ANSWERS

U11.4 A tag as a fix

Vision · University · about 40 min

What this lesson is about

Range and bearing to a landmark you know the position of, which is a position fix.

- One tag, if you know your heading
- Two tags, if you do not
- Where a tag's pose estimate is weak
- Fixing and driving
- A fix is not a filter

Questions

6 marks in all

1. Tag 20 is at (40, 160). Facing heading 0, the robot sees it at a bearing of -20 degrees and a range of 50 cm. What does this print? [1 mark]

```
import math
Tx, Ty = 40.0, 160.0
h, b, d = 0.0, -20.0, 50.0
x = Tx - d * math.sin(math.radians(h + b))
y = Ty - d * math.cos(math.radians(h + b))
print(round(x, 1), round(y, 1))
```

Answer:

```
57.1 113.0
```

The direction to the tag is h + b = -20 degrees. x = 40 - 50 sin(-20) = 40 + 17.1 = 57.1 and y = 160 - 50 cos(20) = 160 - 47.0 = 113.0.

2. A one tag fix is taken at a range of 100 cm with a heading that is 1 degree wrong. How far sideways does the fix move, in cm to 2 decimal places? [1 mark]

Answer: 1.75 (accept within 0.05). One degree is 0.01745 rad, and 100 x 0.01745 = 1.75 cm. A tag fix is only as good as the heading that interprets it.

3. Why do two tags make the heading observable when one tag does not? [1 mark]

- ☐ **A** Two tags give four measurements, two ranges and two bearings, for three unknowns x, y and h
- ☐ **B** Two tags always form a right angle with the robot
- ☐ **C** The second tag's bearing is measured in the mat frame
- ☐ **D** Averaging two ranges removes the heading error

Answer: A. One tag gives two measurements for three unknowns, so the heading must come from elsewhere. Two tags over-determine the pose.

4. Put the parts of an AprilTag pose estimate in order from most to least reliable. [1 mark]

Number the lines 1 to 3 to put them in the right order.

___ `Range, from apparent size`

___ `Out-of-plane rotation, from the corner geometry`

___ `Bearing, from the centroid`

> **Answer:**
>
> ```
> Bearing, from the centroid
> Range, from apparent size
> Out-of-plane rotation, from the corner geometry
> ```
>
> The centroid averages many pixels, range degrades as d squared, and out-of-plane rotation is bistable near square on.

5. A robot steering on a tag's reported yaw sees the estimate jump by 15 degrees between frames while nothing moves. What is the best explanation? [1 mark]

☐ **A** Near square on, tilting the tag a few degrees either way gives almost the same image, so the solver flips between two poses

☐ **B** The bearing is noisy because the centroid is quantised

☐ **C** The camera's focal length is wrong

☐ **D** The tag is too close for its range to be measured

> **Answer:** A. The ambiguity is bistable. A bigger tag, an oblique view or a rigid bundle of tags resolves it, or you use the bearing and range instead.

6. Why is a tag fix not a substitute for a filter? [1 mark]

☐ **A** A fix exists only while a tag is in view, so it has to be combined with dead reckoning, weighted by how much each is worth

☐ **B** A fix is always less accurate than odometry

☐ **C** A filter cannot use range and bearing measurements

☐ **D** A fix gives heading but never position

> **Answer:** A. Predict with odometry, correct with the fix, and carry a variance. A robot that only knows where it is while it sees a tag is not much use.

The task: fix your position from two tags

Tag 20 is at (40, 160) on the mat and tag 21 at (150, 140). Work out where the robot is, print it as `my x:` and `my y:`, and then drive to (130, 100) on the mat and stop there, refixing as you go. Neither `position()` nor `heading()` is allowed: both are the lab's truth, and the point is that two tags are enough.

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

F = 92.4
TAGS = {20: (40.0, 160.0), 21: (150.0, 140.0)}
TX, TY = 130.0, 100.0
set_cv("apriltag")
wait(0.3)
```

The hint students can ask for: Tag 20 is at (40, 160) on the mat and tag 21 at (150, 140). Each one gives a range and a bearing, and a bearing is measured from the robot's nose, so a position worked out from one tag depends on which way the robot is facing. With two tags you have enough to solve for the heading as well: guess a heading, put the robot where each tag then says it is, compare the bearings those positions predict with the bearings you measured, and correct the heading by the difference. A handful of rounds of that converges. Then drive to (130, 100) on the mat, refixing as you go.

A solution

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

F = 92.4
TAGS = {20: (40.0, 160.0), 21: (150.0, 140.0)}
V_MAX, V_LAT = 20.0, 15.0
TX, TY = 130.0, 100.0

def look():
    """Every known tag in view, as (id, bearing in degrees, range in cm)."""
    out = []
    for tag_id, cx, cy, d in apriltags():
        if tag_id in TAGS:
            out.append((tag_id, math.degrees(math.atan((cx - 160) / F)), d))
    return out

def fix(seen, h):
    """Solve for the pose. Each tag puts the robot on a circle of radius d about it, in the
direction
    the bearing gives once the heading is known, so guess the heading and iterate."""
    x = y = 0.0
    for step in range(8):
        sx = sy = 0.0
        for tag_id, b, d in seen:
            tx, ty = TAGS[tag_id]
            a = math.radians(h + b)
            sx += tx - d * math.sin(a)
            sy += ty - d * math.cos(a)
        x, y = sx / len(seen), sy / len(seen)
        if len(seen) < 2:
            break                                   # one tag cannot tell you the heading
        err = 0.0
```

```
        for tag_id, b, d in seen:
            tx, ty = TAGS[tag_id]
            want = math.degrees(math.atan2(tx - x, ty - y))
            err += ((want - b) - h + 180) % 360 - 180
        h += err / len(seen)
    return x, y, h

set_cv("apriltag")
wait(0.3)
x, y, h = fix(look(), 0.0)
print("my x:", round(x, 1))
print("my y:", round(y, 1))

for tick in range(600):
    seen = look()
    if seen:
        x, y, h = fix(seen, h)
    dx, dy = TX - x, TY - y
    gap = math.hypot(dx, dy)
    if gap < 5:
        break
    speed = min(14.0, 0.8 * gap)
    wx, wy = speed * dx / gap, speed * dy / gap
    a = math.radians(h)
    cf = 100 * (wx * math.sin(a) + wy * math.cos(a)) / V_MAX
    cl = 100 * (wx * math.cos(a) - wy * math.sin(a)) / V_LAT
    if 0 < abs(cf) < 17:
        cf = 17 * (1 if cf > 0 else -1)            # under the dead band nothing moves at
all
    if 0 < abs(cl) < 17:
        cl = 17 * (1 if cl > 0 else -1)
    drive(cf, cl, 0)
    wait(0.1)
stop()
wait(0.4)
seen = look()
if seen:
    x, y, h = fix(seen, h)
print("stopped at", round(x, 1), round(y, 1), "facing", round(h, 1))
```

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