Instituto Tecnológico de Aeronáutica – ITA
Planejamento e Controle para Robótica Móvel – CM-202
Laboratório 2 – Planejamento de Caminho com RRT

Professor: Marcos Ricardo Omena de Albuquerque Maximo

20 de agosto de 2021

Observação: por questões de compatibilidade, este laboratório deve ser feito utilizando MATLAB 2021a.

1 Introdução

Neste laboratório, implementa-se planejamento de caminhos usando a técnica de *Rapidly-Exploring Random Tree* (RRT).

Seja o espaço de estados $\mathcal{X} \in \mathbb{R}^2$ e o espaço ocupado pelos obstáculos definido por $\mathcal{X}_{obs} \in \mathbb{R}^2$. O espaço livre então é dado por $\mathcal{X}_{free} = \mathcal{X} \backslash \mathcal{X}_{obs}$. Assim, o RRT é um algoritmo que tenta encontrar um caminho factível $\sigma : [0, 1] \to \mathcal{X}_{free}$ com $\sigma(0) = \mathbf{q}_0$ e $\sigma(1) = \mathbf{q}_{goal}$. Note que o RRT não encontra necessariamente um caminho de comprimento mínimo, mas apenas um caminho factível.

2 Tarefas

Para facilitar a implementação do laboratório, foi entregue um código base com os seguintes arquivos:

- `TreeNode.m`*: contém a classe `TreeNode`, que representa um nó de uma árvore.
- `Tree.m`*: contém a classe `Tree`, que representa uma árvore. A implementação utiliza um *array* de nós.
- `Obstacle.m`*: contém a classe `Obstacle`, que representa um obstáculo circular (definido pelo seu centro e raio).
- `RRT.m`*: contém a classe `RRT`, que implementa o algoritmo *Rapidly-Exploring Random Tree* (RRT).
- `circle.m`: função auxiliar para desenhar um círculo.

- `planPathRRT.m`: realiza um planejamento de caminho para um dado caso de teste e apresenta uma animação da construção da árvore.
- `monteCarloRRT.m`: realiza uma simulação de Monte Carlo para obter estatísticas do RRT em uma dada situação. Usa um valor de semente aleatória fixo para reprodutibilidade dos resultados.

Na lista acima, os arquivos marcados com asterisco serão editados pelo aluno, enquanto os demais já estão completamente implementados e não precisam ser editados. Recomenda-se que o aluno inicialmente analise o código entregue para se ambientar. Ademais, recomenda-se ao aluno que leia a documentação de cada método para entender suas entradas e saídas.

Por limitações do MATLAB, não usou-se uma implementação de árvore com uso de ponteiros, como é mais usual. Optou-se por uma implementação baseada em *array* de objetos do tipo `TreeNode`. Cada `TreeNode` contém o índice que seu pai ocupa no *array* que representa a árvore. No caso da raiz, dado que o MATLAB indexa a partir de 1, usa-se o índice 0 como convenção para `null`, i.e. para representar que a raiz não tem pai.

2.1 Implementação de Colisão com Obstáculo

Implemente o método `checkCollision()` da classe `Obstacle`. Como o obstáculo é circular, o teste de colisão é muito simples, basta verificar se o ponto em questão está dentro do círculo. Ademais, para simplificar, na implementação do RRT, não será feito teste de colisão com segmentos de retas, mas apenas com pontos amostrados.

2.2 Implementação de Busca de Vizinho Mais Próximo

Para simplificar, não será usada nenhuma estrutura de dados especial para operação de busca de vizinho mais próximo neste laboratório (obviamente, em uma implementação real, isto é fundamental para um bom desempenho do algoritmo). Assim, implementar o método `findNearest()` de `Tree` com uma busca "força bruta" em cima dos nós contidos no *array* `nodes`. Perceba que o número de nós atual é `numNodes`.

2.3 Implementação da Operação *Extend*

Implemente a operação de *extend* do RRT no método `extend()` de `TreeNode`. Perceba que esta operação calcula

$$\mathbf{q}_{new} = \begin{cases} \mathbf{q}_{rand}, \ ||\mathbf{q}_{rand} - \mathbf{q}_{nearest}|| \leq \Delta, \\ \mathbf{q}_{nearest} + \Delta \left(\mathbf{q}_{rand} - \mathbf{q}_{nearest}\right) / ||\mathbf{q}_{rand} - \mathbf{q}_{nearest}||, \ ||\mathbf{q}_{rand} - \mathbf{q}_{nearest}|| > \Delta, \end{cases} \tag{1}$$

em que $\mathbf{q}_{new}$, $\mathbf{q}_{rand}$ e $\mathbf{q}_{nearest}$ são o ponto adicionado na árvore, o ponto amostrado do espaço livre e o ponto na árvore mais próximo do ponto amostrado, respectivamente. Além disso, Δ é o passo usado no RRT.

2.4 Implementação do RRT

Implementar os seguintes métodos de `RRT`:

- `checkCollisionFree()`: verifica se um determinado ponto está no espaço livre $\mathcal{X}_{free}$, i.e. não colide com obstáculos.
- `sampleFreeSpace()`: amostra do espaço livre $\mathcal{X}_{free}$. Para isso, usar *reject sampling*, i.e. amostrar de $\mathcal{X}$ e então reamostrar (quantas vezes for necessário) se o ponto amostrado estiver em $\mathcal{X}_{obs}$.
- `planPath`: planeja um caminho usando RRT.

Destaca-se:

- Como é boa prática em desenvolvimento de *software*, use métodos e funções já implementados para implementar estes métodos.
- Por questões de reprodutibilidade, em `sampleFreeSpace()`, pede-se amostrar primeiramente do eixo x e depois do eixo y.
- Termine a busca se tiver chegado no objetivo. Considere que isso acontece quando $||\mathbf{q}_{new} - \mathbf{q}_{goal}|| < \varepsilon$. Sugere-se $\varepsilon = 10^{-3}$.
- Se um caminho não for encontrado até o número máximo de iterações, então retorne um caminho vazio e custo infinito (use `Inf` no MATLAB).
- Perceba que se o planejamento for terminado assim que chegar no objetivo, então o método `reconstructPath()` (já implementado) reconstrói o caminho no formato esperado. Além disso, este método ainda calcula o comprimento do caminho.

2.5 Análise da Implementação Desenvolvida

Para analisar sua implementação, fornece-se as funções `planPathRRT` e `monteCarloRRT`. Ambas as funções estão configuradas com os seguintes parâmetros: número máximo de iterações de 1,000, probabilidade de viés para o objetivo $p = 0{,}1$, $\Delta = 0{,}2$, $\mathcal{X} = [0, 10] \times [0, 10]$. Considere que um obstáculo é representado pelo par $([x\ y]^T, r)$, em que $[x\ y]^T$ e r são o centro e o raio, respectivamente. Assim, define-se os seguintes cenários:

'a'. $\mathbf{q}_0 = [1\ 1]^T$, $\mathbf{q}_{goal} = [9\ 9]^T$ e $\mathcal{X}_{obs} = \{([5\ 5]^T, 1)\}$.

'b'. $\mathbf{q}_0 = [1\ 9]^T$, $\mathbf{q}_{goal} = [9\ 1]^T$ e $\mathcal{X}_{obs} = \{([5\ 5]^T, 1), ([3\ 3]^T, 1), ([7\ 7]^T, 1)\}$.

'c'. $\mathbf{q}_0 = [1\ 9]^T$, $\mathbf{q}_{goal} = [9\ 1]^T$ e $\mathcal{X}_{obs} = \{([4\ 4]^T, 1), ([4\ 6]^T, 1), ([6\ 4]^T, 1), ([6\ 6]^T, 1)\}$.

'd'. $\mathbf{q}_0 = [9\ 9]^T$, $\mathbf{q}_{goal} = [9\ 1]^T$ e $\mathcal{X}_{obs} = \{([1\ 5]^T, 0{,}8), ([3\ 5]^T, 1{,}05), ([5\ 5]^T, 1{,}05), ([7\ 5]^T, 1{,}05), ([9\ 5]^T, 1{,}05)\}$.

Para a `planPathRRT`, pede-se:

- Usando o caso 'a', execute o planejamento usando $p \in \{0{,}01; 0{,}1; 0{,}9\}$. Inclua os gráficos gerados no relatório e discuta se você observa aquilo que esperava considerando o significado de p.

- Execute os casos ‘b’, ‘c’ e ‘d’ e inclua os gráficos no relatório.

Quanto à `monteCarloRRT`, pede-se:

- Execute os casos ‘a’, ‘b’, ‘c’ e ‘d’ e inclua uma tabela no relatório resumindo os resultados encontrados. Discuta se os resultados obtidos condizem com o que você esperava intuitivamente a partir do que pode observar dos cenários escolhidos.

Perceba que são executadas 200 execuções de Monte Carlo. Ao fim, as seguintes estatísticas são fornecidas:

1. Porcentagem de instâncias resolvidas.
2. Média de comprimento dos caminhos encontrados (i.e. ignora instâncias que não foram resolvidas).
3. Desvio padrão de comprimento dos caminhos encontrados.
4. Média do número de iterações até parar.
5. Desvio padrão do número de iterações até parar.

3 Instruções

- A entrega da solução desse laboratório consiste de arquivos de código (MATLAB e Simulink) e de um relatório (em `.pdf`), que devem ser submetidos no Google Classroom como um único arquivo `.zip`.
- Compactar todos os arquivos a serem submetidos em um único `.zip` (use obrigatoriamente `.zip`, e **não** outra tecnologia de compactação de arquivos) e anexe esse `.zip` no Google Classroom.
- Use o padrão de nome `<login_ga>_labX.zip`, em que `<login_ga>` é seu login `@ga.ita.br` no Classroom e X é o número do laboratório em questão. Por exemplo, se o login do aluno for `marcos.maximo` e estiver entregando o laboratório 1, o nome do arquivo deve ser `marcos.maximo_lab1.zip`.
- O relatório deve ser sucinto, preocupe-se apenas em incluir discussões e entregáveis solicitados no roteiro. Pede-se apenas um cuidado mínimo na elaboração do relatório: responder adequadamente as perguntas, incluir figuras diretamente no relatório (ao invés de deixar como arquivos separados), usar figuras de boa qualidade, colocar nomes nos eixos dos gráficos, colocar legenda para diferenciar curvas num mesmo gráfico etc.
- **Não** é permitido o uso de funções ou comandos prontos do MATLAB que realizem toda a funcionalidade atribuída a uma certa função cuja implementação foi solicitada. Entretanto, o uso destas funções para verificação das implementações realizadas é encorajado. Em caso de dúvida, consulte o professor.

- A criação de *scripts* e funções auxiliares no MATLAB para execução de experimentos e geração de gráficos é fortemente recomendada para facilitar seu trabalho. Porém, não há necessidade de entregar códigos auxiliares.
- **Não** há necessidade de copiar e colar o código no seu relatório, dado que você também submeterá os arquivos de código. Também **não** há necessidade de explicar sua implementação no relatório, a **não** ser que o roteiro tenha solicitado explicitamente. Porém, organizar e comentar o código é muito salutar, pois ele será o foco da correção.

4 Dicas

- Para calcular a norma do vetor `v`, use `norm(v)`.
- Para gerar um número aleatório entre 0 e 1 no MATLAB, use `rand()`.
- Este código utiliza Programação Orientada a Objetos (POO) em MATLAB. Embora POO seja uma técnica de programação complexa, o uso neste laboratório é bem básico. Caso não saiba como usar POO em MATLAB, recomenda-se estudar o *tutorial* entregue como Material Complementar.