ANSWERS

U12.7 Capstone: the whole robot

Learning, and the capstone · University · about 90 min

What this lesson is about

Calibrate, estimate, localise, plan, follow, arrive, and report. One run, everything in it.

- What it takes
- The structure
- The fix
- What a good submission looks like
- How it would be assessed
- Where to go next

Questions

6 marks in all

1. Tag 21 is at (100, 195) and tag 22 at (40, 195). The robot measures r1 = 46.1 cm to tag 21 and r2 = 67.3 cm to tag 22. What does this print? [1 mark]

```
import math
x1, x2, wall = 100.0, 40.0, 195.0
r1, r2 = 46.1, 67.3
x = (x1 ** 2 - x2 ** 2 - r1 ** 2 + r2 ** 2) / (2 * (x1 - x2))
y = wall - math.sqrt(r1 ** 2 - (x - x1) ** 2)
print(round(x, 1), round(y, 1))
```

Answer:

```
90.0 150.0
```

Subtracting the circle equations gives x = (10000 - 1600 - 2125.2 + 4529.3) / 120 = 90.0, and y = 195 - sqrt(46.1^2 - 10^2) = 150.0.

2. In the two tag fix, why take the negative square root for y? [1 mark]

- ☐ **A** The robot is below the wall the tags are on, so it is at 195 minus the distance, not plus
- ☐ **B** Square roots in Python are negative by default
- ☐ **C** Heading 0 points down the mat
- ☐ **D** The tags face away from the robot

Answer: A. The two circles meet at two points mirrored in the wall. Only the one inside the mat, below y = 195, is possible.

3. After calibration a gyro bias of 0.2 degrees per second is left uncorrected. How many degrees of heading error does that build up over 30 s of driving? [1 mark]

Answer: 6 (accept within 0.01). Every degree per second of leftover bias is a degree of heading error per second: 0.2 x 30 = 6 degrees, which is why the bias is averaged over ten seconds, not one.

4. A tag fix puts the robot 40 cm from where its estimate says it is. What should the program do? [1 mark]

- ☐ **A** Treat it as a probable misread and gate it out, and print the disagreement
- ☐ **B** Jump the estimate to the fix, since the tags are ground truth
- ☐ **C** Average the fix and the estimate equally
- ☐ **D** Stop the run, because the estimator has failed

> **Answer:** A. A fix far outside the estimate's uncertainty is a misread tag, not a revelation. Checking a fix before using it is part of a good submission.

5. Only one of the two tags is in view for a frame. How should that frame be used for the fix? [1 mark]

- ☐ **A** As no fix at all
- ☐ **B** As half a fix, weighted by 0.5
- ☐ **C** By assuming the missing range equals the one seen
- ☐ **D** By using the last range of the missing tag

> **Answer:** A. The formula needs both ranges. One range puts the robot on a circle, not at a point.

6. Which belong in a submission that would earn a good mark? [1 mark]

Tick every answer that is true.

- ☐ **A** An estimator that runs every tick, not dead reckoning with a jump at the end
- ☐ **B** A stated frame, with every reported number in it
- ☐ **C** The run repeated a few times with the spread reported
- ☐ **D** The final estimate with how far out you believe it is
- ☐ **E** The single best run, with the others left out

> **Answer:** A, B, C, D. Repeatability with evidence and readability carry 30 marks between them. One successful run proves the program can pass, not that it works.

The task: the capstone

Reach the green target through the gap, without touching anything, and print `gyro bias:`, `my x:` and `my y:` from your own estimate. Plot `x` and `y` as you go. No `position()`.

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

DT = 0.1
V_MAX, V_LAT = 20.0, 15.0
START = (30.0, 25.0)
TAGS = {21: (100.0, 195.0), 22: (40.0, 195.0)}
```

The hint students can ask for: The robot starts at (30, 25) on a 200 by 200 mat, facing up it. A barrier runs across at y = 100 with a gap between x = 115 and x = 170, and the target is the far left corner past it. Tags 21 and 22 are on the far wall at (100, 195) and (40, 195). Calibrate, dead reckon, plan a route through the gap, take a fix from the tags once they are in view, and park in the target.

A solution

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

DT = 0.1
V_MAX, V_LAT = 20.0, 15.0
START = (30.0, 25.0)
TAGS = {21: (100.0, 195.0), 22: (40.0, 195.0)}
ROUTE = [(142.0, 70.0), (142.0, 125.0), (100.0, 152.0)]
TARGET = (42.0, 172.0)

# ---- calibration: ten seconds standing still buys a gyro worth integrating
rates = []
for i in range(100):
    rates.append(imu()[1])
    wait(DT)
bias = sum(rates) / len(rates)
print("gyro bias:", round(bias, 3))

x, y = START
h = 0.0                     # heading, held near 0 all run, so the camera keeps looking up
the mat
set_cv("apriltag")

def step():
    """One tick of the estimator: flow and gyro forward, the fused heading pulling the
heading back."""
    global x, y, h
    vx, vy = flow()
    rate = imu()[1] - bias
    fused = imu()[0]
    a = math.radians(h + 0.5 * rate * DT)
    x += (vx * math.cos(a) + vy * math.sin(a)) * DT
    y += (-vx * math.sin(a) + vy * math.cos(a)) * DT
    h = h + rate * DT
    while fused - h > 180:
```

```
        fused -= 360
    while h - fused > 180:
        fused += 360
    h = 0.97 * h + 0.03 * fused
    plot("x", x)
    plot("y", y)
    wait(DT)

def hold_heading():
    """The rotation command that keeps the robot square, or nothing when it is close
enough."""
    err = (h + 180) % 360 - 180
    if abs(err) < 1.5:
        return 0
    return max(-30, min(30, -2.0 * err))

def go_to(tx, ty, tol=4.0, limit=500):
    for tick in range(limit):
        dx, dy = tx - x, ty - y
        gap = math.hypot(dx, dy)
        if gap < tol:
            break
        speed = max(6.0, min(13.0, 0.7 * gap))
        wx, wy = speed * dx / gap, speed * dy / gap
        a = math.radians(h)
        drive(100 * (wx * math.sin(a) + wy * math.cos(a)) / V_MAX,
              100 * (wx * math.cos(a) - wy * math.sin(a)) / V_LAT,
              hold_heading())
        step()
    stop()
    for i in range(4):
        step()

def fix():
    """Two tag ranges put the robot on the mat: both tags sit on the far wall, so the
algebra is short."""
    global x, y
    got = {}
    for i in range(12):
        for t in apriltags():
            if t[0] in TAGS:
                got.setdefault(t[0], []).append(t[3])
        step()
    if 21 not in got or 22 not in got:
        return False
    r1 = sum(got[21]) / len(got[21])
    r2 = sum(got[22]) / len(got[22])
    x1, y1 = TAGS[21]
    x2, _y2 = TAGS[22]
    x = (x1 * x1 - x2 * x2 - r1 * r1 + r2 * r2) / (2 * (x1 - x2))
    up = r1 * r1 - (x - x1) ** 2
    if up < 0:
        return False
    y = y1 - math.sqrt(up)
    print("fix:", round(x, 1), round(y, 1))
    return True

for wx, wy in ROUTE:
    go_to(wx, wy)
```

```
fix()
go_to(TARGET[0], TARGET[1], tol=3.0)
print("my x:", round(x, 1))
print("my y:", round(y, 1))
```

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