Simulação de Sistemas RT

Matheus Carvalho Reges

***Abstract*—The simulation of mobile robot systems is essential for validating and optimizing control models before practical application. This study investigates the behavior of a differential drive mobile robot in a real-time system (RT) using Linux RT OS version 5.15.170-rt81 and compares it with a non-RT environment (WSL on Windows). The analysis includes robot simulation and control under loaded and unloaded scenarios, evaluating parameters such as jitter, execution times, and trajectory-following precision.**

The proposed system employs a distributed multithreaded controller to efficiently manage control, simulation, and monitoring tasks. The results highlight the superiority of the RT system in maintaining consistent response times and low jitter levels, even under load conditions. Conversely, the non-RT environment exhibited increased variability, especially under load, affecting control precision.

Quantitative comparisons of metrics such as mean, variance, standard deviation, and extreme values of $T(k)$ (response time) and $J(k)$ (jitter) were conducted. Additionally, trajectory precision was evaluated by measuring the difference between the reference and the robot's actual position. The study underscores the advantages of RT systems for critical control applications and provides insights for future advancements in distributed control systems for mobile robots.

I. Introdução

A simulação de sistemas de robôs móveis é uma prática essencial para validar e otimizar modelos de controle antes de sua aplicação prática. Este trabalho investiga o comportamento de um robô móvel com acionamento diferencial em um sistema de tempo real (RT) utilizando o Linux RT OS versão 5.15.170-rt81, comparando-o com um ambiente sem suporte a RT (WSL no Windows). A análise abrange a simulação e controle do robô, considerando cenários com e sem carga , avaliando parâmetros como jitter, tempos de execução e precisão no seguimento de trajetórias. O desempenho é apresentado por meio de gráficos gerados no GNU Octave, utilizando os arquivos de saída produzidos pelo programa.

II. Aspectos teóricos dos sistemas RT

Sistemas de tempo real (RT) são sistemas projetados para garantir que as tarefas sejam executadas dentro de um tempo específico, conhecido como "prazo". A principal característica dos sistemas RT é que a execução de tarefas dentro de um tempo predeterminado é essencial para o funcionamento correto do sistema. Esses sistemas são fundamentais em aplicações críticas, como controle de robôs, sistemas embarcados, automação industrial, e outros.

A. Classificação dos Sistemas RT

Sistemas de tempo real podem ser classificados em três categorias principais:

- **Sistemas Hard Real-Time (HRT):** Nestes sistemas, o cumprimento do prazo é estritamente obrigatório. Caso uma tarefa não seja completada dentro do prazo, o sistema falha. Exemplos típicos de sistemas HRT incluem sistemas de controle de aeronaves, dispositivos médicos (como marcapassos) e sistemas de segurança automotivos.
- **Sistemas Soft Real-Time (SRT):** Nestes sistemas, o cumprimento do prazo é desejável, mas o sistema pode tolerar falhas ocasionais sem um impacto significativo no desempenho. Sistemas de multimídia e de streaming de vídeo são exemplos típicos de sistemas SRT.
- **Sistemas Firm Real-Time (FRT):** Esses sistemas apresentam características de um sistema HRT, mas com menos rigidez. A falha em cumprir um prazo não leva à falha total do sistema, mas pode causar uma degradação significativa no desempenho.

B. Características de Sistemas RT

As principais características de sistemas RT incluem:

- **Previsibilidade:** Um sistema RT deve ser previsível, ou seja, as tarefas devem ser executadas de forma a garantir que os prazos sejam cumpridos sem falhas.
- **Escalonamento:** O escalonamento de tarefas é essencial em sistemas RT. Algoritmos como Rate Monotonic Scheduling (RMS) e Earliest Deadline First (EDF) são comumente utilizados para gerenciar a execução das tarefas de forma eficiente.
- **Jitter:** Jitter é a variação no tempo de resposta em relação ao tempo esperado. Em sistemas RT, o jitter deve ser minimizado para garantir um comportamento determinístico e confiável do sistema.

C. Requisitos de Sistemas RT

Os sistemas RT possuem requisitos que garantem sua operação eficiente e dentro dos parâmetros exigidos:

- **Determinismo:** O comportamento de um sistema RT deve ser determinístico, ou seja, a execução das tarefas deve ocorrer de forma consistente e previsível, sem desvios.
- **Tempos de resposta rigorosos:** Cada tarefa tem um prazo específico para ser completada. O sistema deve ser capaz de garantir que essas tarefas sejam concluídas dentro do tempo estipulado.
- **Sincronização e comunicação:** A comunicação eficiente entre múltiplas tarefas e a sincronização dessas tarefas é fundamental para o sucesso de um sistema RT. A concorrência e a troca de dados entre as tarefas devem ser bem gerenciadas para evitar condições de corrida e

garantir que as tarefas críticas sejam atendidas no tempo adequado.

D. Ferramentas e Tecnologias para Sistemas RT

Vários sistemas operacionais e ferramentas são utilizadas no desenvolvimento e execução de sistemas RT:

- **RTOS (Sistemas Operacionais de Tempo Real):** Sistemas como o `Linux RT`, `VxWorks` e `FreeRTOS` são projetados para oferecer suporte eficiente para tarefas em tempo real. Eles fornecem uma camada adicional de controle sobre a execução das tarefas, permitindo a priorização e o gerenciamento eficiente dos recursos do sistema.
- **Características do Linux RT:** O `Linux RT`, em sua versão com patches de tempo real, oferece um ambiente adequado para a execução de tarefas em tempo real em hardware convencional. A versão utilizada neste trabalho foi o `Linux RT OS 5.15.170-rt81`. As principais melhorias oferecidas pelo `Linux RT` incluem baixa latência e priorização de tarefas, fundamentais para garantir o desempenho em tempo real.
- **Uso de Multithreading:** O uso de múltiplas threads em sistemas RT permite a distribuição de tarefas de forma paralela, maximizando a utilização de recursos e garantindo que os prazos de execução das tarefas sejam cumpridos. Em sistemas distribuídos, a comunicação entre as threads deve ser bem controlada para evitar problemas de sincronização e concorrência.

E. Desafios de Sistemas RT

Embora os sistemas RT sejam altamente eficazes, eles apresentam desafios significativos, tais como:

- **Gerenciamento de recursos:** Em sistemas com múltiplas tarefas e recursos limitados (como memória e capacidade de processamento), o gerenciamento eficiente desses recursos é essencial para garantir que as tarefas sejam concluídas dentro dos prazos exigidos.
- **Interrupções e concorrência:** O tratamento eficiente de interrupções e a gestão de tarefas concorrentes são fundamentais para a operação de sistemas RT, evitando condições de corrida e garantindo a integridade dos dados.
- **Desempenho em sistemas distribuídos:** Quando o sistema envolve múltiplos processadores ou dispositivos, a comunicação entre esses componentes deve ser eficiente para garantir que as tarefas sejam executadas dentro dos prazos exigidos. A latência de comunicação e a coordenação entre as diferentes partes do sistema precisam ser minimizadas.

III. Problema proposto (simulação de um sistema de controle distribuído multithread)

O objetivo deste trabalho é simular o comportamento de um robô móvel com acionamento diferencial em um ambiente de tempo real. O sistema proposto envolve a implementação de um controlador distribuído utilizando múltiplas threads para dividir as funcionalidades de controle, simulação e monitoramento.

A simulação busca avaliar o comportamento do robô em termos de controle de movimento, precisão na execução de trajetórias e desempenho com e sem carga, para observar a influência no *jitter* e no tempo de resposta. Para alcançar esses objetivos, são analisados os seguintes cenários:

1) Sistema sem carga, em um SO não RT;
2) Sistema com carga, em um SO não RT;
3) Sistema sem carga, em um SO RT;
4) Sistema com carga, em um SO RT.

Esses cenários permitem investigar como diferentes condições influenciam o desempenho do sistema, considerando métricas como precisão do controle, impacto da carga, variações no *jitter* e tempos de resposta. Além disso, serão realizadas comparações quantitativas dos valores de **média**, **variância**, **desvio padrão**, e os **valores máximos e mínimos** de $T(k)$ e $J(k)$, onde $T(k)$ representa o tempo de resposta e $J(k)$ o *jitter*.

Também será avaliada a diferença entre a referência e a posição real à frente do robô, permitindo medir a precisão do controle em diferentes condições. A abordagem multithread é essencial para garantir que tarefas críticas sejam realizadas de forma eficiente, mesmo em ambientes de maior complexidade.

IV. Sistema Desenvolvido

O sistema desenvolvido foi estruturado para realizar a simulação e controle de um robô com cálculo e implementação dos sinais de controle e dinâmica do movimento. A seguir, descrevem-se os principais componentes:

A. Arquitetura Geral

O sistema foi desenvolvido em linguagem **C**, organizada em múltiplos arquivos modulares para facilitar o desenvolvimento e a manutenção. Os componentes principais incluem:

1) **Cálculos de Controle e Simulação**: Implementados nos arquivos `estate.c`, `simulation.c` e `simulation_model.c`.
2) **Manipulação de Matrizes**: Uso de transformações matriciais para controle do robô.
3) **Multithreading**: Coordenação das simulações e do armazenamento de dados usando threads para paralelismo.
4) **Armazenamento e Registro de Dados**: Dados de tempo, posição e controle são registrados em arquivos para análise.

B. Cálculo com Carga e Sem Carga

O sistema foi projetado para funcionar em dois modos distintos:

1) **Sem Carga**: Simulação do movimento do robô sem considerar efeitos adicionais de peso ou restrições externas.
2) **Com Carga**: Simulação incluindo dinâmicas mais complexas relacionadas à carga adicional no robô.

Ambos os modos compartilham o mesmo núcleo de cálculos, com diferenças nos parâmetros que impactam diretamente as funções de controle e movimentação.

C. Código Fonte

Os arquivos principais do sistema incluem:

1) `estate.c`: Este arquivo implementa a função `computeRobotEstate`, que calcula o movimento do robô, atualizando sua posição (`posX`, `posY`) e orientação (`orientacao`) com base na velocidade angular e no intervalo de tempo (`deltaTime`).

Transformações matriciais foram implementadas por meio das funções `criaMatrizTransformacao` e `criaMatrizTransformacaoInv`, essenciais para cálculos de controle e simulação.

2) `main.c`: Responsável por inicializar o sistema, configurar os parâmetros, criar e gerenciar as threads principais e auxiliares, além de finalizar adequadamente os recursos.

3) `simulation.c`: Contém a lógica da simulação do robô e coleta de dados. O controle do jitter e da temporização é feito com funções específicas que garantem a execução precisa das operações.

4) `simulation_model.c`: Implementa o modelo de entrada do sistema e as referências que guiam os cálculos de controle. Contém também a lógica para simulações separadas em coordenadas `X` e `Y`, além da função de cálculo do sinal de controle.

V. Simulações efetuadas e resultados obtidos

As simulações foram realizadas com o objetivo de comparar o desempenho do sistema em dois cenários: um com suporte a sistemas RT, utilizando o Linux RT OS, e outro sem suporte a RT, utilizando o WSL no Windows. O foco da comparação foi o *jitter*, tempos de execução e precisão no seguimento de trajetórias. Foram realizadas simulações em diferentes condições de carga (com e sem threads adicionais) para observar o impacto de múltiplas threads no desempenho do sistema. Os resultados mostraram que a execução em tempo real ofereceu vantagens significativas em termos de precisão no controle e redução do *jitter*, especialmente quando o sistema estava sobre carga de trabalho.

Os resultados foram obtidos com α_1 e α_2 iguais a 3.000.

A. Resultados por Cenário

1) **Sistema com carga, em um SO não RT:**

Gráfico de Trajetória: O robô segue uma trajetória definida com consistência relativamente alta.
Métricas Quantitativas:

- $T(k)$:
  - Média: 0,010341 s
  - Variância e Desvio Padrão: Valores muito baixos indicam tempos de resposta consistentes.
  - Valores Máximos e Mínimos: Baixa diferença $(0,010561 - 0,010051)$, indicando estabilidade no tempo de resposta.
- $J(k)$:
  - Média: 0,000341 s

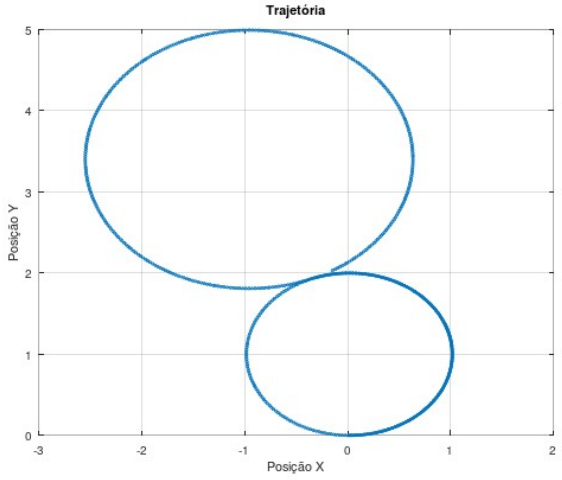

Fig. 1. Gráfico de $y_c(t)$ amostrado para um horizonte de simulação de 20s.

Tabela Comparativa: Média, Variância, Desvio Padrão, Máximo, Mínimo

Métrica	T(k)	Jitter(k)
Média	0.010341	0.000341
Variância	0.000001	0.000001
Desvio Padrão	0.000919	0.000919
Máximo	0.051058	0.041058
Mínimo	0.010051	0.000051

Fig. 2. Tabela com métricas.

  - Baixa variabilidade e desvio padrão, com um intervalo pequeno entre máximo e mínimo.

2) **Sistema sem carga, em um SO não RT:**

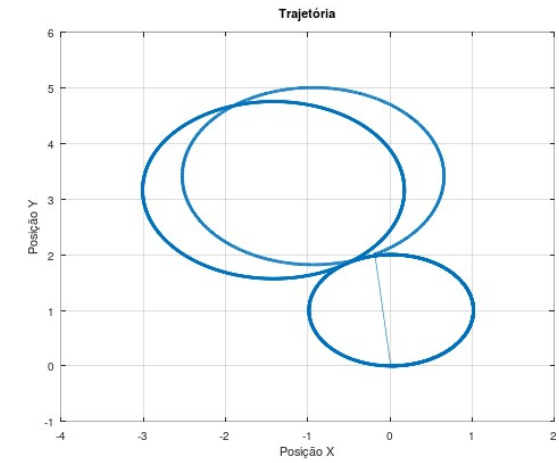

Fig. 3. Gráfico de $y_c(t)$ amostrado para um horizonte de simulação de 20s.

Gráfico de Trajetória: O robô segue uma trajetória definida com consistência relativamente alta, mas há desvios em pontos de curva mais acentuados.
Métricas Quantitativas:

- $T(k)$:

Tabela Comparativa: Média, Variância, Desvio Padrão, Máximo, Mínimo

Métrica	T(k)	Jitter(k)
Média	0.010341	0.000341
Variância	0.000001	0.000001
Desvio Padrão	0.000919	0.000919
Máximo	0.051058	0.041058
Mínimo	0.010051	0.000051

Fig. 4. Tabela com métricas.

  - Média: 0,010341 s
  - Variância e Desvio Padrão: Valores muito baixos indicam tempos de resposta consistentes.
  - Valores Máximos e Mínimos: Baixa diferença $(0,010561 - 0,010051)$, indicando estabilidade no tempo de resposta.
- $J(k)$:
  - Média: 0,000341 s
  - Baixa variabilidade e desvio padrão, com um intervalo pequeno entre máximo e mínimo.

3) **Sistema com carga, em um SO RT:**

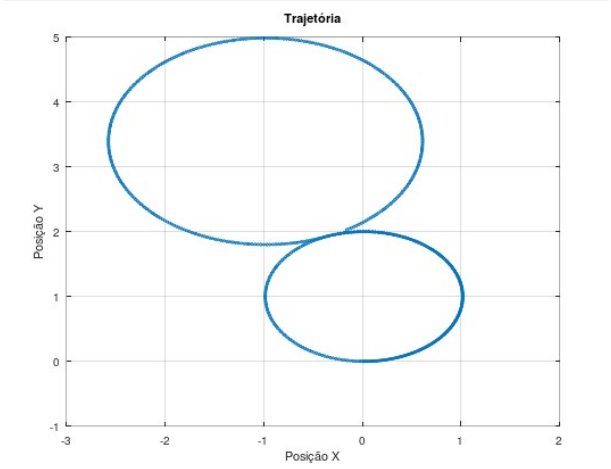

Fig. 5. Gráfico de $y_c(t)$ amostrado para um horizonte de simulação de 20s.

Gráfico de Trajetória: O robô segue uma trajetória suave. A curva parece bem definida, sugerindo que o controle do sistema está estável e com baixa interferência, mas com algumas curvas acentuadas.

Tabela Comparativa: Média, Variância, Desvio Padrão, Máximo, Mínimo

Métrica	T(k)	Jitter(k)
Média	0.010290	0.000290
Variância	0.000000	0.000000
Desvio Padrão	0.000115	0.000115
Máximo	0.010486	0.000486
Mínimo	0.010074	0.000074

Fig. 6. Tabela com métricas.

Métricas Quantitativas:

- $T(k)$:
  - Média: 0,010290 s
  - Variância e Desvio Padrão: Ainda baixos, mas aumentaram em comparação ao caso sem carga.
  - Máximo e mínimo $(0,010486$ e $0,010074)$: Mostram maior dispersão.
- $J(k)$:
  - Média: 0,000290 s
  - Variância e desvio padrão: Mostram uma oscilação mais elevada.
  - Máximo e mínimo $(0,000486$ e $0,000074)$: Apresentam maior dispersão.

4) **Sistema sem carga, em um SO RT:**

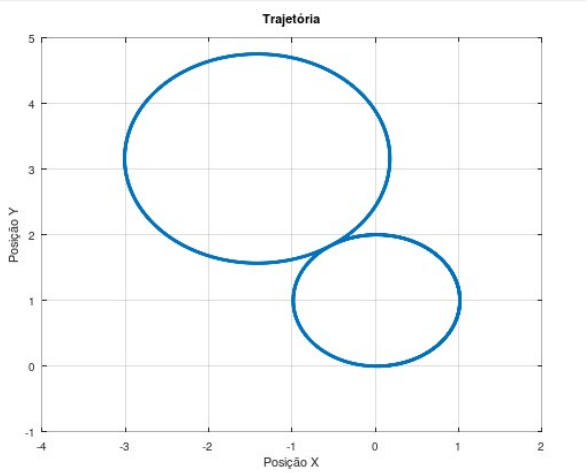

Fig. 7. Gráfico de $y_c(t)$ amostrado para um horizonte de simulação de 20s.

Gráfico de Trajetória: O robô segue uma trajetória suave. A curva parece bem definida, sugerindo que o controle do sistema está estável e com baixa interferência.

Tabela Comparativa: Média, Variância, Desvio Padrão, Máximo, Mínimo

Métrica	T(k)	Jitter(k)
Média	0.010373	0.000373
Variância	0.000000	0.000000
Desvio Padrão	0.000093	0.000093
Máximo	0.010545	0.000545
Mínimo	0.010078	0.000078

Fig. 8. Tabela com métricas.

Métricas Quantitativas:

- $T(k)$:
  - Média: 0,010373 s
  - Variância e desvio padrão: Praticamente nulos, indicando consistência no tempo de execução das tarefas.
  - Máximo e mínimo: Muito próximos da média, reforçando a estabilidade do sistema.
- $J(k)$:
  - Média: 0,000373 s

- Variância e desvio padrão: Praticamente nulos, evidenciando jitter desprezível.
- Máximo e mínimo: Mostram que as flutuações no jitter são muito pequenas.

VI. Discussão dos Resultados

A. Sistema Sem Carga em um SO Não-RT

No cenário sem carga, em um sistema operacional não em tempo real (SO Não-RT), os resultados indicaram um desempenho satisfatório para o robô. O gráfico de trajetória mostrou consistência, com desvios limitados, especialmente em curvas mais acentuadas. As métricas quantitativas destacaram a estabilidade do sistema, com tempos de resposta e jitter apresentando valores muito baixos de variância e desvio padrão. Isso sugere que, sem carga, o SO Não-RT consegue oferecer uma precisão elevada no controle.

As diferenças entre os valores máximos e mínimos de $T(k)$ foram reduzidas (0,010561 s e 0,010051 s), confirmando a previsibilidade no tempo de execução. Da mesma forma, o jitter apresentou uma média baixa (0,000341 s) e valores máximos e mínimos próximos, reforçando a consistência. A ausência de carga contribuiu para minimizar os desafios impostos ao controle do robô.

B. Sistema Com Carga em um SO Não-RT

Ao introduzir uma carga, o SO Não-RT exibiu limitações no controle. O gráfico de trajetória revelou desvios mais significativos, especialmente em curvas, indicando que o sistema perdeu precisão ao lidar com condições mais adversas.

Embora a média de $T(k)$ tenha permanecido igual à do cenário sem carga (0,010341 s), a variância e o desvio padrão aumentaram ligeiramente, evidenciando maior variabilidade no tempo de resposta. Similarmente, o jitter apresentou maior variabilidade e dispersão nos valores máximos e mínimos. Isso reflete a sensibilidade do SO Não-RT à carga adicional, expondo suas limitações para lidar com tarefas de maior complexidade de forma consistente.

C. Comparativo Geral em um SO Não-RT

O impacto da carga no SO Não-RT foi mais notório na precisão do controle e na variabilidade de $T(k)$. Apesar de a média dos tempos de resposta e jitter não terem mudado significativamente, a variância e o desvio padrão aumentaram. Esses resultados sugerem que, embora o sistema consiga manter desempenho razoável sem carga, sua robustez diminui consideravelmente com a introdução de fatores externos.

D. Sistema Sem Carga em um SO RT

Sob um SO em tempo real (SO RT), o robô manteve uma trajetória consistente e bem definida, como mostrado pelos gráficos. As métricas quantitativas evidenciaram um desempenho altamente previsível, com variâncias e desvios padrões praticamente nulos para $T(k)$ e $J(k)$. A estabilidade nos valores máximos e mínimos também destacou o controle eficiente do sistema em um cenário sem carga.

E. Sistema Com Carga em um SO RT

Mesmo com a introdução de carga, o SO RT conseguiu manter a trajetória do robô dentro de um padrão visualmente consistente. Pequenas variações numéricas foram observadas nas métricas, como um aumento moderado na variância e no desvio padrão de $T(k)$ e $J(k)$. Contudo, esses valores permaneceram baixos em termos absolutos, sugerindo que o SO RT lidou bem com a complexidade adicional imposta pela carga.

F. Comparativo Geral em um SO RT

O SO RT apresentou desempenho superior em ambos os cenários (com e sem carga). Embora a carga tenha introduzido pequenas variações, as métricas continuaram a indicar altos níveis de estabilidade e previsibilidade. A capacidade do SO RT de manter a precisão e o controle, mesmo sob condições adversas, reflete sua adequação para aplicações que requerem alta confiabilidade temporal.

G. Impacto Geral da Carga e Considerações Finais

A comparação entre os dois cenários revelou diferenças marcantes no desempenho do sistema. O impacto da carga foi mais evidente no SO Não-RT, destacando suas limitações em cenários de maior complexidade. Por outro lado, o SO RT demonstrou maior robustez, garantindo que o sistema mantivesse comportamento previsível e eficiente, mesmo com carga.

Esses resultados reforçam a importância de selecionar o SO adequado conforme os requisitos do sistema. Para aplicações que demandam alta precisão e previsibilidade, o uso de um SO RT é claramente vantajoso, enquanto um SO Não-RT pode ser suficiente para cenários de menor complexidade e sem carga significativa.

VII. Conclusão

Este trabalho apresentou uma análise detalhada do comportamento de um robô móvel com acionamento diferencial em ambientes com e sem suporte a sistemas de tempo real (RT). A abordagem multithread demonstrou ser eficaz na simulação e controle do robô, dividindo de maneira eficiente as responsabilidades de controle, simulação e monitoramento.

Os resultados mostraram que o Linux RT OS foi capaz de manter tempos de resposta mais consistentes e baixos níveis de jitter, mesmo em condições de carga. Por outro lado, o ambiente sem suporte a RT (WSL no Windows) apresentou maior variabilidade no desempenho, especialmente sob carga, evidenciando suas limitações para aplicações de controle que exigem alta precisão e previsibilidade.

A precisão do controle foi avaliada por meio da diferença entre a referência e a posição real do robô, com o sistema em tempo real exibindo melhor aderência às trajetórias planejadas. Além disso, as métricas quantitativas destacaram a robustez do sistema RT ao lidar com cenários complexos, enquanto o SO não RT mostrou maior sensibilidade às variações externas.

Em suma, a análise comparativa reforça a importância de utilizar sistemas RT em aplicações críticas, onde o controle

preciso e a estabilidade temporal são fundamentais. Este estudo fornece uma base sólida para a implementação prática de sistemas de controle distribuído em robôs móveis e abre caminhos para futuras melhorias no desempenho e escalabilidade.

VIII. Referências

References

[1] Stankovic, J. A. *Real-Time Systems: Theory and Practice.* Springer, 2018.

[2] Buttazzo, G. C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Springer, 2011.

[3] The Linux Foundation. *Documentation on PREEMPT-RT Patch.* Disponível em: https://wiki.linuxfoundation.org/realtime/.

[4] Yaghmour, K., Masters, J., Ben-Yossef, G., & Gerum, P. *Building Embedded Linux Systems.* O'Reilly Media, 2020.

[5] Audsley, N. C., Burns, A., Richardson, M. F., & Tindell, K. W. *Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling.* Software Engineering Journal, 1993.

[6] Liu, C. L., & Layland, J. W. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.* Journal of the ACM, 1973.

[7] Craig, J. J. *Introduction to Robotics: Mechanics and Control.* Pearson, 2005.

[8] Siciliano, B., & Khatib, O. *Springer Handbook of Robotics.* Springer, 2016.

[9] Eaton, J. W., Bateman, D., Hauberg, S., & Wehbring, R. *GNU Octave Manual.* Network Theory Ltd, 2016.

[10] Tanenbaum, A. S., & Van Steen, M. *Distributed Systems: Principles and Paradigms.* Pearson, 2007.

[11] Herlihy, M., & Shavit, N. *The Art of Multiprocessor Programming.* Elsevier, 2012.