**Instituto Tecnológico de Aeronáutica – ITA**

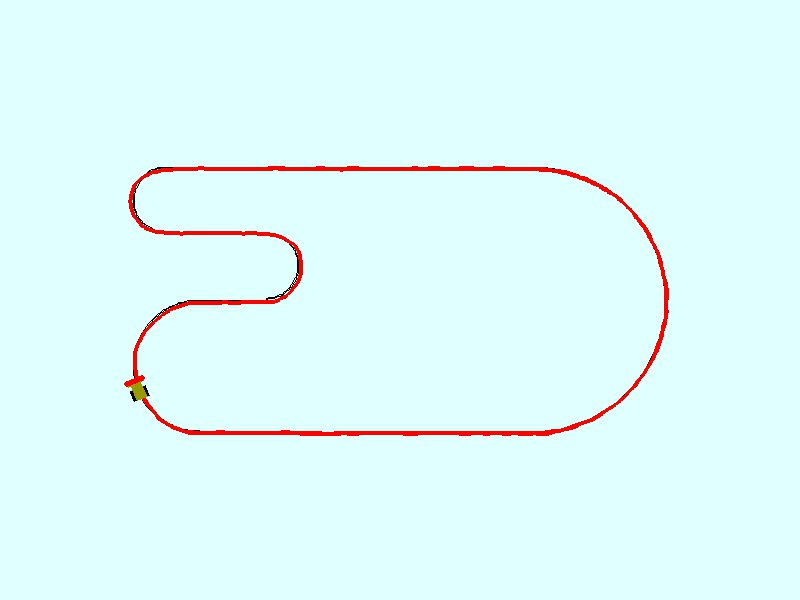
**Inteligência Artificial para Robótica Móvel – CT-213**

**Professor: Marcos Ricardo Omena de Albuquerque Maximo**

**Laboratório 11 (Aula 12) – Aprendizado por Reforço Livre de Modelo**

1. **Introdução**

Nesse laboratório, seu objetivo é implementar algoritmos de Aprendizado por Reforço (RL) Livre de Modelo, a saber Sarsa e Q-Learning, e com eles resolver o problema do robô seguidor de linha. A Figura 1 mostra a trajetória do robô após aprendizado com Q-Learning. Note que a trajetória obtida por você pode ser diferente, pois sua execução do algoritmo de aprendizado pode convergir para outro máximo local.



**Figura 1:** trajetória do robô seguidor de linha após aprendizado com Q-Learning.

1. **Descrição do Problema**

O problema ser resolvido é o aprendizado por reforço de um robô seguidor de linha. O controlador mantém velocidade linear constante (não sujeita a otimização/aprendizado nesse caso), enquanto utiliza uma política aprendida por aprendizado por reforço livre de modelo para escolher a velocidade angular.

em que é a velocidade angular comandada e é o erro de seguimento da linha. Perceba que esse problema possui espaços de estados e de ações contínuos. Como serão usados métodos tabulares, que são apenas capazes de lidar com espaços de estados e de ações discretos, é necessário discretizar o problema. Para isso, discretiza-se em 9 valores diferentes com a adição de 1 estado para representar linha não-detectada (totalizando 10 estados possíveis) e usa-se 9 valores diferentes para discretizar a ação (igualmente distribuídos entre - e ). Para mapear do estado contínuo para discreto, considera-se o estado discreto mais próximo.

Como recompensa, unicamente penalizar o robô devido ao desvio em relação à linha mostrou eficaz nesse caso:

em que é um fator de normalização igual à largura do sensor de linha, usado para que o valor de erro fique entre . Assim, no caso do robô detectar a linha, a reward máxima é 0 e a mínima é -1. No caso em que o robô simplesmente não detecta a linha (pois ela está fora do alcance do sensor), usa-se .

Perceba que não é possível comparar a política aprendida nesse caso com o controlador otimizado com PSO no lab 4, pois o simulador sofreu alterações importantes para esse lab (a saber, tornou-se a dinâmica “mais fácil” com remoção de atrasos e aumento da taxa de amostragem do controlador).

1. **Código Base**

O código base já implementa a simulação do seguidor de linha. Segue uma breve descrição dos arquivos fornecidos:

* constants.py: arquivo de constantes.
* line\_follower.py: implementa o robô seguidor de linha, tanto sua simulação quanto seu controlador.
* low\_pass\_filter.py: implementa um filtro de primeiro ordem discreto. Esse filtro é utilizado para simulador a dinâmica da roda do robô.
* main.py: arquivo principal, que roda a otimização do seguidor de linha.
* reinforcement\_learning.py: implementação dos algoritmos de Aprendizado por Reforço Livre de Modelo.
* simulation.py: implementa a simulação do robô seguidor de linha.
* test\_rl.py: teste com MDP simples do correto funcionamento dos algoritmos de Aprendizado por Reforço.
* track.py: classe que representa o circuito de linha.
* utils.py: diversas classes e métodos utilitários.

O foco da sua implementação nesse laboratório é o arquivo reinforcement\_learning.py. Além disso, é interessante usar o arquivo test\_rl.py para verificar sua implementação dos algoritmos antes de tentar o aprendizado de política no seguidor de linha.

1. **Tarefas**
   1. **Implementação dos algoritmos de RL**

Sua primeira tarefa é a implementação dos algoritmos de RL. Para isso, implemente os métodos marcados com “Todo” do arquivo reinforcement\_learning.py. Algumas dicas:

* Para simplificar, não há necessidade de escalonar ou .
* Por questões de código, decidiu-se usar a mesma interface de código para o Sarsa e o Q-Learning. Assim, perceba que:
  + o método get\_greedy\_action() do Sarsa na realidade executa uma política -greedy, dado que o Sarsa é um algoritmo *on-policy*.
  + O método learn() do Q-Learning recebe next\_action (ação A’ executada no estado S’) para manter compatibilidade com o Sarsa, porém o algoritmo não usa essa informação.
* A interface com os algoritmos de RL é um pouco diferente da mostrada nos slides. Essa forma de implementar foi escolhida por ser mais conveniente para o caso deste laboratório. O código para uso dessas classes pode ser visto em test\_rl.py:

| for i in range(num\_episodes):  state = np.random.randint(0, num\_states)  action = rl\_algorithm.get\_exploratory\_action(state)  for j in range(num\_iterations):  next\_state = dynamics(state, action, num\_states)  reward = reward\_signal(state, action, num\_states)  next\_action = rl\_algorithm.get\_exploratory\_action(next\_state)  rl\_algorithm.learn(state, action, reward, next\_state, next\_action)  state = next\_state  action = next\_action |
| --- |

* Use test\_rl.py para testar seus algoritmos de RL. O MDP de teste nesse *script* considera um “corredor” (“tabuleiro” unidimensional) de num\_states células. As ações são STOP (ficar parado), LEFT (mover-se para esquerda) e RIGHT (mover-se para direita). As ações são consideradas determinísticas, no sentido de que sempre são executadas com total certeza. Além disso, esse MDP considera que ocorre “*wrap*” nos extremos do corredor: quando se executa LEFT na célula mais à esquerda, o agente surge na célula mais à direita. Analogamente, executar RIGHT na célula mais à direita faz o agente surgir na célula mais à esquerda. Finalmente, o estado objetivo é a célula mais à direita, de modo o agente recebe recompensa -1 em todas as células, exceto na célula objetivo, em que recebe recompensa 0.
* Há uma linha indicada com “Todo” no *script* test\_rl.py que permite escolher entre os dois algoritmos. Coloque no seu relatório os resultados obtidos rodando esse *script* e discuta se eles são o que você espera intuitivamente.
* **Não** há necessidade de implementar o salvamento e carregamento do estado do algoritmo no disco (para ser capaz de recuperar o algoritmo caso ocorra alguma falha).
  1. **Aprendizado da política do robô seguidor de linha**

O aprendizado da política do robô seguidor de linha é executado através do arquivo main.py. Enquanto os algoritmos de RL não estiverem implementados, o robô ficará se movendo em círculos, pois a implementação “falsa” dos algoritmos de RL enviam sempre ação 0, que no caso da implementação significa girar com o máximo de velocidade para a esquerda. Além da execução da simulação em si, o código base fornece algumas opções de interação com o usuário através de teclas do teclado:

* A: ativa/desativa o modo acelerado (*accelerated mode*). Quando em modo acelerado, a simulação é executada até 200x mais rápido que tempo real (depende também da capacidade do seu computador). Você também pode controlar esse fator através das setas do teclado (para cima/para baixo aumenta/diminui em 1x, enquanto para direita/para esquerda aumenta/diminui em 10x).
* T: ativa/desativa o treinamento (aprendizado) do robô. Se o treinamento estiver desativado, o robô usa a melhor política encontrada pelo algoritmo de aprendizado até então (perceba que como Sarsa é *on-policy*, a melhor política dele ainda é -greedy). Além disso, se o treinamento estiver desativado, salva uma captura da tela no fim do episódio.
* P: plota gráfico do histórico do aprendizado (retorno de cada episódio). Também mostra graficamente a tabela Q (como uma imagem) e a política determinística que seria obtida através de greedy(Q) (perceba que isso não necessariamente é a política que o algoritmo de RL segue). Além disso, salva estes gráficos em formato .png para serem incluídos no relatório.

Lembre de sempre rodar a otimização em modo acelerado para evitar esperar tanto. Nos meus testes, com cerca de 20-30 episódios, o algoritmo já era capaz de completar a pista algumas vezes. Não há necessidade de esperar o algoritmo de RL convergir completamente para os propósitos deste laboratório. Assim, adote 500 episódios como suficiente para os propósitos deste laboratório.

Finalmente, destaco que mantive no código o circuito “simples” para caso alguém tenha muita dificuldade em otimizar no circuito “complexo”. Para fazer essa troca, troque a linha 209 para “track = create\_simple\_track()”. Por favor, fale comigo antes de entregar o lab caso precise fazer isso.

Inclua no seu relatório gráficos relativos ao aprendizado (os que são plotados quando se aperta P) e a melhor trajetória obtida durante o aprendizado (salva quando se desativa o treinamento). Discuta também o que você observou durante o processo de aprendizado. Ademais, faça isso para os dois algoritmos: Sarsa e Q-Learning.

A propósito, antes que me perguntem: o circuito não está quebrado nas curvas! Eu confesso que tentei arrumar isso, mas a pygame desenha arcos desse jeito mal-feito mesmo. O que importa mais é que a simulação que roda por debaixo está correta.

1. **Entrega**

A entrega consiste do código e de um relatório, submetida através do Google Classroom. Modificações nos arquivos do código base são permitidas, desde que o nome e a interface dos scripts “main” não sejam alterados. A princípio, não há limitação de número de páginas para o relatório, mas pede-se que seja sucinto. O relatório deve conter:

* Breve descrição em alto nível da sua implementação.
* Figuras que comprovem o funcionamento do seu código.

Por limitações do Google Classroom (e por motivo de facilitar a automatização da correção), entregue seu laboratório com todos os arquivos num único arquivo **.zip** (**não** utilize outras tecnologias de compactação de arquivos) com o seguinte padrão de nome: “**<login\_email\_google\_education>\_labX.zip**”. Por exemplo, no meu caso, meu login Google Education é **marcos.maximo**, logo eu entregaria o lab 11 como “**marcos.maximo\_lab11.zip**”. **Não** crie subpastas para os arquivos da sua entrega, **deixe todos os arquivos na “raiz” do .zip**. Os relatórios devem ser entregues em formato **.pdf**.

1. **Dicas**

* Para pegar o máximo elemento de certo array em numpy, faça:

max = np.max(array)

* Para pegar o índice do máximo elemento de certo array em numpy, faça:

index = np.argmax(array)