ANSWERS

6.2 Following a line

Seeing more · Robot club · about 20 min

What this lesson is about

cx to rotation, the angle as a look-ahead, speed on the straights.

- One step
- The whole line
- Using the angle
- Speed on the straights

Questions

7 marks in all

1. `follow_step` uses rotation `(cx - 160) * 0.4`. What rotation does a line at `cx = 100` give? [1 mark]

> **Answer:** -24. 100 minus 160 is -60, and -60 times 0.4 is -24.

2. The rotation for a line at `cx = 100` is negative. What does the robot do? [1 mark]

- ☐ **A** Turns anticlockwise, towards the line on its left
- ☐ **B** Turns clockwise, away from the line
- ☐ **C** Slides left without turning
- ☐ **D** Stops, because the error is negative

> **Answer:** A. Negative rotation is anticlockwise, and the line is on the left, so the robot turns towards it.

3. A student writes `drive(60, 0, (160 - cx) * 0.4)`. What happens? [1 mark]

- ☐ **A** The robot steers away from the line and loses it
- ☐ **B** It follows the line more smoothly
- ☐ **C** It follows the line backwards
- ☐ **D** Nothing changes, the order does not matter

> **Answer:** A. Swapping the subtraction flips the sign, so every correction pushes the line further off-centre.

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

```
def rotation(cx, angle):
    return (cx - 160) * 0.3 + angle * 1.5

print(rotation(160, 10))
print(rotation(200, -8))
```

> **Answer:**
>
> ```
> 15.0
> 0.0
> ```
>
> The first line is centred but bending right, so the angle term starts the turn early. In the second, the two terms cancel out.

5. Adding `angle * 1.5` to the rotation is the beginning of which kind of control? [1 mark]

- ☐ **A** Derivative: it reacts to where the error is heading
- ☐ **B** Bang-bang: full on or full off
- ☐ **C** Integral: it adds up past errors
- ☐ **D** Open loop: it ignores the sensor

> **Answer:** A. `cx` says where the line is, `angle` says where it is going, so the robot starts turning before it drifts off-centre.

6. At each bend the line swings well off-centre, up to 85 pixels, before the robot has turned. What does that mean? [1 mark]

- ☐ **A** Nothing is wrong: a P controller only turns once there is an error, and the line is still in view
- ☐ **B** The camera is badly aligned
- ☐ **C** The gain is negative
- ☐ **D** The robot has lost the line

> **Answer:** A. To turn at all, a proportional controller needs some error, so it always lags a bend a little. That is fine as long as the line stays in view.

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

```
for angle in [0, 5, 20, -25]:
    speed = 80 if abs(angle) < 10 else 40
    print(angle, speed)
```

> **Answer:**
>
> ```
> 0 80
> 5 80
> 20 40
> -25 40
> ```
>
> A small angle means a straight, so go fast. `abs` makes a bend to the left count the same as one to the right.

The task: follow the line

Follow the line to the green zone, staying within 6 cm of it for at least 85% of the run.

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

# drive forward at 60 for 90 cm, then stop
forward(60, distance=90)
```

The hint students can ask for: Follow the line to the green zone at the far end, staying within 6 cm of it. Steer from cx: the line left of centre means turn left.

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')
while position()[1] < 85:
    if not follow_step(60, 0.4):
        forward(30)
    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.