Laboratório 4 – Métodos de Otimização Baseados em População

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 uma otimização dos parâmetros do controlador de um robô seguidor de linha usando *Particle Swarm Optimization* (PSO). Para tal feito, o robô deve manter velocidade linear constante e usar um controlador PID para a sua velocidade angular o que requer a otimização de quatro parâmetros, motivo pelo qual o PSO foi empregado.

Metodologia e descrição em alto nível do algoritmo utilizado:

Com a velocidade angular escrita da forma $\omega = K_p e + K_i \int ede + K_d \dot{e}$. Os parâmetros sujeitos a otimização são a velocidade linear e as os três ganhos do controlador, totalizando 4 parâmetros.

Como medida de qualidade, utilizou-se: $f(x) = \sum_{k=1}^{N} (v_k * dot(r_k, t_k) - w * |e_k|)$ em que N é a duração do treinamento, v_k é a velocidade linear (projetada no eixo local do robô) executada pelo robô no instante k, r_k é um vetor (bidimensional) unitário que aponta na direção do robô no instante k, t_k é o vetor tangente à atual posição no caminho no instante k, $|e_k|$ é o módulo do erro em relação à linha e w é um peso para fazer um compromisso entre se manter no centro da linha e seguir o caminho rapidamente.

Assim, para a implementação do P.S.O. foi criada uma classe *“Particle”*, que armazena as características de cada partícula do algoritmo P.S.O., são elas a sua posição, velocidade, valor atual com base na função de qualidade, posição da partícula que gerou o melhor valor dela até então, esse valor e uma *booleana* para determinar se o valor de uma partícula já foi atribuído numa dada geração.

Além disso, a implementação do *Particle Swarm Optimization* em si necessitou a implementação de subfunções na classe, foram elas:

1) *get_best_position()*, que retorna a melhor posição entre todas as partículas até então;
2) *get_best_value()*, que retorna o melhor valor até então encontrado;
3) *get_position_to_evaluate()*, que retorna a posição de uma partícula numa dada geração para calcular o seu valor;
4) *advance_generation()*, que avança a geração de partículas, alterando a posição de todas elas seguindo a lógica do P.S.O.;
5) *notifiy_evaluation(value)*, que informa o P.S.O. que uma certa partícula teve o seu valor calculado naquela geração.

Note que a função *advance_generation()* deveria ser empregada assim que não houvesse mais posições para calcular o valor numa dada geração, logo, ela deve ser chamada dentro da função *get_position_to_evaluate()* quando não houver mais partículas para calcular.

O algoritmo P.S.O. implementado foi inicialmente testado em um programa subjacente de otimização de uma função matemática simples para então ser aplicado ao robô em questão.

Resultados:

Ao final da implementação do código e 3000 gerações de treino para o robô, obteve-se os seguintes resultados:

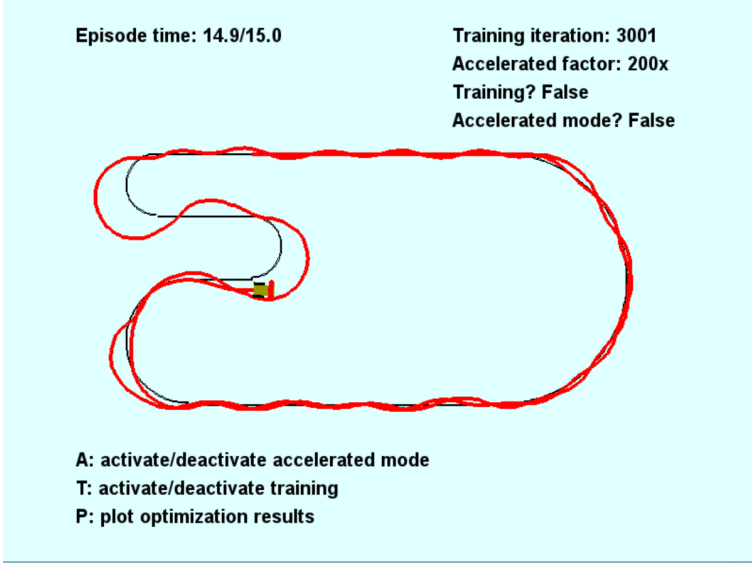

Figura 1: Caminho percorrido pelo robô após 3000 iterações de treino.

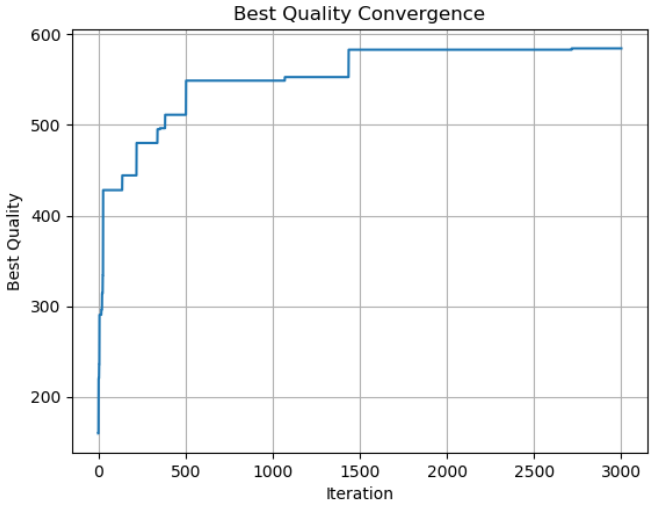

Figura 2: Gráfico da melhor qualidade obtida até então nas iterações

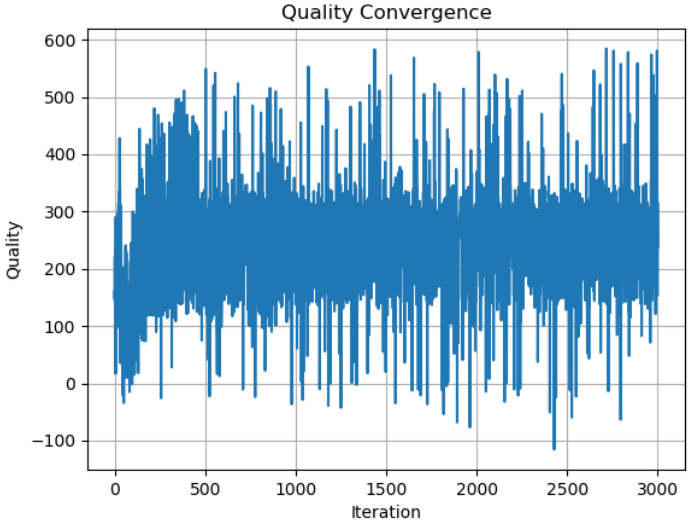

Figura 3: Gráfico da melhor qualidade em cada iteração

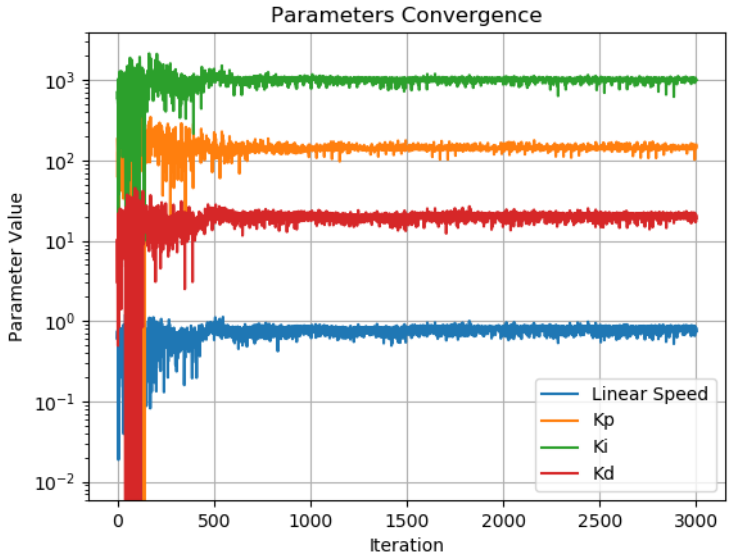

Figura 4: Gráfico do valor dos parâmetros otimizados em função da interação

Qualidade	***Velocidade Linear***	***Ganho Proporcional***	***Ganho integrativo***	***Ganho derivativo***
584.55866744782	0.857446	139.585621	1056.528629	21.847080

Tabela 1: Qualidade ótima obtida pelo P.S.O. e respectivos parâmetros otimizados

Assim, conclui-se que os *Particle Swarm Optimization* foi eficiente na execução do problema em questão, tendo em vista que o robô conseguiu completar quase duas voltas no circuito e teve uma trajetória similar ao formato do circuito. Os desvios poderiam ser menores utilizando uma diferente função de qualidade, pois o robô praticamente ignorou o ganho de qualidade nas curvas mais fechadas em prol de ir mais rápido para ficar ainda mais tempo nas retas.