ANSWERS

U7.7 Project: the kidnapped robot

Localisation · University · about 50 min

What this lesson is about

No idea where it starts. Work it out, then drive somewhere on purpose.

- Two phases
- What to watch on the chart
- The structure
- Readings while turning
- What comes next

Questions

7 marks in all

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

```
import math
SIGMA = 3.5
particles = [40.0, 42.0, 44.0, 150.0, 152.0]
measured = 157.0
w = [math.exp(-(((200 - y) - measured) ** 2) / (2 * SIGMA * SIGMA)) + 1e-12 for y in
particles]
total = sum(w)
weights = [v / total for v in w]
print(round(sum(v * y for v, y in zip(weights, particles)), 1))
```

Answer:

```
42.2
```

The reading 157 means y is about 43, so the three particles near 40 to 44 share the weight and the two near 150 predict about 49 and get essentially none. The weighted mean is 42.2.

2. The depth sensor faces the far wall, so its readings pin down y. The robot turns ninety degrees to the left. What does a cloud weighed against the new readings pin down? [1 mark]

- ☐ **A** x
- ☐ **B** y again, more precisely
- ☐ **C** The heading
- ☐ **D** Both x and y at once

Answer: A. One sensor measures the wall it points at and nothing else. Facing a side wall, the reading depends on x, so two clouds one after the other give both coordinates.

3. In phase one the robot faces the far wall, and the cloud finds y but stays a stripe across the mat. What cuts the stripe down to a spot? [1 mark]

- ☐ **A** A reading from a different direction, such as the left wall once a ninety degree turn has finished
- ☐ **B** Shuffling sideways and reading the far wall again
- ☐ **C** Averaging more readings of the far wall
- ☐ **D** Using a smaller sigma

> **Answer:** A. Every reading of the far wall, from anywhere, only says how far the robot is from that wall, which is y. A reading of a side wall depends on x, so the pair pins down both.

4. A project filter weighs its x cloud against every reading, including those taken during the ninety degree turn towards the left wall. It ends 58 cm from the robot with a spread of 5 cm. Why? [1 mark]

- ☐ **A** The readings during the turn were at a slant, so the cloud collapsed where they seemed to point, and once no guess was near the right x it could not come back
- ☐ **B** The turn made the odometry heading drift by 2 to 3 degrees
- ☐ **C** Five cm of spread is too much jitter
- ☐ **D** The left wall is further away than the far wall

> **Answer:** A. Weighed against readings its model cannot explain, the cloud moved to the wrong place with full confidence, and resampling can only copy the guesses it has. Skipping readings until the turn is finished found x to within 1 cm. A few degrees of gyro drift are harmless, because the sensor's cone forgives them.

5. During the run the estimate suddenly jumps by 60 cm. What has most likely happened? [1 mark]

- ☐ **A** The cloud had two clusters and has just resampled onto one of them
- ☐ **B** The motion noise is too large
- ☐ **C** The effective sample size has risen to N
- ☐ **D** The robot has driven into a wall

> **Answer:** A. The weighted mean of two clusters sits between them; when one cluster dies the estimate jumps. Whether it chose correctly is the interesting question.

6. In phase two the spread starts climbing steadily. What does that mean? [1 mark]

- ☐ **A** The measurements have stopped being informative and the filter is effectively dead reckoning
- ☐ **B** The filter is converging on the true position
- ☐ **C** Too many particles are being injected
- ☐ **D** The resampling threshold is too low

> **Answer:** A. Motion noise grows the spread and measurements shrink it. If only the first is happening, no reading is doing any work.

7. Which defence specifically lets the filter recover if the robot is picked up and moved mid-run? [1 mark]

- ☐ **A** Injecting a small percentage of particles scattered over the whole mat every tick
- ☐ **B** Adding jitter to each resampled copy
- ☐ **C** Resampling only when neff is below N/2
- ☐ **D** Using a larger number of particles at the start

> **Answer:** A. Jitter only keeps diversity near the current cloud. Scattered particles are the only guesses that can be near a place the cloud has written off.

The task: the kidnapped robot

Work out where the robot woke up, print it as `found:`, and then drive into the green corner. `position()` is not allowed anywhere.

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

DT, N, SIGMA = 0.1, 400, 3.5
particles = [random.uniform(0, 200) for i in range(N)]
```

The hint students can ask for: The robot wakes up somewhere on the mat and does not know where. The depth sensor only measures the wall it is pointing at, so one cloud can only find one axis: localise y facing the far wall, drive to the right y, then turn to face the left wall and localise x the same way. Print the y you worked out before you set off.

A solution

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

DT = 0.1
N = 400
SIGMA = 3.5
V_MAX = 20.0

def localise(predict, ticks=25, jitter=0.8):
    """A cloud over the whole mat, weighed against what the sensor would read at each
place."""
    particles = [random.uniform(0, 200) for i in range(N)]
    for t in range(ticks):
        measured = distance()
        w = []
        for v in particles:
            d = predict(v) - measured
            w.append(math.exp(-d * d / (2 * SIGMA * SIGMA)) + 1e-12)
        total = sum(w)
        weights = [v / total for v in w]
        step = 1.0 / N
        r = random.uniform(0, step)
        c = weights[0]
        i = 0
        fresh = []
        for m in range(N):
            u = r + m * step
            while u > c and i < N - 1:
                i += 1
                c += weights[i]
            fresh.append(particles[i] + random.gauss(0, jitter))
        particles = fresh
        wait(DT)
    return sum(particles) / N

# facing the far wall: the reading is 200 minus y, and driving forward increases y
found_y = localise(lambda y: 200 - y)
```

```
print("found:", round(found_y, 1))

y = found_y
for tick in range(500):
    error = 50.0 - y
    if abs(error) < 4:
        break
    drive(100 * max(-12.0, min(12.0, 0.6 * error)) / V_MAX, 0, 0)
    y += flow()[1] * DT
    wait(DT)
    if tick % 5 == 0:
        y = 0.7 * y + 0.3 * (200 - distance())
stop()
wait(0.4)

# face the left wall: now the reading IS x, and driving forward reduces it
turn_left(45, angle=90)
wait(0.5)
x = localise(lambda v: v)

for tick in range(500):
    error = x - 50.0                # too far right means drive forward
    if abs(error) < 4:
        break
    drive(100 * max(-12.0, min(12.0, 0.6 * error)) / V_MAX, 0, 0)
    x -= flow()[1] * DT
    wait(DT)
    if tick % 5 == 0:
        x = 0.7 * x + 0.3 * distance()
stop()
print("in the corner near", round(x, 1), round(y, 1))
```

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