ANSWERS

U8.5 Ray casting the map

Mapping · University · about 35 min

What this lesson is about

The forward model: given a map and a pose, what would the sensor read?

- Why it matters
- The cost, and what everybody does about it
- Doing it
- Unknown is not occupied

Questions

6 marks in all

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

```
CELL = 5.0
occupied_row = 6
y = 2.0
r = 0.0
while int((y + r) / CELL) < occupied_row:
    r += 1.0
surface = 33.0
print(r, surface - y)
```

Answer:

```
28.0 31.0
```

The caster stops on entering row 6 at y = 30, so it predicts 28 cm, but the surface inside the cell is 31 cm away. That shortfall is the half-cell bias.

2. A ray caster's predictions are consistently a few centimetres shorter than the real readings. Why? [1 mark]

☐ **A** It stops on entering an occupied cell, so it reports the near edge of the cell rather than the surface inside it

☐ **B** The sensor adds a constant offset to every reading

☐ **C** The steps of 1 cm skip past thin walls

☐ **D** Unknown cells are being treated as blocked

Answer: A. The bias averages about half a cell and is systematic, so a localiser scored on it is pulled forwards unless it is modelled or absorbed into sigma.

3. A particle filter has 300 particles and uses 2 beams each, at 10 Hz, and each ray cast marches 150 steps. How many grid lookups is that per second? [1 mark]

Answer: 900000. 300 x 2 x 10 x 150 = 900,000. With all 8 beams and 500 particles it is six million, which is why beams are subsampled.

4. What does a likelihood field precompute, and what does it save? [1 mark]

- ☐ **A** The distance from every cell to the nearest occupied cell, so scoring a beam is one lookup at its end point instead of a march
- ☐ **B** The ray cast from every cell in every direction, so no casting is needed at run time
- ☐ **C** The log odds of every cell, so the grid need not be stored
- ☐ **D** The frontier cells, so exploration is faster

> **Answer:** A. One lookup per beam, and it is smoother to optimise against since there is no jump where a ray just clips a corner.

5. A planner uses a ray cast to ask whether a straight route is safe. How should that caster treat unknown cells? [1 mark]

- ☐ **A** As blocked, so the robot never drives into a place it has not looked at
- ☐ **B** As free, so it is only scored on evidence it has
- ☐ **C** As occupied only if they are next to a known wall
- ☐ **D** It makes no difference to the answer

> **Answer:** A. Unknown as free suits localisation and map checking; unknown as blocked suits a planner. The same system uses both, so the caller should choose.

6. Why can using two of the eight beams improve the estimate as well as the speed? [1 mark]

- ☐ **A** Neighbouring beams are correlated, and treating eight near-identical beams as independent makes the filter overconfident
- ☐ **B** The outer beams are always less accurate than the inner ones
- ☐ **C** Fewer beams mean less half-cell bias
- ☐ **D** The weights underflow when more than two are multiplied

> **Answer:** A. Multiplying the likelihoods of beams that carry the same information counts that information several times.

The task: predict a reading from the map

Standing at (100, 50), build a map from the scans. Then ray cast from a pose 20 cm behind the robot and print `predicted:`. Drive back 20 cm, measure, and print `measured:` and `error:`.

```
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, 50.0
grid = [0.0] * (W * W)
```

The hint students can ask for: Build a small map standing still, then march a ray out of it from a pose the robot is not at yet: one step at a time until a cell is occupied, and the distance you have marched is the prediction. Then drive back 20 cm and see whether the map was telling the truth.

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, 50.0
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)

for i in range(25):
    integrate()
    wait(0.1)

# the forward model: march out of the map until a cell is occupied
def cast(x, y, h, limit=250.0):
    th = math.radians(h)
```

```
    sx, sy = math.sin(th), math.cos(th)
    r = 0.0
    while r < limit:
        c, row = int((x + r * sx) / CELL), int((y + r * sy) / CELL)
        if not (0 <= c < W and 0 <= row < W):
            return limit
        if grid[row * W + c] > 1.0:
            return r
        r += 1.0
    return limit

predicted = cast(START_X, START_Y - 20.0, 0.0)
print("predicted:", round(predicted, 1))

backward(50, distance=20)
wait(0.3)
readings = []
for i in range(8):
    readings.append(distance())
    wait(0.1)
measured = sum(readings) / len(readings)
print("measured:", round(measured, 1))
print("error:", round(abs(predicted - measured), 1))
```

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