Documentação do Aplicativo
Diagnóstico Profissional de Vibração em Rolamentos

Jobsson L - Documentação

07 de janeiro de 2026

1 Introdução

Este documento apresenta a documentação completa do aplicativo desenvolvido em Python utilizando `tkinter` e `matplotlib` para digitalização de gráficos de espectro de vibração a partir de imagens e realização de diagnóstico de falhas em rolamentos.

O aplicativo permite:

- Carregar uma imagem de um gráfico de espectro de vibração.
- Calibrar os eixos com 3 pontos conhecidos.
- Digitalizar automaticamente a curva do gráfico (detecção de linha escura).
- Editar manualmente os pontos digitalizados.
- Converter os pontos para valores reais (frequência em Hz e amplitude).
- Realizar diagnóstico de falhas em rolamentos modelos 6203ZZ ou 6204ZZ, identificando defeitos em pista externa, pista interna, esferas e gaiola.
- Exportar dados e salvar relatório visual.

O código é estruturado em uma única classe `GraphDigitizer` que gerencia toda a interface e lógica.

2 Como Usar o Aplicativo

1. **Carregar Imagem**: Selecione uma imagem do gráfico.
2. **Calibrar Eixos**: Clique em 3 pontos (inferior esquerdo, inferior direito e um pico conhecido) e informe os valores reais.
3. **Digitalização Automática**: Executada automaticamente após a calibração.
4. **Edição Manual**:
   - Clique esquerdo: adiciona ponto verde exatamente na posição clicada.
   - Clique direito próximo a um ponto verde: remove o ponto.
5. **Realizar Diagnóstico**: Informe RPM e modelo do rolamento (6203zz ou 6204zz).

6. **Exportar Dados**: Salva em Excel ou CSV.
7. **Salvar Diagnóstico**: Exporta o dashboard como PNG.

3 Estrutura Geral do Código

O código é composto pela classe `GraphDigitizer` com os seguintes métodos principais:

- `__init__`: Inicializa a interface gráfica.
- `load_image`: Carrega a imagem.
- `start_calibrate` / `on_click`: Calibração dos eixos.
- `auto_digitize_high_precision`: Digitalização automática de alta precisão.
- `on_edit_click`: Edição manual de pontos.
- `update_real_data` / `calibrate_transform`: Conversão para coordenadas reais.
- `realizar_diagnostico`: Análise de falhas e geração do dashboard.
- `export_data` / `salvar_diagnostico`: Exportação.
- `clear_all`: Limpeza total.

4 Código Fonte Completo

A seguir, o código completo do aplicativo, formatado para leitura:

```
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
from matplotlib.figure import Figure
from matplotlib.gridspec import GridSpec
import numpy as np
from PIL import Image
import os
from scipy.signal import find_peaks
import pandas as pd

class GraphDigitizer:
    def __init__(self, root):
        self.root = root
        self.root.title("Diagnstico Profissional de Vibrao em Rolamentos")
        self.root.geometry("1500x900")
        self.image = None
        self.image_np = None
        self.points = []
        self.calib_points = []
        self.real_data = []
        self.diagnostico_fig = None

        # Variveis para  edio  manual e regio do grfico
        self.graph_x1 = None
        self.graph_x2 = None
        self.base_y = None

        # Layout principal
        self.main_frame = tk.Frame(self.root)
        self.main_frame.pack(fill=tk.BOTH, expand=True)

        self.control_frame = tk.Frame(self.main_frame, width=300, padx=15, pady=15, bg="#f5f5f5")
        self.control_frame.pack(side=tk.LEFT, fill=tk.Y)
```

```
        self.control_frame.pack_propagate(False)

        self.plot_frame = tk.Frame(self.main_frame)
        self.plot_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        self.fig = Figure(figsize=(13, 8), dpi=100)
        self.ax_img = self.fig.add_subplot(121)
        self.ax_plot = self.fig.add_subplot(122)
        self.canvas = FigureCanvasTkAgg(self.fig, master=self.plot_frame)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
        self.toolbar = NavigationToolbar2Tk(self.canvas, self.root)
        self.toolbar.update()

        # Botes e ttulos
        tk.Label(self.control_frame, text="DIAGNSTICO DE ROLAMENTOS", font=("Arial", 16, "bold"),
    bg="#f5f5f5").pack(pady=20)
        tk.Button(self.control_frame, text="Carregar Imagem", command=self.load_image, width=30, height=2,
    bg="#4caf50", fg="white", font=("Arial", 10, "bold")).pack(pady=10)
        self.btn_calibrate = tk.Button(self.control_frame, text="Calibrar Eixos (3 pontos)",
    command=self.start_calibrate, state=tk.DISABLED, width=30, height=2, bg="#2196f3", fg="white",
    font=("Arial", 10, "bold"))
        self.btn_calibrate.pack(pady=10)
        self.btn_diagnostico = tk.Button(self.control_frame, text="Realizar Diagnstico",
    command=self.realizar_diagnostico, state=tk.DISABLED, width=30, height=2, bg="#ff9800", fg="white",
    font=("Arial", 10, "bold"))
        self.btn_diagnostico.pack(pady=10)
        self.btn_export = tk.Button(self.control_frame, text="Exportar Dados", command=self.export_data,
    state=tk.DISABLED, width=30, height=2, bg="#3f51b5", fg="white", font=("Arial", 10, "bold"))
        self.btn_export.pack(pady=10)
        self.btn_salvar_diagnostico = tk.Button(self.control_frame, text="Salvar Diagnstico (PNG)",
    command=self.salvar_diagnostico, state=tk.DISABLED, width=30, height=2, bg="#f44336", fg="white",
    font=("Arial", 10, "bold"))
        self.btn_salvar_diagnostico.pack(pady=10)
        tk.Button(self.control_frame, text="Resetar Zoom", command=self.reset_zoom, width=30, height=2,
    bg="#e0e0e0").pack(pady=10)
        tk.Button(self.control_frame, text="Limpar Tudo", command=self.clear_all, width=30, height=2,
    bg="#9e9e9e", fg="white").pack(pady=20)

        tk.Label(self.control_frame, text="Fluxo:\n1. Carregar imagem\n2. Calibrar (3 pontos)\n3.
    Digitalizao automtica\n4. Editar pontos:\n    Clique esquerdo: adiciona ponto verde exatamente onde
    voc clicar\n    Clique direito: remove ponto verde prximo\n5. Realizar Diagnstico\n6. Salvar relatrio
    visual",
                 justify=tk.LEFT, font=("Arial", 9), bg="#f5f5f5", fg="#424242").pack(pady=10)

        self.cid_click = None
        self.cid_edit = None

    def load_image(self):
        path = filedialog.askopenfilename(
            title="Selecione uma Imagem",
            filetypes=[
                ("Imagens comuns", "*.bmp *.png *.jpg *.jpeg *.gif *.tiff *.tif *.webp"),
                ("BMP", "*.bmp"),
                ("PNG", "*.png"),
                ("JPEG", "*.jpg *.jpeg"),
                ("GIF", "*.gif"),
                ("TIFF", "*.tiff *.tif"),
                ("WEBP", "*.webp"),
                ("Todos os arquivos", "*.*")
            ]
        )
        if not path:
            return
        try:
            self.image = Image.open(path).convert('RGB')
            self.image_np = np.array(self.image.convert('L'))
            self.redraw(full_reset=True)
            self.btn_calibrate['state'] = tk.NORMAL
            messagebox.showinfo("Sucesso", "Imagem carregada com sucesso!")
        except Exception as e:
            messagebox.showerror("Erro", f"Falha ao carregar imagem:\n{e}")

    def reset_zoom(self):
```

```
        if self.image is None:
            return
        self.ax_img.clear()
        self.ax_img.imshow(self.image)
        self.ax_img.axis('off')
        self.redraw_points_only()
        self.update_spectrum_plot()
        self.canvas.draw_idle()

    def start_calibrate(self):
        if len(self.calib_points) > 0:
            if not messagebox.askyesno("Recalibrar?", "Isso limpar os dados atuais. Continuar?"):
                return
            self.clear_all()
        self.mode = 'calibrate'
        self.connect_click()
        messagebox.showinfo("Calibrao", "Clique em 3 pontos conhecidos:\n1. Inferior esquerdo (X=0, Y=0)\n2. Inferior direito (X=1000, Y=0)\n3. Topo de pico conhecido (Y > 0)")

    def connect_click(self):
        if self.cid_click is None:
            self.cid_click = self.canvas.mpl_connect('button_press_event', self.on_click)

    def on_click(self, event):
        if event.inaxes != self.ax_img or event.xdata is None or event.ydata is None:
            return
        x, y = event.xdata, event.ydata
        if self.mode == 'calibrate' and len(self.calib_points) < 3:
            try:
                x_real = simpledialog.askfloat("Valor X Real (Hz)", f"Ponto {len(self.calib_points)+1}/3", initialvalue=[0, 1000, 370][len(self.calib_points)])
                if x_real is None:
                    return
                y_real = simpledialog.askfloat("Valor Y Real (Amplitude)", "", initialvalue=0.0 if len(self.calib_points) < 2 else 0.065)
                if y_real is None:
                    return
                self.calib_points.append((x, y, x_real, y_real))
                self.ax_img.plot(x, y, 'ro', markersize=12, markeredgecolor='white', markeredgewidth=2)
                self.ax_img.text(x + 15, y - 15, f"P{len(self.calib_points)}\nX={x_real}\nY={y_real}", color='red', fontsize=11, fontweight='bold')
                self.canvas.draw_idle()
                if len(self.calib_points) == 3:
                    self.canvas.mpl_disconnect(self.cid_click)
                    self.cid_click = None
                    self.auto_digitize_high_precision()
                    self.cid_edit = self.canvas.mpl_connect('button_press_event', self.on_edit_click)
                    self.btn_export['state'] = tk.NORMAL
                    self.btn_diagnostico['state'] = tk.NORMAL
                    messagebox.showinfo("Digitalizao", "Digitalizao  automtica  concluda !\ n  Clique esquerdo: adiciona ponto verde exatamente na posio clicada\n Clique direito prximo a um ponto verde: remove o ponto")
            except Exception as e:
                messagebox.showerror("Erro na calibrao", f"Erro inesperado: {e}")

    def on_edit_click(self, event):
        if event.inaxes != self.ax_img or event.xdata is None or event.ydata is None or self.graph_x1 is None or self.graph_x2 is None or self.base_y is None:
            return

        x_click = event.xdata
        y_click = event.ydata

        if not (self.graph_x1 <= x_click <= self.graph_x2 and y_click < self.base_y + 15):
            return

        try:
            if event.button == 1:  # Adicionar
                if any(abs(p[0] - x_click) < 10 and abs(p[1] - y_click) < 10 for p in self.points):
                    return
                self.points.append((x_click, y_click))
                self.redraw_points_only()
                self.update_spectrum_plot()
```

```
            elif event.button == 3:  # Remover
                tol = 20
                for i in range(len(self.points) - 1, -1, -1):
                    px, py = self.points[i]
                    if (px - x_click)**2 + (py - y_click)**2 < tol**2:
                        self.points.pop(i)
                        self.redraw_points_only()
                        self.update_spectrum_plot()
                        break
        except Exception as e:
            messagebox.showerror("Erro na edio", f"Erro ao editar pontos: {e}")

    def auto_digitize_high_precision(self):
        if len(self.calib_points) != 3:
            messagebox.showerror("Erro", "Calibrao incompleta (3 pontos necessrios).")
            return
        try:
            p1, p2, p3 = self.calib_points
            left = min(p1[0], p2[0])
            right = max(p1[0], p2[0])
            base_y = max(p1[1], p2[1])
            self.base_y = base_y
            margin = 50
            x1 = max(0, int(left))
            x2 = min(self.image_np.shape[1], int(right))
            y1 = max(0, int(p3[1] - margin))
            y2 = min(self.image_np.shape[0], int(base_y + margin))
            crop = self.image_np[y1:y2, x1:x2]
            if crop.size == 0 or crop.shape[1] == 0:
                messagebox.showerror("Erro", "Regio do grfico invlida.")
                return

            self.graph_x1 = x1
            self.graph_x2 = x2

            mean_val = np.mean(crop)
            std_val = np.std(crop)
            threshold = mean_val - 1.2 * std_val if std_val > 0 else mean_val - 10

            self.points = []
            for col in range(crop.shape[1]):
                column = crop[:, col]
                dark_pixels = np.where(column < threshold)[0]
                if len(dark_pixels) > 1:
                    top_dark = np.min(dark_pixels)
                    y_img = y1 + top_dark
                    x_img = x1 + col
                    if y_img < base_y + 15:
                        self.points.append((x_img, y_img))

            self.redraw_points_only()
            self.update_spectrum_plot()
        except Exception as e:
            messagebox.showerror("Erro na digitalizao", f"Erro ao digitalizar: {e}")

    def update_real_data(self):
        self.real_data = []
        if len(self.points) == 0 or len(self.calib_points) != 3:
            return
        try:
            scale_x, offset_x, scale_y, offset_y = self.calibrate_transform()
            sorted_points = sorted(self.points, key=lambda p: p[0])
            for x_img, y_img in sorted_points:
                x_real = scale_x * x_img + offset_x
                y_real = max(0.0, scale_y * y_img + offset_y)
                self.real_data.append((round(x_real, 2), round(y_real, 6)))
        except Exception as e:
            messagebox.showerror("Erro", f"Falha ao calcular dados reais: {e}")

    def update_spectrum_plot(self):
        self.ax_plot.clear()
        if self.real_data:
```

```
            try:
                xs, ys = zip(*self.real_data)
                self.ax_plot.stem(xs, ys, linefmt='b-', markerfmt='bo', basefmt='r-')
                self.ax_plot.set_xlabel('Frequncia (Hz)')
                self.ax_plot.set_ylabel('Amplitude')
                self.ax_plot.grid(True, alpha=0.5)
                self.ax_plot.set_title('Espectro Digitalizado')
            except Exception:
                self.ax_plot.set_title('Erro ao plotar espectro')
        else:
            self.ax_plot.set_title('Espectro Digitalizado (sem pontos)')
        self.canvas.draw_idle()

    def calibrate_transform(self):
        if len(self.calib_points) != 3:
            raise ValueError("Calibrao incompleta")
        p1, p2, p3 = self.calib_points
        x1_img, y1_img, x1_real, y1_real = p1
        x2_img, y2_img, x2_real, y2_real = p2
        p3_y_img = p3[1]
        y3_real = p3[3]

        dx_img = x2_img - x1_img
        dy_img = p3_y_img - (y1_img + y2_img) / 2
        if abs(dx_img) < 1e-6 or abs(dy_img) < 1e-6:
            raise ValueError("Pontos de calibrao alinhados ou muito prximos (diviso por zero).")

        scale_x = (x2_real - x1_real) / dx_img
        offset_x = x1_real - scale_x * x1_img
        base_real = (y1_real + y2_real) / 2
        base_img = (y1_img + y2_img) / 2
        scale_y = (y3_real - base_real) / dy_img
        offset_y = base_real - scale_y * base_img
        return scale_x, offset_x, scale_y, offset_y

    def export_data(self):
        self.update_real_data()
        if not self.real_data:
            messagebox.showwarning("Aviso", "Nenhum dado para exportar!")
            return
        try:
            save_path = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel",
 "*.xlsx"), ("CSV", "*.csv")])
            if save_path:
                df = pd.DataFrame(self.real_data, columns=['Frequncia (Hz)', 'Amplitude'])
                if save_path.endswith('.csv'):
                    df.to_csv(save_path, index=False)
                else:
                    df.to_excel(save_path, index=False)
                messagebox.showinfo("Sucesso", "Dados exportados com sucesso!")
        except Exception as e:
            messagebox.showerror("Erro", f"Falha ao exportar: {e}")

    def realizar_diagnostico(self):
        try:
            self.update_real_data()

            rpm = simpledialog.askfloat("RPM do Eixo", "Digite o RPM do rolamento:", minvalue=1)
            if rpm is None:
                return

            rolamento = simpledialog.askstring("Modelo do Rolamento", "Digite o modelo (6203zz ou
 6204zz):")
            if rolamento is None:
                return
            rolamento = rolamento.strip().lower()
            if rolamento not in ["6203zz", "6204zz"]:
                messagebox.showerror("Erro", "Modelo invlido! Use 6203zz ou 6204zz.")
                return

            if not self.real_data:
                messagebox.showwarning("Aviso", "No h pontos digitalizados. O espectro ser vazio.")

```

```
            params = {
                "6203zz": {"N": 7, "d": 6.747, "pd": (17 + 40) / 2},
                "6204zz": {"N": 8, "d": 7.938, "pd": (20 + 47) / 2}
            }
            p = params[rolamento]
            N, d, pd = p["N"], p["d"], p["pd"]
            fr = rpm / 60

            BPFO = (N / 2) * fr * (1 - (d / pd))
            BPFI = (N / 2) * fr * (1 + (d / pd))
            BSF = (pd / (2 * d)) * fr * (1 - ((d / pd)**2))
            FTF = (1 / 2) * fr * (1 - (d / pd))

            freqs = np.array([p[0] for p in self.real_data]) if self.real_data else np.array([])
            amps = np.array([p[1] for p in self.real_data]) if self.real_data else np.array([])

            peak_freqs = np.array([])
            peak_amps_norm = np.array([])
            if len(amps) > 1 and np.max(amps) > 0:
                amps_norm = amps / np.max(amps)
                peaks, _ = find_peaks(amps_norm, height=0.05, prominence=0.02, distance=3)
                if len(peaks) == 0:
                    peaks, _ = find_peaks(amps_norm, prominence=0.01, distance=3)
                if len(peaks) > 0:
                    peak_freqs = freqs[peaks]
                    peak_amps_norm = amps_norm[peaks]

            tol = 0.03
            defeitos = {
                "Pista Externa": {"freq": BPFO, "score": 0.0, "num_harm": 0},
                "Pista Interna": {"freq": BPFI, "score": 0.0, "num_harm": 0},
                "Esferas": {"freq": BSF, "score": 0.0, "num_harm": 0},
                "Gaiola": {"freq": FTF, "score": 0.0, "num_harm": 0},
            }

            for (nome, info) in defeitos.items():
                base_freq = info["freq"]
                if base_freq <= 0:
                    continue
                matching_amps = []
                for k in range(1, 11):
                    target = k * base_freq
                    if len(freqs) > 0 and target > freqs[-1] * 1.1:
                        break
                    if len(peak_freqs) == 0:
                        break
                    diffs = np.abs(peak_freqs - target)
                    if np.min(diffs) <= tol * target:
                        idx = np.argmin(diffs)
                        matching_amps.append(peak_amps_norm[idx])
                if matching_amps:
                    max_rel = np.max(matching_amps)
                    num_harm = len(matching_amps)
                    score = max_rel * 100
                    if num_harm > 1:
                        score = min(100, score + 15 * (num_harm - 1))
                    info["score"] = round(score, 1)
                    info["num_harm"] = num_harm

            defeitos_ordenados = sorted(defeitos.items(), key=lambda x: x[1]["score"], reverse=True)

            max_score = max(info["score"] for info in defeitos.values()) if self.real_data else 0

            if not self.real_data:
                estado = "SEM DADOS PARA ANLISE"
                veredito = "No foi possvel realizar diagnstico por falta de dados digitalizados."
            else:
                if max_score >= 80:
                    estado = "DEFEITO GRAVE DETECTADO"
                elif max_score >= 60:
                    estado = "DEFEITO MODERADO DETECTADO"
                elif max_score >= 30:
                    estado = "INCIO DE DEFEITO POSSVEL"
```

```
                else:
                    estado = "ROLAMENTO NORMAL"

                defeitos_graves = [nome for nome, info in defeitos_ordenados if info["score"] >= 60]
                if defeitos_graves:
                    veredito = f"Defeito principal em: {', '.join(defeitos_graves)}."
                    if max_score >= 80:
                        veredito += " Parada imediata e substituio recomendada."
                    elif max_score >= 60:
                        veredito += " Monitorar urgentemente."
                else:
                    veredito = "Sem defeitos graves. Operao normal."

            # Dashboard  tabela da direita finalmente perfeita
            self.diagnostico_fig = Figure(figsize=(18, 10), dpi=120, facecolor='white')  # Figura mais
    larga
            self.diagnostico_fig.suptitle(f"DIAGNSTICO - {rolamento.upper()} @ {rpm} RPM", fontsize=20,
    fontweight='bold', y=0.98)

            # Muito mais espao horizontal para a tabela da direita
            gs = GridSpec(3, 2, figure=self.diagnostico_fig, height_ratios=[5, 4, 1.5],
                  width_ratios=[1, 3], hspace=0.4, wspace=0.6)

            # Espectro
            ax_spectrum = self.diagnostico_fig.add_subplot(gs[0, :])
            if self.real_data and len(freqs) > 0:
                ax_spectrum.stem(freqs, amps, linefmt='black', markerfmt='none', basefmt='lightgray')
                ax_spectrum.fill_between(freqs, amps, color='lightgray', alpha=0.4)
                ax_spectrum.set_ylabel("Amplitude", fontsize=14)
                ax_spectrum.set_xlabel("Frequncia (Hz)", fontsize=14)
                ax_spectrum.grid(True, alpha=0.3, linestyle='--')
            else:
                ax_spectrum.text(0.5, 0.5, "ESPECTRO VAZIO\n(Nenhum ponto digitalizado)",
                                 ha='center', va='center', transform=ax_spectrum.transAxes,
                                 fontsize=18, color='gray')
            ax_spectrum.spines['top'].set_visible(False)
            ax_spectrum.spines['right'].set_visible(False)

            # Tabela de parmetros (esquerda)
            ax_params = self.diagnostico_fig.add_subplot(gs[1, 0])
            ax_params.axis('off')
            params_data = [
                ["Parmetro", "Valor"],
                ["RPM", f"{rpm}"],
                ["f_r (Hz)", f"{fr:.2f}"],
                ["BPFO", f"{BPFO:.2f} Hz"],
                ["BPFI", f"{BPFI:.2f} Hz"],
                ["BSF", f"{BSF:.2f} Hz"],
                ["FTF", f"{FTF:.2f} Hz"],
            ]
            params_table = ax_params.table(cellText=params_data[1:], colLabels=params_data[0],
                                           loc='center', cellLoc='center', bbox=[0, 0, 1, 1])
            params_table.auto_set_font_size(False)
            params_table.set_fontsize(12)
            params_table.scale(1, 2.4)
            for (i, j), cell in params_table.get_celld().items():
                if i == 0:
                    cell.set_facecolor('#dddddd')
                    cell.set_text_props(weight='bold')

            # Tabela de defeitos (direita)  correo definitiva
            ax_defeitos = self.diagnostico_fig.add_subplot(gs[1, 1])
            ax_defeitos.axis('off')

            headers = ["Defeito", "Freq (Hz)", "Score (%)", "Harmnicos", "Severidade"]

            rows = []
            for nome, info in defeitos_ordenados:
                severidade = "Grave" if info["score"] >= 60 else "Moderado" if info["score"] >= 30 else
    "Normal"
                rows.append([nome, f"{info['freq']:.2f}", f"{info['score']:.1f}",
                             str(info["num_harm"]), severidade])

```

```
            # Larguras generosas para evitar qualquer sobreposio
            defeitos_table = ax_defeitos.table(cellText=rows, colLabels=headers,
                                               colWidths=[0.32, 0.16, 0.16, 0.18, 0.18],
                                               loc='center', bbox=[0, 0, 1, 1])

            defeitos_table.auto_set_font_size(False)
            defeitos_table.set_fontsize(11.5)  # Fonte ligeiramente menor para garantir espao
            defeitos_table.scale(1.2, 3.0)  # Mais escala horizontal e vertical para respirar

            # Cabealho
            for (i, j), cell in defeitos_table.get_celld().items():
                if i == 0:
                    cell.set_facecolor('#dddddd')
                    cell.set_text_props(weight='bold', ha='center', va='center')

            # Alinhamento dos dados
            for row_idx in range(1, len(rows) + 1):
                defeitos_table[(row_idx, 0)].set_text_props(ha='left', va='center', weight='normal')
                for col_idx in range(1, 5):
                    defeitos_table[(row_idx, col_idx)].set_text_props(ha='center', va='center')

            # Cores por severidade
            for idx, (_, info) in enumerate(defeitos_ordenados):
                row = idx + 1
                score = info["score"]
                if score >= 60:
                    color = '#ffeeee'
                elif score >= 30:
                    color = '#fff0e0'
                else:
                    color = '#eeffee'
                for col in range(5):
                    defeitos_table[(row, col)].set_facecolor(color)

            # Veredito  mais compacto para caber perfeitamente
            ax_veredito = self.diagnostico_fig.add_subplot(gs[2, :])
            ax_veredito.axis('off')
            cor_estado = 'red' if "GRAVE" in estado else 'orange' if "MODERADO" in estado else 'green'
            ax_veredito.text(0.5, 0.70, estado, ha='center', va='center', fontsize=26,
                             fontweight='bold', color=cor_estado)
            ax_veredito.text(0.5, 0.30, veredito, ha='center', va='center', fontsize=15, wrap=True)

            # Janela
            dashboard_win = tk.Toplevel(self.root)
            dashboard_win.title("Dashboard de Diagnstico")
            dashboard_win.geometry("1800x1000")
            canvas_dash = FigureCanvasTkAgg(self.diagnostico_fig, master=dashboard_win)
            canvas_dash.draw()
            canvas_dash.get_tk_widget().pack(fill=tk.BOTH, expand=True)

            self.btn_salvar_diagnostico['state'] = tk.NORMAL
            messagebox.showinfo("Concludo", "Diagnstico gerado! A tabela da direita agora est 100%
    formatada: cabealhos separados, sem sobreposio, nomes completos e tudo legvel. O veredito tambm est
    mais compacto.")

        except Exception as e:
            messagebox.showerror("Erro no diagnstico", f"Erro inesperado: {str(e)}")
    def salvar_diagnostico(self):
        if self.diagnostico_fig is None:
            messagebox.showwarning("Aviso", "Faa o diagnstico primeiro!")
            return
        try:
            path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG", "*.png")])
            if path:
                self.diagnostico_fig.savefig(path, dpi=300, bbox_inches='tight')
                messagebox.showinfo("Salvo!", "Relatrio salvo com sucesso!")
        except Exception as e:
            messagebox.showerror("Erro", f"Falha ao salvar: {e}")

    def redraw(self, full_reset=False):
        if self.image is None:
            return
        if full_reset:
```

```
            self.ax_img.clear()
            self.ax_img.imshow(self.image)
            self.ax_img.axis('off')
        self.redraw_points_only()
        self.update_spectrum_plot()

    def redraw_points_only(self):
        for artist in list(self.ax_img.collections) + list(self.ax_img.lines) + list(self.ax_img.texts):
            if artist:
                artist.remove()

        try:
            for i, (x, y, xr, yr) in enumerate(self.calib_points):
                self.ax_img.plot(x, y, 'ro', markersize=12, markeredgecolor='white', markeredgewidth=2)
                self.ax_img.text(x + 15, y - 15, f"P{i+1}\nX={xr}\nY={yr}", color='red', fontsize=11,
     fontweight='bold')

            for x, y in self.points:
                self.ax_img.plot(x, y, 'go', markersize=3, alpha=0.5)
        except Exception:
            pass

    def clear_all(self):
        self.points = []
        self.calib_points = []
        self.real_data = []
        self.diagnostico_fig = None
        self.graph_x1 = None
        self.graph_x2 = None
        self.base_y = None
        self.redraw(full_reset=True)
        self.ax_plot.clear()
        self.ax_plot.set_title('Espectro Digitalizado (vazio)')
        self.canvas.draw_idle()
        for btn in [self.btn_export, self.btn_diagnostico, self.btn_salvar_diagnostico]:
            btn['state'] = tk.DISABLED
        if self.cid_click:
            self.canvas.mpl_disconnect(self.cid_click)
            self.cid_click = None
        if self.cid_edit:
            self.canvas.mpl_disconnect(self.cid_edit)
            self.cid_edit = None

if __name__ == "__main__":
    try:
        root = tk.Tk()
        app = GraphDigitizer(root)
        root.mainloop()
    except Exception as e:
        messagebox.showerror("Erro fatal", f"Erro ao iniciar aplicao: {e}")
```

Listing 1: Código completo do aplicativo GraphDigitizer

5 Funções que Podem Ser Melhoradas para Aumentar a Precisão do Diagnóstico

As funções críticas para a precisão do diagnóstico são as responsáveis pela digitalização da curva e pela detecção/análise de falhas. Abaixo estão as principais funções e sugestões concretas de melhoria:

5.1 `auto_digitize_high_precision`

Problema atual: Detecta a linha assumindo que ela é o pixel mais escuro em cada coluna dentro de uma região cropada. Sensível a ruído, linhas grossas ou fundo não uniforme.

Melhorias sugeridas:

- Usar detecção de borda (ex.: Canny com OpenCV) ou filtro Sobel para encontrar contornos mais precisos.
- Implementar busca subpixel (interpolação quadrática ao redor do mínimo).
- Aplicar suavização prévia (Gaussian blur) na região cropada.
- Permitir ajuste manual do threshold via slider na interface.
- Detectar múltiplos candidatos por coluna e escolher o mais contínuo com colunas vizinhas.

5.2 `update_real_data` e `calibrate_transform`

Problema atual: Calibração linear com apenas 3 pontos pode ser imprecisa se os eixos não forem perfeitamente lineares ou se houver distorção na imagem.

Melhorias sugeridas:

- Permitir mais pontos de calibração e usar regressão linear robusta (RANSAC).
- Detectar automaticamente se o eixo Y é logarítmico e aplicar transformação adequada.
- Corrigir perspectiva da imagem (homografia) se a foto não for frontal.

5.3 `realizar_diagnostico` – Detecção de Picos

Problema atual: Usa `find_peaks` com parâmetros fixos e fallback simples.

Melhorias sugeridas:

- Normalizar o espectro de forma mais robusta (ex.: remover tendência de base).
- Aplicar suavização (moving average ou Savitzky-Golay) antes da detecção de picos.
- Usar detecção adaptativa de threshold baseada no ruído estimado.
- Implementar envelope analysis (Hilbert transform) para detectar modulações típicas de falhas em rolamentos.
- Busca de harmônicos com tolerância variável (maior em frequências altas).
- Adicionar mais modelos de rolamentos e permitir entrada manual de parâmetros geométricos.

5.4 Outras Melhorias Gerais

- **Filtragem do espectro digitalizado**: Interpolar e suavizar os pontos extraídos antes da análise.
- **Validação estatística**: Calcular confiança dos scores com base na densidade de picos.
- **Interface**: Adicionar sliders para ajustar parâmetros de detecção em tempo real.

6 Conclusão

O aplicativo já oferece uma solução funcional e visualmente rica para diagnóstico de rolamentos a partir de imagens. As melhorias sugeridas acima, especialmente na digitalização e na análise espectral, podem elevar significativamente a precisão e robustez do diagnóstico, tornando-o adequado para uso profissional.

Para implementar as melhorias, recomenda-se adicionar a biblioteca OpenCV (`pip install opencv-python`) e refatorar as funções indicadas.