ANSWERS

U9.5 Potential fields

Planning · University · about 35 min

What this lesson is about

The goal pulls, the obstacles push, and the robot slides downhill into a local minimum.

- The two terms
- The local minimum
- What people do about it
- The scene here

Questions

7 marks in all

1. What does this program print? [1 mark]

```
K, REACH, CAP = 120000.0, 45.0, 60.0
for d in (10.0, 30.0, 50.0):
    m = 0.0
    if d < REACH:
        m = min(CAP, K * (1.0 / d - 1.0 / REACH) / (d * d))
    print(d, round(m, 2))
```

Answer:

```
10.0 60.0
30.0 1.48
50.0 0.0
```

At 30 cm the push is 120000 x (1/30 - 1/45) / 900 = 1.48. At 10 cm the formula gives 93.3 and the cap holds it to 60, and at 50 cm it is beyond the reach, so zero.

2. What does this program print? [1 mark]

```
import math
ox, oy, ow, oh = 80.0, 10.0, 30.0, 90.0
for x, y in ((60.0, 40.0), (120.0, 120.0)):
    nx, ny = min(max(x, ox), ox + ow), min(max(y, oy), oy + oh)
    print((nx, ny), round(math.hypot(x - nx, y - ny), 2))
```

Answer:

```
(80.0, 40.0) 20.0
(110.0, 100.0) 22.36
```

Clamping each axis into the rectangle gives the nearest point: (80, 40), 20 cm away, and the corner (110, 100), sqrt(10 squared + 20 squared) = 22.36 cm away.

3. Why can a local minimum in a potential field not be tuned away? [1 mark]

☐ **A** It comes from adding the fields together, and a method that only sees the gradient where it stands cannot be complete

☐ **B** It is caused by rounding in the arithmetic, which smaller time steps would fix

☐ **C** It only happens when the repulsion constant is too large

☐ **D** It only happens when the goal is inside the reach of an obstacle

> **Answer:** A. The sum of a pull and several pushes can be zero away from the goal. A field will stop there and cannot tell you why.

4. Which of these situations commonly trap a potential field controller? [1 mark]

Tick every answer that is true.

☐ **A** A wall square across the line to the goal

☐ **B** A U shaped obstacle opening towards the robot

☐ **C** Two obstacles with a gap between them

☐ **D** A single block with the goal up and to one side of it

> **Answer:** A, B, C. Opposite pull and push cancel, both sides of a U push back, and the pushes across a gap close it. With the goal to one side the forces are not opposite, so the robot slides round.

5. What do modern navigation stacks do with potential fields? [1 mark]

☐ **A** Use a field or similar reactive method as a local layer following a route from a global planner such as A*

☐ **B** Use them as the only planner, with random walks when they stall

☐ **C** Nothing: they have been abandoned entirely

☐ **D** Use them to build the occupancy grid

> **Answer:** A. The global planner knows the whole free space; the local layer dodges whatever wanders into the way. Each does the job it is good at.

6. Why does the attraction have a FLOOR rather than easing smoothly all the way to zero? [1 mark]

☐ **A** Below about 15 percent of full command the motors do not turn, so a pull that fades to zero leaves the robot parked short of the goal

☐ **B** Without it the robot orbits the goal

☐ **C** It prevents the repulsion from blowing up near a wall

☐ **D** It makes the field free of local minima

> **Answer:** A. Keep the pull above the dead band and stop on distance instead.

7. Why does the repulsion use the factor (1/d - 1/d0) rather than just 1/d? [1 mark]

☐ **A** It falls smoothly to zero at the edge of the reach d0, instead of switching off with a jolt

☐ **B** It stops the push blowing up close to the obstacle

☐ **C** It makes the push point towards the goal

☐ **D** It makes the field a navigation function

> **Answer:** A. At d = d0 the factor is exactly zero. The push still blows up close in, which is why it is capped.

The task: slide round the block

Drive to the green corner at (170, 160) using a potential field and nothing else: no route, no grid, just a velocity worked out fresh each tick from the pull and the push. Plot `pull` and `push`, the size of each part, and do not touch the block or the mat edges.

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

DT = 0.1
V_MAX, V_LAT = 20.0, 15.0
START = (30.0, 40.0)
GOAL = (170.0, 160.0)
BLOCK = (80.0, 10.0, 30.0, 90.0)
```

The hint students can ask for: Every tick, work out a velocity rather than a route: a pull towards (170, 160) and a push away from the nearest point of the block and of each mat edge, added together, capped at a sensible speed and turned into drive() with the inverse kinematics from U2. Plot the size of each part so you can see the push take over as the robot closes on the block.

A solution

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

DT = 0.1
V_MAX, V_LAT = 20.0, 15.0
START = (30.0, 40.0)
GOAL = (170.0, 160.0)
BLOCK = (80.0, 10.0, 30.0, 90.0)
REACH = 45.0          # how far the push from an obstacle carries
K_PUSH = 120000.0     # how hard it pushes
TOP = 14.0            # the fastest the robot is allowed to go

def nearest_on(rect, x, y):
    ox, oy, ow, oh = rect
    return (min(max(x, ox), ox + ow), min(max(y, oy), oy + oh))

for tick in range(600):
    px, py = position()
    x, y = START[0] + px, START[1] + py
    dx, dy = GOAL[0] - x, GOAL[1] - y
    gap = math.hypot(dx, dy)
    if gap < 8.0:
        break

    # the pull: towards the goal, easing off as it arrives but never below the motors' dead
band
    strength = min(13.0, 4.0 + 0.3 * gap)
    ax, ay = strength * dx / gap, strength * dy / gap

    # the push: away from the nearest point of the block and of each mat edge
    rx = ry = 0.0
    for cx, cy in (nearest_on(BLOCK, x, y), (x, 0.0), (x, 200.0), (0.0, y), (200.0, y)):
        ox, oy = x - cx, y - cy
```

```
        d = math.hypot(ox, oy)
        if d < 1e-6 or d > REACH:
            continue
        m = min(60.0, K_PUSH * (1.0 / d - 1.0 / REACH) / (d * d))
        rx += m * ox / d
        ry += m * oy / d

    vx, vy = ax + rx, ay + ry
    speed = math.hypot(vx, vy)
    if speed > TOP:
        vx, vy = vx * TOP / speed, vy * TOP / speed
    plot("pull", strength)
    plot("push", math.hypot(rx, ry))

    h = math.radians(heading())
    body_x = vx * math.cos(h) - vy * math.sin(h)
    body_y = vx * math.sin(h) + vy * math.cos(h)
    spin = (heading() + 180) % 360 - 180
    drive(100 * body_y / V_MAX, 100 * body_x / V_LAT, max(-30.0, min(30.0, -0.8 * spin)))
    wait(DT)
stop()
print("arrived")
```

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