ANSWERS

U4.6 Choosing the filter

Noise and filtering · University · about 25 min

What this lesson is about

Matching the cut-off to the signal you care about, by measurement rather than by taste.

- Ask what you are trying to keep
- A procedure that works
- Measuring instead of guessing
- The standing robot is only half the test

Questions

6 marks in all

1. The lesson gives alpha ~ 2 / (window + 1). What alpha gives an effective window of 9 readings? [1 mark]

> **Answer:** 0.2 (accept within 0.001). 2 / (9 + 1) = 0.2, matching the table in U4.3 where alpha 0.2 is roughly a window of 9.

2. This adds how far a robot at 20 cm/s travels during each filter's delay. What does it print? [1 mark]

```
DT, SPEED = 0.1, 20
for a in (0.05, 0.1, 0.3, 0.6):
    lag = (1 - a) / a * DT
    print(a, round(lag, 2), round(SPEED * lag, 1))
```

> **Answer:**
>
> ```
> 0.05 1.9 38.0
> 0.1 0.9 18.0
> 0.3 0.23 4.7
> 0.6 0.07 1.3
> ```
>
> The lags are 1.9, 0.9, 0.23 and 0.07 s, so the robot moves 38, 18, 4.7 and 1.3 cm before the filtered value catches up.

3. A stationary test shows alpha = 0.05 leaves the smallest spread. Why might it still be the wrong choice? [1 mark]

- ☐ **A** Standing still hides the delay; while driving it reports a distance the robot passed a second or more ago
- ☐ **B** Smaller spread always means more bias
- ☐ **C** alpha = 0.05 amplifies noise when the robot moves
- ☐ **D** Spread measured standing still is always an underestimate by a factor of two

> **Answer:** A. On a still robot more smoothing always looks better because nothing is changing. Test a filter on the motion it will actually be used for.

4. The signal you care about and the noise are at the same frequency. What does the lesson say? [1 mark]

- ☐ **A** No filter can separate them, so tuning will not help
- ☐ **B** Use a lower alpha
- ☐ **C** Use a median filter instead
- ☐ **D** Use a higher alpha

> **Answer:** A. A filter separates by frequency. When signal and noise overlap in frequency there is nothing to separate them by, and recognising that early saves a lot of tuning.

5. Put the lesson's filter-choosing procedure in order. [1 mark]

Number the lines 1 to 4 to put them in the right order.

___ `Turn that ratio into an effective window and an alpha`

___ `Decide how much noise the controller's input can tolerate`

___ `Check the resulting delay against the system's time constant`

___ `Measure the noise sigma with the robot still`

> **Answer:**
>
> ```
> Measure the noise sigma with the robot still
> Decide how much noise the controller's input can tolerate
> Turn that ratio into an effective window and an alpha
> Check the resulting delay against the system's time constant
> ```
>
> Measure, decide the requirement, derive alpha, then check the cost in delay against the plant.

6. The delay of the alpha you need is too large for the system's time constant. What does the lesson say the answer is? [1 mark]

- ☐ **A** A better sensor or a slower controller
- ☐ **B** A more sophisticated filter with the same alpha
- ☐ **C** A median filter in series
- ☐ **D** Measuring sigma again

> **Answer:** A. If the noise needs more smoothing than the loop can afford in delay, the fix is upstream (less noise) or downstream (a controller that tolerates the delay).

The task: tune the filter

Standing still, run at least three alphas at once, print the spread each one leaves, and print `best alpha:`.

```
from bugbot import *
connect()

ALPHAS = [0.05, 0.1, 0.3, 0.6]
```

The hint students can ask for: Run the same still robot through three or four filters at once, each with a different alpha, and measure the spread of each filtered signal. The smallest spread wins here, because nothing is moving; U4.5 is where that stops being the whole story.

A solution

```
from bugbot import *
connect()

DT = 0.1
ALPHAS = [0.05, 0.1, 0.3, 0.6]
state = {a: distance() for a in ALPHAS}
history = {a: [] for a in ALPHAS}

for i in range(120):
    raw = distance()
    for a in ALPHAS:
        state[a] = a * raw + (1 - a) * state[a]
        history[a].append(state[a])
    wait(DT)

best, best_spread = ALPHAS[0], 1e9
for a in ALPHAS:
    vals = history[a][20:]
    mean = sum(vals) / len(vals)
    spread = (sum((v - mean) ** 2 for v in vals) / len(vals)) ** 0.5
    print("alpha", a, "spread", round(spread, 2))
    if spread < best_spread:
        best, best_spread = a, spread
print("best alpha:", best)
```

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