ANSWERS

U12.6 The reality gap, and honest evaluation

Learning, and the capstone · University · about 40 min

What this lesson is about

Why a policy tuned in one world fails in another, and how to report what it does without lying.

- Why a tuned policy breaks
- Domain randomisation
- What else closes the gap
- Honest evaluation

Questions

6 marks in all

1. A braking policy is run five times with the noise turned up. What does this print for the mean and spread of the gaps it left? [1 mark]

```
gaps = [31.0, 27.5, 36.2, 29.1, 22.4]
mean = sum(gaps) / len(gaps)
spread = (sum((g - mean) ** 2 for g in gaps) / len(gaps)) ** 0.5
print(round(mean, 1), round(spread, 1), round(max(abs(g - 30.0) for g in gaps), 1))
```

Answer:

```
29.2 4.5 7.6
```

The mean of 29.2 cm looks close to the 30 cm target, but the spread is 4.5 cm and the worst run is 7.6 cm out. The spread is the story.

2. A braking threshold tuned with no sensor noise stops the robot too early once noise is added. Why? [1 mark]

☐ **A** The first time a noisy reading dips past the threshold counts as a crossing, before the true distance gets there

☐ **B** Noise makes the average reading smaller

☐ **C** The robot drives faster with noise on

☐ **D** The threshold was tuned on the wrong mat

Answer: A. The mean is still right, but a policy that brakes on one reading meets the whole spread, and the early side of it fires first.

3. A policy is tuned across three noise levels at once instead of one. What is the trade? [1 mark]

☐ **A** It is worse in the quiet world than a policy tuned there, in exchange for performance you can predict across worlds

☐ **B** It is better in every world

☐ **C** It needs no evaluation afterwards

☐ **D** It removes the need for feedback

Answer: A. Domain randomisation gives up peak performance against one model you do not fully believe, as regularisation and gain margin do.

4. Why is the first successful run a biased sample of how well a policy works? [1 mark]

- ☐ **A** You stop looking when it works, so the run you report is more likely a good one than a typical one
- ☐ **B** The first run is always slower
- ☐ **C** The battery is fuller on the first run
- ☐ **D** It is not biased, only noisy

> **Answer:** A. Run it n times and report the mean, the spread, n itself and the failures.

5. Which are honest ways to report a learned policy? [1 mark]

Tick every answer that is true.

- ☐ **A** Tune at one noise level and report at another
- ☐ **B** Report the number of runs and include the failures
- ☐ **C** Compare it with the obvious hand-written feedback controller
- ☐ **D** Report the worst case when the worst case is what matters
- ☐ **E** Drop runs where the robot hit the wall as outliers
- ☐ **F** Report the best of ten runs

> **Answer:** A, B, C, D. Hold out conditions, not just data, and say what the baseline is. Discarding failures or quoting the best run measures your selection, not the policy.

6. If you can choose only one defence against the reality gap, which does the lesson recommend? [1 mark]

- ☐ **A** Prefer feedback, because a closed loop tolerates a model that is 20 percent wrong
- ☐ **B** Randomise the simulator more widely
- ☐ **C** Tune for longer in the lab
- ☐ **D** Use a larger hypothesis class

> **Answer:** A. An open loop is only as good as its model. A closed loop measures the result and corrects it.

The task: tune it, then stress it

Tune the reading at which the robot starts braking, so that it ends up in the green band, 30 cm from the wall. Plot `score` as you tune. Then turn the noise up with `set_noise()`, run the tuned policy three times, and print `mean:` and `spread:` of the gaps it left. Put the noise back where it was for the run you hand in. Start every trial from the same distance, and start it well clear of the threshold you are testing. A trial that begins two centimetres above the braking point triggers on the first unlucky reading and scores nonsense, and the search will believe it.

```
from bugbot import *
connect()

DT = 0.1
CRUISE = 85
GAP = 30.0
START_GAP = 55.0
```

The hint students can ask for: The robot should end up in the green band, 30 cm from the wall. Tune the reading at which it starts braking, by trials: drive at the wall, stop, measure the gap you left, score it, back off well clear of the threshold and try another one. Then turn the noise up with set_noise(), run the tuned policy a few times, and report the mean and the spread. Put the noise back to where you tuned it before the run you hand in.

A solution

```
from bugbot import *
connect()

DT = 0.1
CRUISE = 85
GAP = 30.0                 # where the robot should end up, in cm from the wall
START_GAP = 55.0

def turn_cmd():
    """Square to the wall, so that driving forward does not quietly become driving
diagonally."""
    err = (imu()[0] + 180) % 360 - 180
    if abs(err) < 2.0:
        return 0
    cmd = max(18.0, min(30.0, abs(1.5 * err)))     # below 15 the drive does nothing at all
    return -cmd if err > 0 else cmd

def gap_now(n=8):
    """The mean of n depth readings: one reading on its own has centimetres of noise on
it."""
    vs = []
    for i in range(n):
        vs.append(distance())
        wait(DT)
    return sum(vs) / len(vs)

def settle_at(target):
    """Park roughly this far from the wall, so the next trial starts where the last one
did."""
    for i in range(120):
        err = gap_now(2) - target
        if abs(err) < 3:
```

```
            break
        cmd = max(20.0, min(100.0, abs(3.0 * err)))
        drive(cmd if err > 0 else -cmd, 0, turn_cmd())
        wait(DT)
    stop()
    wait(0.3)

def run(threshold):
    """Drive at the wall and brake when the filtered reading passes the threshold. Returns
the gap left."""
    smooth = gap_now(3)
    for i in range(200):
        drive(CRUISE, 0, turn_cmd())
        smooth = 0.7 * smooth + 0.3 * distance()
        if smooth < threshold:
            break
        wait(DT)
    stop()
    wait(0.6)
    return gap_now()

settle_at(START_GAP)
threshold, step = GAP, 6.0
best = abs(run(threshold) - GAP)
for i in range(6):
    settle_at(START_GAP)
    trial = threshold + step
    s = abs(run(trial) - GAP)
    if s < best:
        threshold, best = trial, s
    else:
        step = -step * 0.6
    plot("score", best)
    plot("threshold", threshold)
print("threshold:", round(threshold, 1), "scored", round(best, 2))

# the same policy in a rougher world than the one it was tuned in
set_noise(2.0, depth=6.0)
gaps = []
for i in range(3):
    settle_at(START_GAP)
    gaps.append(run(threshold))
mean = sum(gaps) / len(gaps)
spread = (sum((g - mean) ** 2 for g in gaps) / len(gaps)) ** 0.5
print("mean:", round(mean, 1))
print("spread:", round(spread, 2))

# and the run that is handed in, back at the noise level the task states
set_noise(1.0, depth=3.0)
settle_at(START_GAP)
run(threshold)
```

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