ANSWERS

U3.2 Write your own odometry

Odometry and drift · University · about 30 min

What this lesson is about

flow() and imu() in, a pose out, in about ten lines. Then check it against the truth.

- The update
- Which heading to rotate by
- The whole thing
- Why not use the heading straight from the IMU

Questions

6 marks in all

1. One odometry update, with a deliberately long 1 s step so the numbers are easy. The robot starts at heading 0, reads (vx, vy) = (0, 20) and turns at 60 deg/s. What does this print? [1 mark]

```
import math
DT = 1.0
x = y = h = 0.0
vx, vy, rate = 0.0, 20.0, 60.0
h_mid = h + 0.5 * rate * DT
a = math.radians(h_mid)
x += (vx * math.cos(a) + vy * math.sin(a)) * DT
y += (-vx * math.sin(a) + vy * math.cos(a)) * DT
h += rate * DT
print(round(h_mid, 1), round(x, 2), round(y, 2), round(h, 1))
```

Answer:

```
30.0 10.0 17.32 60.0
```

The midpoint heading is 0 + 0.5 × 60 × 1 = 30. The 20 cm step is rotated by 30 degrees: x = 20 sin 30 = 10.0, y = 20 cos 30 = 17.32. The heading then moves on to 60.

2. Why does the odometry update rotate flow() by the heading before adding it to x and y? [1 mark]

- ☐ **A** flow() measures in the body frame, and the state is kept in the world frame
- ☐ **B** flow() measures in the world frame, and the state is kept in the body frame
- ☐ **C** It corrects for the gyro bias
- ☐ **D** It converts centimetres per second into centimetres

Answer: A. The velocity arrives as right and forward, the position is kept as world x and y, and the rotation matrix from U2.2 connects them. Multiplying by dt is what turns a speed into a distance.

3. Why is rotating by the heading at the start of the step, rather than the midpoint heading, a problem worth fixing? [1 mark]

- ☐ **A** While turning it leans the same way every step, so it is a bias that accumulates all run
- ☐ **B** It adds noise that averages out, so it only matters on short runs
- ☐ **C** It makes the heading itself wrong, not the position
- ☐ **D** It is only a problem when the robot is driving straight

> **Answer:** A. The error is small each step but always in the same direction, which makes it bias rather than noise. The midpoint heading removes most of it at no cost.

4. The loop step is 0.1 s and the robot turns at 60 deg/s. By how many degrees does the heading change within one step, which is the gap between turning first and moving first? [1 mark]

> **Answer:** 6. 60 × 0.1 = 6 degrees. The midpoint heading splits that difference, using 3 degrees past the starting heading.

5. How do the fused heading imu()[0] and the integrated gyro rate differ? [1 mark]

- ☐ **A** The fused heading is noisy and wanders slowly but has no growing error; the integrated rate is smooth over a second but drifts without limit
- ☐ **B** The integrated rate is noisy but never drifts; the fused heading is smooth but drifts without limit
- ☐ **C** Both drift without limit at the same rate
- ☐ **D** The fused heading is exact, so integrating the rate is never needed

> **Answer:** A. They fail differently, so each is better over a different timescale. Combining them, weighted by trust, is the complementary filter in U6.1.

6. In the task, why must the turn be made with drive() inside your loop rather than with turn_right(angle=90)? [1 mark]

- ☐ **A** A blocking turn runs the world without running your loop, so the turn never reaches your estimate
- ☐ **B** turn_right() resets the gyro, which clears your heading
- ☐ **C** turn_right() uses position() internally, which the task forbids
- ☐ **D** turn_right() turns too fast for flow() to measure

> **Answer:** A. Odometry only knows what happened on the ticks it ran. A blocking call hides a whole corner from it.

The task: your own odometry

Drive at least 90 cm with a corner in it, running your own odometry, and print `my x:` and `my y:` at the end. `position()` is not allowed anywhere in this task, not even to print. Turn with `drive()` inside your loop: a blocking `turn_right(angle=90)` runs the world without running your loop, so the turn never reaches your estimate.

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

DT = 0.1
x = y = h = 0.0
```

The hint students can ask for: Keep x, y and h as your own variables and update them every tick, including while turning. Blocking calls like turn_right(angle=90) run the world without running your loop, so the turn never reaches your estimate: drive with drive() inside the loop instead. position() is not allowed anywhere in this task, not even to print.

A solution

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

DT = 0.1
x = y = h = 0.0

def step(n=1):
    """n odometry updates, one per tick, whatever the robot is doing."""
    global x, y, h
    for i in range(n):
        vx, vy = flow()
        rate = imu()[1]
        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(30)
stop()
step(4)

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.