



# 5 ROBOTIC PATHWAY PLANNING

**FEB 2023**

**Dr. Liu Fan**

**isslf@nus.edu.sg**



# Module objective

## Knowledge and understanding

- Understand the robotic path planning modelling.
- Understand various path planning algorithms.

## Key skills

- Apply various robotic path planning algorithms.



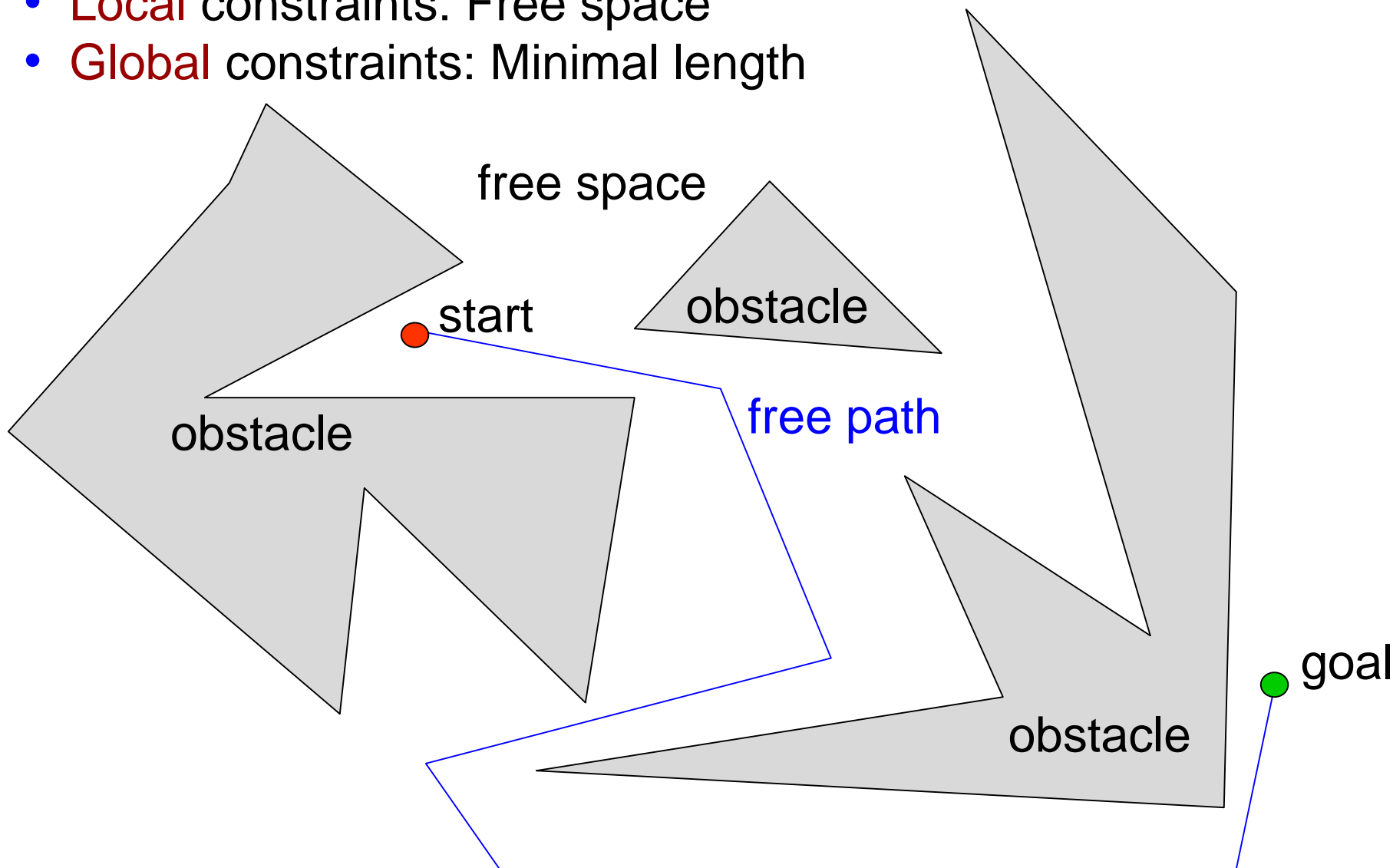
# Major reference

- [Intermediate] Nikolaus Correll. ***Introduction to Autonomous Robots***, Magellan Scientific, 2016, available at <https://github.com/correll/Introduction-to-Autonomous-Robots/releases>.
- [Intermediate] M. Ben-Ari and F. Mondada, ***Elements of Robotics***, Springer, 2018, available at <https://www.springer.com/gp/book/9783319625324>
- [Advanced] Steven M. LaValle, ***Planning algorithms***, Cambridge University Press, 2006, <http://planning.cs.uiuc.edu/>
- [Practical] A collection of Python examples for robotic systems, <https://github.com/AtsushiSakai/PythonRobotics>

- Introduction and motivation
- Various algorithms of robotic path planning
- Workshop 3 – Robotic pathway planning
- Demo for robotic path planning in Python

- **Introduction and motivation**
- Various algorithms of robotic path planning
- Workshop 3 – Robotic pathway planning
- Demo for robotic path planning in Python

- **Local** constraints: Free space
- **Global** constraints: Minimal length

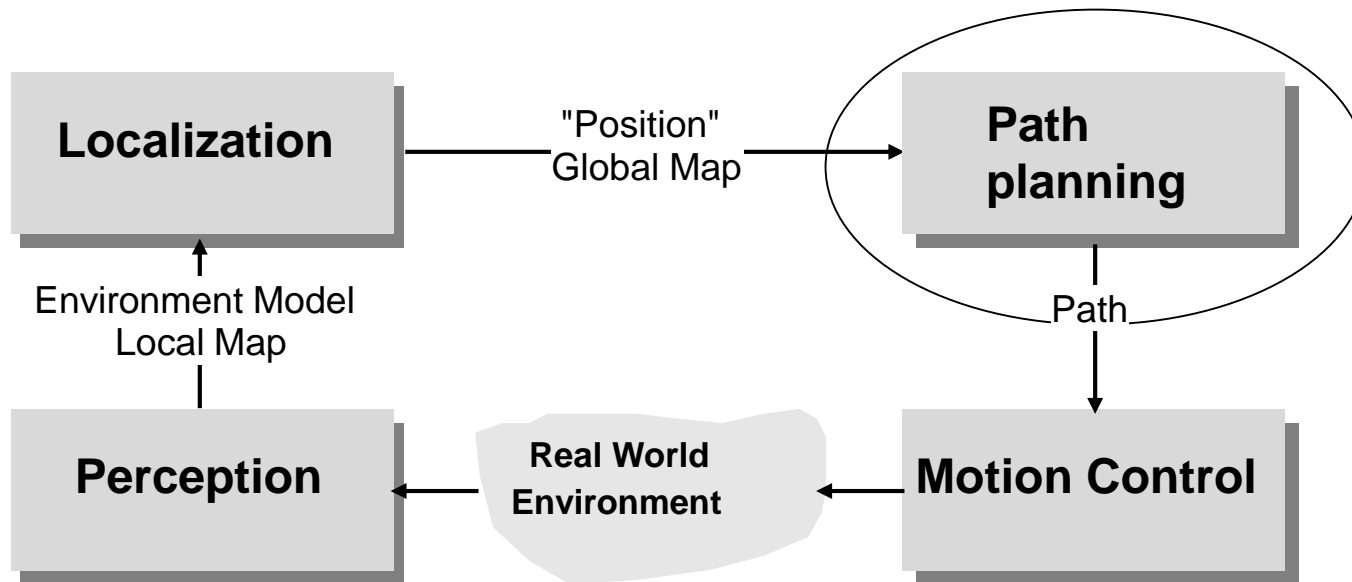


- The motion planning algorithms provide transformations how to move a robot (object) considering all operational constraints.



Source: Jan Faig, Robot Motion Planning, Lecture 9, A4M36PAH - Planning and Games.

- Path planning is a key component for robotic navigation







## Path planning

- Find a feasible **geometric path** for moving a mobile system from a starting position to a goal position
  - Given a geometric model of the environment with the obstacles and the free space
  - A path is feasible if it meets the kinematic constraints of the mobile system and avoids collision with obstacles

## Motion planning

- Find a feasible trajectory in space and time, which consists of both **feasible path and control law** along that path that meets the mobile system's dynamic constraints (speed and acceleration)
  - Relies on path planning



# Requirements of path planning



NUS  
National University  
of Singapore



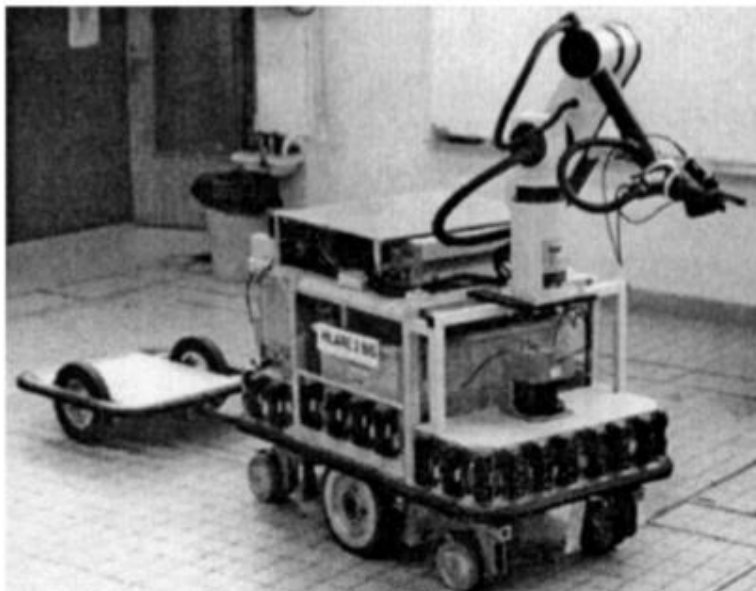
- Process inputs from sensors and communication channels, models of the environment and of the robot
- Handle noisy and partial knowledge of the state from information acquired through sensors and communication channels
- Direct integration of planning with acting, sensing, and learning



- **Car-like robot**
  - Three configuration parameters are needed to characterize its position:  $x, y, \theta$ 
    - Path planning defines a path in this space
  - The parameters are not independent
    - E.g., changing  $\theta$  requires changing  $x$  and  $y$
- **Mechanical arm** with  $n$  rotational joints
  - $n$  configuration parameters
    - Each gives the amount of rotation for one of the joints
  - Hence,  $n$ -dimensional space
  - Also, min/max rotational constraints for each joint

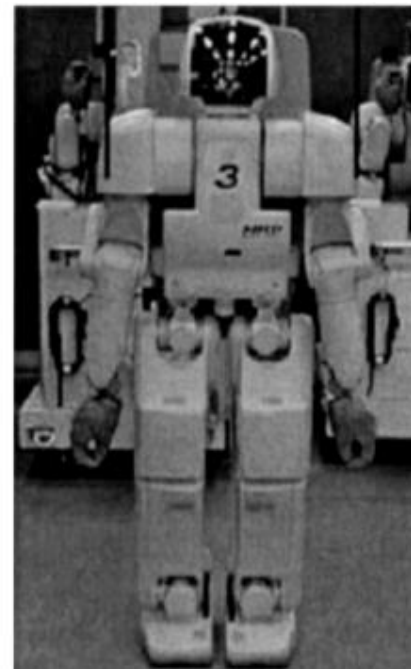


# Robotic configuration parameters



Example: 10 configuration parameters

- 6 for arm
- 4 for platform & trailer

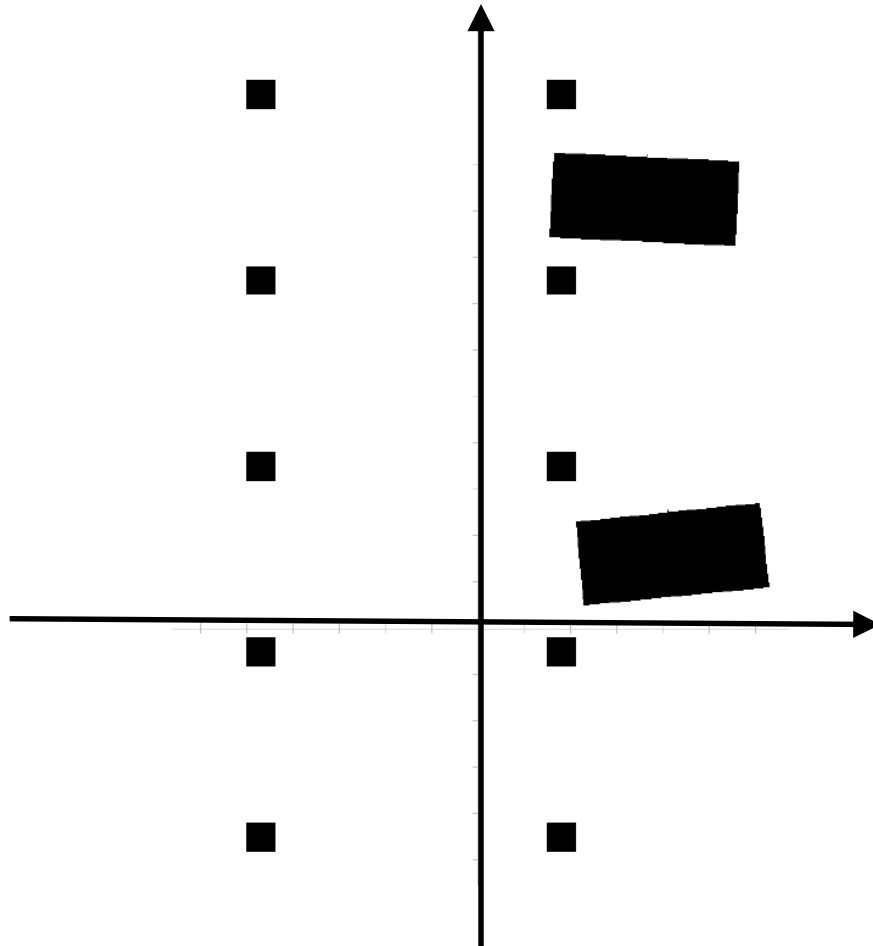


Example: 52 configuration parameters

- 2 for the head
- 7 for each arm
- 6 for each leg
- 12 for each hand

# Robotic configuration space

Workspace:  $(x, y)$



Free workspace:  $W_{free}$



Obstacle workspace:  $W_{obst}$

Source: Michal Cap, Motion Planning for Autonomous Vehicles, [https://cw.fel.cvut.cz/old/\\_media/courses/ae4m36pah/motion-planning.pdf](https://cw.fel.cvut.cz/old/_media/courses/ae4m36pah/motion-planning.pdf)

# Robotic configuration space



NUS  
National University  
of Singapore

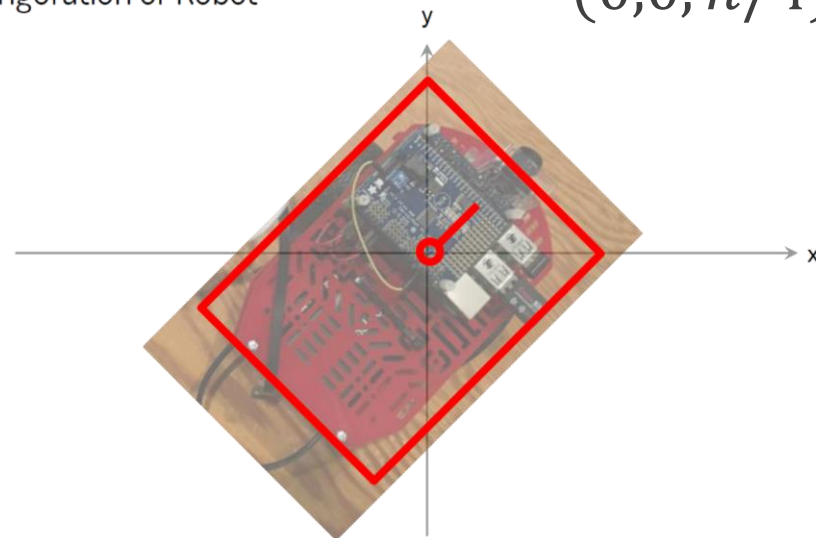
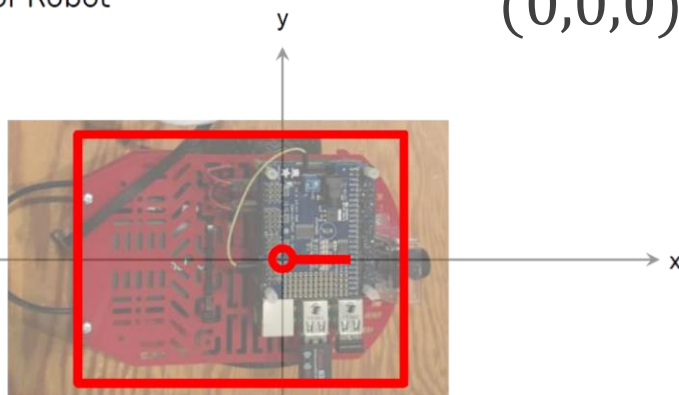


Configuration of Robot

$(0,0,0)$

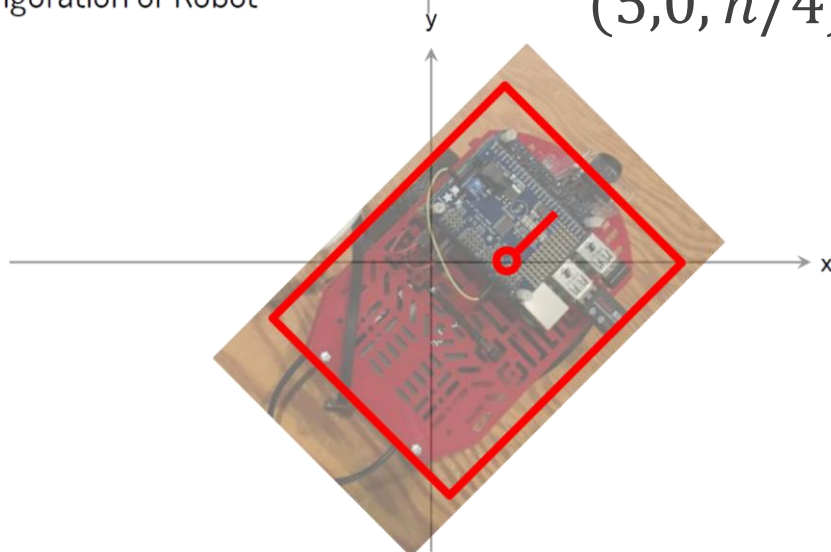
Configuration of Robot

$(0,0,n/4)$



Configuration of Robot

$(5,0,n/4)$

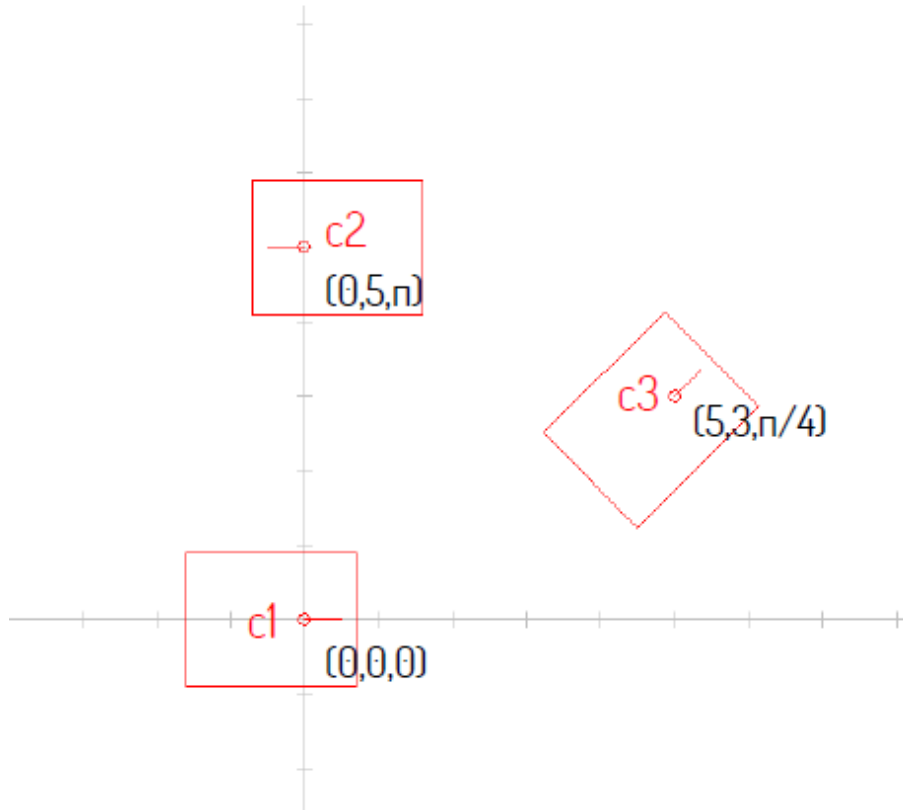


Configuration space:  $(x, y, \theta)$

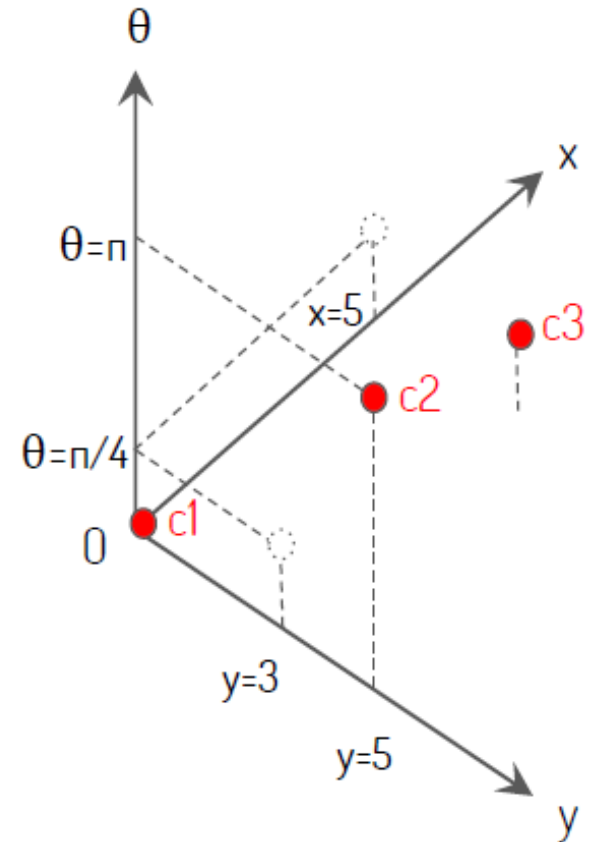
Source: Michal Cap, Motion Planning for Autonomous Vehicles, [https://cw.fel.cvut.cz/old/\\_media/courses/ae4m36pah/motion-planning.pdf](https://cw.fel.cvut.cz/old/_media/courses/ae4m36pah/motion-planning.pdf)

# Robotic configuration space

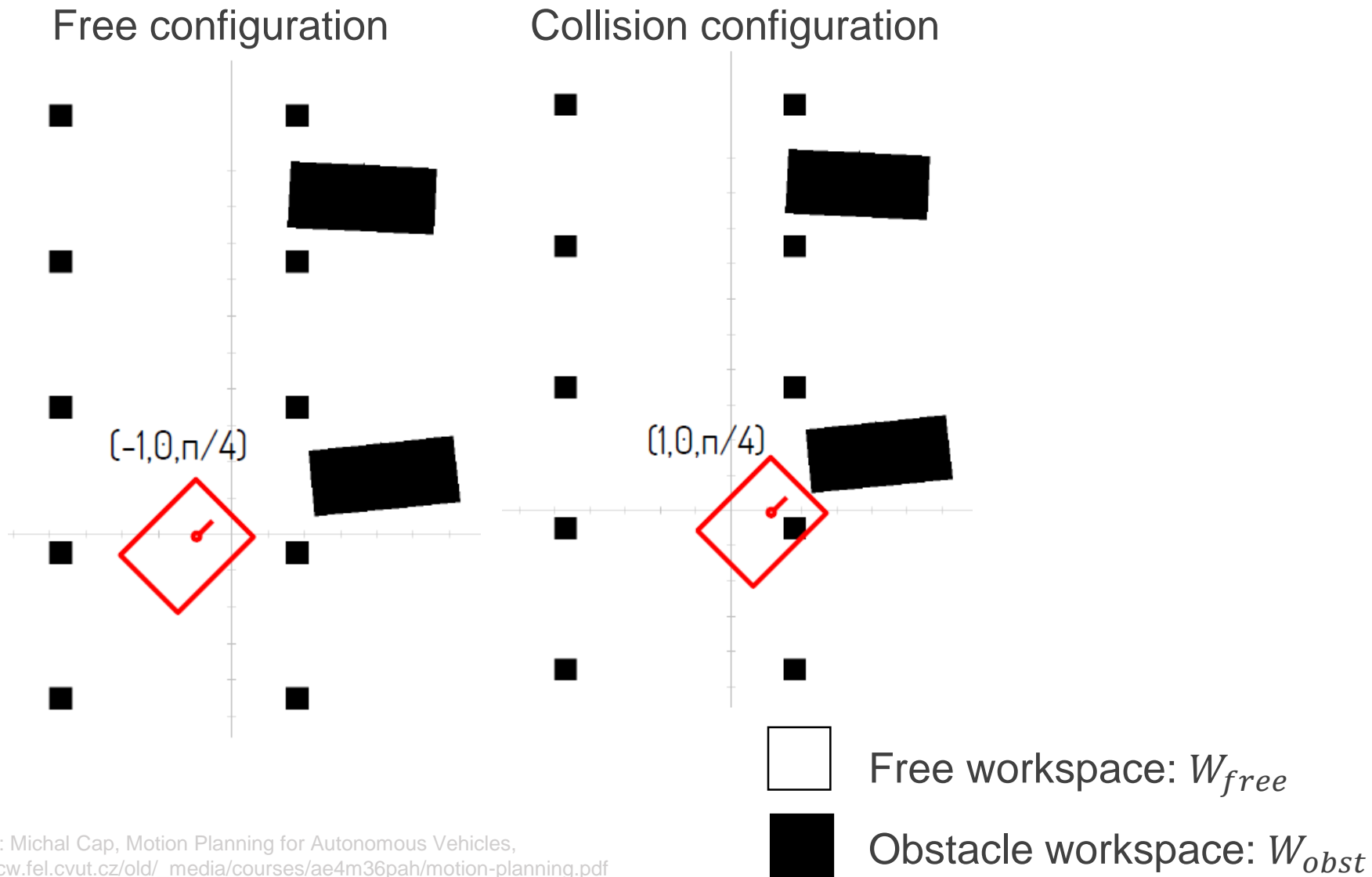
Workspace:  $(x, y)$



Configuration space:  $(x, y, \theta)$



# Robotic configuration space



Source: Michal Cap, Motion Planning for Autonomous Vehicles,  
[https://cw.fel.cvut.cz/old/\\_media/courses/ae4m36pah/motion-planning.pdf](https://cw.fel.cvut.cz/old/_media/courses/ae4m36pah/motion-planning.pdf)

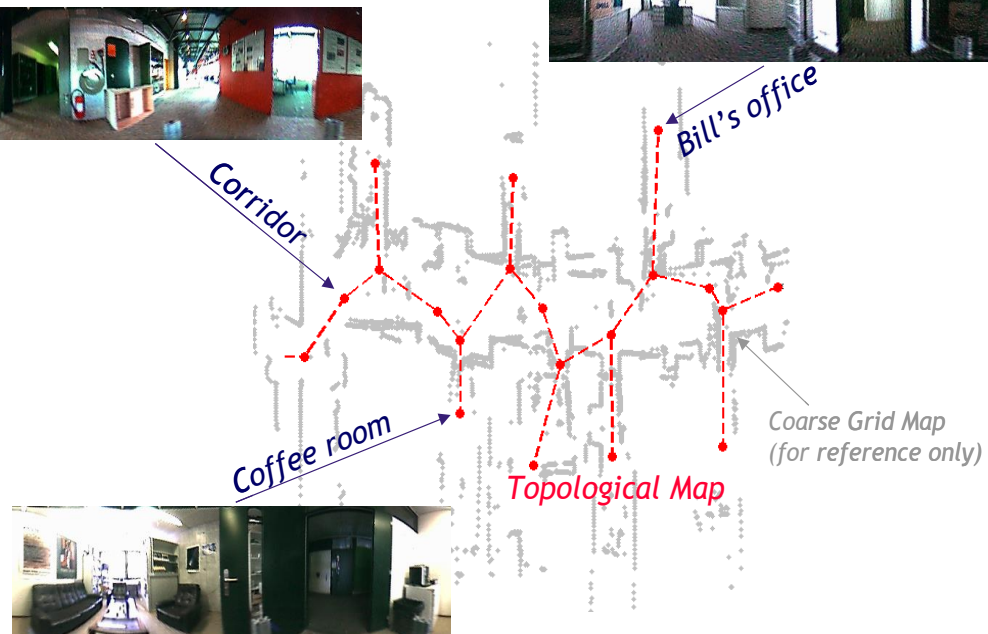




# Path planning problem

Objective: **Find a path in work space** (physical space) from the initial position to the goal position avoiding all collisions with obstacles.

- Transformation of the map into a representation for planning
- Plan a path on the transformed map
- Send motion commands to controller

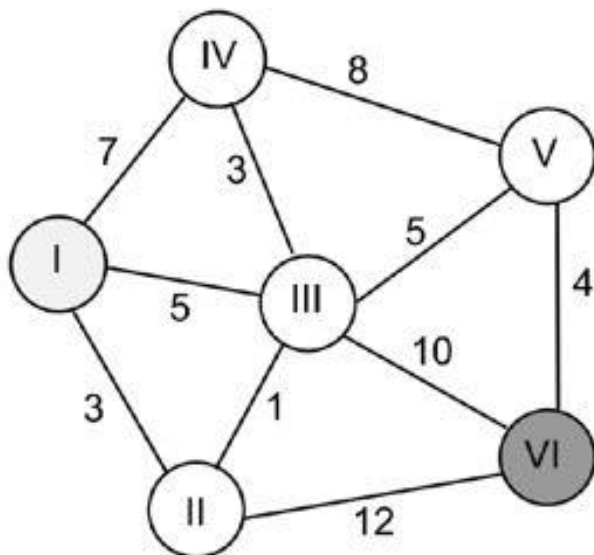


Source: Cornelia Fermüller, Path planning, CMSC498F, CMSC828K (Spring 2016), Robotics and Perception,  
<http://users.umi.acs.umd.edu/~fer/cmsc498F-828K/cmsc-498F-828K.htm>

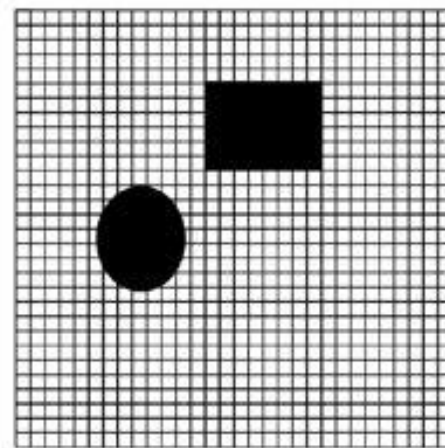
- Introduction and motivation
- **Various algorithms of robotic path planning**
- Workshop 3 – Robotic pathway planning
- Demo for robotic path planning in Python



# Map representation



Topological map  
(Continuous  
coordinates)



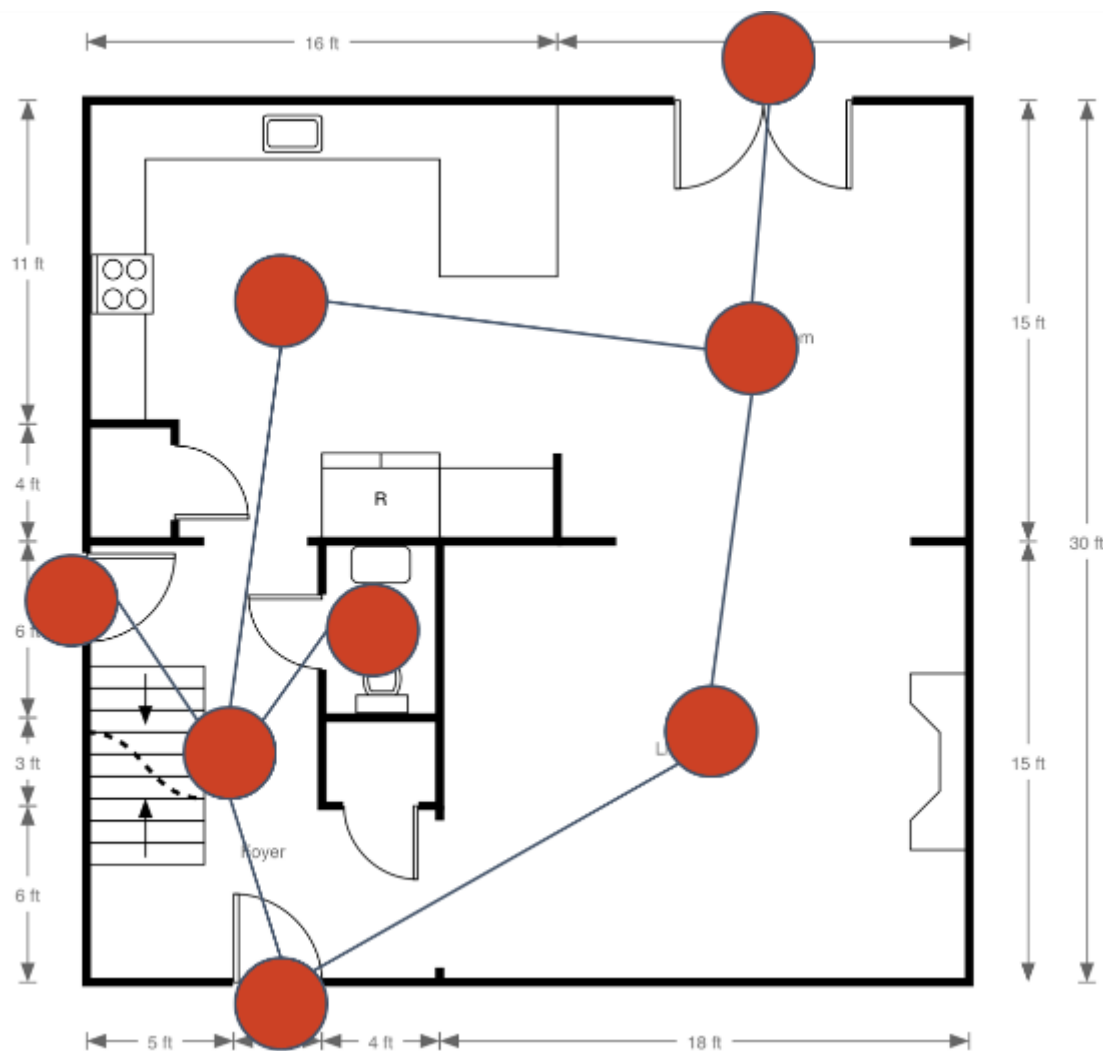
Occupancy grid  
map (Discrete  
coordinates)



# Topological map

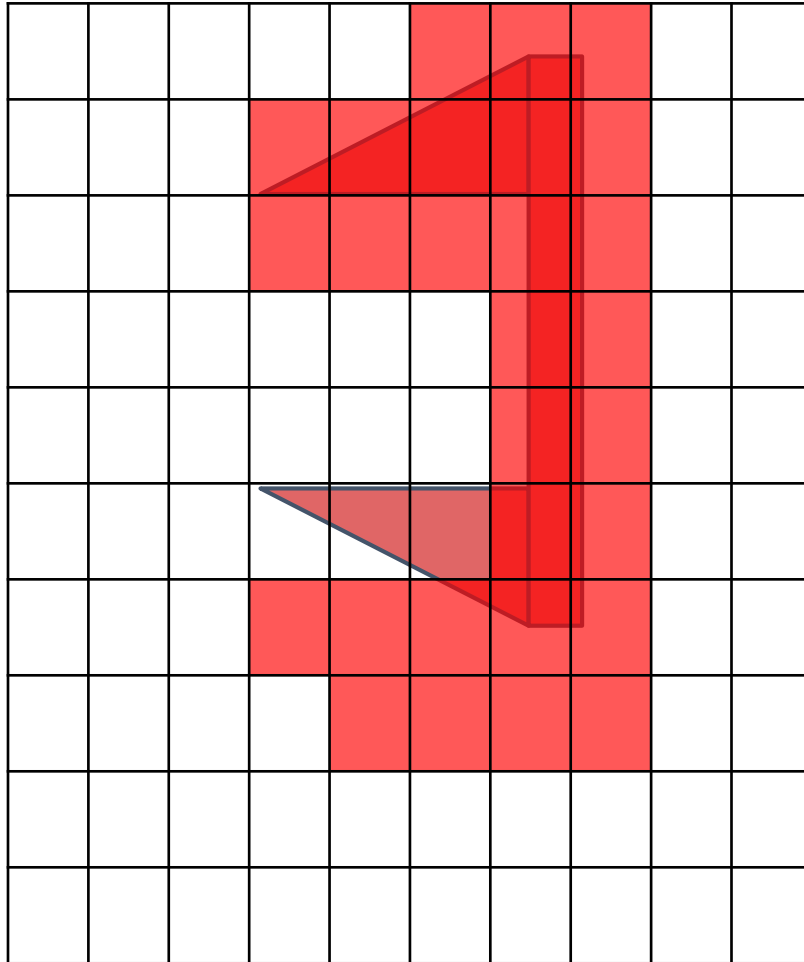


NUS  
National University  
of Singapore



- Edges can carry weights
- Many graph search algorithms
- Good abstract representation
- Assumption: Robot can travel between nodes
- Trade-off in Number of nodes: Complexity vs. accuracy

Source: Cornelia Fermüller, Path planning, CMSC498F, CMSC828K (Spring 2016), Robotics and Perception, <http://users.umiacs.umd.edu/~fer/cmssc498F-828K/cmssc-498F-828K.htm>

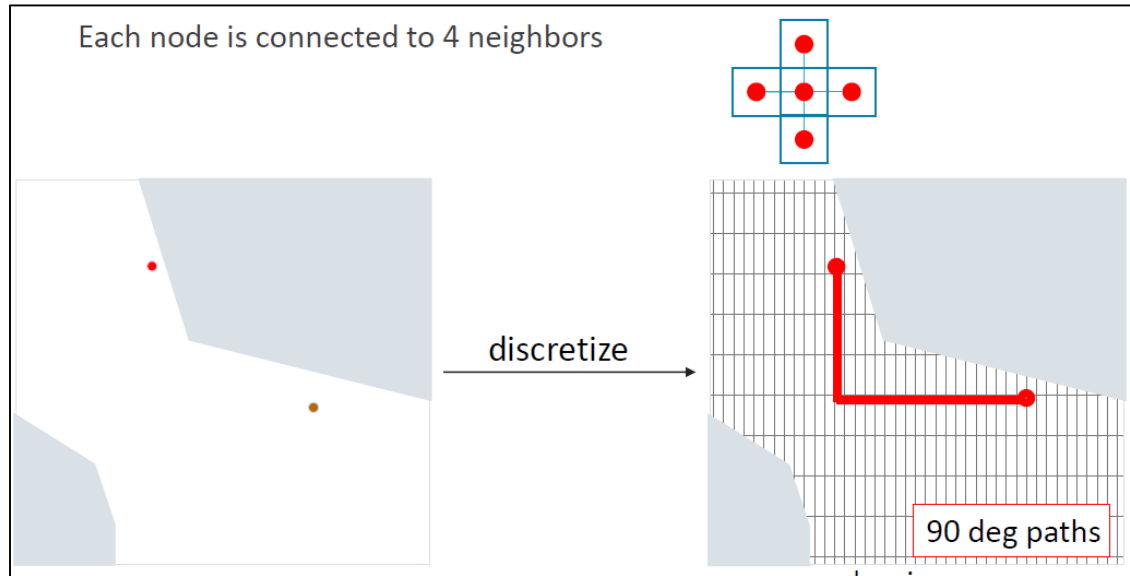


- Two dimensional array of values
- Each cell represents a square of real world
- Typically a few centimeters to a few dozen centimeters

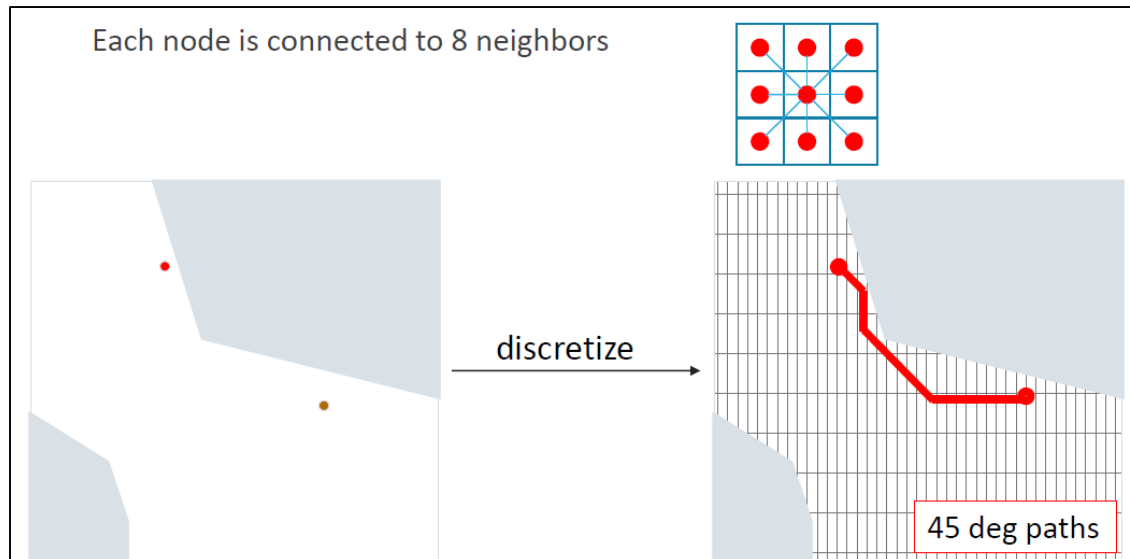
## S-RBS/Module 5 Robotic Pathway planning

Page 21

# Grid map: Neighborhood



4-connected neighborhood



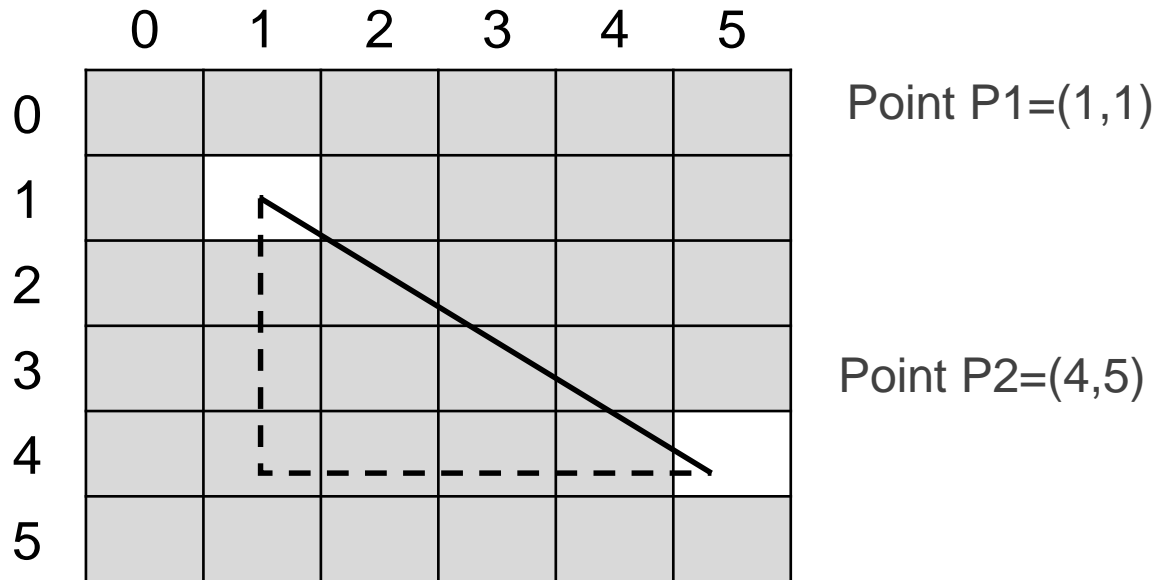
8-connected neighborhood

Source: Cornelia Fermüller, Path planning, CMSC498F, CMSC828K (Spring 2016), Robotics and Perception, <http://users.umiacs.umd.edu/~fer/cmssc498F-828K/cmssc-498F-828K.htm>

# Grid map: Distance metric

Given two points,  $P1 = (r1, c1), P2 = (r2, c2)$ ,

- **Manhattan distance** is defined as  $|r1 - r2| + |c1 - c2|$
- **Euclidean distance** is defined as  $\sqrt{(r1 - r2)^2 + (c1 - c2)^2}$

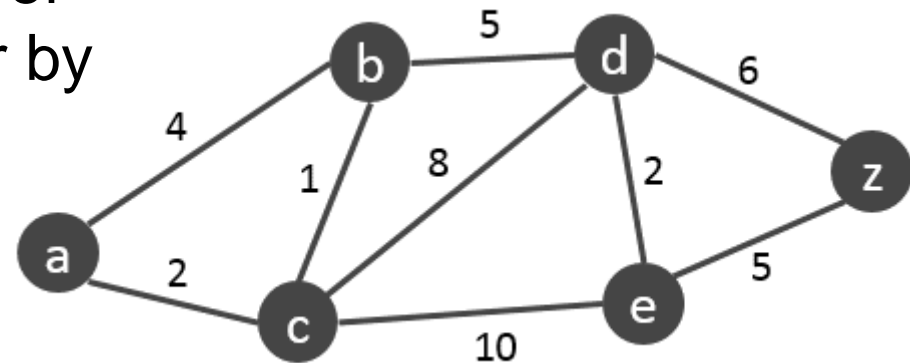


$$\text{Manhattan distance} = |1 - 4| + |1 - 5| = 7$$

$$\text{Euclidean distance} = \sqrt{(1 - 4)^2 + (1 - 5)^2} = 5$$

# Graph representation

A graph is a set of **nodes** and **edges**.  
The nodes (also called vertices or objects) are connected together by the edges (also called links or connections or arrows or arcs).



Set of nodes	a, b, c, d, e, z	
Set of edges	Node a	$a \rightarrow b, a \rightarrow c$
	Node b	$b \rightarrow a, b \rightarrow c, b \rightarrow d$
	Node c	$c \rightarrow a, c \rightarrow b, c \rightarrow d, c \rightarrow e$
	Node d	$d \rightarrow b, d \rightarrow c, d \rightarrow e, d \rightarrow z$
	Node e	$e \rightarrow d, e \rightarrow c, e \rightarrow z$
	Node z	$z \rightarrow d, z \rightarrow e$

Reference: <https://www.redblobgames.com/pathfinding/grids/graphs.html>





# Path planning methods



- **Dijkstra's method:** Instead of exploring all possible paths equally, it favors lower cost paths.



- **A\* method:** A modification of Dijkstra's Algorithm that is optimized for a single destination. Dijkstra's Algorithm can find paths to all locations; A\* finds paths to one location, or the closest of several locations. It prioritizes paths that seem to be leading closer to a goal.

# Dijkstra's method on grid (1)

	0	1	2	3	4	5
0						
1						
2						
3						
4	<b>S</b>					<b>G</b>

## Experimental setup

- Cell **S** is the starting cell, Cell **G** is the goal cell.
- Cells that contain obstacles are shown in black.
- Robot only can sense and move to a neighbour of the cell it currently occupies.
- The 4-connected neighbour is applied, that means the robot can move horizontally and vertically, not diagonally.
- The cell position is denoted by its row and column index, such as the starting cell S is denoted as (4,0).

# Dijkstra's method on grid (2)

	0	1	2	3	4	5
0						
1						
2						
3	1	2				
4	0 <sup>S</sup>	1				G

	0	1	2	3	4	5
0	4	5		7	8	9
1	3	4		6	7	8
2	2	3	4	5		
3	1	2		6	7	8
4	0 <sup>S</sup>	1		7	8	9 <sup>G</sup>

## Method

- Step 1: For each cell, we incrementally **mark each cell with the number of steps needed to reach from the starting cell.**
  - The cell (4,0) is marked as 0, the cell (3,0) is marked as 1.

# Dijkstra's method on grid (3)

	0	1	2	3	4	5
0	4	5		7	8	9
1	3	4		6	7	8
2	2	3	4	5		
3	1	2		6	7	8
4	0 <b>S</b>	1		7	8	9 <b>G</b>

	0	1	2	3	4	5
0	4	5		7	8	9
1	3	4		6	7	8
2	2	3	4	5		
3	1	2		6	7	8
4	0 <b>S</b>	1		7	8	9 <b>G</b>

## Method

- Step 1: For each cell, we incrementally mark each cell with the number of steps needed to reach from the starting cell.
  - The cell (4,0) is marked as 0, the cell (3,0) is marked as 1.
- Step 2: Find a shortest path by **working backwards from the goal cell until the starting cell**. For each cell marked  $n$ , we choose a cell marked  $n - 1$ .

Note: There might be multiple solutions!



# A\* method on grid (1)



	0	1	2	3	4	5
0	9	8		6	5	4
1	8	7		5	4	3
2	7	6	5	4		
3	7	5		3	2	1
4	<b>S</b> 5	4		2	1	<b>G</b> 0

The above figure is the manually-defined heuristic function used in this example.

## Key idea

- Uses extra information to guide the search
- Consider **not only the number of steps from the starting cell, but also a heuristic function that gives an indication of the estimated distance of each cell to the destination.**
- The cost function  $f(x, y)$  is defined as  $f(x, y) = g(x, y) + h(x, y)$ .



# A\* method on grid (2)



	0	1	2	3	4	5
0	9	8		6	5	4
1	8	7		5	4	3
2	7	6	5	4		
3	7	5		3	2	1
4	S 5	4		2	1	G 0

	0	1	2	3	4	5
0	4 13 9	5 13 8		7 13 6	8 13 5	9 13 4
1	3 11 8	4 11 7		6 11 5	7 11 4	8 11 3
2	2 9 7	3 9 6	4 9 5	5 9 4		
3	1 8 7	2 7 5		6 9 3	7 9 2	8 9 1
4	0 5 5	1 5 4		7 9 2	8 9 1	9 9 0
	S					G

Method

- **Step 1:** We incrementally mark the cell with the cost function as

g	f
	h



# A\* method on grid (3)



	0	1	2	3	4	5
0	9	8		6	5	4
1	8	7		5	4	3
2	7	6	5	4		
3	1 8 7	2 7 5		3	2	1
4	0 5 <b>S</b> 5	1 5 4		2	1	<b>G</b> 0

## Method

- Step 1: We incrementally mark the cell with the cost function.
- Step 2:** We maintain a list of cells, **open cells**, the cells that have not yet been expanded, as the notation  $(r, c, f)$ , where  $r$  and  $c$  are two and column of the cell and  $f$  is the cost function value.
- Each time, when the cell is expanded, it is removed from the list and new cells are added. The list is ordered so that cells with lowest values appear first

The first three lists of open cells  
 $(4,0,5)$   
 $(4,1,5), (3,0,8)$   
 $(3,1,7), (2,0,9)$



# A\* method on grid (4)



	0	1	2	3	4	5				
0	9	8		6	5	4				
1	8	7		5	4	3				
2	7	3	9	4	9	5	9			
3	1	8	2	7	6	9	7	9	8	9
4	0	5	1	5				9	9	
	S	5		4		2	1	G	0	

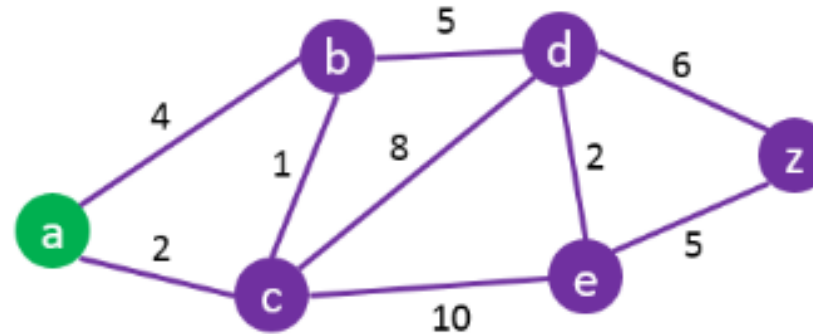
## Method

- Step 1: We incrementally mark the cell with the cost function.
- Step 2: We maintain a list of cells, open cells, the cells that have not yet been expanded, as the notation  $(r, c, f)$ , where  $r$  and  $c$  are two and column of the cell and  $f$  is the cost function value.
- Each time, when the cell is expanded, it is removed from the list and new cells are added. The list is ordered so that cells with lowest values appear first.
- Step 3: Expand cells until the goal cell is reached.



# Dijkstra's method on graph

**Objective:** Find the shortest path from a to z



Dijkstra's algorithm steps:

- 1) Mark the initial node with a current distance 0 and the rest with infinity
- 2) Set the unvisited node with the smallest current distance as the current node C
- 3) For each node N connected to the current node C: add the current distance of C with the weight of the edge connecting C-N. if it's smaller than the current distance of N, update it as the new current distance of N.
- 4) Mark the current node C as visited.
- 5) If there are unvisited nodes. Go to step 2

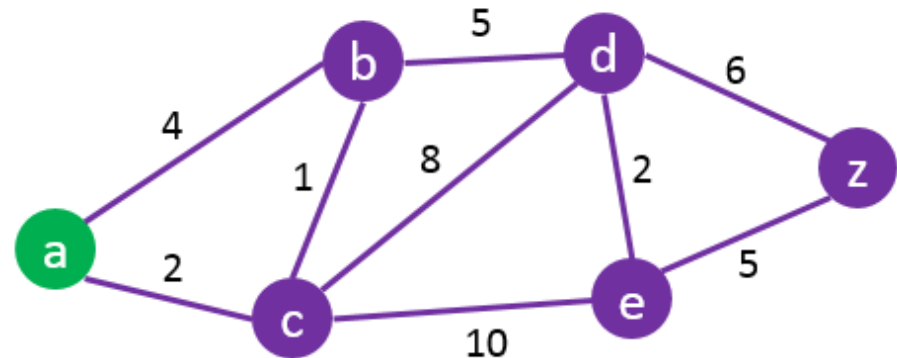
Reference: <https://www.101computing.net/dijkstras-shortest-path-algorithm/>



# Dijkstra's method on graph (1)

## Step1:

Start by setting the starting node(A) as the current node.  
Set the current distance from other nodes to A with infinity.



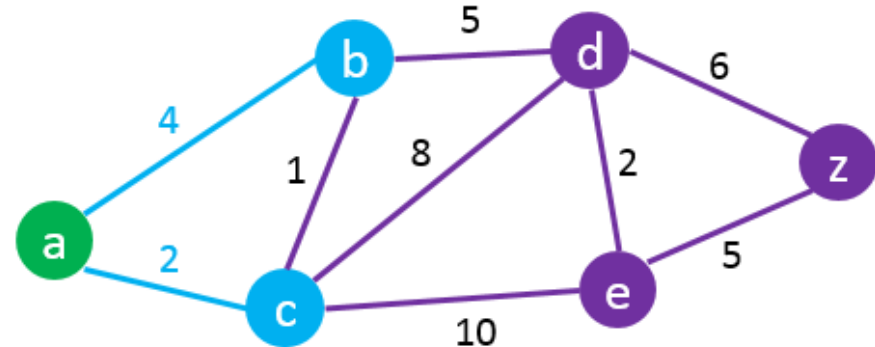
Node	Status	Shortest Distance From A	Previous Node
A	Current Node	0	
B		$\infty$	
C		$\infty$	
D		$\infty$	
E		$\infty$	
Z		$\infty$	



# Dijkstra's method on graph (2)

## Step2:

Check all the nodes connected to A and update their “**Distance from A**” and set their “**previous node**” to “A”.

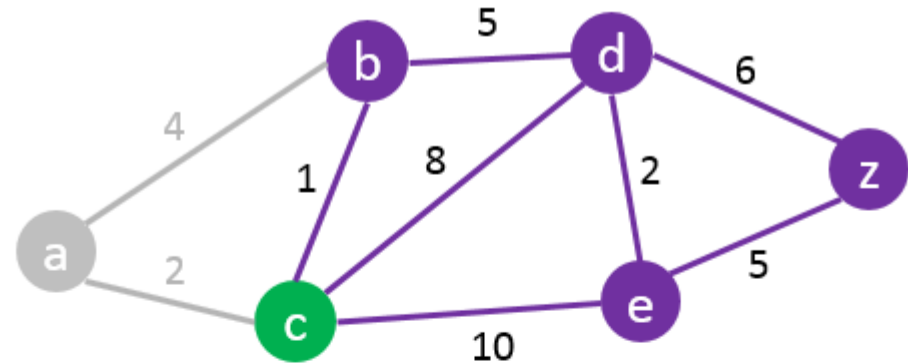


Node	Status	Shortest Distance From A	Previous Node
A	Current Node	0	
B		$\infty$ 4	A
C		$\infty$ 2	A
D		$\infty$	
E		$\infty$	
Z		$\infty$	

# Dijkstra's method on graph (3)

## Step3:

Set the current node A to **“visited”** and use the closest unvisited node to A as the **current node** (e.g. in this case: Node C).



Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B		$\infty$ 4	A
C	Current Node	$\infty$ 2	A
D		$\infty$	
E		$\infty$	
Z		$\infty$	

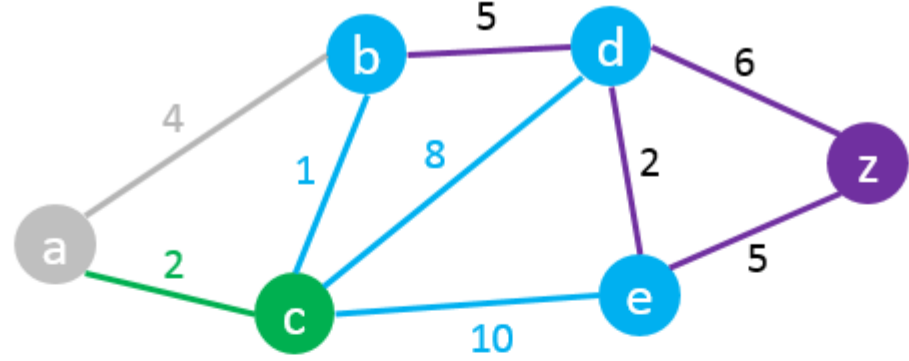


# Dijkstra's method on graph (4)



## Step4:

Check all unvisited nodes connected to the current node and add the distance from A to C to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.



C -> B:  $2 + 1 = 3 < 4$  – Change Node B

C -> D:  $2 + 8 = 10 < \infty$  – Change Node D

C -> E:  $2 + 10 = 12 < \infty$  – Change Node E

Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B		<del>4</del> $2+1=3$	C
C	Current Node	2	A
D		<del><math>\infty</math></del> $2+8=10$	C
E		<del><math>\infty</math></del> $2+10=12$	C
Z		$\infty$	



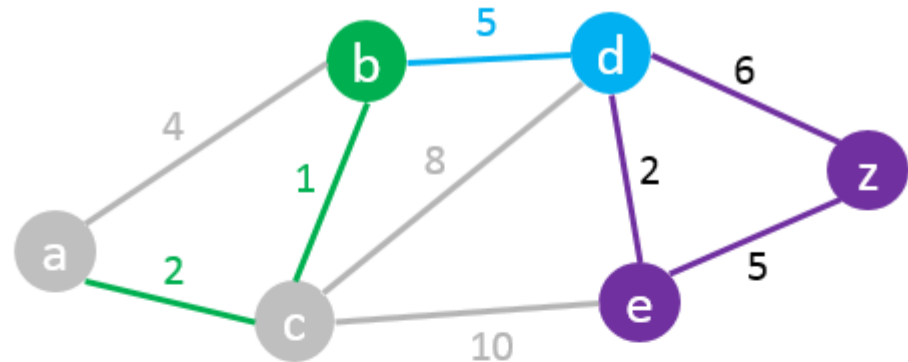
# Dijkstra's method on graph (5)

## Step5:

Set the current node C status to Visited.

We then repeat the same process always picking the closest unvisited node to A as the current node.

In this case node B becomes the current node.



Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Current Node	3	C
C	Visited Node	2	A
D		10	C
E		12	C
Z		$\infty$	

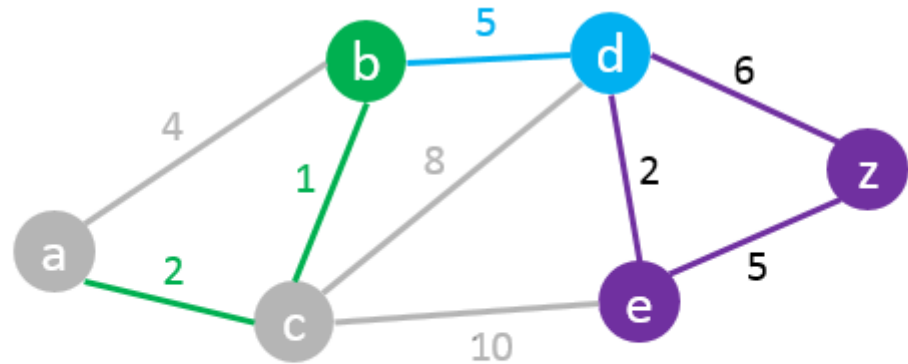


# Dijkstra's method on graph (6)

## Step6:

B  $\rightarrow$  D  $3+5 = 8 < 10$  – Change  
Node D

Next “Current Node” will be D  
as it has the shortest distance  
from A amongst all unvisited  
nodes.



Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Current Node	3	C
C	Visited Node	2	A
D		<del>10</del> $3+5=8$	B
E		12	C
Z		$\infty$	



# Dijkstra's method on graph (7)

## Step7:

D  $\rightarrow$  E  $8+2 = 10 < 12$  –

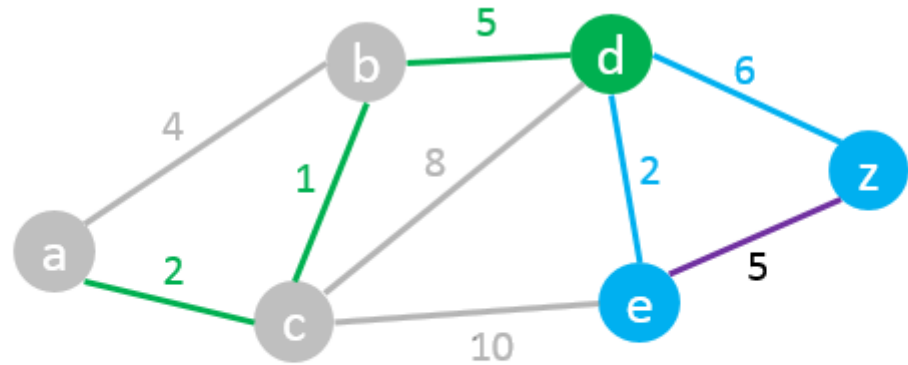
Change Node E

D  $\rightarrow$  Z  $8+6 = 14 < \infty$  – Change

Node Z

We found a path from A to Z  
but it may not be the shortest  
one yet. So we need to carry  
on the process.

Next “**Current Node**”: E



Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Visited Node	3	C
C	Visited Node	2	A
D	Current Node	8	B
E		<del>12</del> $8 + 2 = 10$	D
Z		<del><math>\infty</math></del> $8 + 6 = 14$	D





# Dijkstra's method on graph (8)



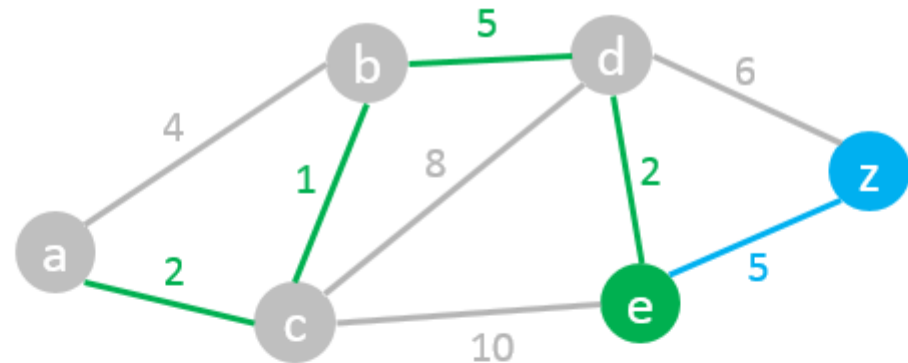
NUS  
National University  
of Singapore



Step8:

E  $\rightarrow$  Z  $10+5 = 15 > 14$  –

We do **not** change node Z.



Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Visited Node	3	C
C	Visited Node	2	A
D	Visited Node	8	B
E	Current Node	10	D
Z		14 $10 + 5 = 15$	D



# Dijkstra's method on graph (9)

## Step9:

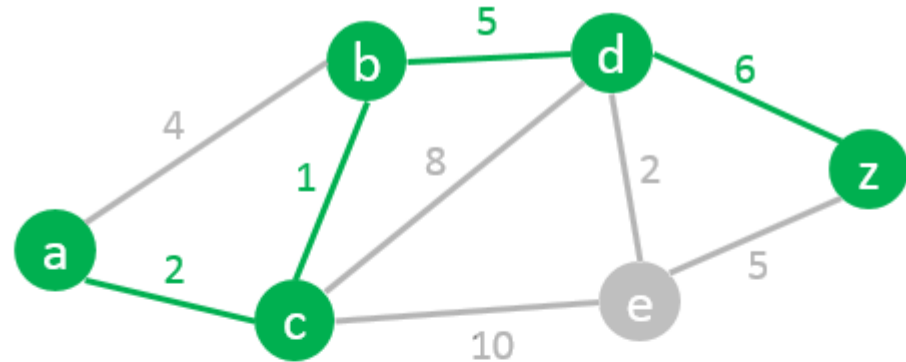
We found the shortest path from A to Z.

Read the path from Z to A using the previous node column:

$Z > D > B > C > A$

So the Shortest Path is:

**A – C – B – D – Z with a length of 14**

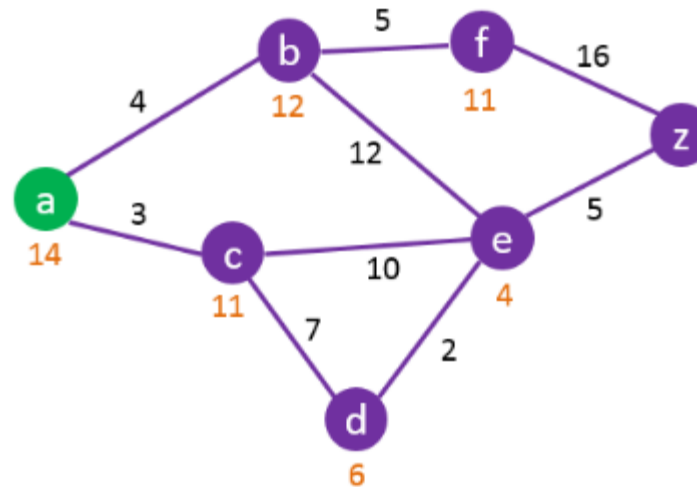


Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Visited Node	3	C
C	Visited Node	2	A
D	Visited Node	8	B
E	Visited Node	10	D
Z	Current Node	14	D



# A\* method on graph

**Objective:** Find the shortest path from a to z



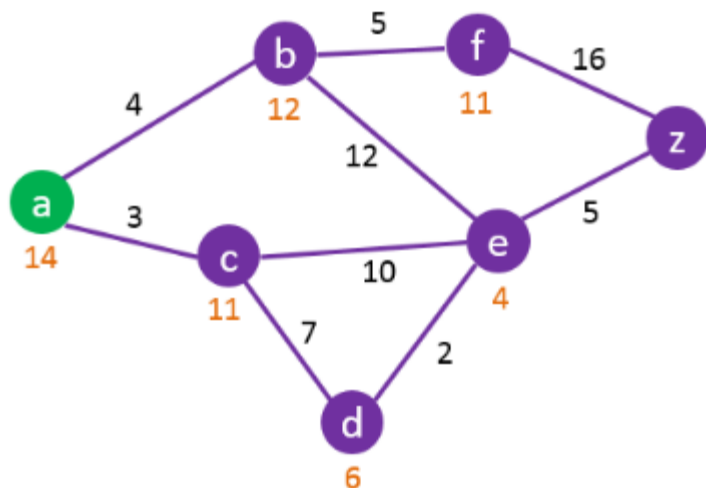
A\* algorithm:

- 1) It is an alternative to the Dijkstra's algorithm. It is used to find the shortest path between two nodes of a weighted graph. The A\* Search algorithm performs better than the Dijkstra's algorithm because of its use of **heuristics**. This technique finds minimal cost solutions.
- 2) The cost function is as follow:

$$f(n) = g(n) + h(n)$$



# A\* method on graph (1)



- Consider both distance from the starting node and the heuristic distance to the goal node.
- The distance of straight line is used as heuristic distance in this example.
- Start by setting the starting node A as the current node.

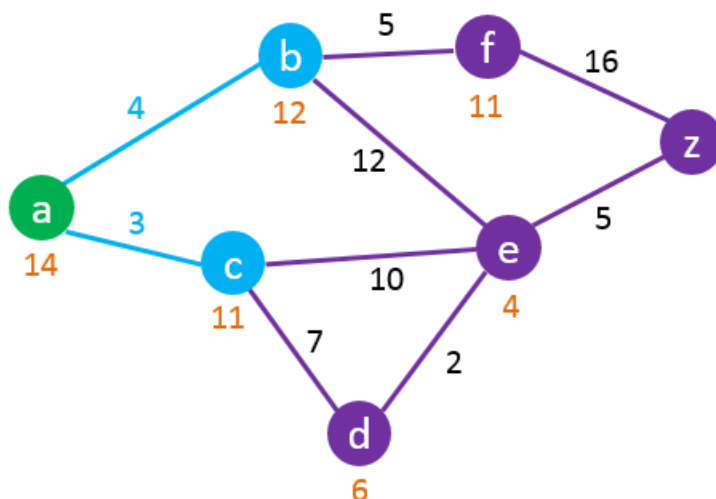
Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Current	0	14	14	
B		$\infty$	12		
C		$\infty$	11		
D		$\infty$	6		
E		$\infty$	4		
F		$\infty$	11		
Z		$\infty$	0		

\* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Source: <https://www.101computing.net/a-star-search-algorithm/>



# A\* method on graph (2)



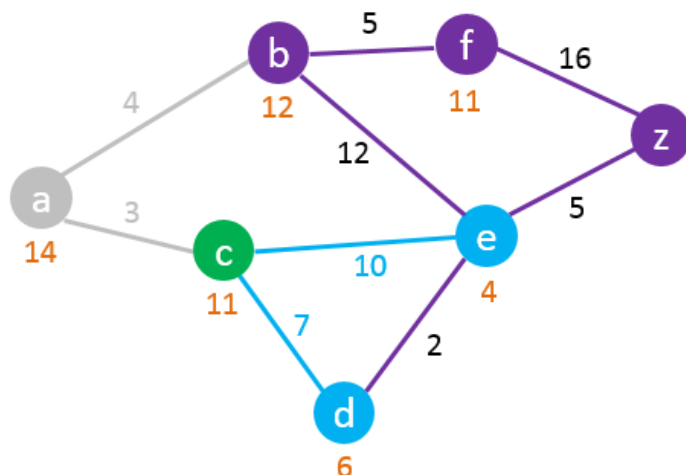
- Check all the nodes connected to A and update their “**Shortest Distance from A**” and set their “**previous node**” to “A”.
- Update their total distance by adding the shortest distance from A and the heuristic distance to Z.

Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Current	0	14	14	
B		$\infty$ 4	12	16	A
C		$\infty$ 3	11	14	A
D		$\infty$	6		
E		$\infty$	4		
F		$\infty$	11		
Z		$\infty$	0		
* Total Distance = Shortest Distance from A + Heuristic Distance to Z					

Source: <https://www.101computing.net/a-star-search-algorithm/>



# A\* method on graph (3)



- Set the current node (A) to “visited” and use the unvisited node with the smallest total distance as the current node (Node C).
- Check all unvisited nodes connected to the current node and add the distance from A to C to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

- C -> D:  $3 + 7 = 10 < \infty$ , so change Node D
- C -> E:  $3 + 10 = 13 < \infty$ , so change Node E
- The next current node (unvisited node with the shortest total distance) could be either node B or node D. Let’s use node B.

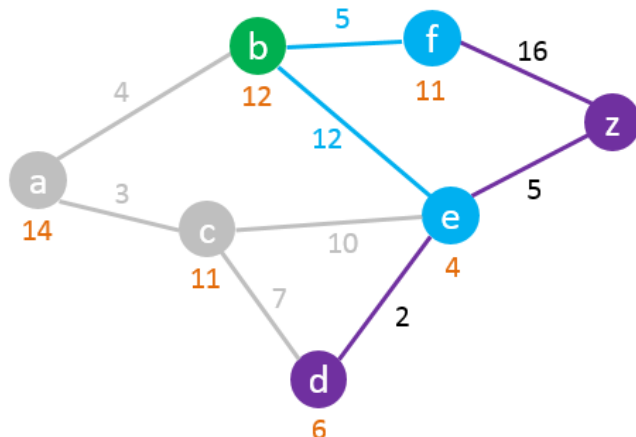
Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B		4	12	16	A
C	Current	3	11	14	A
D		$\infty$ $3+7=10$	6	16	C
E		$\infty$ $3+10=13$	4	17	C
F		$\infty$	11		
Z		$\infty$	0		

\* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Source: <https://www.101computing.net/a-star-search-algorithm/>



# A\* method on graph (4)



- Check all unvisited nodes connected to the current node (B) and add the distance from A to B to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

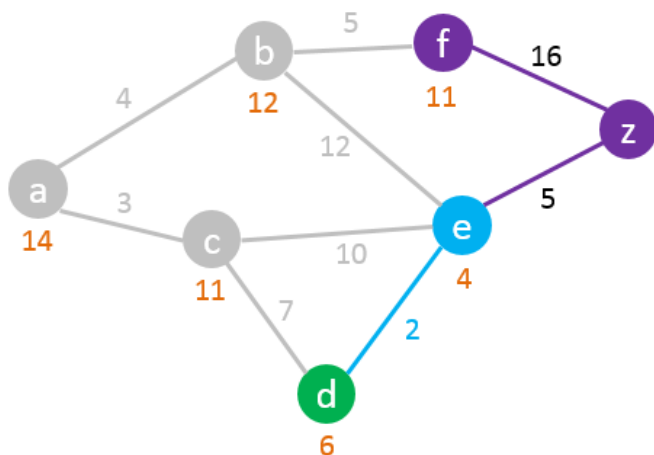
Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Current	4	12	16	A
C	Visited	3	11	14	A
D		10	6	16	C
E		13 4+12=16	4	17	C
F		∞ 4+5=9	11	20	B
Z		∞	0		

\* Total Distance = Shortest Distance from A + Heuristic Distance to Z

- B -> E:  $4 + 12 = 16 > 13$  – Do not change Node E
  - B -> F:  $4 + 5 = 9 < \infty$  – Change Node F
- The next current node (unvisited node with the shortest total distance) is D.



# A\* method on graph (5)



- Check all unvisited nodes connected to the current node (D) and add the distance from A to D to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Visited	4	12	16	A
C	Visited	3	11	14	A
D	Current	10	6	16	C
E		<del>13</del> 10+2=12	4	16	D
F		9	11	20	B
Z		$\infty$	0		

\* Total Distance = Shortest Distance from A + Heuristic Distance to Z

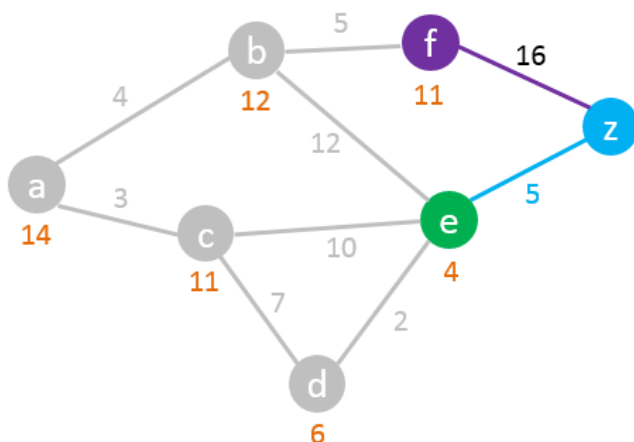
- D -> E:  $10 + 2 = 12 < 13$  – Change Node E

The next current node (unvisited node with the shortest total distance) is E.





# A\* method on graph (6)



- Check all unvisited nodes connected to the current node (E) and add the distance from A to E to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

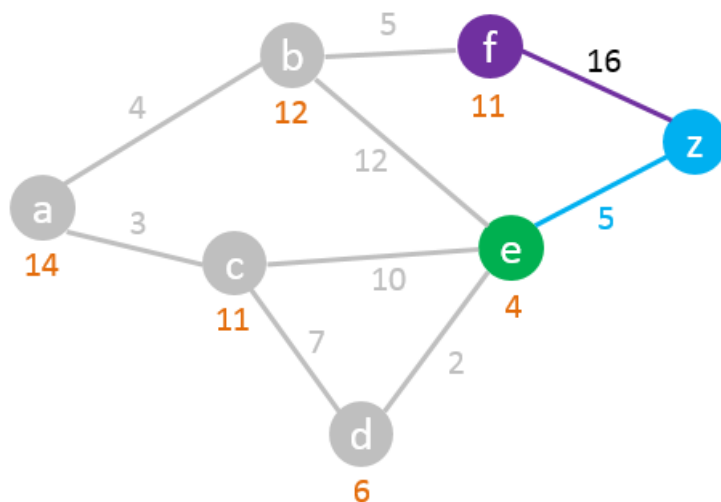
- E -> Z:  $12 + 5 = 17 < \infty$  – Change Node Z

Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Visited	4	12	16	A
C	Visited	3	11	14	A
D	Visited	10	6	16	C
E	Current	12	4	16	D
F		9	11	20	B
Z		$\infty$ 12+5=17	0	17	E

\* Total Distance = Shortest Distance from A + Heuristic Distance to Z



# A\* method on graph (7)



We found a path from A to Z, but is it the shortest one?

Check all unvisited nodes. In this example, there is only one unvisited node (F).

However its total distance (20) is already greater than the distance we have from A to Z (17) so there is no need to visit node F as it will not lead to a shorter path.

We found the shortest path from A to Z.

Read the path from Z to A using the previous node column:

Z > E > D > C > A

So the Shortest Path is:

**A – C – D – E – Z with a length of 17**

Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Visited	4	12	16	A
C	Visited	3	11	14	A
D	Visited	10	6	16	C
E	Visited	12	4	16	D
F		9	11	20	B
Z	Current	17	0	17	E

\* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Source: <https://www.101computing.net/a-star-search-algorithm/>

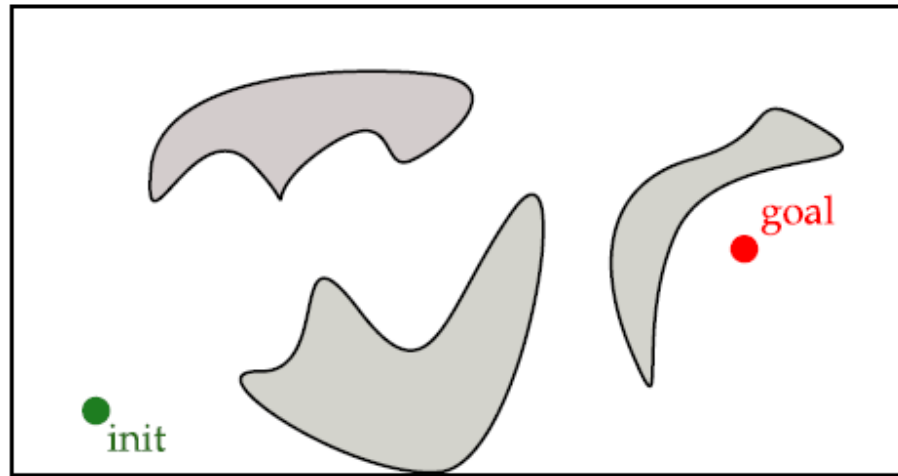
# Path planning: Sampling-based method



NUS  
National University  
of Singapore



- When the state space is large complete solutions are often infeasible.
- Sampling-based planners create possible paths by randomly adding points to a tree until some solution is found



Source: Cornelia Fermüller, Path planning, CMSC498F, CMSC828K (Spring 2016), Robotics and Perception, <http://users.umiacs.umd.edu/~fer/cmssc498F-828K/cmssc-498F-828K.htm>



# Summary of path planning algorithms



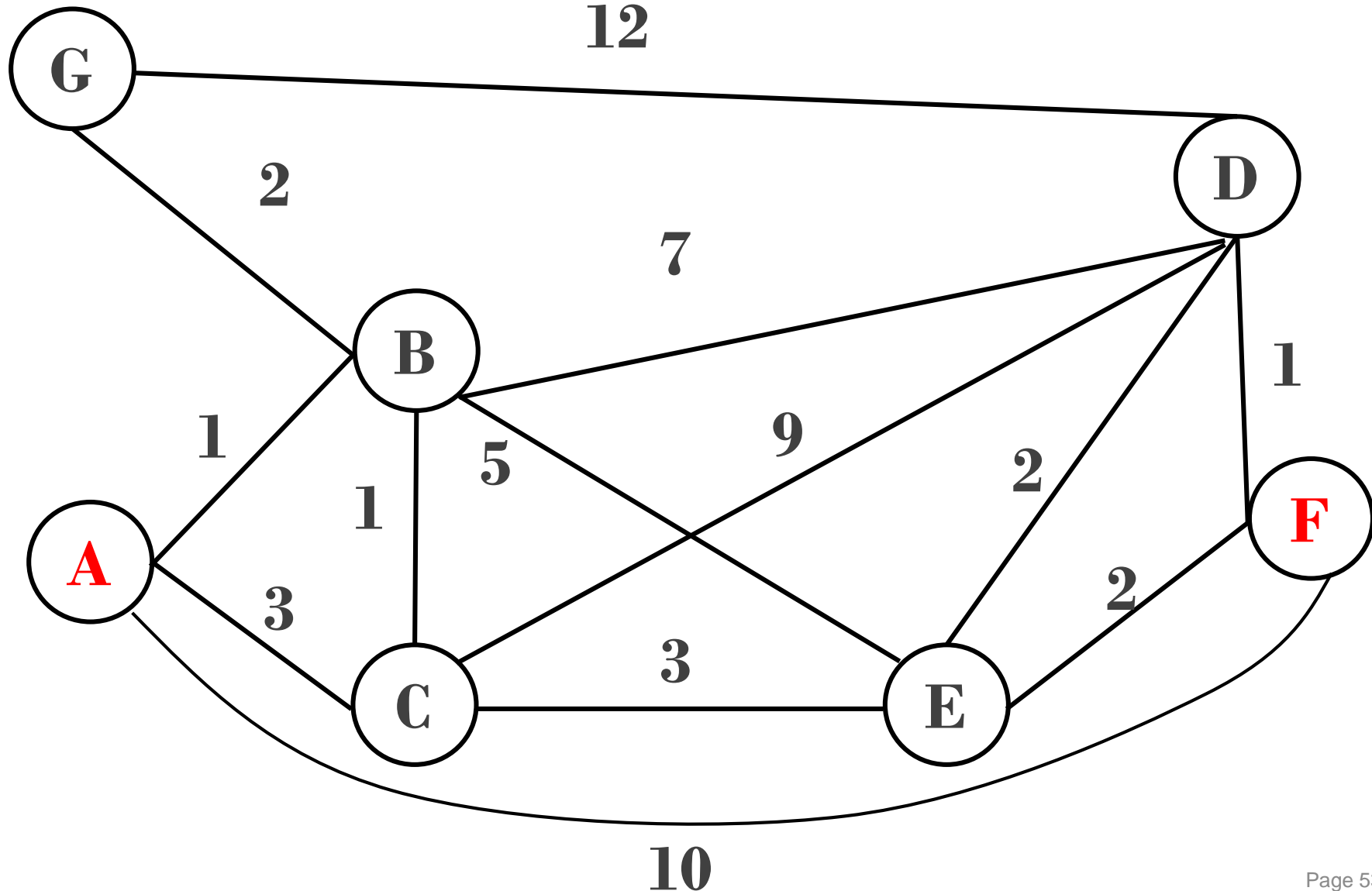
NUS  
National University  
of Singapore



- First step in addressing a planning problem is choosing a map representation. Reduce robot to a point-mass by inflating obstacles.
- Grid-based algorithms are complete, sampling-based ones probabilistically complete, but usually faster
- Most real planning problems require combination of multiple algorithms

# Exercise

Find the shortest path from A to F using Dijkstra's method



- Introduction and motivation
- Various algorithms of robotic path planning
- **Workshop 3 – Robotic pathway planning**
- Demo for robotic path planning in Python



# Workshop 3 – Robotic pathway planning

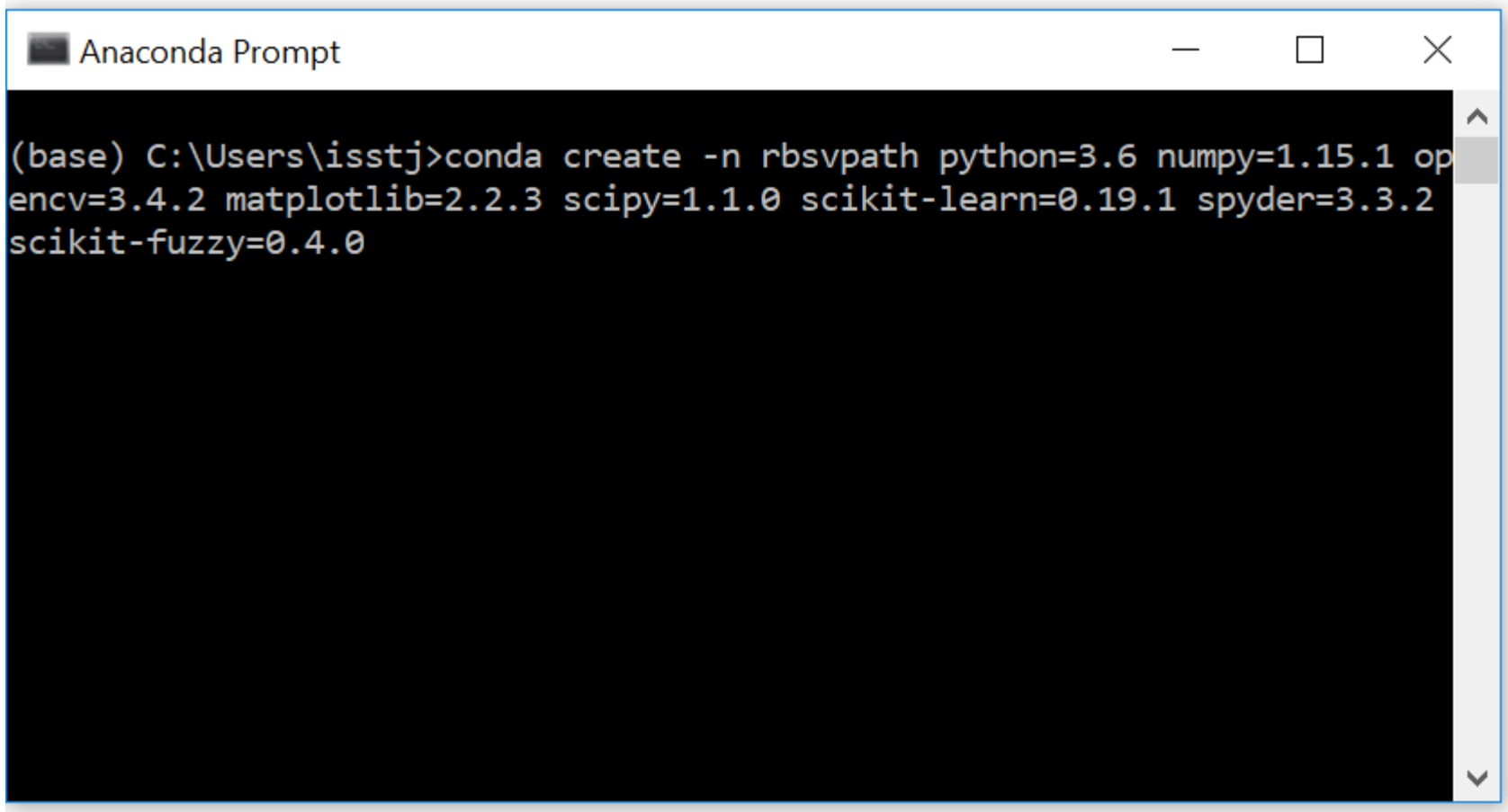
- Refer to workshop 3 document.

- Introduction and motivation
- Various algorithms of robotic path planning
- Workshop 3 – Robotic pathway planning
- **Demo for robotic path planning in Python**



- Type following commands in **Anaconda Prompt**

*conda create -n **rbsvpath** python=3.6 numpy=1.15.1 opencv=3.4.2 matplotlib=2.2.3  
scipy=1.1.0 scikit-learn=0.19.1 spyder=3.3.2 scikit-fuzzy=0.4.0*



```
(base) C:\Users\isstj>conda create -n rbsvpath python=3.6 numpy=1.15.1 op  
encv=3.4.2 matplotlib=2.2.3 scipy=1.1.0 scikit-learn=0.19.1 spyder=3.3.2  
scikit-fuzzy=0.4.0
```

- Use the following command to list all environments:  
*conda info --envs*
- If the environment *rbsvpath* is in the list, to activate the environment:  
*conda activate rbsvpath*
- If the environment *rbsvpath* is not in the list, use the command **in previous slide** to create it
- After create/activate the environment *rbsvpath*, start jupyter notebook and open: ***Robotic pathway planning***.
- After you complete it, deactivate current environment:  
*conda deactivate*

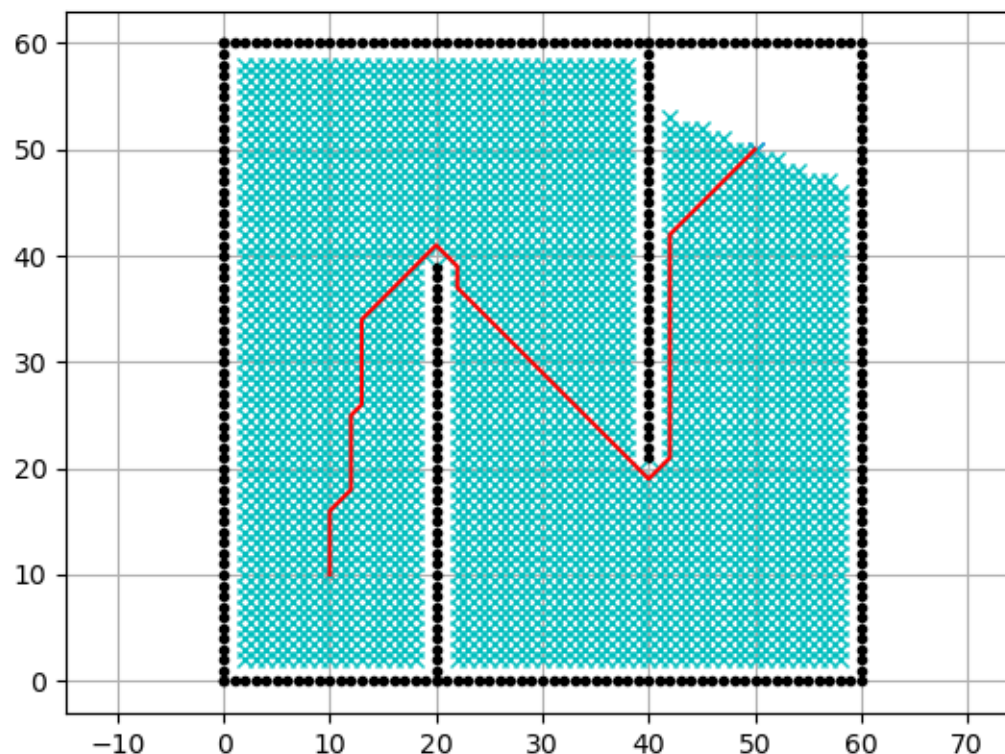
# Demo: Dijkstra's method



NUS  
National University  
of Singapore



- # start and goal position
- $sx = 10.0$
- $sy = 10.0$
- $gx = 50.0$
- $gy = 50.0$



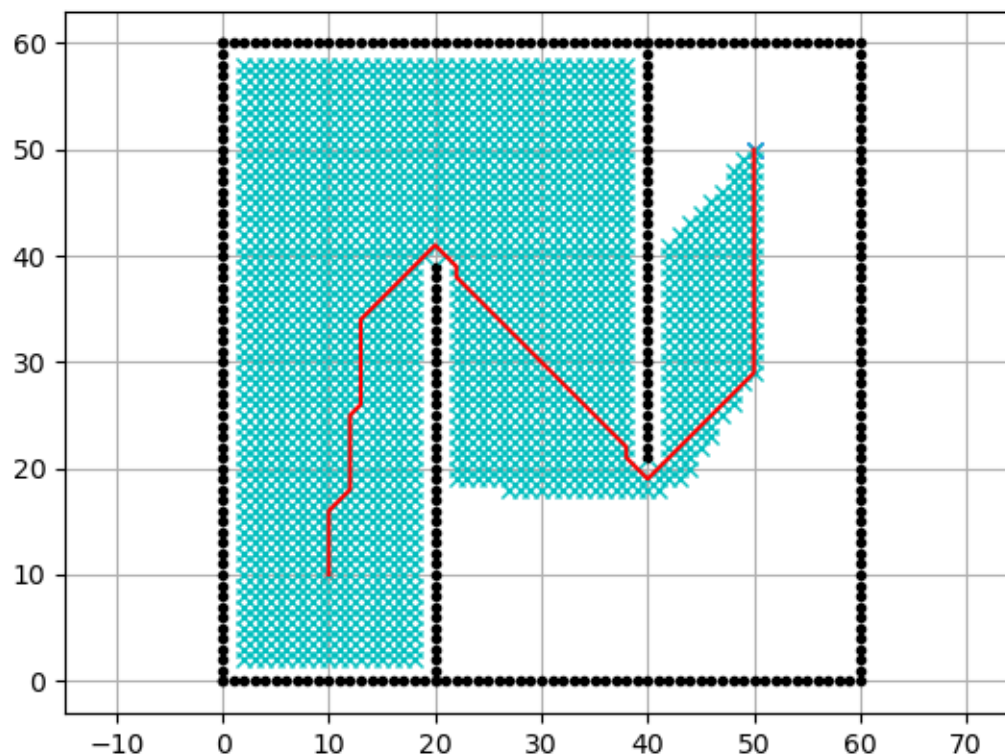
Reference: <https://github.com/AtsushiSakai/PythonRobotics>



# Demo: A\* method



- # start and goal position
- $sx = 10.0$
- $sy = 10.0$
- $gx = 50.0$
- $gy = 50.0$



Reference: <https://github.com/AtsushiSakai/PythonRobotics>

# Thank you😊

Dr Liu Fan

Email: [isslf@nus.edu.sg](mailto:isslf@nus.edu.sg)

# Extra slides





# Applications of Graphs



**NUS**  
National University  
of Singapore



- Navigation systems
- Web pages links
- Data transmission
- Social networks
- Fluid dynamics
- Critical path analysis/ Path planning
- 3D modelling
- Games and simulations

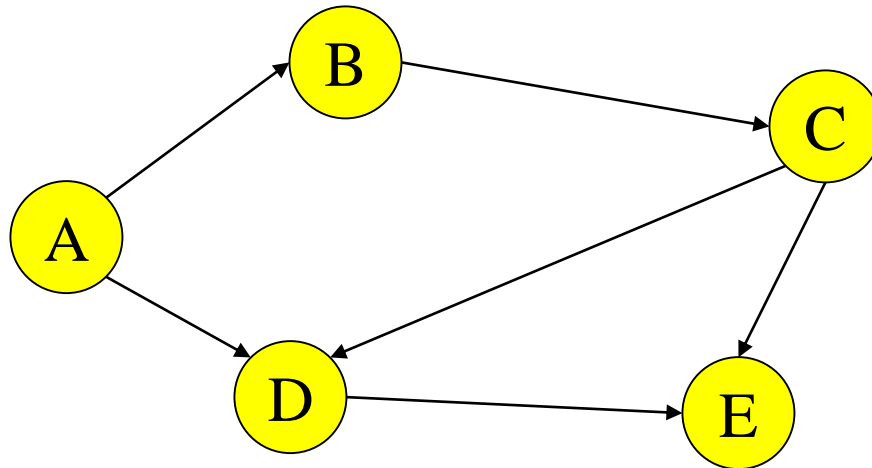


# Terminology of graph

- Vertex
- Edge
- Directed graph
- Undirected graph
- Weighted graph
- Path
- cycle



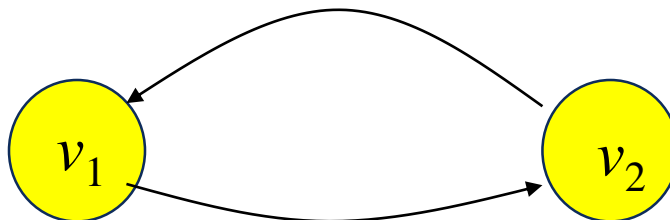
- A graph  $G$  is an ordered pair of a set  $V$  of vertices and a set  $E$  of edges:  $G=(V,E)$ .
- Where  $V$  is a set of nodes, also called vertices, and  $E$  is a set of edges, also called links.
- Example:
- $V = \{A, B, C, D, E\}$
- $E = \{(A,B),(A,D),(B,C),(C,D),(C,E),(D,E)\}$





# Directed vs Undirected Graphs

- If the order of edge pairs  $(v_1, v_2)$  matters, the graph is directed (also called a **digraph**):  $(v_1, v_2) \neq (v_2, v_1)$



- If the order of edge pairs  $(v_1, v_2)$  does not matter, the graph is called an undirected graph: in this case,  $(v_1, v_2) = (v_2, v_1)$

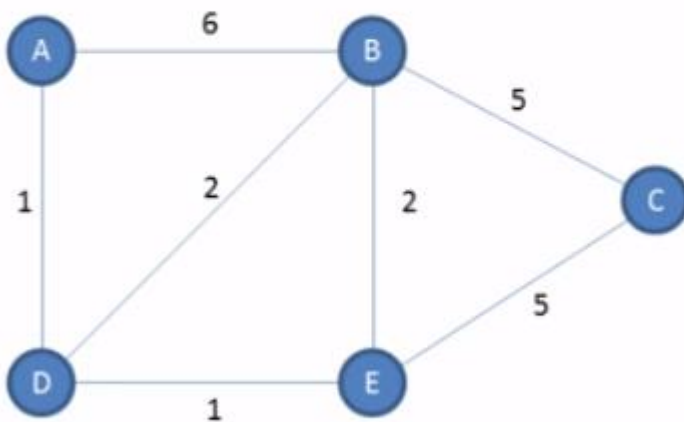




# Representations of graph

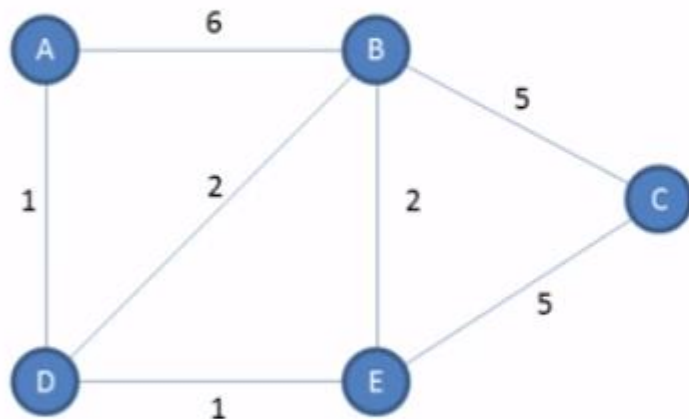
- Adjacent List
- Adjacency Matrix

## Adjacency List



Vertices	Each vertex's adjacency list
A	B D
B	A D E C
C	B E
D	A B E
E	D B C

## Adjacency Matrix



	0	1	2	3	4
	A	B	C	D	E
0	A		6		1
1	B	6		5	2
2	C		5		
3	D	1	2		
4	E		2	5	1

Dim Graph(4, 4) As Double

Graph(0, 1) = 6

Graph(0, 3) = 1

Graph(1, 0) = 6

Graph(1, 2) = 5

Graph(1, 3) = 2

Graph(1, 4) = 2

Graph(2, 1) = 5

Graph(2, 4) = 5

Graph(3, 0) = 1

Graph(3, 1) = 2

Graph(3, 4) = 1

Graph(4, 1) = 2

Graph(4, 2) = 5

Graph(4, 3) = 1



# Dijkstra's Algorithm



NUS  
National University  
of Singapore



- Let distance of start vertex from start vertex = 0
- Let distance of all other vertices from start = infinity
- While vertices remain unvisited
  - Visit unvisited vertex with smallest known distance from start vertex(call this current vertex)
  - For each unvisited neighbour of the current vertex
    - Calculate the distance from start vertex
    - If the calculate distance of this vertex is less than the known distance
      - Update shortest distance to this vertex
      - Update the previous vertex with the current vertex
    - End if
  - Next unvisited neighbour
  - Add the current vertex to the list of visited vertices
- End while



# A\* Algorithm



NUS  
National University  
of Singapore



```
Initialise open and closed lists
Make the start vertex current
Calculate heuristic distance of start vertex to destination
Calculate f value for start vertex( $f = g+h$ , where  $g = 0$ )
WHILE current vertex is not the destination
    FOR each vertex adjacent to current
        IF vertex not in closed list and not in open list THEN
            Add vertex to open list
            Calculate distance from start( $g$ )
            Calculate heuristic distance to destination ( $h$ )
            Calculate f value( $f = g+h$ )
            IF new f value < existing f value or there is no existing f value THEN
                Update f value
                Set parent to be the current vertex
            END IF
        END IF
    NEXT adjacent vertex
    Add current vertex to closed list
    Remove vertex with lowest f value from open list and make it current
END WHILE
```



# Exercise solution



Node	Status	Shortest distance from A	Previous NNode
A	Current visited	0	
B	Current Visited	$\infty$ , 1	A
C	Current Visited	<del><math>\infty</math></del> 2	A B
D		<del><math>\infty</math></del> 7	<del>B</del> E
E	Current Visited	<del><math>\infty</math></del> 5	<del>B</del> C
F	current	<del><math>\infty</math></del> 7	A E
G		$\infty$ 3	B

Shortest path from A to F: A -> B -> C->E->F, total cost: 7