ANSWERS

U3.4 Calibration

Odometry and drift · University · about 30 min

What this lesson is about

Measuring the gyro bias and the flow scale, and taking both out of the estimate.

- Calibrating the gyro
- Calibrating the flow scale
- The order matters
- Doing it in one program
- What calibration cannot fix

Questions

6 marks in all

1. A robot is driven exactly 100 cm, and its integrated flow says 96 cm. What scale factor should multiply future readings? Give three decimal places. [1 mark]

> **Answer:** 1.042 (accept within 0.001). scale = true distance / integrated distance = 100 / 96 = 1.042.

2. A gyro with a bias of 0.3 deg/s is integrated over five 0.1 s ticks, once raw and once with the bias subtracted. What does this print? [1 mark]

```
DT = 0.1
bias = 0.3
rates = [0.3, 0.3, 30.3, 30.3, 0.3]
raw = cal = 0.0
for r in rates:
    raw += r * DT
    cal += (r - bias) * DT
print(round(raw, 2), round(cal, 2))
```

> **Answer:**
>
> ```
> 6.15 6.0
> ```
>
> The robot really turned 30 deg/s for 0.2 s, which is 6 degrees. Raw integration adds 0.3 × 0.5 = 0.15 degrees of bias to give 6.15; the calibrated sum is exactly 6.0.

3. Why calibrate the gyro before the flow scale? [1 mark]

☐ **A** Calibrating the scale means driving, and steering with an uncalibrated gyro curves the path and corrupts the distance being measured

☐ **B** Calibrating the gyro needs the robot to move, so it has to come first while the path is known

☐ **C** The order does not matter, because the two errors are independent

☐ **D** The flow scale depends on the heading, so it changes after every turn

> **Answer:** A. Fix the heading, then the distance. The gyro calibration is done standing still, so it needs nothing else to be right first.

4. What is the name for recalibrating an inertial sensor at every moment the vehicle is known to be still, as a shoe-mounted pedestrian tracker does once per step? [1 mark]

> **Answer:** zero velocity update. A zero velocity update treats each known stationary moment as a free calibration, which keeps the bias estimate fresh.

5. Which of these can calibration not fix? [1 mark]

Tick every answer that is true.

- ☐ **A** A gyro bias that drifts with temperature and time
- ☐ **B** Slip between the drive and the mat
- ☐ **C** Errors the model has no name for
- ☐ **D** A constant gyro bias
- ☐ **E** A constant flow scale error

> **Answer:** A, B, C. Calibration subtracts or divides out errors that are constant and modelled. A bias that wanders, slip, and anything unmodelled are left, which is why calibration buys time but does not bound the error.

6. Why can the flow scale not be calibrated using only the robot's own sensors? [1 mark]

- ☐ **A** It needs a true distance from outside, such as a tape measure, a wall at a known place or a landmark
- ☐ **B** The flow sensor cannot be read while the robot is moving
- ☐ **C** The gyro has to be recalibrated at the same time
- ☐ **D** Integrating flow() twice would give the scale directly

> **Answer:** A. The robot's own measurement of distance is the thing in doubt, so something else has to say what the true distance was. Calibration always costs an external reference.

The task: a calibrated lap

Calibrate the gyro, then dead reckon at least 120 cm with turns in it. Print `bias:`, `my x:` and `my y:`. The position tolerance is tighter than the uncalibrated task, so the calibration has to be real.

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

DT = 0.1
# stand still first and learn the bias
```

The hint students can ask for: Stand still for a few seconds first and average the gyro. Subtract that bias from every turn rate afterwards. The tolerance here is tighter than the uncalibrated task, so the calibration has to be doing real work.

A solution

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

DT = 0.1

# stand still and learn the gyro's bias
rates = []
for i in range(40):
    rates.append(imu()[1])
    wait(DT)
bias = sum(rates) / len(rates)

x = y = h = 0.0

def step(n=1):
    global x, y, h
    for i in range(n):
        vx, vy = flow()
        rate = imu()[1] - bias
        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 += rate * DT
        wait(DT)

forward(75)
step(45)
stop()
step(4)

drive(0, 0, 55)
while h < 88:
    step()
stop()
step(4)

forward(75)
step(45)
stop()
```

```
step(4)

print("bias:", round(bias, 2))
print("my x:", round(x, 1))
print("my y:", round(y, 1))
print("truth", position())
```

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