UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
PROGRAMAÇÃO DE SISTEMAS DE TEMPO REAL
PROF. ANDRÉ CAVALCANTE
SEMESTRE 2023/1

LABORATÓRIO 3 - Simulação de Sistemas com Threads

1. Objetivos

Este laboratório tem como objetivo continuar o sistema de controle baseado no movimento de um robô móvel, porém implementando o conceito de múltiplas *threads*. Além disso, também será necessário importar as bibliotecas criadas no Laboratório 1 para realizar os cálculos necessários e gerar gráficos das funções resultantes por meio do software *GeoGebra*.

2. Problema

A simulação do laboratório passado foi feita a partir de um robô móvel, que tem seu acionamento diferencial descrito pelo seguinte modelo de espaço de estados:

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

Também foi feito um programa na linguagem C que simulou a resposta desse sistema para a entrada $u(t)$ descrita abaixo:

$$u(t) = \begin{cases} 0 & \text{, para t} < 0 \\ \begin{bmatrix} 1 \\ 0.2\pi \end{bmatrix} & \text{, para } 0 \leqslant t < 10 \\ \begin{bmatrix} 1 \\ -0.2\pi \end{bmatrix} & \text{, para } t \geqslant 10 \end{cases}$$

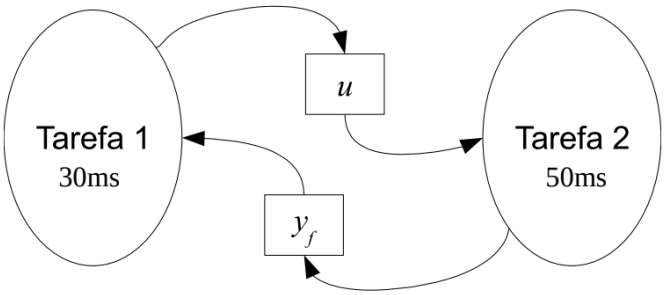
Agora é preciso dividir a simulação em duas tarefas:

- A primeira fará a simulação em si;
- E a segunda fará a geração de $u(t)$ e a amostragem de $y_f(t)$, dado por:

$$y_f(t) = x(t) + \begin{bmatrix} 0.5 \times D \times cos(x_3) & 0 & 0 \\ 0 & 0.5 \times D \times sen(x_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} x(t)$$

A partir disso, será feito um programa na linguagem C que simula a resposta desse sistema utilizando múltiplas tarefas (*threads*). Na imagem a seguir temos um diagrama que representa o programa com a implementação de *threads*:

Figura 1. Diagrama que representa o fluxo do programa.



Fonte: CAVALCANTE, André.

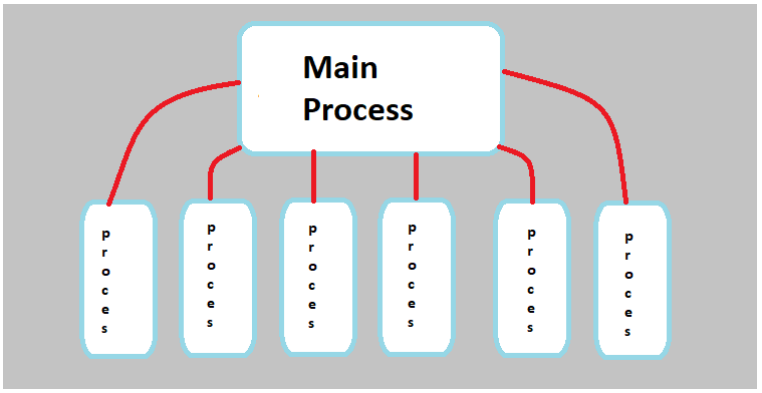
3. Introdução Teórica

Antes de começar a implementar o programa e apresentar alguns resultados, primeiro é necessário explicar como funciona um sistema com múltiplas *threads*.

3.1. *Threads*

Uma *thread* ou tarefa é um fluxo de sequência única dentro de um processo. Elas são uma forma de melhorar uma aplicação por meio do paralelismo. Por exemplo, em um navegador web, cada guia aberta pode ser uma *thread* diferente.

Figura 2. Representação de multithreading em C.



Fonte: TUFAN, Burak Hamdi.

Apesar de possuir algumas características semelhantes a um processo, uma tarefa acaba operando de forma mais rápida pelos seguintes fatores:

- A criação de *threads* é muito mais rápida;

- A alternância de contexto entre *threads* é mais rápida;
- As *threads* podem ser encerradas facilmente;
- E a comunicação entre elas é mais rápida.

Agora, para escrever um programa multithreading na linguagem C, é preciso utilizar a biblioteca POSIX Threads (ou Pthreads), e, para a implementação do pthread, foi utilizado o compilador gcc.

Primeiramente, foi implementado uma função que ficará responsável por realizar a simulação e também calcular o valor de $u(t)$ em um intervalo de 30 ms.

Figura 3. Código da função que realiza a Tarefa 1.

```
16  // Função que será executada pela thread 1 quando ela for iniciada no main
17  void *tarefa_1(void *arg) {
18      // Mensagem para sinalizar que a tarefa começou
19      printf("Tarefa 1 começou!\n");
20
21      // Função para calcular u(t)
22      u = calcula_u(t);
23
24      // Aguarda 30 ms
25      usleep(30000);
26
27      // Mensagem para sinalizar que a tarefa terminou
28      printf("Tarefa 1 terminou!\n");
29  }
```

Fonte: Visual Studio Code.

Depois, foi implementado uma outra função que ficará responsável por calcular o valor de $yf(t)$ em um intervalo de 50 ms.

Figura 4. Código da função que realiza a Tarefa 2.

```
32  // Função que será executada pela thread 2 quando ela for iniciada no main
33  void *tarefa_2(void *arg) {
34      // Mensagem para sinalizar que a tarefa começou
35      printf("Tarefa 2 começou!\n");
36
37      // Função para calcular u(t)
38      u = calcula_u(t);
39
40      // Função para calcular yf(t)
41      yf = calcula_yf(t, u, diametro);
42
43      // Aguarda 50 ms
44      usleep(50000);
45
46      // Mensagem para sinalizar que a tarefa terminou
47      printf("Tarefa 2 terminou!\n");
48  }
49
```

Fonte: Visual Studio Code.

E, por fim, as duas *threads* foram criadas dentro de um *while* que fica dentro da função principal *main*. A condição de parada é se o tempo decorrido, que é sempre incrementado, chegar a 20 segundos.

Figura 5. Código da estrutura responsável por criar as *threads*.

```c
// Enquanto o tempo limite (20s) não for atingido...
while (t <= 20.0) {

    // 1º) Cria as duas threads
    pthread_create(&thread_1, NULL, tarefa_1, NULL);
    pthread_create(&thread_2, NULL, tarefa_2, NULL);

    // 2º) Espera pelo término das duas threads
    pthread_join(thread_1, NULL);
    pthread_join(thread_2, NULL);

    // 3º) Escreve no arquivo o resultado das operações
    escrever_linha(nome_arquivo, t, u, yf);

    // 4º) Incrementa a variável responsável pelo tempo decorrido do programa
    t++;

    // 5º) Começa um loop de espera até completar 1 segundo
    sleep(1);
}
```

Fonte: Visual Studio Code.

4. Hierarquia de Diretórios Utilizada

Antes de ter uma visão sobre os arquivos fontes gerados, é preciso discutir sobre a estrutura dos diretórios. Na imagem abaixo pode-se observar todos os 9 arquivos fontes utilizados no programa e mais 2 pastas:

Figura 6. Estrutura dos diretórios utilizada.

Fonte: Explorador de Arquivos do Windows 11.

- *.vscode*: pasta criada pelo Visual Studio Code e que contém algumas configurações do projeto em questão, podendo ser definições de depuração, tarefas e/ou configurações específicas do ambiente;
- *Documentos*: pasta que contém este relatório; o arquivo oferecido pelo professor e o arquivo .ggb, no qual os gráficos dos resultados estão.

Além disso, todos esses arquivos estão na pasta *Laboratório 3*, e o mesmo será entregue para avaliação na forma de arquivo “.zip”.

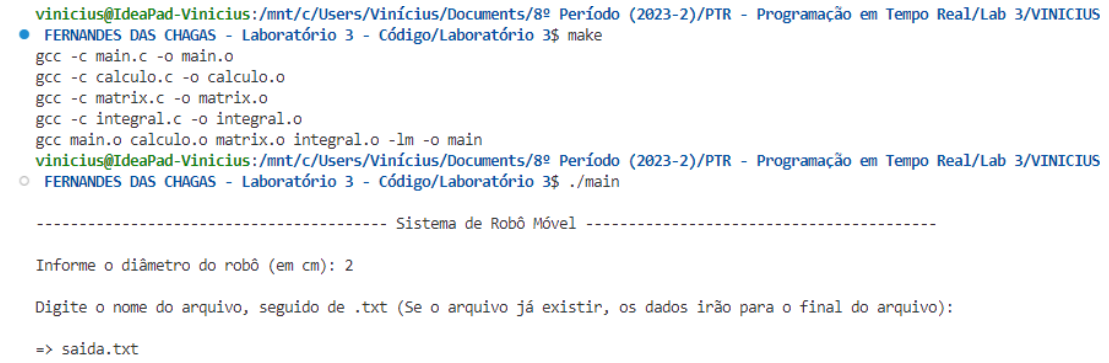
5. Resultados

Depois de implementar todos os cálculos necessários para o robô e ao executar o programa usando o software Visual Studio Code (VS Code), os seguintes resultados foram obtidos:

5.1. Saída do Programa

Ao compilar o programa com êxito, o seguinte menu de interação pode ser observado, em que o usuário precisa informar o tamanho do diâmetro do robô e também o nome de um arquivo .txt no qual o mesmo queira salvar os resultados de t, $u(t)$ e $yf(t)$ durante a simulação:

Figura 7. Compilação do código main.c, com o diâmetro sendo de 2 cm.



Fonte: Visual Studio Code.

É importante destacar que uma modificação foi feita: agora o programa é executado em tempo real, ou seja, a saída é gerada a cada período de tempo (de 0 a 20 segundos).

Figura 8. Arquivo ASCII gerado como saída.

saida.txt

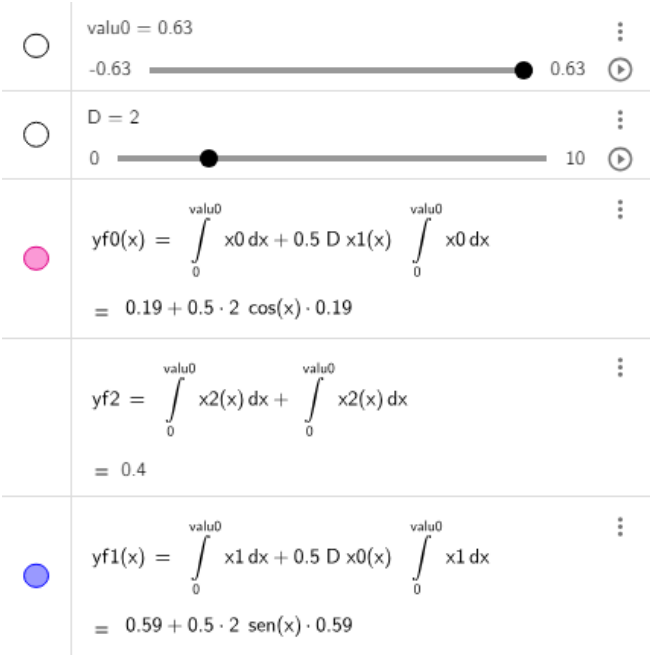
```
 1   t = 0   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 2   t = 1   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 3   t = 2   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 4   t = 3   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 5   t = 4   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 6   t = 5   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 7   t = 6   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 8   t = 7   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
 9   t = 8   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
10   t = 9   uT = [1.00, 0.63]   yfT = [0.38, 0.70, 0.39]
11   t = 10  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
12   t = 11  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
13   t = 12  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
14   t = 13  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
15   t = 14  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
16   t = 15  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
17   t = 16  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
18   t = 17  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
19   t = 18  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
20   t = 19  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
21   t = 20  uT = [1.00, -0.63]  yfT = [0.38, -0.70, 0.39]
```

Fonte: Visual Studio Code.

5.2. Gráfico da Saída Gerada

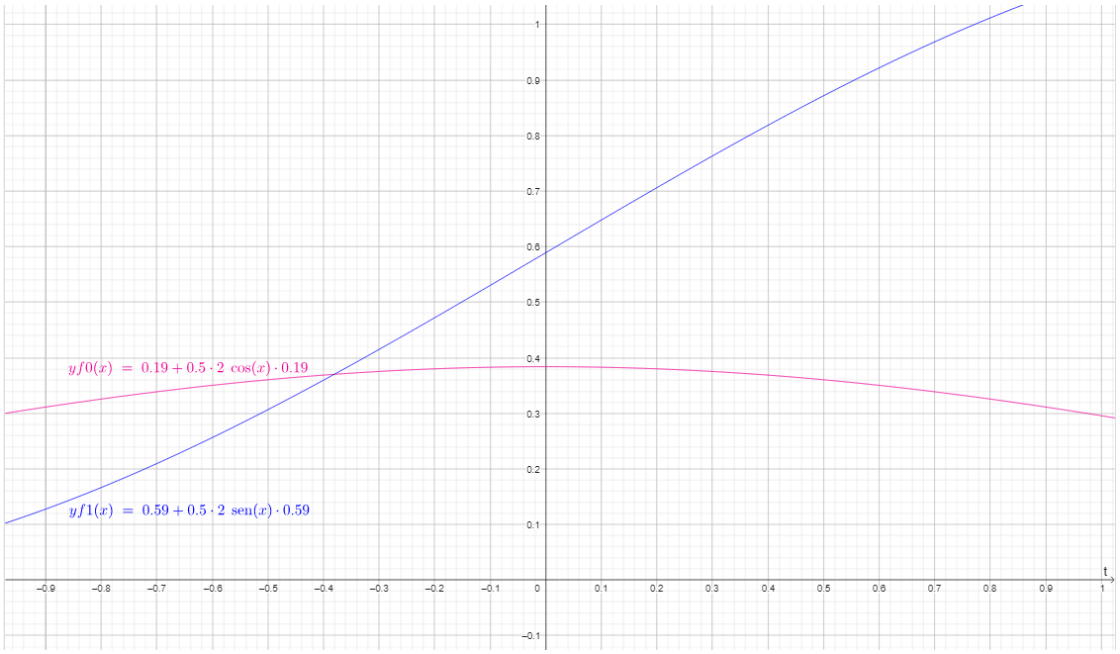
Utilizando o software GeoGebra, foi possível montar os gráficos de $yf(t)$ para um diâmetro de 2 cm e variando de acordo com o valor de $u(t)$:

Figura 9. Valores definidos de $yf(t)$ para $u(t) = [1.00;\ 0.63]$.



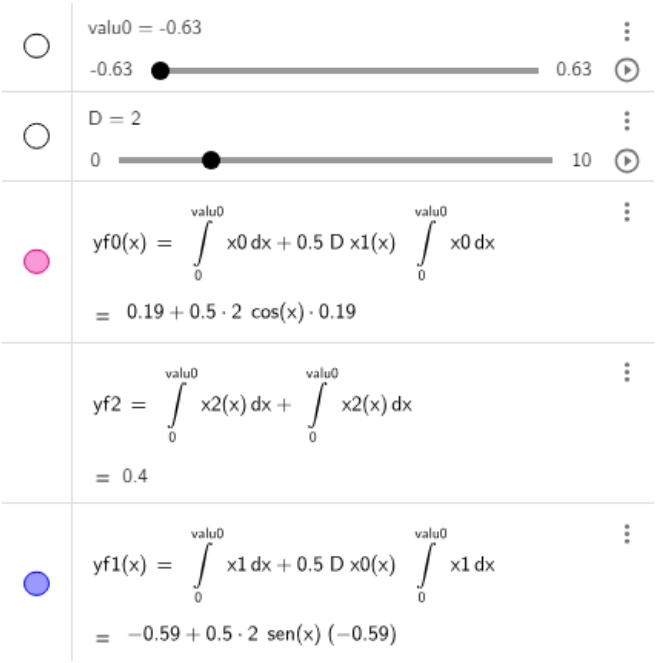
Fonte: GeoGebra.

Figura 10. Representação gráfica de $yf(t)$ para $u(t) = [1.00;\ 0.63]$.



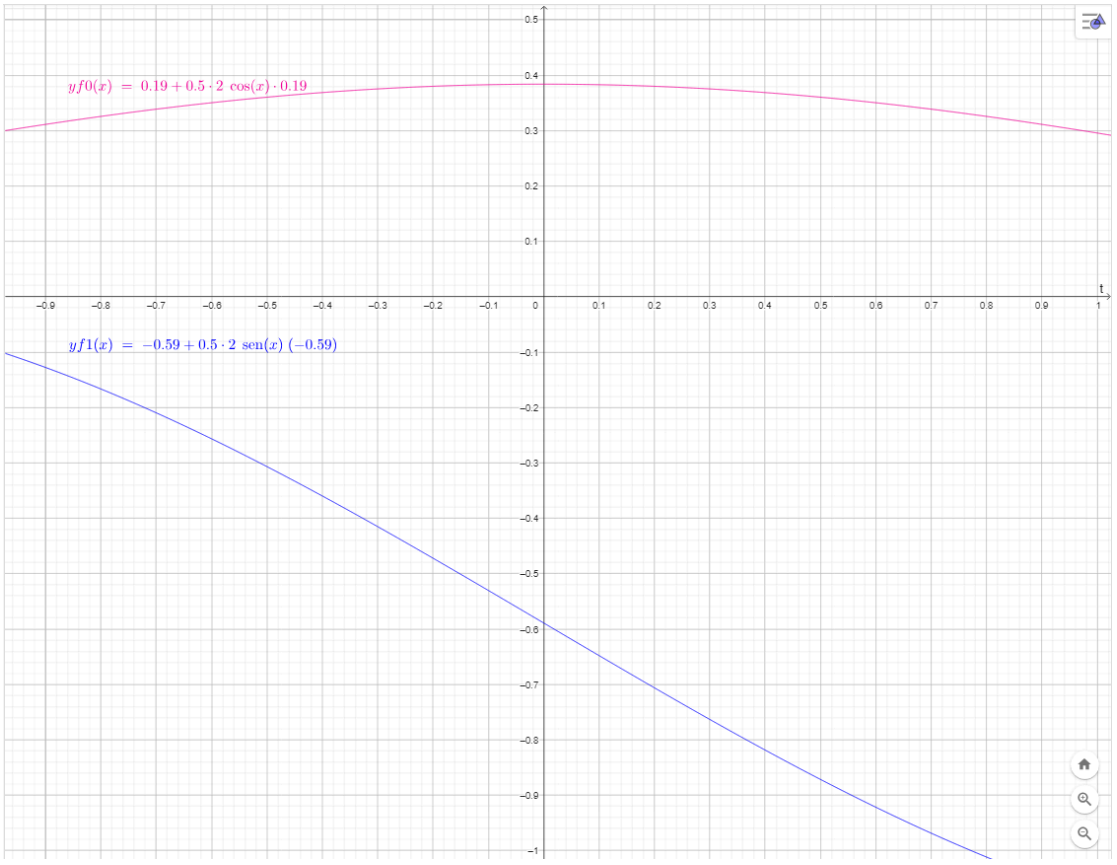
Fonte: GeoGebra.

Figura 11. Valores definidos de $yf(t)$ para $u(t) = [1.00; \ -0.63]$.



Fonte: GeoGebra.

Figura 12. Representação gráfica de $yf(t)$ para $u(t) = [1.00; \ -0.63]$.



Fonte: GeoGebra.

6. Conclusão

Neste trabalho, foi realizado um algoritmo na linguagem C , utilizando o Visual Studio Code, e com o intuito de simular um sistema de controle simples com multithreading. Além disso, foi gerado um arquivo ASCII com a saída gerada pelo código e gráficos que mostram o comportamento do sistema, utilizando o software GeoGebra.

Ao observar os resultados obtidos nos tópicos anteriores, podemos concluir que este laboratório se mostrou eficiente ao criar uma simulação de um sistema de controle que simulasse o comportamento de um robô móvel com acionamento diferencial.

7. Referências

ARORA, Himanshu. Introduction To Linux Threads – Part I, II e III. The Geek Stuff. 30/03/2012. Disponível em: <http://www.thegeekstuff.com/2012/03/linux-threads-intro/>. Acesso em 15 de fevereiro de 2024.

AZEVEDO, Caio. **Introdução, notação e revisão sobre matrizes**. IMECC-Unicamp. Acesso em 15 de fevereiro de 2024.

JOMAR. **Matrizes**. Universidade Federal do Paraná. Acesso em 15 de fevereiro de 2024.

OGATA, K. (1995). **Introduction to Discrete-Time Control Systems**. Englewood Cliffs, New Jersey, EUA: Prentice Hall.

SANTANA, Adrielle C. **Sistemas discretos em espaço de estados**. Disponível em: <http://professor.ufop.br/sites/default/files/adrielle/files/aula_8_3.pdf>. Acesso em 15 de fevereiro de 2024.

SILVA, Guilherme S. **Integrais**. Todo Estudo. Disponível em: <https://www.todoestudo.com.br/matematica/integrais>. Acesso em 15 de fevereiro de 2024.