UFAM

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

Trabalho 2: Sistema de Controle Distribuído Simples

Aluno: MATHEUS ROCHA CANTO

INTRODUÇÃO

O objetivo deste laboratório foi desenvolver uma simulação de um sistema dinâmico — especificamente um robô móvel com acionamento diferencial — utilizando o paradigma de programação com múltiplas threads em linguagem C. O sistema, descrito por um modelo em espaço de estados, decomposto em duas tarefas concorrentes principais:

- A simulação do modelo do robô (com período de 50ms);
- A geração de sinais de controle e amostragem de dados.

Além da implementação, o trabalho inclui uma análise de desempenho temporal, medindo o jitter do sistema sob duas condições distintas: uma operação normal ("sem carga") e outra sob estresse de CPU, induzido por uma terceira thread de carga. Para a visualização e análise dos dados de trajetória gerados, foi utilizado o software **GNU Octave**, com um script dedicado para a criação de gráficos comparativos, que permitiram avaliar o impacto da carga de CPU no comportamento do sistema.

ESTRUTURA DA PASTA UTILIZADA

O projeto foi organizado na seguinte estrutura para garantir a separação de responsabilidades:

- **`Trabalho 2/`** (Pasta Raiz):
  - **data/:** Armazena os dados gerados e os gráficos.
  - **`include/`:** Contém os arquivos de cabeçalho das interfaces (**`.h`**).
  - **scripts/:** Script 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.

O módulo **matrix**, por exemplo, é totalmente agnóstico ao problema do robô e pode ser diretamente copiado para qualquer outro projeto que necessite de álgebra linear. O módulo **robot** encapsula um modelo específico, mas serve como um template; para simular um robô diferente, bastaria alterar as equações internas em **robot.c**, enquanto a estrutura de threads e a lógica de controle em **main.c** poderiam ser amplamente reaproveitadas. Por fim, o **Makefile** é genérico o suficiente para ser adaptado a novos projetos com mínima alteração, bastando adicionar os novos arquivos fonte.

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 toda a lógica de negócio específica do robô onde define a estrutura de dados para o estado do robô (**`RobotState`**) e implementa as equações cinemáticas do modelo em espaço de estados, como a atualização do vetor de entrada `u(t)` e o cálculo do novo estado `x(t)`.
- **plot_results.m:** Usada para carregar os dados de saída gerados após os resultados da simulação (**.txt**). Utilizando o GNU Octave, será gerado o plot dos gráficos para demonstrar o resultado da simulação com e sem carga, que então será salva na pasta **data/**
- **`main.c`:** É o ponto de entrada e o orquestrador da simulação. Incluem a análise de argumentos de linha de comando (para ativar o modo com carga), a inicialização do estado do robô e do mutex, a criação e gerenciamento das três threads (**Controle/IO, Simulação e Carga**), a medição do desempenho temporal e a finalização limpa 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_results.m`** no GNU Octave.

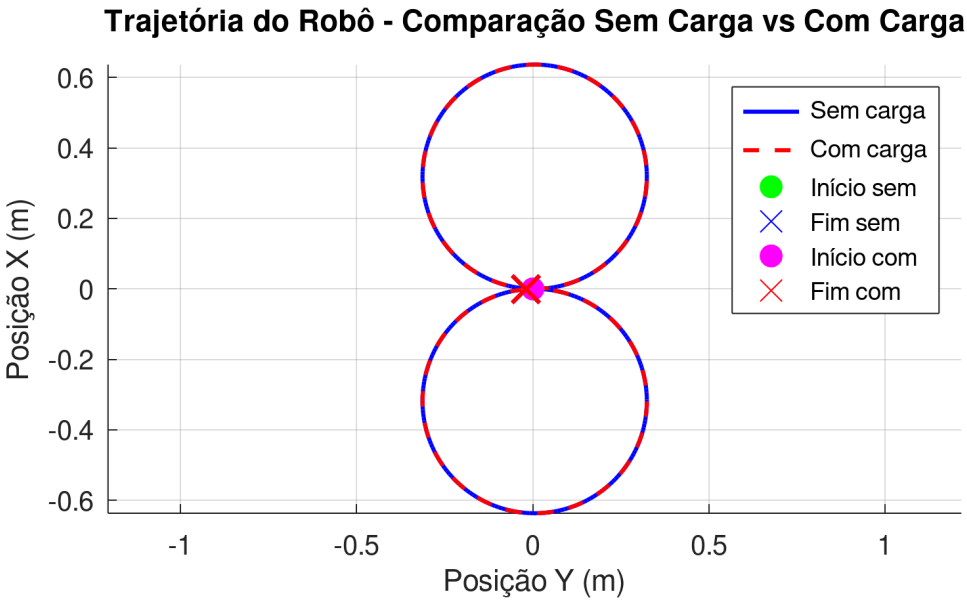

Figura 1: *Trajetória do robô nos cenários: Com e Sem Carga.*

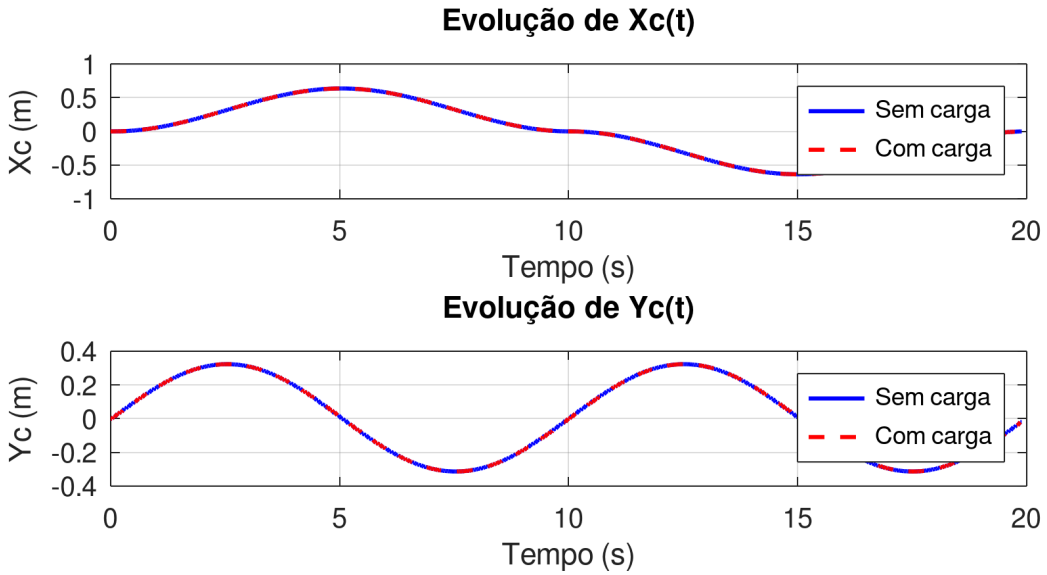

Figura 2: *Evolução temporal das posições* ***Xc(t)*** *e* ***Yc(t)****.*

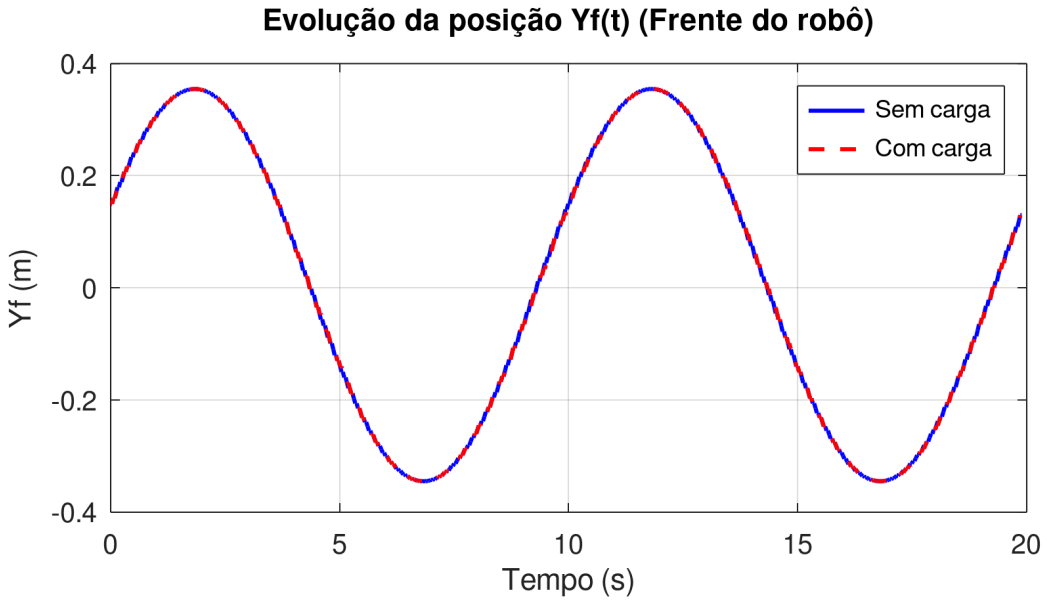

Figura 3: *Evolução da posição* ***Yf(t)*** *da frente do robô.*

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	30,142005	0,142005	30,167007	0,167007
Variância	0,013015	0,013015	0,002044	0,002044
Desvio Padrão	0,114082	0,114082	0,045212	0,045212
Mínimo	30,037555	0,037555	30,028488	0,028488
Máximo	30,848892	0,848892	30,283607	0,283607

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

ANÁLISE CRÍTICA DOS RESULTADOS

A análise dos resultados confirma o comportamento teoricamente esperado. Conforme a **Tabela 4**, o desvio padrão da tarefa de 30ms no cenário "Sem Carga" foi de **0.114 ms**, mais que o dobro do cenário "Com Carga", que foi de apenas **0.045 ms**. Além disso, o pico de latência (período máximo) atingiu **31.84 ms** no sistema "Sem Carga", um atraso extremo de quase **2 ms**, enquanto o sistema "Com Carga" teve um pico muito menor, de **30.28 ms**.

O sistema “Sem Carga” está constantemente sujeito a interrupções esporádicas e de alta prioridade de outros processos do sistema, então “roubam” tempo de CPU da simulação em momentos aleatórios e causando uma maior variância no período. Por fim, ao introduzir uma thread de carga constante e intensiva, forçamos o escalonador do sistema operacional a tomar uma decisão mais estruturada. A hipótese seria que o escalonador pode ter migrado a thread de carga para um núcleo de CPU específico e deixando as outras threads da simulação em outro núcleo, resultando em um menor jitter.

Em contrapartida, os gráficos (**Figura 1, 2** e **3**) mostram que a trajetória é visualmente idêntica em ambos os cenários. Isso indica que o jitter medido, embora significativo, foi insuficiente para causar um erro de posição acumulado que alterasse o resultado macroscópico da simulação. O maior pico de atraso medido foi de aproximadamente **1.85 ms** (no caso "Sem Carga"). Durante esse pequeno intervalo de tempo, com uma velocidade de **0.2 m/s**, o erro de posição acumulado seria de apenas **0.2m/s × 0.00185s ≈ 0.00037 metros**, ou **0.37 milímetros**. Este erro é insignificante na escala da simulação e tende a ser compensado nos ciclos seguintes, não resultando em um desvio macroscópico da trajetória planejada.

CONCLUSÃO

O laboratório permitiu implementar com sucesso a simulação de um sistema dinâmico com múltiplas threads. A análise de desempenho temporal demonstrou de forma prática a complexidade e a natureza não determinística do agendamento de tarefas em um sistema operacional de propósito geral. Este experimento evidencia a necessidade de Sistemas Operacionais de Tempo Real (**RTOS**) para aplicações onde garantias temporais estritas são críticas.

Esses resultados indicam que, enquanto a trajetória cinemática do robô não foi visivelmente afetada pela carga de CPU, o desempenho temporal do sistema sofreu um impacto significativo e contraintuitivo. Observou-se que a execução no cenário "Sem Carga" apresentou uma instabilidade temporal (maior variância e picos de jitter) drasticamente maior do que no cenário "Com Carga".

Este fenômeno demonstra a natureza não-determinística de um sistema operacional de propósito geral, onde a interferência de processos de fundo imprevisíveis pode degradar mais a performance do que uma carga de trabalho constante e conhecida. A coleta de timestamps de alta resolução no nível do SO foi, portanto, crucial para capturar e analisar este comportamento complexo do escalonador de tarefas.