ANSWERS

U10.7 Project: drive the route

BugBotLab

Following a trajectory · University · about 50 min

What this lesson is about

A full route through waypoints, marked on how closely it was followed and how long it took.

- The route
- The structure
- Slowing down for what matters
- Where to look when it fails
- What comes next

Questions

6 marks in all

1. What does this print for the project's route? [1 mark]

```
import math
PATH = [(30.0, 30.0), (30.0, 150.0), (120.0, 150.0), (120.0, 60.0), (170.0, 60.0)]
cum = [0.0]
for (ax, ay), (bx, by) in zip(PATH, PATH[1:]):
    cum.append(cum[-1] + math.hypot(bx - ax, by - ay))
print(cum)
print(round(cum[-1] / 13.0, 1))
```

Answer:

```
[0.0, 120.0, 210.0, 300.0, 350.0]
26.9
```

The legs are 120, 90, 90 and 50 cm, so the cumulative arc lengths end at 350 cm, which at 13 cm/s is 26.9 s of driving.

2. The speed rule is cruise x max(0.45, 1 - off / 14) with a cruise of 13 cm/s. What speed does it give when the robot is 10 cm off the path, in cm/s to 2 decimal places? [1 mark]

Answer: 5.85 (accept within 0.01). 1 - 10 / 14 = 0.29, which is below the 0.45 floor, so the speed is 13 x 0.45 = 5.85 cm/s.

3. Why is slowing down when off the path self correcting? [1 mark]

- ☐ **A** Drifting wide costs speed, which gives the correction time to work, which brings the speed back
- ☐ **B** Lower speed shortens the look-ahead distance
- ☐ **C** The dead band disappears at low speed
- ☐ **D** Slowing down removes the drive lag

Answer: A. The cross-track correction has more time per centimetre of travel, so the robot returns to the line and the rule hands the speed back.

4. `off path` is small on the straights but spikes at every corner, and the robot clips the crate. Where should you look? [1 mark]

- ☐ **A** The look-ahead cutting the corners: reduce L, slow down for the bend, or both
- ☐ **B** The cross-track sign convention
- ☐ **C** The cruise speed is too low to finish in time
- ☐ **D** The path itself goes through the crate

> **Answer:** A. Good straights rule out the cross-track loop. Spikes at corners are the chord cut of the look-ahead, and the crate beside the route is where that cut lands.

5. A follower that points its velocity at the look-ahead point stays within 1.3 cm of this route at L = 3. A classmate's robot turns to face the look-ahead point and drives forwards only, and it weaves along the straights. What is the likely cause? [1 mark]

- ☐ **A** Steering through the heading with a look-ahead too small for the speed
- ☐ **B** The look-ahead is too large
- ☐ **C** The corner speed rule is too cautious
- ☐ **D** The route is longer than 350 cm

> **Answer:** A. A robot that steers can only move sideways by turning, so with a short L the drive lag lets the heading carry it across the line. At 13 cm/s it weaves at L = 3 and settles by L = 5. Raise L, lower the speed, or point the velocity instead, which has no heading to overshoot.

6. The corner rule reduces speed in proportion to the angle between the path direction at the projection and at the look-ahead point. What idea is that? [1 mark]

- ☐ **A** The preview idea of pure pursuit, applied to the throttle instead of the steering
- ☐ **B** Feedforward of the reference speed
- ☐ **C** A dead band inverse on the speed
- ☐ **D** Time scaling from the tracking error

> **Answer:** A. The look-ahead point sees the bend coming, so the robot slows before it reaches the corner rather than after it has gone wide.

The task: drive the route

Follow the tape from (30, 30) round to the green finish, within 10 cm of it for at least 85 percent of the run, without touching the crate, inside 90 seconds. Plot `off path` and `speed`, and print `drove:`, how far the robot actually travelled.

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

DT = 0.1
START = (30.0, 30.0)
PATH = [(30.0, 30.0), (30.0, 150.0), (120.0, 150.0), (120.0, 60.0), (170.0, 60.0)]
CRUISE, LOOK = 13.0, 16.0
```

The hint students can ask for: The tape runs (30, 30), (30, 150), (120, 150), (120, 60), (170, 60), and the crate sits beside the middle of it. Follow the path with a look-ahead point, slow down where the path bends or where you have drifted off it, and keep the whole run inside the time limit.

A solution

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

DT = 0.1
V_MAX, V_LAT = 20.0, 15.0
START = (30.0, 30.0)
PATH = [(30.0, 30.0), (30.0, 150.0), (120.0, 150.0), (120.0, 60.0), (170.0, 60.0)]
CRUISE, LOOK = 13.0, 16.0

cum = [0.0]
for (ax, ay), (bx, by) in zip(PATH, PATH[1:]):
    cum.append(cum[-1] + math.hypot(bx - ax, by - ay))
TOTAL = cum[-1]

def project(px, py):
    best, at = 1e9, 0.0
    for i in range(len(PATH) - 1):
        ax, ay = PATH[i]
        bx, by = PATH[i + 1]
        vx, vy = bx - ax, by - ay
        l2 = vx * vx + vy * vy
        t = 0.0 if l2 == 0 else max(0.0, min(1.0, ((px - ax) * vx + (py - ay) * vy) / l2))
        qx, qy = ax + t * vx, ay + t * vy
        d = math.hypot(px - qx, py - qy)
        if d < best:
            best, at = d, cum[i] + t * math.sqrt(l2)
    return at, best

def point_at(s):
    s = max(0.0, min(TOTAL, s))
    for i in range(len(PATH) - 1):
        if s <= cum[i + 1] or i == len(PATH) - 2:
            seg = cum[i + 1] - cum[i]
            u = 0.0 if seg == 0 else (s - cum[i]) / seg
            ax, ay = PATH[i]
```

```
            bx, by = PATH[i + 1]
            return (ax + (bx - ax) * u, ay + (by - ay) * u)
    return PATH[-1]

def band(u):
    if abs(u) < 5.0:
        return 0.0
    return max(17.0, min(100.0, u)) if u > 0 else min(-17.0, max(-100.0, u))

def steer(wx, wy):
    h = math.radians(heading())
    vx = wx * math.cos(h) - wy * math.sin(h)
    vy = wx * math.sin(h) + wy * math.cos(h)
    e = (0.0 - heading() + 180) % 360 - 180
    rot = 0.0 if abs(e) < 3 else max(-30.0, min(30.0, 2.0 * e))
    drive(band(100 * vy / V_MAX), band(100 * vx / V_LAT), band(rot))

def bend(s):
    """how much the path turns between here and the look-ahead point, in degrees"""
    ax, ay = point_at(s)
    bx, by = point_at(s + LOOK * 0.5)
    cx, cy = point_at(s + LOOK)
    a1 = math.atan2(by - ay, bx - ax)
    a2 = math.atan2(cy - by, cx - bx)
    return abs(math.degrees(a2 - a1) + 180) % 360 - 180

drove = 0.0
last = position()
for tick in range(800):
    px, py = position()
    drove += math.hypot(px - last[0], py - last[1])
    last = (px, py)
    x, y = START[0] + px, START[1] + py
    s, off = project(x, y)
    plot("off path", off)
    if TOTAL - s < 3 and math.hypot(PATH[-1][0] - x, PATH[-1][1] - y) < 4:
        break
    tx, ty = point_at(s + LOOK)
    gap = math.hypot(tx - x, ty - y)
    speed = min(CRUISE, 2.0 + 0.6 * (TOTAL - s))
    speed *= max(0.45, 1.0 - off / 14.0)                # off the path: slow down and get
back on it
    speed *= max(0.55, 1.0 - abs(bend(s)) / 120.0)      # a corner coming: slow down for
it
    plot("speed", speed)
    if gap > 1e-6:
        steer(speed * (tx - x) / gap, speed * (ty - y) / gap)
    wait(DT)
stop()
wait(0.4)
px, py = position()
drove += math.hypot(px - last[0], py - last[1])
print("drove:", round(drove, 1))
print("took:", round(clock(), 1))
```

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