ANSWERS

U7.3 Moving the cloud

Localisation · University · about 30 min

What this lesson is about

The motion update: every particle drives, and every particle's own error goes with it.

- How much noise to add
- This is where the model goes

Questions

6 marks in all

1. A motion update moves every particle by the measured motion with no random term. What goes wrong? [1 mark]

- ☐ **A** The cloud keeps its shape for ever, so the filter claims to be as certain after ten metres as at the start
- ☐ **B** The cloud spreads without limit and the estimate becomes vague
- ☐ **C** The particles drift away from the robot's true motion
- ☐ **D** The weights stop summing to one

> **Answer:** A. The random part is the filter's honest statement that dead reckoning is losing accuracy. Without it the spread never grows.

2. The noise added per step is much smaller than the robot's real motion error. What is the likely result? [1 mark]

- ☐ **A** An over-confident cloud that shrinks onto a wrong answer and refuses to be corrected
- ☐ **B** A vague estimate because the cloud spreads faster than readings can pull it in
- ☐ **C** Exactly the same estimate, since the measurement update corrects it every tick
- ☐ **D** The effective sample size rises towards N

> **Answer:** A. Too little noise gives over-confidence; too much gives vagueness. The check is whether the truth stays inside the cloud.

3. Each particle gets independent noise with standard deviation 0.35 cm per step. After 40 steps, what spread (standard deviation) should the cloud have, in cm to two decimal places? [1 mark]

> **Answer:** 2.21 (accept within 0.01). Independent errors add in variance, so the spread is 0.35 x sqrt(40) = 0.35 x 6.32 = 2.21 cm, the random walk from U3.3.

4. You double the per-step noise standard deviation. What happens to the spread of the cloud after 5 seconds? [1 mark]

- ☐ **A** It doubles
- ☐ **B** It quadruples
- ☐ **C** It grows by a factor of sqrt(2)
- ☐ **D** It stays the same, because it depends only on the number of steps

> **Answer:** A. The spread after n steps is sigma x sqrt(n), which is proportional to sigma. The variance quadruples, the standard deviation doubles.

5. A Kalman filter for the same robot would use a process noise Q of 0.16 cm squared per step. What standard deviation, in cm, is a reasonable starting point for the per-particle motion noise? [1 mark]

> **Answer:** 0.4 (accept within 0.001). Q is a variance; particle noise is drawn with a standard deviation, so take sqrt(0.16) = 0.4 cm.

6. Which of these belong in the motion update of a particle filter? [1 mark]

Tick every answer that is true.

- ☐ **A** Speed proportional to the command, with a dead band
- ☐ **B** More error when turning than when driving straight
- ☐ **C** Removing any particle that would pass through a wall
- ☐ **D** Weighting each particle by how well it predicts the distance reading

> **Answer:** A, B, C. Anything known about how the robot moves goes in the motion update, and killing impossible particles costs one line. Weighting against a reading is the measurement update.

The task: move the cloud

Start every particle at zero, move them with the flow sensor plus their own noise as the robot drives at least 40 cm, plot `mean y` and `spread y`, and print both at the end.

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

DT, N = 0.1, 300
ys = [0.0] * N
```

The hint students can ask for: Start every particle at zero. Each tick, move every one of them by the measured flow, plus a small random amount of its own. The cloud tracks the robot and slowly spreads, which is dead reckoning uncertainty made visible.

A solution

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

DT = 0.1
N = 300
ys = [0.0] * N

forward(70)
for i in range(60):
    v = flow()[1]
    ys = [y + v * DT + random.gauss(0, 0.35) for y in ys]
    mean = sum(ys) / N
    spread = (sum((y - mean) ** 2 for y in ys) / N) ** 0.5
    plot("mean y", mean)
    plot("spread y", spread)
    wait(DT)
stop()
mean = sum(ys) / N
spread = (sum((y - mean) ** 2 for y in ys) / N) ** 0.5
print("mean y:", round(mean, 1))
print("spread y:", round(spread, 2))
```

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