ANSWERS

U1.3 Truth and belief

The robot as a system · University · about 25 min

What this lesson is about

position() is the lab's overhead camera. odometry() is what the robot has, and it drifts.

- Watch belief come apart from truth
- Why dead reckoning drifts
- Turning the noise off

Questions

6 marks in all

1. Why is using position() as the input to a controller treated as cheating? [1 mark]

- ☐ **A** It is the lab's overhead measurement of the robot, which a real robot does not have once it leaves the room
- ☐ **B** It is less accurate than odometry()
- ☐ **C** It reports the position in the body frame, not the world frame
- ☐ **D** It updates too slowly to be used inside a control loop

> **Answer:** A. position() and heading() are the equivalent of motion capture: there to mark your work. The robot itself only carries flow() and imu().

2. A gyro reads 0.4 degrees per second too high. Integrated for 90 seconds, how many degrees of heading error does that bias alone produce? [1 mark]

> **Answer:** 36. Bias adds up linearly: 0.4 × 90 = 36 degrees, in the same direction every run.

3. How do the two kinds of sensor error grow when dead reckoning integrates them? [1 mark]

- ☐ **A** Zero-mean noise grows with the square root of time; bias grows linearly with time
- ☐ **B** Noise grows linearly with time; bias grows with the square root of time
- ☐ **C** Both grow linearly with time
- ☐ **D** Noise averages away completely; only bias grows

> **Answer:** A. Noise is a random walk, so its errors partly cancel and grow as the square root of time. A bias never cancels, so it marches on linearly, and linear soon overtakes square root.

4. Driving a square, the heading error gets a little bigger at every corner. Why does that matter more than a position error of the same size? [1 mark]

- ☐ **A** A heading error rotates every step taken after it, so a small error at a corner becomes a large position error down the next straight
- ☐ **B** The gyro bias only acts while the robot is turning
- ☐ **C** The flow sensor is noisier while turning than while driving straight
- ☐ **D** odometry() restarts its count at each corner

> **Answer:** A. Heading error is the expensive one in dead reckoning. The gyro bias acts all the time; it is the travel after the corner that turns heading error into distance.

5. Driving the same square with set_noise(0), odometry() and position() agree exactly, yet the robot still has not come back to where it started. What does that show? [1 mark]

- ☐ **A** set_noise() changes the sensors, not the drive: the drive still curves, and the robot now measures the curve exactly
- ☐ **B** set_noise(0) also turns off the drive's coupling, so the square is perfect
- ☐ **C** odometry() stops working when the noise is off
- ☐ **D** The gyro bias is still switched on

> **Answer:** A. With perfect sensors, every centimetre of dead reckoning error disappears: the error in the normal run came from the sensors. The drive is still the same imperfect drive, which is why a controller still has work to do.

6. At the end of a run, odometry() gives (12.0, 118.5) and position() gives (15.0, 114.5). What does this print? [1 mark]

```
import math
ox, oy = 12.0, 118.5
tx, ty = 15.0, 114.5
print("error:", round(math.hypot(ox - tx, oy - ty), 1))
```

> **Answer:**
>
> ```
> error: 5.0
> ```
>
> The gaps are 3 cm in x and 4 cm in y, and the distance is the square root of $3^2 + 4^2$, which is 5.0 cm.

The task: how far out is it?

Drive at least 120 cm, with at least one turn in it. At the end, print how far `odometry()` is from `position()`, as `error: 4.3`.

```
from bugbot import *
connect()

forward(70, distance=50)
```

The hint students can ask for: Drive at least a metre and a bit, with a turn or two in it. At the end read both odometry() and position() and work out the distance between the two points.

A solution

```
from bugbot import *
connect()

for leg in range(3):
    forward(70, distance=50)
    turn_right(40, angle=90)
stop()
ox, oy, oh = odometry()
tx, ty = position()
gap = ((ox - tx) ** 2 + (oy - ty) ** 2) ** 0.5
print("odometry", ox, oy, "truth", tx, ty)
print("error:", round(gap, 1))
```

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