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

Erick Pereira da Silva
Victor Hugo de Andrade Coelho
Arthur Campelo de Queiroz
João Pedro Silva Rodrigues

MARÉS ATMOSFÉRICAS

Rio de Janeiro
2025

Erick Pereira da Silva
Victor Hugo de Andrade Coelho
Arthur Campelo de Queiroz
João Pedro Silva Rodrigues

MARÉS ATMOSFÉRICAS

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

MARÉS ATMOSFÉRICAS

Aprovada em : ____ / _____ / _____ Conceito: _____

_____
Prof. XXXXXXXXXXXXXX
ETE Ferreira Viana

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

_____
Prof. Marcelo de Almeida Duarte
ETE Ferreira Viana
Orientador

ID:

_____
Prof. XXXXXXXXXXXXXX
ETE Ferreira Viana
ID:

_____
Prof. XXXXXXXXXXXXXX
ETE Ferreira Viana
ID:

Rio de Janeiro

2025

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

RESUMO

SILVA, Erick Pereira da; QUEIROZ, Arthur Campelo de; COELHO, Victor Hugo de Andrade; RODRIGUES, João Pedro Silva. *Marés Atmosféricas (em desenvolvimento)*. 2025. Trabalho de Conclusão de Curso - Curso Técnico em Eletrônica, Escola Técnica Estadual Ferreira Viana, Fundação de Apoio à Escola Técnica, Rio de Janeiro, 2025.

O projeto tem como objetivo desenvolver um sistema baseado em um microcontrolador e componentes eletrônicos projetados para monitorar variações climáticas e analisar oscilações na pressão atmosférica, conhecidas como marés atmosféricas. Permitindo a coleta de dados para análises meteorológicas e ambientais com mais perfeição e eficácia, auxiliando também no ramo de pesquisas e desenvolvimento na área especializada, com o intuito de propor uma melhor visão mais aprofundada desses fenômenos, a fim de oferecer melhorias para pesquisas dos fenômenos atmosféricos para os profissionais empenhados em estudar o assunto. O projeto apresentará um diferencial comparado aos demais por gerar gráficos, além de criar um banco de dados e sinalizar sobre a ocorrência dessas oscilações na pressão atmosférica. Utilizando a plataforma de prototipagem Arduino, aliado a componentes eletrônicos, permitindo a coleta de dados e uma caixa de proteção para os componentes eletrônicos. Levando em consideração que todos os componentes serão comprados de forma virtual. Dessa forma, o projeto apresenta um alto potencial para auxiliar pesquisas aplicadas, fornecendo sinalizações sobre alterações climáticas e permitindo um monitoramento ambiental mais preciso.

Palavras-chave: Arduino; Pressão Atmosférica; Marés Atmosféricas; Sensores.

INTRODUÇÃO

As marés atmosféricas são variações periódicas na pressão do ar causadas principalmente pela ação do Sol e da Lua sobre a atmosfera terrestre. Embora sejam menos conhecidas que as marés oceânicas, que movimentam os corpos d'água, essas oscilações influenciam a circulação dos ventos e a dinâmica do clima em escala global. Elas são fundamentais para o transporte de energia entre diferentes camadas da atmosfera, especialmente das partes mais baixas, como a troposfera, para regiões mais altas, como a mesosfera e termosfera.

Essas marés se dividem em dois grandes grupos: solares e lunares. Cada uma delas apresenta variações com períodos específicos – às solares ocorrem a cada 24 horas (diurnas) e 12 horas (semidiurnas), enquanto as lunares têm períodos de aproximadamente 24,8 horas e 12,4 horas, respectivamente. Os efeitos dessas marés são mais evidentes em regiões tropicais, onde as mudanças entre o dia e a noite são mais marcantes, favorecendo padrões de pressão mais definidos.

Pesquisas científicas, inclusive aquelas disponíveis no repositório Science Direct, mostram que o estudo das marés atmosféricas já acontece há mais de um século. Elas são detectadas tanto por registros de pressão ao nível do solo quanto por dados geomagnéticos. Outro fenômeno importante nesse contexto é a propagação de ondas gravitacionais na atmosfera, que interagem com as marés e com outros sistemas como ondas planetárias. Essas interações complexas ajudam a moldar o comportamento da atmosfera superior.

Diante disso, este projeto tem como principal objetivo medir e analisar as variações na pressão do ar relacionadas às marés solares e lunares, utilizando sensores de alta sensibilidade. Com os dados coletados, será possível identificar padrões regulares e entender melhor como esses fenômenos se manifestam localmente.

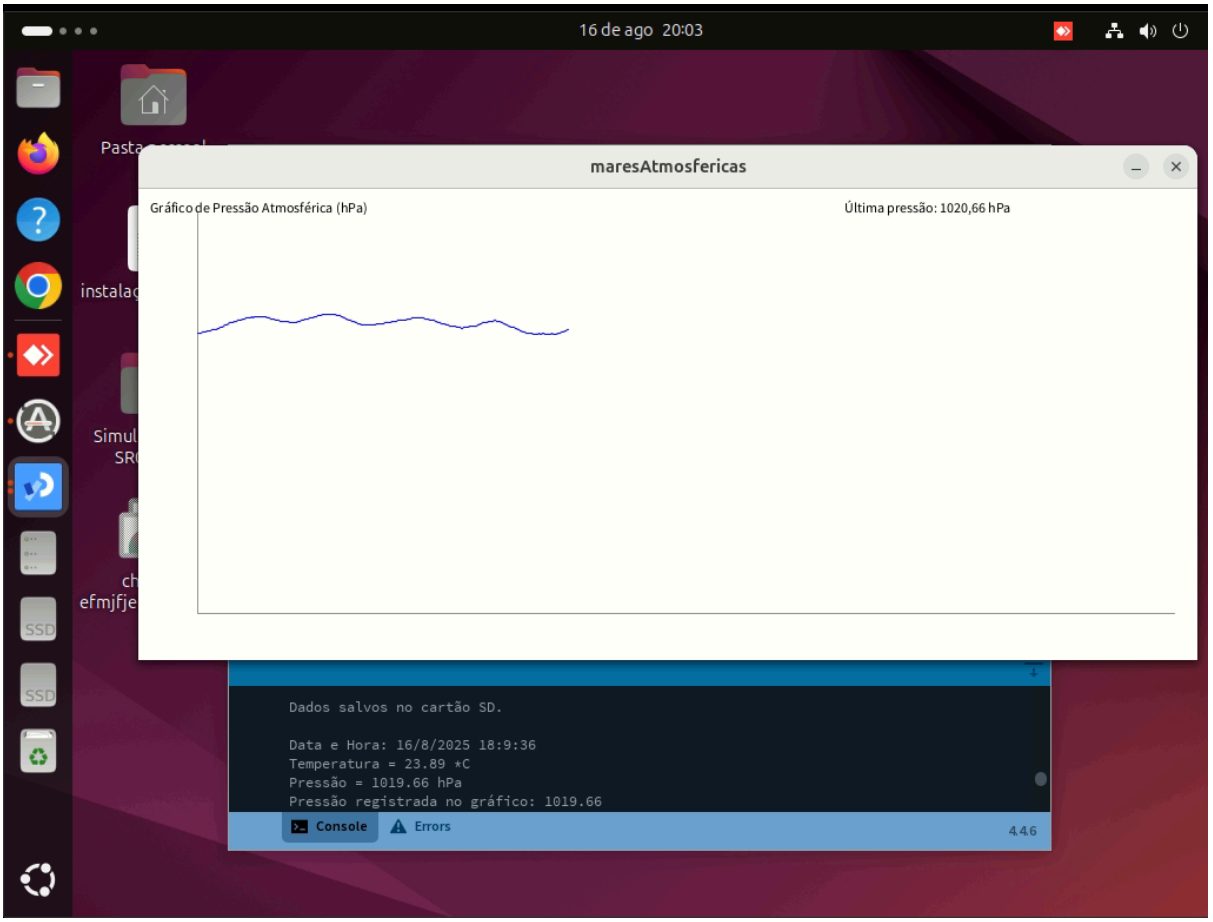
Compreender as marés atmosféricas é um passo importante para aprimorar o conhecimento sobre a circulação do ar e os processos climáticos em diferentes escalas. Ao analisar essas variações com mais detalhes, conseguimos não só observar como a energia se movimenta na atmosfera, mas também melhorar os modelos que ajudam a prever o tempo e o clima. Assim, este projeto busca contribuir de forma significativa para a ciência atmosférica, criando uma base sólida para estudos futuros e aplicações práticas.

Para isso, será desenvolvido um sistema automatizado baseado na plataforma Arduino, um microcontrolador poderoso e versátil, o sensor BME 280, conhecido por sua precisão na medição de pressão e temperatura, será utilizado para capturar os dados necessários e o Shield Data Logger com RTC DS1307 . Os dados serão armazenados automaticamente e poderão ser acessados em uma pasta que será criada automaticamente ao decorrer do funcionamento, exibindo o gráfico em tempo real, identificando tendências em caso de alterações significativas. Os materiais serão adquiridos por plataformas online, facilitando a montagem e reposição de peças.

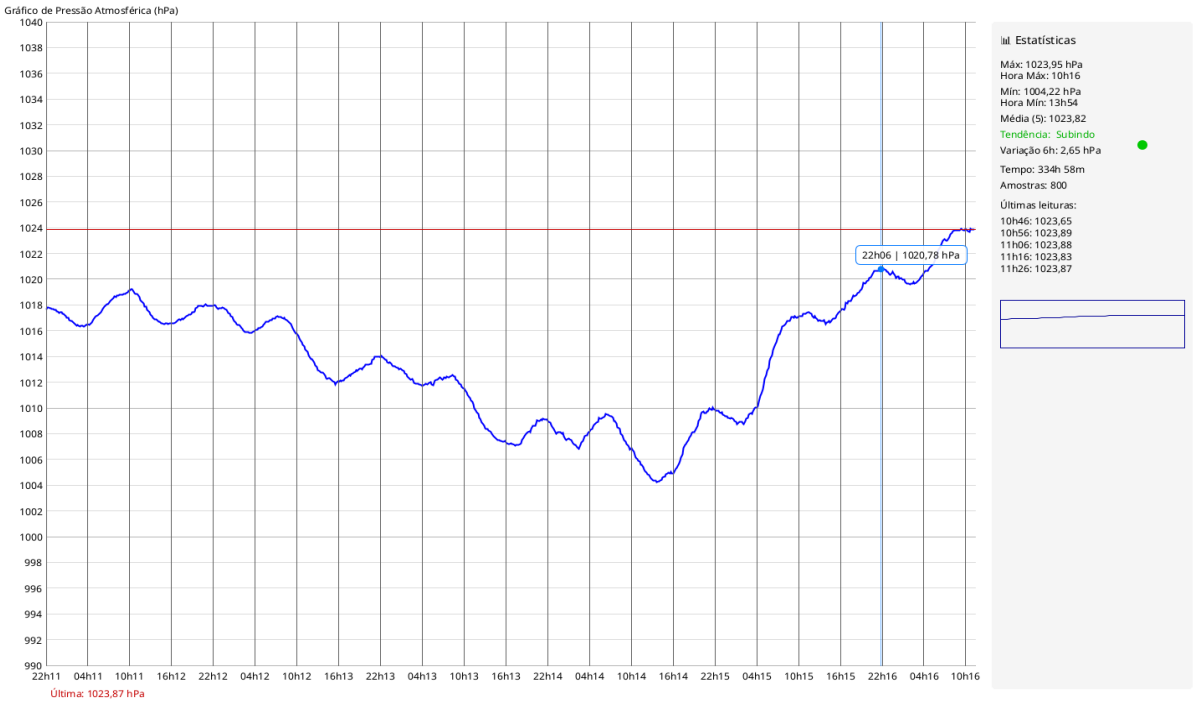
Por ser modular, o sistema poderá ser expandido ou adaptado conforme o projeto evolui, o que aumenta sua utilidade em diversas situações de estudo. Com isso, a proposta se apresenta como uma solução inovadora, acessível e eficiente para o monitoramento das marés atmosféricas. Ao unir tecnologia de baixo custo com precisão científica, este projeto apoia diretamente o avanço das pesquisas meteorológicas, promovendo uma melhor compreensão dos fenômenos que regem a nossa atmosfera.

GRÁFICO DO PROJETO

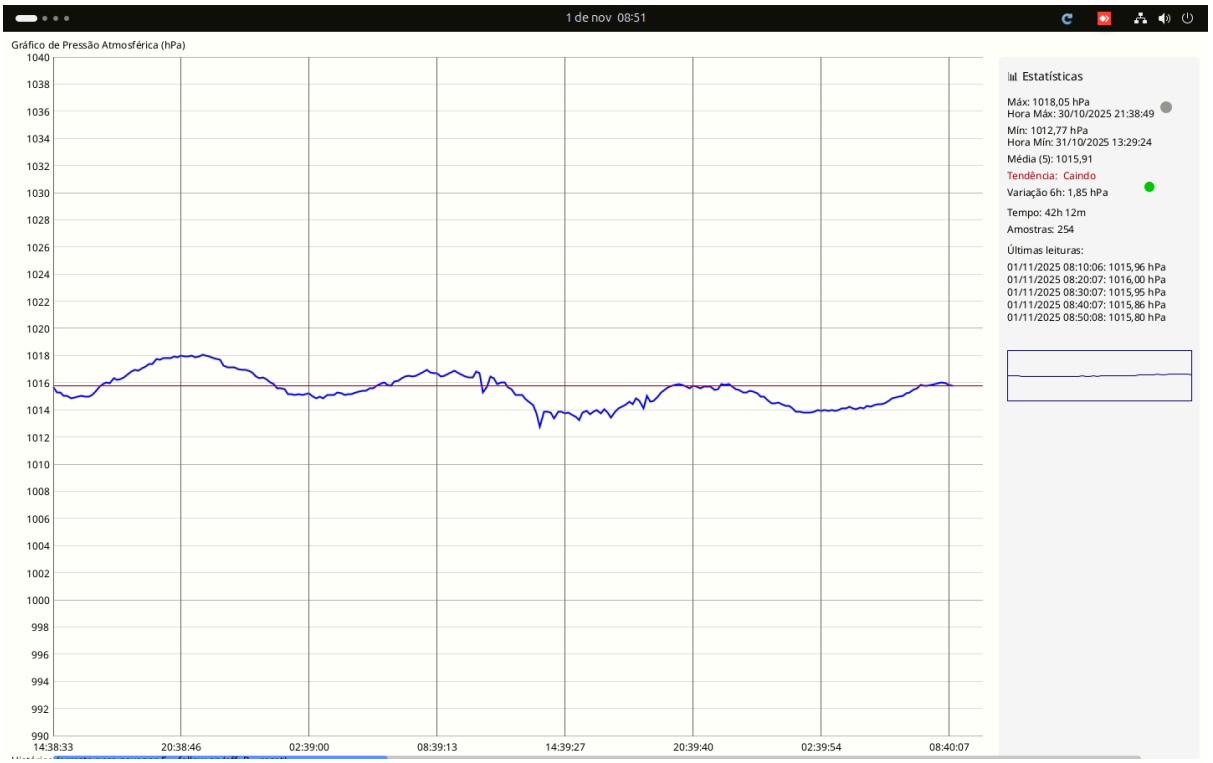
Este foi o primeiro esboço do nosso projeto, desenvolvido de forma simples. A partir de meados de agosto, iniciamos um processo contínuo de aprimoramento e inovação, especialmente na interação e na visualização do gráfico. Com essas melhorias, as medições tornaram-se mais precisas, apresentando valores exatos obtidos diretamente a partir do sensor conectado ao Arduino.



Este gráfico foi do mês de Setembro a Outubro, onde demos um grande salto trazendo muitas novidades, como uma nova interface melhorando a visualização das medições com linhas precisas, além das estatísticas gerais que foram apresentadas à direita.



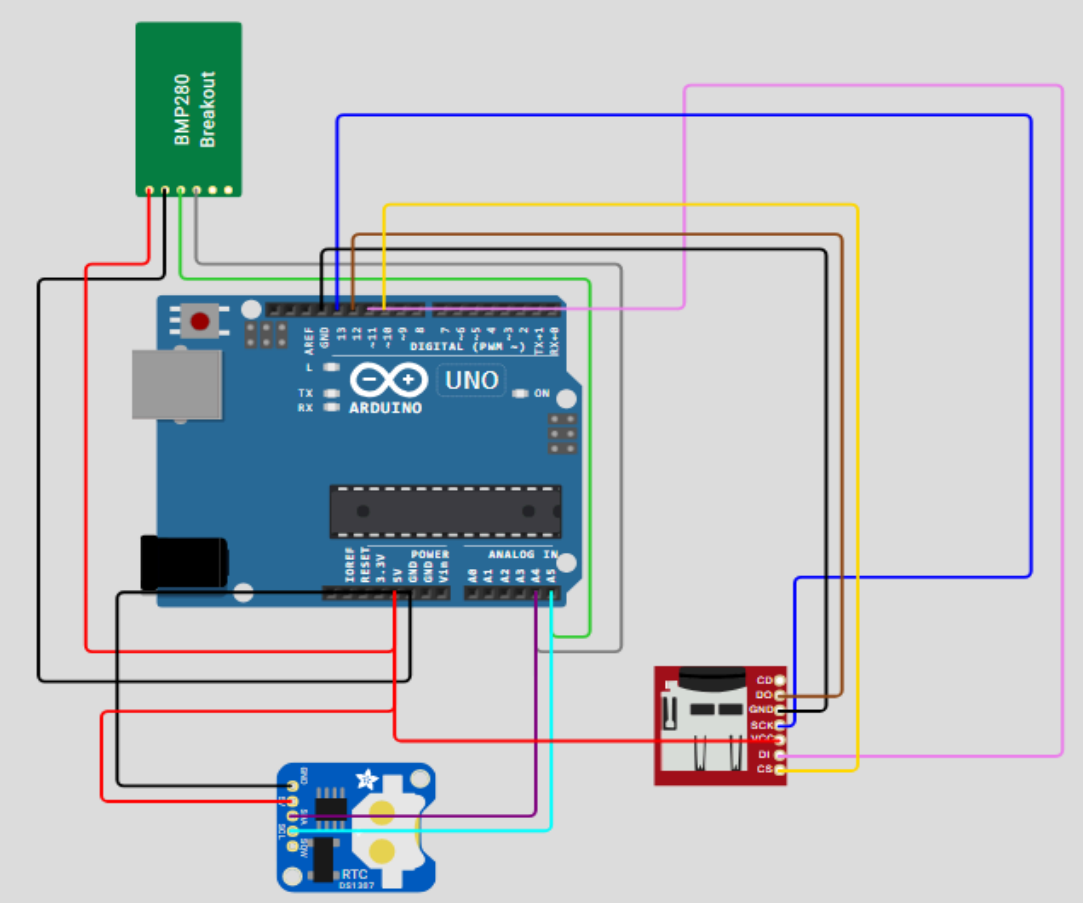
Se encontra na imagem abaixo o gráfico mais atualizado do projeto, que demonstra a evolução dos dados obtidos. Implementamos ciclos de medição configurados para ocorrer a cada seis horas, proporcionando maior precisão na coleta das informações. Além disso, foram adicionadas novas funcionalidades, como zoom, sliders e um painel com as estatísticas gerais do gráfico, tornando a análise mais completa e interativa em comparação com o gráfico anterior.



MATERIAIS UTILIZADOS

Componente	Preço	Frete	Link	Total
BME280	55,90	0,00	Mercado Livre link	55,90
Arduino Uno	89,90	11,83	Eletrogate link	101,73
Shield Data Logger com RTC DS1307	29,99	17,73	Casa da Robótica link	47,72
Custo Total	175,79	29,56	_	205,35

DIAGRAMA DE FUNCIONAMENTO



LINK PARA A SIMULAÇÃO

[Link]

DIÁRIO DE BORDO

O projeto iniciado no dia 17/02/2025 o qual foi nomeado de “marés atmosféricas” tem como principal objetivo dar suporte para o recente estudo das oscilações diárias que acontecem na atmosfera utilizando a plataforma de prototipagem do arduino uno r3 para o seu desenvolvimento.

Em seguida, no dia 24/02/2025 foi desenvolvido o resumo preliminar do projeto que apresentava a ideia de desenvolver todo o trabalho com base na utilização de microcontroladores e componentes eletrônicos para as medições das oscilações diárias da pressão atmosférica.

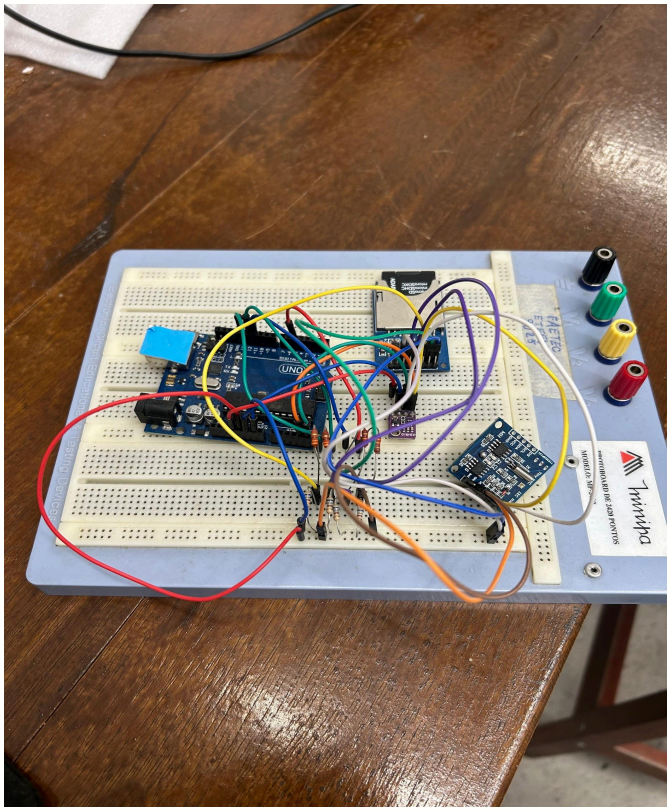
Recentemente no dia 26/05/2025 foi encontrada uma dificuldade em uma das etapas da realização do projeto que foi a etapa de montagem. Onde foram identificados erros/defeitos nos sensores BMP280 que não se comunicavam da maneira correta com o microcontrolador.
Foi realizada a compra de novos sensores para novas fases de teste deste trabalho.

A partir do mês de Junho ao mês de Setembro de 2025 aprimoramos nosso código Arduino removendo o cartão SD por conta da formatação, estava dando muitos erros e não estávamos conseguindo dar continuidade em nosso projeto. Mudamos nosso código Processing, melhoramos o gráfico incluindo mais informações e deixando-o mais completo de acordo com as medições que foram feitas ao longo do tempo. Adicionamos também uma sinalização, mostrando que o projeto segue fazendo suas medições..

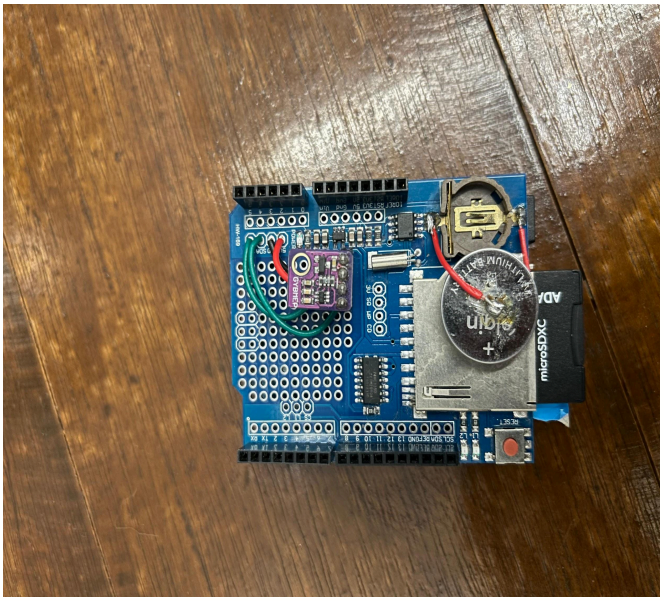
Em meados de Outubro, reta final para a conclusão do projeto, realizamos ajustes finais necessários. Entre as tarefas, instalamos novamente o cartão SD e a correção de problemas que ele havia apresentado anteriormente e implementando ao gráfico a opção de zoom e sliders para melhor visualização e interação.

Última semana de manutenção do projeto dias 3/11, 6/11 e 10/11. Mudamos praticamente toda a estrutura física do projeto nesses últimos dias. Descobrimos que o mau funcionamento do cartão veio a acontecer por conta do protoboard e corrigimos o erro implementando um Shield Data Logger com RTC e cartão SD, soldando a bateria do mesmo e os terminais do BME280 na Shield. Um ponto positivo desta mudança, além do funcionamento definitivo, é a redução no tamanho do projeto e uma visibilidade melhor.

PROJETO ANTIGO



PROJETO ATUAL



CONCLUSÃO

O desenvolvimento deste projeto representa um avanço significativo na busca por soluções acessíveis, eficientes e tecnicamente consistentes para o monitoramento das marés atmosféricas. Ao integrar a plataforma Arduino com sensores de alta precisão, como o BME280, e um sistema de registro automático baseado no Shield Data Logger, foi possível criar um dispositivo capaz de captar, armazenar e apresentar dados de forma clara e confiável. Essa capacidade de monitoramento contínuo, aliada à geração de gráficos e ao armazenamento estruturado das informações, amplia as possibilidades de análise e contribui diretamente para o entendimento das oscilações da pressão atmosférica.

Em um cenário em que a compreensão da dinâmica atmosférica se torna cada vez mais importante para aplicações científicas, ambientais e tecnológicas, este projeto demonstra grande potencial para auxiliar pesquisadores, estudantes e profissionais da área. A modularidade do sistema permite sua adaptação a diferentes contextos de estudo, possibilitando que futuras versões incorporem novos sensores, funcionalidades ou métodos de análise, tornando-o ainda mais completo e relevante.

Além disso, ao empregar tecnologia de baixo custo e alto valor educativo, o projeto reforça a importância de iniciativas que aproximem a pesquisa acadêmica da prática experimental. Essa abordagem fomenta o aprendizado, incentiva a inovação e contribui para a formação de profissionais mais preparados para as demandas contemporâneas do campo da meteorologia e da eletrônica.

Assim, conclui-se que a proposta aqui apresentada não apenas cumpre seu objetivo de monitorar e analisar as marés atmosféricas, como também estabelece as bases para estudos mais aprofundados sobre a dinâmica da atmosfera. Trata-se de um passo importante na construção de ferramentas que unem simplicidade, precisão e impacto científico, promovendo uma compreensão mais ampla dos fenômenos que moldam o comportamento do clima e da atmosfera terrestre.

REFERÊNCIAS

Science Direct, Google Acadêmico

AGRADECIMENTOS

Queremos expressar nossa mais sincera gratidão aos professores Luis Henrique Monteiro de Castro, Marcelo de Almeida Duarte e a todos os demais docentes da FAETEC – Escola Técnica Estadual Ferreira Viana, que, com dedicação, paciência e compromisso, nos orientaram e acompanharam ao longo de toda a nossa trajetória no curso técnico em Eletrônica.

Estendemos nossos agradecimentos à FAETEC, pelo constante incentivo à formação técnica de qualidade; à FAPERJ, pelo apoio essencial ao desenvolvimento científico e tecnológico realizado no CIBERLAB onde tivemos o prazer de realizar todo esse projeto; e ao CIBERLAB, pelo fornecimento dos recursos, da infraestrutura e do ambiente que tornaram possível a realização deste projeto de conclusão de curso.

Este trabalho é o resultado de um esforço coletivo — fruto da confiança, do apoio e da colaboração de todos que acreditaram em nosso potencial. A cada um, deixamos registrado nosso mais profundo respeito, gratidão e reconhecimento.

ANEXO A
CÓDIGO PROCESSING

```
import processing.serial.*;

Serial porta;
String leituraSerial = "";

final int BUFFER_SIZE = 850;

float[] pressao = new float[BUFFER_SIZE];
String[] tempo = new String[BUFFER_SIZE];
int count = 0;

int inicioMillis;

// Máximo e mínimo persistentes
float maxRegistrado = -9999;
float minRegistrado = 99999;
String horaMax = "--";
String horaMin = "--";

// Histórico para tendência
int HISTORICO_TENDENCIA = 5;

// ======= Captura automática =======
int intervaloPrint = 86400000; // 24 horas em milissegundos
int ultimoPrint = 0;

// ======= Luz indicadora piscando =======
boolean luzAtiva = true;
int ultimoPisca = 0;
int intervaloPisca = 500; // 0.5s

// ======= Controle de rolagem (slider) e zoom =======
float inicioVisivel = 0;           // índice inicial visível (pode ir até BUFFER_SIZE - pontosVisiveis)
float pontosVisiveis = 200;        // quantos pontos aparecem no gráfico (alterado por zoom)
float minPontosVisiveis = 10;      // zoom máximo (mostrar poucas amostras)
float maxPontosVisiveis = BUFFER_SIZE; // zoom mínimo (mostrar tudo)
boolean arrastandoSlider = false;
float sliderX = 0;
float sliderLargura = 200;
float sliderY;
boolean autoFollow = true;         // quando true, a view acompanha o último dado

// Layout do gráfico
int leftMargin = 60;
int topMargin = 30;
int bottomMargin = 60;
int panelWidth = 250;
int grafLeft, grafRight, grafTop, grafBottom;

void setup() {
  fullScreen(); // modo tela cheia
  println(Serial.list());
  if (Serial.list().length > 0) {
    // tenta abrir a primeira porta (ajuste se precisar)
    porta = new Serial(this, Serial.list()[0], 9600);
    porta.bufferUntil('\n');
  } else {
    println("Nenhuma porta serial encontrada.");
  }
  inicioMillis = millis();
  textFont(createFont("Arial", 12));

  grafLeft = leftMargin;
  grafRight = width - panelWidth - 20;
  grafTop = topMargin;
  grafBottom = height - bottomMargin;
}

void draw() {
  background(255);

  // Atualiza áreas do gráfico (caso a janela mude)
  grafRight = width - panelWidth - 20;
  grafBottom = height - bottomMargin;

  // Se autoFollow estiver ligado, posiciona inicioVisivel para acompanhar o último ponto
  if (autoFollow) {
    // queremos que o último ponto fique visível no lado direito do gráfico
    float novoInicio = count - pontosVisiveis;
    if (novoInicio < 0) novoInicio = 0;
    // permitir que novoInicio não ultrapasse BUFFER_SIZE - pontosVisiveis (área futura)
    inicioVisivel = constrain(novoInicio, 0, max(0, BUFFER_SIZE - pontosVisiveis));
  } else {
    // garante limites caso pontosVisiveis tenha mudado
    inicioVisivel = constrain(inicioVisivel, 0, max(0, BUFFER_SIZE - pontosVisiveis));
  }

  // --- Cor de fundo de alerta se pressão crítica ---
  if (count > 0) {
    float ultimo = pressao[max(0, count - 1)];
    if (ultimo < 1000) background(255, 230, 230); // alerta queda
    else if (ultimo > 1030) background(230, 255, 230); // alerta alta
  }
```

```
  // título
  fill(0);
  textAlign(LEFT);
  text("Gráfico de Pressão Atmosférica (hPa)", 10, 20);

  // eixos
  stroke(150);
  line(grafLeft, grafTop, grafLeft, grafBottom);
  line(grafLeft, grafBottom, grafRight, grafBottom);

  // eixo Y - escala 990 a 1040
  fill(0);
  textAlign(RIGHT, CENTER);
  for (int p = 990; p <= 1040; p += 2) {
    float y = map(p, 990, 1040, grafBottom, grafTop);
    noStroke();
    text(p, grafLeft - 5, y);
    if (p % 10 == 0) stroke(180);
    else stroke(220);
    line(grafLeft, y, grafRight, y);
  }

  // ======== GRÁFICO COM ROLAGEM (permite ver "futuro") ========
  stroke(0, 0, 255);
  strokeWeight(2);
  noFill();
  beginShape();

  int fimVisivel = int(inicioVisivel + pontosVisiveis);
  // fimVisivel pode ser maior que count (área futura) — só vamos plotar pontos que existam
  for (int i = int(inicioVisivel); i < fimVisivel; i++) {
    if (i < count) {
      float xPos = map(i, inicioVisivel, inicioVisivel + pontosVisiveis, grafLeft, grafRight);
      float yPos = map(pressao[i], 990, 1040, grafBottom, grafTop);
      vertex(xPos, yPos);
    }
    // se i >= count => ponto futuro (não plotar nada, deixa espaço em branco)
  }
  endShape();
  strokeWeight(1);

  // eixo X (adaptado à faixa visível)
  textAlign(CENTER);
  for (int j = int(inicioVisivel); j < fimVisivel; j += 36) {
    float xi = map(j, inicioVisivel, inicioVisivel + pontosVisiveis, grafLeft, grafRight);
    stroke(100);
    line(xi, grafBottom, xi, grafTop);
    fill(0);
    if (j < count && tempo[j] != null) {
      // Extrai apenas a parte da hora do formato "dd/MM/yyyy HH:mm:ss"
      String[] partesTempo = split(tempo[j], " ");
      String apenasHora = partesTempo.length > 1 ? partesTempo[1] : tempo[j];
      text(apenasHora, xi, grafBottom + 18);
    } else {
      // ponto futuro: mostra rótulo vazio (ou poderia mostrar "--:--:--")
      text("--:--:--", xi, grafBottom + 18);
    }
  }

  // linha última leitura (se estiver dentro da janela visível desenha a linha)
  if (count > 0) {
    float ultimo = pressao[count - 1];
    // só desenha linha se último estiver dentro do intervalo visível real (i < count)
    if ( (count - 1) >= inicioVisivel && (count - 1) < inicioVisivel + pontosVisiveis) {
      float yUlt = map(ultimo, 990, 1040, grafBottom, grafTop);
      stroke(200, 0, 0);
      line(grafLeft, yUlt, grafRight, yUlt);
      noStroke();
      fill(200, 0, 0);
      textAlign(LEFT);
      text("Última: " + nf(ultimo, 1, 2) + " hPa", grafLeft + 5, grafBottom + 40);
    } else {
      // se último estiver fora da janela (por exemplo quando você está vendo uma região futura),
      // mostra valor em um pequeno rótulo à esquerda com indicação que não está visível.
      noStroke();
      fill(120);
      textAlign(LEFT);
      text("Última (fora da vista): " + nf(ultimo, 1, 2) + " hPa", grafLeft + 5, grafBottom + 40);
    }
  }

  // --- Painel lateral ---
  int panelX = width - panelWidth;
  fill(245);
  noStroke();
  rect(panelX, topMargin, panelWidth - 10, height - topMargin - 30, 6);

  fill(0);
  textAlign(LEFT);
  textSize(14);
  text("📊 Estatísticas", panelX + 10, topMargin + 28);
  textSize(12);

  if (count > 0) {
    text("Máx: " + nf(maxRegistrado, 1, 2) + " hPa", panelX + 10, topMargin + 58);
    text("Hora Máx: " + horaMax, panelX + 10, topMargin + 72);
    text("Mín: " + nf(minRegistrado, 1, 2) + " hPa", panelX + 10, topMargin + 92);
    text("Hora Mín: " + horaMin, panelX + 10, topMargin + 106);

    float media = calcularMedia(HISTORICO_TENDENCIA);
    text("Média (" + HISTORICO_TENDENCIA + "): " + nf(media, 1, 2), panelX + 10, topMargin + 126);
```

```
  String tendencia = calcularTendencia();
  if (tendencia.contains("Subindo")) fill(0, 180, 0);
  else if (tendencia.contains("Caindo")) fill(200, 0, 0);
  else fill(80);
  text("Tendência: " + tendencia, panelX + 10, topMargin + 146);
  fill(0);

  float variacao6h = calcularVariacao6h();
  text("Variação 6h: " + nf(variacao6h, 1, 2) + " hPa", panelX + 10,
topMargin + 166);
 }

 // Indicador de estabilidade
 drawEstabilidade(panelX + 180, topMargin + 155);

 // Luz piscante (sistema ativo)
 if (millis() - ultimoPisca > intervaloPisca) {
  luzAtiva = !luzAtiva;
  ultimoPisca = millis();
 }
 if (luzAtiva) {
  fill(255, 0, 0);
 } else {
  fill(150);
 }
 noStroke();
 ellipse(panelX + 200, topMargin + 60, 14, 14);
 fill(0);

 int segundos = (millis() - inicioMillis) / 1000;
 int horas = segundos / 3600;
 int minutos = (segundos % 3600) / 60;
 fill(0);
 text("Tempo: " + horas + "h " + minutos + "m", panelX + 10,
topMargin + 190);
 text("Amostras: " + count, panelX + 10, topMargin + 210);

 text("Últimas leituras:", panelX + 10, topMargin + 235);
 int linhasMostrar = min(5, count);
 for (int i = 0; i < linhasMostrar; i++) {
  int idx = count - linhasMostrar + i;
  text(tempo[idx] + ": " + nf(pressao[idx], 1, 2) + " hPa", panelX +
10, topMargin + 255 + (i * 15));
 }

 drawSparkline(panelX + 10, topMargin + 350, panelWidth - 30,
60);

 // ==============================
 // 🔹 INTERAÇÃO COM O CURSOR (hover no gráfico)
 // ==============================
 if (mouseX > grafLeft && mouseX < grafRight && mouseY >
grafTop && mouseY < grafBottom && count > 0) {
  // Mapeia mouseX para índice flutuante dentro da janela visível
  float proporcaoX = map(mouseX, grafLeft, grafRight,
inicioVisivel, inicioVisivel + pontosVisiveis);
  int maxIndex = max(0, count - 1);
  int indiceHover = int(constrain(proporcaoX, 0, maxIndex));

  if (indiceHover >= 0 && indiceHover < count) {
   // Posição gráfica correspondente ao ponto
   float xHover = map(indiceHover, inicioVisivel, inicioVisivel +
pontosVisiveis, grafLeft, grafRight);
   float yHover = map(pressao[indiceHover], 990, 1040,
grafBottom, grafTop);

   // Linha vertical no ponto
   stroke(0, 120, 255, 120);
   line(xHover, grafTop, xHover, grafBottom);

   // Linha horizontal leve
   stroke(0, 120, 255, 80);
   line(grafLeft, yHover, grafRight, yHover);

   // Marcador no ponto
   fill(0, 120, 255);
   noStroke();
   ellipse(xHover, yHover, 8, 8);

   // Caixa de informações (tooltip)
   float sampleTextW = textWidth("00/00/0000 00:00:00 0000.00
hPa");
   float boxW = sampleTextW + 12;
   float boxH = 34;
   float boxX = xHover + 15;
   float boxY = yHover - boxH - 8;

   // Garante que o tooltip fique visível dentro da área do gráfico
   if (boxX + boxW > grafRight) boxX = xHover - boxW - 15;
   if (boxY < grafTop) boxY = grafTop + 8;

   fill(255, 255, 255, 230);
   stroke(0, 120, 255);
   rect(boxX, boxY, boxW, boxH, 6);

   fill(0);
   textAlign(LEFT, TOP);
   text(tempo[indiceHover], boxX + 6, boxY + 4);
   text(nf(pressao[indiceHover], 1, 2) + " hPa", boxX + 6, boxY +
18);
  }
 }

 // ==============================
 // 🔹 SLIDER: permite ver áreas FUTURAS (região sem dados)
 // ==============================
 sliderY = height - 30;
 float sliderTotalWidth = width - 140; // faixa ocupada pelo slider
 float sliderMaxX = 60 + sliderTotalWidth;
 // proporção do handle de acordo com quantos pontos são
visíveis no total BUFFER_SIZE
 float proporcaoHandle = pontosVisiveis / float(BUFFER_SIZE);
 sliderLargura = sliderTotalWidth * proporcaoHandle;
 // posição do handle de acordo com inicioVisivel (map de
0..(BUFFER_SIZE - pontosVisiveis))
 float maxInicio = max(0, BUFFER_SIZE - pontosVisiveis);
 if (maxInicio == 0) sliderX = 60;
 else sliderX = map(inicioVisivel, 0, maxInicio, 60, sliderMaxX -
sliderLargura);
```

```
  // desenha faixa e handle
  stroke(180);
  line(60, sliderY, sliderMaxX, sliderY);
  noStroke();
  fill(200);
  rect(60, sliderY - 6, sliderTotalWidth, 12, 4); // faixa de fundo
  fill(100, 150, 255);
  rect(sliderX, sliderY - 6, sliderLargura, 12, 4); // handle

  fill(0);
  textAlign(LEFT);
  text("Histórico (arraste para navegar, F = follow on/off, R =
reset)", 10, sliderY + 4);

  // ==============================
  // 🔹 CAPTURA AUTOMÁTICA DE TELA
  // ==============================
  if (millis() - ultimoPrint >= intervaloPrint) {
    salvarPrint();
    ultimoPrint = millis();
  }
}

// ---------- FUNÇÕES AUXILIARES ----------

float calcularMedia(int n) {
  if (count == 0) return 0;
  int inicio = max(0, count - n);
  float soma = 0;
  for (int i = inicio; i < count; i++) soma += pressao[i];
  return soma / (count - inicio);
}

String calcularTendencia() {
  if (count < 2) return "—";
  int idxRef = max(0, count - HISTORICO_TENDENCIA);
  float p1 = pressao[idxRef];
  float p2 = pressao[count - 1];
  if (p2 > p1 + 0.1) return "🔼 Subindo";
  if (p2 < p1 - 0.1) return "🔽 Caindo";
  return "🔘 Estável";
}

float calcularVariacao6h() {
  // supondo 6h = 36 amostras (se amostragem for 10 min por
exemplo) — mantenha conforme seu esquema
  if (count < 36) return 0;
  return pressao[count - 1] - pressao[count - 36];
}

void drawSparkline(int x, int y, int w, int h) {
  if (count < 2) return;
  stroke(0, 0, 150);
  noFill();
  rect(x, y, w, h);
  beginShape();
  for (int i = max(0, count - 50); i < count; i++) {
    float px = map(i, count - 50, count - 1, x, x + w);
    float py = map(pressao[i], 990, 1040, y + h, y);
    vertex(px, py);
  }
  endShape();
}

void drawEstabilidade(int x, int y) {
  if (count < 2) return;
  float variacao = abs(pressao[count - 1] - pressao[max(0, count -
10)]);
  noStroke();
  if (variacao < 0.5) fill(0, 200, 0);
  else if (variacao < 1.0) fill(255, 200, 0);
  else fill(255, 0, 0);
  ellipse(x, y, 12, 12);
}

// ---------- LEITURA SERIAL (mantive seu parser "press...hpa")
----------
void serialEvent(Serial p) {
  leituraSerial = p.readStringUntil('\n');
  if (leituraSerial != null) {
    leituraSerial = leituraSerial.trim();
    String linhaMinuscula = leituraSerial.toLowerCase();
    if (linhaMinuscula.contains("press") &&
linhaMinuscula.contains("hpa")) {
      try {
        String[] partes = split(linhaMinuscula, "=");
        if (partes.length > 1) {
          float valor_hpa = float(trim(partes[1].replace("hpa",
"").replace(",", ".")));
          if (count < BUFFER_SIZE) {
            pressao[count] = valor_hpa;
            tempo[count] = horaAtual();
            count++;
          } else {
            for (int i = 1; i < BUFFER_SIZE; i++) {
              pressao[i - 1] = pressao[i];
              tempo[i - 1] = tempo[i];
            }
            pressao[BUFFER_SIZE - 1] = valor_hpa;
            tempo[BUFFER_SIZE - 1] = horaAtual();
          }
          if (valor_hpa > maxRegistrado) { maxRegistrado =
valor_hpa; horaMax = horaAtual(); }
          if (valor_hpa < minRegistrado) { minRegistrado = valor_hpa;
horaMin = horaAtual(); }
        }
      } catch (Exception e) {
        println("Erro ao interpretar: " + leituraSerial + " -> " + e);
      }
    }
  }
}

String horaAtual() {
  return nf(day(), 2) + "/" + nf(month(), 2) + "/" + year() + " " +
         nf(hour(), 2) + ":" + nf(minute(), 2) + ":" + nf(second(), 2);
}

// ======== FUNÇÃO PARA CAPTURA DE TELA ========
```

```
void salvarPrint() {
  String dataHora = year() + "-" + nf(month(), 2) + "-" + nf(day(), 2)
+ "_"
                  + nf(hour(), 2) + "-" + nf(minute(), 2) + "-" +
nf(second(), 2);
  String nomeArquivo = "capturas/print_" + dataHora + ".png";
  saveFrame(nomeArquivo);
  println("📸 Captura salva em: " + nomeArquivo);
}

// ======== CONTROLE DO MOUSE E TECLADO (slider + zoom
+ follow + reset) ========
void mousePressed() {
  // detectar clique no handle do slider (área +-10 px)
  if (mouseY > sliderY - 12 && mouseY < sliderY + 12 && mouseX
> 50 && mouseX < width - 50) {
    // clique na área do slider
    // inicia arraste apenas se clicou sobre o handle
    if (mouseX >= sliderX && mouseX <= sliderX + sliderLargura) {
      arrastandoSlider = true;
      autoFollow = false; // ao arrastar, desliga auto-follow para que
o usuário possa navegar manualmente
    }
  }
}

void mouseDragged() {
  if (arrastandoSlider) {
    float sliderTotalWidth = width - 140;
    float sliderMaxX = 60 + sliderTotalWidth;
    // calcula nova sliderX com centro no mouse
    sliderX = constrain(mouseX - sliderLargura / 2, 60, sliderMaxX -
sliderLargura);
    float maxInicio = max(0, BUFFER_SIZE - pontosVisiveis);
    if (maxInicio > 0) {
      float novaPos = map(sliderX, 60, sliderMaxX - sliderLargura, 0,
maxInicio);
      inicioVisivel = constrain(novaPos, 0, maxInicio);
    } else {
      inicioVisivel = 0;
    }
  }
}

void mouseReleased() {
  arrastandoSlider = false;
}

void mouseWheel(MouseEvent event) {
  float e = event.getCount(); // positivo = scroll down (afastar),
negativo = scroll up (aproximar)
  // ajusta pontosVisiveis proporcionalmente
  float fator = 1 + e * 0.12;
  pontosVisiveis *= fator;
  pontosVisiveis = constrain(pontosVisiveis, minPontosVisiveis,
maxPontosVisiveis);
  // atualizar inicioVisivel para manter a mesma região central
  inicioVisivel = constrain(inicioVisivel, 0, max(0, BUFFER_SIZE -
pontosVisiveis));
  // se autoFollow ligado, aplica follow imediato
  if (autoFollow) {
    float novoInicio = count - pontosVisiveis;
    if (novoInicio < 0) novoInicio = 0;
    inicioVisivel = constrain(novoInicio, 0, max(0, BUFFER_SIZE -
pontosVisiveis));
  }
}

void keyPressed() {
  if (key == 'r' || key == 'R') {
    // reset zoom e posição (vai para início = 0, mostra padrão)
    pontosVisiveis = min(200, BUFFER_SIZE);
    inicioVisivel = max(0, count - pontosVisiveis);
    autoFollow = true;
  } else if (key == 'f' || key == 'F') {
    // toggle auto-follow
    autoFollow = !autoFollow;
    if (autoFollow) {
      float novoInicio = count - pontosVisiveis;
      if (novoInicio < 0) novoInicio = 0;
      inicioVisivel = constrain(novoInicio, 0, max(0, BUFFER_SIZE -
pontosVisiveis));
    }
  }
}
```

ANEXO B
CÓDIGO ARDUINO

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <RTClib.h>
#include <SPI.h>
#include <SD.h>

#define SEALEVELPRESSURE_HPA (1013.25)
#define SD_CS_PIN 10  // CS do cartão SD na shield logger

Adafruit_BME280 bme;
RTC_DS1307 rtc;
File arquivo;

String pastaAtual;

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

  if (!bme.begin(0x76)) {
    Serial.println("Erro ao inicializar BME280!");
    while (1);
  }

  if (!rtc.begin()) {
    Serial.println("Erro ao inicializar RTC!");
    while (1);
  }

  if (!SD.begin(SD_CS_PIN)) {
    Serial.println("Falha ao inicializar SD!");
    while (1);
  }

  Serial.println("SD inicializado com sucesso!");

  DateTime now = rtc.now();
  pastaAtual = "D" + String(now.year() % 100) +
               String(now.month()) +
               String(now.day());

  // Cria a pasta, se não existir
  if (!SD.exists(pastaAtual)) {
    if (SD.mkdir(pastaAtual)) {
      Serial.print("📁 Pasta criada: ");
      Serial.println(pastaAtual);
    } else {
      Serial.println("❌ Falha ao criar pasta!");
    }
  } else {
    Serial.print("📁 Pasta já existe: ");
    Serial.println(pastaAtual);
  }
}

void loop() {
  DateTime now = rtc.now();
  float press = bme.readPressure() / 100.0F;
  float temp = bme.readTemperature();
  float hum = bme.readHumidity();
  float alt = bme.readAltitude(SEALEVELPRESSURE_HPA);

  // Envia para Processing
  Serial.print("Pressao="); Serial.print(press, 2); Serial.println("
hPa");
  Serial.print("Temperatura="); Serial.print(temp, 2); Serial.println("
C");
  Serial.print("Umidade="); Serial.print(hum, 2); Serial.println(" %");
  Serial.print("Altitude="); Serial.print(alt, 2); Serial.println(" m");
  Serial.println();

  // Caminho completo do arquivo
  String caminhoArquivo = pastaAtual + "/LOG.CSV";

  arquivo = SD.open(caminhoArquivo, FILE_WRITE);

  if (arquivo) {
    arquivo.print(now.day()); arquivo.print('/');
    arquivo.print(now.month()); arquivo.print('/');
    arquivo.print(now.year()); arquivo.print(',');
    arquivo.print(now.hour()); arquivo.print(':');
    arquivo.print(now.minute()); arquivo.print(':');
    arquivo.print(now.second()); arquivo.print(',');
    arquivo.print(press, 2); arquivo.print(',');
    arquivo.print(temp, 2); arquivo.print(',');
    arquivo.print(hum, 2); arquivo.print(',');
    arquivo.print(alt, 2);
    arquivo.println();
    arquivo.close();

    Serial.print("✅ Gravado em ");
    Serial.println(caminhoArquivo);
  } else {
    Serial.print("❌ Erro ao abrir arquivo: ");
    Serial.println(caminhoArquivo);
  }

  delay(600000);
}
```