Laboratório 12 – Aprendizado por reforço livre de modelo

Inteligência Artificial para Robótica Móvel – CT-213

Aluno: Caio Graça Gomes

Professor: Marcos Ricardo Omena de Albuquerque Maximo

INTRODUÇÃO:

Nesse laboratório, teve-se por objetivo implementar dois algoritmos de aprendizado por reforço livre de modelo: *Sarsa* e *Q-Learning* para resolver o problema de um robô seguidor de linha.

METODOLOGIA E DESCRIÇÃO EM ALTO NÍVEL DO ALGORITMO UTILIZADO:

O problema consistiu no aprendizado por reforço de um robô seguidor de linha, implementando algoritmos do `reinforcement_learning.py`. O controlador mantém velocidade linear constante enquanto utiliza uma política aprendida pelos algoritmos *Sarsa* ou *Q-Learning* para a escolha da velocidade angular do robô.

$$\omega = \pi(\omega|e)$$

Sendo π a política aprendida, ω a velocidade angular comandada para o robô e e o erro so seguimento de linha. Como recompensa para o aprendizado, foi adotada uma tal que dependesse do quadrado do erro e.

$$reward = -(e/w_l)^2$$

Em que w_l é um fator de normalização de modo que o valor do erro fique no intervalo $[-1,\ 1]$. No caso em que o robô não detecta linha, foi utilizado $reward = -5$.

Com isso em mente, implementou-se as seguintes funções:

1) `greedy_action(q, state)`: retorna a ação “gulosa” para um dado estado;

2) `epsilon_greedy_action(q, state, epsilon)`: retorna a ação *epsilon greedy*, isto é, executa uma ação aleatória com probabilidade $epsilon$, e executa a ação gulosa com probabilidade $1 - epsilon$;
3) `get_greedy_action(self, state)` da classe `class Sarsa(RLAlgorithm)`: retorna a ação *epsilon greedy* para o *Sarsa*;
4) `learn(self, state, action, reward, next_state, next_action)` da classe `class Sarsa(RLAlgorithm)`: atualiza o valor para um dado estado e ação com base no próximo estado e próxima ação para o *Sarsa*, fazendo $Q(s,\ a) = Q(s,\ a) + \alpha(R + \gamma Q(s',\ a') - Q(s,\ a))$;
5) `get_greedy_action(self, state)` da classe `class QLearning(RLAlgorithm)`: retorna a ação *epsilon greedy* para o *Q-Learning*;
6) `learn(self, state, action, reward, next_state, next_action)` da classe `class QLearning(RLAlgorithm)`: atualiza o valor para um dado estado e ação com base no próximo estado para o *Q-Learning*, fazendo $Q(s,\ a) = Q(s,\ a) + \alpha(R + \gamma max(Q(s',\ a')) - Q(s,\ a))$.

Os algoritmos foram inicialmente testados em `test_rl.py` e posteriormente executados para o robô seguidor de linha em `main.py`.

RESULTADOS

Resultados do test_rl.py:

Sarsa:

Figura 1: Resultados do `test_rl.py` para o *Sarsa*.

```
Action-value Table:
[[ -9.52214289  -8.2983354  -10.29279418]
 [-10.16875939  -9.55020669 -11.30786141]
 [-11.12256172 -10.26120951 -11.15609937]
 [-11.6609     -11.3174249  -11.81130471]
 [-12.43246979 -12.21066556 -12.20327384]
 [-11.82649377 -11.82475809 -11.2512986 ]
 [-11.06150722 -11.57773166 -10.24523927]
 [-10.52183858 -11.27641836  -9.2941253 ]
 [ -9.51791689 -10.32052095  -8.47299655]
 [ -7.41746977  -8.39900898  -8.25659819]]
Greedy policy learnt:
[L, L, L, L, R, R, R, R, R, S]
```

Q-Learning:

Figura 2: Resultados do `test_rl.py` para o *Q-Learning*.

```
Action-value Table:
[[-1.99       -1.         -2.9701    ]
 [-2.96711133 -1.99       -3.93899874]
 [-3.50212588 -2.9701     -4.16923562]
 [-4.34927156 -3.94039898 -4.65371665]
 [-5.11574532 -4.89927611 -4.89942699]
 [-4.34851447 -4.95102106 -3.94039895]
 [-3.77341335 -3.98919749 -2.9701    ]
 [-2.96765049 -3.93902413 -1.99      ]
 [-1.99       -2.9701     -1.        ]
 [ 0.         -0.99       -0.99      ]]
Greedy policy learnt:
[L, L, L, L, L, R, R, R, R, S]
```

Conforme esperado, o *Q-Learning* obteve maiores valores na tabela, pois ele encontra o caminho ótimo, enquanto o *Sarsa* fica limitado pelo *epsilon greedy*.

Resultados do main.py:

Sarsa:

Figura 3: Trajeto ótimo obtido pelo algoritmo *Sarsa* após 500 iterações de treino.

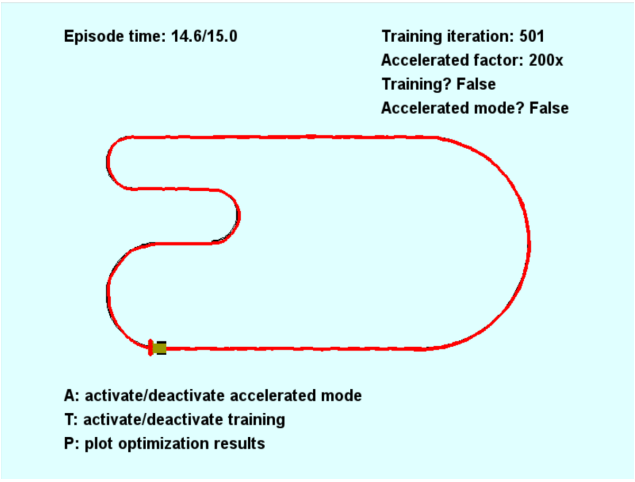


Figura 4: Gráfico da convergência da recompensa do robô usando *Sarsa*.

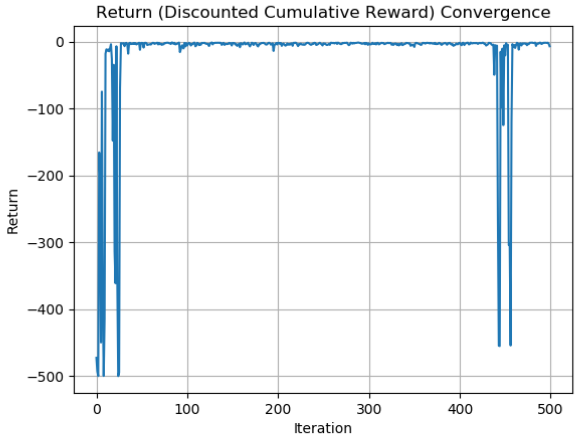


Figura 5: Tabela Q dos estados e respectivas ações usando *Sarsa*.

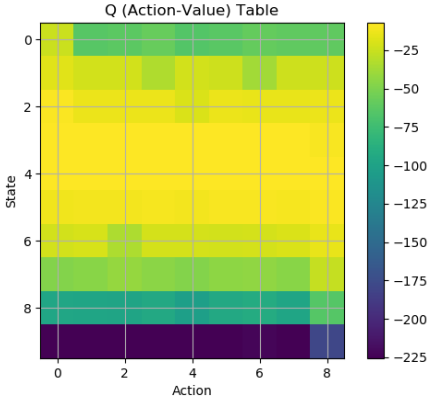


Figura 6: Tabela do *Greedy Policy* em Sarsa.

Q-Learning:

Figura 7: Trajeto ótimo obtido pelo *Q-Learning* após 500 iterações de treino.

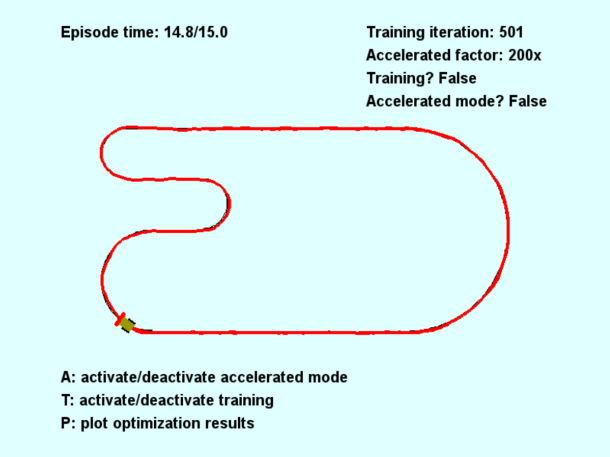


Figura 8: Gráfico da convergência da recompensa do robô usando *Q-Learning*.

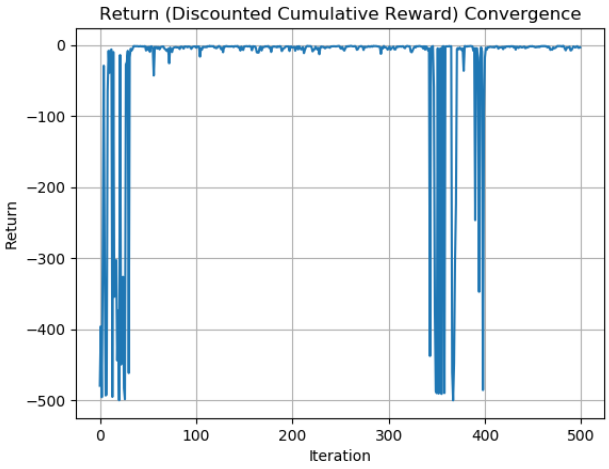


Figura 9: Tabela Q dos estados e respectivas ações usando *Q-Learning*.

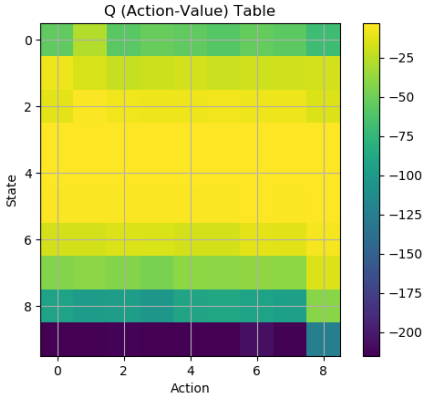
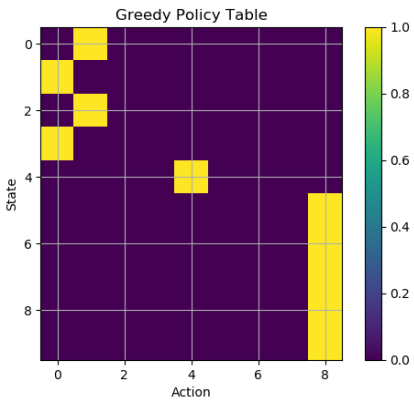


Figura 10: Tabela do *Greedy Policy* em Q-Learning.



Conforme esperado, o algoritmo *Q-Learning* obteve um resultado ótimo melhor, no entanto, ao encontrar a política ótima, percebe-se que o *Q-Learning* corre muitos riscos e foi bem comum observar o robô escapando da trajetória, apesar do *print* específico apresentado no relatório não mostrar isso. Isso não se concretizou com o *Sarsa*, onde o robô sempre permanecia na linha, pois adota uma política mais conservadora.