

Projeto Efeito Hall- H.E.L.L.O., world!

Edélio G. M. de Jesus, Gisela Ceresér Kassick, Giulia Sales Ferreira, Leonardo R. dos S. Vieira

Professores orientadores: Daniel Roberto Cassar, James Moraes de Almeida e Leandro Nascimento Lemos

*Ilum Escola de Ciência, Centro Nacional de Pesquisa em Energia e Materiais (CNPem),
CEP 13083-970, Campinas, São Paulo, Brasil.*

SUMÁRIO

1. Introdução
2. Código em C++ para Arduino
3. Medição de Efeito Hall e manipulação de dados no VSCode
4. Simulação
5. Simulador Livre
6. Conclusão
7. Referências

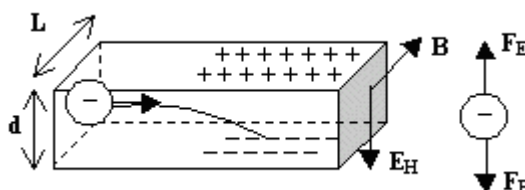
1. INTRODUÇÃO

O Efeito Hall descreve o desvio de elétrons em um circuito sob influência de um campo magnético e a consequente diferença de potencial em direção perpendicular. É uma expressão da força de Lorentz

$$\vec{F} = q\vec{E}_y + q\vec{v}_y \times \vec{B}_x$$

Em que $q\vec{E}_y$ é a força no sentido da corrente e $q\vec{v}_y \times \vec{B}_x$ é a força magnética, sendo perpendicular à velocidade e ao campo B.

Figura 1



Elétron sendo desviado, gerando tensão entre duas faces opostas. Imagem retirada de:
https://www.if.ufrgs.br/tex/fis142/mod08/m_s03.html.

O H.E.L.L.O., world! — Hall Effect Lab for Learning and Observation é um programa que tem como objetivo oferecer uma forma simples de visualizar o efeito Hall em um elétron como partícula clássica, a fim de facilitar a aprendizagem desse fenômeno físico visual e interativamente. O usuário pode analisar tanto graficamente quanto em uma simulação como se comporta esse fenômeno em diferentes parâmetros, e ainda pode obter os dados necessários de maneira experimental com um sensor hall e Arduino.

Para isso, foram criados três códigos: um em C++ para programar o Arduino, um em Python que roda no VSCode e outro em python para rodar no ambiente Glowscrip. Esses códigos serão explicados nos pontos a seguir do presente relatório.

2. CÓDIGO EM C++ PARA O ARDUINO

O código em linguagem Arduino (baseado em C++) tem como objetivo ler o valor de tensão proveniente de um sensor Hall conectado ao pino analógico do Arduino e enviar esse valor convertido para tensão (em Volts) via porta serial para o computador. Esse valor será usado posteriormente pelo programa em Python.

2.1 Definição de variáveis e constantes:

Logo no início do código, são definidas as variáveis globais necessárias para a leitura e conversão:

- `pinoSensor`: constante que define qual pino analógico está sendo utilizado para a leitura do sensor. Neste caso, é o A0, o que significa que o fio de saída do sensor Hall está conectado à entrada analógica A0 do Arduino.
- `valorHall`: variável do tipo `int` que armazenará o valor lido pelo conversor analógico-digital (A/D) do Arduino. Esse valor varia entre 0 e 1023, representando uma faixa de 0 a 5 Volts.
- `tensao`: variável do tipo `float` que armazenará a conversão do valor analógico lido para um valor de tensão real, com precisão de ponto flutuante.

2.2 Função `setup()`:

A função `setup()` é executada uma vez quando o Arduino é ligado ou reiniciado. Aqui, duas ações são realizadas:

- Inicialização da comunicação serial com o computador, com taxa de transmissão de 9600 bauds, usando `Serial.begin(9600)`; isso é fundamental para permitir que o Arduino envie dados para o programa Python.
- Configuração do pino de entrada: `pinMode(pinoSensor, INPUT)`- informa ao Arduino que o pino A0 será utilizado apenas para leitura, já que o sensor envia dados ao Arduino.

2.3 Função `loop()`:

A função `loop()` roda continuamente após a execução do `setup()`. Seu papel neste código é aguardar um comando vindo do computador e, quando o comando correto for recebido, fazer a leitura do sensor e enviar o resultado.

O loop inicia verificando se há dados disponíveis na porta serial com “`Serial.available() > 0`”. Em seguida, lê o comando enviado pelo programa em python com “`Serial.readStringUntil('\n')`”. Com isso, o Arduino irá ler a string enviada até encontrar uma quebra de linha (`\n`), garantindo que todo o comando seja lido corretamente.

Tendo lido o comando recebido, o Arduino verifica se ele é igual a "Medir" (a resposta esperada). Caso seja, o Arduino inicia a leitura do sensor através do `analogRead(pinoSensor)`, recebendo o valor analógico vindo do sensor Hall (de 0 a 1023). Esse valor é proporcional à

tensão medida, sendo 0 para 0 V e 1023 para 5 V. O valor lido é convertido para tensão em Volts a partir da seguinte fórmula:

$$\text{tensão} = \text{valor lido} \cdot 5/1023$$

Finalmente, é realizado o envio do valor da tensão para o computador via `Serial.println(tensao, 4)`, que imprime o valor com quatro casas decimais de precisão, facilitando a análise posterior no Python.

3. MEDIÇÃO DO EFEITO HALL E MANIPULAÇÃO DOS DADOS NO VSCO

3.1 IMPORTAÇÕES

O programa inicia com a importação das bibliotecas necessárias para seu funcionamento:

1. VPython – utilizada para a criação de gráficos interativos e simulações visuais;
2. Serial – utilizada para estabelecer a comunicação com o Arduino via porta serial;
3. Time – utilizada para definir pausas entre comandos, garantindo o tempo necessário para resposta do Arduino.

3.2 INICIALIZAÇÃO DA COMUNICAÇÃO COM O ARDUINO

A comunicação com o Arduino é iniciada por meio da porta serial COM3 (ajustável conforme o ambiente). Essa conexão é atribuída à variável `arduino`. Após a conexão, o programa aguarda 2 segundos para garantir que o Arduino tenha tempo suficiente para inicializar antes de receber comandos.

3.3 FUNÇÃO RECEBER_DADOS_ARDUINO()

Essa função não requer argumentos e realiza a medição de valores de tensão Hall em cinco distâncias fixas, utilizando um sensor conectado ao Arduino. Ao final, retorna três estruturas de dados organizadas:

1. Lista com as distâncias (em cm);
2. Dicionário com os campos magnéticos calculados para cada distância;
3. Dicionário com as tensões Hall medidas.

Inicialmente, há a definição das seguintes variáveis:

- Dois dicionários vazios: `dados_tensao` e `campos_por_distancia`, que armazenarão os valores medidos.
- A lista de distâncias, fixada como `[1, 2, 3, 4, 5]`, representando as distâncias do ímã ao sensor.
- Duas constantes físicas: corrente (4,2 mA) e sensibilidade do sensor (1,8 mV/G).
- A variável `numero_medida`, que controla o ponto de medição atual e inicia em 1.

Em seguida, um laço `while` é executado enquanto `numero_medida` for menor que 6. A cada iteração, o usuário é instruído a posicionar o ímã na distância correta e confirmar a ação digitando o número correspondente. Ao receber a confirmação, o programa envia um comando ao Arduino para realizar a medição e aguarda brevemente a resposta. A resposta (já convertida em volts pelo próprio Arduino) é lida, convertida para float e subtraída de 2,5 V, valor típico de offset de sensores Hall. O valor resultante é exibido ao usuário e definido como a variável da tensão hall.

O usuário escolhe se deseja repetir a medição. Caso opte por repetir, os dados não são armazenados (a variável da tensão hall retorna a 0) e o loop reinicia. Caso contrário, a tensão Hall é armazenada no dicionário dados_tensao e o campo magnético é calculado pela fórmula $B = \frac{V_H}{S \cdot I}$, onde B é o campo magnético, V_H é a tensão Hall, I é a corrente, e S é a sensibilidade. O valor de B é então registrado no dicionário campo_por_distancia e, por fim, soma-se 1 à variável numero_medida e a tensão Hall é reiniciada para zero.

Ao final das cinco medições, a comunicação serial é encerrada (arduino.close()), há um print para avisar o usuário que as medidas foram confirmadas e os dados coletados são retornados para uso posterior.

3.4 FUNÇÃO RECEBER_DADOS_USUARIO()

Essa função sem argumentos recebe os dados inseridos manualmente pelo usuário. Ela retorna três estruturas de dados:

1. Lista com as distâncias fornecidas (em cm);
2. Dicionário com os campos magnéticos calculados teoricamente;
3. Dicionário com as tensões Hall correspondentes.

Inicialmente, o usuário informa, através de um input, quantos pontos de distância deseja analisar. Em seguida, insere manualmente cada uma das distâncias, que são armazenadas em uma lista.

Depois, são definidas constantes físicas que serão usadas nos cálculos:

- Corrente (I): 4,2 mA (4.2×10^{-3} A);
- Densidade de portadores (n): $1 \times 10^{22} \text{ m}^{-3}$;
- Carga do elétron (q): $1,6 \times 10^{-19}$ C;
- Espessura do sensor (t): 1×10^{-4} m;
- Constante (k): 0,0035 (ajustada empiricamente para representar o decaimento do campo magnético com a distância em cm).

A seguir, dois dicionários (campos_por_distancia e dados_tensao) vazios são inicializados para armazenar os dados. Para cada distância, são realizados os seguintes cálculos:

- 1) Campo magnético:

$$B = \frac{k}{d^3}$$

- 2) Tensão Hall:

$$V_H = \frac{B \cdot I}{n \cdot q \cdot t}$$

Os resultados são armazenados nos dicionários, organizados por distância, e também são impressos para o usuário. Ao final, a função retorna os três conjuntos de dados armazenados e citados anteriormente (lista de distâncias e dicionários de campo e tensão).

3.5 FUNÇÃO PLOTAR_GRAFICOS()

Essa função é responsável por gerar visualmente um gráfico da relação entre distância e tensão Hall, utilizando a biblioteca VPython.

As entradas da função são:

1. Lista de distâncias (em cm);
2. Dicionário campos_por_distancia (não utilizado nesta versão);
3. Dicionário dados_tensao.

A função cria uma janela gráfica com título "Tensão Hall x Distância", eixos rotulados e dimensões definidas (600x400 pixels). Utiliza os seguintes componentes da biblioteca VPython:

- graph: para configurar o sistema de coordenadas;
- gcurve: para desenhar a curva que conecta os pontos.

Para cada distância, a função extrai o valor correspondente da tensão Hall (armazenado como primeiro elemento de uma lista) e o plota no gráfico. Caso não haja valor para uma determinada distância, assume-se 0 para evitar falhas.

3.6 BLOCO DA SIMULAÇÃO

O penúltimo bloco serve como a definição da função de simulação 3D do efeito hall. Essa função será explicada mais detalhadamente no ponto 2 deste relatório, já que é utilizada em dois dos três códigos apresentados no repositório.

3.7 BLOCO PRINCIPAL DE EXECUÇÃO

Esse é o ponto de entrada do programa e é executado apenas quando o script é executado diretamente. Ele organiza o fluxo de interação com o usuário e conecta todas as funções descritas anteriormente.

Primeiramente, exibe mensagens de boas-vindas, apresentando o projeto como *H.E.L.L.O.* (Hall Effect Lab for Learning and Observation). Em seguida, solicita, via input, ao usuário que escolha entre dois modos de operação:

- Modo 1: Inserção manual de dados teóricos;
- Modo 2: Coleta de dados via Arduino.

Com base na escolha, a função correspondente é chamada- receber_dados_usuario() para o modo 1 e receber_dados_arduino() para o modo 2. Se a escolha for inválida, o programa exibe uma mensagem e é encerrado.

Após a coleta dos dados (com sucesso), chama a função plotar_graficos() para visualização dos resultados. O gráfico é plotado em uma janela diferente.

Tendo visto o gráfico, o usuário é convidado a selecionar uma distância para realizar uma simulação mais detalhada do efeito Hall, chamando a função simular_efeito_hall() com base no valor de tensão Hall correspondente à distância escolhida. A simulação irá aparecer na mesma janela do gráfico, logo abaixo dele.

Por fim, o programa aguarda que o usuário pressione *Enter* para finalizar a execução e fechar a janela gráfica.

4. SIMULAÇÃO

O primeiro trecho do código referente à simulação consiste na criação da cena a partir do método *canvas* do módulo *Vpython* (Figura 2). Nela são estabelecidos os seguintes parâmetros: título, tamanho da cena bem como sua posição na página *localhost* que será criada.

Figura 2

```
##### Cena #####
scene = canvas(title="Efeito Hall com Campo Elétrico Dinâmico",
               width=800, height=500, align="right")
```

A seguir, foi necessário declarar uma série de variáveis (Figura 3), divididas em dois grupos: as constantes físicas e os parâmetros do sensor – o que inclui a estrutura física e as características de condutividade. Buscou-se usar os valores correspondentes à realidade, para tanto consultou-se o *datasheet* do sensor utilizado.

Figura 3

```
# ===== Constantes físicas =====
q = -1.6e-19 # carga do elétron (C)
m = 9.11e-31 # massa do elétron (Kg)
v0 = vector(0, 0, 0) # velocidade inicial no eixo x
dt = 2e-11 # passo de tempo (pequeno para alta aceleração)
i = 4.2e-3 # intensidade da corrente (A)

# ===== Parâmetros do sensor =====
s = 1.8e-3 # V/Gauss
w = 1e-3 # m (largura do condutor)
L = 5e-3 # comprimento lateral (x)
t = 1e-4 # espessura
sigma = 1e3 # condutividade do silício dopado (S/m)
n_portadores = 1e22 # densidade de portadores (semicondutor silício)
```

Foi então definida a função de criação das esferas que representam os elétrons (Figura 4).

Figura 4

```
# Criação de Elétrons
def criar_eletron():
    e = sphere(pos=vector(-5e-3, 0, 0), radius=0.1e-5,
               color=color.cyan, make_trail=True)
    e.vel = v0
    eletrons_moveis.append(e)
```

O cálculo do campo elétrico (Figura 5) levou em consideração as características do sensor. Enquanto isso, para o campo magnético (Figura 6), foi necessário realizar uma aproximação. Por fim, para a tensão Hall utilizou-se a equação padronizada (Figura 6).

Figura 5

```
# Campo elétrico gerador da corrente
A = w * t
J = i / A
E_intensidade = J / sigma
E = vector(-E_intensidade, 0, 0)
```

Figura 6

```
B_intensidade = campo_magnetico_maximo / (distancia_livre**3) # campo magnético em Tesla
Tensao_hall = (B_intensidade * i) / (n_portadores * q * t) # tensão Hall em Volts
```

Segue a parte que propriamente realiza a simulação do efeito Hall (figura 7). Nela são calculadas as forças elétrica e magnética a partir dos campos anteriormente definidos, bem como o campo elétrico hall que surge a partir do acúmulo de cargas. Foi utilizada a primeira lei de Newton para que o movimento de cada elétron fosse atualizado com o passar do tempo. Além disso, foram definidos os limites da caixa criada, a fim de que os elétrons sumissem assim que encostassem nas bordas da caixa que representa o condutor.

Figura 7

```
# Atualiza movimento
for e in eletrons_movels[:]:
    # Força elétrica (externa)
    F_elet = q * E if campo_eletrico_ativo else vector(0, 0, 0)

    # Força magnética
    F_mag = q * cross(e.vel, B) if campo_magnetico_ativo else vector(0, 0, 0)

    # Campo Hall dinâmico
    delta_n = n_direita - n_esquerda
    # Campo Hall real (a partir do acúmulo de carga)
    A_face = L * w # área da face onde há acúmulo
    sigma_s = (abs(q) * 1e3 * delta_n) / A_face # densidade por área
    epsilon_0 = 8.85e-12 # permissividade do vácuo
    E_hall = vector(0, 0, sigma_s / epsilon_0)

    # Força de Hall efetiva
    F_elet_hall = q * E_hall

    # Movimento
    a = (F_elet + F_mag + F_elet_hall) / m
    e.vel += a * dt
    e.pos += e.vel * dt

    if e.pos.z < -bloco.size.z / 2:
        n_esquerda += 1
        eletrons_acumulados.append(e)
        eletrons_movels.remove(e)
        e.visible = False
    elif e.pos.z > bloco.size.z / 2:
        n_direita += 1
        eletrons_acumulados.append(e)
        eletrons_movels.remove(e)
        e.visible = False

    # Fora da área do bloco (em x ou y)
    if abs(e.pos.x) > bloco.size.x / 2 or abs(e.pos.y) > bloco.size.y / 2:
        eletrons_movels.remove(e)
        e.visible = False
```

5. SIMULADOR LIVRE

5.1. SIMULADOR LIVRE x SIMULADOR ARDUINO-DEPENDENTE

O simulador livre conta com uma simples interface que disponibiliza interações pelas quais se pode modificar os valores do campo magnético máximo e da distância, bem como

permite ligar e desligar o circuito ou o próprio campo magnético, sempre com a atualização da simulação a tempo.

Note que o código foi desenvolvido para ser utilizado no GlowScript, uma plataforma online gratuita muito útil para simulações com VPython. Pelo GlowScript transpilar de Python para Javascript, não apenas algumas funções embutidas mudam (como *time* para *clock*), mas também há um impedimento crítico: não é possível chamar funções a serem utilizadas no Arduino, ou seja, não há como integrar o modo Arduino-dependente com o livre, não ao menos com o GlowScript.

Por razão não bem compreendida, apesar de o código seguir os preceitos do VPython, tratados a seguir, verificou-se que executá-lo a partir do VSCode resulta em seu mal funcionamento e no consequente fechamento da janela *localhost* (manifesta pelo código) no navegador padrão do usuário. Para o caso de *notebook* Jupyter, a execução também é mal executada, mas pelo próprio funcionamento desse meio.

5.2. ESTRUTURA DO SIMULADOR LIVRE

Como o VPython exige para suas simulações 3D sempre haver algo em execução, o código está particularmente dividido em 5 setores: cena (corpo permanente), chaves, funções internas, funções dos botões e *while-mestre*. O arquivo da simulação livre contempla a amálgama total e independente da interface, da simulação em si e do gráfico.

O corpo permanente introduz o código imutável com o canvas/cena em que ocorrerá a visualização do efeito hall, caso contrário um canvas qualquer seria criado automaticamente. Além disso, há acompanhamento do título e do botão-mestre da simulação (isto é, o botão primeiro de todas as ações possíveis). Tenha em mente a padronização de espaços visuais expressos por “`esp_l = wtext(text=“
”)`”. O alinhamento (“`wtext(align=[...])`”) é também relevante para a organização visual da interface. Além disso, são definidas as constantes a serem utilizadas ao longo do código (figura 8).

Figura 8

```
Web VPython 3.2

##### Cena #####

scene = canvas(title="H.E.L.L.O World! <br>Efeito Hall com Campo Elétrico
Dinâmico", width=800, height=500, align="right")
esp_l = wtext(text="<br>", align="left")
modo_livre = button(bind=mestre_livre, text="Simulação Livre", disabled=False)

# ===== Constantes físicas =====
q = -1.6e-19      # carga do elétron (C)
m = 9.11e-31      # massa do elétron (Kg)
v0 = vector(0, 0, 0) # velocidade inicial no eixo x
dt = 2e-11        # passo de tempo (pequeno para alta aceleração)
i = 4.2e-3         # intensidade da corrente (A)

# ===== Parâmetros do sensor =====
s = 1.8e-3        # V/Gauss
w = 1e-3          # m (largura do condutor)
L = 5e-3          # comprimento lateral (x)
t = 1e-4          # espessura
sigma = 1e3       # condutividade do silício dopado (S/m)
n_portadores = 1e22 # densidade de portadores (semicondutor silício dopado)
```


A secção “chaves” (figura 9) é dedicada inteiramente à dinâmica do *while_mestre*, a qual pouco a seguir será discutida.

Figura 9

```
##### Chaves #####

## Gráfico Livre
inicializacao_livre = False
key_grafico_livre = False
ha_grafico_livre = False

## Simulação Geral
escopo_simulacao = False
key_simulacao = False
ha_objetos = False
```

Em “funções internas” (figura 9) se encontram recursos específicos a serem utilizados uma vez –o recebimento das variáveis selecionadas e a plotagem do gráfico sequencial – ou diversas vezes – a criação do elétron-partícula.

Figura 10

```
##### Funções Internas #####

# Criação de Elétrons
def criar_eletron():
    e = sphere(pos=vector(-5e-3, 0, 0), radius=0.1e-5,
                color=color.cyan, make_trail=True)
    e.vel = v0
    eletrons_moveis.append(e)

# Coleta das informações necessárias para a plotagem do gráfico
def dados_usuario(n, campo_magnetico_maximo, i, n_portadores, q, t):
    distancias = list(range(1, n + 1))

    """
    Para dipolos magnéticos tais que a distância é muito maior que qualquer dimensão de sua face
    aparente:
    B(d) = (μ₀/4π) * (μ/d**3),
    em que μ₀ é a permeabilidade magnética no vácuo e μ é o momento dipolar.
    Consideremos o campo magnético máximo com d = 0,01 m, isto é, 0,01 B.cm.
    """

    campos_por_distancia = {}
    dados_tensao = {}

    for d in distancias:
        B_intensidade = campo_magnetico_maximo / (d**3) # campo magnético em Tesla
        tensao = (B_intensidade * i) / (n_portadores * -q * t) # tensão Hall em Volts
        campos_por_distancia[d] = [B_intensidade]
        dados_tensao[d] = [tensao]

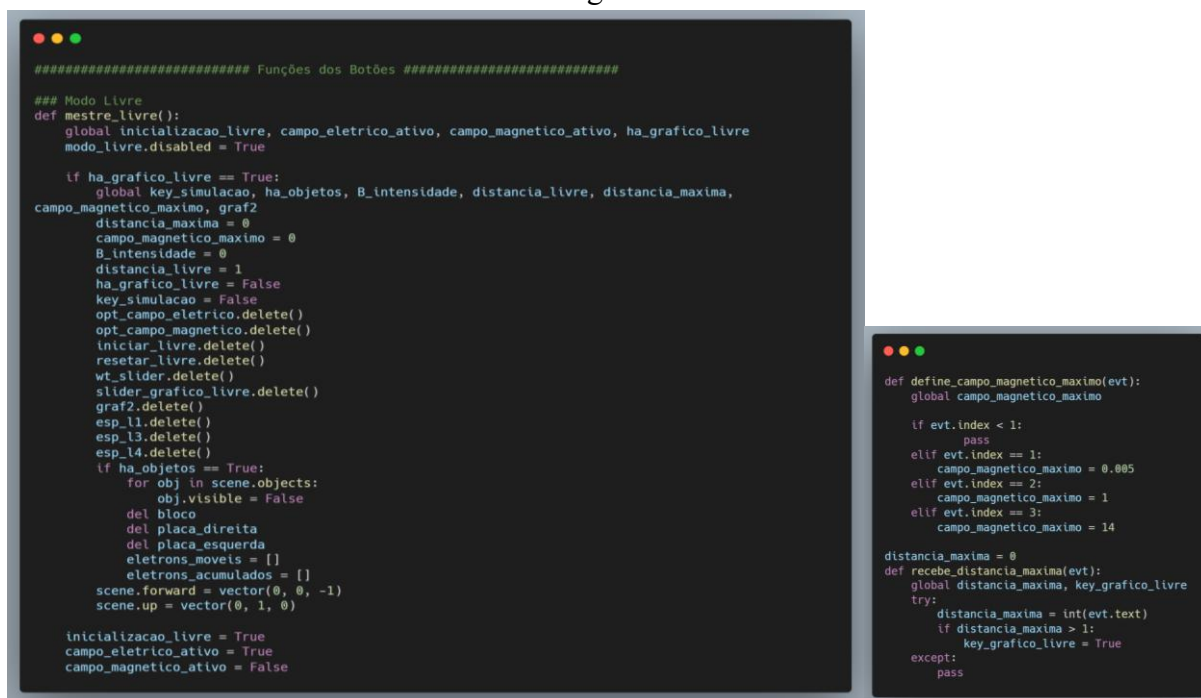
    return distancias, campos_por_distancia, dados_tensao

# Plotagem do gráfico Tensão Hall (V) x Distância (cm)
def plotar_graficos(distancias, campos_por_distancia, dados_tensao, campo_magnetico_maximo):
    global graf2
    graf2 = graph(title=f"Tensão Hall x Distância a {campo_magnetico_maximo}T máx.",
                  xtitle="Distância (cm)", ytitle="Tensão Hall (V)", width=400, height=300)
    curva2 = gcurve(color=color.blue, label="Dados")
    for d in distancias:
        tensao = dados_tensao[d][0] if dados_tensao[d] else 0
        curva2.plot(d, tensao)
```

No setor “funções dos botões” (figuras 10, 11, 12 e 13) se encontram todas as ações exercidas pelos botões, menus, *sliders* e *checkboxes* quando por exigência do usuário. Essas interações podem ocorrer em qualquer momento que convém, senão por necessidade de encadeamento de ações para manter a coesão do simulador.

Uma característica essencial dos interativos em VPython é que necessitam de apenas uma definição ao longo da execução e podem reclamar a qualquer momento as ações definidas em suas funções, ou seja, são independentes da temporalidade circular do *while-mestre*.

Figuras 10 e 11



A função *mestre_livre* define o botão-mestre “Simulador Livre”, de forma a ativar a chave *inicializacao_livre* e, assim, permitir o uso do simulador. Note que se apertado o botão após a já ativação da simulação, há, além das ações originais, uma reinicialização geral do simulador de modo a retornar todas as variáveis numéricas, todas as chaves (booleanas) e todos os objetos interativos e de cálculo e visualização ao estado original.

Vale destacar que, salvo essa anterior, todas as funções possuem nomes autoindicativos. Além disso, os argumentos “evt” ou “s” indicam informações fornecidas pelos próprios interativos. É o caso do menu de seleção de campo magnético, do espaço de inserção (paralelo ao “input” em Python geral) da distância máxima a ser possibilitada no *slider* de distância e na plotagem do gráfico, dos *checkboxes* liga-desliga do campo magnético e do campo elétrico (do circuito) e do slider que atualiza a distância e o campo magnético efetivo (e, por consequência, a tensão hall) ao modo do usuário.

Figuras 12 e 13

```

## Modo Livre -> Gráfico -> Simulação Livre
B_intensidade = 0
distancia_livre = 1
def atualiza_distancia_livre(s):
    global distancia_livre, B_intensidade, Tensao_hall, wt_slider, campo_magnetico_maximo, i,
    n_portadores, q, t
    distancia_livre = s.value
    B_intensidade = campo_magnetico_maximo / (distancia_livre**3) # campo magnético em Tesla
    Tensao_hall = (B_intensidade * i) / (n_portadores * q * t) # tensão Hall em Volts
    wt_slider.text = f"Dist. (cm): {distancia_livre:.2f}\n"

def iniciar_simulacao():
    global escopo_simulacao
    escopo_simulacao = True
    iniciar_livre.disabled = True
    resetar_livre.disabled = False

def resetar_simulacao():
    global key_simulacao, ha_objetos, campo_eletrico_ativo, campo_magnetico_ativo
    key_simulacao = False
    ha_objetos = False
    iniciar_livre.disabled = False
    resetar_livre.disabled = True
    opt_campo_eletrico.checked = True
    opt_campo_magnetico.checked = False
    for obj in scene.objects:
        obj.visible = False
    del bloco
    del placa_direita
    del placa_esquerda
    eletrons_movels = []
    eletrons_acumulados = []
    scene.forward = vector(0, 0, -1)
    scene.up = vector(0, 1, 0)

    campo_magnetico_ativo = True
    campo_magnetico_ativo = False

```

```

def checkbox_campo_eletrico(evt):
    global campo_eletrico_ativo
    if evt.checked:
        campo_eletrico_ativo = True
    else:
        campo_eletrico_ativo = False

def checkbox_campo_magnetico(evt):
    global campo_magnetico_ativo
    if evt.checked:
        campo_magnetico_ativo = True
    else:
        campo_magnetico_ativo = False

```

Por fim, é executado permanentemente o *while-mestre* (figuras 14 e 15), a fim de que o simulador não feche. Neste momento, a manipulação das chaves é essencial, pois se pretende executar apenas uma vez cada interativo, o gráfico para dados campo magnético máximo e distância máxima e as condições fixas da simulação, mas ainda é necessário atualizar os valores durante a simulação. Para tanto, houve divisão em 4 partes:

1. Permitida a inicialização do simulador pelo botão “Simulação Livre”, pede-se para selecionar o ímã e inserir alguma distância máxima acima de 1 (em centímetros);
2. Confirmada a inserção dessas dados, o gráfico é plotado de uma só vez (com seu próprio *laço for*);
3. Acabado o gráfico, o escopo da animação é definido;
4. Definido o escopo, os cálculos são realizados e um *laço for* atualiza a distância de cada elétron, sendo um novo elétron adicionado apenas a cada 0,5 segundo.

Ao fim de cada etapa, exceto a última, a chave para prosseguir à próxima é ativada e a chave da própria etapa é desativada, assim evitando repetições incorretas a cada ciclo do *laço while*.

6. CONCLUSÃO

O projeto H.E.L.L.O., world! — Hall Effect Lab for Learning and Observation alcançou seu objetivo de tornar o estudo do Efeito Hall mais acessível e interativo. Por meio da combinação entre medições experimentais com Arduino, visualizações gráficas em Python e simulações em VPython, foi possível representar esse fenômeno físico de forma clara, permitindo que usuários compreendam tanto sua base teórica quanto seus desdobramentos práticos.

Como aprimoramento futuro, propõe-se o desenvolvimento de uma interface unificada que reúna todos os módulos do projeto — incluindo coleta de dados teórica e experimental,

visualização gráfica e simulação — em um único ambiente. Essa centralização traria ainda mais fluidez à experiência do usuário e ampliaria o alcance didático do simulador, especialmente em contextos educacionais.

7. REFERÊNCIAS:

- HONEYWELL Sensing and Control. SS49E Linear Hall Effect Sensor. Version 2.00, May 2001. Disponível em: <https://sensing.honeywell.com/hall-effect-sensors-ss49e>. Acesso em: 06 jun. 2025.
- RYNDACK COMPONENTES. Usando um sensor de efeito Hall. Blog Ryndack, 20 mar. 2025. Disponível em: <https://blog.ryndackcomponentes.com.br/usando-um-sensor-de-efeito-hall/>. Acesso em: 06 jun. 2025.
- UNIVERSIDADE DE SÃO PAULO. Aula 2.2: propriedades elétricas – efeito Hall. São Paulo: EEL-USP, 2012. Disponível em: <https://sistemas.eel.usp.br/docentes/arquivos/5840726/LOM3035/Aula2.2-PropriedadesEletricas-EfeitoHall.pdf>. Acesso em: 6 jun. 2025.
- UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. O efeito Hall. Instituto de Física – FIS142, módulo 8. 2004. Disponível em: https://www.if.ufrgs.br/tex/fis142/mod08/m_s03.html. Acesso em: 24 jun. 2025.
- SANTOS, Ronilson Sousa. Análise experimental do efeito Hall em semicondutores. 2023. Trabalho de Conclusão de Curso (Graduação em Física) - Universidade Federal do Tocantins, Porto Nacional, 2023. Disponível em: <https://umbu.uft.edu.br/bitstream/11612/6356/1/RONILSON%20SOUSA%20SANTOS-TCC-F%3%8dSICA.pdf>. Acesso em: 11 jun. 2025.