GOVERNO DO ESTADO DO RIO DE JANEIRO
SECRETARIA DE ESTADO DE CIÊNCIA, TECNOLOGIA E INOVAÇÃO - SECTI
FUNDAÇÃO DE APOIO À ESCOLA TÉCNICA - FAETEC
ESCOLA TÉCNICA ESTADUAL FERREIRA VIANA - ETEFV COORDENAÇÃO DE ELETRÔNICA

Andrei Viegas Carvalho
Arthur Quintas de Almeida Melo
George David Pereira Alves Filho
Marcos Vinicius Soares de Sousa
Miguel Nascimento de Souza

TRAÇADOR DE CURVAS

Rio de Janeiro
2025

Andrei Viegas Carvalho
Arthur Quintas de Almeida Melo
George David Pereira Alves Filho
Marcos Vinicius Soares de Sousa
Miguel Nascimento de Souza

TRAÇADOR DE CURVAS

Trabalho de Conclusão de Curso apresentado à Coordenação de Eletrônica da ETE Ferreira Viana, da Fundação de apoio à Escola Técnica, como requisito parcial para a obtenção da habilitação profissional de Técnico de Nível Médio em Eletrônica sob a orientação do Professor Luis Henrique Monteiro de Castro e do Professor Marcelo de Almeida Duarte.

Rio de Janeiro
2025

FAETEC GOVERNO DO ESTADO RIO DE JANEIRO Secretaria de Ciência, Tecnologia e Inovação

Andrei Viegas Carvalho
Arthur Quintas de Almeida Melo
George David Pereira Alves Filho
Marcos Vinicius Soares de Sousa
Miguel Nascimento de Souza

TRAÇADOR DE CURVAS

Aprovada em : ____ / _____ / _____ Conceito: _____

_____
Prof. Luis Henrique M. de Castro
ETE Ferreira Viana
Orientador

_____
Prof. Marcelo de Almeida Duarte
ETE Ferreira Viana
Orientador

_____ Prof.
XXXXXXXXXXXXXX
ETE Ferreira Viana
ID:

_____ Prof.
XXXXXXXXXXXXXX
ETE Ferreira Viana
ID:

_____ Prof.
XXXXXXXXXXXXXX
ETE Ferreira Viana
ID:

Rio de Janeiro
2025

Sumário

RESUMO.....5
INTRODUÇÃO..... 6
LISTA DE COMPONENTES E ORÇAMENTO..... 8
CIRCUITO ELETRÔNICO.....9
DIÁRIO DE BORDO..... 10
REFERÊNCIAS.....12
CÓDIGO DO PROCESSING.....13
CÓDIGO DO ARDUINO.....24

RESUMO

CARVALHO, Andrei Viegas; MELO, Arthur Quintas de Almeida; FILHO, George David Pereira Alves; SOUSA, Marcos Vinicius Soares; SOUZA, Miguel Nascimento. *Traçador de Curvas*. 2025. Trabalho de Conclusão de Curso (em desenvolvimento) - Curso Técnico em Eletrônica, Escola Técnica Estadual Ferreira Viana, Fundação de Apoio à Escola Técnica, Rio de Janeiro, 2025.

A proposta deste trabalho é desenvolver um dispositivo que forneça as curvas características de tensão e corrente em componentes eletrônicos, como: resistores, diodos, LEDs e etc. Com o objetivo de visualizar os fenômenos elétricos relacionados a cada um deles de maneira dinâmica. A escolha da placa de Arduino se deve à sua facilidade de uso e ampla documentação, facilitando a aplicação técnica no projeto. O sistema realiza uma varredura controlada para exibir as curvas V x I dos componentes eletrônicos, com foco em contextos educacionais e experimentais. Devido isso, na montagem do hardware, trabalhamos com baixas potências, buscando a integridade do dispositivo e das peças sendo analisadas. No desenvolvimento do programa no software (Processing) o objetivo é realizar as operações e funções necessárias para montagem do gráfico.

Palavras-chave: Traçador de Curvas; Arduino; Característica; Software; Gráfico; Processing

INTRODUÇÃO

Antes do surgimento dos semicondutores, os traçadores de curvas utilizavam válvulas eletrônicas, como o Tektronix 570. Os primeiros traçadores de semicondutores também empregavam válvulas, já que os dispositivos semicondutores disponíveis na época não eram capazes de realizar todas as funções necessárias.

O Octopus, (conforme visto no anexo 2 de “referências” citado no fim do documento), é um traçador de curvas caseiro usado para testar componentes eletrônicos diretamente em placas de circuito impresso, sem precisar removê-los. Ele funciona com baixa tensão (cerca de 1V) e corrente limitada (menos de 1mA), sendo seguro até para componentes sensíveis como transistores, diodos e capacitores. Conectado a um osciloscópio, o dispositivo exibe padrões visuais que ajudam a identificar o tipo de componente, sua polaridade e possíveis falhas como curto-circuitos ou soldas frias. Fácil de construir com componentes simples, o Octopus se destaca por sua eficácia e praticidade, sendo muito útil na manutenção eletrônica.

Com o tempo, os traçadores evoluíram para sistemas totalmente em estado sólido, automatizados para facilitar a operação, capturar dados automaticamente e proteger tanto o equipamento quanto o dispositivo em teste (DUT). Atualmente, os traçadores modernos (como exemplo o Keysight B1505A) realizam três tipos principais de análise: corrente–tensão (I–V), capacitância–tensão (C–V) e corrente–tensão em regime transitório ultra-rápido.

No contexto do nosso trabalho propomos desenvolver um traçador de curvas baseado na plataforma Arduino, cuja função principal consiste em realizar a varredura controlada e apresentar tensão e corrente dos componentes eletrônicos em um gráfico I (V). Dessa forma, torna-se possível visualizar, de maneira prática, as curvas características de componentes eletrônicos.

Nosso objetivo é criar um sistema de registro de curvas de componentes eletrônicos, e promover sua aplicação em contextos educacionais e experimentais. Para tanto, foram definidos objetivos específicos a serem alcançados. Entre eles o desenvolvimento de um hardware adequado, para preservar o componente e o dispositivo; e desenvolver um programa no software (processing) capaz de realizar as operações e funções necessárias para montagem do gráfico.

A meta central consiste em criar e ampliar a quantidade de instrumentos de medição acessível para análise experimental de componentes eletrônicos em ambientes de teste e ensino. Portanto, a questão norteadora deste projeto pode ser formulada como: de que forma é

possível projetar e implementar um traçador de curvas a partir de componentes simples, que permita determinar os comportamentos de componentes eletrônicos básicos? Esse questionamento guia o desenvolvimento desse projeto. A justificativa deste trabalho baseia-se em ampliar o acervo de ferramentas educacionais disponíveis em laboratórios de eletrônica.

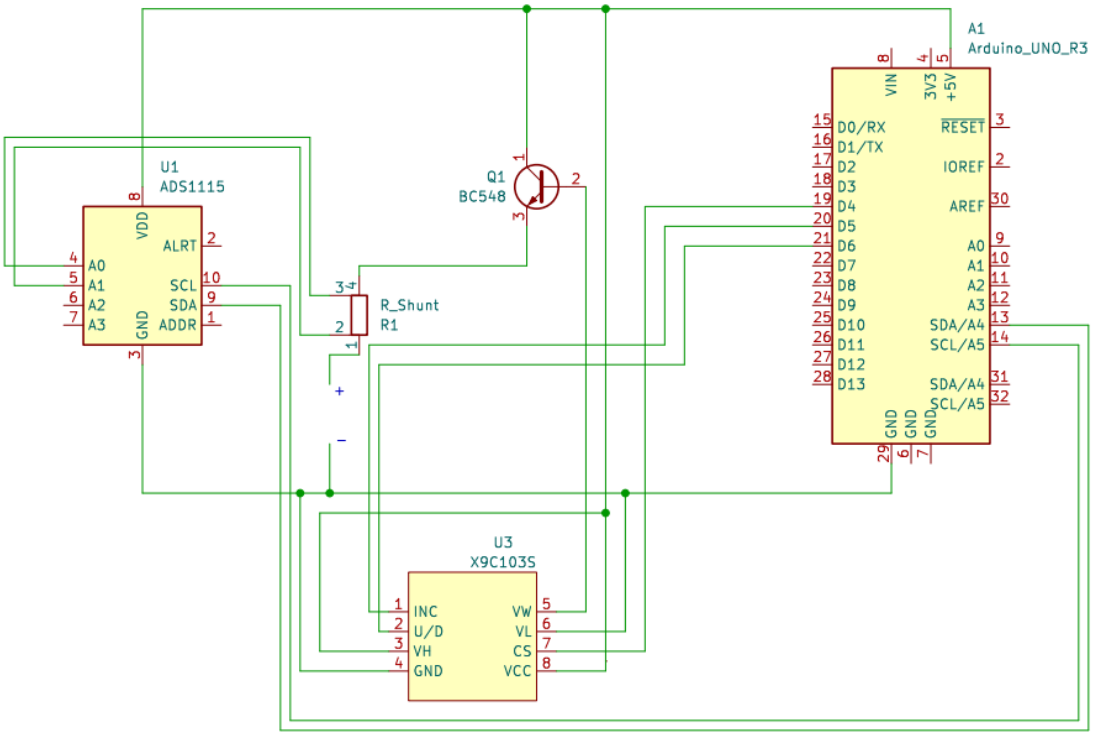
Nossa proposta é semelhante à do traçador de curvas Octopus. Apesar do propósito comum, as abordagens diferem significativamente. O Octopus é simples e direto, mas utiliza tensão alternada da rede elétrica, o que pode representar riscos e não permite o registro de dados. Já o traçador com Arduino adota uma solução digital, com controle preciso da varredura, maior segurança para os componentes e possibilidade de visualização gráfica via software (Processing), além do registro dos dados para análise posterior. Assim, o trabalho desenvolvido representa uma "modernização" do conceito original, mantendo a função essencial de análise, mas com vantagens em controle, segurança e aplicabilidade no ensino.

Não se incluem no escopo dispositivos de alta potência ou aplicações industriais em larga escala. O enfoque recai sobre componentes de uso comum em laboratórios didáticos, como diodos de silício, transistores de sinal e resistores, restringindo-se a faixas de tensão e corrente seguras para experimentação. Essa delimitação visa adequar o escopo às condições de ensino e aos recursos disponíveis no contexto acadêmico.

LISTA DE COMPONENTES E ORÇAMENTO

Componentes	Loja	Preço	Links
Arduino Uno R3	Eletrogate	R$71,16	LINK
Conversor (ADS1115)	Eletrogate	R$27,46	LINK
Potenciômetro (X9C103S)	Eletrogate	R$33,16	LINK
Resistor (330 Ω)	Eletrogate	R$0,86	LINK
Transistor (bc548)	Eletrogate	R$0,19	LINK

CIRCUITO ELETRÔNICO



DIÁRIO DE BORDO

Data	*Atividade Realizada*	*Anotações*
05/03	**Criação do grupo e definição do projeto**	Grupo formado por Arthur, Andrei, Miguel, Marcos e George. Início do documento oficial.
09/03	**Primeiro modelo do documento do projeto**	Documentação inicial com ideia e apresentação dos integrantes.
14/03	**Listagem dos materiais necessários**	Para iniciar o nosso projeto nos baseamos no projeto de Anuradha Gunawardhana. https://lkbrilliant.github.io/2022/02/06/Curve-Tracer/
16/03	**Correção do texto do documento**	
31/03	**Correção do texto do projeto**	Adição da lista de componentes, margem de preço (~R$200) e ideia do circuito.
06/04	**Correção do texto do documento**	Mais ajustes no documento.
07/04	**Sem avanços concretos**	Início dos estudos para desenvolvimento do traçador.
26/04	**Pesquisas sobre o tema**	Tomamos ciência do traçador de Newton C. Braga, para nos ajudar nas nossas pesquisas
28/04	**Fim das pesquisas e primeiras simulações**	Base definida com o livro “Experiências de eletrônica com Arduino e Processing”. Mas ainda tivemos dúvidas sobre o software para visualização.
04/05	**Correção do circuito, primeiro rascunho do código Arduino e correção do documento**	Início do desenvolvimento prático e revisamos o projeto inteiro, desde o software, hardware e documento.
19/05	**Simulação e edições no código do projeto**	Ajustes no código após testes.
20/05	**Edições no circuito**	Melhorias no circuito físico.
26/05	**Simulação, edições no código e no circuito**	Continuamos com os testes, atualizando código e circuito
02/06	**Edições no circuito**	Ajustes técnicos no circuito.
08/06	**Edições no código**	Refinamento do código.
15/06	**Edições no código e no circuito**	Mudança no código e no circuito com base nos testes.
16/06	**Simulação e edições no código do circuito**	Mudança no código e no circuito com base nos testes.
22/06	**Edições no código do projeto**	Otimização do código.
23/06	**Simulação física**	Primeira simulação física do traçador.

30/06	**Simulação física e edições no código**	Testes e ajustes no software.
12/07	**Edições no código**	Continuação dos ajustes.
14/07	**Simulação física e edições no código**	Estudo de potenciômetro e conversor A/D. Última semana antes do recesso.
04/08	**Edições no código**	Retorno ao projeto após recesso.
10/08	**Edições no código**	Continuação dos ajustes.
11/08	**Edições no código e circuito definitivo**	Finalização do circuito.
18/08	**Atualizações no código**	Adicionamos o diário de bordo ao documento do projeto, e recebemos sugestões do nosso orientador para a otimização da interface do programa.
25/08	**atualização do documento do projeto**	Atualizamos os códigos do documento e a previsão de preço.
01/09	**atualização do documento do projeto**	Atualizamos o código e o documento por completo.
08/09	**revisão dos códigos**	Revisamos e atualizamos os códigos do processing
15/09	**revisão dos códigos**	Atualizamos o código do processing, código e o circuito prontos.
20/09	**montagem do circuito na PCB**	Começamos a montar o circuito para apresentação, fora do protoboard.
30/10	**montagem do circuito na PCB e slides**	Nas últimas semanas nós nos dedicamos a montar o circuito na PCB, e os slides para a apresentação.
03/11	**montagem do circuito na PCB e slides**	Obtivemos progresso na montagem do circuito na PCB e aperfeiçoamos os slides da apresentação.
06/11	**montagem do circuito na PCB**	Continuamos na montagem do circuito, nenhuma atualização fora essa.
13/11	**montagem do circuito na PCB e slides**	Continuamos na montagem do circuito.
17/11	**Defesa do projeto**	Apresentamos o projeto pros professores e alunos do curso
24'/11	**Corrigimos alguns erros, e reparamos danos danos na PCB**	Corrigimos alguns erros de solda, e trocamos alguns fios que foram danificados.

REFERÊNCIAS

1. BRAGA, Newton. Traçador de curvas (INS152). Instituto NCB. Disponível em: https://newtoncbraga.com.br/usando-os-instrumentos/3650-ins152.html Acesso em: 30 abril. 2025.

2. LUDLOW, David. THE OCTOPOS. Georgian Bay Amateur Radio Club Inc. Jan. 1975. Disponível em: https://www.gbarc.ca/archive/nl1975may.pdf Acesso em: 02 Maio. 2025.

3. traçador de curvas. In: WIKIPÉDIA: a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Curve_tracer. Acesso em: 02 Maio. 2025.

4. SANTOS, Raul Morais dos; SILVA, Nuno Miguel Santos Pinto da. Experiências de Eletrônica Apoiadas em Arduino e Processing. Edição 1,Vila Real - Portugal: Núcleo Editorial e Gráfico dos SDB UTAD, 2013

CÓDIGO DO PROCESSING

```
import processing.serial.*;
import processing.pdf.*;

// Estados do programa
final int COLETANDO = 0;
final int VISUALIZANDO = 1;
int estado = VISUALIZANDO;

// Dados
Serial arduinoPort;
ArrayList<Float> tensao = new ArrayList<Float>();
ArrayList<Float> corrente = new ArrayList<Float>();
String portaSelecionada = "COM10";
boolean arduinoConectado = false;
int hoverIndex = -1;

// Escala
float escalaTensao = 5.0;
float escalaCorrente = 10.0;
boolean escalaDefinida = false;

// Coleta
int pontosColetados = 0;
int totalPontos = 100;

// Animação
int pontosPlotados = 0;
boolean animacaoConcluida = false;

// PDF
boolean salvandoPDF = false;
String caminhoPDF = "";

void setup() {
  size(1100, 700);
  smooth();
  iniciarSerial();
  textFont(createFont("Arial", 14));

  // Ponto inicial (0,0) apenas para referência
  tensao.add(0.0);
  corrente.add(0.0);
}
```

```
void draw() {
  if (salvandoPDF) {
    beginRecord(PDF, caminhoPDF);
    desenharAreaGraficoCompleto();
    endRecord();
    salvandoPDF = false;
    println("PDF salvo com sucesso: " + caminhoPDF);
    return;
  }

  background(240);

  desenharGrade();
  desenharEixos();
  desenharCurva();
  desenharPainelDados();
  desenharBotoes();
  desenharStatusConexao();

  if (hoverIndex >= 0) {
    desenharTooltip(hoverIndex);
    desenharLinhasGuia(hoverIndex);
  }

  if (!animacaoConcluida) {
    pontosPlotados++;
    if (pontosPlotados >= tensao.size() - 1) { // -1 porque ignoramos o primeiro ponto
      pontosPlotados = tensao.size() - 1;
      animacaoConcluida = true;
    }
  }
}

void desenharAreaGraficoCompleto() {
  float exportWidth = 800;
  float exportHeight = 600;
  float margin = 60;

  translate((width - exportWidth) / 2, (height - exportHeight) / 2);
  background(255);

  stroke(180, 220);
  strokeWeight(0.5);

  int divisoes = 10;
```

```
for (int i = 0; i <= divisoes; i++) {
  float v = (escalaTensao / divisoes) * i;
  float x = map(v, 0, escalaTensao, margin, exportWidth - margin);
  line(x, margin, x, exportHeight - margin);
}

for (int i = 0; i <= divisoes; i++) {
  float c = (escalaCorrente / divisoes) * i;
  float y = map(c, 0, escalaCorrente, exportHeight - margin, margin);
  line(margin, y, exportWidth - margin, y);
}

stroke(0);
strokeWeight(1.5);
line(margin, exportHeight - margin, exportWidth - margin, exportHeight - margin);
line(margin, exportHeight - margin, margin, margin);

textAlign(CENTER);
textSize(12);
fill(0);

for (int i = 0; i <= divisoes; i++) {
  float v = (escalaTensao / divisoes) * i;
  float x = map(v, 0, escalaTensao, margin, exportWidth - margin);
  line(x, exportHeight - margin, x, exportHeight - margin + 5);

  if (v == int(v)) {
    text(str(int(v)), x, exportHeight - margin + 25);
  } else {
    text(nf(v, 0, 1), x, exportHeight - margin + 25);
  }
}

textAlign(RIGHT);
for (int i = 0; i <= divisoes; i++) {
  float c = (escalaCorrente / divisoes) * i;
  float y = map(c, 0, escalaCorrente, exportHeight - margin, margin);
  line(margin - 5, y, margin, y);

  if (c == int(c)) {
    text(str(int(c)), margin - 10, y + 5);
  } else {
    text(nf(c, 0, 1), margin - 10, y + 5);
  }
}
```

```
  textAlign(CENTER);
  textSize(14);
  text("Tensão (V)", exportWidth / 2, exportHeight - margin + 45);

  pushMatrix();
  translate(margin - 35, exportHeight / 2);
  rotate(-HALF_PI);
  text("Corrente (mA)", 0, 0);
  popMatrix();

  // Ordena os pontos por tensão para o PDF (ignorando o primeiro ponto)
  ArrayList<PVector> pontosOrdenados = ordenarPontosPorTensao();

  if (pontosOrdenados.size() >= 2) {
    stroke(#2196F3);
    strokeWeight(2);
    noFill();
    beginShape();

    for (PVector ponto : pontosOrdenados) {
      float x = map(ponto.x, 0, escalaTensao, margin, exportWidth - margin);
      float y = map(ponto.y, 0, escalaCorrente, exportHeight - margin, margin);
      vertex(x, y);
    }
    endShape();

    fill(#2196F3);
    noStroke();
    for (PVector ponto : pontosOrdenados) {
      float x = map(ponto.x, 0, escalaTensao, margin, exportWidth - margin);
      float y = map(ponto.y, 0, escalaCorrente, exportHeight - margin, margin);
      ellipse(x, y, 4, 4);
    }
  }

  textAlign(CENTER);
  textSize(16);
  fill(0);
  text("Curva Característica", exportWidth / 2, margin - 25);
  textSize(12);
  text("Data: " + day() + "/" + month() + "/" + year() + " - " +
       nf(hour(), 2) + ":" + nf(minute(), 2), exportWidth / 2, margin - 5);
}

ArrayList<PVector> ordenarPontosPorTensao() {
```

```
  ArrayList<PVector> pontos = new ArrayList<PVector>();
  // Começa do índice 1 para ignorar o primeiro ponto (0,0)
  for (int i = 1; i < tensao.size(); i++) {
    pontos.add(new PVector(tensao.get(i), corrente.get(i)));
  }

  // Ordena por tensão (crescente)
  for (int i = 0; i < pontos.size() - 1; i++) {
    for (int j = i + 1; j < pontos.size(); j++) {
      if (pontos.get(i).x > pontos.get(j).x) {
        PVector temp = pontos.get(i);
        pontos.set(i, pontos.get(j));
        pontos.set(j, temp);
      }
    }
  }

  return pontos;
}

void desenharGrade() {
  stroke(180, 220);
  strokeWeight(1);

  int divisoes = 10;

  for (int i = 0; i <= divisoes; i++) {
    float v = (escalaTensao / divisoes) * i;
    float x = map(v, 0, escalaTensao, 80, width-280);
    line(x, 60, x, height-80);
  }

  for (int i = 0; i <= divisoes; i++) {
    float c = (escalaCorrente / divisoes) * i;
    float y = map(c, 0, escalaCorrente, height-80, 60);
    line(80, y, width-280, y);
  }
}

void desenharEixos() {
  stroke(0);
  strokeWeight(1.5);
  line(80, height-80, width-280, height-80);
  line(80, height-80, 80, 60);

  int divisoes = 10;
```

```
  textAlign(CENTER);
  textSize(11);
  fill(0);

  for (int i = 0; i <= divisoes; i++) {
    float v = (escalaTensao / divisoes) * i;
    float x = map(v, 0, escalaTensao, 80, width-280);

    line(x, height-80, x, height-75);

    if (v == int(v)) {
      text(str(int(v)), x, height-60);
    } else {
      text(nf(v, 0, 1), x, height-60);
    }
  }

  textAlign(RIGHT);
  for (int i = 0; i <= divisoes; i++) {
    float c = (escalaCorrente / divisoes) * i;
    float y = map(c, 0, escalaCorrente, height-80, 60);

    line(80, y, 85, y);

    if (c == int(c)) {
      text(str(int(c)), 75, y+5);
    } else {
      text(nf(c, 0, 1), 75, y+5);
    }
  }

  textAlign(CENTER);
  textSize(12);
  text("Tensão (V)", (width-200)/2, height-40);

  pushMatrix();
  translate(40, height/2-40);
  rotate(-HALF_PI);
  text("Corrente (mA)", 0, 0);
  popMatrix();
}

void desenharLinhasGuia(int index) {
  if (index <= 0 || index >= tensao.size()) return;
```

```
  float x = map(tensao.get(index), 0, escalaTensao, 80, width-280);
  float y = map(corrente.get(index), 0, escalaCorrente, height-80, 60);

  stroke(#FF9800, 150);
  strokeWeight(1);
  line(x, height-80, x, y);
  line(80, y, x, y);

  fill(#FF9800);
  noStroke();
  ellipse(x, height-80, 6, 6);
  ellipse(80, y, 6, 6);
}

void desenharCurva() {
  if (tensao.size() < 3) return; // Pelo menos 2 pontos além do (0,0)

  // Ordena os pontos por tensão (ignorando o primeiro ponto)
  ArrayList<PVector> pontosOrdenados = ordenarPontosPorTensao();

  // Desenha pontos individuais durante a animação
  fill(#2196F3);
  noStroke();
  for (int i = 0; i < min(pontosPlotados, pontosOrdenados.size()); i++) {
    PVector ponto = pontosOrdenados.get(i);
    ellipse(
      map(ponto.x, 0, escalaTensao, 80, width-280),
      map(ponto.y, 0, escalaCorrente, height-80, 60),
      5, 5
    );
  }

  // Conecta os pontos ordenados por tensão quando a animação terminar
  if (animacaoConcluida && pontosOrdenados.size() >= 2) {
    stroke(#2196F3);
    strokeWeight(2);
    noFill();
    beginShape();

    for (int i = 0; i < pontosOrdenados.size(); i++) {
      PVector ponto = pontosOrdenados.get(i);
      vertex(
        map(ponto.x, 0, escalaTensao, 80, width-280),
        map(ponto.y, 0, escalaCorrente, height-80, 60)
      );
    }
```

```
    endShape();
  }

  // Destaca o ponto sob o mouse
  if (hoverIndex > 1 && hoverIndex < tensao.size()) { // Começa do índice 2
    float x = map(tensao.get(hoverIndex), 0, escalaTensao, 80, width-280);
    float y = map(corrente.get(hoverIndex), 0, escalaCorrente, height-80, 60);

    fill(#FFC107);
    stroke(#FF9800);
    ellipse(x, y, 10, 10);
    noStroke();
  }
}

void desenharPainelDados() {
  float panelX = width - 260;
  float panelY = 60;
  float panelWidth = 220;
  float panelHeight = 60;

  fill(255, 240);
  stroke(180);
  rect(panelX, panelY, panelWidth, panelHeight, 8);

  fill(50);
  textAlign(LEFT);
  textSize(14);

  if (pontosColetados > 0) {
    float porcentagem = (pontosColetados * 100.0) / totalPontos;
    text("Concluído: " + nf(porcentagem, 0, 1) + "%", panelX + 15, panelY + 25);
    text("Pontos: " + pontosColetados + "/" + totalPontos, panelX + 15, panelY + 45);
  }
}

void desenharBotoes() {
  desenharBotao("Reset (R)", width-120, #2196F3);
  desenharBotao("PDF (E)", width-240, #F44336);
}

void desenharBotao(String texto, float x, color cor) {
  fill(cor);
  noStroke();
  rect(x, height-60, 100, 40, 6);
  fill(255);
```

```
  textAlign(CENTER, CENTER);
  text(texto, x + 50, height-40);
}

void desenharTooltip(int index) {
  if (index <= 1 || index >= tensao.size()) return; // Começa do índice 2

  float x = map(tensao.get(index), 0, escalaTensao, 80, width-280);
  float y = map(corrente.get(index), 0, escalaCorrente, height-80, 60);

  String info = "V: " + nf(tensao.get(index), 0, 3) + " V\n" +
                "I: " + nf(corrente.get(index), 0, 3) + " mA";

  float tooltipW = textWidth(info.split("\n")[0]) + 20;
  float tooltipH = 50;
  float tooltipX = constrain(x + 15, 20, width - tooltipW - 20);
  float tooltipY = constrain(y - 45, 20, height - tooltipH - 20);

  fill(255, 245);
  stroke(200);
  rect(tooltipX, tooltipY, tooltipW, tooltipH, 5);

  fill(0);
  textAlign(LEFT, CENTER);
  text(info, tooltipX + 10, tooltipY + tooltipH/2);
}

void desenharStatusConexao() {
  fill(arduinoConectado ? #4CAF50 : #F44336);
  noStroke();
  ellipse(width-250, 30, 12, 12);

  fill(80);
  textAlign(LEFT, CENTER);
  text("Arduino " + (arduinoConectado ? "conectado" : "desconectado"), width-230, 30);
}

void iniciarSerial() {
  try {
    if (arduinoPort != null) {
      arduinoPort.stop();
      arduinoPort = null;
      delay(1000);
    }
    arduinoPort = new Serial(this, portaSelecionada, 9600);
    arduinoPort.bufferUntil('\n');
```

```
    arduinoConectado = true;
    println("Conectado na porta: " + portaSelecionada);
  } catch (Exception e) {
    arduinoConectado = false;
    println("Erro na conexão serial: " + e.getMessage());
  }
}

void serialEvent(Serial port) {
  try {
    if (pontosColetados >= totalPontos) {
      return;
    }

    String data = port.readStringUntil('\n').trim();

    if (data != null && data.contains(";")) {
      String[] partes = split(data, ';');

      if (partes.length == 2) {
        float v = float(partes[0]);
        float i = float(partes[1]);

        // Usa o primeiro ponto apenas para definir a escala
        if (!escalaDefinida) {
          escalaTensao = ceil(v);
          escalaCorrente = ceil(i);
          if (escalaTensao < 1) escalaTensao = 1;
          if (escalaCorrente < 1) escalaCorrente = 1;
          escalaDefinida = true;
          println("Escala definida: " + escalaTensao + "V, " + escalaCorrente + "mA");
          return; // Não adiciona este ponto aos dados
        }

        // Adiciona pontos normais após definir a escala
        tensao.add(v);
        corrente.add(i);
        pontosColetados++;

        if (pontosColetados >= totalPontos) {
          animacaoConcluida = false;
          pontosPlotados = 0;
          println("Coleta concluída! Total de pontos: " + pontosColetados);
        }
      }
    }
```

```
  } catch (Exception e) {
    println("Erro ao processar dados: " + e.getMessage());
  }
}

void mouseMoved() {
  hoverIndex = -1;
  // Começa do índice 2 (ignora o ponto 0,0 e o primeiro ponto usado para escala)
  for (int i = 2; i < tensao.size(); i++) {
    float x = map(tensao.get(i), 0, escalaTensao, 80, width-280);
    float y = map(corrente.get(i), 0, escalaCorrente, height-80, 60);
    if (dist(mouseX, mouseY, x, y) < 8) {
      hoverIndex = i;
      break;
    }
  }
}

void keyPressed() {
  if (key == 'r' || key == 'R') {
    resetarColeta();
  } else if (key == 'e' || key == 'E') {
    selecionarLocalPDF();
  }
}

void mousePressed() {
  if (mouseX > width-120 && mouseX < width-20 && mouseY > height-60 && mouseY <
height-20) {
    resetarColeta();
  } else if (mouseX > width-240 && mouseX < width-140 && mouseY > height-60 && mouseY
< height-20) {
    selecionarLocalPDF();
  }
}

void selecionarLocalPDF() {
  selectOutput("Salvar gráfico como PDF", "arquivoPDFSelecionado");
}

void arquivoPDFSelecionado(File selection) {
  if (selection == null) return;

  String caminho = selection.getAbsolutePath();
  if (!caminho.toLowerCase().endsWith(".pdf")) {
    caminho += ".pdf";
```

```
  }

  caminhoPDF = caminho;
  salvandoPDF = true;
}

void resetarColeta() {
  if (arduinoPort != null) {
    arduinoPort.stop();
    arduinoPort = null;
  }

  tensao.clear();
  corrente.clear();
  pontosColetados = 0;
  pontosPlotados = 0;
  animacaoConcluida = false;
  escalaDefinida = false;
  escalaTensao = 5.0;
  escalaCorrente = 10.0;

  // Ponto inicial (0,0) apenas para referência
  tensao.add(0.0);
  corrente.add(0.0);

  iniciarSerial();

  println("Coleta resetada e Arduino reiniciado!");
}
```

CÓDIGO DO ARDUINO

```
#include <Wire.h>
#include <Adafruit_ADS1X15.h>

// Configuração do X9C103S (Potenciômetro Digital)
#define INC_PIN 2
#define UD_PIN  3
#define CS_PIN  4
const int totalSteps = 99; // Total de passos para X9C103S (10kΩ)

// Configuração do ADS1115
```

```
Adafruit_ADS1115 ads;
const int ADS1115_ADDRESS = 0x48;

const int R = 330; // Resistor shunt (Ω)

bool completed = false; // controla se a varredura já foi feita

void setup() {
  Serial.begin(9600);

  // Inicializa o potenciômetro digital
  pinMode(INC_PIN, OUTPUT);
  pinMode(UD_PIN, OUTPUT);
  pinMode(CS_PIN, OUTPUT);
  digitalWrite(CS_PIN, HIGH); // Desabilita inicialmente

  // Inicializa o ADS1115
  if (!ads.begin(ADS1115_ADDRESS)) {
    Serial.println("Falha ao inicializar ADS1115!");
    while (1);
  }
  ads.setGain(GAIN_TWOTHIRDS); // ±6.144V (FSR = 6.144V)

  // --- Leitura inicial na resistência mínima ---
  setPotentiometer(99); // resistência mínima
  delay(20); // estabilização

  float leituraA0 = ads.readADC_SingleEnded(0) * 0.0001875; // volts
  float leituraA1 = ads.readADC_SingleEnded(1) * 0.0001875; // volts

  float vComponente = leituraA1;
  float iComponente = (leituraA0 - vComponente) / R * 1000; // mA
  if (iComponente < 0) iComponente = 0;
  if (vComponente < 0) vComponente = 0;


  Serial.print(vComponente, 3);
  Serial.print(";");
  Serial.println(iComponente, 3);
```

```
  delay(500); // espera antes de iniciar varredura normal
}

void loop() {
  if (!completed) {
    for (int step = 0; step <= totalSteps; step++) {
      setPotentiometer(step);
      delay(20); // estabilização

      float leituraA0 = ads.readADC_SingleEnded(0) * 0.0001875; // volts
      float leituraA1 = ads.readADC_SingleEnded(1) * 0.0001875; // volts

      float vComponente = leituraA1;
      float iComponente = (leituraA0 - vComponente) / R * 1000; // mA
      if (iComponente < 0) iComponente = 0;
      if (vComponente < 0) vComponente = 0;

      Serial.print(vComponente, 3);
      Serial.print(";");
      Serial.println(iComponente, 3);

      delay(50);
    }
    completed = true; // encerra após varredura completa
  }
}

void setPotentiometer(int value) {
  value = constrain(value, 0, totalSteps);

  digitalWrite(CS_PIN, LOW);   // habilita
  digitalWrite(UD_PIN, LOW);   // direção: diminuir resistência

  // reset para passo 0
  for (int i = 0; i < totalSteps; i++) {
    digitalWrite(INC_PIN, HIGH);
    delayMicroseconds(1);
    digitalWrite(INC_PIN, LOW);
    delayMicroseconds(1);
  }
```

```
  // vai até o valor desejado
  digitalWrite(UD_PIN, HIGH);  // direção: aumentar resistência
  for (int i = 0; i < value; i++) {
    digitalWrite(INC_PIN, HIGH);
    delayMicroseconds(1);
    digitalWrite(INC_PIN, LOW);
    delayMicroseconds(1);
  }

  digitalWrite(CS_PIN, HIGH);  // desabilita e grava posição
}
```