ANSWERS

U7.4 Weighing the guesses

BugBotLab

Localisation · University · about 35 min

What this lesson is about

The measurement update: how likely is this reading if the robot were there?

- The sensor model is the interesting part
- Normalising, and the effective sample size
- Do not lie about sigma

Questions

7 marks in all

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

```
import math
sigma = 3.0
measured = 144.0
for predicted in (144.0, 147.0, 150.0):
    d = predicted - measured
    w = math.exp(-d * d / (2 * sigma * sigma))
    print(predicted, round(w, 3))
```

Answer:

```
144.0 1.0
147.0 0.607
150.0 0.135
```

A particle one sigma out gets exp(-1/2) = 0.607, and one two sigma out gets exp(-2) = 0.135. The weight falls off fast, which is how a reading rules places out.

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

```
weights = [0.4, 0.3, 0.2, 0.1]
neff = 1.0 / sum(w * w for w in weights)
print(round(neff, 2))
```

Answer:

```
3.33
```

The squares are 0.16 + 0.09 + 0.04 + 0.01 = 0.30, and 1 / 0.30 = 3.33. Four particles, but only about three and a third are really contributing.

3. A cloud has 100 particles. After normalising, two particles have weight 0.5 each and every other particle has weight 0. What is the effective sample size? [1 mark]

Answer: 2. neff = 1 / (0.5 squared + 0.5 squared) = 1 / 0.5 = 2. Only two particles are doing any work, whatever N is.

4. Why add a tiny floor such as 1e-12 to every weight? [1 mark]

- ☐ **A** One unexpected reading could otherwise make every weight zero, and normalising would divide by zero
- ☐ **B** It stops the weights summing to more than one
- ☐ **C** It makes the likelihood a proper Gaussian
- ☐ **D** It raises the effective sample size to N after every update

> **Answer:** A. With the floor the filter survives a surprise reading and recovers; without it the filter dies.

5. The sensor's real noise has a standard deviation of 3 cm, but the filter uses sigma = 0.5 cm. What happens? [1 mark]

- ☐ **A** The filter becomes over-confident and can throw away the true pose because of one unlucky reading
- ☐ **B** The filter becomes more accurate, because a smaller sigma sharpens the estimate
- ☐ **C** Nothing changes, because the weights are normalised afterwards
- ☐ **D** The effective sample size rises, because more particles fit the reading

> **Answer:** A. Normalising does not undo the shape: with a sigma that small almost every particle, including the true one on a noisy reading, gets negligible weight. Slightly larger than the truth is the safer error.

6. What is the quantity 1 / (sum of the squared normalised weights) called? [1 mark]

> **Answer:** effective sample size. The effective sample size is N when all weights are equal and 1 when one particle has all the weight. U7.5 uses it to decide when to resample.

7. The robot is 80 cm from the far wall and has turned 60 degrees towards the left wall. The depth sensor reads 152 cm. A filter weighs its cloud with predicted = 200 - y. What happens? [1 mark]

- ☐ **A** The guesses near y = 48 win, about 70 cm from the truth, because 200 - y is only right while the robot faces the far wall square on
- ☐ **B** Nothing goes wrong, because the weights are normalised afterwards
- ☐ **C** Every weight becomes exactly zero and the filter stops
- ☐ **D** The cloud stays where it was, because a slanted reading carries no weight

> **Answer:** A. At a slant the sensor looks further to reach a wall, so the reading is long, and a square-on model reads that as a robot much further from the far wall. Weigh the cloud only while the robot is square on, and skip readings while it turns.

The task: weigh the guesses

Scatter particles over the mat, take a reading, weight them, and print `best y:` (the weighted mean) and `neff:`.

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

N, SIGMA = 500, 3.0
particles = [random.uniform(0, 200) for i in range(N)]
```

The hint students can ask for: Each particle predicts what the sensor would read if the robot were there: 200 minus its y. Weight it by exp(-(predicted - measured)^2 / (2*sigma^2)), normalise the weights, and the weighted mean is your estimate. Neff is 1 over the sum of the squared normalised weights.

A solution

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

N = 500
SIGMA = 3.0
particles = [random.uniform(0, 200) for i in range(N)]

readings = []
for i in range(10):
    readings.append(distance())
    wait(0.1)
measured = sum(readings) / len(readings)

weights = []
for y in particles:
    predicted = 200 - y
    d = predicted - measured
    weights.append(math.exp(-d * d / (2 * SIGMA * SIGMA)) + 1e-12)
total = sum(weights)
weights = [w / total for w in weights]

best = sum(w * y for w, y in zip(weights, particles))
neff = 1.0 / sum(w * w for w in weights)
print("best y:", round(best, 1))
print("neff:", round(neff, 1))
```

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