WORKSHEET

U8.2 The inverse sensor model

BugBotLab

Mapping · University · about 30 min

Name _____ Class _____ Date _____

What this lesson is about

One reading is two statements: free all the way along the ray, occupied at the end of it.

- Why it is called inverse
- The model, in full
- Where the beam actually points
- One ray, drawn
- The beam is not a line

Questions

7 marks in all

1. A mapper marks only the cell at the end of each beam as occupied. What is wrong with the map it builds? [1 mark]

- ☐ **A** It outlines the walls but knows nothing about the floor, which is what a planner needs
- ☐ **B** It puts a ring of imaginary wall around the robot
- ☐ **C** The walls never become confident
- ☐ **D** It comes out mirrored about the diagonal

2. Which of these is the inverse sensor model? [1 mark]

- ☐ **A** Given a reading and a pose, what the world is like along the beam
- ☐ **B** Given the world and a pose, what the sensor will read
- ☐ **C** Given two readings, how the robot moved between them
- ☐ **D** Given the map, which cells are frontiers

3. The sensor finds nothing within range and returns its maximum. How should the mapper treat that beam? [1 mark]

- ☐ **A** Mark the cells along it free, and mark nothing occupied
- ☐ **B** Mark the cell at maximum range occupied, as for any reading
- ☐ **C** Ignore the beam entirely
- ☐ **D** Mark every cell along it occupied

4. Why does the free marking stop one cell short of the hit? [1 mark]

- ☐ **A** Otherwise the free update fights the occupied update in the same cell and the wall never becomes confident
- ☐ **B** The last cell is always outside the mat
- ☐ **C** It corrects for the half-cell bias of a ray caster
- ☐ **D** Free cells next to a wall are unsafe for a planner

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

```
import math
CELL = 5.0
x, y, h = 100.0, 50.0, 90.0
a, r = 0.0, 40.0
th = math.radians(h + a)
hit_x = x + r * math.sin(th)
hit_y = y + r * math.cos(th)
print(round(hit_x, 1), round(hit_y, 1))
print(int(hit_x / CELL), int(hit_y / CELL))
```

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

```
CELL = 5.0
X0, Y0 = 100.0, 50.0
d = 100.0
free = set()
r = 0.0
while r < d - CELL:
    free.add((int(X0 / CELL), int((Y0 + r) / CELL)))
    r += CELL / 2
print(len(free), (int(X0 / CELL), int((Y0 + d) / CELL)))
```

7. A mapper's walls come out mirrored about the diagonal of the mat, and otherwise look plausible. What is the most likely bug? [1 mark]

- ☐ **A** The sine and cosine are the wrong way round in the hit position
- ☐ **B** The heading is in degrees instead of radians
- ☐ **C** The free marking does not stop one cell short
- ☐ **D** A timeout is being marked as a hit

The task: one ray, two statements

Take a reading, run the inverse sensor model along it, and print `free:`, the number of different cells the ray passes through, and `wall y:`, the mat y of the cell it ended in.

```
from bugbot import *
connect()

CELL = 5.0
X0, Y0 = 100.0, 50.0
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/u8-2-the-inverse-sensor-model/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Step in whole cells instead of half and count the cells again, then do the same for a 100 cm beam 30 degrees to the right. Why does only the slanting one lose cells?
2. Mark the free cells all the way to `d` instead of stopping short. What happens to the hit cell?
3. Work out how far apart two neighbouring beams of `scan()` are at 50 cm and at 150 cm.