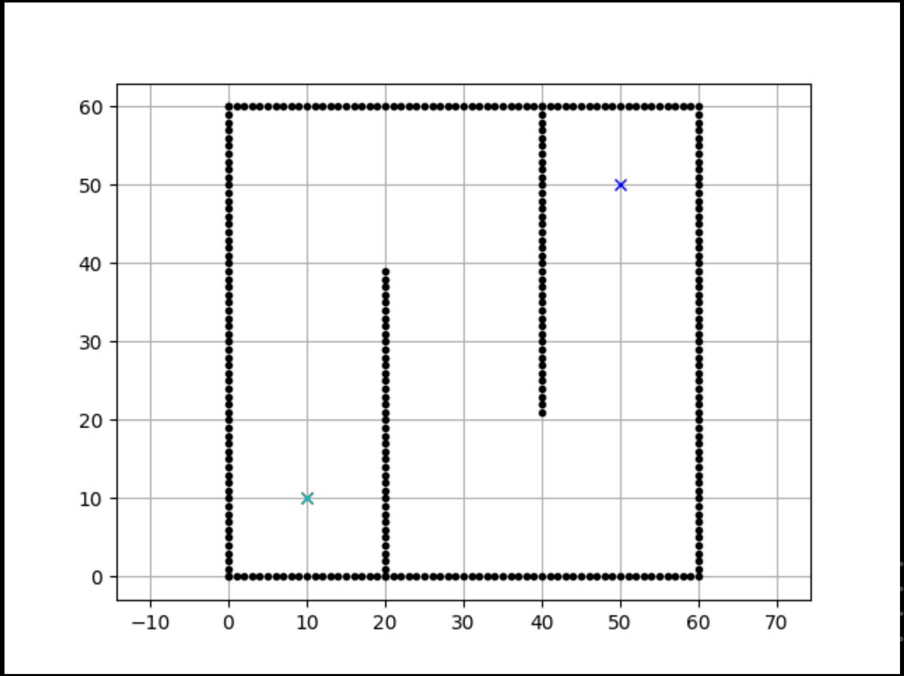


Visualization

Dijkstra's



A*



For those interested

- The problem we have been going through and trying to solve is what is also known as “Single Source Shortest Paths (SSSP)” Problem.

- A great resource to learn and visualise different algorithms used to solve said problem and other concepts of graph theory (outside of scope of this course) can be found here.

Local Planners

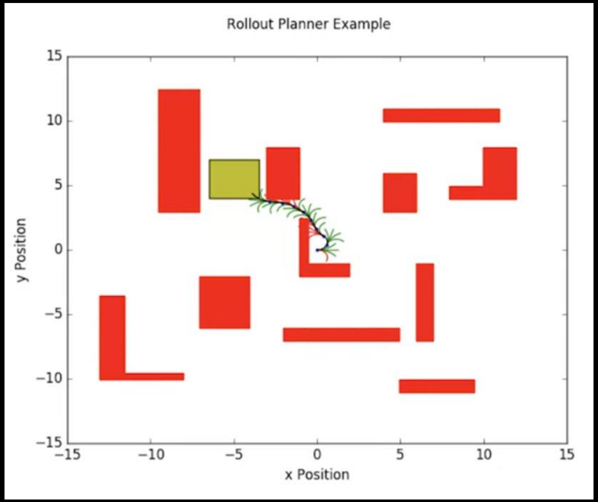
Local Planners

- The aim of a local planner is to transforms the global path to suitable waypoints, while taking into consideration of dynamic obstacles and vehicle constraints.

- It results in velocity commands (geometry_msg/Twist aka /cmd_vel in move_base) that are sent to the robot to be performed.

Trajectory Rollout

- Uses trajectory propagation to generate candidate set of trajectories
- Among collision-free trajectories, choose trajectory that makes most progress to goal



Trajectory Set Generation

Weston Robot

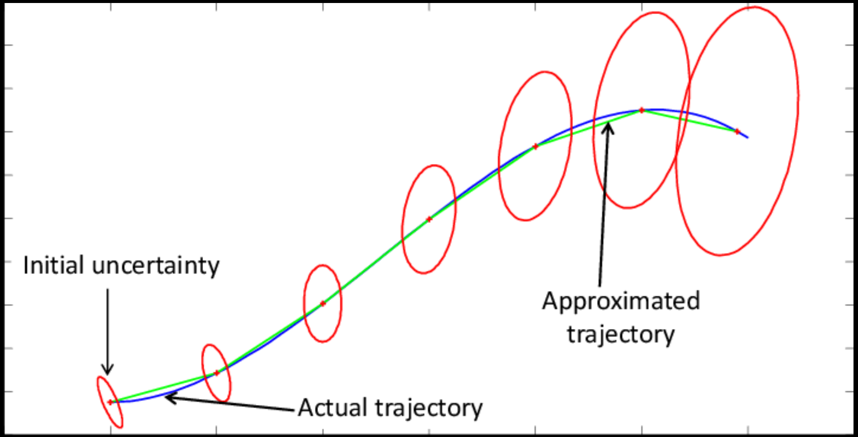
- Each trajectory corresponds to a fixed control input
  - uniformly sampled across a range of possible inputs

More sampled trajectories	Less sampled trajectories
More maneuverability	Improves computation time

- In TrajectoryPlannerROS, this can be changed using the parameters under “Forward Simulation”

Trajectory Propagation

- Generating “future states” along trajectories by propagating state forward using kinematic model of robot
- Take into account the following variables:
  - proximity to
    - obstacles
    - goal
    - global path
  - speed of robot

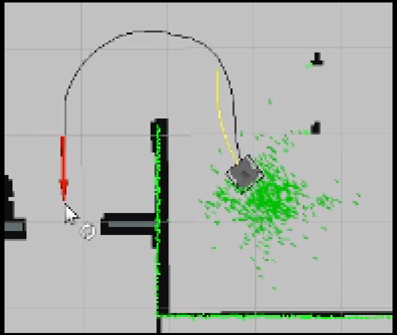


Weston Robot

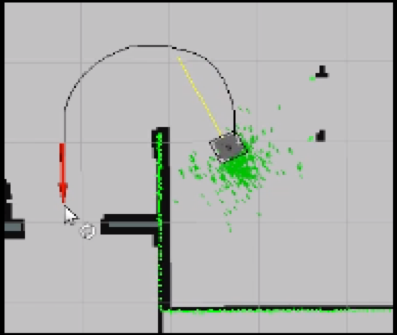
Selecting Trajectory

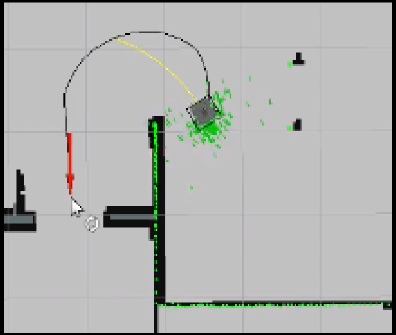
Recap from last week

$$
\begin{aligned}
cost &= \texttt{path\_distance\_bias} * (\text{distance}(m) \text{ to path from the endpoint of the trajectory}) \\
&+ \texttt{goal\_distance\_bias} * (\text{distance}(m) \text{ to local goal from the endpoint of the trajectory}) \\
&+ \texttt{occdist\_scale} * (\text{maximum obstacle cost along the trajectory in obstacle cost (0-254)})
\end{aligned}
$$

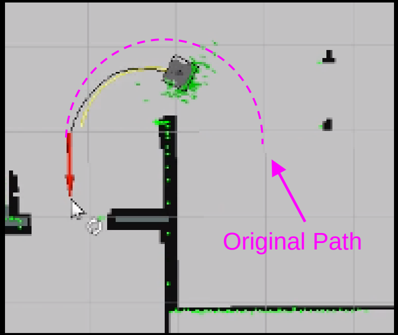


Trying to stay within path





Steering from path and attempting to reach goal



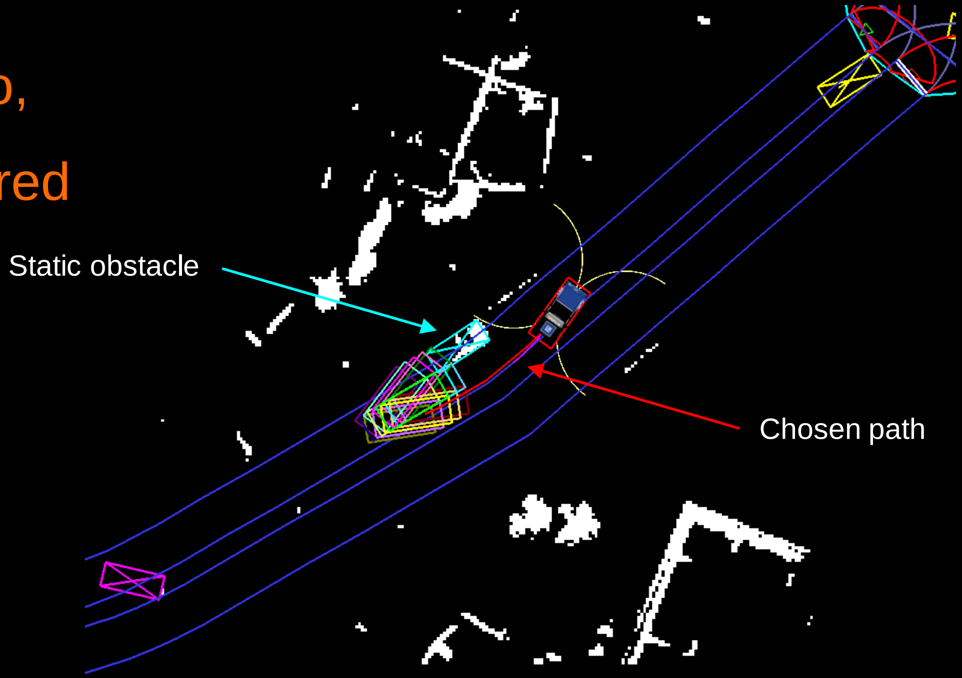
Changing path and trying to stay within new path

Selecting Trajectory

- The trajectory that is selected for execution usually
  - deviates the least from global path
    - can be tuned by modifying the cost function (like the one in the previous slide).
  - collision-free (static and dynamic obstacles)
    - checked by comparing to perception and static maps.

Example

- Set of goals being planned to, with resulting path shown in red



Example

- Trajectories generated by local planner to track this path