ANSWERS

U11.1 The pinhole camera

BugBotLab

Vision · University · about 30 min

What this lesson is about

One pixel is one ray. What that buys you, and what it does not.

- The model
- What a pixel is worth
- The tangent matters
- A bearing is not a position
- Getting the depth back

Questions

7 marks in all

1. The BugBot's camera is 320 pixels wide with a 120 degree field of view. What does this print? [1 mark]

```
import math
f = (320 / 2) / math.tan(math.radians(120 / 2))
bearing = math.degrees(math.atan((240 - 160) / f))
print(round(f, 1), round(bearing, 1))
```

Answer:

```
92.4 40.9
```

f = 160 / tan(60 degrees) = 92.4 px, and column 240 is atan(80 / 92.4) = 40.9 degrees right of the nose.

2. Which line works out the focal length in pixels correctly? [1 mark]

- ☐ **A** f = 160 / math.tan(math.radians(60))
- ☐ **B** f = 160 / math.tan(60)
- ☐ **C** f = 160 * math.tan(math.radians(60))
- ☐ **D** f = 320 / math.tan(math.radians(120))

Answer: A. math.tan takes radians, so 60 must be converted first. Half the width goes with half the field of view, and the focal length divides by the tangent.

3. A blob's centre is at column 200. What is its bearing from straight ahead, in degrees to 1 decimal place, using f = 92.4? [1 mark]

Answer: 23.4 (accept within 0.1). bearing = atan((200 - 160) / 92.4) = atan(0.433) = 23.4 degrees.

4. The linear guess bearing = (u - 160) x 0.375 degrees is used instead of the pinhole formula. How many degrees wrong is it at column 200? Give 1 decimal place. [1 mark]

Answer: 8.4 (accept within 0.1). The linear guess gives 40 x 0.375 = 15.0 degrees, the pinhole gives 23.4, so it is 8.4 degrees short. They only agree at the centre and the very edge.

5. Why can a single pixel not tell you where an object is? [1 mark]

- ☐ **A** Depth appears only as a divisor, so every point along a ray lands on the same pixel
- ☐ **B** The image is too low resolution to locate the object
- ☐ **C** The principal point is not exactly at the centre
- ☐ **D** Lens distortion moves the pixel

> **Answer:** A. Doubling X and Z gives the same u. A pixel names a direction, and the range has to come from a second constraint.

6. Which of these add the second constraint needed to get depth from a camera? [1 mark]

Tick every answer that is true.

- ☐ **A** Knowing the object's real size
- ☐ **B** Knowing the object is on a flat floor, with the camera's height and tilt
- ☐ **C** A second camera a known distance away
- ☐ **D** The same camera at two times, with the motion between them known
- ☐ **E** Averaging the bearing over many frames
- ☐ **F** A higher resolution sensor

> **Answer:** A, B, C, D. All four add a second ray or a known size. Averaging and more pixels make the bearing better but still give only a direction.

7. A tag is seen at column 220 and the detector reports it 50 cm away. What does this print? [1 mark]

```
import math
F = 92.4
bearing = math.degrees(math.atan((220 - 160) / F))
print(round(bearing, 1), round(50 * math.sin(math.radians(bearing)), 1), round(50 *
math.cos(math.radians(bearing)), 1))
```

> **Answer:**
>
> ```
> 33.0 27.2 41.9
> ```
>
> The bearing is atan(60 / 92.4) = 33.0 degrees, so the tag is 50 sin 33.0 = 27.2 cm right and 50 cos 33.0 = 41.9 cm ahead.

The task: turn a pixel into a bearing

Tag 7 is on the mat and the robot is standing still, facing along +y. Print `bearing:`, the angle from straight ahead to the tag in degrees, and `across:`, how far to the right of the robot's nose line the tag actually is in centimetres.

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

F = 92.4
set_cv("apriltag")
wait(0.3)
```

The hint students can ask for: The detector gives you a column in the picture and a range. The column is an angle: the camera is 320 pixels wide across 120 degrees, and the relation between a pixel offset from the centre and an angle is a tangent, not a straight line. The sideways offset needs the range as well, because a bearing on its own is a whole ray of possible places.

A solution

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

F = 92.4                      # focal length in pixels: 160 / tan(60 degrees)

set_cv("apriltag")
wait(0.3)
tags = [t for t in apriltags() if t[0] == 7]
tag_id, cx, cy, dist = tags[0]

bearing = math.degrees(math.atan((cx - 160) / F))
across = dist * math.sin(math.radians(bearing))
print("bearing:", round(bearing, 2))
print("across:", round(across, 1))
```

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