ANSWERS

4.3 Visual servoing

Vision · Robot club · about 20 min

What this lesson is about

Steer from cx, speed from distance: docking at a tag.

- Two errors from one tag
- Steer while driving
- Speed from distance
- When the tag disappears

Questions

6 marks in all

1. A tag gives two errors at once. Which pairing does the docking loop use? [1 mark]

- ☐ **A** `cx` sets the rotation, `distance` sets the forward speed
- ☐ **B** `cx` sets the forward speed, `distance` sets the rotation
- ☐ **C** `cy` sets the rotation, `id` sets the speed
- ☐ **D** `distance` sets both

> **Answer:** A. Off-centre means turn, far away means drive. Two errors, two outputs, every tick.

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

```
stop_at = 13
for dist in [40, 20, 14]:
    speed = max(20, min(70, (dist - stop_at) * 3))
    print(speed)
```

> **Answer:**
>
> ```
> 70
> 21
> 20
> ```
>
> 40 gives 81, clamped down to 70. 20 gives 21. 14 gives 3, clamped up to 20.

3. With `stop_at = 13` and `speed = max(20, min(70, (dist - stop_at) * 3))`, what speed does the robot use when the tag is 25 cm away? [1 mark]

> **Answer:** 36. (25 - 13) * 3 is 36, which is between 20 and 70, so the clamp leaves it alone.

4. Why does the speed have a lower limit of 20, the `max(20, ...)`? [1 mark]

- ☐ **A** So the robot does not crawl to a halt before it reaches the stopping distance
- ☐ **B** So it never drives faster than 20
- ☐ **C** So it can reverse if it gets too close
- ☐ **D** Because the camera cannot see below 20 cm

> **Answer:** A. Close to the tag, `(dist - stop_at) * 3` is tiny, too small to move the robot. The minimum keeps it creeping in.

5. Near the tag, the camera loses it for one frame and the loop does `if not tags: break`. Why is that a poor choice? [1 mark]

- ☐ **A** The robot stops dead even though the tag is almost certainly still there
- ☐ **B** `break` restarts the loop from the top
- ☐ **C** It makes the robot spin forever
- ☐ **D** It is an error to break out of a while loop

> **Answer:** A. Losing a tag for a frame is normal. Better to keep going for a few ticks, or turn slowly towards where it was last seen.

6. Put one tick of the visual servoing loop in order. [1 mark]

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

___ `Read the tags with `apriltags()``

___ `Work out the bearing error and the distance error`

___ `Send one `drive` with a speed and a rotation`

___ `Wait 0.1 s, then go round again`

> **Answer:**
>
> ```
> Read the tags with `apriltags()`
> Work out the bearing error and the distance error
> Send one `drive` with a speed and a rotation
> Wait 0.1 s, then go round again
> ```
>
> Read, two errors, two outputs, repeat. That is the whole lesson in four lines.

The task: dock at the marker

Drive to marker 1 and stop about 12 cm in front of it, square to it: the card faces straight down the mat, so that means heading 0, within 12 degrees. The docking loop above arrives at an angle and ends facing 17, so square up at the end. The starter drives at a fixed speed and only stops once it has lost sight of the tag, by then on top of the card.

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

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

# camera: the tag detector
set_cv("apriltag")
# again and again, for ever
while True:
    # every tag in view: [id, cx, cy, distance], nearest first
    tags = apriltags()
    if not tags:
        # leave the loop
        break
    # forward, sideways, rotation: -100 to 100 each, until the next command
    drive(50, 0, bearing_of(tags[0][1]) * 3)
    # 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: Drive to marker 1 and stop about 12 cm in front of it, facing it. Steer from the tag's cx, let the distance set the speed, and square up at the end: the card faces down the mat, so facing it squarely is heading 0.

A solution

```
from bugbot import *
connect()
def bearing_of(cx):
    return (cx - 160) * 120 / 320      # pixels to degrees, 120 degree view across 320
pixels
def find(tag_id):
    # spin until the tag is in view, then square up to it
    while True:
        tags = [t for t in apriltags() if t[0] == tag_id]
        if tags:
            break
        turn_right(40)
        wait(0.1)
    stop()
    wait(0.3)

def approach(tag_id, stop_at):
    while True:
        tags = [t for t in apriltags() if t[0] == tag_id]
        if not tags:
            turn_right(30)
            wait(0.1)
```

```
            continue
        tag = tags[0]
        cx, dist = tag[1], tag[3]
        if dist <= stop_at:
            break
        rot = bearing_of(cx) * 3
        speed = max(20, min(70, (dist - stop_at) * 3))
        drive(speed, 0, rot)
        wait(0.1)
    stop()
set_cv('apriltag')
approach(1, 13)
err = (0 - heading() + 180) % 360 - 180      # square up: the card faces down the mat, so
facing it is heading 0
turn_right(30, angle=err)
```

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