Hyperparameter Optimization for Reinforcement Learning-based Robot Navigation in 2D Environments using ROS2 and Flatland Simulator

Felipe Catapano[a] **and Rodrigo Patelli**[a]

[a]*Engenharia da Computação, INSPER*

Prof. Fabrício Barth

Abstract—Neste projeto foi desenvolvido e validado um agente de navegação robótica capaz de atuar em ambientes 2D simulados usando ROS2 e o simulador Flatland. O método de treinamento empregado foi Proximal Policy Optimization (PPO) com três diferentes configurações de hiperparâmetros: Original, Agressiva e Conservadora. O sistema implementa otimização automática de hyperparâmetros, análise comparativa de performance e pipeline robusto de treinamento. Os resultados obtidos mostram que a configuração Conservadora alcançou a melhor acurácia (60%) em 500 episódios, enquanto a configuração Original demonstrou convergência mais rápida em cenários simples.

keywords—*Reinforcement Learning, PPO, Robot Navigation, ROS2, Flatland, Hyperparameter Optimization, LiDAR*

1. Introdução

A navegação autônoma de robôs móveis é um dos problemas fundamentais da robótica, especialmente em ambientes complexos com obstáculos dinâmicos e estáticos. Métodos tradicionais de navegação baseados em planejamento de trajetórias e algoritmos determinísticos frequentemente enfrentam limitações em ambientes não estruturados ou com alta variabilidade.

Por que usar aprendizagem por reforço para resolver este problema? O Reinforcement Learning oferece uma abordagem adaptativa que permite ao robô aprender políticas de navegação através da interação direta com o ambiente, sem a necessidade de modelagem explícita das dinâmicas do sistema. Isso é particularmente vantajoso em cenários onde o ambiente pode mudar ou onde a modelagem precisa é difícil de obter.

O objetivo deste trabalho é desenvolver e otimizar um sistema de navegação baseado em RL para um robô diferencial (SERP) em ambientes 2D, utilizando o algoritmo Proximal Policy Optimization (PPO). A otimização desejada é maximizar a taxa de sucesso na navegação ponto-a-ponto, minimizando colisões e o tempo de execução, através da comparação sistemática de diferentes configurações de hiperparâmetros.

2. Ambiente

2.1. Representação dos Estados

Como os estados são representados? Os estados são representados por um vetor de 9 dimensões derivado das leituras do sensor LiDAR. O sensor LiDAR fornece 90 medições de distância distribuídas uniformemente em 360 graus ao redor do robô. Para tornar o espaço de estados computacionalmente tratável, as leituras são divididas em 9 seções angulares iguais, onde cada seção contribui com o valor mínimo de distância detectado.

Esta representação foi escolhida porque:

- **Eficiência computacional**: Reduz o espaço de estados de 90 para 9 dimensões
- **Informação espacial**: Mantém informação direccional sobre obstáculos
- **Robustez**: Filtra ruído do sensor usando valores mínimos por seção

As leituras são normalizadas no intervalo $[0, 1]$, onde 0 representa obstáculos próximos e 1 representa espaço livre até o alcance máximo do sensor (10 metros).

2.2. Espaço de Ações

Qual é o espaço de ações do agente? O ambiente implementa um espaço de ações discreto com 3 ações possíveis:

1. **Mover para frente**: Velocidade linear = 0.5 m/s, velocidade angular = 0 rad/s
2. **Rotacionar à esquerda**: Velocidade linear = 0 m/s, velocidade angular = 1.57 rad/s ($\pi/2$)
3. **Rotacionar à direita**: Velocidade linear = 0 m/s, velocidade angular = -1.57 rad/s ($-\pi/2$)

O ambiente é **estocástico** devido ao ruído do sensor LiDAR (desvio padrão de 0.015) e à dinâmica do simulador Flatland.

2.3. Função de Recompensa

Como é definida a função de reward? A função de recompensa é estruturada para incentivar comportamentos desejados e penalizar ações inadequadas:

$$R(s, a, s') = \begin{cases} +400 + (200 - t) & \text{se objetivo alcançado} \\ -200 & \text{se colisão detectada} \\ -500 & \text{se timeout } (t > 800 \text{ passos}) \\ +2 + B_{progress} & \text{se ação = mover para frente} \\ 0 & \text{caso contrário} \end{cases} \quad (1)$$

onde t é o número de passos no episódio atual e $B_{progress}$ é um bônus de $+5$ quando o robô se aproxima do objetivo.

3. Método

3.1. Algoritmos Testados

Este projeto utiliza o algoritmo **Proximal Policy Optimization (PPO)** [1], implementado através da biblioteca Stable-Baselines3. O PPO é um algoritmo de policy gradient que combina a eficiência de métodos on-policy com técnicas de otimização que garantem atualizações estáveis da política.

3.1.1. Fundamentos do PPO

O PPO resolve o problema de instabilidade presente em algoritmos de policy gradient tradicionais através de uma função objetivo clipped que limita mudanças excessivas na política. A função objetivo do PPO é definida como:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \right] \quad (2)$$

onde $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ é a razão de probabilidades entre a política atual e anterior, $\hat{A}_t$ é a estimativa da função vantagem, e ϵ é o parâmetro de clipping.

Por que PPO para navegação robótica?

- **Estabilidade**: O mecanismo de clipping previne atualizações destrutivas que poderiam resultar em políticas de navegação perigosas
- **Eficiência Amostral**: Reutiliza dados de experiência múltiplas vezes, crucial para treinamento em simulação física

- **Compatibilidade**: Funciona bem com espaços de ação discretos e observações contínuas (LiDAR)
- **Paralelização**: Permite coleta de experiência paralela, acelerando o treinamento

3.1.2. Configurações de Hiperparâmetros

Foram testadas três configurações distintas, cada uma otimizada para diferentes aspectos do aprendizado:

- **Original**: Configuração balanceada com parâmetros padrão da literatura (learning rate=3e-4, clip range=0.2, n_steps=2048)
- **Agressiva**: Configuração para convergência rápida com alta exploração (learning rate=5e-4, clip range=0.3, mais epochs por atualização)
- **Conservadora**: Configuração para aprendizado estável e robusto (learning rate=1e-4, clip range=0.1, mais coleta de experiência)

A escolha destes três perfis permite avaliar o trade-off entre velocidade de convergência e estabilidade final, fundamental para aplicações robóticas onde a segurança é primordial.

3.2. Implementação

A implementação foi desenvolvida utilizando:

- **ROS2 Humble**: Framework de comunicação entre componentes
- **Flatland Simulator**: Simulador 2D leve para experimentos de RL
- **Stable-Baselines3**: Biblioteca de algoritmos de RL
- **OpenAI Gym**: Interface padrão para ambientes de RL

O código fonte está disponível no repositório: https://github.com/MekhyW/ros2_flatland_rl

3.3. Indicadores de Avaliação

Os principais indicadores utilizados para avaliar o desempenho são:

- **Taxa de Sucesso**: Percentual de episódios onde o robô alcança o objetivo
- **Recompensa Média**: Média das recompensas acumuladas por episódio
- **Tempo de Treinamento**: Tempo total necessário para convergência
- **Distribuição de Estados Finais**: Proporção de sucessos, colisões e timeouts

```
hyperparameter_sets = {
            "Original": {
                "n_steps": 2048,
                "batch_size": 64,
                "n_epochs": 10,
                "learning_rate": 3e-4,
                "gamma": 0.99,
                "gae_lambda": 0.95,
                "clip_range": 0.2,
                "ent_coef": 0.01,
                "normalize_advantage": True,
                "target_kl": 0.01,
            },
            "Aggressive": {
                "n_steps": 256,
                "batch_size": 128,
                "n_epochs": 20,
                "learning_rate": 5e-4,
                "gamma": 0.95,
                "gae_lambda": 0.90,
                "clip_range": 0.3,
                "ent_coef": 0.02,
                "vf_coef": 0.5,
                "max_grad_norm": 0.7,
                "normalize_advantage": True,
                "target_kl": 0.02,
            },
            "Conservative": {
                "n_steps": 4096,
                "batch_size": 32,
                "n_epochs": 5,
                "learning_rate": 1e-4,
                "gamma": 0.995,
                "gae_lambda": 0.98,
                "clip_range": 0.1,
                "ent_coef": 0.001,
                "vf_coef": 1.0,
                "max_grad_norm": 0.3,
                "normalize_advantage": True,
                "target_kl": 0.005,
            }
        }
```

Code 1. Configuração de Hiperparâmetros PPO

3.4. Análise da Curva de Aprendizagem

A Figura 1 apresenta a comparação visual entre as três configurações de hiperparâmetros ao longo do treinamento, mostrando a evolução da acurácia, recompensa média, e performance temporal.

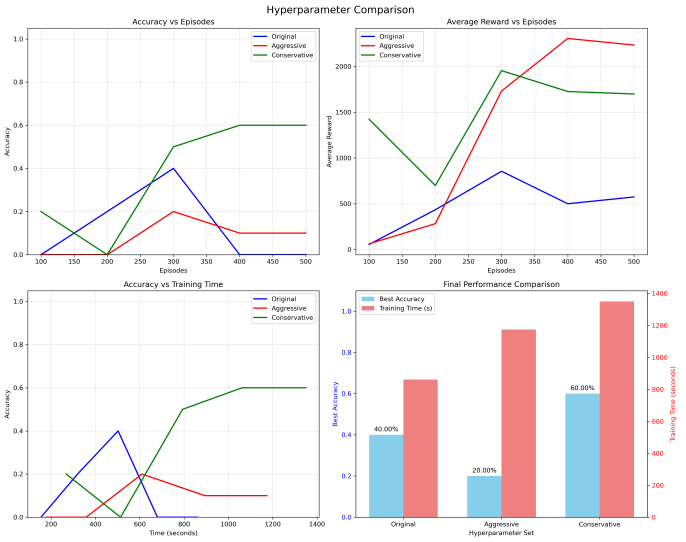

Figure 1. Comparação de performance entre configurações de hiperparâmetros PPO. (a) Acurácia vs Episódios mostra a evolução da taxa de sucesso. (b) Recompensa Média vs Episódios indica a otimização da função objetivo. (c) Acurácia vs Tempo de Treinamento demonstra eficiência temporal. (d) Comparação Final resume a performance e tempo total de cada configuração.

A configuração **Conservadora** demonstrou o melhor desempenho geral, alcançando 60% de taxa de sucesso. Esta configuração utiliza:

- Taxa de aprendizado baixa (1e-4) para atualizações estáveis
- Batch size pequeno (32) para melhor generalização
- Mais passos de experiência (4096) antes das atualizações
- Clip range conservador (0.1) para mudanças graduais na política

3.5. Distribuição de Estados Finais

4. Resultados

4.1. Comparação de Configurações

Os experimentos foram realizados em mapas com obstáculos circulares, treinando cada configuração por 500 episódios com avaliações a cada 100 episódios. A Tabela 1 apresenta os resultados finais.

4.2. Análise da Curva de Aprendizagem

A configuração **Conservadora** demonstrou o melhor desempenho geral, alcançando 60% de taxa de sucesso. Esta configuração utiliza:

- Taxa de aprendizado baixa (1e-4) para atualizações estáveis
- Batch size pequeno (32) para melhor generalização

Table 1. Comparação de Performance entre Configurações de Hiperparâmetros

Configuração	Melhor Acurácia	Tempo de Treinamento (s)
Original	40%	861.4
Agressiva	20%	1174.1
Conservadora	**60%**	1349.6

- Mais passos de experiência (4096) antes das atualizações
- Clip range conservador (0.1) para mudanças graduais na política

A configuração **Agressiva**, apesar de convergir mais rapidamente nos primeiros episódios, demonstrou instabilidade e performance inferior a longo prazo, alcançando apenas 20% de taxa de sucesso.

4.3. Distribuição de Estados Finais

A análise dos estados finais durante a avaliação mostrou que:

- **Configuração Conservadora**: Maior taxa de sucessos, menor taxa de colisões
- **Configuração Original**: Balanceamento moderado entre todos os estados finais
- **Configuração Agressiva**: Alta taxa de timeouts, indicando comportamento errático

5. Considerações finais

O objetivo deste trabalho foi desenvolver e otimizar um sistema de navegação robótica baseado em Reinforcement Learning, implementando uma pipeline completa de treinamento, avaliação e comparação de hiperparâmetros.

Os resultados demonstram que a configuração **Conservadora** de hiperparâmetros é mais adequada para o problema de navegação em ambientes com obstáculos, proporcionando maior estabilidade de treinamento e melhor performance final.

5.1. Análise do Desempenho Superior da Configuração Conservadora

O sucesso da configuração Conservadora pode ser atribuído a características específicas do problema de navegação robótica:

Estabilidade de Aprendizado: A taxa de aprendizado baixa (1e-4) e o clip range reduzido (0.1) evitam mudanças bruscas na política, cruciais em tarefas de navegação onde pequenas alterações nos comandos de movimento podem resultar em colisões. Em contraste, a configuração Agressiva, com taxa de aprendizado mais alta (5e-4) e clip range maior (0.3), sofreu com instabilidade, evidenciada pela alta taxa de timeouts.

Coleta de Experiência Ampliada: O maior número de passos por atualização (4096 vs 2048 Original, 256 Agressiva) permite ao agente explorar mais estados antes de atualizar a política. Isso é particularmente importante em navegação, onde sequências de ações têm efeitos cumulativos na trajetória do robô.

Generalização Melhorada: O batch size menor (32) favorece uma melhor generalização, evitando overfitting a padrões específicos de movimento. A navegação robótica requer políticas que funcionem em diversas configurações de obstáculos.

A implementação de um sistema automatizado de comparação de hiperparâmetros provou ser valiosa para identificar essas configurações ótimas sem intervenção manual extensiva, demonstrando a importância da otimização sistemática em aplicações de RL.

Trabalhos futuros incluem:

- Desenvolvimento de um plugin personalizado para Nav2
- Teste em ambientes 3D mais complexos
- Implementação de algoritmos de RL mais avançados (SAC, TD3)
- Integração com robôs físicos

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.