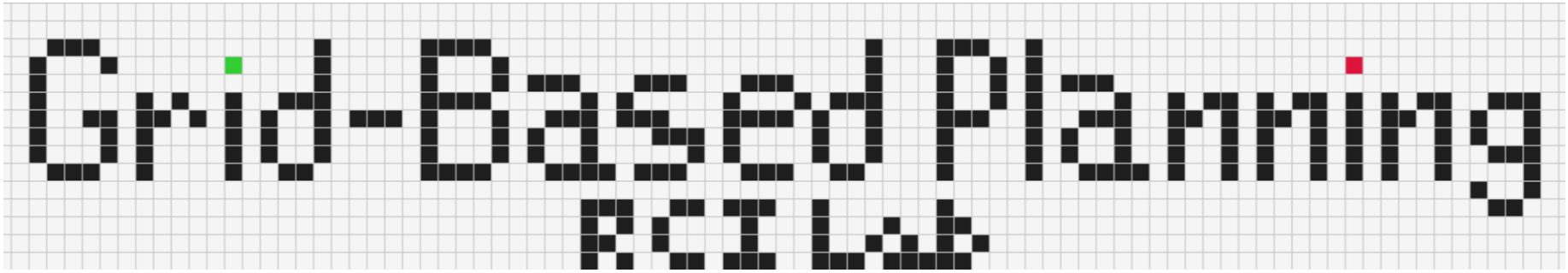
Introduction of Grid-Based Planning



Speaker: Taehyun Jung

RCI LAB

KHU

Table of Contents

01 What is the Grid?

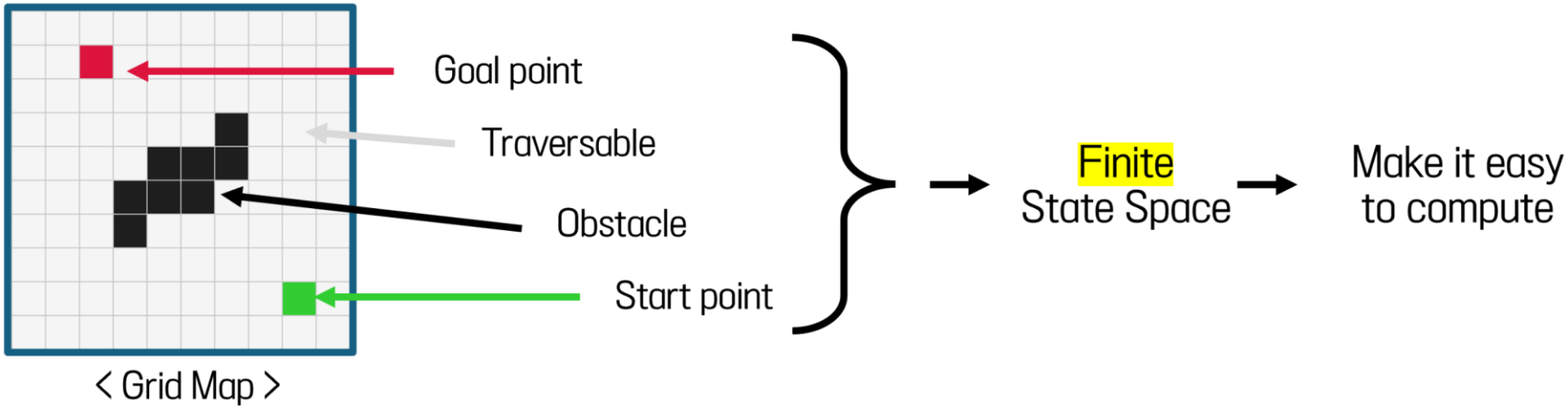
01 What is the Grid?

Grid

A model that simplifies a complex continuous space into a set of uniform cells that are easy for a computer to understand.

Key Features of Grid

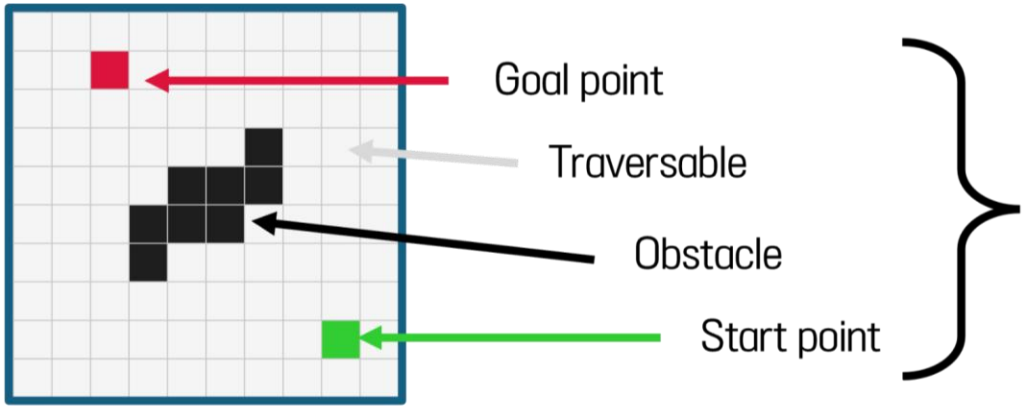
- **Discrete Space**: Represents the world as a set of distinct cells.
- **Simplicity**: Each cell holds simple information, such as 'traversable' or 'obstacle'.
- **Coordinates**: Every cell has a unique address, like (x, y).



< Grid Map >

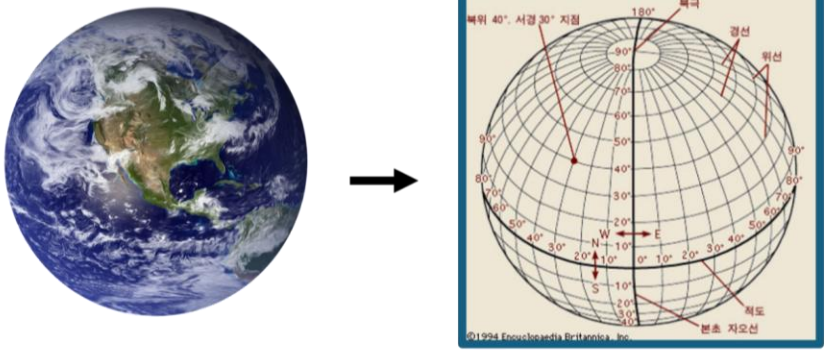
01 What is the Grid?

- Simplicity: Each cell holds simple information, such as 'traversable' or 'obstacle'.
- Coordinates: Every cell has a unique address, like (x, y).



< Grid Map >

Finite State Space → Make it easy to compute



< Finite State Space of Earth >

I'm located at 37.5665 degrees north latitude and 126.9780 degrees east longitude.

01 What is the Grid?

How many meters per cell?

Resolution (r)

- Definition: how much real-world distance a single grid cell represents.
- Unit: meters per cell [m/cell]

※ There can be another length scale(or unit) instead of meters

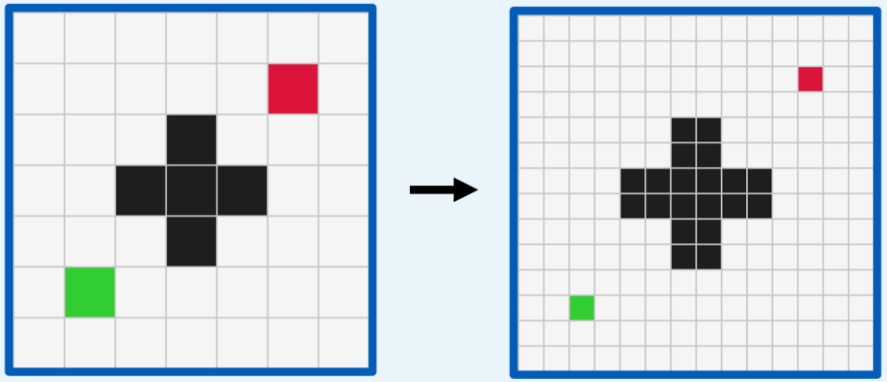
- World (x, y) → Grid (i, j), total counts of cells N

$$i = \left\lfloor \frac{y - y_{\min}}{r} \right\rfloor, \quad j = \left\lfloor \frac{x - x_{\min}}{r} \right\rfloor, \quad N \approx \frac{L_x L_y}{r^2} \quad \begin{cases} L_x : \text{length of width} \\ L_y : \text{length of height} \end{cases}$$

- It directly controls **detail, computation, and memory.**

< Low Resolution >

- Fewer cells
- Detail ↓
- Computation ↓
- Memory ↓



< High Resolution >

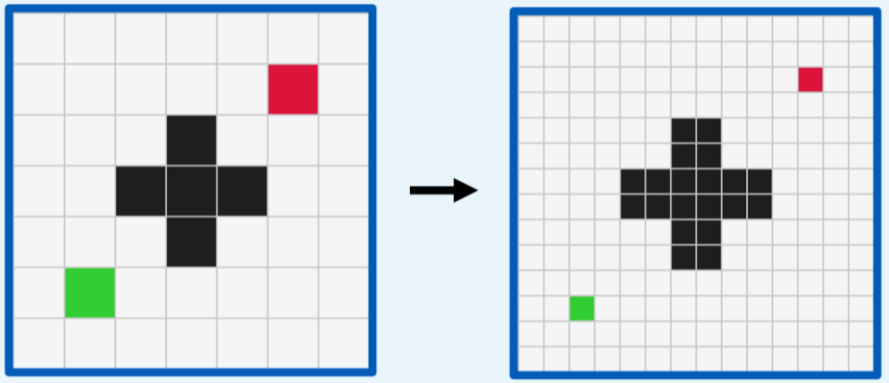
- More cells
- Detail ↑
- Computation ↑
- Memory ↑

01 What is the Grid?

How many meters per cell?

- It directly controls **detail, computation, and memory.**

< Low Resolution >
- Fewer cells
- Detail ↓
- Computation ↓
- Memory ↓



< High Resolution >
- More cells
- Detail ↑
- Computation ↑
- Memory ↑

Example)

- Map: Width(L_x) = 1m, Height(L_y): 1m
- Resolution: r_{low} = 1/7 [m/cell], r_{high} = 1/14 [m/cell]

$$\rightarrow \begin{cases} N_{\text{low}} = 49 \text{ cells} \\ N_{\text{high}} = 196 \text{ cells} \end{cases}$$

- The computational time complexity of Dijkstra/A* in the worst case: $O(N \log N)$

02 Occupancy Grid

02 Occupancy Grid

Occupancy Grid

A map that encodes whether each grid cell is occupied by an obstacle.

Representation

State	Integer Convention (ROS)	Probabilistic	Meaning
Free space	0	Low p_{occ}	Traversable
Occupied space	100	High p_{occ}	Non-traversable (collision)
Unknown space	-1	Undetermined p_{occ}	Unobserved or Uncertain area

- For probabilistic maps, choose a threshold τ, declare occupied if $p_{occ} \geq \tau$.
- We should choice the **policy** for unknown area :
  1. **Conservative**: treat as blocked or assign very high cost (avoid exploration into unknown).
  2. **Aggressive**: treat as free (faster paths but higher collision risk).
  3. **Compromise**: assign medium-to-high cost and let the planner decide.

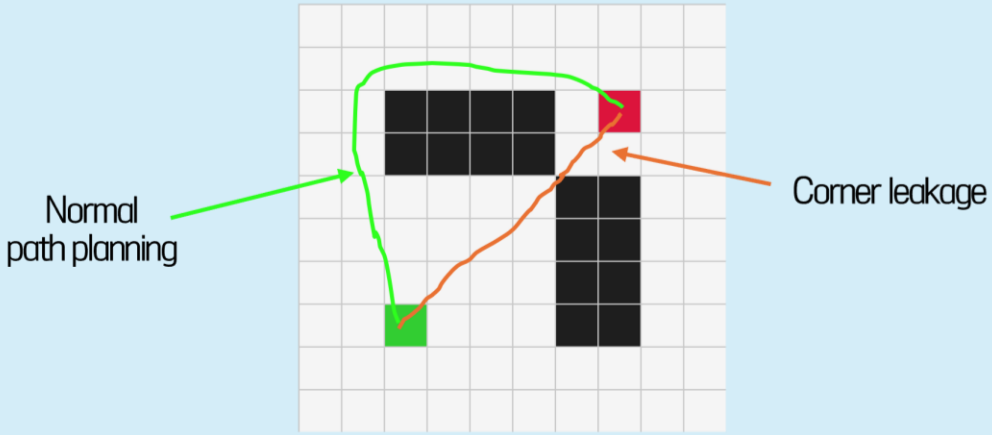
02 Occupancy Grid

Occupancy Grid

A map that encodes whether each grid cell is occupied by an obstacle.

Limitations

- Occpancy grid only represent whether robot can pass.
- Corner leakage on diagonals:



→ Solution: Combine occupancy with connectivity rules and inflation.

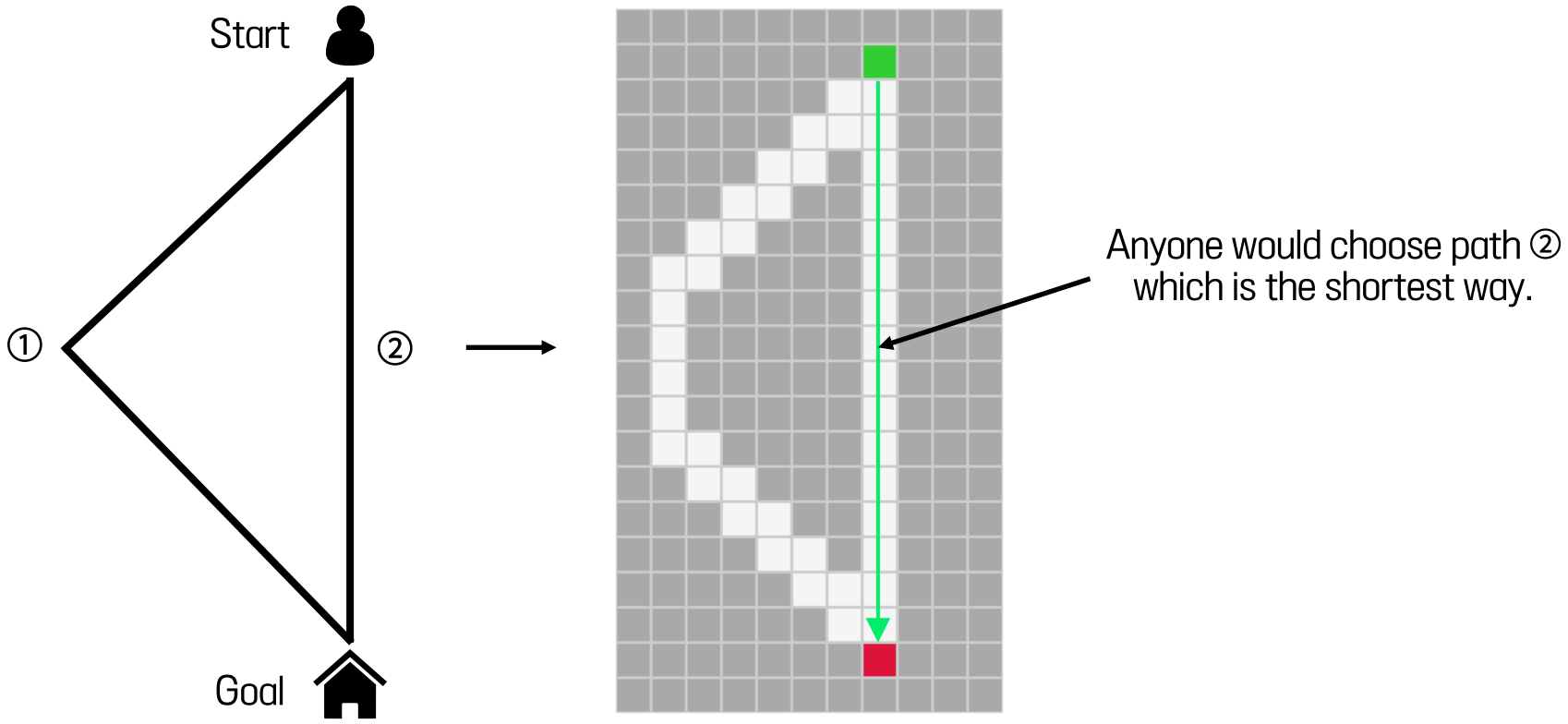
03 Costmap

03 Costmap

Cost

A scalar score for undesirability.

Smaller is better.

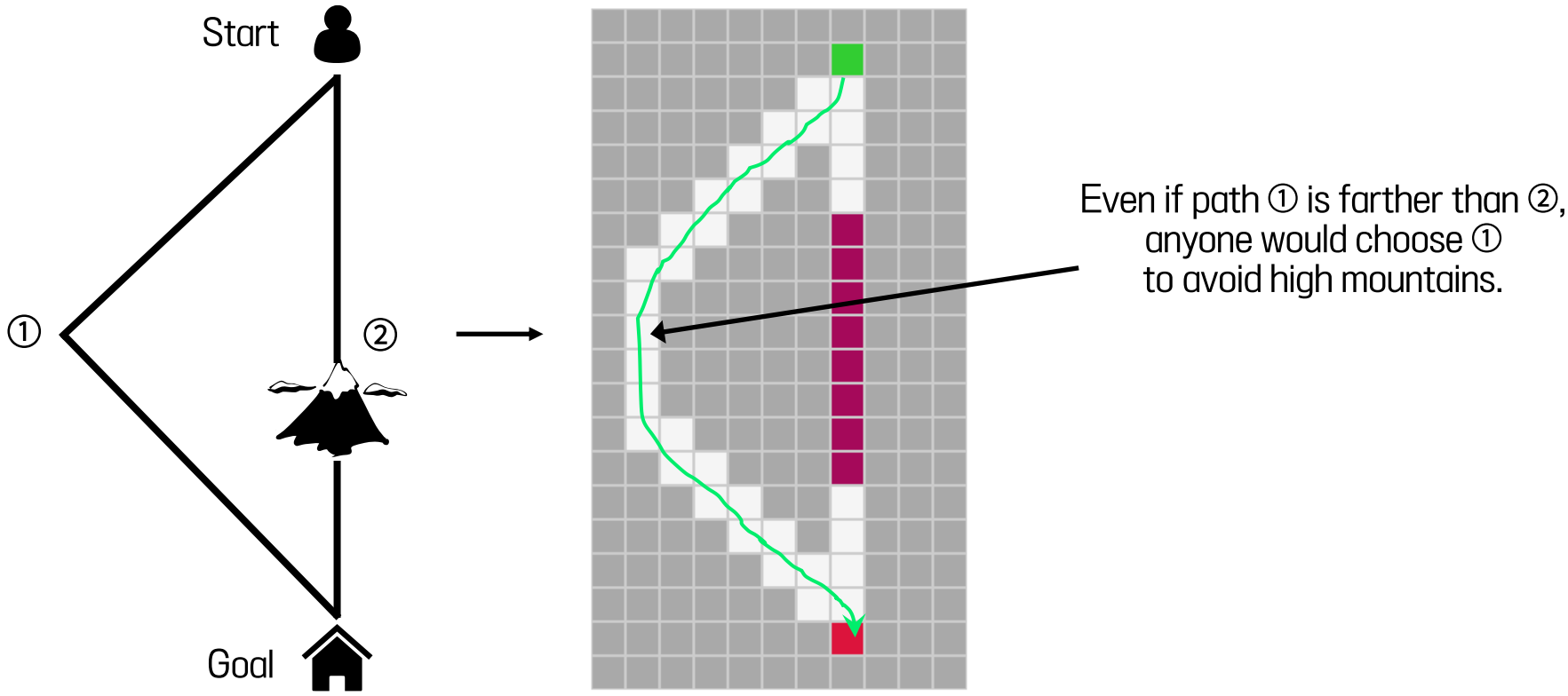


03 Costmap

Cost

A scalar score for undesirability.

Smaller is better.

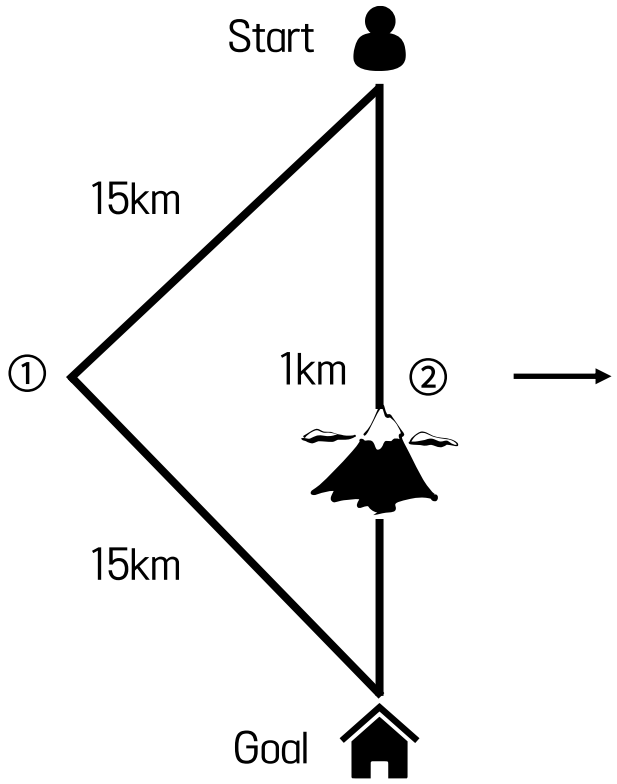


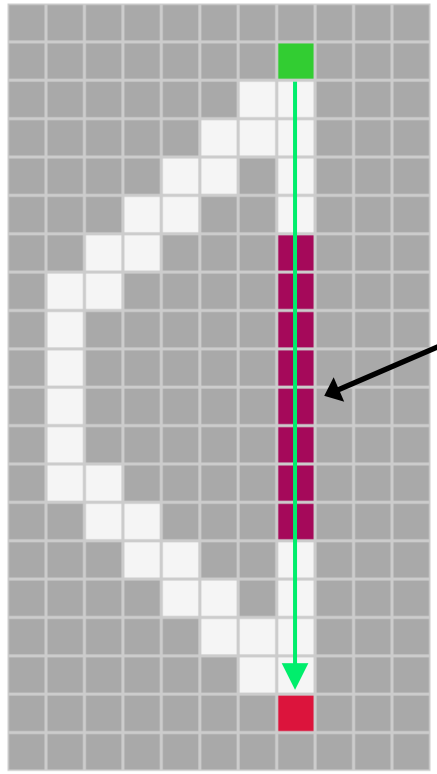
03 Costmap

Cost

A scalar score for undesirability.

Smaller is better.





If path ① is too far,
it may be better to climb the mountains.

03 Costmap

Cost

$$\text{PathCost} = \sum_{(u \to v \in \text{path})} \text{Cost}(u \to v)$$

$$\text{where} \quad \text{Cost}(u \to v) = \underbrace{\text{move\_cost}(u, v)}_{\text{Length}} + \beta \cdot \underbrace{\text{sample}(c(u), c(v))}_{\text{Safety}}$$

Trade-off between length and safety

※ Reference of the distance

	Manhattan	Octile	Chebyshev
Definition	$\lvert x_1 - x_2 \rvert + \lvert y_1 - y_2 \rvert$	$\sqrt{2} \times \min(\Delta x, \Delta y) + \lvert \Delta x - \Delta y \rvert$	$\min(\lvert \Delta x_1 - \Delta x_2 \rvert, \lvert \Delta y_1 - \Delta y_2 \rvert)$
Characteristic	Used when robots can only travel in certain directions	Particularly useful for path planning of diagonal-moving robots in grid based planning	Suitable when assuming diagonal travel costs equal to straight travel costs

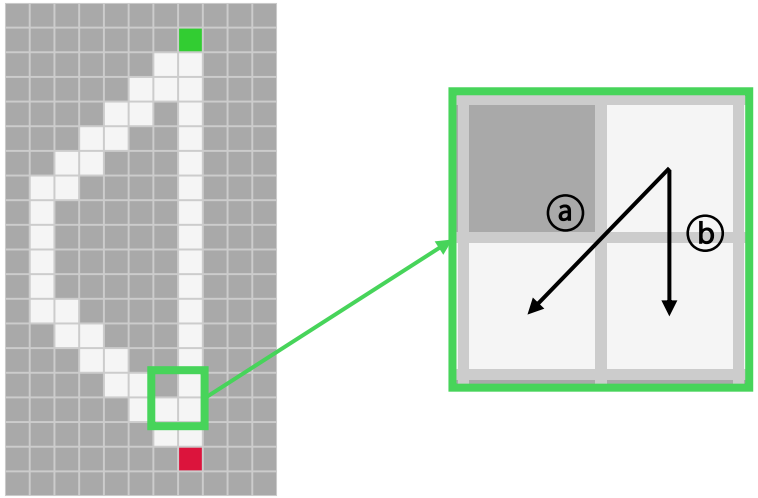
03 Costmap

Cost

$$\text{PathCost} = \sum_{(u \to v \in \text{path})} \text{Cost}(u \to v)$$

$$\text{where} \quad \text{Cost}(u \to v) = \underbrace{\text{move\_cost}(u, v)}_{\text{Length}} + \beta \cdot \underbrace{\text{sample}(c(u), c(v))}_{\text{Safety}}$$

Trade-off between length and safety



In this tutorial,
Manhattan & Octile distance will be used.

ⓐ Diagonal : $\text{move\_cost}(u, v) = \sqrt{2}$

ⓑ Straight : $\text{move\_cost}(u, v) = 1$

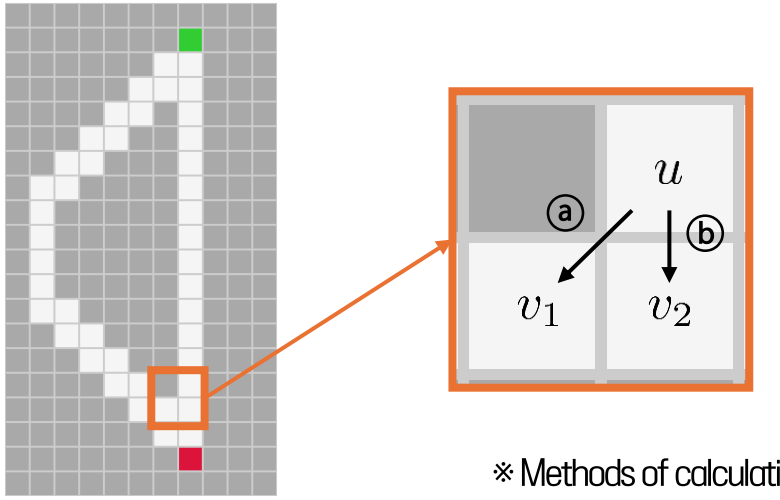
03 Costmap

Cost

$$\text{PathCost} = \sum_{(u \to v \in \text{path})} \text{Cost}(u \to v)$$

$$\text{where} \quad \text{Cost}(u \to v) = \underbrace{\text{move\_cost}(u, v)}_{\text{Length}} + \beta \cdot \underbrace{\text{sample}(c(u), c(v))}_{\text{Safety}}$$

Trade-off between length and safety



ⓐ : $\text{sample}(c(u), c(v_1)) = \dfrac{c(u) + c(v_1)}{2}$

ⓑ : $\text{sample}(c(u), c(v_2)) = \dfrac{c(u) + c(v_2)}{2}$

※ Methods of calculating costs are determined by trade-offs such as speed, safety, and complexity.

03 Costmap

Cost Convention in ROS

Cost uses an 8-bit integer(0 ~ 255) scale in ROS Navigation stack(ROS1 costmap_2d, ROS2 Nav2).

Value	Meaning	Note
0	Free space	Traversable
1 ~ 252	Inflation/Potential	The closer to the obstacle, the larger cost. (Detailed values are determined by Inflation layer.)
253	Inscribed inflated obstacle	Substantial collision range. Treat it an evasive/prohibitive level when planning.
254	Lethal obstacle	Actual obstacle core. Treat it a prohibitive level when planning.
255	No information (Unknown)	Unobserved/uncertain areas. According to policy, it is blocked or set high cost.

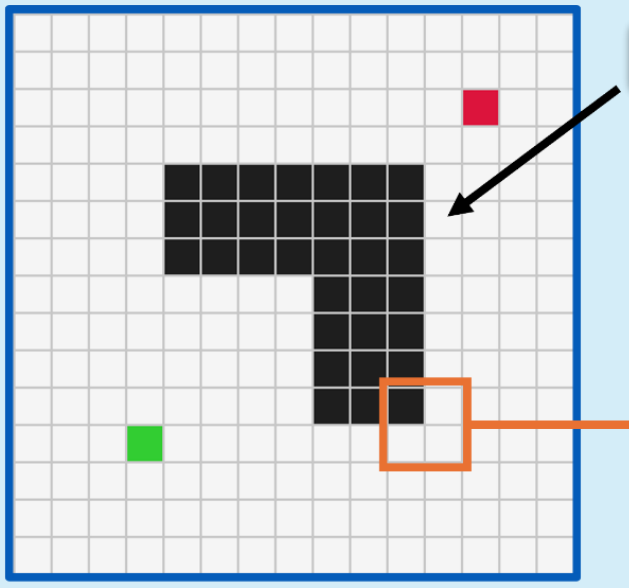
03 Costmap

Costmap

A scalar cost field built from occupancy (and optionally other layers).

Occupancy grid tells only can/can't pass.
Costmap adds how preferable each traversable cell is, combining shortness vs. safety.

Components of costmap



Obstacle core

Occupied cells are non-traversable and should receive a very large cost C_{obs} in your chosen scale.

254	0
0	0

ex) 255 scale: fixed to 254

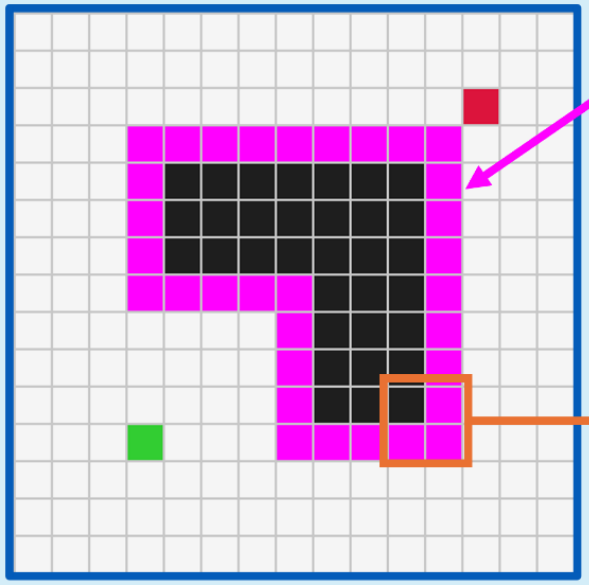
03 Costmap

Costmap

A scalar cost field built from occupancy (and optionally other layers).

Occupancy grid tells only can/can't pass.
Costmap adds how preferable each traversable cell is, combining shortness vs. safety.

Components of costmap



Inflation ring

Around each obstacle, assign a distance-dependent penalty.

The closer a free cell is to an obstacle, the higher its cost.

This prevents corner-cutting paths that graze obstacles.

254	200
200	180

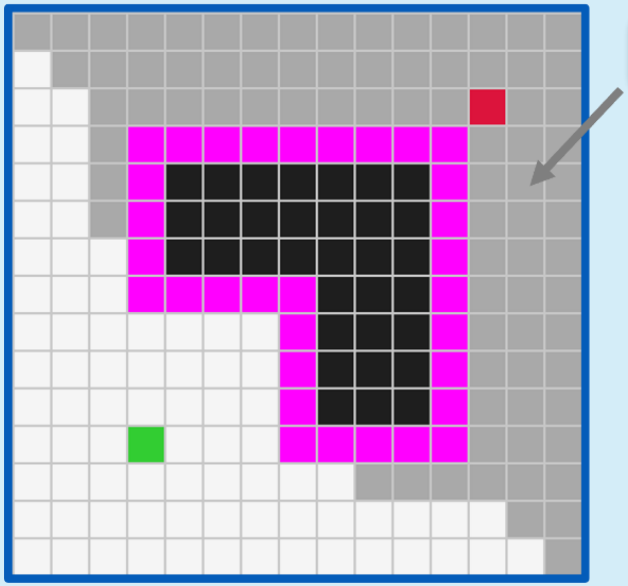
03 Costmap

Costmap

A scalar cost field built from occupancy (and optionally other layers).

Occupancy grid tells only can/can't pass.
Costmap adds how preferable each traversable cell is, combining shortness vs. safety.

Components of costmap



Unknown

If you decided not to hard-block Unknown in the occupancy layer, you can encode caution here by assigning a high but finite cost to unknown cells.

This keeps exploration possible yet discouraged compared to well-observed free space.

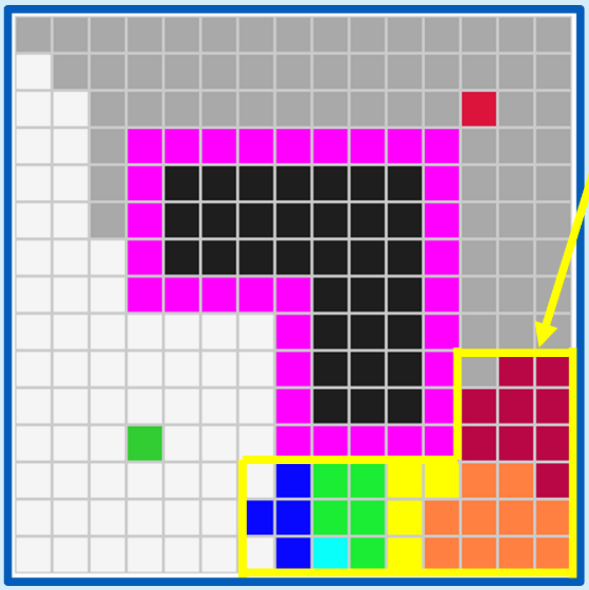
03 Costmap

Costmap

A scalar cost field built from occupancy (and optionally other layers).

Occupancy grid tells only can/can't pass.
Costmap adds how preferable each traversable cell is, combining shortness vs. safety.

Components of costmap



Additional layers

Real systems may add layers for slope, sound, radiation, etc., then combine layer via a weighted sum.

For this tutorial, stick to a clean baseline:
obstacle core + inflation (+ optional unknown)

04 Inflation

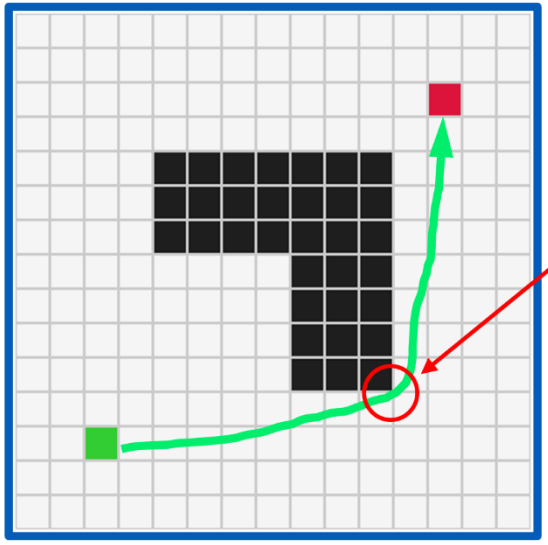
04 Inflation

Why do we need inflation?

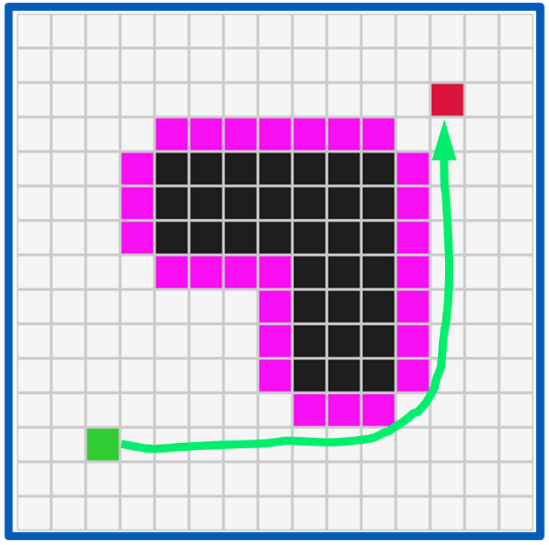
Occupancy grid tells only can/can't pass.

Real routes must also consider how far robots keep from obstacles.

Inflation creates a high-cost halo around obstacles so that a planner naturally maintains clearance and reduces unrealistic corner-cutting using cost only.



< without inflation >

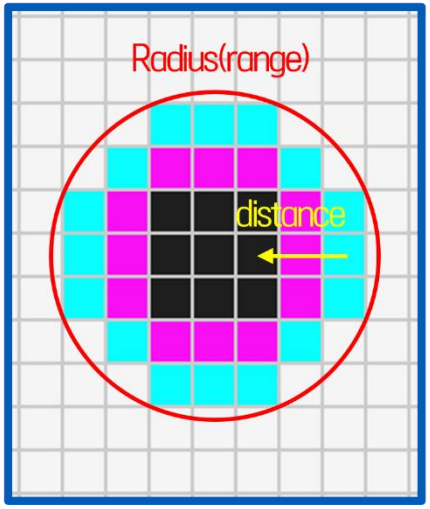


< inflation applied >

04 Inflation

Core Intuitions of Inflation

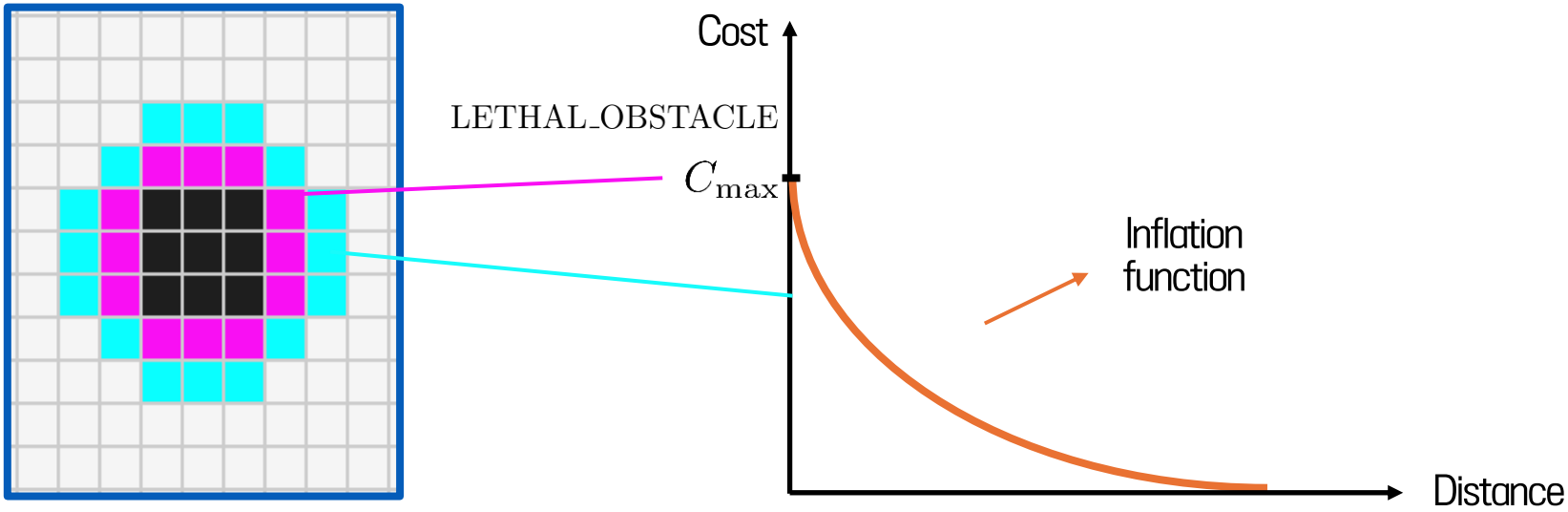
- Distance-based cost
  → Each cell's cost depends on its nearest-obstacle distance d.
- Closer → Higher cost; Farther → Lower cost
  ($d = 0$ marks the non-traversable core.)
- Radius R : the effective range of inflation. → Choose from robot size + margin.
  → For $d > R$, set cost to zero(or very small) so far-away cells remain free.



04 Inflation

Mathematical Models for Inflation Costs

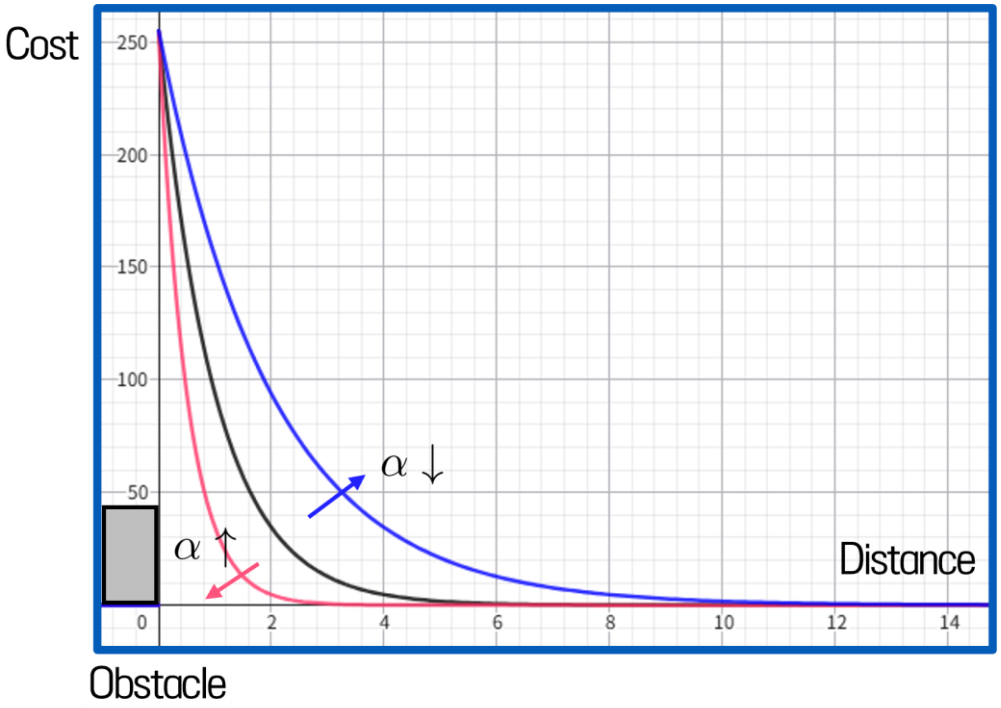
- Design the inflation function to match your intent. → Not a rule!
- Shape the inflation curve to encode your desired clearance and detour preference.
- The essentials are higher cost near obstacles, lower cost father away. → depending on d



04 Inflation

Example Models – Exponential Decay

$$C_{\text{infl}}(d) = \begin{cases} \text{Obstacle core} & d = 0 \\ C_{\max} e^{-\alpha d} & 0 < d \leq R \\ 0 & d > R \end{cases}$$

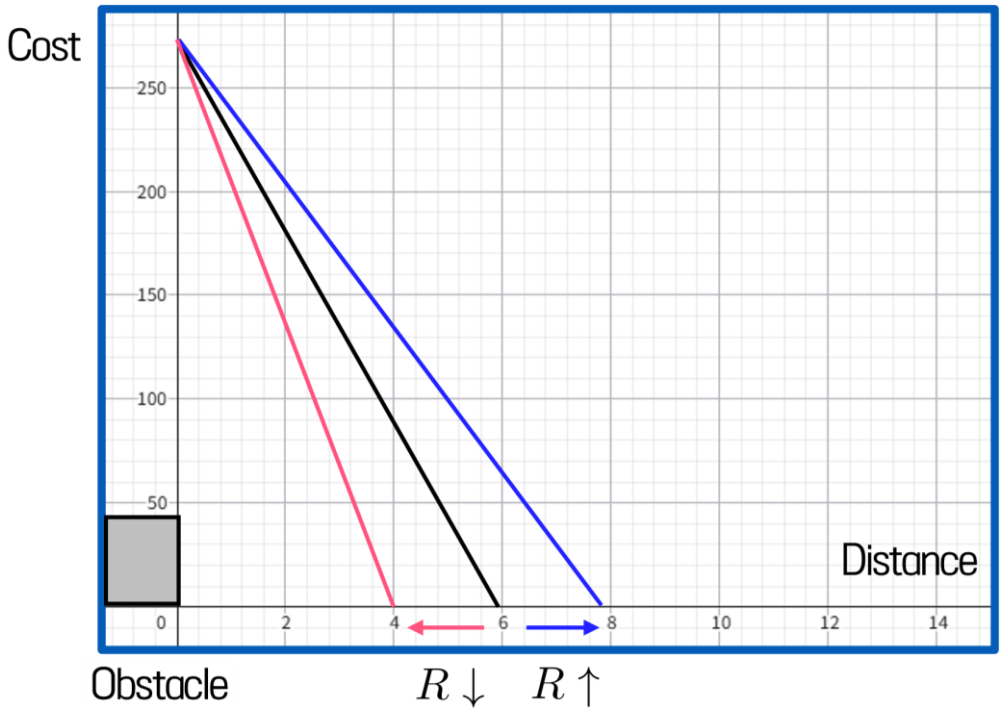


- Strong near the core → drops quickly just a bit away.
- Tuning Parameter α : strong near-core penalty, fast falloff.
- Exponential method is mainly used.

04 Inflation

Example Models – Linear Lamp

$$C_{\text{infl}}(d) = C_{\max} \max(0,\, 1 - \frac{d}{R})$$



- Linear decrease → very intuitive and predictable.

04 Inflation

Application of Inflation in a Real ROS/Nav2 Environment

Location: /opt/ros/humble/include/nav2_costmap_2d/nav2_costmap_2d/inflation_layer.hpp

```
144
145  /** @brief  Given a distance, compute a cost.
146   * @param  distance The distance from an obstacle in cells
147   * @return A cost value for the distance */
148  inline unsigned char computeCost(double distance) const
149  {
150    unsigned char cost = 0;
151    if (distance == 0) {
152      cost = LETHAL_OBSTACLE;
153    } else if (distance * resolution_ <= inscribed_radius_) {
154      cost = INSCRIBED_INFLATED_OBSTACLE;
155    } else {
156      // make sure cost falls off by Euclidean distance
157      double factor =
158        exp(-1.0 * cost_scaling_factor_ * (distance * resolution_ - inscribed_radius_));
159      cost = static_cast<unsigned char>((INSCRIBED_INFLATED_OBSTACLE - 1) * factor);
160    }
161    return cost;
162  }
```

254

253

d

α

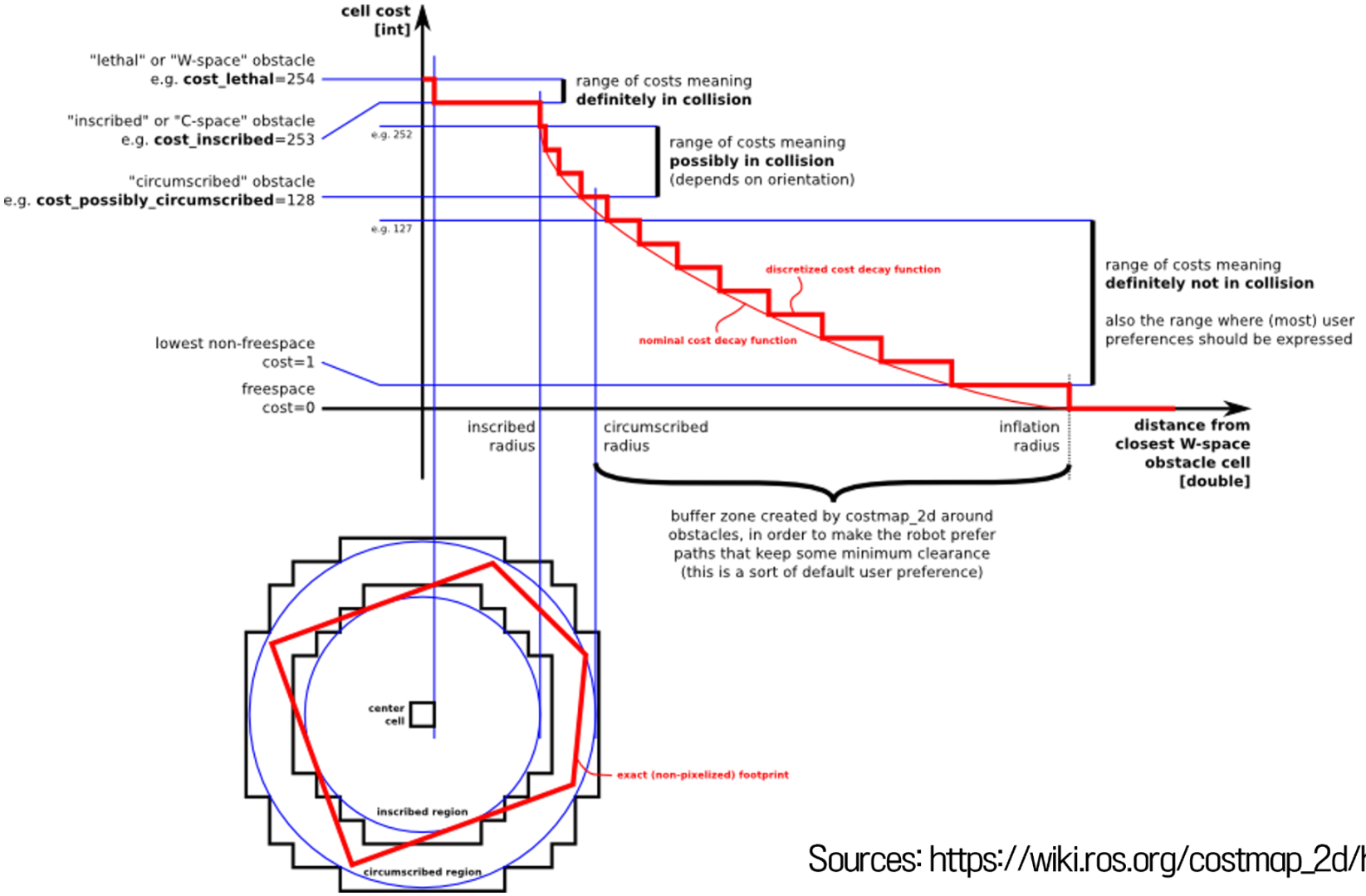
$C_{\max}(= 252)$

$$C_{\text{infl}}(d) = \begin{cases} \text{Obstacle core} & d = 0 \\ C_{\max} e^{-\alpha d} & 0 < d \leq R \\ 0 & d > R \end{cases}$$

< Exponential Decay >

04 Inflation

- Application of Inflation in a Real ROS/Nav2 Environment



Sources: https://wiki.ros.org/costmap_2d/hydro/inflation

05 Connectivity

05 Connectivity

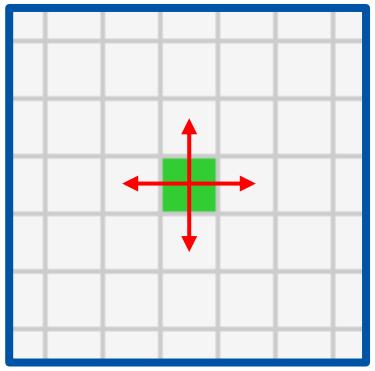
What does connectivity mean in path planning?

Connectivity defines which neighbor cells are allowed to move to from a given cell.

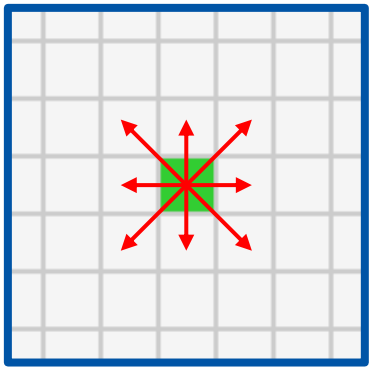
Standard Choices of Connectivity

- 4 – connected (Von Neumann) : Up, Down, Left, Right → Manhattan
- 8 – connected (Moore) : 4-connected + 4 diagonals → Octile

※ With 8-connected moves, paths can be shorter and straighter. However, it can be more expensive of more branching.



< 4 - connected >



< 8 - connected >

$$C_{\text{step}} = \begin{cases} 1 & \text{cardinal moves} \\ \sqrt{2} & \text{diagonals (if } 8-\text{connected)} \end{cases}$$

< per-step cost >

05 Connectivity

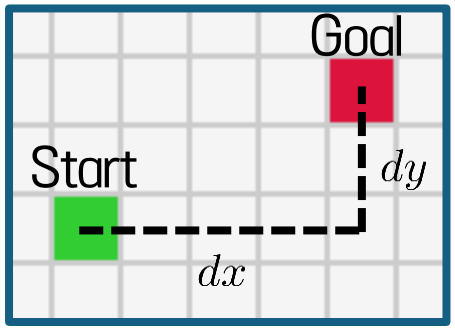
Heuristic Alignment (A* Preview)

- 4-connected → Manhattan Distance (admissible/consistent)

$$h = |dx| + |dy|$$

- 8-connected → Octile Distance ($\text{with straight} = 1,\ \text{diagonal} = \sqrt{2}$)

$$h = (|dx| + |dy|) + (\sqrt{2} - 2) \cdot \min(|dx|, |dy|)$$



05 Connectivity

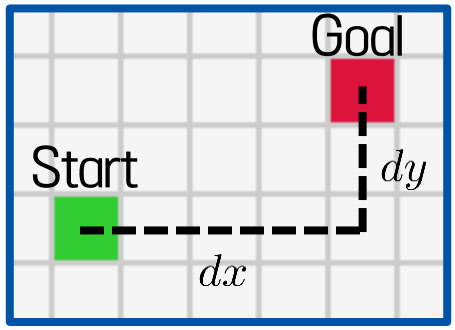
Heuristic Alignment (A* Preview)

- 4-connected → Manhattan Distance (admissible/consistent)

$$h = |dx| + |dy|$$

- 8-connected → Octile Distance (with straight $= 1$, diagonal $= \sqrt{2}$)

$$h = (|dx| + |dy|) + (\sqrt{2} - 2) \cdot \min(|dx|, |dy|)$$



$$dx = 4$$
$$dy = 2$$

< 4-connected >

$$\begin{aligned} h &= |dx| + |dy| \\ &= |4| + |2| \\ &= 6 \end{aligned}$$

05 Connectivity

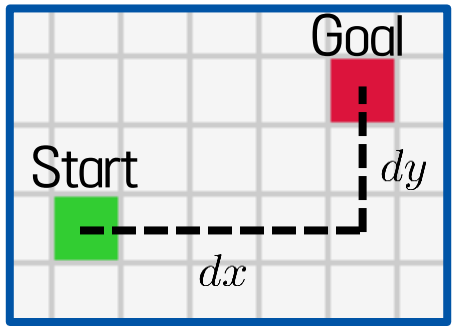
Heuristic Alignment (A* Preview)

- 4-connected → Manhattan Distance (admissible/consistent)

$$h = |dx| + |dy|$$

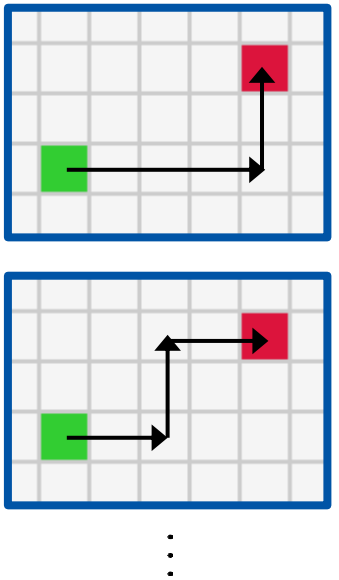
- 8-connected → Octile Distance (with straight $= 1$, diagonal $= \sqrt{2}$)

$$h = (|dx| + |dy|) + (\sqrt{2} - 2) \cdot \min(|dx|, |dy|)$$



< 4-connected >

$$h = |dx| + |dy|$$
$$= |4| + |2|$$
$$= 6$$



05 Connectivity

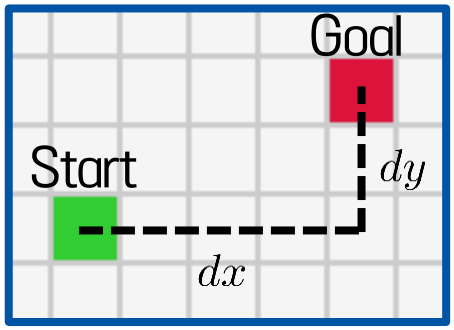
Heuristic Alignment (A* Preview)

- 4-connected → Manhattan Distance (admissible/consistent)

$$h = |dx| + |dy|$$

- 8-connected → Octile Distance ($\text{with straight} = 1,\ \text{diagonal} = \sqrt{2}$)

$$h = (|dx| + |dy|) + (\sqrt{2} - 2) \cdot \min(|dx|, |dy|)$$



$$dx = 4$$
$$dy = 2$$

< 8-connected >

$$\begin{aligned} h &= (|dx| + |dy|) + (\sqrt{2} - 2) \cdot \min(|dx|,\ |dy|) \\ &= (|4| + |2|) + (\sqrt{2} - 2) \cdot |2| \\ &= 2 + 2\sqrt{2} \end{aligned}$$

05 Connectivity

Heuristic Alignment (A* Preview)

- 4-connected → Manhattan Distance (admissible/consistent)

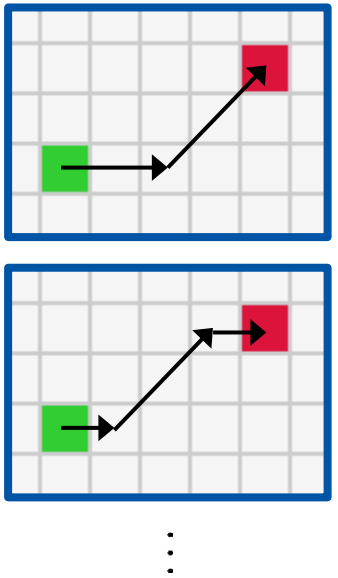
$$h = |dx| + |dy|$$

- 8-connected → Octile Distance ($\text{with straight} = 1,\ \text{diagonal} = \sqrt{2}$)

$$h = (|dx| + |dy|) + (\sqrt{2} - 2) \cdot \min(|dx|, |dy|)$$

< 8-connected >

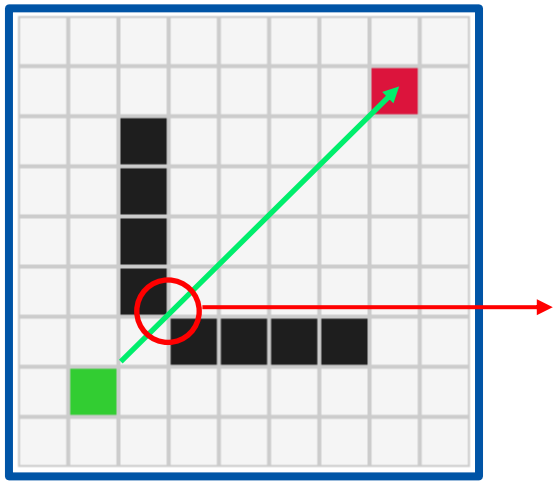
$$\begin{aligned} h &= (|dx| + |dy|) + (\sqrt{2} - 2) \cdot \min(|dx|, |dy|) \\ &= (|4| + |2|) + (\sqrt{2} - 2) \cdot |2| \\ &= 2 + 2\sqrt{2} \end{aligned}$$



05 Connectivity

Diagonal Safety Rule (Anti Corner-Cutting)

Issue: With 8-connected grids, a path may squeeze between two obstacle corners via a diagonal.



There is no problem theoretically,
but in real world, it is wrong path.

Solution(Rule) :

→ Allow a diagonal $(i, j) \to (i \pm 1, j \pm 1)$ only if both adjacent cardinal cells $(i, j \pm 1), (i \pm 1, j)$ are free.

Effects :

- It robustly prevents corner-cutting.
- It works best together with inflation.

05 Connectivity

Comparison

- 4-connected vs. 8-connected

	4-connected	8-connected
Branching	Lower (often faster)	Higher (more list expansions, computation ↑)
Path Length & Straightness	Can be longer	Often shorter & straighter
Corner-cut Risk	Low	High (mitigate with safety rule)
Heuristic	Manhattan	Octile
Implementation	Simple	Moderate (add safety rule)