Sumário

1) Introdução

O problema proposto para esse trabalho consiste em um processo industrial composto por um tanque cônico, onde a área da seção transversal vaira com a altura, conforme a seguinte figura:

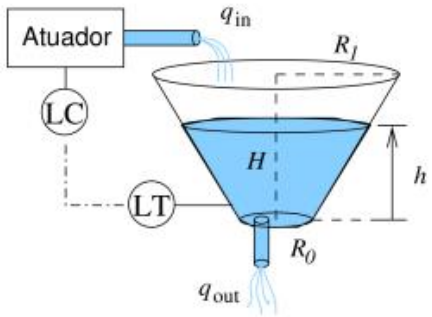

Figura 1: Processo industrial a ser simulado e controlado.

Esse processo contém alguns instrumentos que realizam importantes funções, como o transmissor de nível **LT** que fornece o nível do tanque *h(t)*. Além desse, existe um atuador que atua na entrada de líquido do tanque *qin(t)* e é controlado pelo controlador de nível **LC.** Por fim, a saída do tanque ocorre por meio de uma vazão não manipulada, expressa por:

$$qout(t) = Cv\sqrt{h}$$

Onde, *Cv é a constante de descarga da saída do tanque.*

A partir da modelagem física do problema, conseguiu-se encontrar uma a dinâmica não linear desse sistema, que é expressa pela seguinte equação:

$$\dot{h}(t) = \frac{-C_v\sqrt{h(t)}}{\pi[R_0 + \alpha h(t)]^2} + \frac{1}{\pi[R_0 + \alpha h(t)]^2}u(t)$$

Onde *u(t) = qin(t)* e $\alpha = \frac{R1-R0}{H}$.

Dessa forma, com as informações do processo e a equação dinâmica não linear do sistema, o objetivo deste trabalho foi simular esse processo industrial, através de um método de controle de nível e uma interface de interação com a planta.

Para isso, foram definidos os parâmetros do tanque da seguinte forma:

H (altura total do tanque) = 100m

R1 (raio superior) = 18m

R0 (raio inferior) = 10m

Cv (constante de descarga) = 1

A escolha desses parâmetros foi baseada nos valores encontrados para o metodo de integração Runge Kutta, pois com eles, a altura do tanque *h(t)* apresentou bons valores para a simulação dessa dinâmica. O valor de H (altura total do tanque) foi a única, previamente, escolhida, pois com esse valor seria possível trabalhar com porcentagem mais facilmente.

2) Metodologia

Para a realização deste trabalho, foi criado um programa que dispara 2 threads (*process_thread e softPLC_thread)* e 1 processo (*synoptic_process).* Além disso, algumas funções auxiliares foram desenvolvidads para simular todo o processo em estudo (*float dhdt, float RungeKutta e float Controlador_on_off)*.

a) Runge Kutta

Na primeira etapa do código, foi desenvolvida o método de integração do tipo Runge Kutta. Esse método requer apenas derivadas de primeira ordem e pode fornecer aproximações precisas com erros de truncamento de ordem h^2, h^3 e h^4. Foi escolhido o método de 4º ordem. As equações para esse método de integração são:

$$y_{i+1} = y_i + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right)$$
$$k_4 = f\left(x_i + h, y_i + hk_3\right)$$

A partir disso, duas funções foram desenvolvidas no código, a *float dhdt* e a *float rungeKutta.* A primeira delas, contém as atribuições dos parâmetros do processo escolhidos e o cálculo da derivada da altura h(t) em relação ao tempo. Ela recebe como parâmetros um valor de tempo (t), um valor para altura (h) e um valor para a vazão de entrada (VazaoDeEntrada), variáveis necessárias para o cálculo com a dinâmica do processo.

```
float dhdt(float t, float h, float VazaoDeEntrada)
{
    const float  pi = 3.14159265358979f;
    const float cv = 1.0; //Coeficiente de Descarga do tanque
    const float AlturaTanque = 100.0; //H
    const float RaioInferior = 10.0; //r0
    const float RaioSuperior = 18.0; //r1
    float alpha;
    alpha = (RaioSuperior-RaioInferior)/AlturaTanque;

    return((VazaoDeEntrada - (cv*sqrt(h)))/(pi*pow((RaioInferior+alpha*h),2)));
}
```

Figura 2 – Código da função *dhdt*

Dessa forma, ficou possível desenvolver a segunda função, *rungeKutta*,

que termina de realizar o método de integração escolhido. Nessa função são realizados os cálculos das equações descritas acima, utilizando a função já criada *dhdt*. Com isso, consegue-se encontrar uma aproximação para o valor da altura h(t) do tanque. Foram definidas 50 interações para encontrar a aproximação. Por fim, essa função recebe como parâmetro a altura inicial e o valor da vazão de entrada do tanque.

```
float rungeKutta(float h0, float VazaoDeEntrada)
{

    float k1, k2, k3, k4;
    float step = 0.02;
    float hn = h0;
    float tn = 0.0;

    // Interação acontece pelo tempo definido

    for (int i=1; i<=50; i++)
    {
        // Aplica Runge Kuntta para encontrar o proximo valor de h

        k1 = step*dhdt(tn, hn, VazaoDeEntrada);
        k2 = step*dhdt(tn + 0.5*step, hn + 0.5*k1, VazaoDeEntrada);
        k3 = step*dhdt(tn + 0.5*step, hn + 0.5*k2, VazaoDeEntrada);
        k4 = step*dhdt(tn + step, hn + k3,VazaoDeEntrada);

        // Atualiza o valor de h

        hn = hn + (1.0/6.0)*(k1 + 2*k2 + 2*k3 + k4);;

    tn = tn + step;
    }

    if (hn<0.0)
    {
        return 0.0;
    }
    else
    {
        return hn;
    }

}
```

Figura 3 – Código da função *rungeKutta*

b) Controlador On Off

Em seguida, foi desenvolvida a função *Controlador_on_off*. Essa função realiza o controle do processo do tanque cônico. Entre diversos métodos de controle, esse foi escolhido pela simplicidade e por atender as necessidades do problema. Seu funcionamento se baseia numa vazão de entrada de valor fixo, fazendo com que o atuador consiga apenas desligar (off) ou ligar (on) a entrada. Dessa forma, utilizando o valor de referência passado como parâmetro e valor atual da altura do tanque h(t), realiza-se o controle do processo.

Uma medida de segurança foi colocada para evitar que o fluido do processo transborde. Esse intertravamento consiste no bloqueio da vazão de entrada quando o tanque estiver com o nível maior ou igual a 98%. Mesmo se a referência for maior que esse valor, o controlador limita, por segurança, o atuador.

```
float Controlador_on_off(float referencia, float h0)
{

    float Vazao_Maxima_Entrada = 30.0;
    float Vazao_entrada;
    if(h0<= 98 )   // Limite de segurança
    {
        if(h0>referencia)
        {
            Vazao_entrada = 0;
        }
        else
        {
            Vazao_entrada = Vazao_Maxima_Entrada;
        }
        return Vazao_entrada;
    }
    else
    {
        Vazao_entrada = 0;
        return Vazao_entrada;
    }
}
```

Figura 4 – Código da função *Controlador_on_off*

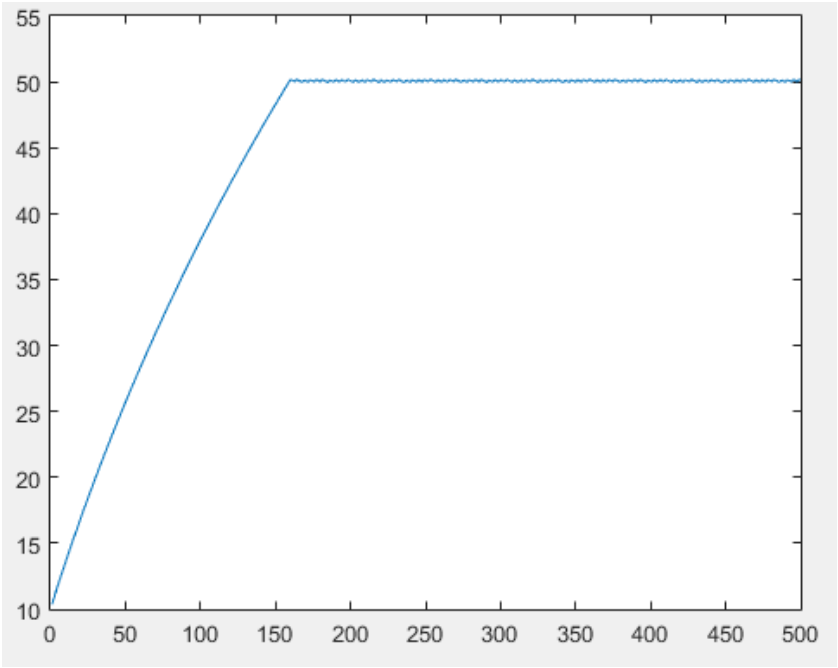

Figura 5 – Valor da altura do tanque a cada interação com ação do controlador. Altura inicial = 10 e Referência = 50

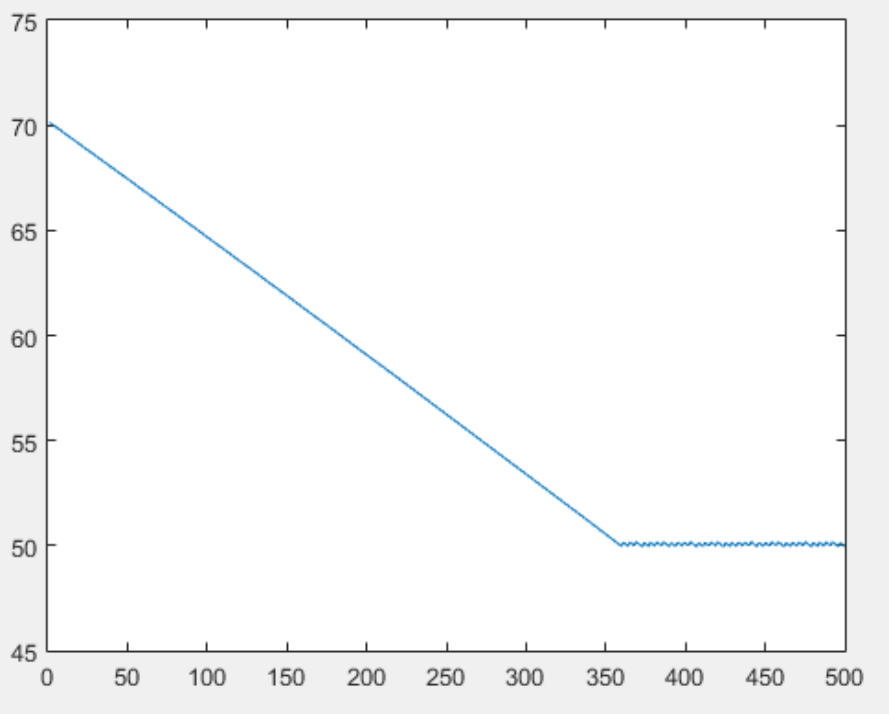

Figura 6 – Valor da altura do tanque a cada interação com ação do controlador. Altura inicial = 70 e Referência = 50

c) Process_thread

A partir das duas funções desenvolvidas, já foi possível disparar as threads do processo. A primeira thread realiza a simulação da equação dinâmica do tanque, utilizando a função já criada *rungeKutta*. O período da simulação escolhido foi de 100ms e foi utilizado um mutex para proteger as variáveis globais de vazão de entrada e altura do tanque. Além disso, nessa thread tem a abertura do arquivo de texto “historiador” para o armazenamento dos dados do processo.

```
void process_thread()
{
    int espera;
    int i=0;
    ofstream out;
    out.open("historiador.txt");
    while(true)
    {
        try
        {
            std::lock_guard<std::mutex> lock(qin_e_altura_mutex);
            Altura_do_liquido = rungeKutta(Altura_do_liquido, qin);
            if(i==20)
            {
                //cout << "h= " << Altura_do_liquido << " m" << endl;
                //cout << "qin= " << qin << " m3/s" << endl;
                out <<"h="<< Altura_do_liquido<< "/" << "qin= "<<qin<<"/"<<"href= "<<referencia<<endl;
                i=0;
            }
            espera = 0;
            i++;
        }
        catch (std::logic_error&)
        {
```

```
            espera = 1;
        }

        if(espera==0)
        {
            std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }

    }

}
```

Figura 7 – Código da função *process_thread*

d) SoftPLC_thread

A próxima função desenvolvida contém a lei de controle implementada no processo. Além disso, essa thread faz comunicação via socket TCP/IP com o *synoptic_process,* logo ela possui todas as etapas para a realização dessas trocas de mensagens via socket. Dessa forma, essa thread envia os valores da altura e das vazões do processo, contendo o sinal do controlador para o atuador. Além disso, o perído de execução é igual a metade da frequência da thread anterior, ou seja, o dobro do perído. Com isso, o valor ficou de 200ms. Por fim, como feito anteriormente, as variáveis globais foram protegidas com mutex e foi realizado o teste de erro com as funções *try e catch.*

```
// Creating socket file descriptor
server_fd = socket(AF_INET, SOCK_STREAM, 0);

// Forcefully attaching socket to the port 8080
setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT | SO_RCVTIMEO, (const char*)&tv, sizeof(tv));


address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

// Forcefully attaching socket to the port 8080
bind(server_fd, (struct sockaddr *)&address, sizeof(address));

listen(server_fd, 3);

new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
```

Figura 8 – Código da função *softPLC_thread* – Comunicação via socket

```
while(true)
{
    try
    {
        std::lock_guard<std::mutex> lock(qin_e_altura_mutex);
        qin = Controlador_on_off(referencia, Altura_do_liquido);
        altura.str("");
        vazao.str("");
        altura << Altura_do_liquido;
        vazao << qin;
        valor_atura = altura.str();
        valor_qin = vazao.str();
        msg = "h= " + valor_atura + "m"+ " qin= " + valor_qin + " m3/s";
        espera = 0;
    }
    catch (std::logic_error&)espera = 1;

    if(espera==0)
    {
        int n = msg.length();
        char mensagem[n+1];
        strcpy(mensagem, msg.c_str());
        std::this_thread::sleep_for(std::chrono::milliseconds(200));
        valread = read( new_socket, buffer, 1024);
        //printf("%s\n",buffer );
        send(new_socket, mensagem, strlen(mensagem), 0 );
    }
```

Figura 9 – Código da função *softPLC_thread* – Troca de mensagens e ação de controle

e) Synoptic_Process

Por fim, a última função desenvolvida foi o processo do sinótico. Ele emula um sistema supervisório de para a teleoperação do controle do sistema de tanque. Para isso, como dito acima, ela realiza comunicação via socket TCP/IP com a thread *softPLC_thread.* Com isso, ele recebe os valores desejados para a exibição na tela de operação. Além disso, através dele, o usuário consegue enviar um valor de altura referência para o controlador realizar a ação de controle.

```
void synoptic_process()
{
    std::this_thread::sleep_for(std::chrono::seconds(1));
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    int i=0;
    string Altura_ref = "8";
    ofstream out;
    out.open("historiador.txt"); // Aquivo no qual as informações são registradas
    auto future = std::async(std::launch::async, GetLineFromCin);

    //Cria socket para comunicação

    sock = socket(AF_INET, SOCK_STREAM, 0);

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);


    connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
```

```
    while(true)
    {
        send(sock, href, strlen(href), 0 );
        valread = read( sock, buffer, 1024);
        if(i==10)
        {
            int n = strlen(buffer);
            buffer[n-1] = ' ';
            buffer[n] = '\0';
            cout << buffer << endl;
            i=0;
        }
        i++;
        //std::this_thread::sleep_for(std::chrono::seconds(3));
    }
}
```

Figura 10 – Código da função *synoptic_Process*

3) Testes

Foram realizados dois testes para a resolução do problema. No primeiro, a altura de referência para o tanque foi de 8m e a altura inicial foi de 0m. Além disso, a vazão de entrada fixa foi de 30 m³/s.

```
h= 0.1889m | qin= 30 m3/s | qout= 0.4346 m3/s   | href= 8m
h= 0.3756m | qin= 30 m3/s | qout= 0.6129 m3/s   | href= 8m
h= 0.5609m | qin= 30 m3/s | qout= 0.7489 m3/s   | href= 8m
h= 0.7448m | qin= 30 m3/s | qout= 0.863 m3/s    | href= 8m
h= 0.9275m | qin= 30 m3/s | qout= 0.9631 m3/s   | href= 8m
h= 1.109m | qin= 30 m3/s | qout= 1.053 m3/s     | href= 8m
h= 1.29m | qin= 30 m3/s | qout= 1.136 m3/s      | href= 8m
h= 1.469m | qin= 30 m3/s | qout= 1.212 m3/s     | href= 8m
h= 1.648m | qin= 30 m3/s | qout= 1.284 m3/s     | href= 8m
h= 1.825m | qin= 30 m3/s | qout= 1.351 m3/s     | href= 8m
h= 2.002m | qin= 30 m3/s | qout= 1.415 m3/s     | href= 8m
h= 2.178m | qin= 30 m3/s | qout= 1.476 m3/s     | href= 8m
h= 2.353m | qin= 30 m3/s | qout= 1.534 m3/s     | href= 8m
h= 2.527m | qin= 30 m3/s | qout= 1.59 m3/s      | href= 8m
h= 2.7m | qin= 30 m3/s | qout= 1.643 m3/s s     | href= 8m
h= 2.873m | qin= 30 m3/s | qout= 1.695 m3/s     | href= 8m
h= 3.045m | qin= 30 m3/s | qout= 1.745 m3/s     | href= 8m
h= 3.216m | qin= 30 m3/s | qout= 1.793 m3/s     | href= 8m
h= 3.386m | qin= 30 m3/s | qout= 1.84 m3/s      | href= 8m
h= 3.556m | qin= 30 m3/s | qout= 1.886 m3/s     | href= 8m
h= 3.725m | qin= 30 m3/s | qout= 1.93 m3/s      | href= 8m
h= 3.893m | qin= 30 m3/s | qout= 1.973 m3/s     | href= 8m
h= 4.06m | qin= 30 m3/s | qout= 2.015 m3/s      | href= 8m
h= 4.227m | qin= 30 m3/s | qout= 2.056 m3/s     | href= 8m
```

Figura 11 – Historiador.txt – Teste 1

E o segundo teste, a altura de referência se manteve em m e a altura inicial foi de 15m. A vazão de entrada também continuou em 30m³/s.

```
h= 14.97m | qin= 0 m3/s | qout= 3.869 m3/s  | href= 8m
h= 14.95m | qin= 0 m3/s | qout= 3.867 m3/s  | href= 8m
h= 14.93m | qin= 0 m3/s | qout= 3.864 m3/s  | href= 8m
h= 14.91m | qin= 0 m3/s | qout= 3.862 m3/s  | href= 8m
h= 14.89m | qin= 0 m3/s | qout= 3.859 m3/s  | href= 8m
h= 14.87m | qin= 0 m3/s | qout= 3.857 m3/s  | href= 8m
h= 14.85m | qin= 0 m3/s | qout= 3.854 m3/s  | href= 8m
h= 14.83m | qin= 0 m3/s | qout= 3.852 m3/s  | href= 8m
h= 14.81m | qin= 0 m3/s | qout= 3.849 m3/s  | href= 8m
h= 14.8m | qin= 0 m3/s | qout= 3.846 m3/s   | href= 8m
h= 14.78m | qin= 0 m3/s | qout= 3.844 m3/s  | href= 8m
h= 14.76m | qin= 0 m3/s | qout= 3.841 m3/s  | href= 8m
h= 14.74m | qin= 0 m3/s | qout= 3.839 m3/s  | href= 8m
h= 14.72m | qin= 0 m3/s | qout= 3.836 m3/s  | href= 8m
h= 14.7m | qin= 0 m3/s | qout= 3.834 m3/s   | href= 8m
h= 14.68m | qin= 0 m3/s | qout= 3.831 m3/s  | href= 8m
h= 14.66m | qin= 0 m3/s | qout= 3.829 m3/s  | href= 8m
h= 14.64m | qin= 0 m3/s | qout= 3.826 m3/s  | href= 8m
h= 14.62m | qin= 0 m3/s | qout= 3.824 m3/s  | href= 8m
h= 14.6m | qin= 0 m3/s | qout= 3.821 m3/s   | href= 8m
h= 14.58m | qin= 0 m3/s | qout= 3.819 m3/s  | href= 8m
h= 14.56m | qin= 0 m3/s | qout= 3.816 m3/s  | href= 8m
h= 14.54m | qin= 0 m3/s | qout= 3.814 m3/s  | href= 8m
h= 14.52m | qin= 0 m3/s | qout= 3.811 m3/s  | href= 8m
h= 14.5m | qin= 0 m3/s | qout= 3.808 m3/s   | href= 8m
h= 14.48m | qin= 0 m3/s | qout= 3.806 m3/s  | href= 8m
h= 14.47m | qin= 0 m3/s | qout= 3.803 m3/s  | href= 8m
```

Figura 12 – Historiador.txt – Teste 2

Por fim, um imagem da tela de sinótico onde os valores requiridos pelo usuário são exibidos na tela e é possível ver que o usuário digitou outro valor de referência para o controlador.

Figura 13 – Tela do sinótico

4) Conclusão

Com a execução deste trabalho foi possível a compreensão e a familiarização com muitos dos conceitos abordados durante as aulas teóricas. Mais do que isso, foi possível ter um contato mais aprofundado com o paradigma da automação em tempo real, permitindo uma expansão das habilidades e ferramentas de desenvolvimento dos membros.

Além disso, conseguiu-se atingir os objetivos esperados, onde foi feita a simulação de um processo industrial com uma ação de controle e a apresentação através de uma interface de interação. Dessa forma, os conceitos de processos, threads e diretivas de sincronização foram implementados com êxito.