TP1

🎓 Fabio Fernandes
Lorrayne Cristine

1. Visão Geral

Este código simula a geração de uma árvore vascular 2D usando o método de **Otimização de Crescimento por Condução** (CCO) para distribuir vasos de maneira eficiente em um espaço. A árvore é composta por **nós** conectados por **vasos**. O objetivo é adicionar novos nós terminais à árvore existente enquanto minimiza o custo total da estrutura, que depende do comprimento e diâmetro dos vasos.

2. Estruturas Principais

2.1. Classe `Node`

Representa um nó na árvore.

- **Atributos:**
  - `x`, `y`: Coordenadas do nó no espaço 2D.
  - `flow`: Fluxo associado ao nó.
  - `parent`: Referência ao nó pai.
  - `children`: Lista de nós filhos conectados a este nó.
- **Métodos:**
  - `position()`: Retorna a posição do nó como um array NumPy.

2.2. Classe `Vessel`

Representa um vaso que conecta dois nós (pai e filho).

- **Atributos:**
  - `parent`: Nó pai.
  - `child`: Nó filho.
  - `length`: Comprimento do vaso (calculado).
  - `diameter`: Diâmetro do vaso (calculado).
- **Métodos:**
  - `compute_length()`: Calcula o comprimento do vaso usando a distância euclidiana entre o nó pai e o nó filho.
  - `compute_diameter()`: Calcula o diâmetro do vaso. No código atual, é simplificado como inversamente proporcional ao número de filhos do nó pai.

2.3. Classe `Tree`

Representa a árvore vascular em si.

- **Atributos:**
  - `root`: Nó raiz da árvore.

- `vessels`: Lista de todos os vasos na árvore.
- `nodes`: Lista de todos os nós na árvore.
- `kd_tree`: KD-Tree usada para buscas rápidas de vizinhos próximos.
- `collision_radius`: Raio de colisão para evitar sobreposição de nós.
- `gamma`: Parâmetro para controle de custo na função de CCO.

- **Métodos:**
  - `add_terminal_node(x, y, flow)`: Adiciona um novo nó terminal na posição `(x, y)` com fluxo `flow`.
  - `find_nearest_node(x, y)`: Encontra o nó mais próximo da posição `(x, y)` usando a KD-Tree, pois ajuda a encontrar rapidamente o nó existente mais próximo para conectar o novo nó. .
  - `find_valid_parent_node(x, y, initial_node)`: Encontra um nó pai válido para o novo nó, que não cause colisões e respeite as restrições.
  - `check_collision(x, y, parent_node)`: Verifica se adicionar um nó na posição `(x, y)` causaria colisão.
  - `do_edges_intersect(p1, p2, q1, q2)`: Verifica se dois segmentos de linha (definidos pelos pontos `p1, p2` e `q1, q2`) se interceptam.
  - `plot_tree()`: Plota a árvore usando `matplotlib`.
  - `compute_total_cost()`: Calcula o custo total da árvore baseado no comprimento e diâmetro dos vasos.

3. Otimização de Crescimento por Condução (CCO)

3.1. O que é o CCO?

O método de Otimização de Crescimento por Condução (CCO) é uma técnica usada para simular o crescimento de árvores vasculares. O objetivo é minimizar o custo total da estrutura vascular, considerando o comprimento dos vasos e o fluxo através deles.

3.2. Como o CCO foi implementado?

No código fornecido:

1. **Cálculo de Custo Total (`compute_total_cost`)**:
   - A função `compute_total_cost` calcula o custo total da árvore, que é a soma dos custos de todos os vasos. O custo de um vaso é proporcional ao seu comprimento multiplicado pelo diâmetro elevado a uma potência (`gamma`), onde `gamma` é um parâmetro de controle.
2. **Seleção de Nó Pai Válido (`find_valid_parent_node`)**:
   - O método CCO é aplicado na função `find_valid_parent_node`. Aqui, o algoritmo busca encontrar um nó pai para o novo nó terminal de forma a minimizar o custo total. Isso envolve verificar todos os nós disponíveis e avaliar se a adição do novo nó causaria colisões ou não.
3. **Verificação de Colisão (`check_collision`)**:
   - Antes de adicionar um novo nó, o algoritmo verifica se ele estaria colidindo com outros nós ou vasos. Isso garante que a estrutura resultante não tenha sobreposição, mantendo a árvore fisicamente realista.

4. Fluxo Geral de Execução

1. **Criação da Árvore**: O código começa criando um nó raiz central.
2. **Geração de Pontos Terminais**: Pontos terminais são gerados aleatoriamente no espaço.

3. **Adição de Nós**: Para cada ponto terminal, o nó mais próximo é encontrado e, após validações de colisão e custos, um novo nó é adicionado à árvore.
4. **Otimização Local**: A posição dos nós pode ser ajustada localmente para minimizar o custo total da árvore.
5. **Visualização**: A árvore final é plotada para visualização.

5. Conclusão

Este código exemplifica a aplicação do método CCO na geração de uma árvore vascular 2D, usando estruturas eficientes como a KD-Tree para melhorar a performance. As técnicas descritas podem ser aplicadas em várias áreas que requerem a modelagem de estruturas ramificadas, como biologia, redes de distribuição, e computação gráfica.

6. Referências

- PREPARATA, F. P.; SHAMOS, M. I. *Computational Geometry: An Introduction*. Berlin: Springer-Verlag, 1985.
- ZAMIR, M. *The Physics of Pulsatile Flow*. New York: Springer, 2000.
- BENTLEY, J. L. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, v. 18, n. 9, p. 509-517, 1975.
- MURRAY, C. D. The Physiological Principle of Minimum Work: I. The Vascular System and the Cost of Blood Volume. *Proceedings of the National Academy of Sciences*, v. 12, n. 3, p. 207-214, 1926.
- SCHREINER, W.; BUXBAUM, P. F. Computer-Optimization of Vascular Trees. *IEEE Transactions on Biomedical Engineering*, v. 40, n. 5, p. 482-491, 1993.
- KERAUTRET, B. et al. OpenCCO: An Implementation of Constrained Constructive Optimization for Generating 2D and 3D Vascular Trees. v. 13, p. 258--279, 1 nov. 2023.
- https://github.com/adelphin/vascularTree-CCO/tree/master