ANSWERS

6.3 Losing the line

Seeing more · Robot club · about 20 min

What this lesson is about

Gaps: carry on, then search with a state machine.

- Keep going first
- Then search
- Choosing the first direction

Questions

7 marks in all

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

```
readings = [[160, 0], [], [], [170, 2], [], [], [], []]
lost = 0
for seen in readings:
    if seen:
        lost = 0
    else:
        lost += 1
print(lost)
```

Answer:

```
4
```

Every sighting resets the counter. After the last sighting come four empty readings, so `lost` is 4.

2. When the line disappears, why carry straight on for a while before searching? [1 mark]

- ☐ **A** Most gaps are short, so the line is usually just ahead
- ☐ **B** The camera needs time to switch detector
- ☐ **C** Searching uses more battery
- ☐ **D** The line always restarts straight ahead

Answer: A. It is the cheapest fix and it works for most gaps. On this mat it does not, which is why the counter leads to a search.

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

```
for sweep in [0, 14, 15, 29, 30, 45]:
    print(sweep, 40 if (sweep // 15) % 2 == 0 else -40)
```

> **Answer:**
>
> ```
> 0 40
> 14 40
> 15 -40
> 29 -40
> 30 40
> 45 -40
> ```
>
> `sweep // 15` counts blocks of 15 ticks. Even blocks slide right at 40, odd blocks slide left at -40.

4. The search slides from side to side instead of turning. Why? [1 mark]

- ☐ **A** The robot keeps its heading, so it already points the right way when it finds the line
- ☐ **B** Sliding is faster than turning
- ☐ **C** The line detector cannot work while turning
- ☐ **D** Turning would make `line()` return an error

> **Answer:** A. Strafing searches without changing which way you face, so following can pick up straight away.

5. Which way should the search try first? [1 mark]

- ☐ **A** The way the last `angle` said the line was heading
- ☐ **B** Always right
- ☐ **C** Always left
- ☐ **D** Backwards, to where the line was

> **Answer:** A. A line heading right when it vanished probably carries on to the right. Store the angle before it is gone.

6. The follower switches to search when `lost > 8`, adding 1 to `lost` on each tick with no line. On which tick without a line does it switch? [1 mark]

> **Answer:** 9. `lost > 8` is first true when `lost` is 9. At ten ticks a second, that is just under one second.

7. The task starter does `if not follow_step(60, 0.4): break`. What happens at the gap? [1 mark]

- ☐ **A** The robot stops at the first gap and never reaches the zone
- ☐ **B** It searches for the line
- ☐ **C** It follows the line through the gap
- ☐ **D** It turns round

> **Answer:** A. Breaking the moment `line()` is empty is the follower that is stuck at the first gap. Count, then search.

The task: find the line again

Follow the line through the gap to the green zone, staying within 6 cm of it for at least 60% of the run.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def follow_step(speed=60, gain=0.4):
    # [cx, angle] of the line ahead, or [] if none
    seen = line()
    if not seen:
        return False
    cx, angle = seen
    # forward, sideways, rotation: -100 to 100 each, until the next command
    drive(speed, 0, (cx - 160) * gain)
    return True

# camera: the line detector
set_cv("line")
while position()[1] < 85:
    if not follow_step(60, 0.4):
        # leave the loop
        break
    # pause 0.1 s (the robot keeps doing what it was told)
    wait(0.1)
# all motors off
stop()
```

The hint students can ask for: The line has a gap. When it disappears, keep going a little, then sweep left and right until it is back. Finish in the green zone.

A solution

```
from bugbot import *
connect()
def follow_step(speed=60, gain=0.4):
    # one tick of line following: steer from where the line crosses the picture
    seen = line()
    if not seen:
        return False
    cx, angle = seen
    error = cx - 160                      # pixels left (-) or right (+) of centre
    drive(speed, 0, error * gain)
    return True
set_cv('line')
lost = 0
state = 'follow'
while position()[1] < 85:
    if state == 'follow':
        if follow_step(60, 0.4):
            lost = 0
        else:
            lost += 1
            forward(40)                   # keep going a little: gaps are usually short
            if lost > 8:
                state = 'search'
                sweep = 0
```

```
    elif state == 'search':
        if line():
            state = 'follow'
        else:
            sweep += 1
            drive(25, 40 if (sweep // 15) % 2 == 0 else -40, 0)   # creep forward while
sweeping side to side
    wait(0.1)
stop()
```

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