ANSWERS

U3.7 Project: the long lap

Odometry and drift · University · about 45 min

What this lesson is about

Two metres of driving, one landmark, and an estimate that has to stay honest.

- What it takes
- The shape of the program
- What comes next

Questions

6 marks in all

1. Put the long lap program in order. [1 mark]

Number the lines 1 to 5 to put them in the right order.

___ `Stop in the home zone and keep updating while stopped`
___ `Steer towards (0, 0) in your own estimated frame, still updating every tick`
___ `Drive the outward part of the lap, updating the odometry every tick`
___ `Stand still and average the gyro rate to learn the bias`
___ `Print my x and my y`

Answer:

```
Stand still and average the gyro rate to learn the bias
Drive the outward part of the lap, updating the odometry every tick
Steer towards (0, 0) in your own estimated frame, still updating every tick
Stop in the home zone and keep updating while stopped
Print my x and my y
```

Calibration comes first, because without it the rest does not matter. The odometry runs throughout, and the way home is steered on the estimate itself.

2. The example program calls step(4) after stop(). Why keep updating the odometry while the robot is stopped? [1 mark]

☐ **A** The robot coasts after the stop and can be nudged while still, and an update skipped is a piece of the path missing

☐ **B** It recalibrates the gyro bias

☐ **C** The plot needs extra points to draw properly

☐ **D** It keeps the deadman from stopping the robot

Answer: A. Motion that happens when the loop is not running never reaches the estimate. Stopping is exactly when robots are most often nudged.

3. The lap is marked to within 10 cm after 200 cm of driving. If heading error were the only source, what is the largest heading error you could afford, in degrees to one decimal place? [1 mark]

Answer: 2.9 (accept within 0.05). d × e = 10 with d = 200 gives e = 0.05 radians, which is 0.05 × 180 / pi = 2.9 degrees.

4. An uncalibrated gyro has a bias of 0.4 deg/s. How many seconds does it take to build up a 2.9 degree heading error, to one decimal place? [1 mark]

> **Answer:** 7.2 (accept within 0.1). 2.9 / 0.4 = 7.2 s. The whole error allowance for a two metre lap is used up in seconds, which is why calibrating the gyro comes first.

5. Which of the steps in the lesson is optional? [1 mark]

- ☐ **A** Taking a fix from tag 3
- ☐ **B** Calibrating the gyro
- ☐ **C** Rotating by the midpoint heading
- ☐ **D** Running the odometry while stopped

> **Answer:** A. The tag fix is optional, and it is the difference between a good estimate and a lucky one. The others are what make the dead reckoning good enough to get home at all.

6. The same lap driven in the other direction ends with a noticeably different error. What is the most likely cause? [1 mark]

- ☐ **A** A leftover gyro bias, which adds to turns one way and takes away from turns the other
- ☐ **B** Flow noise, which is larger in one direction of travel
- ☐ **C** Slip, which only happens on right turns
- ☐ **D** Nothing; the two errors should always be the same size

> **Answer:** A. Zero-mean noise has no preferred direction, but a bias leans one way. Reversing the lap reverses whether it helps or hurts the turns.

The task: the long lap

Two metres of driving, back into the home zone under your own navigation, with `my x:` and `my y:` printed at the end and within 10 cm of the truth.

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

DT = 0.1
# calibrate, then drive the lap, updating every tick
```

The hint students can ask for: Two metres of driving, then home on your own estimate: steer towards (0, 0) in your estimated frame with the inverse kinematics from U2. If the estimate is good you arrive; if it is not, you do not, which is the test. Calibrate the gyro before you move: it costs five seconds and buys most of the accuracy.

A solution

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

DT = 0.1
V_MAX, V_LAT = 20.0, 15.0

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
        plot("x", x)
        plot("y", y)
        wait(DT)

def leg(ticks):
    forward(75)
    step(ticks)
    stop()
    step(4)

def turn_to(target):
    """Turn until our own estimate says we are facing the target heading."""
    drive(0, 0, 55)
    while abs((h - target + 180) % 360 - 180) > 4:
        step()
```

```
    stop()
    step(4)

def go_to(tx, ty):
    """Home in on a point in our own estimated frame, through the inverse kinematics."""
    for tick in range(300):
        dx, dy = tx - x, ty - y
        gap = math.hypot(dx, dy)
        if gap < 4:
            break
        speed = min(13.0, 0.6 * 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, 0)
        step()
    stop()
    step(4)

leg(45)
turn_to(180)
leg(45)
turn_to(360)
h -= 360
leg(45)
turn_to(180)
leg(30)

# home on the estimate, not on the truth
go_to(0.0, 0.0)

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.