ANSWERS

6.8 Project: the delivery

Seeing more · Robot club · about 30 min

What this lesson is about

Follow a line to a marker and back through traffic.

- The plan, drawn
- Two detectors, one camera
- Turning round
- The whole program
- Where next

Questions

6 marks in all

1. Why does the delivery program keep calling `set_cv` inside the loop? [1 mark]

- ☐ **A** The camera runs one detector at a time, and it needs both lines and tags
- ☐ **B** `set_cv` refreshes the picture
- ☐ **C** Each detector only works once
- ☐ **D** It makes the robot drive straighter

> **Answer:** A. Keep the line detector as the default and borrow the tag detector briefly each tick.

2. Put the delivery states in order. [1 mark]

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

___ `turn`
___ `done`
___ `home`
___ `out`

> **Answer:**
>
> ```
> out
> turn
> home
> done
> ```
>
> Out to marker 7, turn round, follow the line home. The give way rule sits above all of them.

3. The give way check comes after the state logic in each tick. Why there? [1 mark]

- ☐ **A** Its `stop()` overrides whatever the state asked for
- ☐ **B** It has to run before `set_cv("line")`
- ☐ **C** The state logic would undo the stop otherwise
- ☐ **D** Checks at the end of a loop run faster

> **Answer:** A. The last command in a tick is the one that counts. That is the subsumption idea from lesson 5.4 in three lines.

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

```
def facing_back(h, h0):
    return abs((h - h0 - 180 + 180) % 360 - 180) < 20

for h in [0, 90, 170, 185, 200]:
    print(h, facing_back(h, 0))
```

Answer:

```
0 False
90 False
170 True
185 True
200 False
```

Facing back from heading 0 means near 180. Within 20 degrees counts, so 170 and 185 pass, and 200 is just outside.

5. What is `h0` for? [1 mark]

- ☐ **A** It remembers the heading when marker 7 was found, so the robot knows when it faces back
- ☐ **B** It is the heading at the start of the program
- ☐ **C** It stores the line's angle
- ☐ **D** It counts ticks in the turn state

Answer: A. Turning round means reaching the opposite of the heading you had at the marker, with the line back in view.

6. A student reads the tags with `set_cv("apriltag")` but forgets to switch back to `set_cv("line")`. What will `follow_step` do? [1 mark]

- ☐ **A** Return `False` every tick, because the line detector is off
- ☐ **B** Follow the line as normal
- ☐ **C** Follow the tags instead of the line
- ☐ **D** Raise an error on the first tick

Answer: A. Only the chosen detector gives results, so `line()` finds nothing. Switch back straight after borrowing the tag detector.

The task: the delivery

Follow the line to marker 7, print `found 7`, and follow it home without touching the red robot.

```
# 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")
# again and again, for ever
while True:
    if not follow_step(55, 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: Follow the line to marker 7, print `found 7`, then follow it back to home, giving way to the red robot both times.

A solution

```
from bugbot import *
connect()
last_error = 0

def follow_step(speed=60, gain=0.4):
    # one tick of line following; when the line drops out of view, creep on turning the way
it went
    global last_error
    seen = line()
    if not seen:
        drive(20, 0, last_error * gain + (25 if last_error >= 0 else -25))
        return False
    cx, angle = seen
    last_error = cx - 160                  # pixels left (-) or right (+) of centre
    if abs(last_error) > 40:
        speed = min(speed, 35)             # the line is swinging across the picture: a
corner, slow down
    drive(speed, 0, last_error * gain)
    return True
set_cv('line')
state = 'out'
for tick in range(690):
    x, y = position()
```

```
    if state == 'out':
        set_cv('apriltag')
        tags = [t for t in apriltags() if t[0] == 7]
        set_cv('line')
        if tags and tags[0][3] < 12:
            print('found 7')
            state = 'turn'
            h0 = heading()
        else:
            follow_step(55, 0.4)
    elif state == 'turn':
        turn_right(60)
        facing_back = abs((heading() - h0 - 180 + 180) % 360 - 180) < 20   # within 20
degrees of the way we came
        if facing_back and line():
            state = 'home'
    elif state == 'home':
        if x < -20 and y < 25:
            break
        follow_step(55, 0.4)
    # give way: robot 102 crossing close ahead
    set_cv('apriltag')
    others = [t for t in apriltags() if t[0] == 102 and t[3] < 25]
    set_cv('line')
    if others or (state == 'home' and distance() < 12):
        stop()
    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.