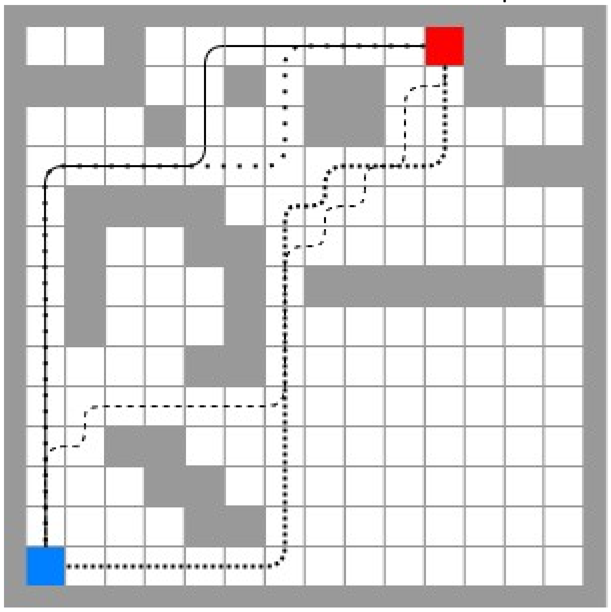
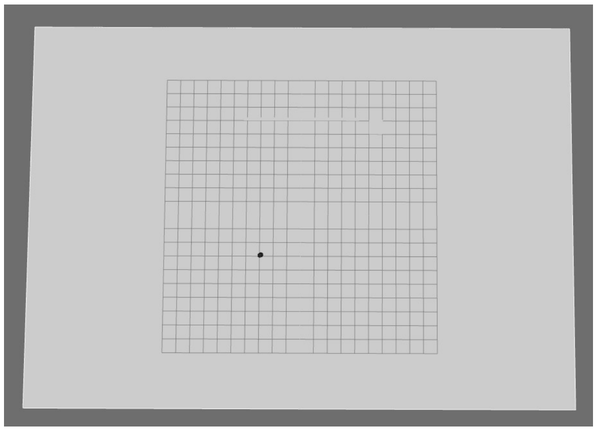
Dijkstra Algorithm

Friday, January 27, 2023 5:59 AM

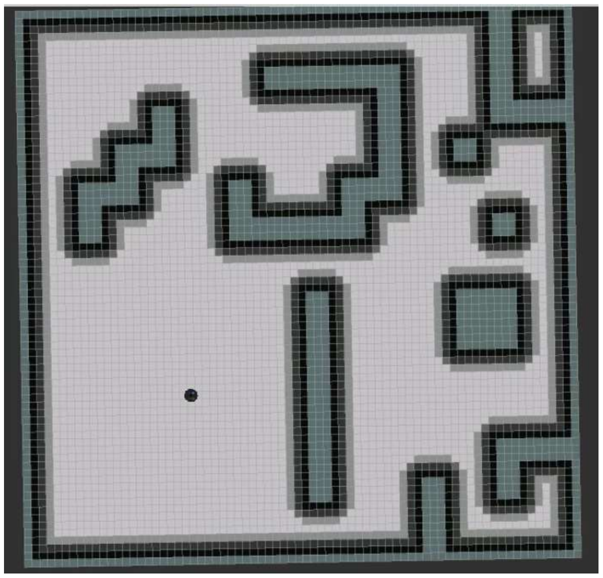
Let's suppose that the error is able to generate a occupancy grid map and is able to localize itself in that map.
What is the best route that the robot can plan?



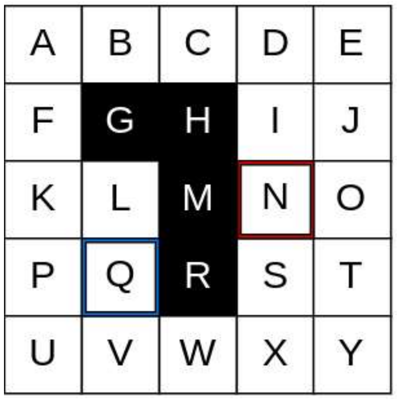
Robot Gazebo World:



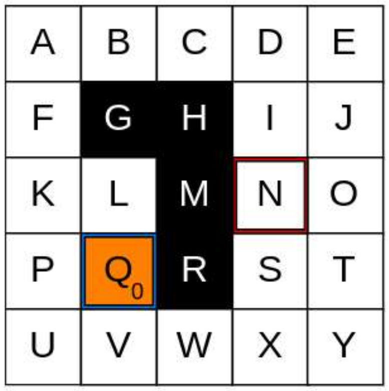
Robot RVIZ (Occupancy Grid Map):



Example:
The example is based on a 5x5 Occupancy Grid Map, we want to go from Q to N cell, black cells are occupied cells



Algorithm Start:
This algorithm always keeps track of the shortest distance from the start distance to each individual cell, we can
name this value as cost of a node, before starting the initial node gets a cost of zero. Then we add this value to a open list. Open list nodes will have the orange color.

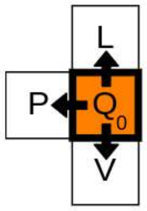


Iterative Process:
Step 1) Pick current node
We pick the node inside the open list with the lowest cost. This will be the current node. At first it will only contain

node Q. We will use a black outline to identify the current node.

Step 2) Neighbors of current node

Then we examine the neighbors of the current node. We will only consider the traversable and available cells.



Step 3) Update travel distance value, store parent node

Once we have identified the neighbors of the current node, we need to update their cost. Then we set the parent
node of each neighbor. The parent node is the node from which the update of the cost came from (which is always)
the current node. We will show each grid cell's parent node at the bottom of a grid cell, to the right of its cost. To conclude we put each neighbor inside open list.

Let's begin with neighbor L.

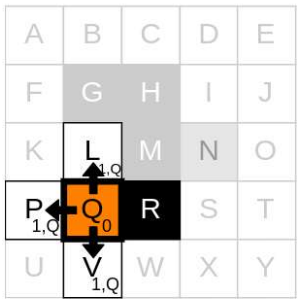
A) We add the cost of the actual node plus the step cost, which is the cost incurred when moving to a neighbor cell. It is assumed that its value is one.

B) We set its parent node, which is Q

C) We put this neighbor inside open list

Then we continue with the rest of neighbors of the current node.

D) We mark the current neighbor as visited, we will add to a list named closed list



Step 4) Repeat the previous steps

After finishing first cycle:

We need to select the current node, so we pick a node inside the open list with the smallest cost. It could be L P V nodes because they have the same cost, in this case we will pick L and mark it as bold black line to identify it as the current node.

We repeat the process by updating the neighbors of L. We ignore nodes inside the closed list which by now contains Q node, also we ignore the occupied cells. Therefore we can only process K node.

This way we update the cost and parent node of K:

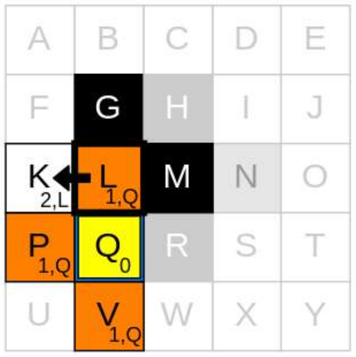
A) Add the cost of L plus the step cost to obtain 2.

B) Set the cost of K to 2

C) Set L as parent of K

D) Add K to open list

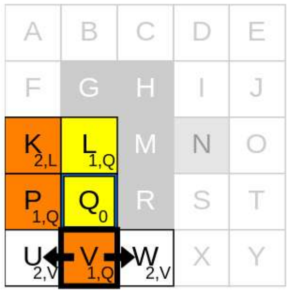
E) Add L to closed list



In the next iteration L will show up yellow to represent that it has been visited.

Third cycle:

Pick a new current node to analyze, we can take P or V , in this case we take V. Both nodes are inside the open list and have the smallest cost (1). Take V and set its cost neighbors U and W. Bot get the same cost (2). Then set V as the parent node of both and add U and W to the open list. Add V to the closed list

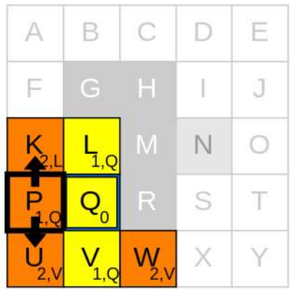


Next cycle (4):

A) Select a new current node, we select P because it has the smallest cost

B) Notice K and U have already a cost, we compare if the new cost is lower or not. As the new cost is 2, it's no necessary to update K and U. Likewise the parent node is not updated and remains it was.

C) Finally mark P as visited and add it to the closed list



Next cycle (5):

A) Get a new current node (K)

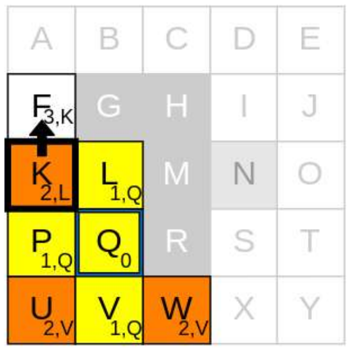
B) Get neighbor nodes (Ignore occupied and visited cells), F is only available

C) Update F cost (K cost plus step cost [3])

D) Set K as parent node of F

E) Add F to the open list

F) Add K to the closed list



Next cycle (6):

A) Get a new current node (W and U have the smallest cost)

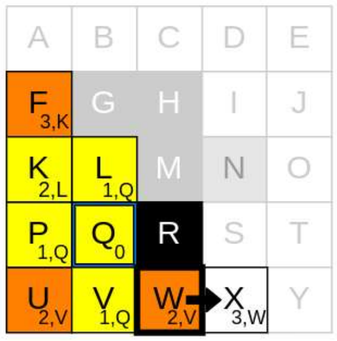
B) Get neighbor nodes (Ignore occupied and visited cells), X is only available

C) Update X cost (W cost plus step cost [3])

D) Set W as parent node of X

E) Add X to the open list

F) Add W to the closed list

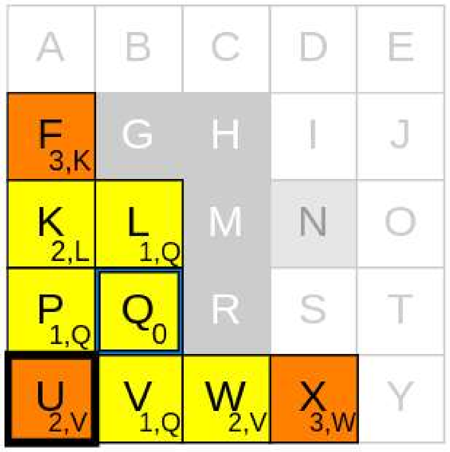


Next cycle (7):

A) Get a new current node (U have the smallest cost)

B) Get neighbor nodes, it doesn't have neighbors available

C) Add W to the closed list



Next cycle (8):

A) Get a new current node (F and X have the smallest cost)

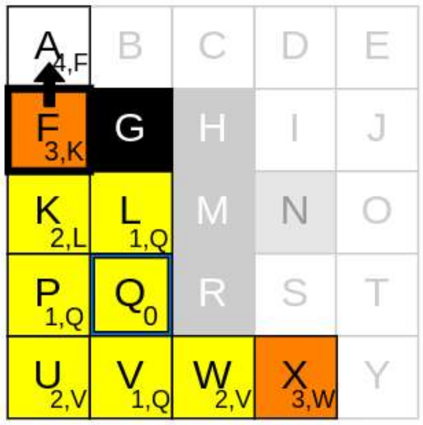
B) Get neighbor nodes (Ignore occupied and visited cells), A is only available

C) Update A cost (W cost plus step cost [4])

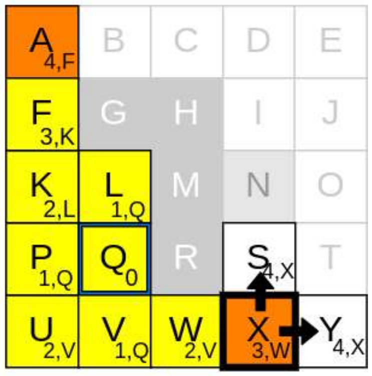
D) Set F as parent node of A

E) Add A to the open list

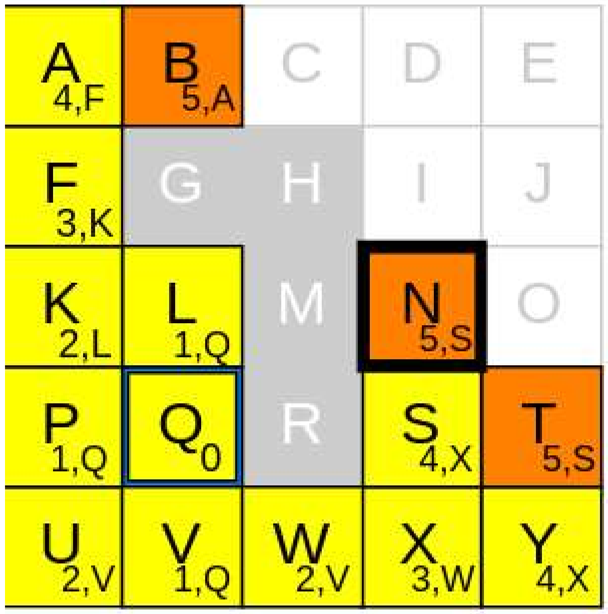
F) Add F to the closed list



Next cycle (9):
A) Get a new current node (X has the smallest cost)
B) Get neighbor nodes [S,Y]
C) Update neighbors cost [S(4),Y(4)]
D) Set X as parent node of S and Y
E) Add S and Y to the open list
F) Add X to the closed list



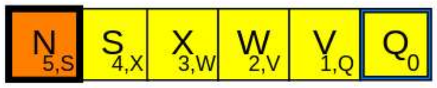
Next cycle (13):
We set N as the current Node. We have found our goal



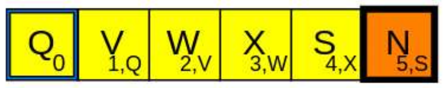
We have finished the process of the mapping to collect data. Now we need to build the shortest path. Once we have reached the target node, we can extract the shortest path by following every single's

parent node until we reach the start node.
First we take the target node and add it to a new list. Then we look up for the parent node of it. We add its parent (S) to the list. In the same way we add the parent of S which is X until we get the node Q, which has no parent node.

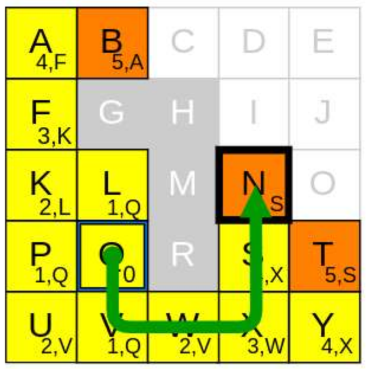
Once the backtracking is complete, the newly created list will contain the shortest path as a sequence of grid cells to visit, but in the wrong order:



To have the path from start to end, so all we need to do is reverse it:



Now we are done with the path finding process!


A 4,F
B 5,A
C
D
E
F 3,K
G
H
I
J
K 2,L
L 1,Q
M
N S
O
P 1,Q
Q 0
R
S ,X
T 5,S
U 2,V
V 1,Q
W 2,V
X 3,W
Y 4,X