ANSWERS

U3.1 Integrating velocity

Odometry and drift · University · about 25 min

What this lesson is about

The odometry update, one step at a time, and what the step size costs you.

- The simplest case
- Reason one: the step is not an instant
- Reason two: the reading is not the speed
- Reason three: the loop is not exactly 0.1 s

Questions

6 marks in all

1. Speeds in cm/s are read every 0.1 s as the robot speeds up. This integrates them two ways. What does it print? [1 mark]

```
speeds = [0.0, 8.0, 14.0, 18.0, 20.0, 20.0]
DT = 0.1
euler = 0.0
trap = 0.0
for i in range(1, len(speeds)):
    euler += speeds[i] * DT
    trap += 0.5 * (speeds[i] + speeds[i - 1]) * DT
print("euler:", round(euler, 2))
print("trapezium:", round(trap, 2))
```

Answer:

```
euler: 8.0
trapezium: 7.0
```

Euler uses each new reading for the whole step: (8 + 14 + 18 + 20 + 20) × 0.1 = 8.0 cm. The trapezium rule averages each pair: (4 + 11 + 16 + 19 + 20) × 0.1 = 7.0 cm. While accelerating, Euler overestimates.

2. Which integration method takes the speed read at one instant and uses it as if it were the average over the whole step? [1 mark]

Answer: Euler. That is Euler integration, and its error grows with the step size. The trapezium rule is the cheap improvement.

3. When do the trapezium rule and Euler integration give the same answer? [1 mark]

- ☐ **A** Once the speed is steady
- ☐ **B** While the robot is speeding up
- ☐ **C** Only when the step is very small
- ☐ **D** Never, because the trapezium rule always averages two readings

Answer: A. If this reading equals the last one, their average is the same reading. The methods differ only while the speed is changing.

4. A loop multiplies by DT = 0.1, but its real period is 0.11 s. The robot drives at a steady 20 cm/s for 50 passes. By how many centimetres does the integral fall short of the true distance? [1 mark]

> **Answer:** 10. The true distance is 20 × 50 × 0.11 = 110 cm and the estimate is 20 × 50 × 0.1 = 100 cm. Every step is short in the same direction, so the error accumulates rather than cancelling.

5. Why does assuming the loop period, rather than measuring it with clock(), cause an error that grows through the run? [1 mark]

- ☐ **A** The real period is always at least the wait, so every step is short the same way and the errors add up
- ☐ **B** Assumed periods are noisy, and noise adds up as a random walk
- ☐ **C** clock() drifts less than wait(), so it is only needed on long runs
- ☐ **D** The flow sensor's scale changes if the period is not exact

> **Answer:** A. wait(0.1) waits at least 0.1 s and the loop's own work adds more. A one-sided error is a bias, and bias grows linearly. Two lines measuring dt remove it.

6. A straight-line integral of flow() is close to the truth but not exact. Which of these are reasons the lesson gives? [1 mark]

Tick every answer that is true.

- ☐ **A** The reading is one instant, used as the average over the step
- ☐ **B** The flow reading has noise and a scale a few percent off
- ☐ **C** The loop period is not exactly the 0.1 s it is multiplied by
- ☐ **D** The deadman stops the robot between readings
- ☐ **E** position() is only accurate to a few centimetres

> **Answer:** A, B, C. Integration method, sensor error and timing are three separate sources, and it pays to keep them apart. position() is the accurate reference here, and a loop that keeps commanding never trips the deadman.

The task: integrate a velocity

Drive at least 50 cm in a straight line, integrating `flow()` as you go, and print your own answer as `my y: 54.3`. It is marked against where the robot really ended up.

```
from bugbot import *
connect()

DT = 0.1
y = 0.0
forward(70)
```

The hint students can ask for: Drive straight and read flow() every tenth of a second. Each reading is a speed in cm/s, so it contributes speed times the length of the step. Add them up as you go.

A solution

```
from bugbot import *
connect()

DT = 0.1
y = 0.0
forward(70)
for i in range(40):
    vx, vy = flow()
    y += vy * DT
    wait(DT)
stop()
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.