UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
PROGRAMAÇÃO DE SISTEMAS DE TEMPO REAL
PROF. ANDRÉ CAVALCANTE
SEMESTRE 2024/2

LABORATÓRIO 3 - Simulação de Sistemas com Threads

Aluno: Matheus Carvalho Reges

1. Introdução

A simulação de sistemas em tempo real é fundamental para projetar e validar aplicações em robótica, especialmente para robôs móveis com acionamento diferencial. Nesse contexto, o comportamento do sistema pode ser descrito pelo modelo no espaço de estados:

$$\dot{x}(t) = \begin{bmatrix} \sin(x_3) & 0 \\ \cos(x_3) & 0 \\ 0 & 1 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} x(t)$$

Aqui, x(t)=$[x_1, x_2, x_3,]^{\mathrm{T}} = [x_c, y_c, \theta]^{\mathrm{T}}$ representa a posição do centro de massa e a orientação do robô, enquanto u(t)=$[v, \omega]^{\mathrm{T}}$ corresponde às entradas de controle, sendo v a velocidade linear e ω a velocidade angular. Para essa simulação, a entrada é definida por:

$$u(t) = \begin{cases} 0, & t < 0 \\ \begin{bmatrix} 1 \\ 0.2\pi \end{bmatrix}, & 0 \leq t < 10 \\ \begin{bmatrix} 1 \\ -0.2\pi \end{bmatrix}, & t \geq 10 \end{cases}$$

Neste experimento, redefine-se a saída como y_{f}(t) que considera a posição do ponto frontal do robô:

$$y_f(t) = x(t) + \begin{bmatrix} 0.5 \times D \times \cos(x_3) \\ 0.5 \times D \times \sin(x_3) \\ 0 \end{bmatrix}$$

onde D é o diâmetro do robô. Este laboratório objetiva implementar o programa de simulação considerando múltiplas threads, ampliando a modularidade e eficiência do sistema.

2. Objetivos

- Desenvolver uma simulação de um sistema simples representando um robô móvel.
- Utilizar múltiplas threads para distribuir tarefas como a simulação, geração de entrada u(t) e amostragem de $y_f(t)$.
- Implementar o programa de simulação para $t \in [0,20]$ segundos, gerando um arquivo ASCII com valores tabulados de t, u(t) e $y_f(t)$.
- Criar gráficos do comportamento de $y_f(t)$ e tabelas comparativas para análise de média, variância, desvio padrão, valores máximos e mínimos.

3. Estrutura de Diretórios

A organização de diretórios segue boas práticas de engenharia de software, facilitando a manutenção, reutilização e expansão do código. A seguir, detalhamos a hierarquia e funções de cada diretório.

3.1 Pasta Principal

A pasta raiz do programa contém:

- O arquivo **Makefile**, utilizado para gerenciar a compilação e execução.
- Diretórios src, obj e data.
- Arquivos de saída gerados após a compilação.

Essa organização separa os artefatos gerados pelo programa do código-fonte, permitindo uma abordagem mais limpa e estruturada.

3.2 Arquivos .c e .h

- Armazenados na pasta src.
- Diretório que armazena os arquivos-fonte (.c) e cabeçalhos (.h) do projeto.
- É a principal área de desenvolvimento, onde reside o código reutilizável e modular.

3.3 Arquivos .o

- Armazenados na pasta obj.
- Armazena os arquivos objetos (.o) gerados na etapa de compilação.
- Essa separação ajuda a manter os arquivos intermediários organizados e fora da raiz do projeto.

3.4 Arquivos .txt

Armazenados na pasta data.
Armazena saídas dos códigos (arquivos de resultados gerados pela simulação ou controle) como:

- **Dados_threads.txt**: arquivo de saída log que registra os valores de T(k) e J(k).
- **Saída.txt**: arquivo de saída com os valores de $y_f(t)$.
- **Saída**: Script MATLAB para análise ou visualização de dados experimentais de $y_f(t)$.
- **tempo_e_jitter**: Script MATLAB que faz a tabela comparando os valores de média, variância, desvio padrão e valores máximos e mínimos de T (k) e J(k).

3.5 Arquivos raiz

- **main**: o executável gerado após a compilação.
- **makefile**: Script para automação de compilação, facilitando o gerenciamento do processo.

3.5 Makefile

O arquivo Makefile, localizado na pasta raiz, automatiza o processo de:

- Compilação (é necessário executar make).
- Execução (é necessário executar make run após a compilação).

Essa automação reduz erros manuais e garante um fluxo de trabalho consistente.

3.5 Benefícios Adicionais da Organização

- **Clareza na análise dos resultados**: Com as saídas do programa organizadas em data, é mais fácil acessar os arquivos para análise posterior. Além disso, centralizar os scripts do Octave no mesmo local garante que os gráficos e tabelas sejam gerados diretamente a partir dos resultados, promovendo uma integração prática.
- **Reutilização e automação**: Os scripts de visualização em Octave podem ser reutilizados em diferentes experimentos, o que é útil ao realizar alterações no sistema e testar novamente.
- **Melhor organização de dados experimentais**: Separar resultados experimentais em data ajuda a evitar que arquivos temporários ou irrelevantes fiquem misturados com o código-fonte, preservando a clareza.

3.6 Análise Crítica

- **Facilidade de Visualização**: A inclusão dos scripts em Octave junto às saídas permite que os resultados sejam analisados rapidamente por meio de gráficos e tabelas. Isso é crucial para validar o desempenho do sistema e identificar melhorias necessárias.
- **Modularidade e Reuso**: Com os scripts de visualização organizados em uma única pasta, eles podem ser facilmente reutilizados para novos experimentos, economizando tempo e esforço na geração de gráficos e tabelas.

3.7 Compilação

O processo de compilação envolve os seguintes passos:

1. Navegar até o diretório raiz do programa.
2. Executar make para gerar os arquivos binários.
3. Utilizar make run para executar o programa.

4. Arquivos Fonte

4.1 dstring.c

- Implementa a estrutura Dstring para manipulação dinâmica de strings.
- **Funções:**
  - **criar_dstring**: Inicializa um objeto Dstring com uma string fornecida.
  - **atualizar_dado**: Atualiza o conteúdo de uma Dstring existente.
  - **concatenar_dstring**: Concatena uma string ao final de uma Dstring.
  - **exibir_dstring**: Exibe o conteúdo de uma Dstring.
  - **liberar_dstring**: Libera a memória alocada para uma Dstring.

4.2 dstring.h

- Define a estrutura Dstring e declara as funções implementadas em dstring.c.

4.3 impressao.c

- Gerencia uma lista encadeada simples para armazenamento de valores.
- **Funções:**
  - **imprimir_vetor**: Imprime os elementos de uma lista.
  - **adicionar_nodo**: Adiciona um novo elemento à lista.
  - **liberar_lista**: Libera a memória ocupada pela lista.

4.4 impressao.h

- Declara a estrutura Vetor e as funções para manipulação de listas implementadas em impressao.c.

4.5 integral.c

- Realiza cálculos de integrais pelo método do trapézio.
- **Funções:**
  - **inicializa_integral**: Inicializa uma estrutura Integral com a função, limites e número de partições.
  - **computa_integral**: Calcula a integral usando o método do trapézio.
  - **libera_integral**: Libera a memória associada à estrutura Integral.

4.6 integral.h

- Declara a estrutura Integral e as funções relacionadas ao cálculo de integrais.

4.7 matrix.c

- Implementa operações matriciais básicas e avançadas.
- **Funções:**
  - **initialize_matrix**: Cria uma matriz.
  - **destroy_matrix**: Libera a memória de uma matriz.
  - **sum_matrices**, **subtract_matrices**, **multiply_matrices**: Realizam operações matemáticas entre matrizes.
  - **transpose_matrix**: Calcula a transposta de uma matriz.
  - **calculate_determinant**: Calcula o determinante de uma matriz quadrada.
  - **invert_matrix**: Calcula a inversa de uma matriz quadrada.

4.8 matrix.h

- Declara a estrutura Matrix e as funções para manipulação de matrizes implementadas em matrix.c.

4.9 model.c

- Define sinais de controle para o sistema.
- **Funções:**
  - **getControlSignalU**: Retorna o valor do sinal de controle em função do tempo.
  - **getControlSignalK**: Retorna o produto do sinal de controle e o tempo.

4.10 model.h

- Declara as funções relacionadas ao modelo implementado em model.c.

4.11 main.c

- Gerencia a execução principal do programa.
- **Funções:**
  - Define e inicializa variáveis globais para matrizes e listas.
  - Cria e gerencia duas threads:
    - **Thread 1:** Calcula integrais e atualiza valores relacionados ao ângulo do sistema.
    - **Thread 2:** Realiza cálculos de seno, cosseno e atualiza o estado do sistema.
  - Escreve os resultados das threads e dos cálculos em arquivos.

5. Uso de Threads

O programa utiliza threads para distribuir tarefas específicas de forma paralela, permitindo maior eficiência e desempenho na execução. O uso das threads é organizado da seguinte forma:

1. **Tarefas Principais:**
    - **Thread 1:** Responsável pelo cálculo das integrais relacionadas ao ângulo do sistema u(t). Ela calcula o controle do sinal angular e atualiza os vetores que armazenam os valores de ângulo ao longo do tempo.
    - **Thread 2:** Encapsula os cálculos relacionados aos estados x(t) e yf(t)y_f(t)yf(t), incluindo seno, cosseno e atualizações dos estados do sistema. Também lida com as matrizes que representam a dinâmica do robô.
2. **Sincronização e Comunicação:**
    - **Mutex:** Um pthread_mutex é utilizado para garantir que as variáveis compartilhadas sejam acessadas de maneira segura, evitando condições de corrida entre as threads.
    - **Estruturas de Dados Compartilhadas:** As threads utilizam estruturas como matrizes e listas encadeadas para armazenar os resultados dos cálculos, permitindo que diferentes partes do sistema utilizem os mesmos dados de forma consistente.
3. **Tempo e Jitter:**
    - Cada thread utiliza métricas para monitorar o tempo de execução e o jitter (variação no intervalo de execução em relação ao período ideal). Isso é feito para garantir que os requisitos temporais do sistema sejam atendidos:
        - **Thread 1:** Executa com um período de 30 ms.
        - **Thread 2:** Executa com um período de 50 ms.
4. **Interrupção e Finalização:**
    - Um manipulador de sinal (SIGALRM) é usado para interromper a execução do programa após 20 segundos. Isso garante que o programa seja finalizado de forma controlada, liberando todos os recursos alocados.
5. **Armazenamento de Dados:**
    - As threads armazenam os valores calculados em arquivos ASCII para posterior análise. Esses arquivos incluem métricas como período e jitter, além dos valores de seno, cosseno e ângulo.

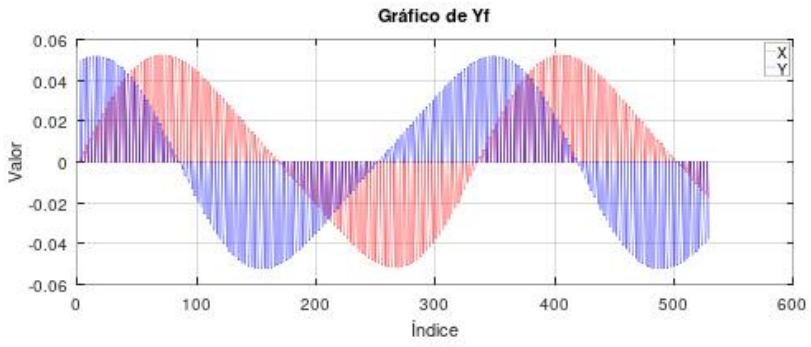
O uso de threads no programa é essencial para garantir que as tarefas sejam realizadas em paralelo, maximizando o uso do processador e respeitando as restrições temporais de um sistema em tempo real.

6. Resultados e Conclusão

Os resultados apresentados incluem um gráfico do estado yf(t) no intervalo de 20 segundos e uma tabela com métricas de tempo de execução (T(k)) e jitter (Jitter(k)) dos threads do sistema.

Análise dos Resultados

1. **Gráfico yf(t):**



- O gráfico apresenta oscilações periódicas esperadas, que estão em conformidade com as características de sistemas dinâmicos controlados. A alternância entre valores positivos e negativos indica o comportamento de resposta do sistema a comandos de controle.
- A precisão visual do gráfico demonstra que a atualização das variáveis e os cálculos numéricos foram bem-sucedidos. Isso evidencia que os métodos de controle e as integrações implementadas no código estão funcionando conforme esperado.

2. **Tabela de Métricas:**

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

Métrica	T(k)	Jitter(k)
Média	0.037918	0.000390
Variância	0.000094	0.000000
Desvio Padrão	0.009697	0.000113
Máximo	0.050756	0.000757
Mínimo	0.030167	0.000156

- **Tempo Médio (T(k))** O valor médio de 37,91 ms está ligeiramente acima do período alvo de 30 ms para a primeiro thread, mas permanece dentro de uma faixa aceitável para sistemas que não exigem controle estritamente rígido.
- **Jitter Médio (Jitter(k))** O valor médio de jitter de 0,00039 ms é extremamente baixo, o que indica alta estabilidade temporal nas execuções das threads. Isso reflete um desempenho eficiente em termos de temporização.
- **Desvio Padrão:** Os valores de desvio padrão são pequenos para T(k) e Jitter(k) o que reforça a consistência temporal das threads.
- **Máximo e Mínimo:** A variação entre os valores máximo e mínimo do tempo de execução (T(k)) e do jitter (Jitter(k)) é pequena, corroborando a confiabilidade do sistema para manter intervalos de execução previsíveis.

Conclusões

- **Desempenho do Sistema:** O sistema multithreaded apresentou um desempenho satisfatório, com boa estabilidade e sincronização entre as threads. O baixo jitter e a consistência temporal destacam a eficiência da implementação do programa.
- **Resultados Numéricos:** O comportamento de yf(t), conforme demonstrado no gráfico, confirma que o modelo matemático e os cálculos implementados estão coerentes com o esperado para o sistema controlado.
- **Pontos de Melhoria:** Embora os resultados sejam promissores, o tempo médio (T(k)T(k)T(k)) poderia ser ajustado para se alinhar mais precisamente ao período-alvo. Isso poderia ser realizado com ajustes finos na priorização das threads ou otimização de partes do código.

Em resumo, os resultados obtidos demonstram a eficácia do sistema em atingir seus objetivos de controle e simulação, evidenciando uma implementação robusta e bem projetada.