ANSWERS

U3.6 An error budget

Odometry and drift · University · about 25 min

What this lesson is about

Predicting how far out you will be before you drive, which is what covariance is for.

- A budget for a 2 m straight run
- Adding the pieces up
- This is covariance, informally

Questions

6 marks in all

1. This adds up the lesson's error budget for a 2 m run with a calibrated gyro, then again with the noise term removed. What does it print? [1 mark]

```
import math
pieces = {"heading": 0.4, "scale": 6.0, "noise": 0.7, "slip": 2.0}
total = math.sqrt(sum(v * v for v in pieces.values()))
print("total:", round(total, 1), "cm")
print("without noise:", round(math.sqrt(total ** 2 - 0.7 ** 2), 1), "cm")
```

Answer:

```
total: 6.4 cm
without noise: 6.3 cm
```

$0.4^2 + 6^2 + 0.7^2 + 2^2 = 40.65$, whose square root is 6.4. Removing the 0.7 cm noise term leaves 6.3: the total moves by less than half a millimetre, so a term much smaller than the largest is not worth working on.

2. Two independent errors of 3 cm each. What is the total, in cm to one decimal place? [1 mark]

Answer: 4.2 (accept within 0.05). Independent errors add in quadrature: the square root of $3^2 + 3^2 = 18$ is 4.2 cm, not 6.

3. An uncalibrated gyro bias of 0.19 deg/s acts for 10 s while the robot drives 200 cm at a steady speed, so the heading error grows steadily from 0 to 1.9 degrees. Using d × e with the average heading error, how far to the side does it end up, to the nearest cm? [1 mark]

Answer: 3 (accept within 0.5). The heading error averages 0.95 degrees over the run, which is 0.0166 radians, and 200 × 0.0166 = 3.3 cm sideways. Using the final 1.9 degrees for the whole run would double it: the robot was only that far out at the very end.

4. Flow noise contributes 0.7 cm over a run of 100 steps. Following the random walk, what does it contribute over 400 steps, in cm? [1 mark]

Answer: 1.4 (accept within 0.05). Four times as many steps multiplies a random walk by the square root of 4, which is 2: 0.7 × 2 = 1.4 cm.

5. With the gyro calibrated, the budget reads heading 0.4 cm, scale 6 cm, noise 0.7 cm and slip 2 cm. What should be worked on next? [1 mark]

- ☐ **A** The flow scale error
- ☐ **B** The noise, by averaging more readings
- ☐ **C** The slip
- ☐ **D** The gyro bias again

> **Answer:** A. Scale is by far the largest term in quadrature. Halving the noise would change the total by less than a millimetre.

6. For that calibrated budget after a long straight run, what shape is the region the robot is likely to end up in? [1 mark]

- ☐ **A** An ellipse, longest along the direction of travel, because the scale error dominates
- ☐ **B** A circle, because the errors are added in quadrature
- ☐ **C** An ellipse, longest across the direction of travel, because heading error dominates
- ☐ **D** A line along the direction of travel, because heading error is zero after calibration

> **Answer:** A. Scale error stretches the estimate along the path and heading error throws it sideways. Here the 6 cm scale term outweighs the 0.4 cm heading term, so the ellipse is long and thin along the run.

The task: an error budget

Print what you expect the dead reckoning error to be over your run, as `predicted error:`, before you drive. Then drive at least 90 cm and print `actual error:`, the real gap between `odometry()` and `position()`.

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

DISTANCE = 100.0
```

The hint students can ask for: A rough budget is enough: a heading error of e degrees over a run of d centimetres puts you about d times e in radians off to one side, and the scale error adds a few percent of d. Print the number you expect before you drive, then the number you got.

A solution

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

DISTANCE = 100.0
HEADING_ERR = 0.7          # degrees: 0.19 deg/s over the 7 s drive, averaged
SCALE_ERR = 0.03           # the flow sensor reads about 3 percent high

sideways = DISTANCE * math.radians(HEADING_ERR)
along = DISTANCE * SCALE_ERR
print("predicted error:", round(math.hypot(sideways, along), 1))

forward(75)
wait(7.0)
stop()
wait(0.6)
ox, oy, oh = odometry()
tx, ty = position()
print("actual error:", round(math.hypot(ox - tx, oy - ty), 1))
```

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