ANSWERS

4.2 Where is it?

Vision · Robot club · about 15 min

What this lesson is about

Pixels to degrees, and a controller that turns to face a tag.

- Pixels to degrees
- Turn to face it
- Stopping cleanly

Questions

6 marks in all

1. Using `bearing = (cx - 160) * 120 / 320`, what is the bearing in degrees of a tag at `cx = 240`? [1 mark]

> **Answer:** 30 (accept within 0.5). 240 minus 160 is 80 pixels, and the formula shares 120 degrees evenly over 320 pixels, 0.375 each, so 30 degrees to the right. The true bearing is nearer 41: the even share reads low away from the centre, which a controller does not mind.

2. Using `bearing = (cx - 160) * 120 / 320`, where is a tag at `cx = 100`? [1 mark]

- ☐ **A** To the left, about 22 degrees
- ☐ **B** To the right, about 22 degrees
- ☐ **C** Straight ahead
- ☐ **D** To the left, about 100 degrees

> **Answer:** A. Under 160 is left. (100 - 160) * 0.375 is -22.5, and negative bearings are to the left.

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

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

for cx in [0, 160, 320]:
    print(bearing_of(cx))
```

> **Answer:**
>
> ```
> -60.0
> 0.0
> 60.0
> ```
>
> The edges of the picture are 60 degrees either side of straight ahead, because the camera sees 120 degrees in total.

4. The loop runs `drive(0, 0, bearing_of(cx) * 3)` and the tag is at `cx = 220`. What does the robot do? [1 mark]

- ☐ **A** Turns clockwise, towards the tag
- ☐ **B** Turns anticlockwise, away from the tag
- ☐ **C** Drives forward
- ☐ **D** Slides right without turning

> **Answer:** A. The bearing is positive, so the rotation is positive, which is clockwise. The tag slides towards the middle of the picture.

5. A student writes `error = (160 - cx) * 120 / 320` and uses `drive(0, 0, error * 3)`. What happens? [1 mark]

- ☐ **A** The robot turns away from the tag until it loses it
- ☐ **B** The robot faces the tag, just more slowly
- ☐ **C** Nothing different, because the sign does not matter
- ☐ **D** The robot drives towards the tag

> **Answer:** A. Reading `cx` the wrong way round flips the sign, so the controller pushes the tag further off-centre instead of pulling it in.

6. Once the error is under 1.5 degrees, the program stops, waits 0.3 s and checks again. Why check again? [1 mark]

- ☐ **A** The robot coasts after stopping, so the tag may have slid off centre
- ☐ **B** The camera needs 0.3 s to switch on
- ☐ **C** The tag might have changed its id
- ☐ **D** To let the battery recover

> **Answer:** A. Stopping is not instant. Settle, then check that you really are still centred before you finish.

The task: face the marker

Turn until marker 3 is in the middle of the picture, then print `centred`. Do not drive.

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

# camera: the tag detector
set_cv("apriltag")
print(apriltags())
```

The hint students can ask for: Marker 3 is off to the right of the picture. Turn until it sits in the middle (cx near 160), then print `centred`. Do not drive.

A solution

```
from bugbot import *
connect()
def bearing_of(cx):
    return (cx - 160) * 120 / 320      # pixels to degrees, 120 degree view across 320
pixels
set_cv('apriltag')
while True:
    tags = apriltags()
    if not tags:
        turn_right(30)
        wait(0.1)
        continue
    error = bearing_of(tags[0][1])
    if abs(error) < 1.5:
        stop()
        wait(0.3)                  # settle, then look again
        if abs(bearing_of(apriltags()[0][1])) < 1.5:
            break
        continue
    drive(0, 0, max(-30, min(30, error * 3)))
    wait(0.05)
stop()
print('centred')
```

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