

Gerador de Sistemas para Controle

Bruna Medeiros da Silva¹, 16/0048711, Felipe Lima Alcântara², 16/0027918

^{1,2}Programa de Engenharia Eletrônica, Faculdade Gama - Universidade de Brasília, Brasil

Resumo—O Gerador de Sistemas é um projeto voltado para educação, mais especificamente para auxiliar na aplicação prática de Sistemas de Controle para engenheiros, sua ideia surgiu pela falta de equipamentos para aplicar, em ambiente laboratorial, o que é aprendido na disciplina teórica, na Faculdade do Gama da Universidade de Brasília (FGA-UnB). Devido a essa carência, atualmente o meio de aplicação é a ferramenta do software MATLAB denominada Simulink, que realiza a construção em blocos e a simulação do sistema de controle. O projeto busca, através do microcontrolador Raspberry Pi, gerar o sistema definido por meio do modelo matemático que o descreve e, com o sistema gerado, fazer com que o aluno consiga inserir um controlador analógico no mesmo, buscando realizar o controle do sistema proposto.

Index Terms—Raspberry Pi, controle, embarcados, microcontroladores, eletrônica

I. INTRODUÇÃO

Sistemas de controle estão presentes em inúmeras aplicações na sociedade moderna: Chuveiros eletrônicos, geladeira, freio ABS, controle de estabilidade, controle de altitude de satélites, controle de órbita, placas fotovoltaicas que seguem a posição do Sol, ar condicionados, entre outros.

O conceito sistema de controle para a Engenharia é definido como um conjunto de subsistemas e processos projetados com o objetivo de obter a saída desejada com um desempenho estabelecido por meio da manipulação da entrada, com pouca ou nenhuma intervenção humana.[1]

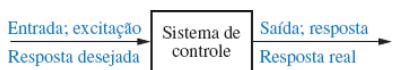


Figura 1. Exemplo simplificado de um Sistema de Controle.

Quando se trata do projeto de um sistema de controle, duas medidas de desempenho são evidenciadas: a resposta transitória e o erro em regime permanente. Por exemplo: um elevador deve subir e descer os andares de forma que não perturbe o passageiro e ao mesmo tempo não demore muito tempo para chegar no andar desejado (resposta transitória e tempo de subida), enquanto que

ao chegar no andar desejado, o elevador deve ainda estar nivelado com o piso o suficiente para não ter nenhum potencial de ser nocivo ao passageiro (erro em regime).

Algumas definições:[2]

- **Plantas:** Conjunto de componentes que agem de maneira integrada para a realização de uma determinada tarefa. É aquilo que deve ser controlado.
- **Variável controlada:** É a grandeza ou a condição aferida e controlada.
- **Sinal de controle:** É a grandeza modificada pelo controlador, buscando modificar o valor da variável controlada.

O sistema da Figura 1, é denominado de malha aberta, sua construção é mais simples, mais barata e mais conveniente para sistemas com saídas de difícil acesso, se comparado ao sistema de malha fechada, Figura 2. Já esse, por ter realimentação, consegue suprir as desvantagens de um sistema de malha aberta, sendo viável para sistemas instáveis e compensando o efeito de perturbações com a comparação entre saída e o sinal de referência.

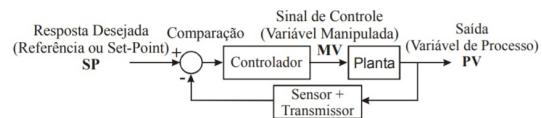


Figura 2. Diagrama de Blocos de um sistema em malha fechada.

Na Universidade de Brasília, UnB, campus Faculdade do Gama, FGA, a disciplina de Sistemas de Controle é ministrada para os cursos de Engenharia Aeroespacial e Eletrônica, com o objetivo de propiciar as primeiras ferramentas para a análise e síntese de sistemas de controle automáticos.

II. JUSTIFICATIVA

Dentro das universidades, sejam federais ou particulares, e principalmente dentro do ramo de Engenharia, é muito comum ouvirmos reclamações e críticas dos alunos relacionadas à necessidade de colocar em prática o que é aprendido nas matérias teóricas.

Além disso, aulas práticas possuem algumas características pouco exploradas ou ainda inexistentes em aulas teóricas, tais como a percepção das diferenças entre simulação e aplicação real e a capacidade de despertar uma habilidade criativa e de gerar conhecimentos práticos e aplicáveis à indústria, o que, por sua vez, se torna cada vez mais necessário no mercado de trabalho.[3]

Atualmente, na FGA, foi estendido o número de créditos da disciplina de Sistemas de Controle, com o intuito de suprir a necessidade da prática por parte dos alunos na disciplina. Entretanto, ainda não há nenhum equipamento apropriado para que os mesmos possam aplicar a matéria, fazendo-os visualizar o conteúdo apenas por meio de simulações, gráficos e equações matemáticas.

Usualmente, as atividades práticas desta disciplina buscam que o aluno desenvolva controladores para plantas já existentes, buscando analisar a diferença de desempenho ao adicionar controladores, alterando-se os coeficientes e tipo de controlador ou simplesmente fechando a malha do sistema, por exemplo.

A ementa da disciplina de Controle Dinâmico, que por sua vez é teórica e bastante parecida com a ementa de Sistemas de Controle ministrada na UnB campus Darcy Ribeiro ministra os seguintes conteúdos:

- Implementação de Controladores Analógicos: Realização de controladores analógicos utilizando amplificadores operacionais.
- Controle Dinâmico de Sistemas Físicos em laboratório: Controle de processos físicos por compensação dinâmica com controladores projetados no domínio-s, no domínio-w. Controladores PID. Controle de processos físicos por realimentação de estado.
- Métodos Computacionais em Controle Dinâmico: Aplicação de métodos computacionais no controle dinâmico de sistemas lineares contínuos no tempo.
- Experiências Demonstrativas: Servomecanismo Posicionado com Controle Lógico e Sensor Óptico/Digital. Automação de Testes e Medidas no Laboratório de Controle Servomecanismos Especiais.

Pensando nisso, será desenvolvido um produto capaz de modelar uma onda de saída de acordo com a função de transferência colocada pelo aluno/professor na entrada do sistema. Esse equipamento possibilitará a quem o utilizar testar seus conhecimentos de sistemas de controle, tais como realimentações, sistemas de malha fechada, projeto de controladores, estabilidade de sistemas, ganho, efeito de polos e zeros, resposta no domínio do tempo e suas características e muito mais.

III. OBJETIVO

Proporcionar aos alunos que estarão cursando a disciplina de Sistemas de Controle ou equivalentes uma interação real com o ambiente da matéria, visto apenas teórica e/ou graficamente, e com as ferramentas estudadas ao longo do curso, capacitando os alunos a atuarem ativamente no desenvolvimento de suas próprias habilidades e afinidade com a matéria.

Para isso será projetado um gerador de Sistemas a serem controlados, de tal forma que o professor ministrante da disciplina consiga inserir o modelo matemático do sistema no gerador e os alunos, ao construírem o controlador adequado para o sistema criado, consigam observar e entender o controle que estará sendo realizado, além de entender como isso realmente pode ser feito na prática.

IV. METODOLOGIA

O projeto será dividido em pontos de controle, com o objetivo de analisar e marcar o desenvolvimento do mesmo. Serão 4 pontos de controle, cujos são definidos da seguinte forma:

- **PC1:** proposta do projeto (justificativa, objetivos, requisitos, benefícios, revisão bibliográfica).
- **PC2:** protótipo funcional do projeto, utilizando as ferramentas mais básicas da placa de desenvolvimento, bibliotecas prontas etc.
- **PC3:** refinamento do protótipo, acrescentando recursos básicos de sistema (múltiplos processos e threads, pipes, sinais, semáforos, MUTEX etc.).
- **PC4:** refinamento do protótipo, acrescentando recursos de Linux em tempo real.

Cada ponto de controle terá um relatório escrito para o mesmo, de forma a agregar o anterior e a evolução do projeto.

Além disso, na agenda dos membros do projeto serão dedicadas um mínimo de quatro horas semanais para o desenvolvimento. O projeto terá parte de desenvolvimento em *Software* e parte em *Hardware*.

1) *Software*: A funcionalidade básica de que o *software* deverá ser capaz, é de receber uma função de transferência para o sistema e a partir desta gerar a forma de onda resultante, com a referência como entrada.

Com o sistema de controle em funcionamento, o software deverá identificar a entrada de realimentação e comparar com a referência (entrada original) enviando o sinal resultante para o controlador analógico, que será projetado pelo aluno.

Com o decorrer do projeto pode-se adicionar algumas funcionalidades no software, tais como a criação de um servidor, para inserir uma função de transferência em

várias Raspberry's ao mesmo tempo. Integração com o *MatLab* e outras funções também são extremamente vi-sadas, porém estes incrementos só serão realizados caso a função básica tenha sido efetivada de forma correta e com um nível satisfatório de eficácia e qualidade de produto.

2) *Hardware*: O projeto do *hardware* deverá conter pelo menos um conversor A/D e um D/A para que haja a transição de dados da Raspberry para o controle ana-lógico e da saída do controle analógico para Raspberry.

O projeto do *hardware* deverá ser acessível, para que o aprendizado de uso seja rápido. Ou seja, o hardware será construído visando uma interface amigável e intuitiva para o usuário, mas que também forneça a ele ferramen-tas e funcionalidades que supram suas necessidades de experimentação.

O desenvolvimento do *hardware* dependerá da efetivação do *software* e por este motivo não tem muitos exigências nesta etapa inicial do projeto.

V. REQUISITOS

Dado o fato de que o projeto será desenvolvido para um fim específico e um público-alvo bem definido, se torna necessária uma determinação de requisitos clara e objetiva. Sendo assim, podemos dividir os nossos requisitos nos seguintes grupos:

A. Requisitos materiais e custos

- Dispositivo de entrada (para que o usuário possa configurar o sistema conforme necessidade, como um teclado, por exemplo). **Preço:** R\$ 40,00;
- Raspberry Pi 3 Model B. **Preço:** R\$ 250,00;
- Conversor Analógico para Digital: *PCF8591P*. **Preço:** R\$ 40,00;

B. Requisitos técnicos

1) *Conhecimento em Sistemas de Controle*: Para re-alizar a aplicação será necessário conhecimento de sis-temas de controle, para que se realize testes de imple-mentação e que a Raspberry realize a função do sistema de forma coerente.

2) *Programação em C*: Dado o foco da disciplina é realizar um projeto utilizando, a Raspberry Pi, se torna imprescindível que os projetistas tenham uma habilidade e domínio sobre a linguagem C, que é a linguagem utili-zada para construir os códigos para a Raspberry, apesar de seu sistema funcionar em um terminal equivalente ao terminal do Linux.

3) *Habilidades em eletrotécnica*: Para a construção da parte estrutural, se faz necessário o uso de protoboards, jumpers, PCBs, entre outros componentes estru-turais. Assim, torna-se necessário habilidade em soldagem e manuseio de destes componentes.

4) *Formatação dos documentos*: A elaboração e ma-nutenção dos documentos produzidos no projeto deverá utilizar *LaTeX* de forma que a apresentação das infor-mações fique organizada, assim como, representará as instruções para a construção do protótipo.

VI. BENEFÍCIOS

Os principais benefícios na realização deste projeto estarão diretamente ligados à comunidade acadêmica da FGA, visto que trará uma experiência antes não disponível para os alunos e também aos professores, que poderão aplicar sua matéria de forma muito mais didática e compreensiva, gerando, possivelmente, um maior interesse pela área.

Pensando nisso, o projeto trará ainda uma alternativa com um equilíbrio entre custos e benefícios/funcionali-dades para a melhoria da formação prática e habilidade experimental dos alunos. Tudo isso contribuirá ainda para a formação de profissionais mais capacitados e preparados para o mercado de trabalho no futuro.

Além disso, os principais beneficiados com a realiza-ção do projeto serão os próprios projetistas, visto que lidarão com problemas e *trade-off* que os capacitarão cada vez mais para o mercado de trabalho e lhes entre-gará uma maior experiência e segurança para adentrar no mercado, além de capacitá-los a produzirem e conduzi-rem projetos de forma mais elaborada e eficiente dentro da própria Universidade.

VII. REVISÃO BIBLIOGRÁFICA

Industrialmente as simulações de sistemas de con-trole embarcados são realizadas utilizando a técnica *Hardware-in-the-loop*(HIL), que tem como objetivo de-senvolver e simular sistemas embarcados de grande complexidade em tempo real.[4]

O HIL serve para reduzir tempos, custos e esforços em concepção e projeto de sistemas dinâmicos, sendo bastante eficaz quando a representação matemática da planta, simulação de planta, é adicionada na plataforma de teste. O sistema embarcado a ser testado interage com esta simulação de planta, tudo ocorrendo em *software*.

A desvantagem deste sistema é a não aplicação e criação de controle analógico e o preço das plataformas modulares, uma controlador PXI da National Instruments custa aproximadamente R\$ 6500,00.

Marcos M. trás uma ideia de laboratório completamente analógico, onde alunos projetariam um controlador PID, porém as plantas ficariam limitadas as plantas geradas por circuitos elétricos, assim não havendo muita aplicação para Aeroespacial, já o projeto proposto aqui poderá ser qualquer tipo de planta a ser modulada matematicamente pelo aluno, para que seja feito o controlador.

Existe também a possibilidade de construção de plantas, Marco Melo demonstra exemplos de plantas criadas para a Disciplina de Controle Digital, [5] porém com uma turma de 45 alunos, como é a oferta para a disciplina de Sistemas de Controle, deveria-se ter um numero grande de plantas para contato do aluno com o sistema, encarecendo a aplicação.

As aplicações encontradas utilizando a Raspberry Pi em sistemas de controle são feitas com o objetivo de desenvolver apenas controladores, não surprendo a necessidade do laboratório.

VIII. DESENVOLVIMENTO

A. Componentes Principais

1) *Raspberry Pi3 B+*: Sendo a versão mais recente dos dispositivos da *Raspberry Pi Foundation*, a Raspberry Pi3 B+, as maiores diferenças estão na conectividade Wi-Fi que agora existe a possibilidade de conexão com frequências de 2,4 GHz e 5 GHz, também a frequência do processador que passou de 1,2 GHz (Raspberry Pi 3) para 1,4 GHz.[6]

Abaixo pode-se conferir a lista de especificações:

- Processador Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC 1.4GHz;
- 1GB LPDDR2 SDRAM;
- 2.4GHz e 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE;
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps);
- 40-pin GPIO header;
- Conector de interface HDMI;
- 4 portas USB 2.0;
- Conector CSI para câmera Raspberry Pi;
- Conector DSI para *display* capacitivo Raspberry Pi;
- Saída de vídeo composto e áudio através do plug P4;
- Slot cartão micro SD para carregamento do sistema operacional e armazenamento de arquivos;
- Entrada de fonte DC micro USB 5V/2.5A;
- Suporte a *Power-over-Ethernet* (PoE) (requer HAT PoE separadamente);

Com os pinos GPIO, pode-se realizar envio e recebimento de dados digitais, comunicação *I2C*, envio e

recebimento serial, entre outras possibilidades, a função de cada pino esta na Figura 3.

Um porém é a falta de conversores analógicos/digitais internos na placa, sendo necessário para o projeto a compra de *CI's* ou módulos para esta conversão.

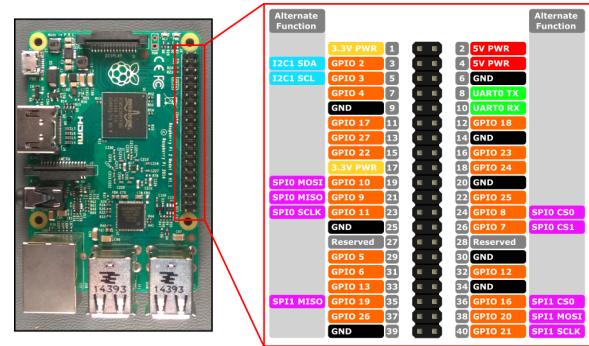


Figura 3. Pinagem Raspberry Pi +.

2) *PCF8591 - Conversor A/D e D/A de 8 bits*: Como a Raspberry Pi não tem em seus pinos um conversor analógico para digital ou digital para analógico, se fez necessário o uso de conversores externos, o escolhido foi o *CI PCF8591*, que é tanto um conversor A/D como D/A.[7]

O tipo de comunicação é *I2C* e uma das facilidades é já existir bibliotecas criadas para utilização deste *CI*. Suas portas são dispostas de acordo com a Figura 4.

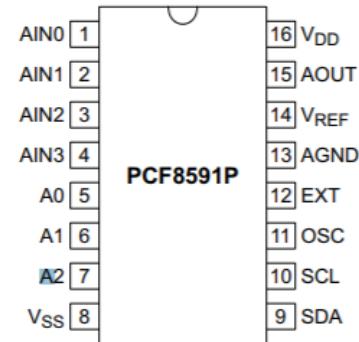


Figura 4. Configuração de pinos do PCF8591P.

B. Ponto de Controle 2

Para este ponto de controle buscou-se conhecer os aspectos básicos da placa Raspberry Pi e testar algumas funções e bibliotecas da linguagem C. O código de conversão Analógico-Digital está no Anexo H.8-A, Digital-Analógico está no Anexo H.8-B e para a geração dos

impulsos, segue o Anexo H.8-C. Segue alguns resultados obtidos com os códigos.

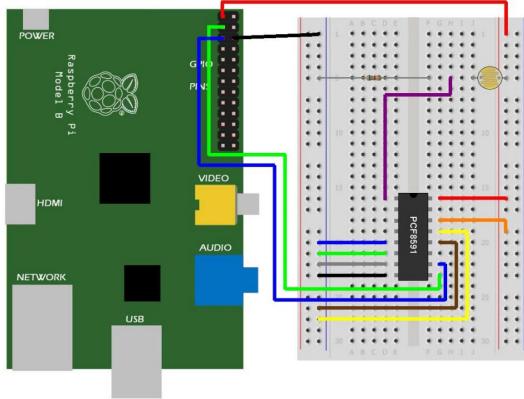


Figura 5. Esquemático do circuito usado no conversor A/D.

Na figura 5 pode ser visualizado o esquemático do circuito para a realização do circuito ADC (*Analog-Digital Converter*). Para isso, as entradas A0, A1 e A2, que são as portas para conexão a passagem do endereço de um hardware periférico (ou escravo) (*hardware slave address*, pelo *datasheet* do CI) foram aterradas, assim como a entrada para o oscilador e o AGND; As entradas SCL e SDA foram conectadas às suas respectivas no Raspberry Pi, o V_{dd} e o V_{ref} foram conectados ao 3.3 Volts e a entrada analógica AIN0 foi conectada ao pino central do potenciômetro, que foi utilizado para testar o funcionamento do hardware e do software. As demais entradas analógicas e a saída DAC não foram conectadas a nada para este código. (H.8-A). Os resultados obtidos podem ser visualizados nas Figuras 6, 7 e 8, em que o potenciômetro é colocado em 2, 5 e 7 k Ω , respectivamente.

Para o código DAC (*Digital-Analog Converter*), foi realizada a mesma montagem, porém com uma alteração na porta de saída DAC (AOUT) do conversor. No código foi feito com que o potenciômetro, conectado na porta AIN0, controlasse o tempo no parâmetro da função *delay* no código e assim alterasse a frequência do led e a velocidade da contagem na tensão nos terminais do mesmo, que varia de 0 Volts a 3.3 Volts.

Calculando o erro pela fórmula 1

$$\frac{V_{esperado} - V_{real}}{V_{esperado}} \cdot 100\% \quad (1)$$

Obtemos para essa medida um erro de

$$E_{POT} = \frac{2 - 1.874118}{2} \cdot 100\% = 6.2941\% \quad (2)$$

```
Value: 209.000000 (de 255)
Value: 2.704706 Volts
Potenciometer: 1.915765 Kohms

Value: 210.000000 (de 255)
Value: 2.717647 Volts
Potenciometer: 1.874118 Kohms

Value: 210.000000 (de 255)
Value: 2.717647 Volts
Potenciometer: 1.874118 Kohms

Value: 210.000000 (de 255)
Value: 2.717647 Volts
Potenciometer: 1.874118 Kohms

Value: 210.000000 (de 255)
Value: 2.717647 Volts
Potenciometer: 1.874118 Kohms

Value: 210.000000 (de 255)
Value: 2.717647 Volts
Potenciometer: 1.874118 Kohms

Value: 210.000000 (de 255)
Value: 2.717647 Volts
Potenciometer: 1.874118 Kohms
```

Figura 6. Resultado obtido com o código ADC e potenciômetro em 2 k Ω .

```
Value: 138.000000 (de 255)
Value: 1.785882 Volts
Potenciometer: 4.872706 Kohms

Value: 138.000000 (de 255)
Value: 1.785882 Volts
Potenciometer: 4.872706 Kohms

Value: 138.000000 (de 255)
Value: 1.785882 Volts
Potenciometer: 4.872706 Kohms

Value: 138.000000 (de 255)
Value: 1.785882 Volts
Potenciometer: 4.872706 Kohms

Value: 138.000000 (de 255)
Value: 1.785882 Volts
Potenciometer: 4.872706 Kohms

Value: 138.000000 (de 255)
Value: 1.785882 Volts
Potenciometer: 4.872706 Kohms

Value: 138.000000 (de 255)
Value: 1.785882 Volts
Potenciometer: 4.872706 Kohms
```

Figura 7. Resultado obtido com o código ADC e potenciômetro em 5 k Ω .

Para o valor de 5k Ω (Figura 7), obtemos

$$E_{POT} = \frac{5 - 4.872706}{5} \cdot 100\% = 2.54588\% \quad (3)$$

```

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 91.000000 (de 255)
Value: 1.177647 Volts
Potenciometer: 6.830118 KOhms

Value: 90.000000 (de 255)
Value: 1.164706 Volts
Potenciometer: 6.871765 KOhms

```

Figura 8. Resultado obtido com o código ADC e potenciômetro em 7 kΩ.

Finalmente, para o valor de 7kΩ (Figura 8), obtemos

$$E_{POT} = \frac{7 - 6.830118}{7} \cdot 100\% = 2.42688\% \quad (4)$$

A partir das equações 2, 3 e 4, torna-se notório que, pelo valor da variação ser praticamente contante, quanto maior o valor, menor o erro percentual. Além disso, apesar da existência de erros, tais valores são considerados aceitáveis e não ultrapassam 10%, validando assim as medidas concebidas.

Para o conversor A/D (anexo H.8-B), alguns dos resultados obtidos podem ser exibidos nas figuras 9, 10 e 11, respectivamente.

Para efeito de melhor visualização, foi exibido o valor de saída direto do conversor na tela do dispositivo, que varia de 0 a 255 e seu respectivo resultado em volts (no pino AOUT), que varia de 0 a 3.3 Volts.



Figura 9. 1º Resultado obtido com o código DAC.

Normalizando cada um dos valores e aplicando a mesma fórmula de erro (equação 1), obtemos:

$$V_{volts} = \frac{2.87}{3.3} = 0.86970 \quad (5)$$

$$V_{AOUT} = \frac{218}{255} = 0.85490 \quad (6)$$

$$E_{2.87V} = \frac{0.85490 - 0.86970}{0.85490} \cdot 100\% = -1.731\% \quad (7)$$



Figura 10. 2º Resultado obtido com o código DAC.

Para os valores da Figura 10, obtemos:

$$V_{volts} = \frac{2.14}{3.3} = 0.64848 \quad (8)$$

$$V_{AOUT} = \frac{162}{255} = 0.63529 \quad (9)$$

$$E_{2.14V} = \frac{0.63529 - 0.64848}{0.63529} \cdot 100\% = -2.076\% \quad (10)$$



Figura 11. 3º Resultado obtido com o código DAC.

Por último, para os valores da Figura 11, temos:

$$V_{volts} = \frac{0.52}{3.3} = 0.15758 \quad (11)$$

$$V_{AOUT} = \frac{35}{255} = 0.13725 \quad (12)$$

$$E_{0.52V} = \frac{0.13725 - 0.15758}{0.13725} \cdot 100\% = -14.812\% \quad (13)$$

Através desses valores, podemos notar que, mesmo com um erro consideravelmente alto para a medida de 0.52 Volts (equação 13), essa alteração não influencia muito no valor final. Além disso, devemos observar que neste caso o erro é praticamente igual para todas as medidas e que só se torna algo tão notório no resultado da Fig. 11 porque o valor é consideravelmente baixo.

C. Ponto de Controle 3

Para o ponto de controle 3, a equipe focou em 2 pontos principais: a geração do gráfico em tempo real e a geração de um sinal de saída que represente a resposta da planta do sistema a ser controlado.

1) Gráfico em tempo real: Para construir um gráfico utilizando a linguagem C, foi necessário utilizar de alguns outros recursos e funcionalidades do sistema da própria Raspberry. Nesse caso, a ferramenta que se mostrou mais eficaz e simples para se trabalhar através de códigos em C, foi o **GNUPlot**, que é uma ferramenta utilizada para a geração de gráficos em geral para sistemas operacionais como Windows, Linux e muitos outros.

Para utilizar o GNUPlot, foi necessário instalar 2 pacotes de bibliotecas, o que foi realizado através de dois comandos: `sudo apt-get install gnuplot-x11` e `sudo apt-get install gnuplot`, no próprio terminal do sistema.

Para o projeto, foi elaborado um código em C que gera um arquivo GNU(.gnu), onde estão descritos todos os comandos a serem realizados dentro da interface de plotagem. A partir daí, é necessário executar o arquivo .gnu criado para que o mesmo plot o gráfico desejado, o que é feito através do comando `load <nome_do_arquivo.gnu>`. O código em C gerador desses comandos pode ser visto no anexo H.8-F.

Para a realização de testes antes de integrar o código aos códigos principais do projeto, utilizou-se de um código que gera um arquivo e imprime nele diversas linhas, cada uma de 2 colunas contendo os valores incrementados na variável armazenada (de 0 a 1000). Esse código pode ser visto no Anexo H.8-E.

2) Geração de sinal da Planta: Para gerar o sinal da planta, foi necessário inicialmente realizar a conversão das funções de transferência para o ambiente digital, utilizou-se o Matlab e a função "*c2d*".[8] Com a função já no domínio Z foi possível escrever a equação de diferenças desta, assim podendo descrever uma saída em função de uma entrada. Na Eq. 14.

$$y[i] = A \cdot x[i] + B \cdot x[i-1] - C \cdot y[i-1] + D \cdot y[i-3] \quad (14)$$

Como é necessário todas estas etapas para encontrar a equação de diferença, estabeleceu-se que o projeto já terá equações pré estabelecidas e selecionáveis assim facilitando a implementação e diminuindo o uso computacional. Para este ponto de controle as funções de transferência escolhidas estão na Fig. 12.

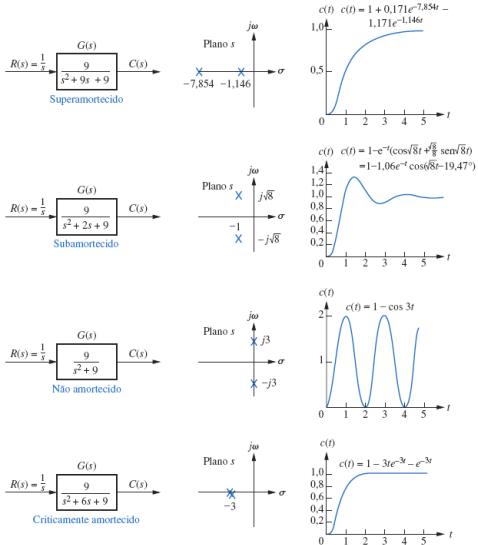


Figura 12. Sistemas de segunda ordem, diagramas de polos e respostas ao degrau.

Realizou-se a digitalização de cada uma e encontrou-se as equações de diferenças descritas no código Anexo H.8-D. Ainda no ambiente do Matlab plotou-se uma comparação entre as respostas ao impulso continua e digital, resultando nas Figs. 13, 14, 15 e 16.

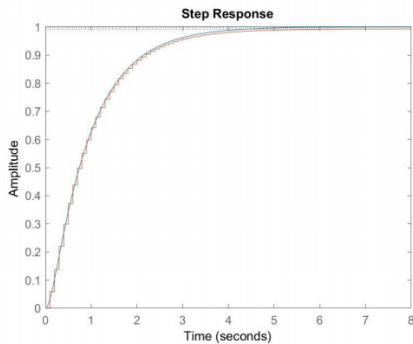


Figura 13. Resultado da Função de Transferência 1.

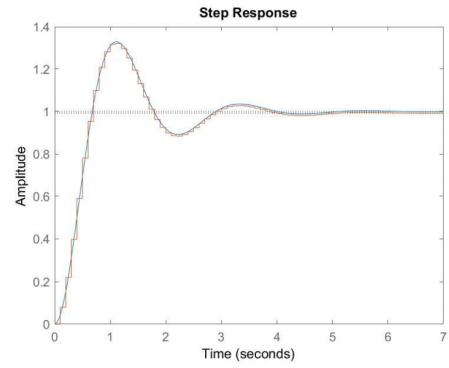


Figura 14. Resultado da Função de Transferência 2.

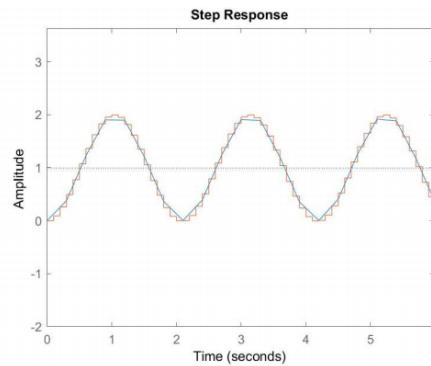


Figura 15. Resultado da Função de Transferência 3.

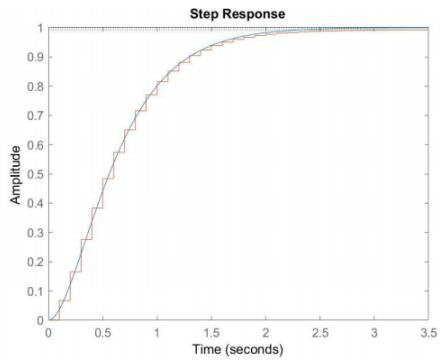


Figura 16. Resultado da Função de Transferência 4.

Em laboratório realizou-se a implementação do código com a Raspberry Pi, resultando nos resultados nas Figs. 17, 18, 19 e 20.

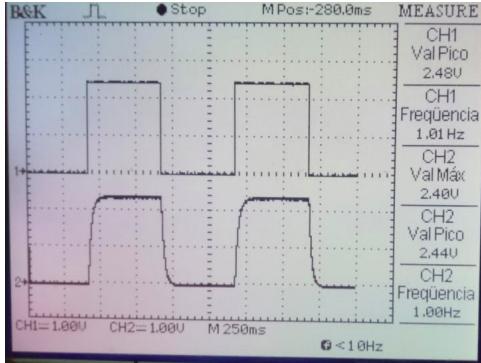


Figura 17. Resultado da Função de Transferência 1 no Osciloscópio.

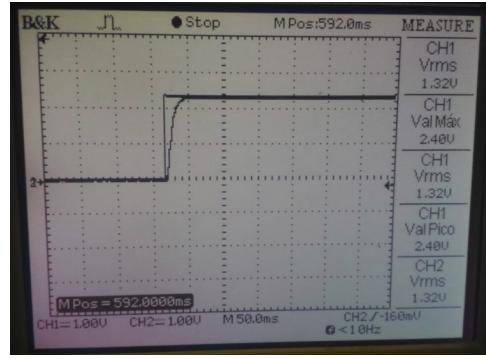


Figura 20. Resultado da Função de Transferência 4 no Osciloscópio.

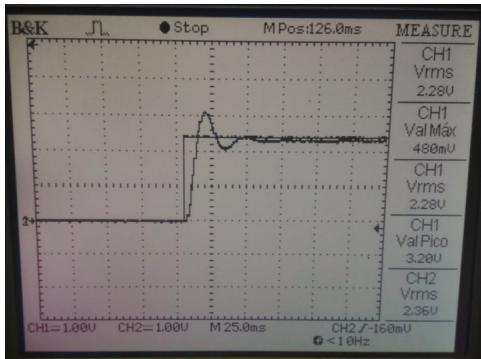


Figura 18. Resultado da Função de Transferência 2 no Osciloscópio.

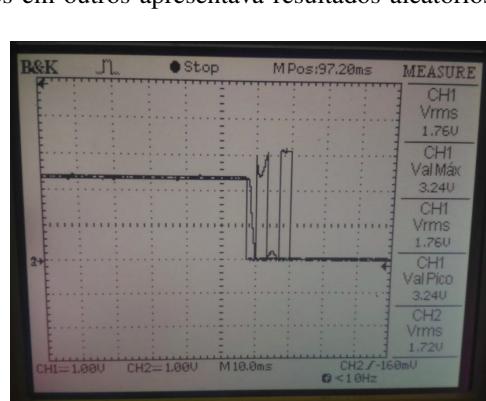


Figura 21. Erro da Função de Transferência 2 no Osciloscópio.

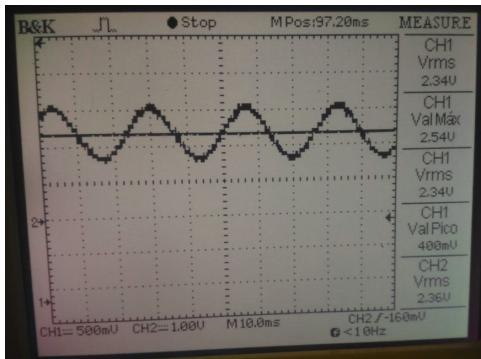


Figura 19. Resultado da Função de Transferência 3 no Osciloscópio.

D. Ponto de Controle 4

No ponto de controle 4, focou-se na resolução de problemas na plotagem dos gráficos e dos problemas na geração das ondas referentes às plantas do sistema a ser controlado pelo aluno.

1) Plotagem Gráfica: Para a plotagem do gráfico, no ponto de controle 3 utilizou-se a ferramenta disponível no terminal da Raspberry *GNUPlot*, entretanto essa forma de plotagem gerou diversos problemas como má alocação de memória, dificuldade em gerar o gráfico automaticamente e erros na atualização em tempo real das informações no gráfico. Para resolver esse problema, no ponto de controle 4, foi utilizada a linguagem *Python* e a ferramenta de plotagem da biblioteca *Matplotlib*.

Com essa alteração (códigos nos anexos H.8-G e H.8-H), se tornou possível a abertura automática do

código em python, que é o responsável pela geração e atualização do gráfico. Para isso, utilizou-se um código em linguagem C, responsável pela geração e verificação constante de atualizações no valor da variável contendo os parâmetros x e y, que serão guardados dentro de um arquivo .csv. Esse mesmo arquivo é aberto paralelamente pelo código em python, que lê o arquivo e, de forma paralela (por thread), plota o gráfico em intervalos de tempo pré-definidos.

Com o uso da solução produzida em python, os maiores problemas foram encontrados pela lentidão característica da biblioteca e da linguagem utilizadas para a geração e atualização de gráficos e por problemas de acesso a arquivos. Entende-se que, pelo fato de ambos os processos (em C e Python) estarem acessando o arquivo continua e paralelamente, é possível que esteja acarretando em erros de assincronismo durante a plotagem, o que acaba gerando problemas no processo.

2) *Geração de ondas*: Para se resolver os problemas na borda de descida dos gráficos, o código realizado no ponto de controle 3, foi alterado para o código do Anexo H.8-I. Com a variável *flag*, foi possível controlar com que a planta só funcionasse a partir da borda de subida e meio período de nível lógico alto, não exigindo uso de circuitos analógicos e não havendo perda para aplicação do controlador. A Figura 22, demonstra que o erro da Função de Transferência 2 foi retirado, Figura 18.

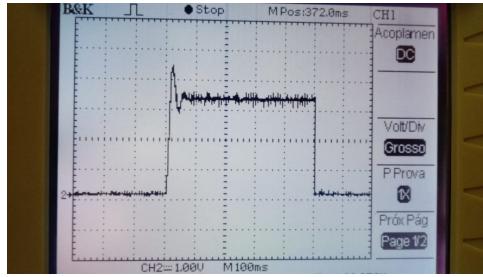


Figura 22. Nova FT, com borda de descida resolvida.

E. Projeto Final

Para o projeto final, foram feitas algumas alterações no escopo do projeto.

Visto que as tentativas de plotagem do gráfico em tempo real se apresentaram falhas e inúteis para o usuário, seja por problemas de tempo, consumo de memória, imprecisão etc., trabalhar na interface com o usuário (em software) e na correção de erros na geração do sinal, que será o que o aluno realmente precisará, se mostrou algo muito mais eficaz e urgente, dadas as limitações de tempo e conhecimento em software por parte dos projetistas.

Dessa forma, como não haverá gráfico em tela, com o projeto atual, o usuário terá que fazer o uso de um osciloscópio para verificar o resultado do que está sendo gerado.

Com relação à interface com o usuário, para realizar essa parte foi utilizado o software *QT Creator*, que é um software construído de forma que a produção da interface possa ser elaborada via software (código), de forma "gráfica" e dinâmica, ou seja, adicionando e configurando botões, gráficos, *labels*, títulos, imagens etc., e de forma híbrida, que foi a forma utilizada no projeto.

É importante ressaltar que, apesar de o software também trabalhar com algumas outras linguagens, a interface foi totalmente construída em C++, o que aumentou a gama de possibilidades do projeto, porém reduziu o nível de habilidade do grupo para manusear a ferramenta, visto que ambos não tinham muito conhecimento na linguagem citada.

Foram tiradas imagens da interface para visualização do layout no Desktop. As figuras de 23 a 29 mostram todas as fases existentes até o momento.

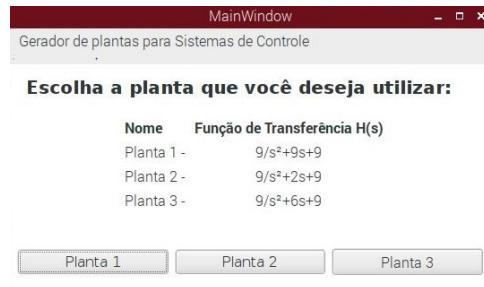


Figura 23. Janela principal da Interface.

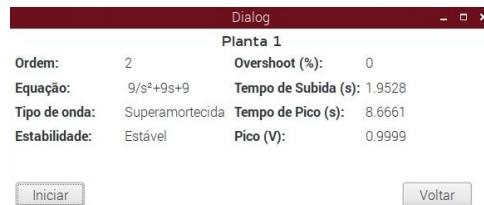


Figura 24. Janela após a Planta 1 ser selecionada.



Figura 25. Janela enquanto a Planta 1 está sendo gerada.



Figura 26. Janela após a Planta 2 ser selecionada.



Figura 27. Janela enquanto a Planta 2 está sendo gerada.

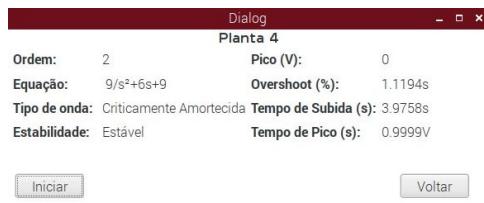


Figura 28. Janela após a Planta 3 ser selecionada.



Figura 29. Janela enquanto a Planta 3 está sendo gerada.

Obs.: Devido à grande quantidade de códigos gerada pelo QT Creator para a construção da interface, esses códigos não serão anexados no corpo deste documento, mas estarão disponíveis junto aos pontos de controle, dentro de outra outra pasta nomeada *códigos*, onde haverá todos os códigos utilizados para o projeto final.

Para facilitar a utilização do código gerador pela interface, foi realizado uma pequena alteração na escolha de planta, ela sendo realizada já na execução do projeto, utilizando os argumentos em linha de comando, *int argc,char**argv*, não se faz necessário discorrer o código, pois não ouve mudança significativa.

F. Conclusões

No inicio do projeto foram estabelecidos necessidades práticas para viabilização do projeto, destas a principal que era a geração de uma onda, a partir de uma função de transferência da planta foi atingida com sucesso, podendo o aluno realizar os experimentos práticos.

No decorrer do projeto notou-se que a comparação entre referência e sinal de realimentação feita no ambiente digital não seria bom para parte educacional, pois o desenvolvimento de um comparador e a análise das ondas de entrada e saída do mesmo fazem parte da análise. Pelo fato dos criadores da ferramenta, não terem alcançado a meta estabelecida de plotagem gráfica, acabou que esta análise só seria possível utilizando um osciloscópio.

A interface agregou facilidade de uso da ferramenta desenvolvida e ainda trouxe a possibilidade de dar ao aluno informações sobre a onda gerada, Figura 24, para um fechamento completo do projeto seria necessário a tentativa de implementação de um circuito controlador, para testes, porém a implementação demandaria tempo e recursos não disponíveis.

REFERÊNCIAS

- [1] N. S. Nise, *Engenharia de Sistemas de Controle*, 7th ed. LTC, 2017.
- [2] K. Ogata, *Engenharia de controle moderno*, 5th ed. Pearson, 2011.
- [3] M. F. M., “Controlador pid analógico: uma abordagem didática em laboratório,” *COBENGE*, 2005.
- [4] N. I. Corporation. (2019) What is hardware-in-the-loop? [Online]. Available: <http://www.ni.com/pt-br/innovations/white-papers/17/what-is-hardware-in-the-loop-.html>
- [5] M. A. A. Melo, “Ensino de sistemas de controle usando aplicações reais em engenharia elétrica,” *COBENGE*, 2005.
- [6] E. Alecrim. (2018) Raspberry pi 3 model b+ melhora conectividade e desempenho, mas mantém preço baixo. [Online]. Available: <https://tecnoblog.net/236329/raspberry-pi-3-model-b-plus/>
- [7] *PCF8591 8-bit A/D and D/A converter*, NXP Semiconductors, 6 2013.
- [8] I. The MathWorks. (2019) c2d-convert model from continuous to discrete time. [Online]. Available: <https://www.mathworks.com/help/control/ref/c2d.html>

ANEXOS

A. Código para conversor Analógico-Digital

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>

#define Address 0x48
#define BASE 64
#define A0 BASE+0
#define A1 BASE+1
#define A2 BASE+2
#define A3 BASE+3
#define POT 10.62

int main(void)
{
    float value;
    wiringPiSetup();
    pcf8591Setup(BASE, Address);
    float pot=POT;

    while(1)
    {
        value = analogRead(A0);
        pot = ((-value*(POT)/255)+POT);

        printf("\nValue : %f (de 255)\n", value);
        printf("Value : %f Volts\n", value *3.3/255);
        printf("Potenciometer : %f KOhms\n", pot);
        delay(1000);
    }
}
```

B. Código para conversor Digital-Analógico

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>

#define Address 0x48
#define BASE 64
#define A0 BASE+0
#define A1 BASE+1
#define A2 BASE+2
#define A3 BASE+3

int main(void)
{
    int valueIn, inc=0, valueOut=0;
    wiringPiSetup();
    pcf8591Setup(BASE, Address);

    while(1)
    {
        valueIn = analogRead(A0);
        if (inc==0)
            analogWrite(A0, valueOut++);
        else
            analogWrite(A0, valueOut--);
        if (valueOut==255)
            inc=1;
        else if (valueOut==0)
            inc=0;
        printf("AOUT: %d\n", valueOut);
        delay(valueIn/100);
    }
}
```

C. Código para gerar um PWM

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>

const int PWM_pin = 1; // GPIO 1
int main (void)
{
    int duty_cicle ;

    if (wiringPiSetup () == -1)
        exit (1) ;

    pinMode (PWM_pin, PWM_OUTPUT) ;

    while (1)
    {
        for (duty_cicle = 0 ; duty_cicle < 1024 ; ++duty_cicle)
        {
            pwmWrite (PWM_pin, duty_cicle) ;
            delay (1) ;
        }
        delay (1);

        for (duty_cicle = 1023 ; duty_cicle >= 0 ; --duty_cicle)
        {
            pwmWrite (PWM_pin, duty_cicle) ;
            delay (1) ;
        }
        delay (1);
    }
}
```

D. Código de gerador de sinal da Planta

```

#include <wiringPi.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <pcf8591.h>
#include <stdio.h>
#include <stdlib.h>

#define Address 0x48
#define BASE 64
#define A0 BASE+0
#define SAIDA 23
#define MEIO_PERIODO (1e6/16)

int main(void)
{
    pid_t filho;
    float y[40] = {0};
    float x[40] = {0};
    int i, escolha;
    wiringPiSetup();
    pcf8591Setup(BASE, Address);
    pinMode(SAIDA, OUTPUT);
    pinMode(ENTRADA, INPUT);
    printf("Escolha uma Funcao de Transferencia:\n");
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    printf("Insira o numero da FT desejada :\n");
    scanf("%d\n", &escolha);
    filho = fork();
    if (filho == 0)
    {
        while (1)
        {
            digitalWrite(SAIDA, HIGH);
            usleep(MEIO_PERIODO);
            digitalWrite(SAIDA, LOW);
            usleep(MEIO_PERIODO);
        }
    }
    else
    {
        while (1)
        {
            for (i = 39; i > 0; i--)
            {
                x[i] = x[i - 1];
                y[i] = y[i - 1];
            }
        }
    }
}

```

```

    usleep(10);
    x[0] = analogRead(A0);
    usleep(10);
    switch(escolha)
    {
        case 1:
            y[0] = (0.05847*x[1]) + (0.00000001*x[2])
                    + (1.348*y[1]) - (0.4066*y[2]);
        break;
        case 2:
            y[0] = (0.08035*x[1]) + (0.00000001*x[2])
                    + (1.738*y[1]) - (0.8187*y[2]);
        break;
        case 3:
            y[0] = (0.08866*x[1]) + (0.00000001*x[2])
                    + (1.911*y[1]) - y[2];
        break;
        case 4:
            y[0] = (0.06667*x[1]) + (0.00000001*x[2])
                    + (1.482*y[1]) - (0.5488*y[2]);
        break;
        default
            y[0] = x[0];
    }
    analogWrite(A0,y[0]);
}
}

```

E. Código para criar e colocar dados em um arquivo .csv (PC3)

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <wiringPi.h>

// function for creating and writing in file
int main()
{
    FILE *fp;
    char filename []="graph_data.csv";
    long int msg=0;

    printf("\nCreating %s file",filename);

    fp=fopen(filename , "w+");
    if (fp == NULL)
    {
        printf("\nOcorreu algum problema
na abertura do arquivo (Parte 1)\n");
        return 1;
    }
    fprintf(fp , "X Axis , Y Axis");
    fclose(fp);
    while(msg<=1000)
    {
        fp=fopen(filename , "a+");
        if (fp == NULL)
        {
            printf("\nOcorreu algum problema
na abertura do arquivo (Parte 2)\n");
            return 1;
        }
        printf("\n%ld , %ld", msg);

        sleep(1);
        fprintf(fp , "\n%ld , %ld", msg , msg);
        fclose(fp);
        msg++;
    }

    return 0;
}
```

F. Código em linguagem C para gerar um código em .gnu

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <wiringPi.h>

// function for creating and writing in file
int main()
{
    FILE *fp;
    int i=0;
    fp = fopen("c_graph.gnu", "w+");
    fprintf(fp, "set_title 'GNU_Plot-C_Real-Time_Graph'\nset_xlabel
'Number'\nset_ylabel 'Saida'\nplot 'graph_data.csv'
using_1:2_title_columnhead_with_lines");
    while(i<1000)
    {
        fprintf(fp, "\npause_3.0\nreread\nset_autoscale");
        i++;
    }
    fclose(fp);
}
```

G. Código para criar e colocar dados em um arquivo .csv (PC4)

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>

int code;

int comp=0, pos=0, mem=0, flag_py=0;

float y[]={6, 8, 20, 2, 7, 25};
float x[(sizeof(y)/sizeof(float))]={1, 2, 3, 4, 5, 6};
char xlabel[]="X_label", ylabel[] = "Y_label";
int size = (sizeof(y)/sizeof(float));
float old_y[]={0, 0, 0, 0, 0, 0};
int reset = 1;
int i=0;

char* itoa(int val, int base);

void ctrl_response();
void* create_arq();
void compare_vec(float* vec1, float* vec2);

int main(void)
{
    pthread_t thread1_id;
    signal(SIGINT, ctrl_response);
    pthread_create(&thread1_id, NULL, create_arq, NULL);
    if(flag_py==0)
    {
        flag_py=1;
        system("sudo_python_real_time_graph.py");
    }
    pthread_join(thread1_id, NULL);
    return 0;
}

void *create_arq()
{
    while(1)
    {
        compare_vec(old_y, y);
        if(comp != 0)
        {
            FILE *fp;
```

```

char filename []="graph_data.csv";
int count=0;
if (mem == 1)
    reset=0;
mem = 1;
signal(SIGINT, ctrl_response);
if(reset==1)
{
    fp=fopen(filename , "w+");
    if (fp == NULL)
    {
        printf("Erro na abertura
do arquivo (Parte_1)\n");
        exit(-1);
    }
    fprintf(fp , "%s ,%s \n" , xlabel , ylabel);
    fclose(fp);
}

code = getpid();
fp=fopen(filename , "a+");
if (fp == NULL)
{
    printf("\nErro na abertura
do arquivo (Parte_2)\n");
    exit(-1);
}
for(count=0;count<size ;count++)
{
    i++;
    //printf("ni = %f\n" , i/1.0);
    fprintf(fp , "%f ,%f \n" , i/1.0 , y[count]);

}
fclose(fp);
memcpy(old_y , y , size );
sleep(10);
}
return 0;
}

void ctrl_response()
{
    char rm_file [] = "sudo_rm_graph_data.csv";
    system(rm_file);
    printf("\n-----><-----\n");
    printf("\n----->_Killing <-----\n");
    printf("\n----->_the <-----\n");
    printf("\n----->_process <-----\n");
}

```

```

printf("\n----->_<-----\n");
kill(code , SIGINT);

exit(1);
}

char* itoa(int val , int base)
{
    static char buf[32] = {0};

    int f = 30;

    for(; val && f ; --f , val /= base)

        buf[f] = "0123456789abcdef"[val % base];

    return &buf[f+1];
}

void compare_vec(float* vec1 , float* vec2)
{
    int c=0;
    int size1=(sizeof(vec1)/sizeof(vec1[0]));
    int size2=(sizeof(vec2)/sizeof(vec2[0]));

    if(size1 != size2)
    {
        printf("Caution !_Vectors_with_different_lenght . ");
        comp = 1;
    }
    else
        for(c=0;c<size1 ;c++)
            if(vec1[c] != vec2[c])
                comp = 1;

    if(comp != 1)
        comp = 0;
}

```

H. Código para gerar um gráfico em Python

```
def read_csv(x, y):
    import csv

    line=0;
    with open( 'graph_data.csv ', 'r ') as csvfile:
        plots = csv . reader ( csvfile , delimiter=',' )
        for row in plots:
            if (line==0):
                xlabel=row[0]
                ylabel=row[1]
            else :
                x.append( float (row[0]))
                y.append( float (row[1]))
            line +=1
    return x, y, xlabel, ylabel

def call_plot(x, y, fig, ax):
    import matplotlib.pyplot as plt
    import time

    #print(x, y)
    ax.plot(x, y, '-.b', lw=2)
    plt.grid(True)
    plt.scatter(x, y)
    fig.canvas.draw()
    plt.show()

def graph_gen():
    import time
    import psutil
    import matplotlib.pyplot as plt
    import os
    import sys
    import thread
    fig = plt.figure()
    ax = fig.add_subplot(111)
    fig.show()
    while(os.path.isfile("graph_data.csv")):
        x, y = [], []
        x, y, xlabel, ylabel = read_csv(x, y)
        thread.start_new_thread(call_plot, (x, y, fig, ax))
        time.sleep(3)

# Main
graph_gen()

plt.close()
```

```
sys.exit()
```

I. Código para ondas fixadas e funcionalidade de reescolha de Planta

```
#include <wiringPi.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <pcf8591.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define Address 0x48
#define BASE 64
#define A0 BASE+0
#define SAIDA 23
#define MEIO_PERIODO (1e6/2)

float y[40] = {0};
float x[40] = {0};
int flag = 0;

void *respostaPlanta(void *unused);
void opcoes(int);
int i, escolha;

int main(void)
{
    pthread_t thread_id2;
    wiringPiSetup();
    pcf8591Setup(BASE, Address);
    pinMode(SAIDA, OUTPUT);
    signal(SIGINT, opcoes);
    printf("Escolha uma Funcao de Transferencia:\n");
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    printf("5) Sair do programa\n");
    printf("Insira o numero da FT desejada :\n");
    scanf("%d", &escolha);
    if(escolha == 5){
        printf("Aplicacao finalizada");
        exit(1);
    }
    pthread_create(&thread_id2, NULL, &respostaPlanta, NULL);

    while(1){
        flag = 1;
        digitalWrite(SAIDA, HIGH);
        usleep(MEIO_PERIODO);
        flag = 0;
        digitalWrite(SAIDA, LOW);
    }
}
```

```

        usleep(MEIO_PERIODO);
    }
}

void *respostaPlanta(void *unused){
    while(1)
    {
        for( i = 39; i >0; i --){
            x[ i ] = x[ i -1 ];
            y[ i ] = y[ i -1 ];
        }
        usleep(10);
        x[0] = analogRead(A0);
        usleep(10);
        if(flag == 1){
            switch (escolha)
            {
                case 1:
                    y[0] = (0.05847*x[1]) + (0.00000001 *x[2])
                    + (1.348*y[1]) - (0.4066*y[2]);
                break;
                case 2:
                    y[0] = (0.08035*x[1]) + (0.00000001 *x[2])
                    + (1.738*y[1]) - (0.8187*y[2]);
                break;
                case 3:
                    y[0] = (0.08866*x[1]) + (0.00000001 *x[2])
                    + (1.911*y[1]) -y[2];
                break;
                case 4:
                    y[0] = (0.06667*x[1]) + (0.00000001 *x[2])
                    + (1.482*y[1]) - (0.5488*y[2]);
                break;
                case 5:
                    exit(1);
                default:
                    y[0] = x[0];
            }
        } else y[0] = x[0];
        analogWrite(A0,y[0]);
    }
}

void opcoes(int sig){
    printf("Sinal_Crtl+C(%d)_recebido !!\n",sig );
    printf("Deseja_alternar_a_funcao_de_transferencia?_Seleciona_a_nova:\n",sig );
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    printf("5) Sair do programa\n");
    scanf("%d",&escolha );
}

```

```
if(escolha == 5){  
    printf("Aplicacao_finalizada");  
    exit(1);  
}  
}
```