ANSWERS

U9.6 Sampling: the RRT

BugBotLab

Planning · University · about 40 min

What this lesson is about

Throwing darts at the free space instead of enumerating it, and what probabilistic completeness does and does not promise.

- The RRT
- What it promises
- Randomness has consequences

Questions

6 marks in all

1. A robot on the 200 by 200 cm mat plans over (x, y, heading) with 10 cm cells and 10 degree heading steps. How many cells does the grid have? [1 mark]

> **Answer:** 14400. 20 x 20 positions times 36 headings is 14,400. At 5 cm and 5 degrees it is 40 x 40 x 72 = 115,200, eight times as many.

2. There is no route from the start to the goal. What does a plain RRT do? [1 mark]

- ☐ **A** It keeps sampling until it is stopped, and can never report that no route exists
- ☐ **B** It reports no route after it has sampled every cell
- ☐ **C** It returns the node nearest to the goal
- ☐ **D** It returns a route through the narrowest wall

> **Answer:** A. Probabilistic completeness only promises that the chance of finding an existing route tends to 1. It says nothing when there is none.

3. Why does an RRT grow outwards into unexplored space without being told to? [1 mark]

- ☐ **A** A sample is extended from the nearest node, and nodes on the edge of the tree have the largest Voronoi regions
- ☐ **B** The goal bias pulls the tree outwards
- ☐ **C** New nodes are only accepted if they are further from the start
- ☐ **D** The step size grows as the tree gets bigger

> **Answer:** A. The probability of a node being extended is proportional to the area of its Voronoi region, which is largest at the frontier of the tree.

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

```
def free(x, y):
    return not (60.0 <= x <= 80.0 and 0.0 <= y <= 100.0)

a, b = (50.0, 50.0), (90.0, 50.0)
print(free(*a), free(*b))
n = 20
points = [(a[0] + (b[0] - a[0]) * k / n, a[1] + (b[1] - a[1]) * k / n) for k in range(n +
1)]
print(all(free(x, y) for x, y in points))
```

Answer:

```
True True
False
```

Both end points are free, but the segment between them crosses the block from x = 60 to 80. Checking only the new node is the most common RRT bug.

5. The goal bias is set so the RRT samples the goal most of the time. What happens? [1 mark]

- ☐ **A** It behaves like greedy best first and gets stuck against a wall between it and the goal
- ☐ **B** It becomes optimal
- ☐ **C** It explores the whole space faster
- ☐ **D** It becomes deterministic and always finds the same route

Answer: A. A bias of 5 to 10 percent is usual. Without any the tree wanders; with too much it keeps trying to grow straight at the goal.

6. Which of these statements are true? [1 mark]

Tick every answer that is true.

- ☐ **A** More samples do not make a plain RRT converge to the optimal path
- ☐ **B** RRT* rewires nearby nodes and is asymptotically optimal
- ☐ **C** A PRM suits a fixed map with many different start and goal queries
- ☐ **D** Shortcutting should test lines of sight against the raw, uninflated obstacles

Answer: A, B, C. The plain RRT provably converges to something non-optimal, RRT* fixes that at more cost per sample, and a PRM samples once for many queries. A shortcut must use the same inflated map, or it cuts corners.

The task: grow a tree into the free space

Build an RRT from (30, 30) to (170, 170) through the same two walls, inflated by 10 cm. Print `nodes:`, how many nodes the tree had when it reached the goal, and `path:`, the length in centimetres of the route through the tree.

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

INFLATE = 10.0
STEP = 12.0
BIAS = 0.1
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: Start the tree at (30, 30). Each round, pick a random point on the mat (and now and then the goal itself), find the nearest node, step about 12 cm from it towards the sample, and keep the new node only if the little segment misses every inflated wall. Stop when a node is within one step of (170, 170) with a clear line to it, then walk the parents back.

A solution

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

INFLATE = 10.0
STEP = 12.0
BIAS = 0.1
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)]

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):
    """The edge is checked by sampling it: a straight line between two free points can
still cross a wall."""
    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
```

```
nodes = [START]
parent = [None]
found = False
for step in range(3000):
    if random.random() < BIAS:
        sx, sy = GOAL
    else:
        sx, sy = random.uniform(0, 200), random.uniform(0, 200)
    near = 0
    best = 1e18
    for k in range(len(nodes)):
        d = (nodes[k][0] - sx) ** 2 + (nodes[k][1] - sy) ** 2
        if d < best:
            best, near = d, k
    nx, ny = nodes[near]
    d = math.hypot(sx - nx, sy - ny)
    if d < 1e-6:
        continue
    reach = min(STEP, d)
    new = (nx + (sx - nx) / d * reach, ny + (sy - ny) / d * reach)
    if not clear((nx, ny), new):
        continue
    nodes.append(new)
    parent.append(near)
    if math.hypot(new[0] - GOAL[0], new[1] - GOAL[1]) <= STEP and clear(new, GOAL):
        nodes.append(GOAL)
        parent.append(len(nodes) - 2)
        found = True
        break

route, k = [], len(nodes) - 1
while k is not None:
    route.append(nodes[k])
    k = parent[k]
route.reverse()
length = sum(math.hypot(b[0] - a[0], b[1] - a[1]) for a, b in zip(route, route[1:]))
print("found:", found)
print("nodes:", len(nodes))
print("path:", round(length, 1))
```

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