ANSWERS

9.6 Sharing what you see

Talking to each other · Robot club · about 20 min

What this lesson is about

Telling a robot without a camera where the ball is.

- Where is the ball?
- Say what, not what you see

Questions

6 marks in all

1. Which message is useful to the Fetcher, which is standing somewhere else? [1 mark]

- ☐ **A** `ball at 20,30`
- ☐ **B** `red blob at cx 210, 40 px wide`
- ☐ **C** `ball in my camera`
- ☐ **D** `turn right 15`

> **Answer:** A. Camera readings only make sense from where you stand. A position on the shared map means the same to everyone.

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

```
import math

def bearing_of(cx):
    return (cx - 160) * 120 / 320

x, y, heading = 10, 0, 0
cx, width = 160, 23
dist = 4 * 92 / max(width, 1)
h = math.radians(heading + bearing_of(cx))
bx, by = x + math.sin(h) * dist, y + math.cos(h) * dist
print(f"ball at {bx:.0f},{by:.0f}")
```

> **Answer:**
>
> ```
> ball at 10,16
> ```
>
> The ball is centred, so straight ahead, and 16 cm away. Facing up the mat from (10, 0), that is (10, 16).

3. The blob's `cx` is 200. Using `bearing_of(cx) = (cx - 160) * 120 / 320`, what is its bearing in degrees? [1 mark]

> **Answer:** 15. (200 - 160) x 120 / 320 = 40 x 0.375 = 15 degrees to the right.

4. Why is the direction to the ball `heading() + bearing_of(cx)`? [1 mark]

- ☐ **A** The bearing is measured from where the camera points, so adding the heading turns it into a direction on the mat
- ☐ **B** To make the number positive
- ☐ **C** Because the camera is on the back of the robot
- ☐ **D** The heading is always zero

> **Answer:** A. Heading says which way the robot faces on the mat; bearing says how far off that the ball is.

5. Why does the code use `max(width, 1)` when working out the distance? [1 mark]

- ☐ **A** So it never divides by zero
- ☐ **B** So the ball is never closer than 1 cm
- ☐ **C** To round the width to a whole number
- ☐ **D** Because the width is measured in metres

> **Answer:** A. A width of 0 would crash the division. Using at least 1 keeps the program running.

6. The robot faces up the mat and the red ball is right of centre in the picture. Roughly where is the ball? [1 mark]

- ☐ **A** Up the mat and to the right, at larger x
- ☐ **B** Up the mat and to the left, at smaller x
- ☐ **C** Behind the robot
- ☐ **D** Exactly straight ahead

> **Answer:** A. A positive bearing is clockwise, which is to the right when facing up the mat.

The task: share what you see

Find the red ball with the camera, work out where it is, and send `ball at <x>,<y>` within 10 cm of the truth. The Fetcher does the rest.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()
# camera: look for red blobs
set_cv('blob', 'red')
# until a blob of that colour is in view
while not blobs():
    # spin clockwise on the spot at 40
    turn_right(40)
    # pause 0.1 s (the robot keeps doing what it was told)
    wait(0.1)
# all motors off
stop()
print(blobs())
```

The hint students can ask for: Find the red ball with the camera, work out where it is from its bearing and its width, and send `ball at <x>,<y>` (from your start). The Fetcher cannot see it, but it can hear you.

A solution

```
from bugbot import *
connect()
import math

def bearing_of(cx):
    return (cx - 160) * 120 / 320

set_cv("blob", "red")
while not blobs():
    turn_right(40)
    wait(0.1)
stop()
wait(0.3)
b = blobs()[0]
width = b[5] - b[3]
dist = 4 * 92 / max(width, 1)                    # a 4 cm ball, 92 px focal length
h = math.radians(heading() + bearing_of(b[0]))
px, py = position()
bx, by = px + math.sin(h) * dist, py + math.cos(h) * dist
send(f"ball at {bx:.0f},{by:.0f}")
print(f"ball at {bx:.0f},{by:.0f}")
wait(5)
print(messages())
```

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