Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT213
Aluno: Vinícius José de Menezes Pereira

Relatório do Laboratório 4 - Otimização com Métodos Baseados em População

1 Breve Explicação em Alto Nível da Implementação

1.1 *Particle Swarm Optimization*

O Particle Swarm Optimization (pso) é um método de otimização baseado no movimento de migração dos pássaros. Nele, cada candidato à solução é chamado de partícula e a população é chamda de enxame. Ele é um método que não otimização que não é somente local (explotation), mas também visa explorar a população (exploration).

Dessa forma, inicialmente temos partículas aleatoriamente dispostas sobre um espaço limitado e com velocidade também aleatória, mas proporicional ao tamanho do espaço. A classe Particle foi inicializada com os atributos posição x limitada por lower-bound e upper-bound, velocidade v limitada por -delta e +delta, onde delta = (upper-bound)-(lower-bound), current-value, que é o atual valor do "custo"encontrado para a partícula, best-value é o melhor valor do "custo"encontrado para a partícula e best-position, que é a posição do melhor valor.

Já a classe do pso inicia com os hiperparâmetros, um vetor de partículas que é inicializado, um contador count, inicializado como zero, e a melhor partícula global best-global. Além disso, temos seus métodos que são executados em sequências até o numero máximo de avaliações na seguinte ordem: get-position-to-evaluate(), que retorna a posição de uma partícula para aquele contador que será avaliada e notify-evaluation(value), que dado um value, qualidade da partícula, atualiza os atributos best-value e best-positon da partícula da iteração. Caso todas as partículas já tenham sido analisadas, ou seja, self.count == self.hp.num-particles, notify-evaluation(value) chama a função advance-generation, que irá atualizar a posição e a velocidade das partículas da geração e determinar o best-global. Para determinar o best-global, para comparar o best-global com o best-value de cada partícula e selecionar o melhor best-value. Para atualizar a posição, basta adicionar a velocidade à posição antiga, o que é equivalente a calcular a posição depois de uma unidade de tempo. Para atualizar a velocidade, calcula-se a soma da velocidade antiga acrescida de um fator de inercia mais a diferença entre a melhor posição da partícula e sua posição atual multiplicada por um número aleatório entre 0 e 1 e um fator cognitivo, responsável pela explotation, mais a diferença entre a melhor posição da global e sua a posição atual multiplicada por um número aleatório entre 0 e 1 e um fator social, responsável pela exploration. A função get-best-position() retorna a posição do melhor global e a get-best-value retorna o melhor valor global.

Na simulation, a função de qualidade evaluate retornava a velocidade linar multiplicada pelo produto escalar entre a direção do vetor velocidade do robo e a tangente a linha a ser seguida menoso módulo do erro multiplicado pelo peso w, que nesse caso foi escolhido como 0.6, apresentando bons resultados. Caso não haja detecção da linha, adicionou-se uma heurística para que o erro suba bruscamente para o valor 1.6.

2 Figuras Comprovando Funcionamento do Código

2.1 Teste do *Particle Swarm Optimization*

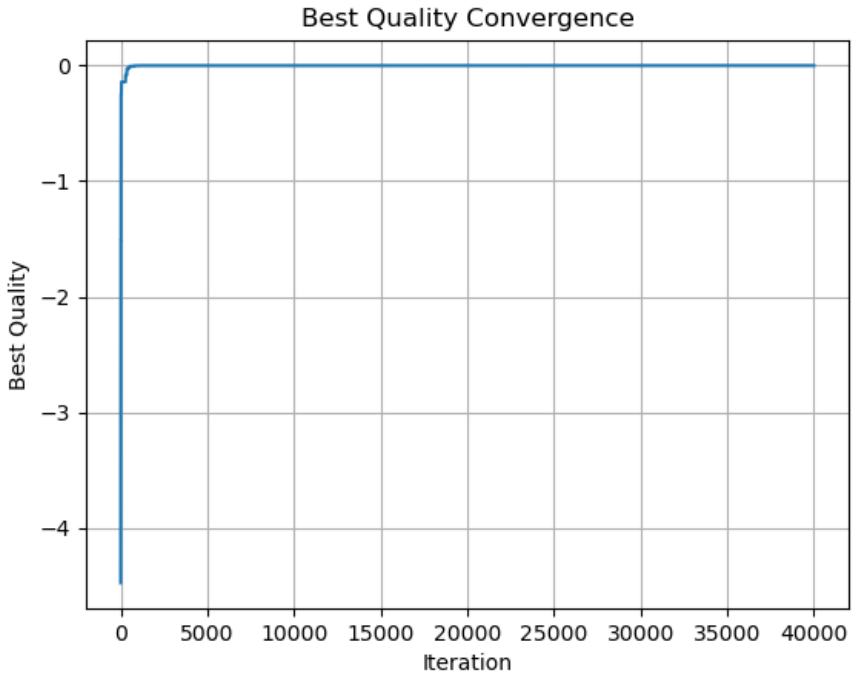

Figura 1: Best Quality Convergence

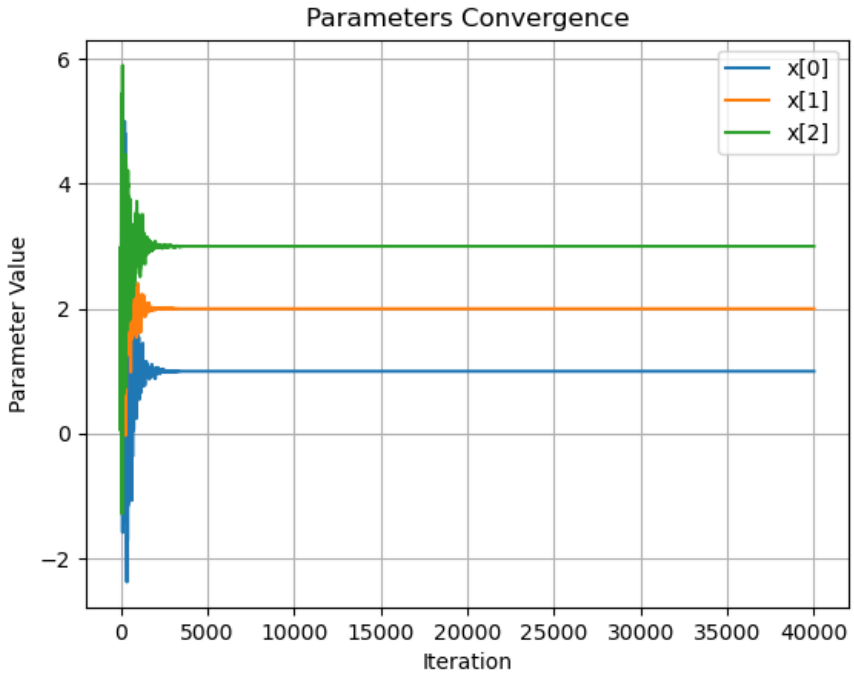

Figura 2: Best parameters convergence

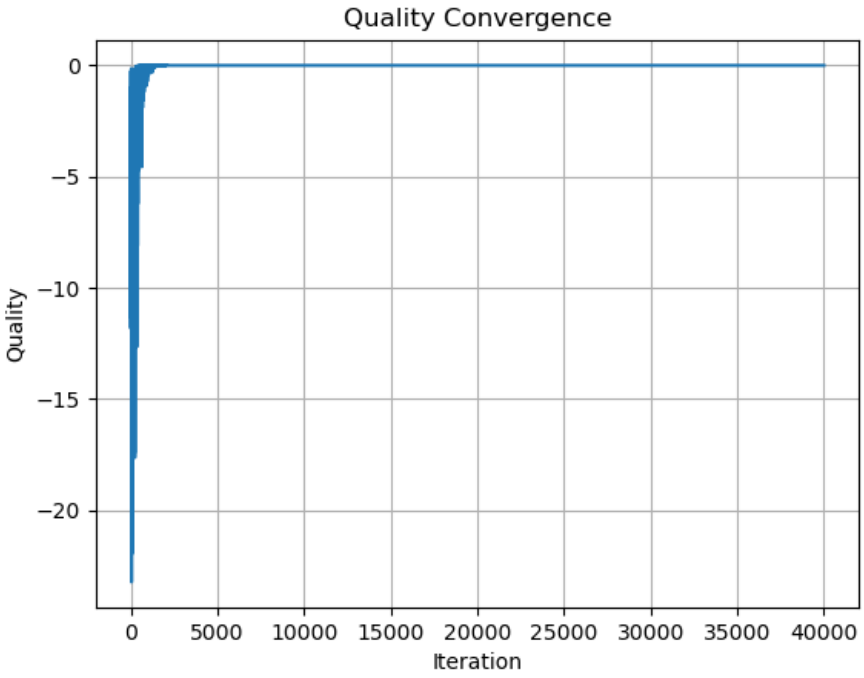

Figura 3: Best Quality Convergence

2.2 Otimização do controlador do robô seguidor de linha

2.2.1 Histórico de Otimização

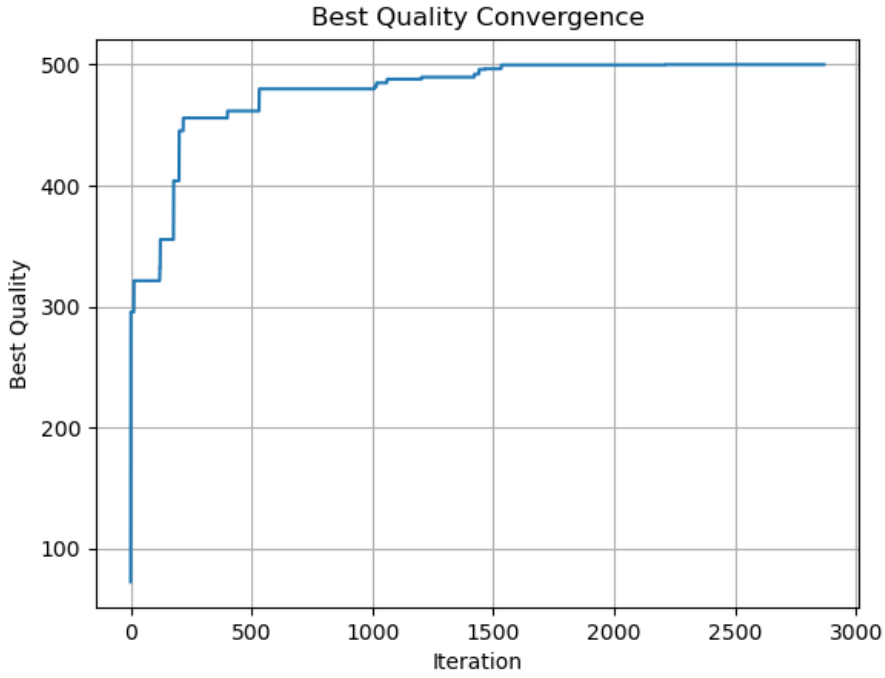

Figura 3: Best Convergence

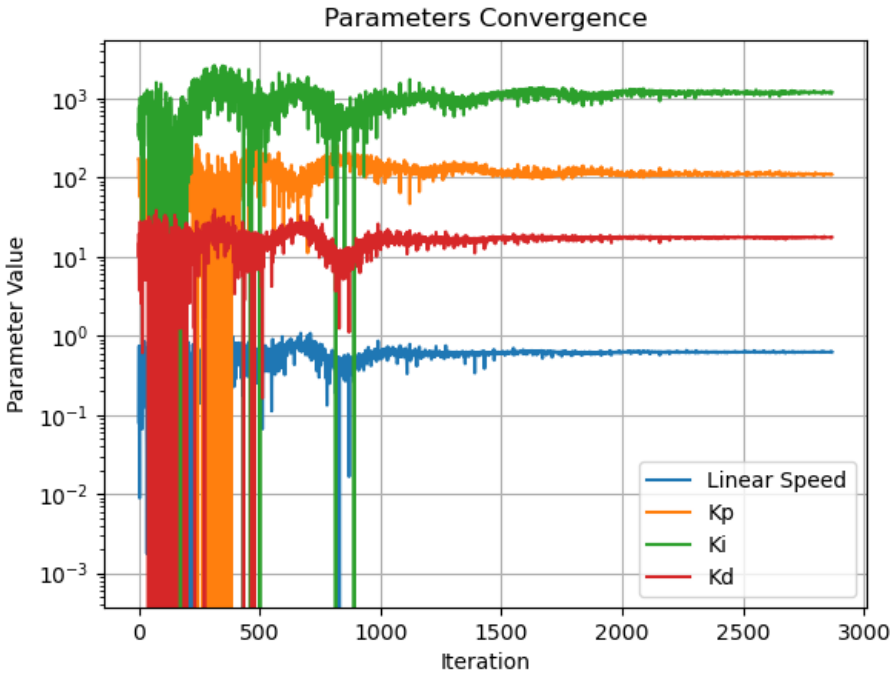

Figura 3: Best parameters

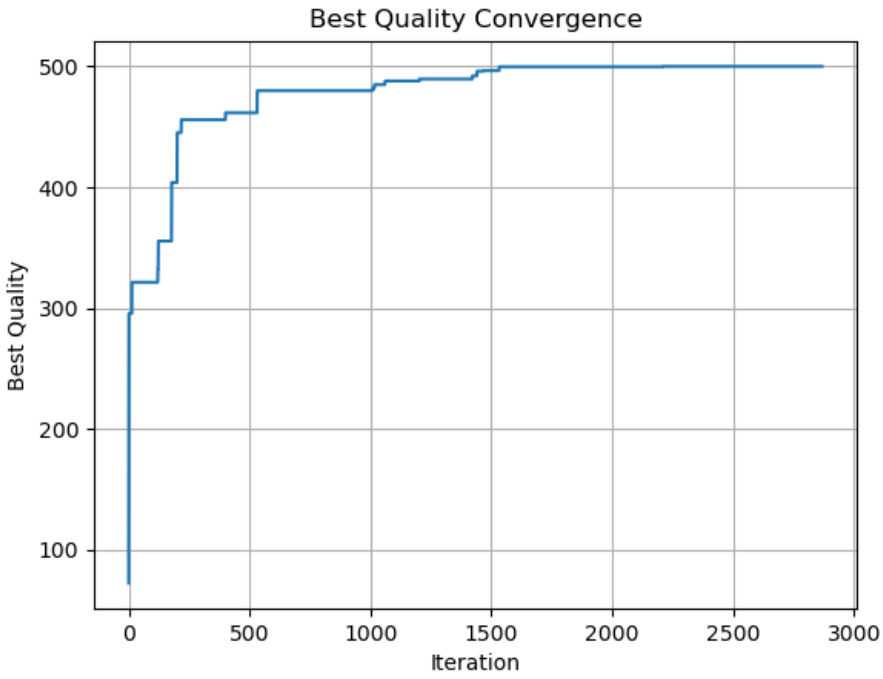

Figura 3: Best Quality Convergence

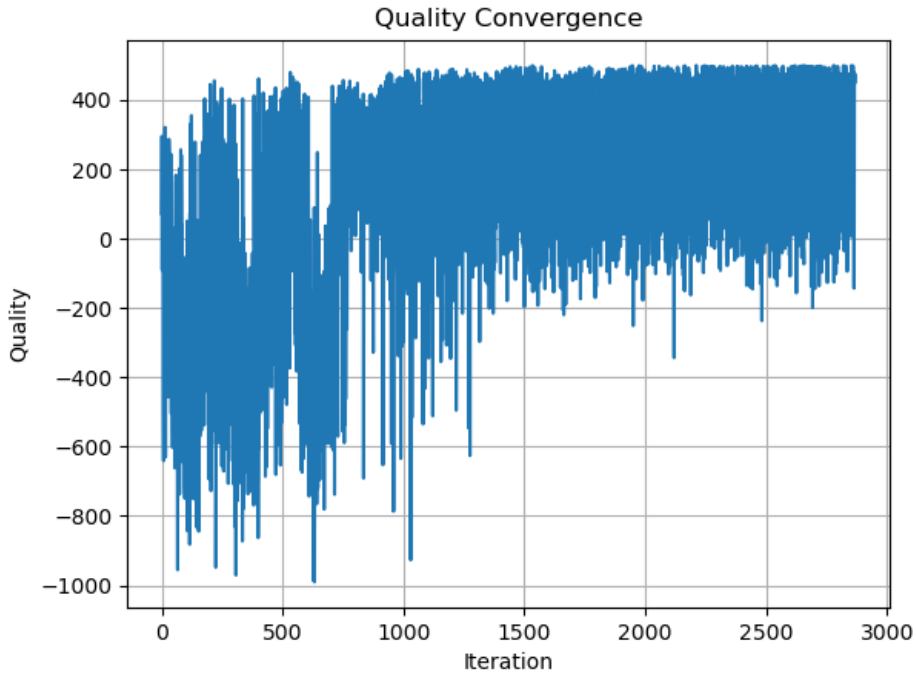

Figura 3: Quality Convergence

2.2.2 Melhor Trajetória Obtida Durante a Otimização

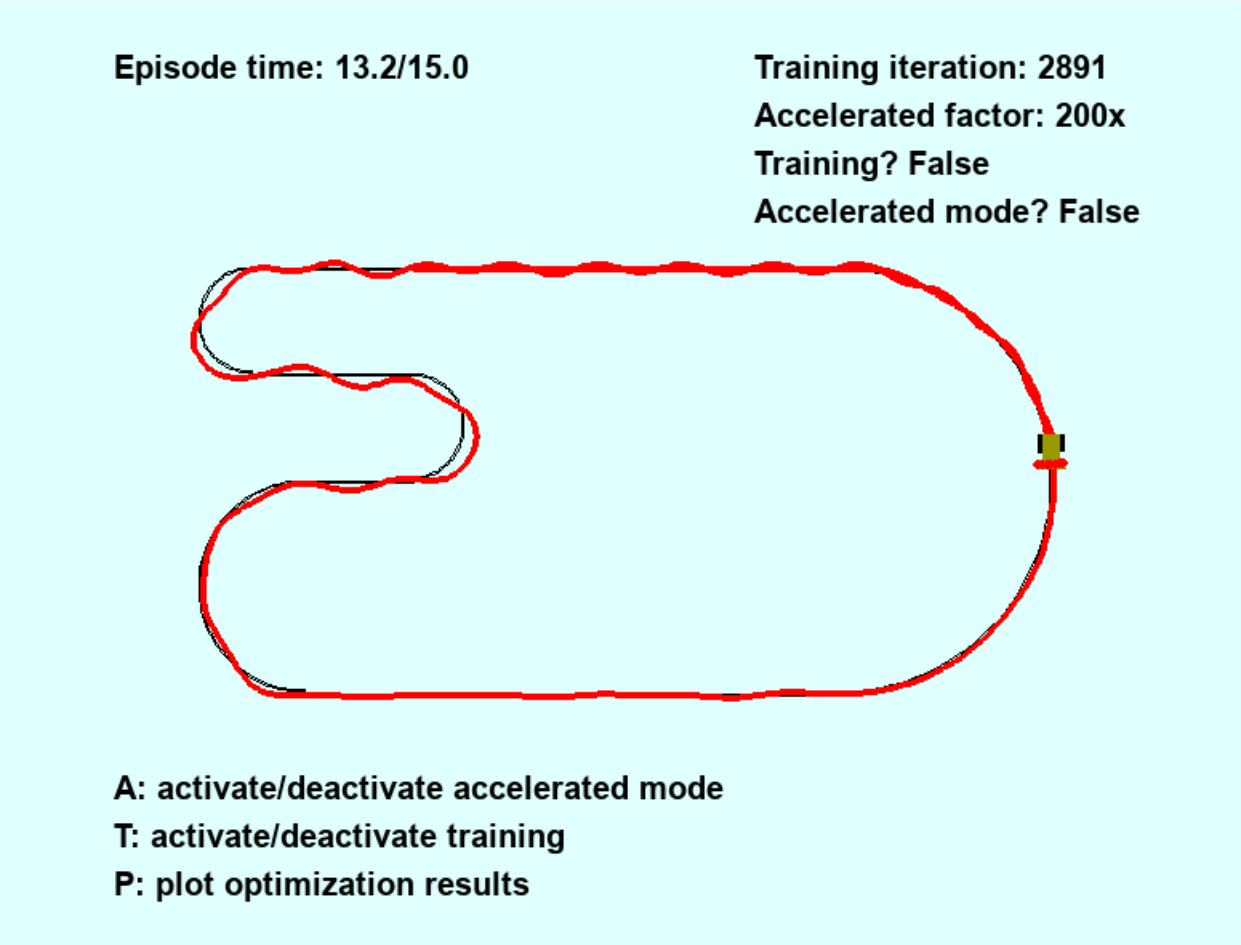

Figura 3: Best solutiuon

3 Discussão sobre o observado durante o processo de otimização

Percebeu-se que a otimização depende muito dos hiperparâmetros e da heurística. Caso o peso da reward fosse mudado, a otimização poderia ser muito lenta e ter um erro muito grade. Além disso, caso na função evaluate do simulation.py não tivesse a heurística de aumentar o erro caso não houvesse detection, foi percebido que para número superiores a 15000 iterações ainda não havia convergência, não conseguindo fazer curvas de maneira adequada, pois se o w fosse muito grande o robo começava a vibrar muito e entrar em ciclo e se o w fosse muito pequena, o robo saia totalmente da trajetória da curva. O w ótimo foi teve valor de 0.6.