ANSWERS

U9.7 Project: plan a route and drive it

Planning · University · about 55 min

What this lesson is about

A plan computed on board, shortened, and then followed across a mat with two walls in it.

- The shape of the program
- Why shortcut at all
- Following waypoints
- When it goes wrong
- What comes next

Questions

6 marks in all

1. Put the stages of the plan and drive program in order. [1 mark]

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

___ `Inflate the walls and the mat edges to build the configuration space`
___ `Print the plan length`
___ `Run A* on the 40 by 40 grid with eight neighbours and the octile heuristic`
___ `Shortcut the grid path down to its corners`
___ `Follow the waypoints with the inverse kinematics`

Answer:

```
Inflate the walls and the mat edges to build the configuration space
Run A* on the 40 by 40 grid with eight neighbours and the octile heuristic
Shortcut the grid path down to its corners
Print the plan length
Follow the waypoints with the inverse kinematics
```

Planning is separate from driving and happens first, producing an artefact you can inspect before a wheel turns.

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

```
import math

def free(x, y):
    return not (40.0 <= x <= 60.0 and y <= 45.0)

def clear(a, b):
    d = math.hypot(b[0] - a[0], b[1] - a[1])
    n = max(2, int(d / 1.5))
    return all(free(a[0] + (b[0] - a[0]) * k / n, a[1] + (b[1] - a[1]) * k / n) for k in
range(n + 1))

path = [(0.0, 0.0), (0.0, 50.0), (50.0, 50.0), (100.0, 50.0), (100.0, 0.0)]
out, i = [path[0]], 0
while i < len(path) - 1:
    j = len(path) - 1
    while j > i + 1 and not clear(path[i], path[j]):
        j -= 1
    out.append(path[j])
    i = j
length = sum(math.hypot(q[0] - p[0], q[1] - p[1]) for p, q in zip(out, out[1:]))
print(len(out), round(length, 1))
```

Answer:

```
4 200.0
```

From (0, 0) every later point is hidden by the block, including (50, 50), whose diagonal clips its corner, so the next point kept is (0, 50). From there (100, 50) is visible, and then (100, 0). (50, 50) is spliced out: 4 points, 50 + 100 + 50 = 200 cm.

3. Why must the shortcut's line of sight test use the same inflated map as the search? [1 mark]

- ☐ **A** Tested against the raw obstacles, a shortcut can cut a corner the search was careful to avoid
- ☐ **B** The raw obstacles are not stored after inflation
- ☐ **C** The inflated map has fewer cells, so the test is faster
- ☐ **D** The search and the shortcut must give the same number of waypoints

Answer: A. The inflation is what makes the robot a point. A line that is clear only for a point with no body will scrape the wall.

4. The plan is inflated by 10 cm, although the chassis only needs 4.95 cm. Why? [1 mark]

- ☐ **A** The grid, the shortcut and the waypoint follower all introduce error, and the plan has to absorb it
- ☐ **B** The mat edges are further away than the walls
- ☐ **C** A* cannot plan with margins below one cell
- ☐ **D** The walls are 10 cm thick

Answer: A. The page's table adds 2 to 5 cm for imperfect following and a cell or two for map error on top of the chassis.

5. The printed plan length is right, but the robot does not end up where it should. Where is the fault? [1 mark]

- ☐ **A** In the follower, not the planner, which is exactly why the plan is printed before driving
- ☐ **B** In the A* heuristic
- ☐ **C** In the inflation margin
- ☐ **D** In the grid cell size

> **Answer:** A. Printing the plan first separates the two problems: a sound plan with a bad drive points at the following code.

6. The robot drives straight past the final waypoint. What is the likely cause? [1 mark]

- ☐ **A** It is fast enough to pass through the arrival tolerance between two ticks, so the speed should ease down as the gap closes
- ☐ **B** A* returned a suboptimal path
- ☐ **C** The shortcut removed the final waypoint
- ☐ **D** The inflation closed the corridor

> **Answer:** A. The arrival test is on distance. A robot moving several centimetres per tick can jump over a small tolerance.

The task: plan a route and drive it

Plan a route from (30, 30) to the green corner at (170, 170), print `plan:`, its length in centimetres, and then drive it. Do not touch either wall or the edge of the mat.

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

DT = 0.1
CELL = 5.0
N = 40
INFLATE = 10.0
V_MAX, V_LAT = 20.0, 15.0
START = (30.0, 30.0)
GOAL = (170.0, 170.0)
WALLS = [(70.0, 0.0, 8.0, 115.0), (125.0, 85.0, 8.0, 115.0)]
```

The hint students can ask for: Plan first and print the length, then drive. Inflate by more than the chassis radius, because the robot follows a plan approximately, and shorten the grid path by joining any two waypoints that can see each other through the inflated map. Then drive waypoint to waypoint with the inverse kinematics, holding the heading near zero so the frames stay easy.

A solution

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

DT = 0.1
CELL = 5.0
N = 40
INFLATE = 10.0        # chassis radius, plus room for how loosely the robot follows a plan
V_MAX, V_LAT = 20.0, 15.0
START = (30.0, 30.0)
GOAL = (170.0, 170.0)
WALLS = [(70.0, 0.0, 8.0, 115.0), (125.0, 85.0, 8.0, 115.0)]
STEPS = [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (1, -1), (-1, 1), (-1, -1)]

def free(x, y):
    if x < INFLATE or y < INFLATE or x > 200 - INFLATE or y > 200 - INFLATE:
        return False
    for ox, oy, ow, oh in WALLS:
        if ox - INFLATE <= x <= ox + ow + INFLATE and oy - INFLATE <= y <= oy + oh +
INFLATE:
            return False
    return True

def clear(a, b):
    d = math.hypot(b[0] - a[0], b[1] - a[1])
    n = max(2, int(d / 1.5))
    for k in range(n + 1):
        t = k / n
        if not free(a[0] + (b[0] - a[0]) * t, a[1] + (b[1] - a[1]) * t):
            return False
```

```
    return True

grid = [[not free(i * CELL + CELL / 2, j * CELL + CELL / 2) for j in range(N)] for i in
range(N)]
start = (int(START[0] / CELL), int(START[1] / CELL))
goal = (int(GOAL[0] / CELL), int(GOAL[1] / CELL))

def octile(c):
    dx, dy = abs(c[0] - goal[0]), abs(c[1] - goal[1])
    return CELL * (max(dx, dy) + (math.sqrt(2) - 1) * min(dx, dy))

g = {start: 0.0}
came = {start: None}
queue = [(octile(start), start)]
done = set()
while queue:
    _f, cur = heapq.heappop(queue)
    if cur in done:
        continue
    done.add(cur)
    if cur == goal:
        break
    for dx, dy in STEPS:
        nxt = (cur[0] + dx, cur[1] + dy)
        if not (0 <= nxt[0] < N and 0 <= nxt[1] < N) or grid[nxt[0]][nxt[1]]:
            continue
        step = CELL * math.hypot(dx, dy)
        if g[cur] + step < g.get(nxt, 1e18):
            g[nxt] = g[cur] + step
            came[nxt] = cur
            heapq.heappush(queue, (g[nxt] + octile(nxt), nxt))

cells, cur = [], goal
while cur is not None:
    cells.append(cur)
    cur = came[cur]
cells.reverse()
points = [(c[0] * CELL + CELL / 2, c[1] * CELL + CELL / 2) for c in cells]

# shorten it: join the furthest pair of waypoints that can still see each other
route = [points[0]]
i = 0
while i < len(points) - 1:
    j = len(points) - 1
    while j > i + 1 and not clear(points[i], points[j]):
        j -= 1
    route.append(points[j])
    i = j

length = sum(math.hypot(b[0] - a[0], b[1] - a[1]) for a, b in zip(route, route[1:]))
print("plan:", round(length, 1))
print("waypoints:", len(route))

def here():
    px, py = position()
    return START[0] + px, START[1] + py

for wx, wy in route[1:]:
    for tick in range(400):
```

```
        x, y = here()
        dx, dy = wx - x, wy - y
        gap = math.hypot(dx, dy)
        if gap < 4.0:
            break
        speed = min(13.0, 3.0 + 0.5 * gap)
        vx, vy = speed * dx / gap, speed * dy / gap
        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()
x, y = here()
print("stopped at", round(x), round(y))
```

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