UFAM

Universidade do Amazonas
Faculdade de Tecnologia
Departamento de Eletrônica e Computação
Engenharia da Computação
Programação de Tempo Real
Prof. André Cavalcante

Trabalho 3: Sistema de Controle Distribuído Não RT

Aluno: MATHEUS ROCHA CANTO

INTRODUÇÃO

Este laboratório aborda a implementação e análise de um sistema de simulação para um robô móvel, empregando programação em C com a biblioteca POSIX Threads (PThreads) sendo o objetivo central decompor o sistema em tarefas paralelas e sincronizadas, cada uma com uma periodicidade distinta, para controlar o robô em uma trajetória de referência.

A comunicação segura entre as threads foi garantida através do uso de "monitores", implementados com mutexes para proteger o acesso a dados compartilhados.

Além da implementação, o trabalho inclui uma análise de desempenho temporal do sistema sob diferentes condições de carga de CPU (Com Carga e Sem Carga). Para a visualização e análise dos dados, foi utilizado o software GNU Octave, com um script dedicado para a criação de gráficos comparativos que permitiram avaliar o desempenho do controlador.

ESTRUTURA DA PASTA UTILIZADA

A estrutura de diretórios do projeto foi organizada para promover a modularidade e a reutilização do código, conforme a seguir:

- **`Trabalho 3/`** (Pasta Raiz):
  - **data/:** Armazena os dados **.txt** gerados e os gráficos **.png** do Octave.
  - **`include/`:** Contém os arquivos de cabeçalho das interfaces (**`.h`**).
  - **scripts/:** Script **.m** do Octave para análise completa dos gráficos
  - **`src/`:** Contém os arquivos de código-fonte (**`.c`**).
  - **`Makefile`:** Arquivo de automação da compilação.
  - **`main`:** Executável final, gerado na raiz do projeto.

A organização foi escolhida para facilitar a reutilização e manutenção do código, separando claramente a interface (**`include`**), a implementação (**`src`**), os dados gerados (**`data`**) e as ferramentas de análise (**`scripts`**) onde esta arquitetura modular não apenas organiza o projeto atual, mas o prepara para futuras expansões e trabalhos.

ARQUIVOS FONTES

A modularização do projeto foi feita através da separação entre interface e implementação nos seguintes arquivos:

- **`matrix.c / matrix.h`:** Implementa um Tipo Abstrato de Dados (**TAD**) para a manipulação de matrizes. Este módulo é genérico e contém todas as funções para criar, liberar e realizar operações algébricas com matrizes, como soma, multiplicação, transposição e cálculo do determinante, reutilizado de um laboratório anterior.
- **`robot.c / robot.h`:** Encapsula o modelo cinemático do robô, implementando as equações de estado e de saída.
- **`reference.c / reference.h`:** Responsável por gerar a trajetória de referência do robô, unicamente para traduzir as equações matemáticas da trajetória
- **`ref_model.c / ref_model.h`:** Simula o modelo de referência de primeira ordem que trabalha a movimentação do robô como objetivo dinâmico para sistema de controle, sendo a peça principal para o propósito de outras threads do sistema.
- **`control.c / control.h`:** Implementa a lógica do controlador e da linearização por realimentação, calculando as entradas **`v(t)`** e **`u(t)`**.
- **`main.c`:** Orquestra a simulação, inicializando e gerenciando as sete threads, os mutexes e o ciclo de vida da aplicação.

RESULTADOS

Nesta seção, são apresentados os gráficos de trajetória e a tabela de estatística gerados pela simulação do robô nos cenários com e sem carga de CPU.

- **Gráficos da Trajetória (Com e Sem Carga):**

  Os gráficos a seguir exibem a trajetória do centro de massa do robô, permitindo uma comparação visual do impacto da carga no caminho percorrido. Os gráficos a seguir foram gerados pelo script **`plot_lab3.m`** no GNU Octave.

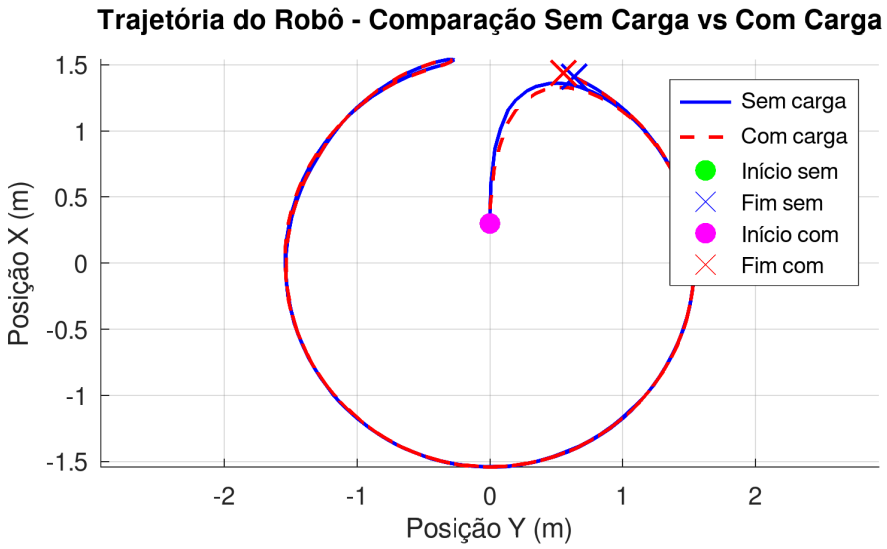

Figura 1: *Trajetória do robô nos cenários: Com e Sem Carga.*

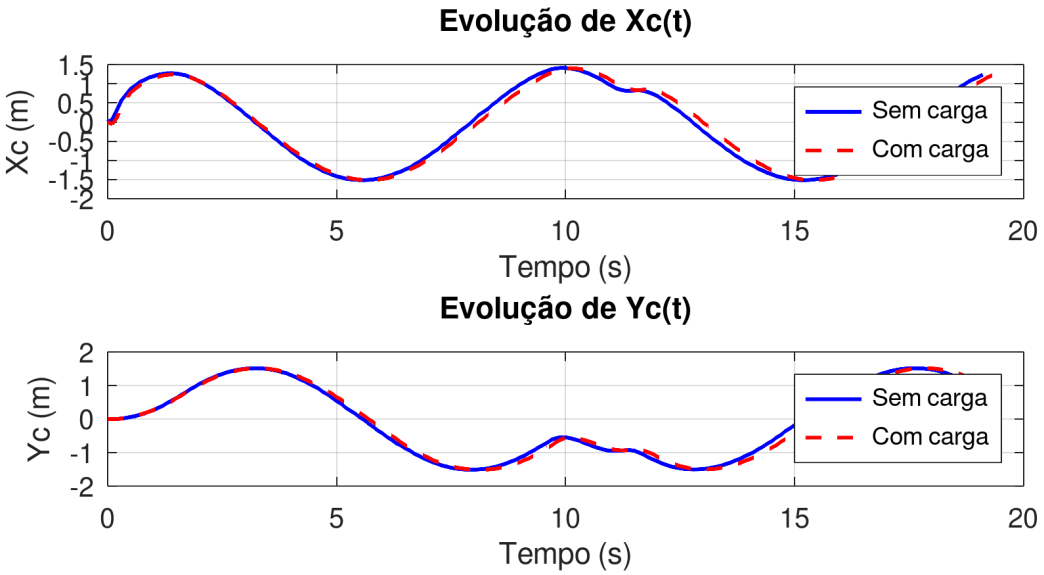

Figura 2: *Evolução temporal das posições* ***Xc(t)*** *e* ***Yc(t)****.*

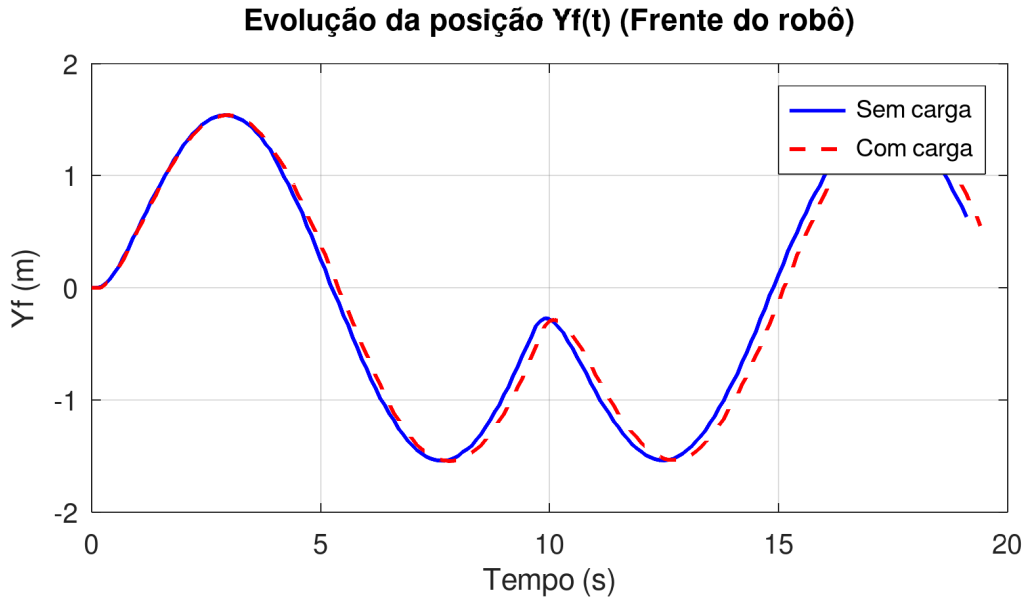

Figura 3: *Evolução da posição* ***Yf(t)*** *da frente do robô.*

- **Tabela de Estatísticas (Análise de Desempenho Temporal):**

A **Tabela 4** apresenta os resultados estatísticos do período $T(k)$ e do jitter $J(k)$ da tarefa de Controle/IO, comparando os cenários com e sem carga.

Métrica	Período T(k) - Sem Carga (ms)	Jitter J(k) - Sem Carga (ms)	Período T(k) - Com Carga (ms)	Jitter J(k) - Com Carga (ms)
Média	104,216471	4,216471	102,986372	2,986372
Variância	1,966739	1,966739	0,761421	0,761421
Desvio Padrão	1,402405	1,402405	0,872594	0,872594
Mínimo	102,365122	2,365122	102,175676	2,175676
Máximo	30,848892	0,848892	110,374070	10,374070

Tabela 4: *Comparativo de Métricas de Tempo para Período e Jitter (Com e Sem Carga).*

ANÁLISE FORMAL DA ESCALONABILIDADE

Para a análise em específico da escalonabilidade, foi utilizada a Análise de Taxa Monotônica (Rate-Monotonic Analysis - RMA), especificamente o teste de utilização de Liu & Layland que verifica se a utilização total da CPU pelo conjunto de tarefas é inferior a um limite teórico, garantindo assim que nenhum prazo seja perdido.

A análise foi realizada para as 5 tarefas periódicas principais do sistema, excluindo a thread de interface, que não possui requisitos de tempo real estritos. O teste de escalonabilidade de Liu & Layland para **n** tarefas estabelece o seguinte limite de utilização da CPU: $U \leq n(2^{1/n} - 1))$.

- **Resultado da análise de Utilização:**

Para cada tarefa, foi medido o Pior Tempo de Computação (*Ci*) nos cenários "Sem Carga" e "Com Carga". Com base nestes dados, a utilização da CPU ($Ui = Ci/Ti$) foi calculada e demonstrada conforme a **Tabela 5**.

Tarefa	Período (Ti)	Ci (Sem Carga)	Utilização (Sem Carga)	Ci (Com Carga)	Utilização (Com Carga)
Robô	30 ms	0.085 ms	0.28%	0.008 ms	0.03%
Linearização	40 ms	0.105 ms	0.26%	0.031 ms	0.08%
Controle	50 ms	0.022 ms	0.04%	0.010 ms	0.02%
Modelo Ref.	50 ms	0.010 ms	0.02%	0.005 ms	0.01%
Geração Ref.	120 ms	0.021 ms	0.02%	0.007 ms	0.01%
Total			**0.62%**		**0.15%**

Tabela 5: *Comparativo de Métricas do Pior Tempo de Computação (Ci) e a utilização da CPU (Com Carga e Sem Carga)*

- **Conclusão da Análise Formal:**

Para o nosso sistema com **n=5** tarefas, o limite é: $Limite = 5(2^{1/5} - 1) \approx 0.743$ ou **74.3%.** A utilização total medida em ambos os cenários (**0.62%** no cenário sem carga e **0.15%** no cenário com carga) é muito inferior ao limite teórico de 74.3%.

Portanto, conclui-se formalmente que o conjunto de tarefas **é escalonável** em ambas as condições. A análise teórica garante que a CPU tem capacidade mais do que suficiente para executar todas as tarefas dentro dos seus respectivos períodos sem perder deadlines.

É importante notar, no entanto, que a escalonabilidade teórica não garante a ausência de jitter, como foi observado na análise de desempenho temporal, destacando a diferença entre um sistema escalonável e um sistema de tempo real previsível.

ANÁLISE CRÍTICA DOS RESULTADOS

A análise dos resultados revela dois pontos principais: o fraco desempenho do controlador e o comportamento contraintuitivo do jitter do sistema.

Análise da Trajetória: Os gráficos (**Figura 1, 2 e 3**) mostram que o controlador não conseguiu fazer o robô seguir a trajetória de referência. O robô desvia-se significativamente do percurso, indicando que a sintonia dos ganhos (**`α=2.0`**) demonstrou baixa para garantir uma resposta rápida para corrigir os erros de percurso.

Atrasos e Dessincronização: A arquitetura multi-thread foi trabalhada com diferentes períodos de amostragem: Tarefas de Controle (50ms), Linearização (40ms) e Simulação do Robô (30ms) que operam de forma assíncrona. Neste caso, cria um atraso inerente entre o momento em que a posição do robô é analisada, o comando de controle é calculado e, finalmente, a nova velocidade é aplicada. Por exemplo, a **`thread_control`** calcula um comando com base numa posição do robô que já está "desatualizada" no momento em que a **`thread_robot_simulation`** o executa.

Em sistemas de controle digital multi-taxa, este atraso (latência) degrada significativamente o desempenho e pode levar à instabilidade, o que explica o fraco seguimento da trajetória observada.

Análise do Desempenho Temporal: A **Tabela 4** mostra que o cenário "Sem Carga" apresentou **maior instabilidade temporal** (maior variância e desvio padrão) do que o cenário "Com Carga". No Linux, o sistema nunca está verdadeiramente ocioso. Há em execução centenas de processos de fundo (serviços de rede, atualizações da interface gráfica, etc.) onde as tarefas são esporádicas e imprevisíveis, podendo interromper as threads da simulação em momentos aleatórios. Estas interrupções causam picos de latência (jitter) elevados e uma maior variabilidade no período de execução das tarefas.

Na introdução de uma thread de carga constante e intensiva, o escalonador do sistema operacional obtém um cenário de carga mais previsível. É provável que o escalonador tenha otimizado a execução, por exemplo, alocando a thread de carga a um núcleo de CPU específico e as threads da simulação a outro. Isto pode ter criado um ambiente de execução paradoxalmente mais "limpo" e isolado para a simulação, resultando em menor jitter.

CONCLUSÃO

Por fim, o trabalho cumpriu com sucesso o seu objetivo primário de projetar e implementar um sistema de controle concorrente para a simulação de um robô móvel. A arquitetura, dividida em sete threads com periodicidades distintas, foi implementada utilizando a biblioteca PThreads, e a sincronização de dados partilhados foi garantida através do uso de monitores (implementados com mutexes). No entanto, a análise experimental revelou os desafios práticos de se executar tarefas de controle em um sistema operacional de propósito geral.

Apesar de a análise formal provar que o sistema era teoricamente escalonável, os resultados práticos mostraram um fraco desempenho do controlador em seguir a trajetória e um comportamento contraintuitivo do jitter sob carga. Estes resultados demonstram que a capacidade teórica da CPU não garante a previsibilidade temporal necessária para um controle de alto desempenho.

Conclui-se que o laboratório serviu como uma demonstração eficaz da diferença fundamental entre um sistema escalonável e um sistema de tempo real, evidenciando a natureza não-determinística dos sistemas operacionais de propósito geral e a necessidade de RTOS para aplicações de controle críticas.