Logotipo

Descrição gerada automaticamente

UNIVERSIDADE FEDERAL DO AMAZONAS

FACULDADE DE TECNOLOGIA

ENGENHARIA DA COMPUTAÇÃO

LABORATÓRIO 1 - PREPARAÇÃO DO AMBIENTE

Daniel Santiago da Silva

MANAUS-AM

2022

Daniel Santiago da Silva

LAB 01: SIMULAÇÃO DE SISTEMAS E REALIMENTAÇÃO

Primeiro Relatório da disciplina de Programação de Sistemas de Tempo Real referente à preparação do ambiente de trabalho que será avaliado pelo professor André Cavalcante junto ao Curso de Engenharia da Computação.

Professor: André Cavalcante

MANAUS-AM

2022

**SUMÁRIO**

1. **OBJETIVOS ...........................................................................................3**
2. **INTRODUÇÃO TEÓRICA ......................................................................4**
3. **ESTRUTURA E ORGANIZAÇÃO DOS DIRETÓRIOS ...........................4**
   1. ARQUIVOS FONTES **.........................................................................4**
   2. ESTRUTURA DE DIRETÓRIOS **........................................................7**
   3. MAKEFILE **..........................................................................................8**
4. **ADTS ....................................................................................................18**
   1. DSTRING**...........................................................................................18**
   2. MATRIX**.............................................................................................20**
   3. INTEGRAL**.........................................................................................21**
5. **CONCLUSÃO ......................................................................................31**
6. **OBJETIVOS**

Os objetivos desse primeiro laboratório de Programação de Sistemas de Tempo Real envolvem o desenvolvimento de ADTs de suporte além da configuração de um ambiente de trabalho em utilizando a linguagem C padrão 17.

As ADTs criadas neste trabalho são a de operações matriciais em ponto flutuante, a de integração de funções e a de criação de strings dinâmicas. Essas bibliotecas terão fundamental importância nos trabalhos que serão feitos ao decorrer do curso.

1. **INTRODUÇÃO TEÓRICA - ADT**

As linguagens de programação têm suas ferramentas e métodos que nos permitem criar tipos de dados além dos primitivos contidos na própria linguagem. Esses tipos são chamados de ADTs (Abstract Data Types) e estão em constante uso na implementação de diversas soluções no mundo. Além de especificar as informações contidas nesses dados, as ADTs envolvem as operações, também conhecidas como métodos, que as manipulam.

Esses métodos são definidos em uma interface conhecida como API (Applications Programming Interface), contudo sua implementação ocorre em outra parte do código. Essa implementação deve levar em conta os objetivos iniciais para a criação da ADT, focando na característica de desejo: desempenho, flexibilidade ou qualquer outro parâmetro de interesse.

Os usuários do dos métodos da ADT são chamados de clientes, contudo seu acesso é limitado apenas a API da abstração, ou seja, a interface tem o papel de dar ao usuário o acesso aos métodos da ADT.

1. **ESTRUTURA E ORGANIZAÇÃO DOS DIRETÓRIOS**
   1. ARQUIVOS FONTESE ESTRUTURA DE DIRETÓRIOS

O trabalho segue a seguinte organização dos diretórios, na pasta raiz há 3 diretórios: as pastas *src* (arquivos fonte), *inc* (interfaces) e *obj* (objetos). Além delas, o diretório raiz contém o arquivo Makefile e a execução do programa com o nome de PTR-Entrega.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figura 1: Diretórios do trabalho

Entre os arquivos fonte da pasta *src* temos as implementações das ADTs (DString, Matrix e Integral) com o código de todas as suas operações no formato “*.c*”. O arquivo que contém a função main com os testes das ADTs também se encontra nesse diretório.

Na pasta *inc* se observa os arquivos “*.h*” do projeto, ou seja, as APIs ou interfaces. Neles há apenas as definições das operações implementadas nos arquivos fontes e estão separados em um arquivo para cada ADT (dstring.h, integrate.h e matrix.h).

O diretório *obj* armazena os objetos gerados nas compilações, da mesma forma que acontece com a interface é gerado um arquivo “*.o*” para cada ADT (dstring.o, integrate.o e matrix.o). Os arquivos objetos são as saídas reais geradas na fase de compilação, comumente escrita em formato binário ou linguagem de máquina, nesse caso eles são gerados por conta da compilação no *gcc* utilizando o parâmetro *-c* que significa “*just compile*” (apenas compile).

Por fim, no diretório raiz se encontra o arquivo Makefile responsável por simplificar o processo de compilar e gerar o arquivo executável que por sua vez também está nesse diretório. Dessa forma para se executar os testes das operações implementadas na pasta src e definidas na pasta inc basta executar o comando make no diretório raiz e em seguida abrir o arquivo executável gerado ./PTR-Entrega.

As ferramentas utilizadas no desenvolvimento do laboratório foram o editor de texto *VSCode*, a compilação em C foi feita em máquina virtual (WSL) contendo o Ubuntu 20.04 e *gcc* instalado pelo *build-essentials*.

* 1. MAKEFILE

Texto

Descrição gerada automaticamenteComo descrito anteriormente, o arquivo Makefile é muito utilizado para se automatizar o processo de compilação dos arquivos fontes e seus includes. Neles são escritos todos os parâmetros e argumentos desejados na hora da compilação além de se informar os diretórios dos códigos do programa. O Makefile utilizado foi o mesmo disponibilizado nos exemplos de ADTs desenvolvido pelo professor André Cavalcante retirando apenas as variáveis relacionadas a outras bibliotecas de um diretório *lib*. Note que executável de teste possui o mesmo nome do diretório raiz do programa.

Figura 2: Comandos gerado pelo Makefile.

1. **ADTS**
   1. DYNAMIC STRING

Esse tipo abstrato de dado tem como objetivo gerar *strings* com alocação dinâmica e é definida por um ponteiro de char que a ponta para a *string* e um número inteiro que indica quantos caracteres estão alocados no *buffer*. A *struct* que define a ADT bem como as definições das operações estão no arquivo *dstring.h.*

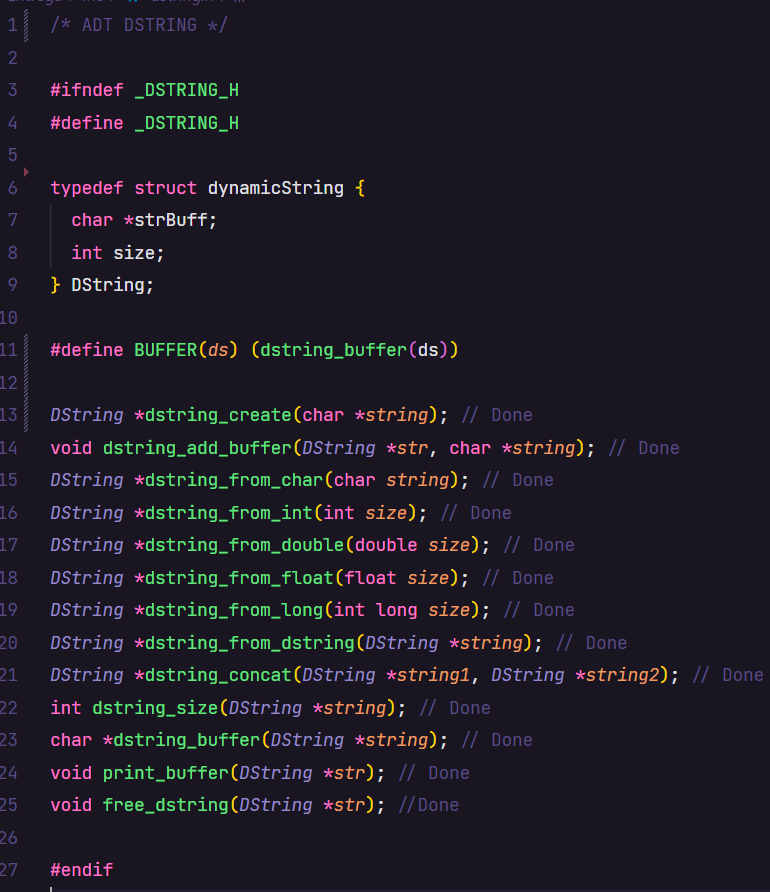


Figura 3: API dstring.h

As funções implementadas nessa ADT são as seguintes:

* ***dstring\_create:***O primeiro método desenvolvido foi criação de uma string dinâmica a partir de um ponteiro do tipo *char*. A função recebe como parâmetro um ponteiro do tipo *char* e o percorre para definir a quantidade de caracteres contidos nele. Em seguida cria uma instância *DString* e atualiza o seu *buffer* com o ponteiro passado e o seu tamanho com o descoberto anteriormente. Seu retorno é o ponteiro para a instância criada do tipo *DString*.

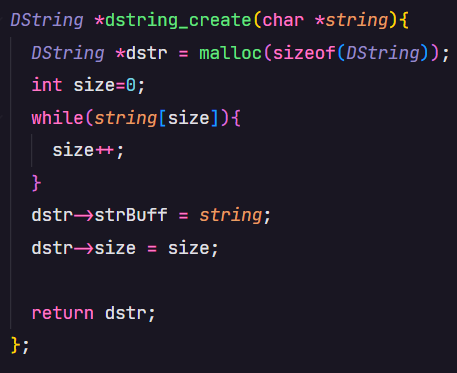


Figura 4: Implementação do *dstring\_create*

* ***dstring\_add\_buffer:***Esse método tem como função adicionar um *buffer* a uma *DString* com memória já alocada. Dessa forma ele recebe como parâmetro um ponteiro do tipo *DString* e um ponteiro do tipo *char*, em seguida copia os valores da *string* para o *buffer* da *DString* até que ocupe todo o espaço de memória alocado previamente ou até que os caracteres da *string* de entrada acabem. Como retorno temos o ponteiro para a *DString* contendo o novo *buffer.*

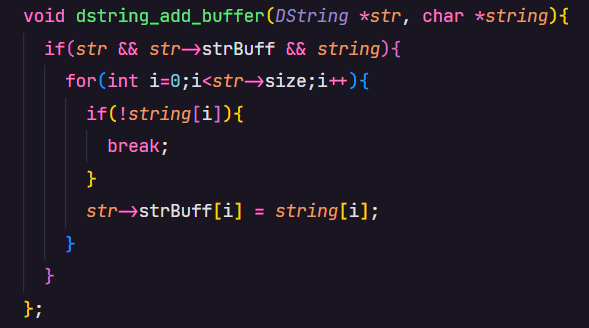
******

Figura 5: Implementação do *dstring\_add\_buffer*

* ***dstring\_size:*** Função que recebe o ponteiro de uma *DString* e retorna o valor salvo do seu tamanho alocado.

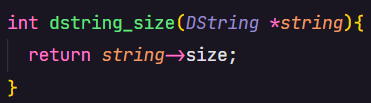
******

Figura 6: Implementação do *dstring\_size*

* ***dstring\_from\_int:*** Função que recebe um inteiro como parâmetro de entrada e cria uma instância do tipo *DString* com a memória alocada no *buffer* para a quantidade de caracteres informada na entrada. Retorna o ponteiro da *DString* criada.

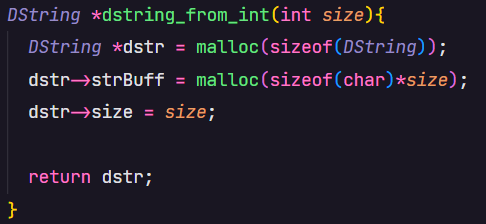
******

Figura 7: Implementação do *dstring\_from\_int*

* ***dstring\_from\_char:*** Função que recebe um *char* como parâmetro de entrada e cria uma instância do tipo *DString* com a memória alocada no *buffer* para esse *char* além de o copiar seu valor. Retorna o ponteiro da *DString* criada.

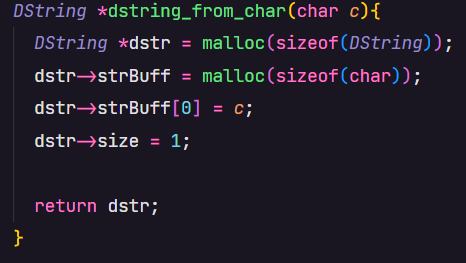
******

Figura 8: Implementação do *dstring\_from\_char*

* ***dstring\_from\_double:*** Função que recebe um ponto flutuante do tipo *double* como parâmetro de entrada e cria uma instância do tipo *DString* com a memória alocada no *buffer* para a quantidade de caracteres informada na entrada. O valor recebido é convertido em inteiro antes que a alocação de memória ocorra. Retorna o ponteiro da *DString* criada.

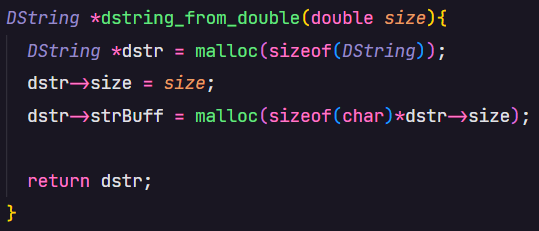
******

Figura 8: Implementação do *dstring\_from\_double*

* ***dstring\_from\_float:*** Função que recebe um ponto flutuante do tipo *float* como parâmetro de entrada e cria uma instância do tipo *DString* com a memória alocada no *buffer* para a quantidade de caracteres informada na entrada. O valor recebido é convertido em inteiro antes que a alocação de memória ocorra. Retorna o ponteiro da *DString* criada.

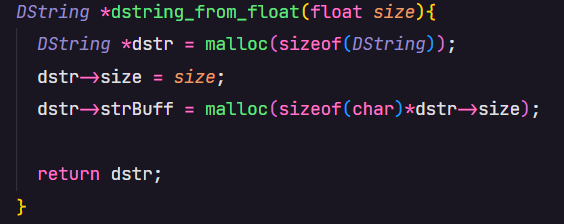
******

Figura 9: Implementação do *dstring\_from\_float*

* ***dstring\_from\_long:*** Função que recebe um inteiro do tipo *long* como parâmetro de entrada e cria uma instância do tipo *DString* com a memória alocada no *buffer* para a quantidade de caracteres informada na entrada. Retorna o ponteiro da *DString* criada.

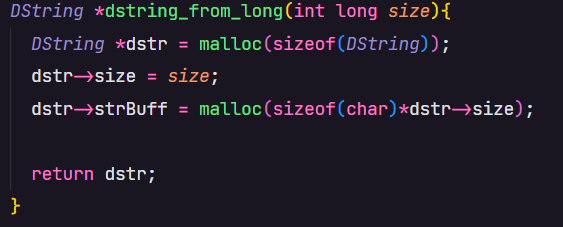
******

Figura 10: Implementação do *dstring\_from\_float*

* ***dstring\_from\_dstring:*** Esse método recebe uma *DString* como entrada e cria uma nova *DString* copiando o tamanho alocado e o *buffer*. Note que esse método copia os valores da *string* dinâmica de entrada. Seu retorno é o ponteiro da *DString* gerada.

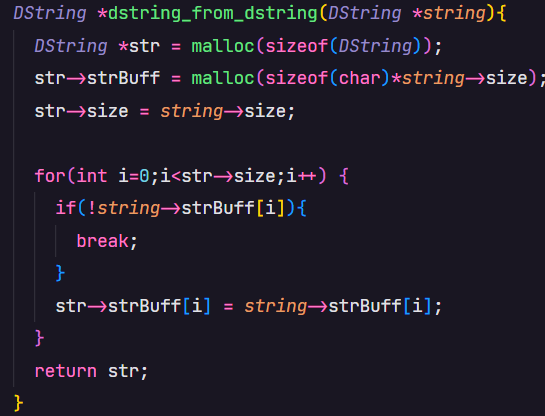
******

Figura 11: Implementação do *dstring\_from\_dstring*

* ***dstring\_concat:*** Esse operador tem como objetivo concatenar duas *DString*. Tem como parâmetros de entrada dois ponteiros do tipo *DString* e gera uma instância do mesmo tipo definindo como tamanho de caracteres de alocação a soma dos tamanhos contidos nas entradas. Para definir os valores do *buffer* da nova *DString* é realizada uma cópia dos caracteres contidos no *buffer* da primeira entrada e logo em seguida a cópia dos caracteres contidos no *buffer* da segunda entrada. O retorno da função é o ponteiro da *DString* gerado para a concatenação.

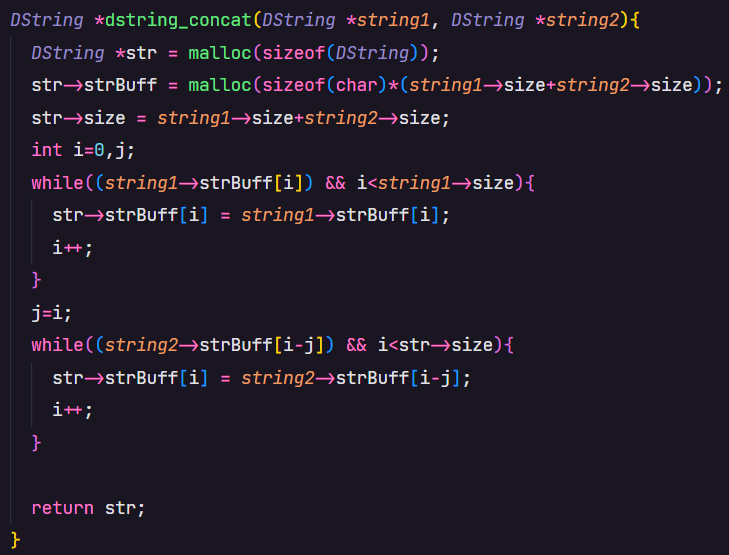
****

Figura 12: Implementação do *dstring\_concat*

* ***print\_buffer:*** Funçãoquerecebe um ponteiro do tipo *DString* e percorre todo o seu *buffer* imprimindo cada valor até que os caracteres terminem ou chegue no tamanho máximo alocado. Essa função não tem retorno.

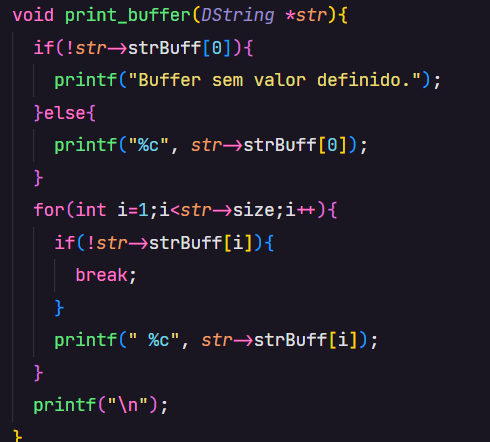
******

Figura 13: Implementação do *print\_buffer*

* ***dstring\_buffer:*** Operador que tem como função retornar o ponteiro *char* do *buffer* de uma instância do tipo *DString.* A entrada é o ponteiro dessa instância e o retorno é o ponteiro do *buffer.*

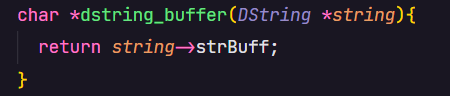
******

Figura 14: Implementação do *dstring\_buffer*

* ***free\_dstring:*** Função que limpa a memória do buffer de uma *DString* e zera o atributo de tamanho alocado. Seu parâmetro é o ponteiro para a *DString* a ser liberada. Não há retorno para esse método.

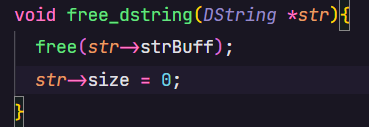
******

Figura 15: Implementação do *dstring\_free*

* 1. INTEGRAL

Essa ADT tem como objetivo de calcular a integral de funções em ponto flutuante. Sua definição é feita a partir de dois números em ponto flutuante que indicam o limite superior e inferior da integral além de um ponteiro para uma função que recebe *double* e retorna *double*. A interface presente no arquivo “*integrate.h”* é apresentada a seguir.

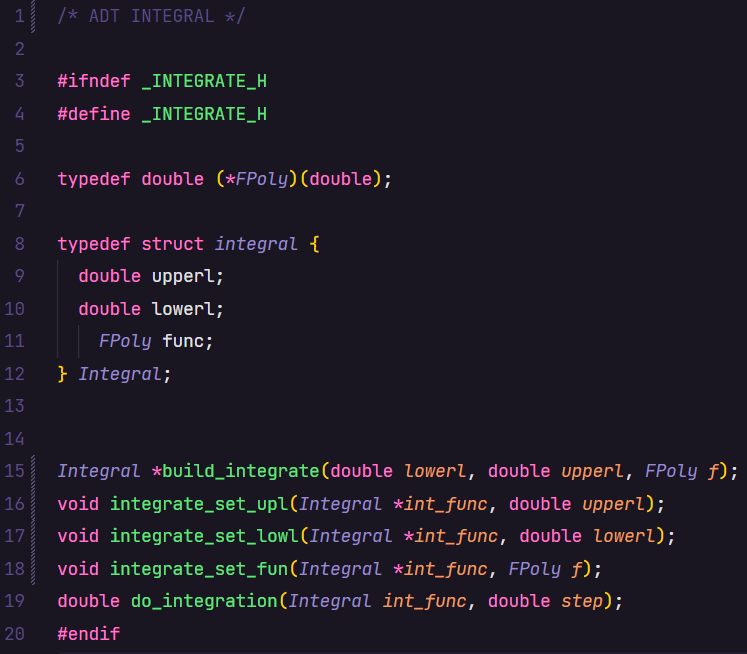


Figura 3: API integrate.h

As funções implementadas nessa ADT são as seguintes:

* ***build\_integrate:*** Essa função cria uma instância do tipo *Integral* a partir de dois parâmetros do tipo *double* que se referem aos limites superiores e inferiores da integral, além de um ponteiro da função a ser integrada. O retorno é o ponteiro para a instância *Integral* criada.
* ***integrate\_set\_upl:*** Método que recebe um ponteiro do tipo *Integral* e um número do tipo *double* e o define como limite superior da integral.
* ***integrate\_set\_lowl:*** Método que recebe um ponteiro do tipo *Integral* e um número do tipo *double* e o define como limite inferior da integral.
* ***integrate\_set\_fun:*** Método que recebe um ponteiro do tipo *Integral* e um ponteiro de uma função e o define como a função a ser integrada.
* ***do\_integration:*** A operação que realiza a integração do

1. **CONCLUSÃO**

As técnicas de controle são muito utilizadas em diversos sistemas da indústria de nosso dia a dia. Dessa forma o presente relatório buscou apresentar algumas dessas técnicas básicas como a representação computacional de um modelo bem como suas simulações em malha aberta e fechada. Além disso foram feitas algumas análises a respeito de como certos parâmetros afetam a dinâmica do projeto.

No primeiro experimento pode-se realizar a representação computacional por alguns métodos lineares e não-lineares bem como avaliar a precisão de alguma dessas aproximações. Para o modelo não-linear foram utilizadas as técnicas de aproximação de Euler e de Rugen-Kutta, bem como a simulação por diagrama de blocos do simulink, sendo que as duas últimas se mostraram mais precisas. No modelo linearizado a partir de um ponto de operação foi realizada uma simulação por espaço de estado e outra por função de transferência.

Ao se analisar a curva para um modelo linear e outra não linear, percebe-se uma diferença mínima em torno do ponto de operação que cresce quanto mais se afasta dele, ou seja, eles se comportam de maneira parecida até certo momento que o modelo não-linear passa a defasar em comparação ao linear.

Em seguida, foram feitas diversas simulações a fim de verificar a atuação de alguns parâmetros da planta na dinâmica do sistema. Ao variarmos as resistências da carga (*R*) e a capacitância do capacitor (*C*) encontramos uma relação proporcional, na qual tanto o máximo sobressinal quanto o tempo de acomodação aumentavam à medida que esses parâmetros subiam, ou seja, seu amortecimento diminuía. Também foi possível observar que para a capacitância à medida que aumentava, o sistema ficava mais lento. Para a potência o comportamento era o oposto, quanto maior o *PO* mais amortecido era o sistema.

O experimento 2 consistiu nas verificações das especificações da planta em malha aberta além da sua simulação em malha fechada com diversos tipos de controladores e sensores. Primeiramente encontrou-se as especificações de tempo de acomodação, tempo de subida, tempo de pico e máximo sobressinal a partir das fórmulas apresentadas na fundamentação teórica e comparamos com as informações extraídas diretamente da simulação. Os resultados obtidos foram muito próximos aos calculados para o tempo de acomodação, tempo de pico e máximo sobressinal, já o tempo de subida apresentou uma diferença bem acentuada, contudo isso já era esperado pois sua fórmula é descrita como apenas uma aproximação.

Em seguida foram feitas algumas simulações em malha fechada com determinados valores de *kp* que satisfaziam as condições de erro de 10%, 5%, 2% e 1%. Esses resultados mostraram os efeitos de um controlador P em malha fechada. Quanto maior era o ganho proporcional, maior era o sobressinal e sua frequência natural, enquanto o erro tendia a diminuir.

Logo após, foram realizadas simulações com controladores PI e PID variando os ganhos integral e o derivado. Para uma ação integral maior, tivemos resultados de uma maior frequência natural e menor acomodação, já quando se aumentava o *kD* era obtido uma maior oscilação e maior amortecimento, demorando muito menos para o sistema se estabilizar.

Por fim, um controlador PID foi projetado com especificações de desempenho pré-definidas, contudo o comportamento não foi satisfatório apresentando tanto um sobressinal muito alto, como um tempo de acomodação também elevado, acrescido da ocorrência de *undershooting*. Ao se aplicar a variação dos sensores, obtivemos uma resposta ainda mais diferente, deixando o sistema instável visto que ele se comportava como uma rampa sem regime estacionário.

Ao final das análises e das simulações, a compreensão sobre a modelagem e representação computacional de modelos foi acrescida consideravelmente, bem como o entendimento sobre as influências na dinâmica do sistema causadas pelos parâmetros da planta e pelas ações dos controladores. Foi possível observar as relações teóricas sendo comprovadas nas simulações, apesar de haver alguns casos em que o comportamento esperado não ocorreu. Assim, este trabalho se mostrou de extremo valor, tanto para o aluno quanto para o professor, podendo assim reforçar ainda mais o conhecimento de ambos os lados, em especial o do discente.

**5. REFERÊNCIAS**

www.ime.usp.br/~pf/estruturas-de-dados/

Last modified: Wed Sep 6 10:30:13 BRT 2017

Paulo Feofiloff

Departamento de Ciência da Computação

Instituto de Matemática e Estatística da USP

Presentation copyright 1995, The Benjamin/Cummings Publishing Company, For use with Data Structures and Other Objects by Michael Main and Walter Savitch.

Translation to portuguese by Prof. Maria Carolina Monard, ICMC-USP. Modifications for C++ language by Prof. José Augusto Baranauskas, FFCLRP-USP, 2005.

BESSA, Isaías V. Roteiro: Simulação de sistemas e realimentação. Manaus: Departamento de Elétrica - UFAM.

PEREIRA, Heverton. Aula 06 - Conversores CC/CC. Viçosa: Departamento de Engenharia Elétrica - UFV.

PETRY, Clóvis A. Introdução aos Conversores CC-CC. Florianópolis: UFSC, 2001.