Introduction to Robotics

Felipe P. Vista IV

Grading

➢ Attendance 5%

Name (Original Name)	User Email	Join Time	Leave Time	Duration (Minutes)
[redacted]		4/12/2021 9:12	4/12/2021 10:14	62
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:13	4/12/2021 9:13	1
		4/12/2021 9:13	4/12/2021 9:14	2
		4/12/2021 9:14	4/12/2021 9:14	1
		4/12/2021 9:14	4/12/2021 9:14	1
		4/12/2021 9:14	4/12/2021 10:14	60

Bad ZOOM User Name (Absent)

➢ Iphone → Not your name
➢ SiAko 202100001 → Wrong order
➢ SiAko → Name only
➢ 202100001 → ID Num only

ZOOM User Name (Present)

➢ University ID Num_Name
➢ 202100001 SiAko → GOOD (Present)

Name (Original Name)	User Email	Total Duration (Minutes)
[redacted]		62
		63
		62
		62
		63
		62
		63

Student Responsibilities

- Download/Install ZOOM app for online lecture
  - *Zoom profile must be your **OASIS ID+name** similar to OASIS*
  - *Ex.: **202061234 YourName***
  - *If you are asked, but no reply, then you'll be out of zoom & mark **absent***
- Regularly login, check OLD IEILMS for updates, notifications
  - *https://ieilmsold.jbnu.ac.kr*
  - *Presentations & lecture videos will be uploaded after class*
- Regularly check Kakao Group Chat for class
  - *Everybody must have a Kakao talk account*
  - *Search & add account "**botjok**", introduce yourself and name of class ("Robotics"), then you will be added to the group chat*

Intro To Robotics

MAPPING

Intro

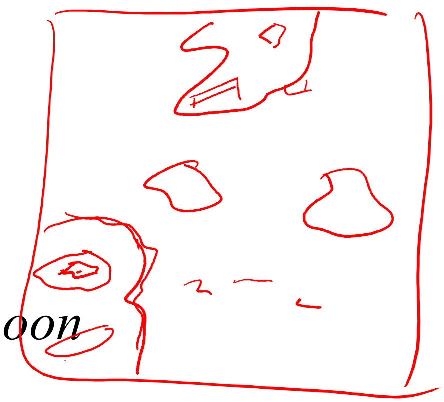
- Robot can localize itself by detecting obstacles
  - *Using obstacle position or other environmental information*
  - *Information normally provided by a map*
- Maps for industrial environments
  - *Relatively easy create (i.e. factories)*
  - *Machines are anchored (fixed)*
- Maps for robotic vacuum cleaner
  - *Less relevant since manufacturer cannot prepare map for each client*
  - *Customers construct map of their apartments (then update if move things)*
- Maps for inaccessible places
  - *Impossible to construct in advance these maps in advance*
  - *i.e. ocean floor*

Intro

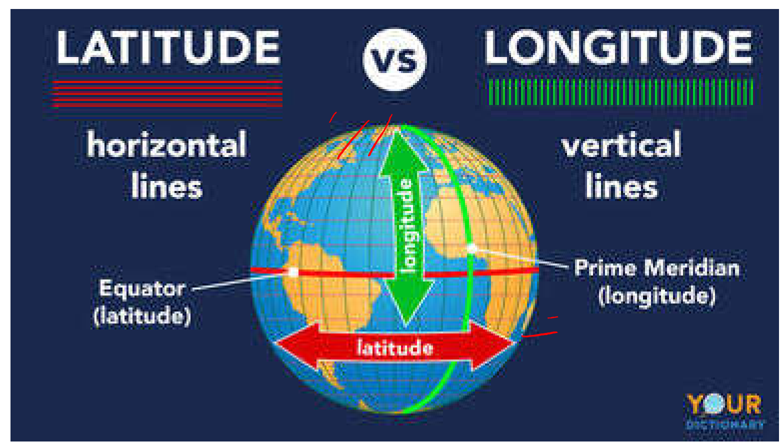
- Solution for robot?
  - *Build own map of the environment*
  - *Building a map requires localization to know where it is*
  - *But localization itself needs a map….. which in turn needs…*
  - *Which gives us a chicken-and-egg problem*
  - *So, how do we solve this?*
- SLAM algorithms
  - *By using* ***simultaneous localization and mapping***
  - *Using valid information even in unexplored parts of the environment*
  - *Refining information during exploration*

https://i.pinimg.com/564x/14/61/bd/1461bd67250fce22fdd1209c492b5534.jpg

SLAM



- What humans used to create geographical maps
- Observation of sun & stars for localization
  - *Get latitude using sextant by measuring sun height at noon*
  - *Accurate measurement of longitude impossible*
  - *Until chronometers (accurate clock) developed*
- As localization improved
  - *Maps also improved*
  - *Not only land & seacoasts but also terrain features*
  - *Such as lakes, forests and mountains*
  - *Also artificial structures like buildings and roads*



https://assets.ltkcontent.com/images/92435/longitude-versus-latitude-earth_0066f46bde.jpg

Mapping

- Discrete & Continuous Maps
- The Content of the Cells of a Grid Map
- Creating a Map by Exploration : Frontier Algorithm
- Mapping Using Knowledge of the Environment
- Numerical Example for a SLAM Algorithm
- Formalization of the SLAM Algorithm

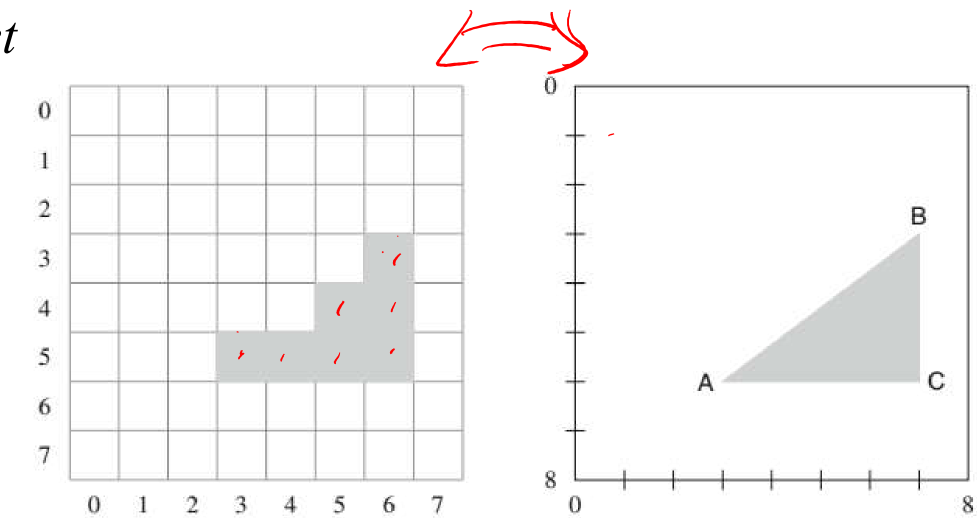
Discrete & Continuous Maps

- Graphical maps
  - *What we're used to*
  - *Printed on papers (before)*
  - *Displayed on computers & smartphones (currently)*
- Non-visual representation
  - *What a robot needs*
  - *Can be stored in memory*
- Techniques for storing maps
  1. *Discrete maps (grid maps)*
  2. *Continuous maps*

Discrete & Continuous Maps

- Discrete map
  - *8 x 8 grid with triangular object*
  - *Object location stored as list of coordinates of each cell covered by object*
  - *Object consists of :*

    $(5,3),(5,4),(5,5),(4,5),(5,6),(4,6),(3,6)$



(a) Discrete map **(b) Continuous map**

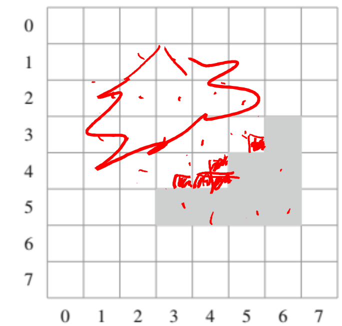
Map of occupied cells of the same object

- Continuous map
  - *Coordinates of boundary positions are stored instead of positions of the object*

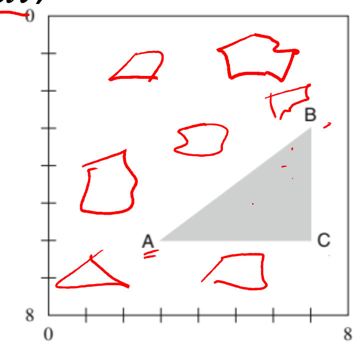
    $A=(6,3), B=(3,7), C=(6,7)$

Discrete & Continuous Maps

- Discrete map
  - *Not very accurate, Hard to recognize object in (a)*
  - *Improve accuracy : finer grid (16 x 16 or 256 x 256) → increased num of grid pts → robot memory req't increase + more powerful processing*
  - *Robot constraints : weight, cost, battery cap, etc (not practical)*
- Continuous map
  - *More efficient if fewer objects & simple shape*
  - *3 pairs of num (b) better representation than 7 pairs (a)*
  - *Easier to compute*
    - If point inside object or not using analytic geometry
  - *IF many objects or very complex shapes → not efficient anymore either for memory or processing requirement*



(a) Discrete map



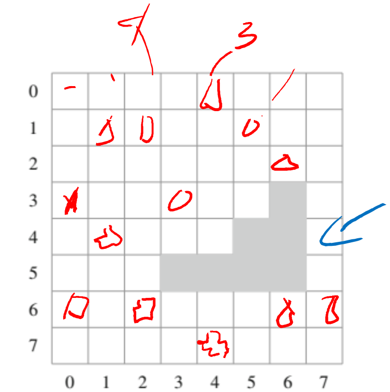
(b) Continuous map

Discrete & Continuous Maps

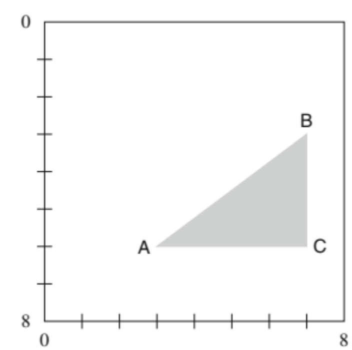
- If many objects or very complex shapes in (b)
  - *not efficient anymore*
  - *either for memory or processing requirement*
- If object in (b) bounded by high-order curves
  - *computation much more difficult*

Ex. : 32 objects of size 1, none touching each other

- Discrete map
  - *32 coordinates*
- Continuous map
  - *must store coordinates of the four corners of each object*



(a) Discrete map



(b) Continuous map

* **Discrete maps** commonly used in mobile robotics to represent the environment.

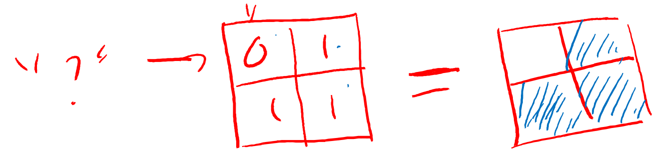
Mapping

- Discrete & Continuous Maps
- The Content of the Cells of a Grid Map
- Creating a Map by Exploration : Frontier Algorithm
- Mapping Using Knowledge of the Environment
- Numerical Example for a SLAM Algorithm
- Activities for Demonstrating the SLAM Algorithm

The Content of the Cells of a Grid Map

- Geographic maps use conventional notations describe env
  - *Colors : green (forests), blue (lakes), red (highways)*
  - *Symbols : sizes of dots (towns), thickness & color of lines (road quality)*
- Grid maps
  - *Each cell store a number*
  - *Must decide what number encodes*
  - *Simplest encoding is one bit for each cell : 0 → empty, 1 → object exists*
  - From (a) : *white* → 0, *gray* → 1

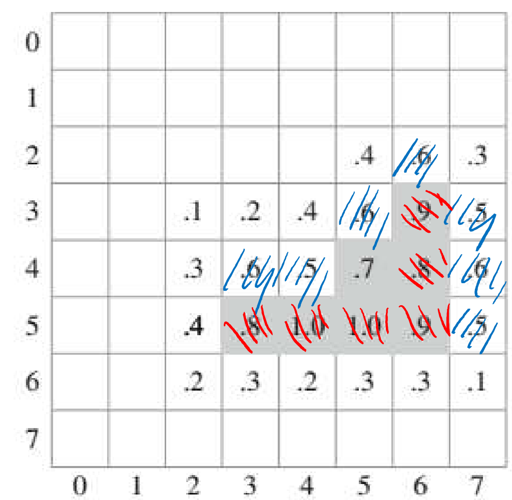
blue



- Sensors
  - *Not accurate*
  - *Difficult to be certain if cell is occupied or not*
  - *Makes sense to assign probability how certain object is in a cell*

The Content of the Cells of a Grid Map

- Figure is copy of (b) with probabilities for each cell
- Cells without number
  - *Assumed to be "0"*
- Cells occupied by the object
  - *Probability of at least 0.7*
- Choice of threshold up to us
  - *If threshold = 0.5*
    - *More cells* considered →
      Make object *bigger* than actually is
    - Since *we know* object is triangle
  - *Higher threshold of 0.7*
    - Give a *better* approximation



(b) **Probabilistic grid map**

Mapping

- Discrete & Continuous Maps
- The Content of the Cells of a Grid Map
- Creating a Map by Exploration : The Frontier Algo
- Mapping Using Knowledge of the Environment
- Numerical Example for a SLAM Algorithm
- Formalization of the SLAM Algorithm

Creating a Map by Exploration : Frontier Algo

- Imagine robo vacuum cleaner
  - *Newly let loose in your pad, Does not have map of your place*
  - *Must explore environment → gather information → construct own map*
- Several ways of exploring environment
  - *Random exploration is the simplest*
- Exploration much more efficient
  - *If robot has partial map of environment to guide its exploration*

https://image.freepik.com/free-vector/robot-vacuum-cleaner-interior-room-concept-home-cleaning-automation-household-remote-charging-station-vector-illustration-flat-cartoon-style_78677-9484.jpg
https://img.favpng.com/6/22/9/clip-art-robotic-vacuum-cleaner-illustration-drawing-png-favpng-yVdKNzDRzyek3kfkwJKJ0nAkp.jpg

Grid Maps with Occupancy Probabilities

- Obstacle probability
  - *Probability there is obstacle in the cell*
  - *Obstacle can be wall, table, anything stops the robot pass through the cell*
  - *"?" → not yet explored*
- In absence of any knowledge
  - *Can assume "0.5" → there is an obstacle*
  - *Could be easily occupied or not*
  - *"?" used instead of "0.5" to clarify unexplored status of the cells*
  - *Unknown cells*

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	1	?	?	?	?	?	0.9	1	0.9	?	?
?	?	?	?	?	1	0.1	0.1	?	?	?	1	0.2	1	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?
?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Grid map w/ occupancy probabilities

Grid Maps with Occupancy Probabilities

- Open cells
  - *Low values (0.1 or 0.2); Center of map free from obstacles*

0.9 & 0.1

- Three known obstacles
  - *Top-right, top-left, bottom-center*
  - *Occupancy probability "0.9" or "1.0"; gray cells*
- **Frontier** cells
  - *Open cells adjacent (left, right, up, down) to one or more unknown cells* ("?")
  - *Frontier is set of frontier cells*
  - *Red line of squares*
    - Boundary between unknown & frontiers

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	1	?	?	?	?	?	0.9	1	0.9	?	?
?	?	?	?	?	1	0.1	0.1	?	?	?	1	0.2	1	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?
?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Grid map w/ occupancy probabilities

The Frontier Algorithm

- Frontier Algorithm
  - *Expand ~~move~~ by exploring frontier*
  - *Robot move to closest frontier cell →*
    sense for obstacles in unknown adjacent cells →
    update map
- The FA (**moving**)
  - *Closest frontier cell is two cells above its initial position*
  - *Blue arrow shows that the robot has moved to that closest cell*

initial

find

robot position

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	1	?	?	?	?	?	0.9	1	0.9	?	?
?	?	?	?	?	1	0.1	0.1	?	?	?	1	0.2	1	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?
?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Robot moves to the frontier

The Frontier Algorithm

- The FA (**sensing**)
  - *Detect obstacles in adjacent unknown cells*
  - *Can detect in all 8 adjacent cells*
  - *Suppose UPPER-LEFT cell certainly contains obstacle → 1.0*
  - *UPPER-RIGHT and ABOVE cells certainly no obstacles → 0.1*
- The FA (**updating**)
  - *Map updated*
  - *The new information*
  - *New position of frontier*

robot position

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	1	?	?	?	?	?	0.9	1	0.9	?	?
?	?	?	?	?	1	0.1	0.1	1	0.1	0.1	1	0.2	1	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?
?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Robot updates unknown cells adjacent to the frontier

The Frontier Algorithm

- Next iteration of FA
  - *Robot moved UP one cell to closest frontier cell*
  - *Detected obstacles in two adjacent unknown cells*
  - *Map updated*
- Upper-right obstacle
  - *Completely known*
- Frontier cell
  - *None in the vicinity of the current position of the robot*

robot position

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	1	?	?	0.8	1	1	0.9	1	0.9	?	?
?	?	?	?	?	1	0.1	0.1	1	0.1	0.1	1	0.2	1	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?
?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Second iteration of the frontier algorithm

The Frontier Algorithm

- Another iteration of FA
  - *Robot blocked by upper right obstacle*
  - *It must avoid this as it moves to nearest frontier cell*

robot position

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	1	?	?	0.8	1	1	0.9	1	0.9	?	?
?	?	?	?	?	1	0.1	0.1	1	0.1	0.1	1	0.2	1	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?
?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Robot avoids an obstacle while moving to next frontier

The Frontier Algorithm

- Overall final iteration of FA
  - *Complete map constructed by the robot*
  - *Explored entire frontier as shown by path with blue arrows*

robot position

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0.8	1	1	1	1	1	1	1	0.8	1	1	0.9	1	0.9	?	?
0.9	0.1	0.1	0.1	0.2	1	0.1	0.1	1	0.1	0.1	1	0.2	1	?	?
1	0.1	0.1	0.1	0.1	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
1	0.1	0.1	0.9	0.1	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
1	0.1	0.3	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.8	0.8	?
1	0.1	1	1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	?
1	0.1	0.1	0.1	0.1	0.8	0.1	0.2	0.1	0.1	0.2	0.2	0.1	0.1	1	?
1	0.1	0.1	0.1	0.1	1	1	1	1	0.9	1	1	0.1	0.1	1	?
0.8	1	1	1	1	0.8	?	?	?	?	?	1	1	1	1	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Map constructed by FA & path explored by the robot

The Frontier Algorithm

```
float array grid    // Grid map
cell list frontier  // List of frontier cells
cell robot          // Cell with robot
cell closest        // Closest cell to robot
cell c              // Index over cells
float low           // Low occupancy probability
```

```
loop
  frontier ← empty
  for all known cells c in the grid
    if grid(c) < low and
      exists unknown neighbor of c
        append c to frontier
  exit if frontier empty

  closest ← cell in frontier nearest robot
  robot ← closest
for all unknown neighbors c of closest
  sense if c is occupied
  mark grid(c) w/ occupancy probability
```

(Algo 9.1) Frontier Algorithm

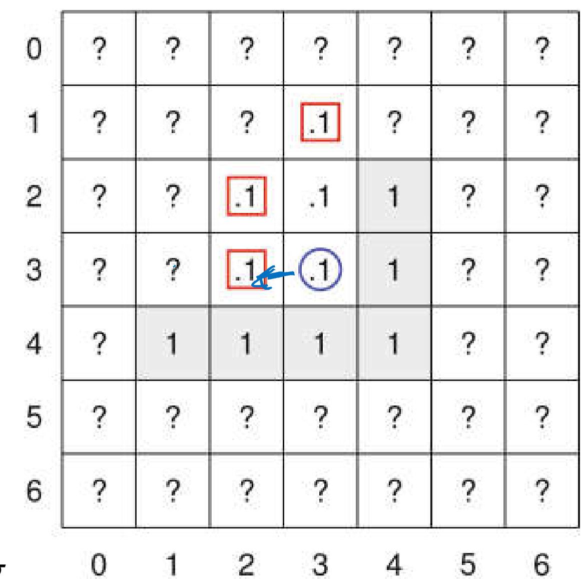
- **For simplicity**
  - Algorithm recomputes frontier at each step
- **More sophisticated**
  - Examine cells in neighborhood of its position
  - Add or remove cells whose status as frontier cells has changed

- **Previous example is relatively simple**
  - 2 rooms connected by a door (Col 6)
  - Also works in more complex environments

- **Can be run in parallel by multiple robots**
  - Each robot explore portion of frontier closest to its position
  - Share their partial maps so that consistency of maps is maintained
- **Map construction much more efficient**
  - Since each robot explore different area

Priority in the Frontier Algorithm

- In the figure
  - *Robot* → *(3, 3)* (Blue circle)
  - *Frontier cells* → *(1, 3), (2, 2), (3, 2)*
  - *5 open cells (3 of which are frontier)*
  - *6 obstacle cells (gray)*
  - *Diagonal neighbours not considered adjacent*
- From Algo 9.1
  - *Distance to frontier cell is the criteria in deciding where to move*
  - *Cell (3, 2) is closest at only 1-step away*
  - *While other frontier cells 2-steps away from robot position*
- We can consider different criterion for movement



Exploration of a labyrinth

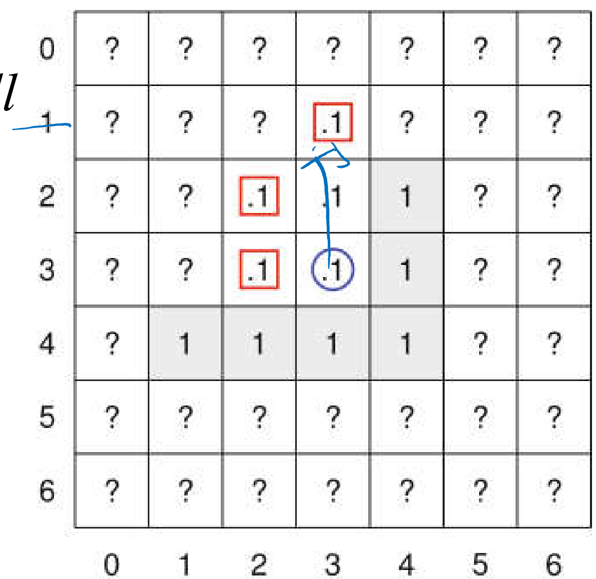
Priority in the Frontier Algorithm

(1, 3)

- Consider different criterion for movement
  - *Number of unknown cells adjacent to frontier cell*
- Frontier cell with more unknown cells
  - *Might make algorithm more efficient*
- Priority of a frontier cell , P_{cell}

$$p_{cell} = \frac{a_{cell}}{d_{cell}},$$

where:
a_{cell} is num adjacent unknown cells
d_{cell} is distance from the robot



Exploration of a labyrinth

- *Priorities of the three frontier cells are :*

$$p_{(3,2)} = 1/1 \Rightarrow 1\ ,\ p_{(2,2)} = 2/2 \Rightarrow 1,\ \ p_{(1,3)} = 3/2 \Rightarrow 1.5$$

- *Cell (1, 3) has highest priority → exploration starts from it*

Mapping

- Discrete & Continuous Maps
- The Content of the Cells of a Grid Map
- Creating a Map by Exploration : The Frontier Algo
- Mapping Using Knowledge of the Environment
- Numerical Example for a SLAM Algorithm
- Formalization of the SLAM Algorithm

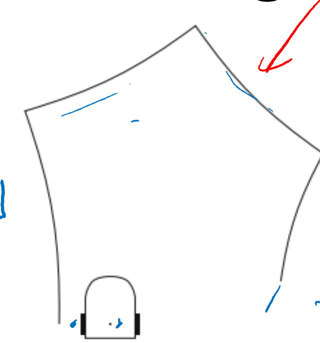
Mapping Using Knowledge of Environment

- Since we know how to explore the environment
  - *Let's consider how to build a map during exploration*
- Robot can localize (from prev section)
  - *With help of external landmarks*
  - *And their representation in map*
- Without *external landmarks*
  - *Only rely on odometry or inertial measurement*
  - *Subject to errors that increase with time*
- How to make map when localization is subject to large errors?
  - *Construct better map*
    - If there is some information on structure of the environment

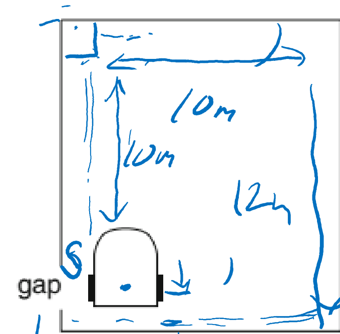
Mapping Using Knowledge of Environment

- **Ex. : Suppose robot constructs plan of room by wall following**
- Differences in real speeds of left & right wheels
  - *Make robot conclude that walls are not straight → (a)*
- Straight walls & perpendicular to each other
  - *If known in advance then → map (b) created*
  - *Sharp turn means → it is 90° corner (2 walls meet) → mapping of corners is correct*
  - *There is error in measuring length of walls → cause "gap" between 1st & last walls*
  - *Small gap not that important for this scenario*
- **Closing a loop** hard to solve if mapping large area
  - *Since robot only has local view of environment*

Gap = expected − actual

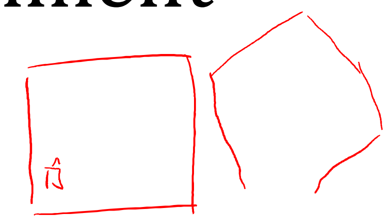


(a) Perceived motion based on Odometry



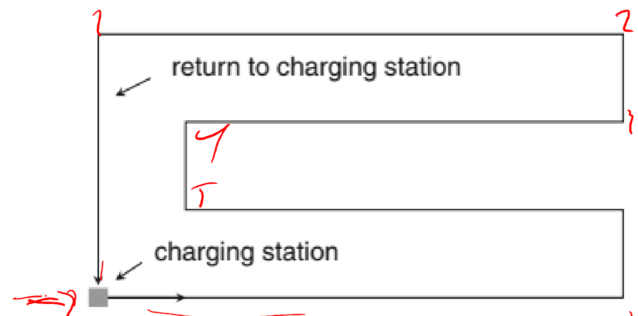
(b) Odometry + Knowledge of walls

Mapping Using Knowledge of Environment



- **Ex. : Robo lawnmower mow lawn back & forth**
- Must close loop by returning to charging station
  - *Make robot conclude that walls are not straight* → (a)
- Using odometry only
  - *Not possible*
  - *Small errors in velocity & heading* → *large errors in position or robot*

    accumulated

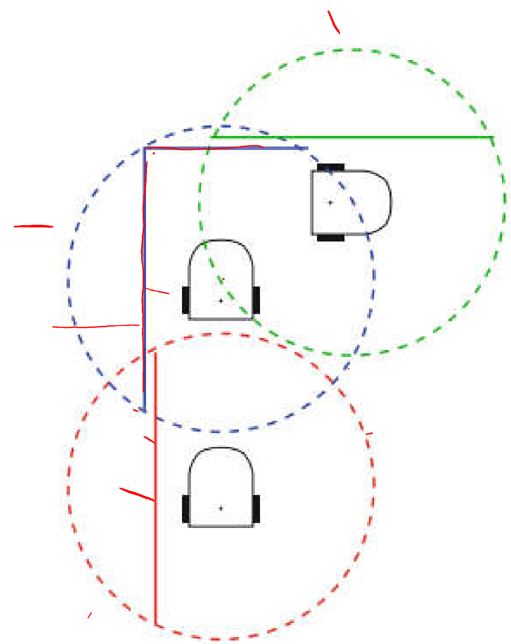


Mowing an area & go back to charging station

- Highly unlikely
  - *Robot mow entire surface* → *return to charging station*
- Close the loop
  - *Using landmarks such as signaling cables in the ground*

Mapping Using Knowledge of Environment

- Map construction significantly improved using sensor data
  - *Information on regular features of environment, especially long range*
  - *I.e.: lines on ground, global orientation, features that **overlap** with other environments*
- (a) Distance sensor measure over large area
  - *Identify features (walls & corners) from measurements taken at a single location*
- Large area measurements
  - *Help identify overlaps between constructed local maps at each location*
  - *Comparing local maps → Localization corrected & Map accurately updated*



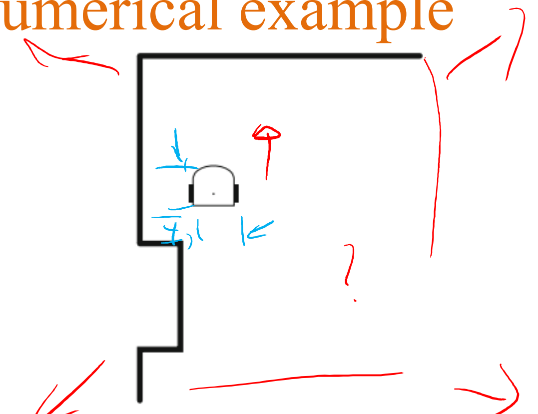
(a) Long-range measurement can detect overlap

Mapping

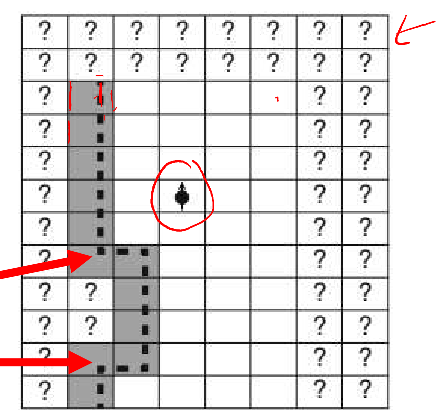
- Discrete & Continuous Maps
- The Content of the Cells of a Grid Map
- Creating a Map by Exploration : The Frontier Algo
- Mapping Using Knowledge of the Environment
- Numerical Example for a SLAM Algorithm
- Formalization of the SLAM Algorithm

Numerical Example for a SLAM Algorithm

- SLAM is quite complicated, we first compute numerical example
- **Ex. : Robot moving to top of diagram (a)**
  - *Robot near a corner of the room*
  - *Projection in wall to its left (supporting pillar)*
- In corresponding map (b)
  - Large dot w/ arrow : *robot & its heading*
  - Dotted line : *real wall*
  - White; gray; "?" : *known free*; *obstacles*; *unexplored*
- Cell considered part of obstacle
  - *If majority of area of cell is behind the wall*
  - *2 horizontal segments near boundary of cell*
    - Since almost all their area behind wall



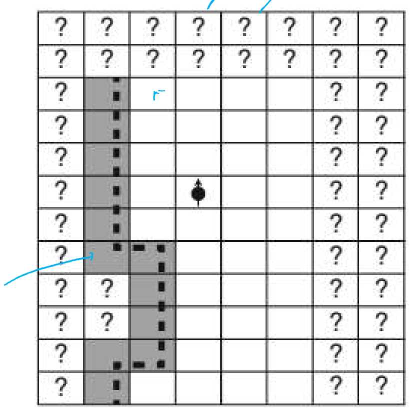
(a) Robot near wall



(b) Corresponding map

Numerical Example for a SLAM Algorithm

- The map is simplified
  - *For details of presenting details of SLAM*
- **First** simplification
  - *Cells are much too large*
  - *Roughly same size as robot itself*
  - *In practice, cells would be much smaller than it*
- **Second** simplification
  - *Each explored cells are specified either way*
  - Free cells : *white*
  - Gray cells : *obstacle*
  - Real SLAM algorithms : *use probabilistic representation*



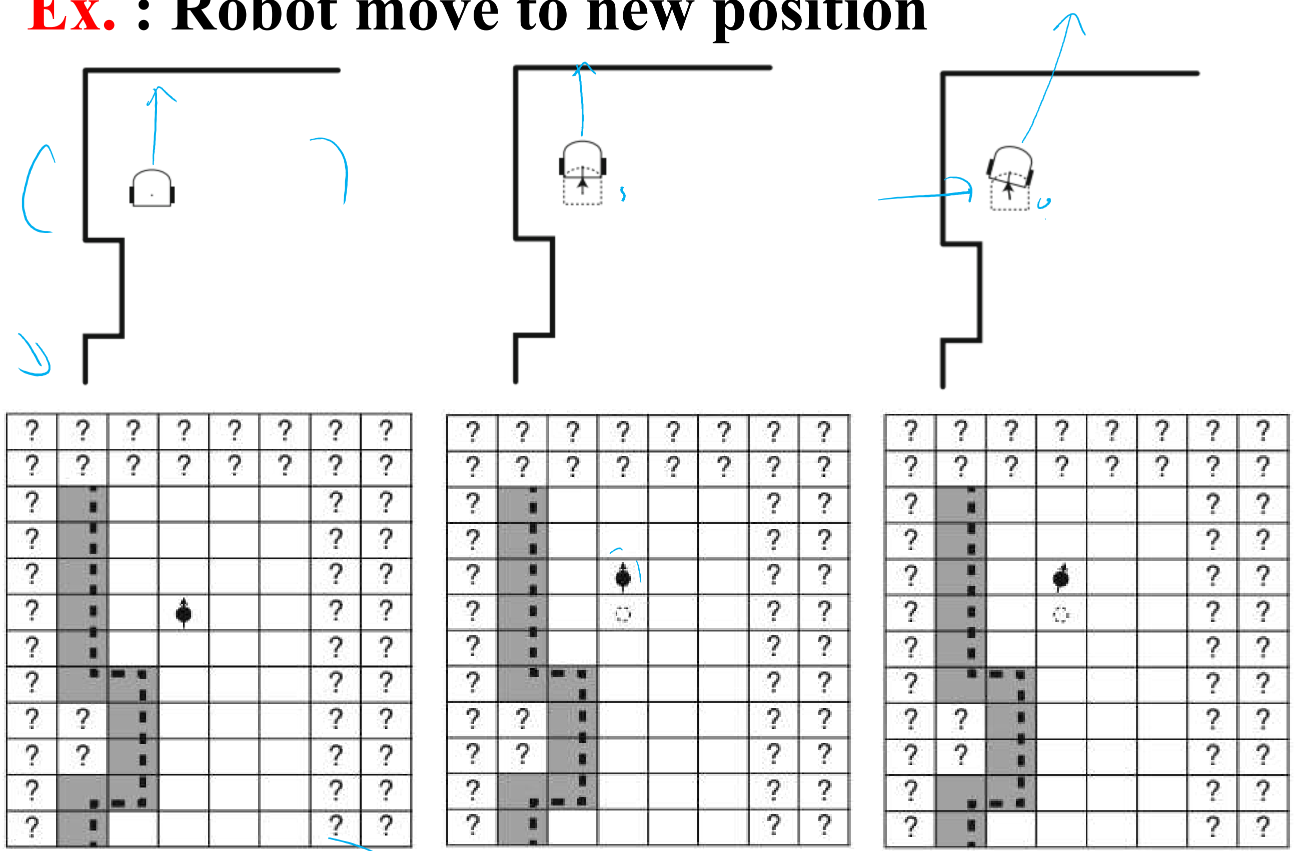
(a) Simplified

	0	1	2	3	4	5	6	7
0								
1								
2						.4	.6	.3
3			.1	.2	.4	.6	.9	.5
4			.3	.6	.5	.7	.8	.6
5			.4	.8	1.0	1.0	.9	.5
6			.2	.3	.2	.3	.3	.1
7								

(b) Probabilistic Example

Numerical Example for a SLAM Algorithm

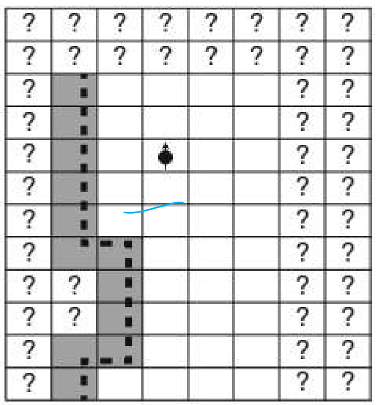
- **Ex. : Robot move to new position**



(a) Original **(b) Intended** **(c) Actual**

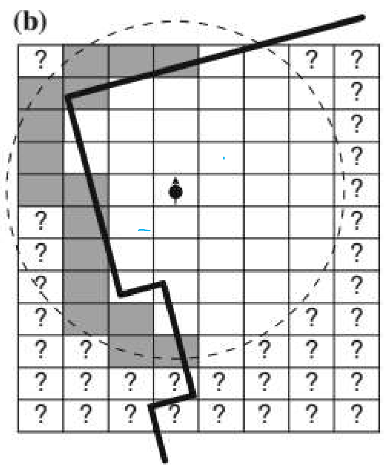
- **Correct position, wrong heading (c)**
  - Since right wheel moves over area with low-friction

Numerical Example for a SLAM Algorithm



(b) Intended

- **For intended perception (b)**
  - *Move one cell upwards*
  - *It can detect obstacles to its left*
  - *Investigate unknown cells in front*



(c) Actual Perception

- **Actual perception (c) is different**
  - *Due to errors in odometry*
  - *Actual position of wall* ***as seen by robot*** *overlaid on the cells*
  - *Cells colored gray is majority of area behind the wall*
- **Assumptions taken**
  - *Sense walls at a distance up to 5x the size of cells (dashed circle)*
  - *Any wall is one cell thick*

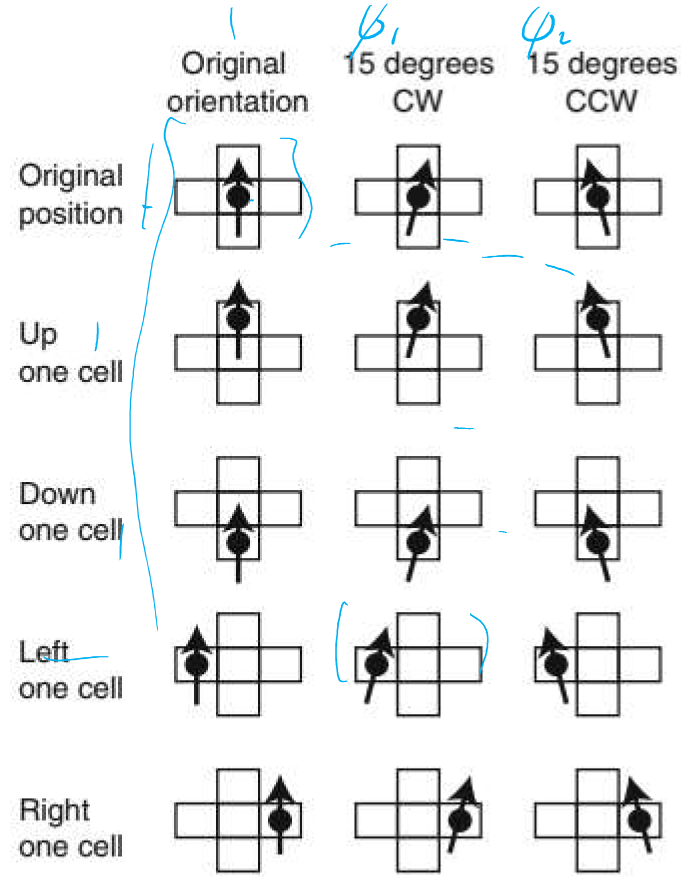
Numerical Example for a SLAM Algorithm

- Clear mismatch between current map & sensor data
  - *w/c should correspond to known part of the map*
  - *Robot not where it is supposed to be*
    - Based on odometry
  - *How is this mismatch corrected?*
- Assume odometry give reasonable estimate of pose (x, y, ϕ)
  - *Position and heading*
- For each small possible error in pose
  - *Compute what perception of current map would be*
  - *Compare w/ actual perception computed from sensor data*
  - *Chose pose w/ best match as actual pose*
  - *Update current map accordingly*

Numerical Example for a SLAM Algorithm

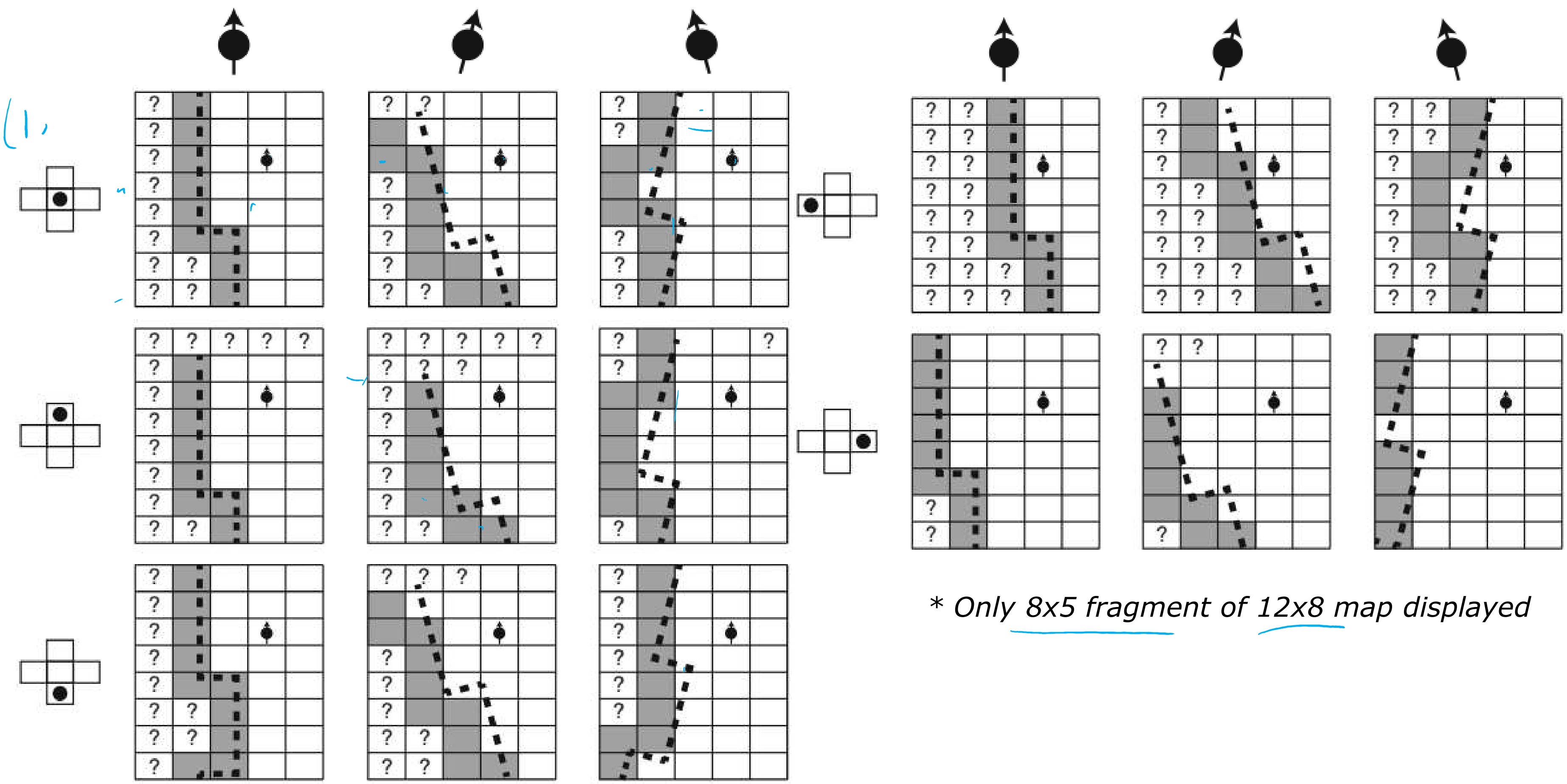
- In example, robot pose assumptions
- For **position**:
  - *In expected cell or*
  - *In any of four neighbours (up, down, left, right)*
- For **heading**:
  - *Correct, or*
  - *Slightly to right (15° CW), or*
  - *Slightly to left (15° CCW)*
- (a) : 5 x 3 possible poses

  (b) : perception of map computed from current map for each pose



(a) Possible poses of the robot

Numerical Example for a SLAM Algorithm

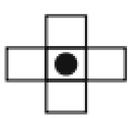


* Only 8x5 fragment of 12x8 map displayed

(b) Estimations of perception for different poses*

Numerical Example for a SLAM Algorithm

- Next step: choose map that give best fit w/ sensor measurements
- **First** : Transform 8x5 maps → 8x5 matrices
  - *Assign values for empty → "-1", obstacle → "+1", other → "0"*
  - *Current map → Matrix (a)*
  - *Perception map for pose* (correct cell, 15° CW) → *Matrix (b)*



0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	1	-1	-1
0	0	1	-1	-1
0	0	1	-1	-1

Matrix (a)

1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	1	-1	-1
0	0	1	1	-1

Matrix (b)

Computation of the matching between two maps

Numerical Example for a SLAM Algorithm

- To compare maps
  - *Multiply elements of corresponding cells*

- Let $m(i, j)$: the (i, j)'th cell of *current* map
  $p(i, j)$: the (i, j)'th cell of *perception* map obtained from *sensor* values

- $S(i, j)$, the ***similarity*** of (i, j)'th cell, is :

$$S(i,j) = m(i,j)\,p(i,j)$$

which can also be expressed as:

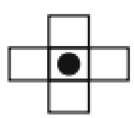
$$\begin{cases} S(i,j) = 1 & \text{if } m(i,j) \neq 0, p(i,j) \neq 0, m(i,j) = p(i,j) \\ S(i,j) = -1 & \text{if } m(i,j) \neq 0, p(i,j) \neq 0, m(i,j) \neq p(i,j) \\ S(i,j) = 0 & \text{if } m(i,j) = 0 \text{ or } p(i,j) = 0 \end{cases}$$

Numerical Example for a SLAM Algorithm

- Matrix (c)
  - *Result for* "Matrix (a) x Matrix (b)"
- A lot of "1"
  - *It tells us that matrices are similar → conclude perception maps similar*

m(i,j)=0 OR p(i,j)=0 → φ

1 2



thinks/ knows

gets/ observing

Matrix (a)

0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	1	-1	-1
0	0	1	-1	-1
0	0	1	-1	-1

x

Matrix (b)

1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	-1	-1	-1
0	1	1	-1	-1
0	0	1	1	-1

=

Matrix (c)

0	-1	1	1	1
0	-1	1	1	1
0	1	1	1	1
0	1	1	1	1
0	1	1	1	1
0	1	-1	1	1
0	0	1	1	1
0	0	1	-1	1

q(n)

Computation of the matching between two maps

Numerical Example for a SLAM Algorithm

- For quantitative result
  - *compute sum of similarities to get single value for any pair m, p :*

$$S = \sum_{i=1}^{8} \sum_{j=1}^{5} S(i, j)$$

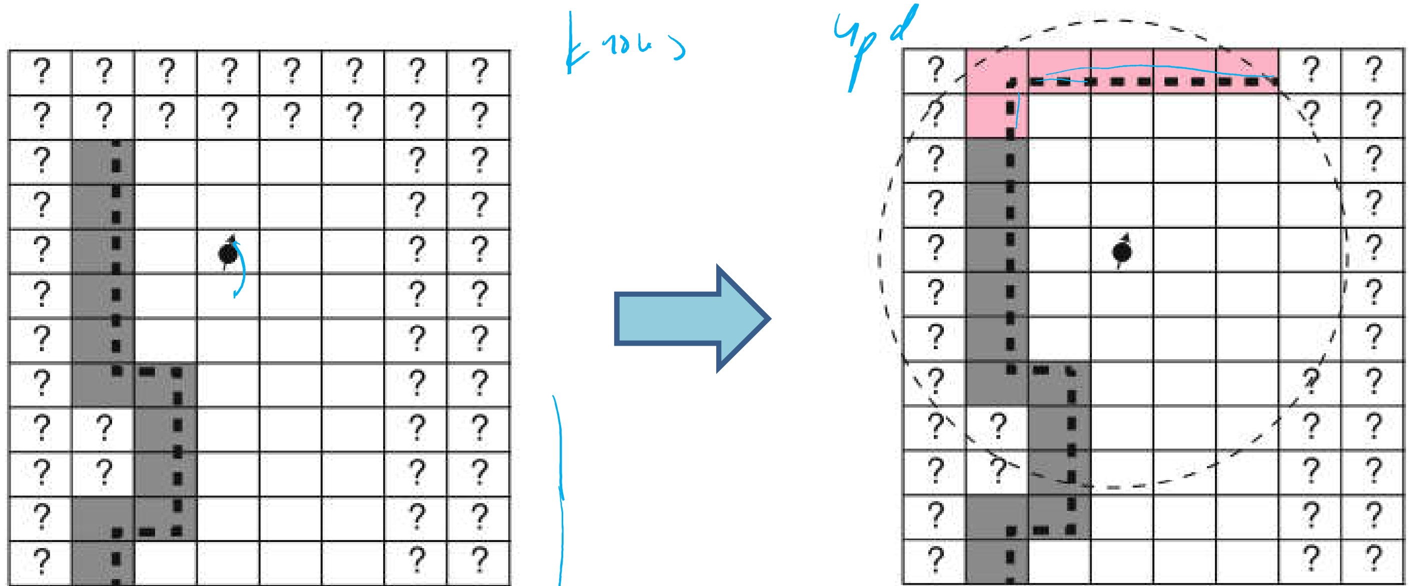
- S for all perception maps compared with current map (a.t)
  - *As expected, highest similarity is for map corresponding to pose (correct position, 15° CW)*

	Intended orientation	15° CW	15° CCW
Intended position	22	**32**	20
Up one cell	23	25	16
Down one cell	19	28	21
Left one cell	6	7	18
Right one cell	22	18	18

(a.t) Similarity *S* of sensor-based map with current map

Numerical Example for a SLAM Algorithm

- With the obtained result
  - *Correct pose*
  - *Use data from perception map to update current map in memory*



Current map **Updated current map**

Map before and after update using data from perception map

Mapping

- Discrete & Continuous Maps
- The Content of the Cells of a Grid Map
- Creating a Map by Exploration : The Frontier Algo
- Mapping Using Knowledge of the Environment
- Numerical Example for a SLAM Algorithm
- Formalization of the SLAM Algorithm

Formalization of the SLAM Algorithm

perceives

- SLAM algorithm that find position
  - *Whose perception map closest to perception map obtained from sensor data*
  - *Robot localized & map updated to what is perceived at this position*

```
matrix m ← partial map       // Current map
matrix p                     // Perception map
matrix e ←                   // Expected map
coordinate c ← initial pos   // Current pos
coordinate n                 // New position
coordinate array T           // Set of test pos
coordinate t                 // Test position
coordinate b ← none          // Best position
```

```
loop
    move a short distance
    /* New position based on odometry */
    n ← odometry(c)
    p ← analyze sensor data
    /* T is positions around n */
    for every t in T
        /* Expected map at least pos */
        e ← expected(m, t)
        if compare(p, e) better than b
            /* Best position so far */
            b ← t

    /* Replace new pos with best pos */
    n ← b
    /* Update map based on new pos */
    m ← update(m, p, n)
    /* Current pos is new pos */
    c ← n
```

Formalization of the SLAM Algorithm

- Algorithm is divided into three phases
  - *First phase : lines 2 - 4*

```
matrix m ← partial map // Current map
matrix p                // Perception map
matrix e  ←             // Expected map
coordinate c ← initial pos // Current pos
coordinate n            // New position
coordinate array T      // Set of test pos
coordinate t            // Test position
coordinate b ← none     // Best position
```

```
01:  loop
02:      move a short distance
         /* New position based on odometry */
03:      n ← odometry(c)
04:      p ← analyze sensor data
```

- **First phase**
  - Robot move short distance
  - New position computed by odometry
  - Analyze sensor data → obtain perception map
  - Assume odometry error relatively small → define set of test positions`

1.5 cell [illegible]

±25°

cm/cell

Formalization of the SLAM Algorithm

- Algorithm is divided into three phases
  - *Second phase : lines 5 - 8*
  - *Third phase : lines 9 - 11*

- Localization
- Mapping
- Loc + Mapp

```
        /* T is positions around n */
05:     for every t in T
          /* Expected map at least pos */
06:       e ← expected(m, t)
07:       if compare(p, e) better than b
            /* Best position so far */
08:         b ← t

        /* Replace new pos with best pos */
09:     n ← b
        /* Update map based on new pos */
10:     m ← update(m, p, n)
        /* Current pos is new pos */
11:     c ← n
```

- **Second phase**
  - Expected map at each position computed → compared with current map
  - Best match is saved

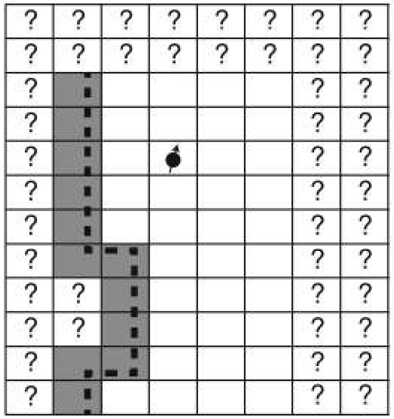
- **Third phase**
  - Position with the best match → becomes new position
  - Current map updated accordingly

More Complicated in Practice

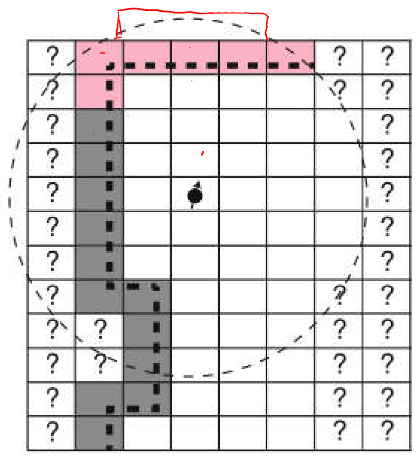
- More complicated
  - *Take into account perception map from sensors limited by sensor's range*
- Overlap is partial
  - *Sensor range doesn't cover entire current map*
  - *Sensors detect obstacles & free areas outside current map*
- Therefore:
  - *Perceived map (**p**) size much smaller than expected map (**e**), and*
  - *Function **compare(p, e)** only compare areas that overlap*
- Also, when updating current map
  - *Areas not previously in the map will be added*

More Complicated in Practice

- There are cells in the current map
  - *Outside five-cell radius of sensor*
  - *Will not be updated*
- Light red cells
  - Current map : *unknown*
    Indicated by "?"
  - From perception map : *now known*
    as part of obstacle
  - This *information used* →
    update the current map →
    obtain updated current map.



Current map



Updated current map

Summary

- **Accurate robotic motion in uncertain environment**
  - *Require robot has map of environment*
  - *Grid map of cells or graph representation of continuous map*
  - *In uncertain environments*
    - Map typically not available before task can be begun by robot
- **Frontier algorithm**
  - *Construct a map as it explores its surrounding*
  - *More accurate maps can be constructed*
    - With some knowledge of its environment
- **SLAM (Simultaneous Localization And Mapping)**
  - *Use iterative process to construct map*
  - *While also correcting errors in localization*

Thank you.