ANSWERS

U11.7 Project: find the one that is green

Vision · University · about 50 min

What this lesson is about

Three identical shapes, one job, and a task the depth sensor cannot begin.

- The job
- What is worth thinking about
- A shape to start from
- What comes next

Questions

6 marks in all

1. Three identical balls differ only in colour. Why can the depth sensor not pick out the green one? [1 mark]

- ☐ **A** It measures a property of space, and identity lives in the surfaces, which only the camera sees
- ☐ **B** Its range is too short to reach the balls
- ☐ **C** Its readings are too noisy
- ☐ **D** It cannot measure bearings

> **Answer:** A. To the depth sensor they are three identical bumps. No cleverness with range data recovers a property it never measured.

2. The survey finds three 6 cm balls. What does this print? [1 mark]

```
import math
F, R = 92.4, 3.0
seen = {"red": (100, 14), "green": (200, 28), "blue": (250, 9)}
for colour, (cx, width) in seen.items():
    got = 2 * R * F / width, math.degrees(math.atan((cx - 160) / F))
    print(colour, "%.0f cm at %.0f degrees" % got)
```

> **Answer:**
>
> ```
> red 40 cm at -33 degrees
> green 20 cm at 23 degrees
> blue 62 cm at 44 degrees
> ```
>
> Range is 554.4 / width: 39.6, 19.8 and 61.6 cm. Bearings are atan((cx - 160) / 92.4): -33, 23 and 44 degrees.

3. Why turn to the green ball's bearing and servo on its apparent size, rather than work out its coordinates and drive there open loop? [1 mark]

- ☐ **A** The range is far worse than the bearing, and closing the loop on size lets the measurement improve as the robot gets closer
- ☐ **B** Coordinates cannot be computed from a bearing and a range
- ☐ **C** Open loop driving is not allowed on the mat
- ☐ **D** Servoing is faster than driving a straight line

> **Answer:** A. Range from size degrades as d squared, so the survey range is the weakest number you have. Near the ball it is a precision instrument.

4. Why must the survey be a separate phase before the approach? [1 mark]

- ☐ **A** The camera runs one detector at a time, so it cannot watch for red while servoing on green
- ☐ **B** The balls move once the robot sets off
- ☐ **C** The depth sensor must be switched off while the camera runs
- ☐ **D** The bearings change units when the robot moves

> **Answer:** A. A small camera pipeline has one frame buffer and one configuration. Switching detectors costs time, so plan the survey and commit.

5. The approach should stop 20 cm from a 6 cm ball. About how many pixels wide should it look, to 1 decimal place, with f = 92.4? [1 mark]

> **Answer:** 27.7 (accept within 0.4). w = 92.4 x 6 / 20 = 27.7 pixels, about 28.

6. Which choices help the robot stop short of the ball without touching it? [1 mark]

Tick every answer that is true.

- ☐ **A** Use the dead band trick rather than a large forward gain
- ☐ **B** Stop on a band of widths rather than one exact width
- ☐ **C** Raise the forward gain so it settles before it can overshoot
- ☐ **D** Plan the approach from the survey range and drive it without looking

> **Answer:** A, B. Overshoot comes from a high gain in a slow loop. A band stops the loop hunting, and driving blind on a poor range is how balls get hit.

The task: find the one that is green

Survey all three balls and print `red:`, `green:` and `blue:`, the range in centimetres to each. Then drive to the green ball and stop within reach of it without touching it. `position()` is not allowed: the robot has to find it by looking.

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

F, R = 92.4, 3.0
TARGET = "green"
```

The hint students can ask for: Three balls, all 6 cm across, all the same size and shape to a depth sensor. Survey first: the camera runs one detector at a time, so set each colour in turn, give it a moment, and record the range and bearing of what you see. Print all three. Then set the detector back to the colour you were sent for and servo onto it as in U11.6, stopping about 20 cm short.

A solution

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

F = 92.4
R = 3.0                              # every ball is 6 cm across
STANDOFF = 22.0
WANT = 2 * R * F / STANDOFF
TARGET = "green"

def survey(colour):
    """Range and bearing of the largest blob of one colour, or None."""
    set_cv("blob", colour)
    wait(0.3)
    seen = blobs()
    if not seen:
        return None
    cx, cy, area, x0, y0, x1, y1, aspect = seen[0]
    return 2 * R * F / (x1 - x0), math.degrees(math.atan((cx - 160) / F))

for colour in ("red", "green", "blue"):
    got = survey(colour)
    if got is None:
        print(colour + ": not in view")
    else:
        print("%s: %.1f cm, bearing %.1f" % (colour, got[0], got[1]))

set_cv("blob", TARGET)
wait(0.2)
last = 1.0
for tick in range(700):
    seen = blobs()
    if not seen:
        drive(0, 0, 25 * last)
        wait(0.1)
        continue
    cx, cy, area, x0, y0, x1, y1, aspect = seen[0]
```

```
    w = x1 - x0
    err = cx - 160

    last = 1.0 if err > 0 else -1.0
    rot = max(-55, min(55, 0.35 * err))
    if abs(err) < 6:
        rot = 0
    elif abs(rot) < 17:
        rot = 17 * (1 if rot > 0 else -1)

    gap = WANT - w
    fwd = max(-40, min(45, 4.0 * gap))
    if abs(gap) < 1.5:
        fwd = 0
    elif abs(fwd) < 18:
        fwd = 18 * (1 if fwd > 0 else -1)
    if abs(err) > 60:
        fwd = 0

    drive(fwd, 0, rot)
    plot("cx", cx)
    plot("width", w)
    wait(0.1)
    if fwd == 0 and rot == 0:          # both errors inside their bands: arrived
        break
stop()
wait(0.5)
print("arrived at the", TARGET, "ball")
```

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