ANSWERS

U10.4 Pure pursuit

Following a trajectory · University · about 35 min

What this lesson is about

Chase a point a fixed distance ahead on the path, and choose the one number that decides everything.

- Why it works
- The trade-off
- Speed-dependent look-ahead
- Finding the look-ahead point

Questions

8 marks in all

1. A car chases a look-ahead point 18 cm ahead that is 3 cm to the side in the body frame. What radius of arc does pure pursuit command, in cm? [1 mark]

> **Answer:** 54 (accept within 0.1). kappa = 2 y / L^2 = 6 / 324 per cm, so the radius is 324 / 6 = 54 cm.

2. What does a longer look-ahead distance cost? [1 mark]

- ☐ **A** Corners are cut, and on a tight bend the robot can take the chord straight across the inside
- ☐ **B** The robot weaves down straight lines
- ☐ **C** The controller needs a higher gain to stay on the path
- ☐ **D** The look-ahead point can move backwards along the path

> **Answer:** A. A long L is smooth and stable, but on a corner the look-ahead point is across the bend. Weaving is the small L failure, and only for a robot that steers like a car.

3. The BugBot follows a circle of radius 40 cm with pure pursuit, pointing its velocity at a look-ahead point L = 18 cm further along. By roughly how much does it track inside the circle, in cm to 1 decimal place? [1 mark]

> **Answer:** 4.0 (accept within 0.15). Moving along the chord to the look-ahead point, it settles where that chord is tangent to its own circle: R (1 - cos(L / R)) = 3.98 cm, about L^2 / (2 R) = 324 / 80 = 4.05 cm. A follower like the task's came 4.5 cm off in the bend.

4. A robot that steers like a car, turning to face the look-ahead point and driving forwards, weaves down a straight line with a period of about 2 s. What is the fix? [1 mark]

- ☐ **A** Raise the look-ahead distance
- ☐ **B** Raise the gain
- ☐ **C** Lower the look-ahead distance
- ☐ **D** Lower the cruise speed to zero on straights

> **Answer:** A. The weave is the drive lag turning an aggressive correction into overshoot: the heading is still swinging when the robot reaches the line. A longer L makes the controller gentler; a higher gain makes it worse. In the simulator it weaves at L = 3 and 14 cm/s and not at L = 5.

5. At L = 3 and 14 cm/s a follower that steers like a car weaves up to 4.1 cm either side of the tape, but the task's follower, which points its velocity at the look-ahead point, stays within 0.5 cm. Why? [1 mark]

- ☐ **A** Pointing the velocity moves the robot sideways directly, so there is no heading to swing past the line while the lag catches up
- ☐ **B** The task's follower has no drive lag
- ☐ **C** Pointing the velocity uses a longer effective look-ahead
- ☐ **D** The car-like follower drives faster on the straights

> **Answer:** A. The car corrects in two steps, turn then drift, and the 0.25 s lag keeps it turning after it is back on the line. The holonomic follower corrects in one step, so the same lag only slows the correction down.

6. Why is projecting onto the path and adding L better than intersecting a circle of radius L with the path? [1 mark]

- ☐ **A** It is defined even when the robot is more than L from the path, and it cannot send the robot backwards
- ☐ **B** It gives a shorter look-ahead distance on corners
- ☐ **C** The circle intersection needs the path to be curved
- ☐ **D** It avoids having to compute arc length

> **Answer:** A. The circle has no intersection exactly when the robot is far off the path, which is when the controller matters most. The projection is defined everywhere and monotone.

7. The lesson's path is built from a straight, a quarter circle as 8 chords, and a straight. What does this print? [1 mark]

```
import math
arc = [(80 + 40 * math.cos(math.radians(180 - k * 11.25)),
        120 + 40 * math.sin(math.radians(180 - k * 11.25))) for k in range(9)]
PATH = [(40.0, 40.0)] + arc + [(150.0, 160.0)]
cum = [0.0]
for (ax, ay), (bx, by) in zip(PATH, PATH[1:]):
    cum.append(cum[-1] + math.hypot(bx - ax, by - ay))
print(round(cum[-1], 1), round(80 + 20 * math.pi + 70, 1))
```

> **Answer:**
>
> ```
> 212.7 212.8
> ```
>
> 80 cm up, 8 chords of 7.84 cm (62.7 cm) round the bend, then 70 cm across: 212.7 cm. The true quarter circle is 20 pi = 62.8 cm, so the polyline is very slightly short.

8. Why is the look-ahead usually made to grow with speed, L = L0 + k v? [1 mark]

- ☐ **A** What the controller needs is a roughly constant preview time, and the lag costs a fixed number of seconds
- ☐ **B** A faster robot is further from the path, so it needs a longer reach
- ☐ **C** It keeps the curvature command constant at all speeds
- ☐ **D** It stops the look-ahead point running off the end of the path

> **Answer:** A. A look-ahead of L at speed v is L / v seconds of preview. Holding that roughly constant means looking further when going faster.

The task: chase the look-ahead point

The tape runs straight from (40, 40) to (40, 120), round a quarter circle of radius 40 centred on (80, 120), then straight to (150, 160). Follow it with pure pursuit, plot `off path` and `speed`, print `path:` (the length of the path) and `drove:` (how far the robot actually travelled), and stop at the far end.

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

DT = 0.1
START = (40.0, 40.0)
CRUISE, LOOK = 11.0, 18.0
```

The hint students can ask for: Project the robot onto the path to get the arc length it has reached, then take the point that much plus the look-ahead further along, and steer the velocity vector at it. The look-ahead is the only number with any real choice in it, so try a few and watch what each one costs.

A solution

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

DT = 0.1
V_MAX, V_LAT = 20.0, 15.0
START = (40.0, 40.0)
CRUISE, LOOK = 11.0, 18.0

arc = [(80 + 40 * math.cos(math.radians(180 - k * 11.25)),
        120 + 40 * math.sin(math.radians(180 - k * 11.25))) for k in range(9)]
PATH = [(40.0, 40.0)] + arc + [(150.0, 160.0)]

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]
print("path:", round(TOTAL, 1))

def project(px, py):
    """the arc length of the nearest point on the path, and how far off the path we are"""
    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))

drove = 0.0
last = position()
for tick in range(650):
    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)) * max(0.4, 1.0 - off / 15.0)
    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))
```

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