**================================================================================ ESPECIFICAÇÃO TÉCNICA COMPLETA MULTÍMETRO INTELIGENTE v1.0 Sistema de Mapeamento e Comparação de Placas Eletrônicas**

**ÍNDICE**

1. VISÃO GERAL DO SISTEMA
2. ARQUITETURA DO SOFTWARE
3. ESTRUTURA DE ARQUIVOS E PASTAS
4. FLUXO DE OPERAÇÃO COMPLETO
5. ESTADOS DO PROGRAMA
6. INTERFACE GRÁFICA (GUI)
7. SISTEMA DE PONTOS DE MEDIÇÃO
8. SISTEMA DE SALVAMENTO (.mip)
9. SISTEMA DE COMPARAÇÃO
10. ESPECIFICAÇÕES TÉCNICAS DETALHADAS
11. CLASSES E MÉTODOS PRINCIPAIS
12. FORMATOS DE DADOS
13. REQUISITOS NÃO-FUNCIONAIS
14. CASOS DE USO DETALHADOS
15. TRATAMENTO DE ERROS
16. COMUNICAÇÃO COM HARDWARE
17. TESTES E QUALIDADE
18. BUILD E DISTRIBUIÇÃO
19. FUTURAS IMPLEMENTAÇÕES

================================================================================

1. VISÃO GERAL DO SISTEMA ================================================================================

**PROPÓSITO**

Sistema desktop para técnicos de eletrônica que permite:

* Carregar foto de placa eletrônica
* Marcar visualmente pontos de medição com formas (círculo/retângulo)
* Registrar automaticamente valores medidos com multímetro (hardware próprio)
* Comparar placa defeituosa com placa de referência (boa)
* Identificar automaticamente pontos com valores divergentes

**PROBLEMA SOLUCIONADO**

* Elimina anotações manuais propensas a erro
* Padroniza processo de medição repetível
* Facilita comparação entre placas do mesmo modelo
* Cria registro visual e permanente das medições
* Acelera diagnóstico destacando divergências automaticamente

**TECNOLOGIAS**

* Linguagem: Python 3.11+
  + Performance melhorada
  + Suporte a typing moderno e dataclasses avançados
  + Match/case para estados
* GUI Framework: PyQt6
  + Model/View framework integrado
  + Sistema de sinais/slots para comunicação entre componentes
  + QGraphicsView para manipulação avançada de imagens
* Processamento de Imagem: Pillow (PIL)
  + Leve e suficiente para todas as operações necessárias
  + Suporte a múltiplos formatos de imagem
* Armazenamento: JSON + ZIP (formato .mip customizado)
  + metadata.json com versionamento de schema
  + points.json com dados dos pontos
  + image.png da placa
  + Checksums SHA-256 para integridade
* Gestão de Dependências: Poetry + requirements.txt
  + Poetry para desenvolvimento
  + requirements.txt exportado para distribuição simples
* Build: PyInstaller
  + Binários standalone para Windows e Linux
  + Empacotamento reprodutível
* Qualidade: Ruff/Black + mypy
  + Formatação consistente
  + Verificação de tipos estática

**TIPO DE MEDIÇÃO**

Sistema focado em MEDIÇÃO DE DIODO (condução reversa)

* Um único valor float por ponto (em Volts)
* Range típico: 0.0V a 2.0V
* Precisão: 3 casas decimais (0.001V)
* Possibilidade de expansão futura para outras medições

**================================================================================ 2. ARQUITETURA DO SOFTWARE**

**PADRÃO ARQUITETURAL**

Arquitetura em camadas com separação clara de responsabilidades:

┌─────────────────────────────────────────────────────────┐ │ CAMADA DE UI │ │ (Views, Widgets customizados, Dialogs) │ └──────────────────┬──────────────────────────────────────┘ │ Sinais/Slots PyQt6 ┌──────────────────▼──────────────────────────────────────┐ │ CAMADA DE CONTROLE │ │ (Controllers, Models Qt, Estado da Aplicação) │ └──────────────────┬──────────────────────────────────────┘ │ ┌──────────────────▼──────────────────────────────────────┐ │ CAMADA DE LÓGICA DE NEGÓCIO │ │ (Gerenciamento de pontos, Cálculos, Validações) │ └──────────────────┬──────────────────────────────────────┘ │ ┌──────────────────▼──────────────────────────────────────┐ │ CAMADA DE PROCESSAMENTO │ │ (Imagem, Serialização, I/O, Hardware) │ └─────────────────────────────────────────────────────────┘

**CAMADAS DETALHADAS**

1. INTERFACE (UI) - PyQt6 └─ QMainWindow com toolbars e status bar └─ QGraphicsView customizado para visualização de imagem └─ QTableView com modelo customizado para pontos └─ Dialogs para salvamento e configurações └─ Captura eventos de mouse/teclado └─ Renderização de overlays (pontos, IDs, preview)
2. CONTROLE E ESTADO └─ Gerenciamento de estados da aplicação └─ Coordenação entre componentes └─ Modelos Qt (QAbstractTableModel) └─ Controle de histórico (desfazer/refazer) └─ Validações de transições de estado
3. LÓGICA DE NEGÓCIO └─ Gerenciamento de pontos (adicionar, remover, editar) └─ Cálculo de diferenças percentuais └─ Aplicação de tolerâncias └─ Classificação de divergências └─ Validações de domínio
4. PROCESSAMENTO └─ Manipulação de imagens (Pillow) └─ Transformações (rotação, crop, redimensionamento) └─ Conversões PIL ↔ QPixmap └─ Serialização/Deserialização JSON └─ Compactação/Descompactação ZIP └─ Comunicação com hardware (thread separada) └─ Cálculo de checksums

**COMUNICAÇÃO ENTRE MÓDULOS**

Utiliza sistema de SINAIS do PyQt6 (pyqtSignal):

* Desacoplamento entre componentes
* Event-driven architecture
* Thread-safe para operações assíncronas
* Observer pattern para atualizações de UI

Exemplos de sinais:

# Em PointManager

point\_added = pyqtSignal(Point)

point\_removed = pyqtSignal(int) # ID do ponto

points\_cleared = pyqtSignal()

measurement\_completed = pyqtSignal(int, float) # ID, valor

# Em ImageEditor

image\_modified = pyqtSignal()

transformation\_applied = pyqtSignal(str) # tipo da transformação

# Em HardwareManager

measurement\_started = pyqtSignal()

measurement\_progress = pyqtSignal(int, int) # atual, total

measurement\_finished = pyqtSignal()

hardware\_error = pyqtSignal(str) # mensagem de erro

**THREADS E PROCESSAMENTO ASSÍNCRONO**

* Thread principal: UI e eventos
* QThread separada: Comunicação com hardware
* QThread separada: Operações de I/O pesadas (salvamento, carregamento)
* Sem bloqueio da UI durante operações longas
* Sinais para comunicação thread-safe
* Timeout e retentativas configuráveis para hardware

**================================================================================ 3. ESTRUTURA DE ARQUIVOS E PASTAS**

multimetro\_inteligente/ │ ├── pyproject.toml # Poetry configuration ├── requirements.txt # Dependências exportadas ├── README.md # Documentação do usuário ├── LICENSE # Licença do software │ ├── src/ │ ├── main.py # Ponto de entrada, QApplication │ │ │ ├── models/ # Modelos de dados │ │ ├── **init**.py │ │ ├── point.py # Dataclass Point │ │ └── project.py # Dataclass BoardProject │ │ │ ├── controllers/ # Lógica de controle │ │ ├── **init**.py │ │ ├── app\_controller.py # Controlador principal │ │ ├── state\_manager.py # Gerenciador de estados │ │ └── point\_manager.py # Gerenciamento de pontos │ │ │ ├── views/ # Componentes de interface │ │ ├── **init**.py │ │ ├── main\_window.py # QMainWindow principal │ │ ├── image\_viewer.py # QGraphicsView customizado │ │ ├── points\_table.py # QTableView + Model │ │ ├── toolbars.py # Toolbars dinâmicas │ │ └── dialogs.py # Dialogs de salvamento, etc │ │ │ ├── widgets/ # Widgets customizados │ │ ├── **init**.py │ │ ├── size\_slider.py # Slider com preview │ │ └── tolerance\_input.py # Input de tolerância │ │ │ ├── processing/ # Processamento │ │ ├── **init**.py │ │ ├── image\_processor.py # Operações PIL │ │ ├── transformations.py # Transformações │ │ ├── persistence.py # Salvamento/Carregamento .mip │ │ └── calculations.py # Cálculos de diferença │ │ │ ├── hardware/ # Comunicação com hardware │ │ ├── **init**.py │ │ ├── base.py # Interface base │ │ ├── serial\_driver.py # Driver serial │ │ └── simulator.py # Simulador para testes │ │ │ ├── utils/ # Utilitários │ │ ├── **init**.py │ │ ├── image\_utils.py # Conversões PIL ↔ QPixmap │ │ ├── validators.py # Validações │ │ └── config.py # Configurações │ │ │ └── resources/ # Recursos │ ├── **init**.py │ ├── icons/ # Ícones (se necessário) │ └── styles/ # Estilos Qt (se necessário) │ ├── tests/ # Testes automatizados │ ├── **init**.py │ ├── unit/ # Testes unitários │ │ ├── test\_point.py │ │ ├── test\_calculations.py │ │ └── test\_persistence.py │ │ │ ├── integration/ # Testes de integração │ │ ├── test\_project\_flow.py │ │ └── test\_hardware\_mock.py │ │ │ └── fixtures/ # Dados de teste │ ├── sample\_board.png │ └── sample\_project.mip │ ├── docs/ # Documentação │ ├── manual\_usuario.md │ ├── architecture.md │ └── hardware\_protocol.md │ └── scripts/ # Scripts auxiliares ├── build.py # Script de build ├── lint.sh # Linting └── package.sh # Empacotamento

**ARQUIVOS init.py OBRIGATÓRIOS**

Cada pasta de módulo DEVE ter **init**.py para imports funcionarem.

Exemplo models/**init**.py:

from .point import Point

from .project import BoardProject

\_\_all\_\_ = ['Point', 'BoardProject']

Exemplo controllers/**init**.py:

from .app\_controller import AppController

from .state\_manager import StateManager

from .point\_manager import PointManager

\_\_all\_\_ = ['AppController', 'StateManager', 'PointManager']

**================================================================================ 4. FLUXO DE OPERAÇÃO COMPLETO**

**FLUXO PRINCIPAL - MAPEAR PLACA BOA**

1. INÍCIO └─ Programa abre em ESTADO INICIAL └─ Janela vazia com 2 botões centralizados:
   * [📂 Abrir Imagem (Ctrl+O)]
   * [📁 Abrir Projeto (Ctrl+P)] └─ Status bar mostra: "Bem-vindo! Comece abrindo uma imagem ou projeto existente."
2. USUÁRIO CLICA "ABRIR IMAGEM" └─ QFileDialog de seleção de arquivo └─ Filtro: "Imagens (\*.png \*.jpg \*.jpeg \*.bmp \*.tiff \*.gif)" └─ Sistema carrega imagem com Pillow └─ Valida dimensões (mínimo 100x100, máximo 10000x10000) └─ Exibe informações: "Imagem carregada: 2048x1536 pixels" └─ Transição automática para ESTADO EDIÇÃO
3. ESTADO EDIÇÃO └─ Toolbar superior aparece:
   * [💾 Salvar (Ctrl+S)] [📸 Exportar (Ctrl+E)] [🆕 Novo]

└─ Toolbar dinâmica com ferramentas de edição:

* + [↩️ Desfazer (Ctrl+Z)] [↪️ Refazer (Ctrl+Shift+Z)]
  + [↻90° Rotacionar] [↻180° Inverter]
  + [⬌H Espelhar Horizontal] [⬍V Espelhar Vertical]
  + [✂️ Recortar] [📏 Redimensionar]
  + [✅ Concluir Edição]

AÇÕES POSSÍVEIS: └─ Rotacionar imagem (90° horário ou 180°) └─ Espelhar (horizontal ou vertical) └─ Recortar região de interesse:

* + Arrastar retângulo de seleção na imagem
  + Confirmar ou cancelar recorte └─ Redimensionar imagem:
  + Dialog com largura/altura ou percentual
  + Manter proporção (checkbox) └─ Desfazer/Refazer alterações (pilha de histórico) └─ Pan: clicar e arrastar move a imagem └─ Zoom: scroll do mouse/touchpad ajusta zoom └─ Ao clicar [✅ Concluir]:
  + Confirmação se houver muitas edições
  + Transição para ESTADO MARCAÇÃO

1. ESTADO MARCAÇÃO └─ Toolbar superior (persistente):
   * [💾 Salvar] [📸 Exportar] [🆕 Novo]

└─ Toolbar dinâmica - MODO CÍRCULO (padrão inicial): ┌────────────────────────────────────────────────────────┐ │ [⭕✓] [⬜] │ S [━━━●━━━━] W │ [✏️(E)] [🗑️] │ [✅ Concluir] │ │ Forma │ Tamanho: 20px │ Editar Limpar│ Próximo │ └────────────────────────────────────────────────────────┘

└─ Ou MODO RETÂNGULO: ┌──────────────────────────────────────────────────────────────┐ │ [⭕] [⬜✓] │ S [━●━] W │ A [━●━] D │ [↔️⇅] │ [✏️] [🗑️] │ [✅] │ │ Forma │ Altura 20│ Largura 30│ Trocar│ Editar Limpar│ │ └──────────────────────────────────────────────────────────────┘

AÇÕES POSSÍVEIS: └─ Escolher forma: [⭕ Círculo] ou [⬜ Retângulo]

└─ Ajustar tamanho:

* + Slider horizontal (movimento visual fluido)
  + Teclas W/S: aumentar/diminuir tamanho principal
  + Teclas A/D: ajustar largura (só retângulo)
  + Shift+Scroll: ajustar tamanho fino
  + Preview transparente aparece ao mover o mouse sobre a imagem

└─ Adicionar ponto:

* + Clicar na imagem no local desejado
  + Ponto aparece com forma, cor e ID
  + ID é sequencial (1, 2, 3...)
  + Cor padrão: vermelho (#FF0000, alpha 120)
  + Texto do ID: branco, negrito, centralizado

└─ Gerenciar pontos:

* + Ctrl+Z: desfazer último ponto
  + Ctrl+Shift+Z: refazer ponto
  + [🗑️ Limpar Todos]:
    - Confirmação: "Tem certeza? Esta ação não pode ser desfeita."
    - Opções: [Cancelar] [Limpar]
  + [✏️ Editar]:
    - Modo edição ativado
    - Clicar em ponto para selecioná-lo
    - Arrastar para reposicionar
    - Delete para remover selecionado
    - ESC para sair do modo edição

└─ Tabela lateral atualiza em tempo real: ┌────┬────────────┬────────────┬────────────┐ │ ID │ Referência │ Comparação │ Diferença │ ├────┼────────────┼────────────┼────────────┤ │ 1 │ -- │ -- │ -- │ │ 2 │ -- │ -- │ -- │ │ 3 │ -- │ -- │ -- │ └────┴────────────┴────────────┴────────────┘

└─ Pan: clicar e arrastar move a imagem └─ Zoom: scroll ajusta visualização

└─ Ao clicar [✅ Concluir]:

* + Validação: mínimo 1 ponto marcado
  + Se < 1: "Marque pelo menos 1 ponto antes de continuar"
  + Se OK: Transição para ESTADO MEDIÇÃO

1. ESTADO MEDIÇÃO └─ Toolbar superior (persistente):
   * [💾 Salvar] [📸 Exportar] [🆕 Novo]

└─ Toolbar dinâmica: ┌──────────────────────────────────────────────────────┐ │ [🔌 Conectar Hardware] │ [▶️ Iniciar] [⏸️ Pausar] │ │ Status: Desconectado │ Progresso: 0/0 │ └──────────────────────────────────────────────────────┘

CONEXÃO COM HARDWARE: └─ Clicar [🔌 Conectar Hardware] └─ Dialog de seleção:

* + Lista de portas seriais disponíveis
  + Opção "Modo Simulação" para testes
  + Botão [Testar Conexão] └─ Ao conectar:
  + Status muda para "Conectado - Pronto"
  + Botão [▶️ Iniciar] fica habilitado

SEQUÊNCIA DE MEDIÇÃO: └─ Clicar [▶️ Iniciar Medição]

└─ Sistema inicia sequência automatizada:

* + Ponto atual pisca em ROSA (#FF69B4)
  + Borda pulsante (animação 0.5s)
  + Label de instrução: "MEÇA O PONTO 1"
  + Aguarda valor do hardware
  + Ao receber valor:
    - Registra valor na coluna "Referência"
    - Para de piscar
    - Volta à cor vermelha normal
    - Avança para próximo ponto
  + Repete até último ponto

└─ Barra de progresso:

* + "Medindo pontos: 3/10 (30%)"
  + Tempo estimado: "~2 min restantes"

└─ Controles durante medição:

* + [⏸️ Pausar]: Para a sequência, permite retomar
  + [⏹️ Cancelar]: Interrompe e volta ao estado anterior
  + [⏭️ Pular Ponto]: Marca como "não medido", continua

└─ Ao concluir todas as medições:

* + Som de conclusão (opcional)
  + Notificação: "Medição concluída! 10/10 pontos medidos."
  + Botão [💾 Salvar] fica destacado
  + Toolbar muda para mostrar [📊 Modo Comparação]

1. SALVAR PROJETO └─ Disponível em qualquer estado após carregar imagem └─ Atalho: Ctrl+S

└─ Clicar [💾 Salvar Projeto]

└─ Dialog "Salvar Projeto" aparece: ┌─────────────────────────────────────────────┐ │ ╔═══════════════════════════════════════╗ │ │ ║ SALVAR PROJETO ║ │ │ ╚═══════════════════════════════════════╝ │ │ │ │ Nome do Projeto: [***] │ │ │ │ Modelo da Placa: [***] │ │ │ │ ☑ Placa 100% funcional? │ │ │ │ Descrição do problema: │ │ [***] │ │ [***] │ │ [\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_] │ │ │ │ Tolerância: [5.0] % │ │ │ │ [❌ Cancelar] [✅ Salvar] │ └─────────────────────────────────────────────┘

└─ VALIDAÇÕES:

* + Nome obrigatório (min 3 caracteres)
  + Modelo obrigatório (min 2 caracteres)
  + Se não funcional: descrição obrigatória (min 10 caracteres)
  + Tolerância: 0.1% a 50.0%

└─ Ao clicar [✅ Salvar]:

* + Validação dos campos
  + Se inválido: destaca campo e mostra erro
  + Se válido: QFileDialog "Salvar Como"
  + Filtro: "Projeto Multímetro (\*.mip)"
  + Sugere nome baseado em "Nome do Projeto"

└─ PROCESSO DE SALVAMENTO:

* + Cria arquivo temporário .mip.tmp
  + Compacta conteúdo:
    - image.png (cópia da imagem editada)
    - metadata.json (informações do projeto)
    - points.json (lista de pontos e medições)
  + Calcula checksums SHA-256
  + Adiciona checksums ao metadata.json
  + Renomeia .mip.tmp para .mip (operação atômica)
  + Marca projeto como "salvo" (sem alterações pendentes)
  + Atualiza título da janela com nome do projeto
  + Status bar: "Projeto salvo com sucesso!"

1. CARREGAR PROJETO └─ Estado INICIAL: Clicar [📁 Abrir Projeto (Ctrl+P)]

└─ QFileDialog de seleção de arquivo └─ Filtro: "Projeto Multímetro (\*.mip)"

└─ PROCESSO DE CARREGAMENTO:

* + Valida que é arquivo ZIP válido
  + Extrai para diretório temporário
  + Verifica presença de arquivos obrigatórios:
    - metadata.json
    - image.png
    - points.json
  + Lê metadata.json e valida schema\_version
  + Verifica checksums SHA-256
  + Se tudo OK: carrega dados
  + Se erro: dialog de erro com detalhes

└─ Sistema carrega:

* + Imagem da placa
  + Pontos marcados (forma, posição, tamanho)
  + Valores de referência (se existirem)
  + Valores de comparação (se existirem)
  + Configurações (tolerância, etc)
  + Metadados (nome, modelo, descrição)

└─ Restaura estado:

* + Se foi salvo em MARCAÇÃO: vai para MARCAÇÃO
  + Se foi salvo em MEDIÇÃO: vai para MEDIÇÃO
  + Toolbars apropriadas para o estado
  + Tabela preenchida com dados salvos

└─ Título da janela: "Multímetro Inteligente - [Nome do Projeto]" └─ Status bar: "Projeto carregado: 10 pontos, 8 medidos"

1. NOVO PROJETO └─ Clicar [🆕 Novo] └─ Atalho: Ctrl+N

└─ Se houver alterações NÃO salvas: ┌─────────────────────────────────────────────┐ │ ╔═══════════════════════════════════════╗ │ │ ║ ⚠️ ALTERAÇÕES NÃO SALVAS ║ │ │ ╚═══════════════════════════════════════╝ │ │ │ │ Existem alterações não salvas no projeto │ │ atual. O que deseja fazer? │ │ │ │ ┌─────────────────────────────────────┐ │ │ │ 🗑️ Descartar alterações │ │ │ │ e criar novo projeto │ │ │ └─────────────────────────────────────┘ │ │ │ │ ┌─────────────────────────────────────┐ │ │ │ ❌ Cancelar │ │ │ │ (voltar ao projeto atual) │ │ │ └─────────────────────────────────────┘ │ │ │ │ ┌─────────────────────────────────────┐ │ │ │ 💾 Salvar e criar novo │ │ │ │ (salva antes de criar novo) │ │ │ └─────────────────────────────────────┘ │ │ │ └─────────────────────────────────────────────┘

└─ Botões SEPARADOS verticalmente para evitar clique acidental

└─ Ao confirmar (qualquer opção que prossiga):

* + Limpa imagem carregada
  + Remove todos os pontos
  + Reseta configurações para padrão
  + Limpa histórico de desfazer/refazer
  + Limpa título da janela
  + Volta para ESTADO INICIAL
  + Status bar: "Pronto para novo projeto"

**FLUXO SECUNDÁRIO - COMPARAR PLACAS**

CENÁRIO: Técnico já mapeou placa BOA e quer comparar com DEFEITUOSA

1. CARREGAR PROJETO DA PLACA BOA └─ [📁 Abrir Projeto] → seleciona .mip da placa boa └─ Sistema carrega:
   * Imagem
   * Pontos marcados
   * Valores de referência já medidos
   * Metadados da placa boa └─ Vai para estado MEDIÇÃO (pois já foi medido) └─ Tabela mostra coluna "Referência" preenchida
2. MODO COMPARAÇÃO └─ Botão [📊 Comparar com Outra Placa] aparece na toolbar

└─ Técnico:

* + Troca a placa física no setup de medição
  + Mantém a mesma posição/orientação
  + Clica [📊 Comparar com Outra Placa]

└─ Sistema:

* + Limpa coluna "Comparação" da tabela
  + Inicia nova sequência de medição
  + Usa os MESMOS pontos já marcados
  + Mesma ordem de medição

└─ Sequência de medição (igual ao fluxo normal):

* + Ponto atual pisca em ROSA
  + Label: "MEÇA O PONTO 1 (COMPARAÇÃO)"
  + Aguarda valor do hardware
  + Registra na coluna "Comparação"
  + Avança para próximo

1. CÁLCULO AUTOMÁTICO DE DIFERENÇAS └─ Para cada ponto medido:
   * Calcula diferença: (Comparação - Referência) / Referência \* 100
   * Atualiza coluna "Diferença" em tempo real
   * Exemplo:
     + Ref: 0.450V, Comp: 0.456V → +1.3%
     + Ref: 0.450V, Comp: 0.120V → -73.3%
2. VISUALIZAÇÃO DE DIFERENÇAS └─ CORES AUTOMÁTICAS baseadas em tolerância:
   * Diferença ≤ tolerância: COR NORMAL (vermelho)
   * Diferença > tolerância: COR ROSA (#FF69B4)

└─ Na IMAGEM:

* + Pontos OK: círculos/retângulos vermelhos
  + Pontos divergentes: círculos/retângulos ROSA

└─ Na TABELA:

* + Linhas OK: fundo branco
  + Linhas divergentes: fundo rosa claro (#FFB6C1)

└─ Exemplo de tabela após comparação: ┌────┬────────────┬────────────┬────────────┐ │ ID │ Referência │ Comparação │ Diferença │ ├────┼────────────┼────────────┼────────────┤ │ 1 │ 0.450V │ 0.456V │ +1.3% │ ← Branco (OK) │ 2 │ 0.450V │ 0.120V │ -73.3% │ ← Rosa (Problema!) │ 3 │ 0.380V │ 0.382V │ +0.5% │ ← Branco (OK) │ 4 │ 0.520V │ 0.580V │ +11.5% │ ← Rosa (Problema!) │ 5 │ 0.390V │ 0.392V │ +0.5% │ ← Branco (OK) └────┴────────────┴────────────┴────────────┘

1. AJUSTE DE TOLERÂNCIA └─ No cabeçalho da tabela: ┌──────────────────────────────────────────┐ │ 📋 Pontos [10] Tolerância: [5.0] % [✓] │ └──────────────────────────────────────────┘

└─ Usuário pode:

* + Editar valor da tolerância (0.1% a 50.0%)
  + Clicar [✓ Aplicar] (ou Enter)
  + Sistema recalcula cores imediatamente
  + Pontos e tabela atualizam em tempo real

1. ANÁLISE E DIAGNÓSTICO └─ Status bar mostra resumo: "Comparação: 8 OK, 2 Diferentes (tolerância: 5.0%)"

└─ Técnico:

* + Identifica visualmente pontos ROSA
  + Foca nesses pontos para diagnóstico
  + Verifica componentes próximos

└─ Opções de exportação:

* + [📸 Exportar Imagem]: PNG com pontos coloridos
  + [📊 Exportar Dados]: CSV ou JSON com resultados

1. SALVAR COMPARAÇÃO └─ [💾 Salvar] atualiza o projeto .mip └─ Mantém valores de referência originais └─ Adiciona valores de comparação └─ Preserva histórico completo

**================================================================================ 5. ESTADOS DO PROGRAMA**

**DIAGRAMA DE ESTADOS**

┌─────────────┐

│ INICIAL │ ← Início do programa

└──────┬──────┘

│

├─── [Abrir Imagem] ────────→ ┐

│ ↓

│ ┌─────────────┐

│ │ EDIÇÃO │

│ └──────┬──────┘

│ │

│ [Concluir Edição]

│ │

│ ↓

│ ┌─────────────┐

│ │ MARCAÇÃO │

│ └──────┬──────┘

│ │

│ [Concluir Marcação]

│ │

│ ↓

│ ┌─────────────┐

│ │ MEDIÇÃO │

│ └──────┬──────┘

│ │

│ [📊 Modo Comparação]

│ │

│ ↓

│ ┌─────────────┐

│ │ COMPARAÇÃO │

│ └─────────────┘

│

└─── [Abrir Projeto] ──→ (Vai para estado salvo)

[🆕 Novo Projeto] de qualquer estado → INICIAL

**DESCRIÇÃO DOS ESTADOS**

1. ESTADO INICIAL
   * Tela vazia com botões de ação
   * Nenhuma imagem carregada
   * Nenhum ponto marcado
   * Aguardando ação do usuário

Ações disponíveis:

* + [📂 Abrir Imagem]
  + [📁 Abrir Projeto]

1. ESTADO EDIÇÃO
   * Imagem carregada e exibida
   * Ferramentas de edição disponíveis
   * Histórico de transformações
   * Pan e zoom ativos

Ações disponíveis:

* + Rotacionar, espelhar, recortar, redimensionar
  + Desfazer/Refazer
  + [✅ Concluir Edição]
  + [💾 Salvar] [📸 Exportar] [🆕 Novo]

1. ESTADO MARCAÇÃO
   * Imagem finalizada (não pode mais editar)
   * Ferramentas de marcação ativas
   * Preview de ponto ao mover mouse
   * Tabela de pontos visível

Ações disponíveis:

* + Escolher forma (círculo/retângulo)
  + Ajustar tamanho
  + Adicionar pontos (clique)
  + Editar pontos existentes
  + Limpar pontos
  + [✅ Concluir Marcação]
  + [💾 Salvar] [📸 Exportar] [🆕 Novo]

1. ESTADO MEDIÇÃO
   * Pontos fixados (não pode mais adicionar/remover)
   * Conexão com hardware
   * Sequência de medição ativa/disponível
   * Valores sendo registrados

Ações disponíveis:

* + Conectar hardware
  + Iniciar/Pausar/Cancelar medição
  + [📊 Modo Comparação] (após medir)
  + [💾 Salvar] [📸 Exportar] [🆕 Novo]

1. ESTADO COMPARAÇÃO
   * Igual ao MEDIÇÃO, mas com foco em comparação
   * Coluna "Comparação" sendo preenchida
   * Cálculo de diferenças em tempo real
   * Coloração automática por tolerância

Ações disponíveis:

* + Iniciar/Pausar/Cancelar medição de comparação
  + Ajustar tolerância
  + Repetir comparação
  + [💾 Salvar] [📸 Exportar] [🆕 Novo]

**TRANSIÇÕES DE ESTADO**

Regras de transição:

1. Estados são unidirecionais no fluxo principal
2. [🆕 Novo Projeto] reseta para INICIAL de qualquer estado
3. [📁 Abrir Projeto] pode ir para qualquer estado (conforme salvo)
4. NÃO é possível voltar de MARCAÇÃO para EDIÇÃO
   * Razão: Manter integridade do grid de pontos
   * Se precisar editar imagem: salvar, criar novo projeto

**CONTROLE DE ESTADO**

Implementação em Python:

from enum import Enum

class AppState(Enum):

INICIAL = "inicial"

EDICAO = "edicao"

MARCACAO = "marcacao"

MEDICAO = "medicao"

COMPARACAO = "comparacao"

class StateManager:

def \_\_init\_\_(self):

self.current\_state = AppState.INICIAL

self.state\_history = [AppState.INICIAL]

def change\_state(self, new\_state: AppState):

"""Transição de estado com validação"""

# Valida transição

if not self.\_is\_valid\_transition(new\_state):

raise ValueError(f"Transição inválida: {self.current\_state} → {new\_state}")

# Atualiza estado

self.current\_state = new\_state

self.state\_history.append(new\_state)

# Emite sinal para atualizar UI

self.state\_changed.emit(new\_state)

def \_is\_valid\_transition(self, new\_state: AppState) -> bool:

"""Valida se a transição é permitida"""

# INICIAL pode ir para qualquer estado (via abrir projeto)

if self.current\_state == AppState.INICIAL:

return True

# Transições válidas

valid\_transitions = {

AppState.EDICAO: [AppState.MARCACAO],

AppState.MARCACAO: [AppState.MEDICAO],

AppState.MEDICAO: [AppState.COMPARACAO],

AppState.COMPARACAO: [AppState.MEDICAO], # Repetir medição

}

return new\_state in valid\_transitions.get(self.current\_state, [])

def can\_go\_back(self) -> bool:

"""Verifica se pode voltar ao estado anterior"""

# Não pode voltar de MARCAÇÃO para EDIÇÃO

return False

def reset(self):

"""Reset para estado inicial"""

self.current\_state = AppState.INICIAL

self.state\_history = [AppState.INICIAL]

self.state\_changed.emit(AppState.INICIAL)

**SINCRONIZAÇÃO UI COM ESTADO**

Cada mudança de estado dispara:

1. Atualização da toolbar superior
2. Troca da toolbar dinâmica
3. Atualização do status bar
4. Ativação/Desativação de ações
5. Atualização de menus
6. Sincronização de ViewModels

**================================================================================ 6. INTERFACE GRÁFICA (GUI)**

**ESTRUTURA GERAL**

╔═══════════════════════════════════════════════════════════╗ ║ Multímetro Inteligente - v1.0 ║ ╠═══════════════════════════════════════════════════════════╣ ║ TOOLBAR SUPERIOR (35px altura, persistente após imagem) ║ ║ [💾 Salvar] [📸 Exportar] [🆕 Novo] ║ ╠═══════════════════════════════════════════════════════════╣ ║ TOOLBAR DINÂMICA (37px altura, muda por estado) ║ ║ [conteúdo varia conforme estado - ver detalhes abaixo] ║ ╠════════════════════════════════════╦══════════════════════╣ ║ ║ 📋 Pontos [5]% [✓OK]║ ║ ╠══════════════════════╣ ║ 🔴 1 ║ Tabela (4 colunas) ║ ║ ║ ║ ║ 🩷 2 ║ ID │Ref │Comp│Dif% ║ ║ ║ 1 │... │... │... ║ ║ 🔴 3 ║ 2 │... │... │... ║ ║ ║ 3 │... │... │... ║ ║ ║ ║ ║ [IMAGEM COM PONTOS] ║ ║ ║ (QGraphicsView) ║ ║ ║ ║ ║ ║ Zoom: 100% ║ ║ ║ Pan: Arrastar ║ ║ ║ Zoom: Scroll ║ ║ ║ ║ ║ ╠════════════════════════════════════╩══════════════════════╣ ║ Status: Pronto │ Pontos: 3 │ Estado: Marcação ║ ╚═══════════════════════════════════════════════════════════╝

**COMPONENTES PRINCIPAIS**

1. JANELA PRINCIPAL (QMainWindow)
   * Tamanho inicial: 1280x800 (ou 1200x800)
   * Mínimo: 1024x600
   * Centralizada na tela
   * Redimensionável
   * Título dinâmico:
     + Sem projeto: "Multímetro Inteligente - v1.0"
     + Com projeto: "Multímetro Inteligente - [Nome do Projeto]"
     + Com alterações: "Multímetro Inteligente - [Nome do Projeto]\*"
2. TOOLBAR SUPERIOR (QToolBar)
   * Altura fixa: 35px
   * Aparece apenas após carregar imagem
   * Ícone + texto nos botões
   * Botões:
     + [💾 Salvar Projeto (Ctrl+S)]
       - Desabilitado se não há alterações
       - Destaque visual se há alterações não salvas
     + [📸 Exportar Imagem (Ctrl+E)]
     + [🆕 Novo Projeto (Ctrl+N)]
   * Sempre visível em: EDIÇÃO, MARCAÇÃO, MEDIÇÃO, COMPARAÇÃO
3. TOOLBAR DINÂMICA (QStackedWidget)
   * Altura fixa: 37px
   * Contém múltiplas toolbars que alternam conforme estado
   * Uma toolbar por estado/submodo
   * Transição suave entre toolbars
4. ÁREA CENTRAL (QSplitter)
   * Orientação: Horizontal
   * Divisão inicial: 70% esquerda (imagem), 30% direita (tabela)
   * Redimensionável pelo usuário
   * Handle visível de 5px
5. PAINEL ESQUERDO
   * Label informativo no topo (removível): "📷 Imagem: 2048x1536 | Zoom: 100% | Pontos: 5"
   * ImageViewer (QGraphicsView customizado)
   * Ocupa todo espaço vertical disponível
   * Fundo: #F5F5F5 (cinza muito claro)
6. PAINEL DIREITO
   * Apenas tabela de pontos
   * Ocupa TODO o espaço vertical
   * Header customizado com controle de tolerância
   * Scroll vertical automático
7. STATUS BAR (QStatusBar)
   * Altura: 25px
   * Três seções:
     + Esquerda: Mensagem principal
     + Centro: Informações de contexto
     + Direita: Estado atual
   * Exemplos:
     + "Pronto │ Pontos: 5 │ Estado: Marcação"
     + "Salvando projeto... │ │ Estado: Medição"
     + "Medindo ponto 3/10 │ ~2 min restantes │ Medição"

**TOOLBARS DINÂMICAS DETALHADAS**

TOOLBAR INICIAL (Estado: INICIAL): ┌────────────────────────────────────────────────────────┐ │ [📂 Abrir Imagem (Ctrl+O)] [📁 Abrir Projeto (Ctrl+P)]│ └────────────────────────────────────────────────────────┘

* Centralizado na tela
* Botões grandes (120x40px cada)

TOOLBAR EDIÇÃO (Estado: EDIÇÃO): ┌───────────────────────────────────────────────────────────────────┐ │ [↩️] [↪️] │ [↻90°] [↻180°] [⬌H] [⬍V] │ [✂️] [📏] │ [✅ Concluir]│ │ Desfazer │ Transformações │ Ajustes│ Avançar │ │ Refazer │ │ │ │ └───────────────────────────────────────────────────────────────────┘

* Grupos separados por linha vertical
* Tooltips ricos:
  + "Desfazer (Ctrl+Z) - Desfaz última transformação"
  + "Rotacionar 90° - Gira imagem no sentido horário"

TOOLBAR MARCAÇÃO - CÍRCULO (Estado: MARCAÇÃO, Forma: Círculo): ┌──────────────────────────────────────────────────────────────────┐ │ [⭕✓] [⬜] │ S [━━━●━━━━━] W 20px │ [✏️(E)] [🗑️] │ [✅ Concluir]│ │ Forma │ Tamanho │ Editar Limpar│ Próximo │ └──────────────────────────────────────────────────────────────────┘

Detalhes do Slider:

* Largura: 180px
* Range: 8px a 60px (raio para círculo)
* Step: 1px
* Valor padrão: 20px
* Label dinâmico atualiza ao mover
* Teclas W/S: incremento de 2px
* Shift+Scroll: incremento fino de 1px

TOOLBAR MARCAÇÃO - RETÂNGULO (Estado: MARCAÇÃO, Forma: Retângulo): ┌────────────────────────────────────────────────────────────────────────┐ │ [⭕] [⬜✓] │ S [━●━] W │ A [━●━] D │ [↔️⇅] │ [✏️] [🗑️] │ [✅ Concluir]│ │ Forma │ Alt: 20px│ Larg: 30px│ Trocar│ Editar Limpar│ Próximo │ └────────────────────────────────────────────────────────────────────────┘

Detalhes dos Sliders:

* Altura (S/W): 8px a 60px, padrão 20px
* Largura (A/D): 8px a 60px, padrão 30px
* Botão [↔️⇅ Trocar]: inverte largura ↔ altura

TOOLBAR MEDIÇÃO (Estado: MEDIÇÃO): ┌──────────────────────────────────────────────────────────────────┐ │ [🔌 Conectar] │ [▶️ Iniciar] [⏸️ Pausar] [⏹️ Cancelar] │ │ Status: OK │ Progresso: 3/10 (30%) │ ~2 min restantes │ └──────────────────────────────────────────────────────────────────┘

Estados do botão Conectar:

* Desconectado: [🔌 Conectar] (cinza)
* Conectando: [🔌 Conectando...] (amarelo, spinner)
* Conectado: [🔌 Conectado ✓] (verde)
* Erro: [🔌 Erro] (vermelho, clicável para tentar novamente)

TOOLBAR COMPARAÇÃO (Estado: COMPARAÇÃO): ┌──────────────────────────────────────────────────────────────────┐ │ [🔄 Repetir Medição] │ [▶️ Iniciar] [⏸️ Pausar] │ │ Comparando com referência │ Progresso: 5/10 │ OK: 3, Dif: 2 │ └──────────────────────────────────────────────────────────────────┘

**DIMENSÕES E ESPAÇAMENTOS**

Toolbars:

* Altura toolbar: 35-37px
* Altura botões: 28-30px
* Largura botões: 30-120px (varia conforme texto)
* Ícones: 16x16px
* Sliders: 120-200px largura
* Espaçamento entre elementos: 5-10px
* Margens: 2-5px

Pontos na imagem:

* Tamanho mínimo: 8px (raio ou lado)
* Tamanho máximo: 60px
* Tamanho padrão: 20px
* Borda: 2px sólida
* Texto ID: tamanho = 25% do tamanho do ponto, mínimo 8pt

Tabela:

* Altura header: 30px
* Altura linha: 24px
* Largura colunas:
  + ID: 50px (fixo)
  + Referência: 33% restante
  + Comparação: 33% restante
  + Diferença: 34% restante

**CORES E ESTILOS**

Paleta principal:

* Fundo geral: #FFFFFF (branco)
* Área de imagem: #F5F5F5 (cinza muito claro)
* Texto principal: #000000 (preto)
* Texto secundário: #666666 (cinza médio)

Pontos:

* Normal: #FF0000 (vermelho), alpha 120 (semi-transparente)
* Divergente: #FF69B4 (rosa HotPink), alpha 150
* Preview: #FF0000 (vermelho), alpha 80 (mais transparente)
* Piscando (medição): #FF69B4 (rosa), animação 0.5s

Tabela:

* Linha normal: #FFFFFF (branco)
* Linha divergente: #FFB6C1 (rosa claro LightPink)
* Linha selecionada: #E3F2FD (azul muito claro)
* Linha hover: #F5F5F5 (cinza muito claro)
* Header: #FAFAFA (cinza quase branco)
* Bordas: #E0E0E0 (cinza claro)

Status bar:

* Fundo: #F0F0F0 (cinza claro padrão)
* Texto normal: #000000
* Texto alerta: #FF6B6B (vermelho claro)
* Texto sucesso: #51CF66 (verde claro)

Botões:

* Sistema padrão PyQt6 (adapta ao OS)
* Hover: leve mudança de cor
* Pressed: efeito de profundidade
* Disabled: 50% opacidade

**FONTES**

Interface geral:

* Família: Sistema padrão (Segoe UI no Windows, SF Pro no macOS)
* Tamanho: 9pt (UI padrão)
* Peso: Normal

IDs dos pontos:

* Família: Sistema padrão, fallback: Arial, sans-serif
* Tamanho: 25% do tamanho do ponto, mínimo 8pt
* Peso: Negrito (Bold)
* Cor: #FFFFFF (branco)
* Contorno: 1px preto para legibilidade

Preview:

* Tamanho: 14pt
* Peso: Negrito

Tabela:

* Família: Sistema padrão
* Tamanho: 9pt
* Header: Negrito

Status bar:

* Tamanho: 8pt

**ÍCONES E SÍMBOLOS**

Utiliza emojis Unicode (funciona em qualquer sistema):

* 📂 Abrir arquivo
* 📁 Abrir pasta/projeto
* 💾 Salvar
* 📸 Exportar/Câmera
* 🆕 Novo
* ↩️ Desfazer
* ↪️ Refazer
* ↻ Rotacionar
* ⬌ Espelhar horizontal
* ⬍ Espelhar vertical
* ✂️ Recortar
* 📏 Redimensionar
* ⭕ Círculo
* ⬜ Retângulo/Quadrado
* ✏️ Editar
* 🗑️ Limpar/Deletar
* ✅ Concluir/OK
* ❌ Cancelar
* 🔌 Conectar
* ▶️ Iniciar/Play
* ⏸️ Pausar
* ⏹️ Parar
* ⏭️ Pular
* 📊 Comparar/Gráfico
* 🔄 Repetir
* ⚠️ Aviso
* 🔴 Ponto normal
* 🩷 Ponto divergente

**INTERAÇÕES DO USUÁRIO**

PAN (Arrastar Imagem):

* Estados: EDIÇÃO, MARCAÇÃO, MEDIÇÃO, COMPARAÇÃO
* Ação: Clicar e arrastar com botão esquerdo
* Cursor: Muda para "mão aberta" ao pressionar
* Suave e responsivo

ZOOM:

* Estados: EDIÇÃO, MARCAÇÃO, MEDIÇÃO, COMPARAÇÃO
* Ação: Scroll do mouse ou touchpad
* Range: 10% a 400%
* Step: 10% por scroll
* Modifiers:
  + Ctrl+Scroll: zoom mais fino (2% por scroll)
  + Shift+Scroll: ajusta tamanho de ponto (em MARCAÇÃO)
* Centraliza no ponto do mouse

ADICIONAR PONTO (em MARCAÇÃO):

* Clique simples na imagem
* Preview aparece ao mover mouse
* Preview segue cursor em tempo real
* Clique confirma posição

EDITAR PONTO (em MARCAÇÃO, modo edição ativado):

* Clicar em ponto para selecionar
* Borda do ponto fica mais grossa
* Arrastar move o ponto
* Delete remove ponto selecionado
* ESC sai do modo edição

**ATALHOS DE TECLADO**

Globais:

* Ctrl+O: Abrir Imagem
* Ctrl+P: Abrir Projeto
* Ctrl+S: Salvar Projeto
* Ctrl+E: Exportar Imagem
* Ctrl+N: Novo Projeto
* Ctrl+Q: Sair
* F11: Tela cheia

Edição:

* Ctrl+Z: Desfazer
* Ctrl+Shift+Z ou Ctrl+Y: Refazer

Marcação:

* Ctrl+Z: Desfazer último ponto
* Ctrl+Shift+Z: Refazer ponto
* W: Aumentar tamanho
* S: Diminuir tamanho
* A: Aumentar largura (retângulo)
* D: Diminuir largura (retângulo)
* E: Ativar modo edição
* Delete: Remover ponto selecionado (em modo edição)
* ESC: Sair do modo edição

Medição:

* Espaço: Iniciar/Pausar medição
* ESC: Cancelar medição

Zoom/Pan:

* +: Zoom in
* -: Zoom out
* 0: Reset zoom (100%)
* Setas: Pan direcional
* Ctrl+Setas: Pan fino

**ACESSIBILIDADE**

Opções para melhorar acessibilidade:

1. Modo Daltônico (Opcional para v2.0):
   * Substitui vermelho/rosa por azul/amarelo
   * Configurável em Preferências
2. Temas:
   * Tema Claro (padrão)
   * Tema Escuro (opcional para v2.0)
3. Tamanhos de Fonte:
   * Pequeno (8pt)
   * Normal (9pt - padrão)
   * Grande (11pt)
   * Muito Grande (13pt)
4. Feedback Visual:
   * Todos os botões têm tooltips
   * Estados claramente diferenciados
   * Confirmações antes de ações destrutivas
5. Feedback Sonoro (Opcional):
   * Som ao completar medição
   * Som de erro
   * Configurável (On/Off)

**RESPONSIVIDADE**

A interface se adapta a diferentes resoluções:

Mínimo (1024x600):

* Splitter ajusta automaticamente
* Scrollbars aparecem se necessário
* Toolbars mantêm funcionalidade

Ideal (1280x800+):

* Layout confortável
* Todos os elementos visíveis
* Experiência otimizada

Grande (1920x1080+):

* Aproveita espaço extra
* Imagem em alta qualidade
* Mais linhas visíveis na tabela

**================================================================================ 7. SISTEMA DE PONTOS DE MEDIÇÃO**

**FORMAS DE PONTOS**

O sistema suporta DUAS formas geométricas:

1. CÍRCULO
   * Definido por: centro (x, y) + raio
   * Tamanho: 8px a 60px (raio)
   * Padrão: 20px
   * Uso: Pontos em componentes circulares, pads, vias
   * Área de click: círculo completo
2. RETÂNGULO
   * Definido por: centro (x, y) + largura + altura
   * Tamanho: 8px a 60px (cada dimensão)
   * Padrão: 20x30px (altura x largura)
   * Uso: Pontos em pinos, trilhas, componentes retangulares
   * Área de click: retângulo completo

**PROPRIEDADES DO PONTO**

Cada ponto tem as seguintes propriedades:

@dataclass

class Point:

id: int # ID sequencial (1, 2, 3...)

x: int # Coordenada X do centro

y: int # Coordenada Y do centro

shape: str # "circle" ou "rectangle"

# Dimensões (conforme forma)

radius: Optional[int] # Para círculo

width: Optional[int] # Para retângulo

height: Optional[int] # Para retângulo

# Valores de medição

reference\_value: Optional[float] # Valor da placa boa (Volts)

compare\_value: Optional[float] # Valor da placa em teste (Volts)

# Metadados

created\_at: str # Timestamp ISO8601

measured\_at: Optional[str] # Timestamp da medição

# Propriedades calculadas

@property

def has\_reference(self) -> bool:

return self.reference\_value is not None

@property

def has\_comparison(self) -> bool:

return self.compare\_value is not None

@property

def difference\_percent(self) -> Optional[float]:

"""Calcula diferença percentual"""

if not self.has\_reference or not self.has\_comparison:

return None

if self.reference\_value == 0:

return None # Evita divisão por zero

diff = (self.compare\_value - self.reference\_value) / self.reference\_value \* 100

return round(diff, 2)

def is\_divergent(self, tolerance\_percent: float) -> bool:

"""Verifica se está fora da tolerância"""

diff = self.difference\_percent

if diff is None:

return False

return abs(diff) > tolerance\_percent

**RENDERIZAÇÃO DE PONTOS**

Pontos são renderizados como QGraphicsItems no QGraphicsView:

1. CÍRCULO:
   * QPainterPath com círculo
   * Borda sólida de 2px
   * Preenchimento semi-transparente
   * Texto centralizado com ID
2. RETÂNGULO:
   * QPainterPath com retângulo
   * Borda sólida de 2px
   * Preenchimento semi-transparente
   * Texto centralizado com ID

Cores:

* Normal (sem comparação): Vermelho #FF0000, alpha 120
* Divergente (fora da tolerância): Rosa #FF69B4, alpha 150
* Piscando (medição ativa): Rosa #FF69B4, animação 0.5s

Texto ID:

* Fonte: Negrito
* Tamanho: 25% do tamanho do ponto (mínimo 8pt)
* Cor: Branco #FFFFFF
* Contorno: 1px preto para contraste

**PREVIEW DE PONTO**

Ao mover o mouse sobre a imagem em modo MARCAÇÃO:

* Preview transparente aparece
* Segue cursor em tempo real
* Mesma forma e tamanho do ponto que será criado
* Cor: Vermelho #FF0000, alpha 80 (mais transparente)
* Não tem ID (é apenas preview)
* Desaparece ao sair da área da imagem

**ANIMAÇÃO DE PISCADA (MEDIÇÃO)**

Durante sequência de medição, o ponto atual pisca:

class BlinkingPointItem(QGraphicsItem):

def \_\_init\_\_(self, point: Point):

super().\_\_init\_\_()

self.point = point

self.opacity = 1.0

self.direction = -0.1 # Diminuindo

# Timer para animação

self.timer = QTimer()

self.timer.timeout.connect(self.animate)

self.timer.start(50) # 50ms = 20fps

def animate(self):

# Oscila opacidade entre 0.3 e 1.0

self.opacity += self.direction

if self.opacity <= 0.3:

self.opacity = 0.3

self.direction = 0.1 # Aumentando

elif self.opacity >= 1.0:

self.opacity = 1.0

self.direction = -0.1 # Diminuindo

self.update() # Redesenha

def stop\_blinking(self):

self.timer.stop()

self.opacity = 1.0

self.update()

Características:

* Frequência: ~2 Hz (2 piscadas por segundo)
* Cor: Rosa #FF69B4
* Range de opacidade: 0.3 a 1.0
* Suave (não pisca instantaneamente)
* Para ao receber medição

**MODO EDIÇÃO DE PONTOS**

Ao ativar [✏️ Editar]:

1. Todos os pontos ficam selecionáveis
2. Cursor muda ao passar sobre ponto
3. Clicar seleciona o ponto:
   * Borda fica mais grossa (4px)
   * Cor da borda: Azul #2196F3
4. Arrastar move o ponto
5. Delete remove o ponto selecionado
6. ESC sai do modo edição

Restrições:

* Não pode mover ponto para fora da imagem
* Não pode sobrepor pontos (aviso se muito próximo)
* Histórico mantém posições anteriores (Ctrl+Z funciona)

**GERENCIAMENTO DE PONTOS**

class PointManager:

def \_\_init\_\_(self):

self.points: List[Point] = []

self.next\_id = 1

self.history: List[List[Point]] = []

self.history\_index = -1

def add\_point(self, x: int, y: int, shape: str, \*\*dimensions) -> Point:

"""Adiciona novo ponto"""

point = Point(

id=self.next\_id,

x=x, y=y,

shape=shape,

created\_at=datetime.now().isoformat(),

\*\*dimensions

)

self.points.append(point)

self.next\_id += 1

self.\_save\_to\_history()

# Emite sinal

self.point\_added.emit(point)

return point

def remove\_point(self, point\_id: int) -> bool:

"""Remove ponto por ID"""

point = self.get\_point(point\_id)

if point:

self.points.remove(point)

self.\_save\_to\_history()

self.point\_removed.emit(point\_id)

return True

return False

def clear\_all(self) -> int:

"""Remove todos os pontos"""

count = len(self.points)

self.points.clear()

self.next\_id = 1

self.\_save\_to\_history()

self.points\_cleared.emit()

return count

def update\_point\_position(self, point\_id: int, x: int, y: int):

"""Atualiza posição de um ponto"""

point = self.get\_point(point\_id)

if point:

point.x = x

point.y = y

self.\_save\_to\_history()

self.point\_moved.emit(point\_id, x, y)

def set\_reference\_value(self, point\_id: int, value: float):

"""Define valor de referência"""

point = self.get\_point(point\_id)

if point:

point.reference\_value = value

point.measured\_at = datetime.now().isoformat()

self.measurement\_updated.emit(point\_id, "reference", value)

def set\_compare\_value(self, point\_id: int, value: float):

"""Define valor de comparação"""

point = self.get\_point(point\_id)

if point:

point.compare\_value = value

self.measurement\_updated.emit(point\_id, "compare", value)

def get\_point(self, point\_id: int) -> Optional[Point]:

"""Busca ponto por ID"""

return next((p for p in self.points if p.id == point\_id), None)

def get\_divergent\_points(self, tolerance: float) -> List[Point]:

"""Retorna pontos fora da tolerância"""

return [p for p in self.points if p.is\_divergent(tolerance)]

def undo(self) -> bool:

"""Desfaz última ação"""

if self.history\_index > 0:

self.history\_index -= 1

self.points = self.history[self.history\_index].copy()

self.points\_restored.emit()

return True

return False

def redo(self) -> bool:

"""Refaz ação desfeita"""

if self.history\_index < len(self.history) - 1:

self.history\_index += 1

self.points = self.history[self.history\_index].copy()

self.points\_restored.emit()

return True

return False

def \_save\_to\_history(self):

"""Salva estado atual no histórico"""

# Remove histórico futuro se estamos no meio

if self.history\_index < len(self.history) - 1:

self.history = self.history[:self.history\_index + 1]

# Adiciona estado atual (deep copy)

self.history.append([Point(\*\*vars(p)) for p in self.points])

self.history\_index += 1

# Limita tamanho do histórico (últimas 50 ações)

if len(self.history) > 50:

self.history.pop(0)

self.history\_index -= 1

**TABELA DE PONTOS**

A tabela lateral exibe todos os pontos com 4 colunas:

┌────┬────────────┬────────────┬────────────┐

│ ID │ Referência │ Comparação │ Diferença │

├────┼────────────┼────────────┼────────────┤

│ 1 │ 0.450V │ 0.456V │ +1.3% │

│ 2 │ 0.450V │ 0.120V │ -73.3% │ ← Rosa

│ 3 │ 0.380V │ -- │ -- │

│ 4 │ -- │ -- │ -- │

└────┴────────────┴────────────┴────────────┘

Implementação:

class PointsTableModel(QAbstractTableModel):

def \_\_init\_\_(self, point\_manager: PointManager, tolerance: float = 5.0):

super().\_\_init\_\_()

self.point\_manager = point\_manager

self.tolerance = tolerance

self.headers = ["ID", "Referência", "Comparação", "Diferença"]

def rowCount(self, parent=QModelIndex()) -> int:

return len(self.point\_manager.points)

def columnCount(self, parent=QModelIndex()) -> int:

return 4

def data(self, index: QModelIndex, role: int):

if not index.isValid():

return None

point = self.point\_manager.points[index.row()]

col = index.column()

if role == Qt.DisplayRole:

if col == 0: # ID

return str(point.id)

elif col == 1: # Referência

return f"{point.reference\_value:.3f}V" if point.has\_reference else "--"

elif col == 2: # Comparação

return f"{point.compare\_value:.3f}V" if point.has\_comparison else "--"

elif col == 3: # Diferença

diff = point.difference\_percent

return f"{diff:+.1f}%" if diff is not None else "--"

elif role == Qt.BackgroundRole:

# Rosa se divergente

if point.is\_divergent(self.tolerance):

return QColor(255, 182, 193) # #FFB6C1 LightPink

return QColor(255, 255, 255) # Branco

elif role == Qt.TextAlignmentRole:

if col == 0:

return Qt.AlignCenter

return Qt.AlignLeft | Qt.AlignVCenter

return None

def headerData(self, section: int, orientation: Qt.Orientation, role: int):

if role == Qt.DisplayRole and orientation == Qt.Horizontal:

return self.headers[section]

return None

def set\_tolerance(self, tolerance: float):

"""Atualiza tolerância e redesenha"""

self.tolerance = tolerance

self.layoutChanged.emit()

**VALIDAÇÕES**

1. Limites da imagem:
   * Ponto não pode ser criado fora da imagem
   * Ao mover, ponto é restrito aos limites
2. Sobreposição:
   * Aviso se pontos muito próximos (< 10px)
   * Não bloqueia, apenas alerta
3. Valores de medição:
   * Range: -2.000V a +2.000V
   * Precisão: 3 casas decimais
   * Valores fora do range geram alerta
4. Quantidade de pontos:
   * Mínimo: 1 ponto (para avançar de MARCAÇÃO)
   * Máximo: 999 pontos (limite prático)

**================================================================================ 8. SISTEMA DE SALVAMENTO (.mip)**

**FORMATO .MIP**

Arquivo .mip é um ZIP compactado contendo:

project.mip (ZIP)

├── metadata.json # Informações do projeto

├── image.png # Imagem da placa (sempre PNG)

└── points.json # Lista de pontos e medições

**SCHEMA VERSIONADO**

O formato usa versionamento semântico (MAJOR.MINOR.PATCH):

* MAJOR: Mudanças incompatíveis (quebra compatibilidade)
* MINOR: Novas features (compatível com versões anteriores)
* PATCH: Bug fixes (totalmente compatível)

Versão atual: 1.0.0

**METADATA.JSON**

Estrutura completa:

{

"schema\_version": "1.0.0",

"app\_version": "1.0.0",

"project\_info": {

"name": "TV LG 42LB5600 - Placa Boa",

"board\_model": "EAX65388005",

"is\_fully\_functional": true,

"problem\_description": "",

"notes": "Placa de referência testada e aprovada"

},

"settings": {

"tolerance\_percent": 5.0,

"measurement\_profile": "diode\_reverse"

},

"timestamps": {

"created\_at": "2025-10-31T14:30:00-03:00",

"modified\_at": "2025-10-31T15:45:00-03:00",

"measured\_at": "2025-10-31T15:20:00-03:00"

},

"image\_info": {

"original\_filename": "placa\_lg\_42lb5600.jpg",

"original\_size": [3024, 4032],

"final\_size": [2048, 2731],

"transformations": [

{"type": "rotate", "angle": 90},

{"type": "crop", "region": [100, 150, 2048, 2731]}

]

},

"statistics": {

"total\_points": 24,

"measured\_points": 24,

"compared\_points": 0

},

"integrity": {

"image\_sha256": "a3b2c1d4e5f6...",

"points\_sha256": "1a2b3c4d5e6f..."

}

}

**POINTS.JSON**

Array de pontos com todas as informações:

[

{

"id": 1,

"position": {

"x": 1204,

"y": 356

},

"shape": {

"type": "circle",

"radius": 18

},

"measurements": {

"reference": {

"value": 0.450,

"timestamp": "2025-10-31T15:10:22-03:00"

},

"compare": {

"value": 0.456,

"timestamp": "2025-10-31T15:45:10-03:00"

}

},

"metadata": {

"created\_at": "2025-10-31T14:50:15-03:00"

}

},

{

"id": 2,

"position": {

"x": 890,

"y": 642

},

"shape": {

"type": "rectangle",

"width": 28,

"height": 20

},

"measurements": {

"reference": {

"value": 0.450,

"timestamp": "2025-10-31T15:10:25-03:00"

},

"compare": {

"value": 0.120,

"timestamp": "2025-10-31T15:45:13-03:00"

}

},

"metadata": {

"created\_at": "2025-10-31T14:50:18-03:00"

}

}

]

**IMAGE.PNG**

* Sempre convertido para PNG (qualidade sem perdas)
* Dimensões preservadas após edições
* Modo RGB (24-bit)
* Sem compressão excessiva (qualidade alta)

**CHECKSUMS SHA-256**

Para garantir integridade dos arquivos:

import hashlib

def calculate\_sha256(file\_path: str) -> str:

"""Calcula SHA-256 de um arquivo"""

sha256 = hashlib.sha256()

with open(file\_path, 'rb') as f:

for chunk in iter(lambda: f.read(4096), b''):

sha256.update(chunk)

return sha256.hexdigest()

Checksums são calculados para:

1. image.png
2. points.json

E armazenados em metadata.json → integrity

**PROCESSO DE SALVAMENTO**

class MipRepository:

@staticmethod

def save(project\_data: dict, file\_path: str) -> bool:

"""

Salva projeto no formato .mip

Args:

project\_data: Dicionário com todos os dados do projeto

file\_path: Caminho completo para salvar (com extensão .mip)

Returns:

True se sucesso, False se erro

"""

try:

# 1. Cria diretório temporário

with tempfile.TemporaryDirectory() as temp\_dir:

# 2. Salva imagem como PNG

image\_path = os.path.join(temp\_dir, "image.png")

project\_data['image'].save(image\_path, format='PNG', quality=95)

# 3. Salva points.json

points\_path = os.path.join(temp\_dir, "points.json")

with open(points\_path, 'w', encoding='utf-8') as f:

json.dump(project\_data['points'], f, indent=2, ensure\_ascii=False)

# 4. Calcula checksums

image\_sha = calculate\_sha256(image\_path)

points\_sha = calculate\_sha256(points\_path)

# 5. Monta metadata

metadata = {

"schema\_version": "1.0.0",

"app\_version": "1.0.0",

\*\*project\_data['metadata'],

"integrity": {

"image\_sha256": image\_sha,

"points\_sha256": points\_sha

}

}

# 6. Salva metadata.json

metadata\_path = os.path.join(temp\_dir, "metadata.json")

with open(metadata\_path, 'w', encoding='utf-8') as f:

json.dump(metadata, f, indent=2, ensure\_ascii=False)

# 7. Cria arquivo ZIP temporário

temp\_zip = file\_path + ".tmp"

with zipfile.ZipFile(temp\_zip, 'w', zipfile.ZIP\_DEFLATED) as zipf:

zipf.write(metadata\_path, "metadata.json")

zipf.write(image\_path, "image.png")

zipf.write(points\_path, "points.json")

# 8. Renomeia atomicamente (evita corrupção)

if os.path.exists(file\_path):

os.replace(temp\_zip, file\_path)

else:

os.rename(temp\_zip, file\_path)

return True

except Exception as e:

logger.error(f"Erro ao salvar projeto: {e}")

return False

**PROCESSO DE CARREGAMENTO**

class MipRepository:

@staticmethod

def load(file\_path: str) -> Optional[dict]:

"""

Carrega projeto do formato .mip

Args:

file\_path: Caminho do arquivo .mip

Returns:

Dicionário com dados do projeto ou None se erro

"""

try:

# 1. Verifica se arquivo existe

if not os.path.exists(file\_path):

raise FileNotFoundError(f"Arquivo não encontrado: {file\_path}")

# 2. Verifica se é ZIP válido

if not zipfile.is\_zipfile(file\_path):

raise ValueError("Arquivo não é um .mip válido (não é ZIP)")

# 3. Extrai para diretório temporário

with tempfile.TemporaryDirectory() as temp\_dir:

with zipfile.ZipFile(file\_path, 'r') as zipf:

zipf.extractall(temp\_dir)

# 4. Verifica arquivos obrigatórios

required\_files = ["metadata.json", "image.png", "points.json"]

for filename in required\_files:

if not os.path.exists(os.path.join(temp\_dir, filename)):

raise ValueError(f"Arquivo obrigatório ausente: {filename}")

# 5. Carrega metadata

metadata\_path = os.path.join(temp\_dir, "metadata.json")

with open(metadata\_path, 'r', encoding='utf-8') as f:

metadata = json.load(f)

# 6. Valida schema version

schema\_version = metadata.get("schema\_version", "0.0.0")

if not MipRepository.\_is\_compatible\_version(schema\_version):

raise ValueError(f"Versão incompatível: {schema\_version}")

# 7. Verifica checksums

image\_path = os.path.join(temp\_dir, "image.png")

points\_path = os.path.join(temp\_dir, "points.json")

expected\_image\_sha = metadata['integrity']['image\_sha256']

expected\_points\_sha = metadata['integrity']['points\_sha256']

actual\_image\_sha = calculate\_sha256(image\_path)

actual\_points\_sha = calculate\_sha256(points\_path)

if actual\_image\_sha != expected\_image\_sha:

raise ValueError("Checksum da imagem não confere (arquivo corrompido)")

if actual\_points\_sha != expected\_points\_sha:

raise ValueError("Checksum dos pontos não confere (arquivo corrompido)")

# 8. Carrega imagem

image = Image.open(image\_path)

# 9. Carrega pontos

with open(points\_path, 'r', encoding='utf-8') as f:

points\_data = json.load(f)

# 10. Retorna dados

return {

'metadata': metadata,

'image': image,

'points': points\_data

}

except Exception as e:

logger.error(f"Erro ao carregar projeto: {e}")

return None

@staticmethod

def \_is\_compatible\_version(schema\_version: str) -> bool:

"""Verifica compatibilidade de versão"""

# Parse version

try:

major, minor, patch = map(int, schema\_version.split('.'))

except:

return False

# Versão atual

current\_major, current\_minor, current\_patch = 1, 0, 0

# Compatível se major é igual e minor <= atual

return major == current\_major and minor <= current\_minor

**VALIDAÇÕES**

Ao salvar:

1. Nome do projeto obrigatório (min 3 caracteres)
2. Modelo obrigatório (min 2 caracteres)
3. Se não funcional: descrição obrigatória (min 10 caracteres)
4. Tolerância: 0.1% a 50.0%
5. Pelo menos 1 ponto marcado

Ao carregar:

1. Arquivo existe
2. É ZIP válido
3. Contém arquivos obrigatórios
4. Schema version compatível
5. Checksums conferem

**MIGRAÇÕES DE VERSÃO**

Para futuras versões:

class MipMigrator:

@staticmethod

def migrate(metadata: dict, from\_version: str, to\_version: str) -> dict:

"""

Migra metadata de uma versão para outra

Exemplo: 1.0.0 → 1.1.0

"""

migrations = {

("1.0.0", "1.1.0"): MipMigrator.\_migrate\_1\_0\_to\_1\_1,

("1.1.0", "1.2.0"): MipMigrator.\_migrate\_1\_1\_to\_1\_2,

}

key = (from\_version, to\_version)

if key in migrations:

return migrations[key](metadata)

return metadata # Sem migração necessária

@staticmethod

def \_migrate\_1\_0\_to\_1\_1(metadata: dict) -> dict:

"""Exemplo de migração"""

# Adiciona novos campos com valores padrão

if 'new\_field' not in metadata:

metadata['new\_field'] = default\_value

# Atualiza schema\_version

metadata['schema\_version'] = "1.1.0"

return metadata

**EXPORTAÇÕES**

Além do .mip, o sistema permite exportar:

1. IMAGEM COM PONTOS (PNG):
   * Imagem original com pontos renderizados
   * IDs visíveis
   * Cores indicando divergências
   * Resolução original mantida
2. DADOS (CSV):
3. ID,Forma,X,Y,Referência(V),Comparação(V),Diferença(%)
4. 1,círculo,1204,356,0.450,0.456,+1.3
5. 2,círculo,890,642,0.450,0.120,-73.3
6. 3,retângulo,1520,890,0.380,0.382,+0.5
7. DADOS (JSON):
   * Mesmo formato de points.json
   * Inclui metadados resumidos
8. RELATÓRIO (HTML) [Futuro]:
   * Página HTML com imagem e tabela
   * Gráficos de distribuição
   * Estatísticas resumidas

**================================================================================ 9. SISTEMA DE COMPARAÇÃO**

**FLUXO DE COMPARAÇÃO**

1. Projeto da placa boa já está aberto
2. Usuário clica [📊 Comparar com Outra Placa]
3. Sistema prepara para nova sequência de medição
4. Valores vão para coluna "Comparação"
5. Cálculos automáticos em tempo real
6. Visualização de divergências

**CÁLCULO DE DIFERENÇAS**

Fórmula:

Diferença (%) = ((Valor\_Comparação - Valor\_Referência) / Valor\_Referência) × 100

Exemplos:

* Ref: 0.450V, Comp: 0.456V → (0.456 - 0.450) / 0.450 × 100 = +1.3%
* Ref: 0.450V, Comp: 0.120V → (0.120 - 0.450) / 0.450 × 100 = -73.3%
* Ref: 0.380V, Comp: 0.382V → (0.382 - 0.380) / 0.380 × 100 = +0.5%

Implementação:

class ComparisonCalculator:

@staticmethod

def calculate\_difference(reference: float, compare: float) -> Optional[float]:

"""

Calcula diferença percentual entre dois valores

Returns:

float: Diferença em percentual (ex: 1.3, -73.3)

None: Se cálculo não for possível

"""

if reference is None or compare is None:

return None

# Evita divisão por zero

if reference == 0:

if compare == 0:

return 0.0

else:

return None # Não definido

diff = ((compare - reference) / reference) \* 100

return round(diff, 2)

@staticmethod

def is\_within\_tolerance(difference: Optional[float], tolerance: float) -> bool:

"""

Verifica se diferença está dentro da tolerância

Args:

difference: Diferença percentual (pode ser None)

tolerance: Tolerância em percentual (ex: 5.0 para ±5%)

Returns:

True se dentro da tolerância ou se difference é None

"""

if difference is None:

return True # Sem comparação = assume OK

return abs(difference) <= tolerance

**TOLERÂNCIA CONFIGURÁVEL**

A tolerância define o limite para considerar um ponto divergente:

* Valor padrão: 5.0% (±5%)
* Range permitido: 0.1% a 50.0%
* Ajustável no header da tabela
* Aplicação em tempo real

Interface:

┌──────────────────────────────────────────────────────┐

│ 📋 Pontos [24] Tolerância: [5.0] % [✓ Aplicar] │

└──────────────────────────────────────────────────────┘

Comportamento:

1. Usuário edita valor no campo
2. Clica [✓ Aplicar] ou pressiona Enter
3. Sistema valida (0.1 a 50.0)
4. Recalcula quais pontos são divergentes
5. Atualiza cores na imagem e tabela imediatamente

class ToleranceWidget(QWidget):

tolerance\_changed = pyqtSignal(float)

def \_\_init\_\_(self, default: float = 5.0):

super().\_\_init\_\_()

layout = QHBoxLayout()

# Label

layout.addWidget(QLabel("Tolerância:"))

# Input

self.input = QDoubleSpinBox()

self.input.setRange(0.1, 50.0)

self.input.setValue(default)

self.input.setDecimals(1)

self.input.setSuffix(" %")

self.input.setFixedWidth(80)

layout.addWidget(self.input)

# Botão aplicar

self.apply\_btn = QPushButton("✓ Aplicar")

self.apply\_btn.clicked.connect(self.apply\_tolerance)

layout.addWidget(self.apply\_btn)

layout.addStretch()

self.setLayout(layout)

# Enter no input também aplica

self.input.editingFinished.connect(self.apply\_tolerance)

def apply\_tolerance(self):

value = self.input.value()

self.tolerance\_changed.emit(value)

**VISUALIZAÇÃO DE DIVERGÊNCIAS**

Pontos divergentes são destacados automaticamente:

NA IMAGEM:

* Cor ROSA (#FF69B4, alpha 150)
* Bordas mais visíveis
* Destaque visual imediato

NA TABELA:

* Linha inteira com fundo rosa claro (#FFB6C1)
* Valor da diferença em negrito
* Fácil identificação

Exemplo visual:

Tabela com Tolerância de 5%:

┌────┬────────────┬────────────┬────────────┐

│ ID │ Referência │ Comparação │ Diferença │

├────┼────────────┼────────────┼────────────┤

│ 1 │ 0.450V │ 0.456V │ +1.3% │ ← Branco (OK)

│ 2 │ 0.450V │ 0.120V │ -73.3% │ ← ROSA (Divergente!)

│ 3 │ 0.380V │ 0.382V │ +0.5% │ ← Branco (OK)

│ 4 │ 0.520V │ 0.495V │ -4.8% │ ← Branco (OK)

│ 5 │ 0.390V │ 0.420V │ +7.7% │ ← ROSA (Divergente!)

└────┴────────────┴────────────┴────────────┘

Se aumentar tolerância para 10%:

- Ponto 5 passa a ser branco (7.7% < 10%)

- Ponto 2 continua rosa (73.3% > 10%)

**ESTATÍSTICAS DE COMPARAÇÃO**

No status bar, resumo em tempo real:

Comparação: 22 OK, 2 Diferentes (tolerância: 5.0%) | 24/24 pontos medidos

Componentes:

* Pontos OK: Dentro da tolerância
* Pontos Diferentes: Fora da tolerância
* Tolerância atual
* Progresso de medição

**ORDENAÇÃO E FILTROS**

Funcionalidades da tabela:

1. ORDENAÇÃO:
   * Clicar no header de qualquer coluna
   * Ordem crescente/decrescente
   * Útil para:
     + Ver maiores divergências primeiro
     + Agrupar por status
2. SELEÇÃO:
   * Clicar em linha seleciona
   * Ponto correspondente na imagem é destacado
   * Ctrl+Click: seleção múltipla
   * Facilita localização visual
3. FILTRO (Futuro):
   * Mostrar apenas divergentes
   * Filtrar por range de diferença

**EXPORTAÇÃO DE COMPARAÇÃO**

Após comparação, permite exportar:

1. IMAGEM ANOTADA:
2. def export\_annotated\_image(self, file\_path: str):
3. """Exporta imagem com pontos coloridos"""
4. # Cria cópia da imagem
5. img = self.image.copy()
6. draw = ImageDraw.Draw(img, 'RGBA')
8. for point in self.points:
9. # Determina cor
10. if point.is\_divergent(self.tolerance):
11. color = (255, 105, 180, 150) # Rosa
12. else:
13. color = (255, 0, 0, 120) # Vermelho
15. # Desenha forma
16. if point.shape == "circle":
17. bbox = [
18. point.x - point.radius,
19. point.y - point.radius,
20. point.x + point.radius,
21. point.y + point.radius
22. ]
23. draw.ellipse(bbox, fill=color, outline=color[:3]+(255,), width=2)
25. # Desenha ID
26. font = ImageFont.truetype("arial.ttf", int(point.radius \* 0.5))
27. text = str(point.id)
28. draw.text((point.x, point.y), text, fill=(255,255,255,255),
29. font=font, anchor="mm")
31. # Salva
32. img.save(file\_path, format='PNG')
33. RELATÓRIO CSV:
34. def export\_comparison\_csv(self, file\_path: str):
35. """Exporta dados da comparação"""
36. with open(file\_path, 'w', newline='', encoding='utf-8') as f:
37. writer = csv.writer(f)
39. # Header
40. writer.writerow([
41. "ID", "Forma", "X", "Y",
42. "Referência(V)", "Comparação(V)", "Diferença(%)",
43. "Status"
44. ])
46. # Dados
47. for point in self.points:
48. status = "DIVERGENTE" if point.is\_divergent(self.tolerance) else "OK"
50. writer.writerow([
51. point.id,
52. point.shape,
53. point.x,
54. point.y,
55. f"{point.reference\_value:.3f}" if point.has\_reference else "",
56. f"{point.compare\_value:.3f}" if point.has\_comparison else "",
57. f"{point.difference\_percent:+.2f}" if point.difference\_percent else "",
58. status
59. ])
61. # Rodapé com estatísticas
62. writer.writerow([])
63. writer.writerow(["RESUMO"])
64. writer.writerow(["Total de pontos:", len(self.points)])
65. writer.writerow(["Pontos OK:", len([p for p in self.points if not p.is\_divergent(self.tolerance)])])
66. writer.writerow(["Pontos divergentes:", len([p for p in self.points if p.is\_divergent(self.tolerance)])])
67. writer.writerow(["Tolerância aplicada:", f"{self.tolerance}%"])
68. RELATÓRIO JSON:
    * Formato estruturado
    * Inclui metadados completos
    * Fácil para processamento automatizado

**================================================================================ 10. ESPECIFICAÇÕES TÉCNICAS DETALHADAS**

**RENDERIZAÇÃO DE IMAGEM**

QGraphicsView customizado para exibir imagem e pontos:

class ImageViewer(QGraphicsView):

"""

Visualizador de imagem com suporte a zoom, pan e overlay de pontos

"""

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Scene

self.scene = QGraphicsScene()

self.setScene(self.scene)

# Imagem base

self.image\_item = None

self.image\_pixmap = None

# Pontos

self.point\_items = {} # {point\_id: QGraphicsItem}

# Preview

self.preview\_item = None

# Estado

self.zoom\_factor = 1.0

self.is\_panning = False

self.pan\_start = QPoint()

# Configurações

self.setRenderHint(QPainter.Antialiasing)

self.setRenderHint(QPainter.SmoothPixmapTransform)

self.setDragMode(QGraphicsView.NoDrag)

self.setTransformationAnchor(QGraphicsView.AnchorUnderMouse)

self.setResizeAnchor(QGraphicsView.AnchorUnderMouse)

self.setBackgroundBrush(QBrush(QColor(245, 245, 245)))

def load\_image(self, pixmap: QPixmap):

"""Carrega imagem na view"""

self.image\_pixmap = pixmap

# Remove imagem anterior

if self.image\_item:

self.scene.removeItem(self.image\_item)

# Adiciona nova imagem

self.image\_item = self.scene.addPixmap(pixmap)

self.image\_item.setZValue(0) # Camada de fundo

# Ajusta cena ao tamanho da imagem

self.scene.setSceneRect(QRectF(pixmap.rect()))

# Fit na view

self.fitInView(self.scene.sceneRect(), Qt.KeepAspectRatio)

self.zoom\_factor = 1.0

def add\_point\_item(self, point: Point) -> QGraphicsItem:

"""Adiciona ponto visual"""

item = PointGraphicsItem(point)

item.setZValue(1) # Camada sobre a imagem

self.scene.addItem(item)

self.point\_items[point.id] = item

return item

def remove\_point\_item(self, point\_id: int):

"""Remove ponto visual"""

if point\_id in self.point\_items:

item = self.point\_items[point\_id]

self.scene.removeItem(item)

del self.point\_items[point\_id]

def wheelEvent(self, event: QWheelEvent):

"""Zoom com scroll do mouse"""

# Fator de zoom

if event.angleDelta().y() > 0:

factor = 1.1 # Zoom in

else:

factor = 0.9 # Zoom out

# Aplica zoom

self.zoom\_factor \*= factor

# Limita zoom (10% a 400%)

self.zoom\_factor = max(0.1, min(4.0, self.zoom\_factor))

# Atualiza transformação

self.setTransform(QTransform().scale(self.zoom\_factor, self.zoom\_factor))

event.accept()

def mousePressEvent(self, event: QMouseEvent):

"""Inicia pan ou ação customizada"""

if event.button() == Qt.LeftButton:

# Se em modo marcação, não faz pan (adiciona ponto)

if self.mode == "marking":

pos = self.mapToScene(event.pos())

self.point\_clicked.emit(int(pos.x()), int(pos.y()))

else:

# Inicia pan

self.is\_panning = True

self.pan\_start = event.pos()

self.setCursor(Qt.ClosedHandCursor)

super().mousePressEvent(event)

def mouseMoveEvent(self, event: QMouseEvent):

"""Pan ou atualiza preview"""

if self.is\_panning:

# Pan

delta = event.pos() - self.pan\_start

self.pan\_start = event.pos()

self.horizontalScrollBar().setValue(

self.horizontalScrollBar().value() - delta.x()

)

self.verticalScrollBar().setValue(

self.verticalScrollBar().value() - delta.y()

)

elif self.mode == "marking" and self.preview\_item:

# Atualiza posição do preview

pos = self.mapToScene(event.pos())

self.preview\_item.setPos(pos)

super().mouseMoveEvent(event)

def mouseReleaseEvent(self, event: QMouseEvent):

"""Finaliza pan"""

if event.button() == Qt.LeftButton and self.is\_panning:

self.is\_panning = False

self.setCursor(Qt.ArrowCursor)

super().mouseReleaseEvent(event)

**PONTOS GRÁFICOS**

class PointGraphicsItem(QGraphicsItem):

"""

Item gráfico para representar um ponto

"""

def \_\_init\_\_(self, point: Point):

super().\_\_init\_\_()

self.point = point

self.is\_blinking = False

self.blink\_opacity = 1.0

# Configurações

self.setFlag(QGraphicsItem.ItemIsSelectable, False)

self.setFlag(QGraphicsItem.ItemIsMovable, False)

def boundingRect(self) -> QRectF:

"""Define área de desenho"""

if self.point.shape == "circle":

r = self.point.radius

return QRectF(-r-2, -r-2, 2\*r+4, 2\*r+4)

else:

w, h = self.point.width, self.point.height

return QRectF(-w/2-2, -h/2-2, w+4, h+4)

def paint(self, painter: QPainter, option, widget):

"""Renderiza o ponto"""

# Determina cor

if self.point.is\_divergent(self.tolerance):

color = QColor(255, 105, 180, 150) # Rosa

else:

color = QColor(255, 0, 0, 120) # Vermelho

# Aplica opacidade se piscando

if self.is\_blinking:

color.setAlpha(int(color.alpha() \* self.blink\_opacity))

# Configurações do painter

painter.setRenderHint(QPainter.Antialiasing)

painter.setPen(QPen(color.darker(), 2))

painter.setBrush(QBrush(color))

# Desenha forma

if self.point.shape == "circle":

painter.drawEllipse(QPointF(0, 0), self.point.radius, self.point.radius)

else:

w, h = self.point.width, self.point.height

painter.drawRect(QRectF(-w/2, -h/2, w, h))

# Desenha ID

font = painter.font()

font.setBold(True)

if self.point.shape == "circle":

font.setPointSize(max(8, int(self.point.radius \* 0.5)))

else:

font.setPointSize(max(8, int(min(self.point.width, self.point.height) \* 0.4)))

painter.setFont(font)

painter.setPen(QPen(Qt.white))

painter.drawText(self.boundingRect(), Qt.AlignCenter, str(self.point.id))

def start\_blinking(self):

"""Inicia animação de piscada"""

self.is\_blinking = True

self.blink\_timer = QTimer()

self.blink\_timer.timeout.connect(self.\_animate\_blink)

self.blink\_timer.start(50) # 20fps

def stop\_blinking(self):

"""Para animação"""

self.is\_blinking = False

self.blink\_opacity = 1.0

if hasattr(self, 'blink\_timer'):

self.blink\_timer.stop()

self.update()

def \_animate\_blink(self):

"""Anima piscada"""

# Oscila entre 0.3 e 1.0

self.blink\_opacity -= 0.1

if self.blink\_opacity <= 0.3:

self.blink\_opacity = 1.0

self.update()

**PROCESSAMENTO DE IMAGEM**

class ImageProcessor:

"""Processamento de imagens usando Pillow"""

@staticmethod

def rotate(image: Image.Image, angle: int) -> Image.Image:

"""Rotaciona imagem (90 ou 180 graus)"""

if angle == 90:

return image.rotate(-90, expand=True)

elif angle == 180:

return image.rotate(180, expand=True)

else:

raise ValueError("Ângulo deve ser 90 ou 180")

@staticmethod

def flip\_horizontal(image: Image.Image) -> Image.Image:

"""Espelha horizontalmente"""

return image.transpose(Image.FLIP\_LEFT\_RIGHT)

@staticmethod

def flip\_vertical(image: Image.Image) -> Image.Image:

"""Espelha verticalmente"""

return image.transpose(Image.FLIP\_TOP\_BOTTOM)

@staticmethod

def crop(image: Image.Image, region: tuple) -> Image.Image:

"""

Recorta região da imagem

Args:

region: (x1, y1, x2, y2)

"""

return image.crop(region)

@staticmethod

def resize(image: Image.Image, size: tuple, keep\_aspect: bool = True) -> Image.Image:

"""

Redimensiona imagem

Args:

size: (width, height)

keep\_aspect: Mantém proporção

"""

if keep\_aspect:

image.thumbnail(size, Image.Resampling.LANCZOS)

return image

else:

return image.resize(size, Image.Resampling.LANCZOS)

@staticmethod

def to\_qpixmap(image: Image.Image) -> QPixmap:

"""Converte PIL Image para QPixmap"""

# Converte para RGB se necessário

if image.mode != 'RGB':

image = image.convert('RGB')

# Converte para bytes

data = image.tobytes('raw', 'RGB')

qimage = QImage(data, image.width, image.height, QImage.Format\_RGB888)

return QPixmap.fromImage(qimage)

@staticmethod

def from\_qpixmap(pixmap: QPixmap) -> Image.Image:

"""Converte QPixmap para PIL Image"""

qimage = pixmap.toImage()

# Converte para RGB888

qimage = qimage.convertToFormat(QImage.Format\_RGB888)

# Extrai bytes

width = qimage.width()

height = qimage.height()

ptr = qimage.bits()

ptr.setsize(height \* width \* 3)

arr = np.frombuffer(ptr, np.uint8).reshape((height, width, 3))

return Image.fromarray(arr)

**HISTÓRICO DE TRANSFORMAÇÕES**

class TransformationHistory:

"""Gerencia histórico de transformações de imagem"""

def \_\_init\_\_(self, max\_size: int = 20):

self.history = []

self.current\_index = -1

self.max\_size = max\_size

def add(self, image: Image.Image, transformation: dict):

"""Adiciona transformação ao histórico"""

# Remove histórico futuro

if self.current\_index < len(self.history) - 1:

self.history = self.history[:self.current\_index + 1]

# Adiciona nova entrada

self.history.append({

'image': image.copy(),

'transformation': transformation

})

self.current\_index += 1

# Limita tamanho

if len(self.history) > self.max\_size:

self.history.pop(0)

self.current\_index -= 1

def undo(self) -> Optional[Image.Image]:

"""Desfaz última transformação"""

if self.current\_index > 0:

self.current\_index -= 1

return self.history[self.current\_index]['image'].copy()

return None

def redo(self) -> Optional[Image.Image]:

"""Refaz transformação desfeita"""

if self.current\_index < len(self.history) - 1:

self.current\_index += 1

return self.history[self.current\_index]['image'].copy()

return None

def can\_undo(self) -> bool:

return self.current\_index > 0

def can\_redo(self) -> bool:

return self.current\_index < len(self.history) - 1

**================================================================================ 11. CLASSES E MÉTODOS PRINCIPAIS**

**MODELO DE DADOS - Point**

from dataclasses import dataclass, field

from typing import Optional

from datetime import datetime

@dataclass

class Point:

"""Representa um ponto de medição"""

# Identificação

id: int

# Posição

x: int

y: int

# Forma

shape: str # "circle" ou "rectangle"

# Dimensões (depende da forma)

radius: Optional[int] = None

width: Optional[int] = None

height: Optional[int] = None

# Medições

reference\_value: Optional[float] = None

compare\_value: Optional[float] = None

# Metadados

created\_at: str = field(default\_factory=lambda: datetime.now().isoformat())

measured\_at: Optional[str] = None

def \_\_post\_init\_\_(self):

"""Validações após inicialização"""

if self.shape not in ["circle", "rectangle"]:

raise ValueError("shape deve ser 'circle' ou 'rectangle'")

if self.shape == "circle" and self.radius is None:

raise ValueError("circle requer radius")

if self.shape == "rectangle" and (self.width is None or self.height is None):

raise ValueError("rectangle requer width e height")

@property

def has\_reference(self) -> bool:

return self.reference\_value is not None

@property

def has\_comparison(self) -> bool:

return self.compare\_value is not None

@property

def difference\_percent(self) -> Optional[float]:

"""Calcula diferença percentual"""

if not self.has\_reference or not self.has\_comparison:

return None

if self.reference\_value == 0:

return None

diff = (self.compare\_value - self.reference\_value) / self.reference\_value \* 100

return round(diff, 2)

def is\_divergent(self, tolerance\_percent: float) -> bool:

"""Verifica se está fora da tolerância"""

diff = self.difference\_percent

if diff is None:

return False

return abs(diff) > tolerance\_percent

def to\_dict(self) -> dict:

"""Converte para dicionário (para JSON)"""

data = {

"id": self.id,

"position": {"x": self.x, "y": self.y},

"shape": {

"type": self.shape

},

"metadata": {

"created\_at": self.created\_at

}

}

# Dimensões

if self.shape == "circle":

data["shape"]["radius"] = self.radius

else:

data["shape"]["width"] = self.width

data["shape"]["height"] = self.height

# Medições

if self.has\_reference or self.has\_comparison:

data["measurements"] = {}

if self.has\_reference:

data["measurements"]["reference"] = {

"value": self.reference\_value,

"timestamp": self.measured\_at

}

if self.has\_comparison:

data["measurements"]["compare"] = {

"value": self.compare\_value

}

return data

@classmethod

def from\_dict(cls, data: dict) -> 'Point':

"""Cria Point a partir de dicionário"""

shape\_type = data["shape"]["type"]

kwargs = {

"id": data["id"],

"x": data["position"]["x"],

"y": data["position"]["y"],

"shape": shape\_type,

"created\_at": data["metadata"]["created\_at"]

}

# Dimensões

if shape\_type == "circle":

kwargs["radius"] = data["shape"]["radius"]

else:

kwargs["width"] = data["shape"]["width"]

kwargs["height"] = data["shape"]["height"]

# Medições

if "measurements" in data:

if "reference" in data["measurements"]:

kwargs["reference\_value"] = data["measurements"]["reference"]["value"]

kwargs["measured\_at"] = data["measurements"]["reference"]["timestamp"]

if "compare" in data["measurements"]:

kwargs["compare\_value"] = data["measurements"]["compare"]["value"]

return cls(\*\*kwargs)

**MODELO DE DADOS - BoardProject**

@dataclass

class BoardProject:

"""Representa um projeto completo"""

# Informações do projeto

name: str

board\_model: str

is\_fully\_functional: bool

problem\_description: str = ""

notes: str = ""

# Configurações

tolerance\_percent: float = 5.0

measurement\_profile: str = "diode\_reverse"

# Dados

image: Optional[Image.Image] = None

points: list[Point] = field(default\_factory=list)

# Metadados

created\_at: str = field(default\_factory=lambda: datetime.now().isoformat())

modified\_at: str = field(default\_factory=lambda: datetime.now().isoformat())

measured\_at: Optional[str] = None

def \_\_post\_init\_\_(self):

"""Validações"""

if len(self.name) < 3:

raise ValueError("Nome deve ter pelo menos 3 caracteres")

if len(self.board\_model) < 2:

raise ValueError("Modelo deve ter pelo menos 2 caracteres")

if not self.is\_fully\_functional and len(self.problem\_description) < 10:

raise ValueError("Descrição do problema é obrigatória para placas não funcionais")

if not (0.1 <= self.tolerance\_percent <= 50.0):

raise ValueError("Tolerância deve estar entre 0.1% e 50.0%")

def update\_modified\_timestamp(self):

"""Atualiza timestamp de modificação"""

self.modified\_at = datetime.now().isoformat()

def get\_statistics(self) -> dict:

"""Retorna estatísticas do projeto"""

return {

"total\_points": len(self.points),

"measured\_points": sum(1 for p in self.points if p.has\_reference),

"compared\_points": sum(1 for p in self.points if p.has\_