ANSWERS

U8.6 Frontiers

BugBotLab

Mapping · University · about 30 min

What this lesson is about

The edge between what is known and what is not, and why a map with holes is still useful.

- The definition
- From cells to targets
- Watching exploration saturate
- Shadows
- When it goes wrong

Questions

6 marks in all

1. What does this program print? [1 mark]

```
rows = ["FFFU",
        "FOFU",
        "FFFF",
        "UUFF"]
H, W = len(rows), len(rows[0])
count = 0
for r in range(H):
    for c in range(W):
        if rows[r][c] == "F":
            for dr, dc in ((1, 0), (-1, 0), (0, 1), (0, -1)):
                rr, cc = r + dr, c + dc
                if 0 <= rr < H and 0 <= cc < W and rows[rr][cc] == "U":
                    count += 1
                    break
print(count)
```

Answer:

```
6
```

F is free, O occupied, U unknown. The free cells with an unknown cell above, below, left or right are (0,2), (1,2), (2,0), (2,1), (2,3) and (3,2), so 6 frontier cells. (2,2) is free but all four of its neighbours are known.

2. Why is driving to a frontier cell guaranteed to produce new information? [1 mark]

- ☐ **A** It is free, so it can be reached, and it borders a cell no beam has ever reached
- ☐ **B** It is always the cell furthest from the robot
- ☐ **C** It is next to an occupied cell, so the robot will see a wall
- ☐ **D** Its log odds are exactly zero

Answer: A. A frontier is a free cell with at least one unknown neighbour. When none are left and everywhere is reachable, the map is finished and the robot knows it.

3. Why can a feature map not support frontier exploration? [1 mark]

- ☐ **A** Absence from its list means both "not there" and "never looked", so it cannot say "unknown"
- ☐ **B** Feature maps cannot store positions accurately enough
- ☐ **C** Feature maps use too much memory for large rooms
- ☐ **D** Feature maps have no free cells to drive to

> **Answer:** A. The grid distinguishes free, occupied and unknown, and the third kind is the one exploration needs.

4. Put the frontier exploration pipeline in order. [1 mark]

Number the lines 1 to 5 to put them in the right order.

___ `Group adjacent frontier cells into regions with a flood fill`

___ `Discard regions smaller than the robot`

___ `Score each remaining region`

___ `Find every free cell with an unknown neighbour`

___ `Drive to the best one`

> **Answer:**
>
> ```
> Find every free cell with an unknown neighbour
> Group adjacent frontier cells into regions with a flood fill
> Discard regions smaller than the robot
> Score each remaining region
> Drive to the best one
> ```
>
> Individual frontier cells come in thousands, so they are grouped, the noise is thrown away, and only then is a target chosen.

5. The robot alternates for ever between two frontiers of equal value on opposite sides of the room. What is the fix? [1 mark]

- ☐ **A** Hysteresis: keep the current target until it is reached or disappears
- ☐ **B** Discard frontier regions smaller than the robot
- ☐ **C** Use eight neighbours instead of four
- ☐ **D** Lower L_MAX so the map changes its mind faster

> **Answer:** A. This is ping-pong. Discarding small regions fixes a different failure, the sliver of unknown along a wall that never closes.

6. A small post leaves a wedge of unknown behind it. Why does moving a short distance sideways often beat driving a long way forward? [1 mark]

- ☐ **A** A second viewpoint a little apart sees round the post and resolves nearly all of the shadow
- ☐ **B** Sideways motion gives less odometry error
- ☐ **C** The sensor has a longer range sideways
- ☐ **D** Driving forward would make the robot collide with the post

> **Answer:** A. The shadow widens with distance from a single viewpoint, so a small change of viewpoint uncovers a large area.

The task: what have I not seen?

Turn on the spot, building a grid as you go. Plot `known`, then print `explored:`, the fraction of cells that are no longer unknown, and `frontier:`, how many free cells have an unknown neighbour.

```
from bugbot import *
import math
connect()

CELL, W = 5.0, 40
L_OCC, L_FREE, L_MAX, FAR = 0.85, -0.4, 8.0, 170.0
START_X, START_Y = 100.0, 100.0
grid = [0.0] * (W * W)
```

The hint students can ask for: Turn on the spot for a good while, integrating every scan into the grid. A cell is unknown while its log odds is near zero; a frontier cell is a free one with an unknown neighbour. Count both, and plot the known fraction so you can watch it stop rising.

A solution

```
from bugbot import *
import math
connect()

CELL, W = 5.0, 40
L_OCC, L_FREE, L_MAX = 0.85, -0.4, 8.0
FAR = 170.0
START_X, START_Y = 100.0, 100.0
grid = [0.0] * (W * W)

def bump(x, y, amount):
    c, r = int(x / CELL), int(y / CELL)
    if 0 <= c < W and 0 <= r < W:
        i = r * W + c
        grid[i] = max(-L_MAX, min(L_MAX, grid[i] + amount))

def integrate():
    px, py = position()
    x, y = START_X + px, START_Y + py
    h = heading()
    for a, d in scan():
        th = math.radians(h + a)
        sx, sy = math.sin(th), math.cos(th)
        r = 0.0
        while r < min(d, FAR) - CELL:
            bump(x + r * sx, y + r * sy, L_FREE)
            r += CELL / 2
        if d < FAR:
            bump(x + d * sx, y + d * sy, L_OCC)

def explored():
    return sum(1 for v in grid if abs(v) > 1.0) / float(W * W)

drive(0, 0, 30)
for tick in range(150):
    integrate()
    plot("known", explored())
```

```
    wait(0.1)
stop()
integrate()

print("explored:", round(explored(), 3))

# a frontier cell is a free cell with an unknown neighbour: the edge of what is known
frontier = 0
for r in range(W):
    for c in range(W):
        if grid[r * W + c] >= -1.0:
            continue
        for dc, dr in ((1, 0), (-1, 0), (0, 1), (0, -1)):
            c2, r2 = c + dc, r + dr
            if 0 <= c2 < W and 0 <= r2 < W and abs(grid[r2 * W + c2]) <= 1.0:
                frontier += 1
                break
print("frontier:", frontier)
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.