

```

In [1]: import os
import tkinter as tk
from tkinter import filedialog, messagebox

import math
import numpy as np
import cv2
import pydicom
from PIL import Image, ImageTk
from scipy.signal import find_peaks
from scipy.optimize import curve_fit

import matplotlib
matplotlib.use("TkAgg")
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.widgets import Slider
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2

def vysoky_kontrast(image_original, image_value):

    # 1 - tvorba preview

    image_norm = cv2.normalize(image_value, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
    preview = cv2.cvtColor(image_norm, cv2.COLOR_GRAY2BGR)

    # 2 - odstranění šumu, zvýraznění hran a nalezení kontur

    blurred = cv2.GaussianBlur(image_norm, (5, 5), 0)
    edges = cv2.Canny(blurred, 5, 20)
    contours, _ = cv2.findContours(edges, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    # 3 - hledání kruhové oblasti fantomu

    contour_circ = None
    max_area = 0

    for cnt in contours:
        area = cv2.contourArea(cnt)
        perimeter = cv2.arcLength(cnt, True)

        if perimeter == 0:
            continue

        circularity = 4 * np.pi * (area / (perimeter * perimeter))

        if circularity > 0.8 and area > max_area:
            contour_circ = cnt
            max_area = area

    center_circ, R_phantom = cv2.minEnclosingCircle(contour_circ)
    S_phantom = (int(round(center_circ[0], 0)), int(round(center_circ[1], 0)))

    cv2.drawContours(preview, [contour_circ], 0, (255, 0, 0), 2)

```

```

cv2.circle(preview, S_phantom, 1, (255, 0, 0), 2)

# 4 - hledání centrálního obdélníku

ratio_rect = 308/255    ### hard-coded konstanta - poměr stran centrálního obdélníku

contour_rect = None
max_area = 0

for cnt in contours:
    rect = cv2.minAreaRect(cnt)
    center_rect, (w_rect, h_rect), angle_rect = rect

    area = cv2.contourArea(cnt)

    if w_rect == 0 or h_rect == 0:
        continue

    else:
        r = max(w_rect, h_rect)/min(w_rect, h_rect)

        if round(r, 2) == round(ratio_rect, 2) and area > max_area:
            max_area = area
            contour_rect = cnt

center_rect, size_rect, angle_rect = cv2.minAreaRect(contour_rect)
size_rect = (int(round(size_rect[0], 0)), int(round(size_rect[1], 0)))
S_rect = (int(round(center_rect[0], 0)), int(round(center_rect[1], 0)))

cv2.drawContours(preview, [contour_rect], 0, (255, 0, 0), 1)
cv2.circle(preview, S_rect, 1, (255, 0, 0), 2)

# 5 - otočení a ořez centrálního obdélníku

    ## otočení obrazu
    (h, w) = image_value.shape[:2]
    M = cv2.getRotationMatrix2D(S_rect, angle_rect, 1.0)
    rotated = cv2.warpAffine(image_value, M, (w, h))    #rotace vy

    ## ořez
    image_value_rect = cv2.getRectSubPix(rotated, size_rect, S_rect)

# 6 - hledání čtverce v centrálním obdélníku

image_norm_rect = cv2.normalize(image_value_rect, None, 0, 255, cv2.NORM_MINMAX)
blurred_rect = cv2.GaussianBlur(image_norm_rect, (5, 5), 0)
edges_rect = cv2.Canny(blurred_rect, 5, 20)
contours_rect, _ = cv2.findContours(edges_rect, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    ## hledání světlého čtverce
    contour_white_sqr = None
    max_area = 0
    mean = 0

    max_value = np.max(image_value_rect)
    min_value = np.min(image_value_rect)

```

```

mean_value = min_value + (max_value-min_value)/2

for cnt in contours_rect:
    mask = np.zeros(image_value_rect.shape, dtype=np.uint8)
    cv2.drawContours(mask, [cnt], -1, color=255, thickness=-1)
    mean_val = cv2.mean(image_value_rect, mask=mask)[0]
    if mean_val >= mean_value:
        mean = mean_val
        area = cv2.contourArea(cnt)
        if area > max_area:
            max_area = area
            contour_white_sqr = cnt

center_white_sqr, size_sqr, angle_sqr = cv2.minAreaRect(contour_white_sqr)
S_white_sqr = (int(round(center_white_sqr[0], 0)), int(round(center_white_sqr[1], 0)))

    ## hledání tmavého čtverce
contour_black_sqr = None
max_area = 0
mean = 0

max_value = np.max(image_value_rect)
min_value = np.min(image_value_rect)

mean_value = min_value + (max_value-min_value)/2

for cnt in contours_rect:
    mask = np.zeros(image_value_rect.shape, dtype=np.uint8)
    cv2.drawContours(mask, [cnt], -1, color=255, thickness=-1)
    mean_val = cv2.mean(image_value_rect, mask=mask)[0]
    area = cv2.contourArea(cnt)
    _, size, __ = cv2.minAreaRect(cnt)
    if round(size[0], 1) == round(size[1], 1):
        if area > max_area:
            max_area = area
            contour_black_sqr = cnt

center_black_sqr, size_black_sqr, angle_black_sqr = cv2.minAreaRect(contour_black_sqr)
S_black_sqr = (int(round(center_black_sqr[0], 0)), int(round(center_black_sqr[1], 0)))
R_circ = int(round(size_black_sqr[0]/3, 0))    ### poloměr kružnice pro výpočet

mask_black = np.zeros_like(image_value_rect, dtype=np.uint8)
cv2.circle(mask_black, S_black_sqr, R_circ, 255, -1) # -1 = vyplněná kružnice
values_black = image_value_rect[mask_black == 255]
black_value_rect = values_black.mean()

    ## dopočítání polohy a signálu šedého čtverce

A = np.array([S_white_sqr[0], S_white_sqr[1]])
B = np.array([S_black_sqr[0], S_black_sqr[1]])
v = B - A
C = B + v

mask_gray = np.zeros_like(image_value_rect, dtype=np.uint8)
cv2.circle(mask_gray, C, R_circ, 255, -1) # -1 = vyplněná kružnice

```

```

values_gray = image_value_rect[mask_gray == 255]
gray_value_rect = values_gray.mean()

# 7 - otočení obdélníku světlým čtvercem v pravém dolním kvadrantu

h, w = size_rect
C_x = h/2
C_y = w/2
x, y = S_white_sqr

if x > C_x and y < C_y:
    image_value_rect = cv2.flip(image_value_rect, 0)
    y = h - y
elif x < C_x and y < C_y:
    image_value_rect = cv2.flip(image_value_rect, -1)
    x = w - x
    y = h - y
elif x < C_x and y > C_y:
    image_value_rect = cv2.flip(image_value_rect, 1)
    x = w - x

S_white_sqr = (x, y)

# 8 - finální ořez vysokokontrastních značek

a_white = int(round(size_sqr[0], 0))

P = [int(round(a_white*0.45, 0)), int(round(a_white*0.45, 0))]
Q = [int(round(size_rect[0] - a_white*0.45, 0)), int(round((size_rect[1] - a_wh

image_value_rect = image_value_rect[P[1]:Q[1], P[0]:Q[0]]

# 9 - vykreslení profilů vysokokontrastních značek

profile_min = np.array(image_value_rect.min(axis=1))
d_profile_min = np.diff(profile_min)
d_profile_min_abs = np.abs(d_profile_min)

peaks, _ = find_peaks(d_profile_min_abs)
borders = peaks[np.argsort(d_profile_min_abs[peaks])[-6:]][::-1]]

borders = sorted(borders)

starts = borders[::2]
ends = borders[1::2]

profiles = []

for y_1, y_2 in zip(starts, ends):
    image_profile = image_value_rect[y_1:y_2][:]
    profile = image_profile.mean(axis=0)
    profiles.append(profile)

# 10 - hledání maxim a minim

profile_1 = profiles[0]

```

```

profile_2 = profiles[1]
profile_3 = profiles[2]

## hledání vrcholů na světlé straně profilu (bílé pruhy v obraze)

peaks_white_1, prop_white_1 = find_peaks(profile_1, width=True, height=True)
peaks_white_2, prop_white_2 = find_peaks(profile_2, width=True, height=True)
peaks_white_3, prop_white_3 = find_peaks(profile_3, width=True, height=True)

## hledání vrcholů na tmavé straně profilu (černé pruhy v obraze)

profile_inverted_1 = -profile_1
profile_inverted_2 = -profile_2
profile_inverted_3 = -profile_3

peaks_black_1, prop_black_1 = find_peaks(profile_inverted_1, width=True)
peaks_black_2, prop_black_2 = find_peaks(profile_inverted_2, width=True)
peaks_black_3, prop_black_3 = find_peaks(profile_inverted_3, width=True)

p_1 = peaks_black_1, peaks_white_1, prop_black_1, prop_white_1
p_2 = peaks_black_2, peaks_white_2, prop_black_2, prop_white_2
p_3 = peaks_black_3, peaks_white_3, prop_black_3, prop_white_3

# 11 - hledání skupin čar

def find_groups(profile, peaks):

    peaks_black, peaks_white, prop_black, prop_white = peaks

    groups = []

    width_black = prop_black["widths"]
    width_white = prop_white["widths"]
    height_white = prop_white["peak_heights"]

    for i, x in enumerate(peaks_black):
        above = [y for y in peaks_white if y > x]
        closest_above = min(above) if above else None

        if i >= len(peaks_black)-4:
            break
        else:
            list_blacks = [x, peaks_black[i+1], peaks_black[i+2], peaks_black[i+3]]
            j = np.where(peaks_white == closest_above)[0][0]
            list_whites = [peaks_white[j], peaks_white[j+1], peaks_white[j+2], peaks_white[j+3]]

            list_width_black = []
            list_width_white = []
            list_height_black = []
            list_height_white = []
            list_location_blacks = []

            for ind, k in enumerate(list_blacks):
                height = profile[k]
                list_height_black.append(height)

```

```

        if ind == 0 or ind == len(list_blacks)-1:
            continue
        else:
            ind_k = np.where(peaks_black == k)[0][0]
            width = width_black[ind_k]
            list_width_black.append(width)

    for l in list_whites:
        ind_l = np.where(peaks_white == l)[0][0]
        width = width_white[ind_l]
        height = height_white[ind_l]
        list_width_white.append(width)
        list_height_white.append(height)

    std_w = round(np.std(list_width_black) + np.std(list_width_white),
    std_h = round(np.std(list_height_white), 2)
    std_h_black = round(np.std(list_height_black), 2)

    for m in range(len(list_blacks)-1):
        dev = list_blacks[m+1] - list_blacks[m]
        list_location_blacks.append(dev)

    std_l_blacks = round(np.std(list_location_blacks), 2)

    if std_w < 0.5 and std_h < 50 and std_h_black < 50 and std_l_blacks
        group = sorted(list_blacks + list_whites)
        groups.append(group)

    return groups

line_groups_1 = find_groups(profile_1, p_1)
line_groups_2 = find_groups(profile_2, p_2)
line_groups_3 = find_groups(profile_3, p_3)

# 12 - vykreslení profilů vysokontrastních linií do grafů

colors = ["green", "blue", "red", "orange", "purple", "yellow"]

a = (6, 3)          # velikost grafů

fig_1, ax_1 = plt.subplots(figsize=a)
ax_1.plot(profile_1)
for i, group in enumerate(line_groups_1):
    ax_1.scatter(group, profile_1[group], color=colors[i])
ax_1.set_title('Nalezené skupiny čar na 1. linii')
ax_1.set_xlabel("Osa x [px]")
ax_1.set_ylabel("Signál")

fig_2, ax_2 = plt.subplots(figsize=a)
ax_2.plot(profile_2)
for i, group in enumerate(line_groups_2):
    ax_2.scatter(group, profile_2[group], color=colors[i])
ax_2.set_title('Nalezené skupiny čar na 2. linii')
ax_2.set_xlabel("Osa x [px]")
ax_2.set_ylabel("Signál")

```

```

fig_3, ax_3 = plt.subplots(figsize=a)
ax_3.plot(profile_3)
for i, group in enumerate(line_groups_3):
    ax_3.scatter(group, profile_3[group], color=colors[i])
ax_3.set_title('Nalezené skupiny čar na 3. linii')
ax_3.set_xlabel("Osa x [px]")
ax_3.set_ylabel("Signál")

# 13 - výpočet kontrastu a MTF

def kontrast(profile, group):
    gray_values = []
    black_values = []
    for i, j in enumerate(group):
        if i % 2 == 0:
            black_values.append(profile[j])
        else:
            gray_values.append(profile[j])

    I_1 = np.mean(gray_values)
    I_2 = np.mean(black_values)

    C = abs(I_1 - I_2)

    return C

kontrast_list = []
mtf_0 = gray_value_rect - black_value_rect
kontrast_list.append(mtf_0)

max_len = max(len(line_groups_1), len(line_groups_2), len(line_groups_3))

for i in range(max_len):
    if i < len(line_groups_1):
        C = kontrast(profile_1, line_groups_1[i])
        kontrast_list.append(C)
    if i < len(line_groups_2):
        C = kontrast(profile_2, line_groups_2[i])
        kontrast_list.append(C)
    if i < len(line_groups_3):
        C = kontrast(profile_3, line_groups_3[i])
        kontrast_list.append(C)

mtf = []
for i in kontrast_list:
    mtf_f = i/mtf_0
    mtf.append(mtf_f)

# 14 - vykreslení MTF do grafu

mtf = np.array(mtf)

freqs_ori = np.array([0.0, 0.5, 0.56, 0.63, 0.71, 0.8, 0.9, 1.0, 1.12, 1.25, 1.

mtf_len = len(mtf)
freqs = freqs_ori[:mtf_len]

```

```

order = np.argsort(freqs)
freqs = freqs[order]
mtf = mtf[order]

fig_mtf, ax_mtf = plt.subplots(figsize=a)
ax_mtf.plot(freqs, mtf, 'o', label='spočtené body MTF')
ax_mtf.set_title('Modulační přenosová funkce - MTF')
ax_mtf.set_xlabel('Prostorová frekvence [lp/mm]')
ax_mtf.set_ylabel('MTF [%]')
ax_mtf.set_ylim(0, 2)
ax_mtf.grid(True)

# 15 - vykreslení fitu a výpočet MTF(30) a MTF(50)

mask = (freqs > 0.3) & (freqs < 2)
freqs_fit = freqs[mask]
mtf_fit = mtf[mask]

def mtf_model(f, a, b):
    return b * np.exp(-(a * f)**2)

popt, _ = curve_fit(mtf_model, freqs_fit, mtf_fit, p0=[0.5, 1])

freqs_smooth = np.linspace(freqs_fit.min() - 0.4, freqs_fit.max() + 1, 1000)
mtf_fit_curve = mtf_model(freqs_smooth, *popt)

ax_mtf.plot(freqs_smooth, mtf_fit_curve, 'm-', label='Gaussovský fit')

mtf_vals = mtf_model(freqs_smooth, *popt)

mtf50 = freqs_smooth[np.argmin(np.abs(mtf_vals - 0.5))]
mtf30 = freqs_smooth[np.argmin(np.abs(mtf_vals - 0.3))]

ax_mtf.plot(mtf50, 0.5, 'g+', markersize=12, markeredgewidth=2, label='MTF(50)')
ax_mtf.plot(mtf30, 0.3, 'r+', markersize=12, markeredgewidth=2, label='MTF(30)')

ax_mtf.legend()

# 15 - úpravy pro export funkce

phantom_parameters = S_phantom, R_phantom
output = preview
plot_list = [fig_mtf, fig_1, fig_2, fig_3]
results = [mtf30, mtf50]

return output, plot_list, phantom_parameters, results

def nizky_kontrast(preview, image_value, list_S_obj, phantom_parameters):

    S_center, R_phantom = phantom_parameters

    R_obj = int(round(0.06*R_phantom, 0))
    R_bg_1 = int(round(0.65*R_phantom, 0))
    R_bg_2 = int(round(0.75*R_phantom, 0))
    R_contrast_prof_1 = int(round(0.76*R_phantom, 0))

```

```

R_contrast_prof_2 = int(round(0.96*R_phantom, 0))

# 1 - vykreslení oblastí pro výpočet SNR a CNR

list_S_bg = []

cv2.circle(preview, S_center, R_bg_1, (0, 255, 0), 1)      # zelená oblast pro S
cv2.circle(preview, S_center, R_bg_2, (0, 255, 0), 1)      # zelená oblast pro S
for i in list_S_obj:
    cv2.circle(preview, i, R_obj, (255, 255, 0), 1)        # žluté oblasti disků s

    dx = i[0] - S_center[0]
    dy = i[1] - S_center[1]

    L = math.sqrt(dx**2 + dy**2)

    d = 0.7*R_phantom

    x = S_center[0] + (d/L)*dx
    y = S_center[1] + (d/L)*dy

    M = (int(round(x, 0)), int(round(y, 0)))

    list_S_bg.append(M)

    cv2.circle(preview, M, R_obj, (0, 255, 0), 1)          # zelené oblasti pro CNR

# 2 - výpočet SNR a CNR

SNRs = []
CNRs = []

for i, j in zip(list_S_obj, list_S_bg):
    Y, X = np.ogrid[:image_value.shape[0], :image_value.shape[1]]
    dist = np.sqrt((X - i[0])**2 + (Y - i[1])**2)
    dist_bg = np.sqrt((X - S_center[0])**2 + (Y - S_center[1])**2)
    dist_bg_cnr = np.sqrt((X - j[0])**2 + (Y - j[1])**2)

    mask_obj = (dist <= R_obj)
    values_obj = image_value[mask_obj]
    mean_value_obj = np.mean(values_obj)

    mask_bg = (dist_bg >= R_bg_1) & (dist_bg <= R_bg_2)
    values_bg = image_value[mask_bg]
    mean_value_bg = np.mean(values_bg)

    std_value_bg = np.std(values_bg)

    differences = values_obj - mean_value_bg
    sum_diff = np.sum(differences)
    SNR = sum_diff/std_value_bg

    mask_bg_cnr = (dist_bg_cnr <= R_obj)
    values_bg_cnr = image_value[mask_bg_cnr]
    mean_value_bg_cnr = np.mean(values_bg_cnr)

```

```

std_value_bg_cnr = np.std(values_bg_cnr)

CNR = (mean_value_obj - mean_value_bg_cnr)/std_value_bg_cnr

SNRs.append(round(SNR, 2))
CNRs.append(round(CNR, 2))

# 3 - vykreslení profilu nízkokontrastních značek

## vykreslení oblasti nízkokontrastních značek

cv2.circle(preview, S_center, R_contrast_prof_1, (0, 0, 255), 1)
cv2.circle(preview, S_center, R_contrast_prof_2, (0, 0, 255), 1)

## výpočet polárních souřadnic pixelů
y, x = np.indices(image_value.shape[:2])
x_c = x - S_center[0]
y_c = y - S_center[1]

r = np.sqrt(x_c**2 + y_c**2)
theta = (np.degrees(np.arctan2(y_c, x_c))) % 360 # úhel v °

## výběr pixelů v mezikruží
mask = (r >= R_contrast_prof_1) & (r <= R_contrast_prof_2)
r_selected = r[mask]
theta_selected = theta[mask]
intensity_selected = image_value[mask]

## vytvoření profilu
num_bins = 1080 # rozlišení úhlů (např. 1°)
bins = np.linspace(0, 360, num_bins + 1)
mean_intensity = np.zeros(num_bins)

for i in range(num_bins):
    in_bin = (theta_selected >= bins[i]) & (theta_selected < bins[i + 1])
    if np.any(in_bin):
        mean_intensity[i] = np.mean(intensity_selected[in_bin])
    else:
        mean_intensity[i] = np.nan

## vytvoření grafu intenzity v závislosti na úhlu
fig_contrast, ax_contrast = plt.subplots(figsize=(6, 3))
ax_contrast.plot(bins[:-1], mean_intensity, color='blue')
ax_contrast.set_xlabel("Úhel [°]")
ax_contrast.set_ylabel("Průměrná intenzita")
ax_contrast.set_title("2D profil intenzity v závislosti na úhlu")
ax_contrast.grid(True)

return preview, SNRs, CNRs, fig_contrast

from tkinter import font

def show_about(parent):
    about = tk.Toplevel(parent)
    about.title("O programu")
    about.resizable(False, False)

```

```

frame = tk.Frame(about)
frame.pack(padx=20, pady=15)

title = tk.Label(
    frame,
    text="TORanalyser",
    font=font.Font(family="Segoe UI", size=14, weight="bold")
)
title.pack(pady=(0, 5))

version = tk.Label(
    frame,
    text="Verze 1.0 - 2026",
    font=font.Font(family="Segoe UI", size=12)
)
version.pack(pady=(0, 10))

author = tk.Label(
    frame,
    text="Autor: Bc. Daniel Pitoňák\n"
        "Přírodovědecká fakulta - Masarykova Univerzita",
    font=font.Font(family="Segoe UI", size=10)
)
author.pack(pady=(0, 10))

contact = tk.Label(
    frame,
    text="Kontakt: 500050@muni.cz, daniel.pitonak@outlook.com\n",
    font=font.Font(family="Segoe UI", size=10)
)
contact.pack(pady=(0, 10))

description = tk.Label(
    frame,
    text=("Program vznikl v rámci diplomové práce\n"
        "„Nalezení optimálního přístupu k hodnocení kvality obrazu\n"
        "v rámci fyzikálních měření“ pod vedením Ing. Jany Tamášové a\n"
        "za významné pomoci Ing. Davida Kozáka.\n\n"),
    justify="left",
    wraplength=420
)
description.pack()

button = tk.Button(
    about,
    text="Zavřít",
    command=about.destroy
)
button.pack(pady=(10, 10))

def show_instructions(parent):
    instructions = tk.Toplevel(parent)
    instructions.title("Návod k použití programu")
    instructions.geometry("600x450")

```

```

frame = tk.Frame(instructions)
frame.pack(fill="both", expand=True, padx=15, pady=15)

body_font = font.Font(family="Segoe UI", size=10)

title = tk.Label(
    frame,
    text="Návod k použití programu TORanalyser",
    font=font.Font(family="Segoe UI", size=13, weight="bold")
)
title.pack(anchor="w", pady=(0, 10))

text = tk.Text(
    frame,
    wrap="word",
    font=font.Font(family="Segoe UI", size=10)
)
text.pack(side="left", fill="both", expand=True)

scrollbar = tk.Scrollbar(frame, command=text.yview)
scrollbar.pack(side="right", fill="y")
text.config(yscrollcommand=scrollbar.set)

text.insert("1.0",
    "1. Načtení DICOM souboru\n"
    "    - Klikněte na tlačítko „Vybrat soubor“ a vyberte radiografický obraz fa"
    "    - Program nejlépe funguje s obrazem pořízeným při napětí 70 kV.\n"
    "    - Pokud nevidíte soubor v očekávaném adresáři, zkuste přepnout druh zob"

    "2. Náhled snímku\n"
    "    - Po načtení se zobrazí základní informace a náhled vybraného snímku.\n"
    "    - V případě načtení nevhodného snímku lze přejít na úvodní obrazovku st"
    "    - Stisknutím zeleného tlačítka „Spustit analýzu“ pokračujte v procesu.\n"

    "3. Průběh analýzy\n"
    "    - Spuštěním analýzy dojde k otevření nového okna, kde je třeba označit"
    "    - Označení bodu lze provést dvojklikem levého tlačítka, odstranění posl"
    "    - Pro zvýšení viditelnosti nízko-kontrastních disků v obraze je třeba z"

    "4. Výsledky\n"
    "    - V oddílu Prostorové rozlišení je k dispozici graf MTF a tři vysoko-ko"
    "    - V oddílu Kontrast se zobrazuje kruhový profil oblasti se 17 nízko-kon"
    "    - V oddílu Výsledky jsou uvedeny hodnoty MTF(30), MTF(50) a poslední SN"
    "    - Náhled snímku zobrazuje kruhový sumační profil v mezikruží označené m"

    "5. Ukončení práce\n"
    "    - Program ukončíte zavřením hlavního okna."
)

text.config(state="disabled")

close_button = tk.Button(
    instructions,
    text="Zavřít",
    command=instructions.destroy

```

```

)
close_button.pack(pady=(10, 0))

def resource_path(relative_path):
    try:
        base_path = sys._MEIPASS
    except Exception:
        base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)

class UrceniStreduNKDisku:
    def __init__(self, parent, ds):
        self.parent = parent
        self.img = ds.pixel_array

        self.body = []
        self.markers = []
        self.max_bodu = 17 # maximálně 17 NK disků

        self.data_min = self.img.min()
        self.data_max = self.img.max()

        # --- ROZLOŽENÍ OKNA (1 řádek, 2 sloupce) ---
        self.fig = Figure(figsize=(14, 8))
        self.ax_img = self.fig.add_subplot(121)
        self.ax_table = self.fig.add_subplot(122)
        self.fig.subplots_adjust(bottom=0.2) # Místo pro slidery dole

        # --- LEVÁ ČÁST - obrázek ---
        self.im = self.ax_img.imshow(self.img, cmap='gray', vmin=self.data_min, vma
        self.ax_img.set_title(f"Výběr středů nízkokontrastních disků\n"
                             f"DVOJKLIK LEVÉHO TLAČÍTKA: výběr bodu\n"
                             f"PRAVÉ TLAČÍTKO: odebrání posledního bodu")
        self.ax_img.axis('off')

        # --- PRAVÁ ČÁST - tabulka ---
        self.ax_table.axis('off')
        self.ax_table.set_title("Seznam vybraných bodů")

        self.canvas = FigureCanvasTkAgg(self.fig, master=self.parent)
        self.canvas.draw()
        self.canvas_widget = self.canvas.get_tk_widget()
        self.canvas_widget.pack(fill=tk.BOTH, expand=True)

        self.toolbar = NavigationToolbar2Tk(self.canvas, self.parent)
        self.toolbar.update()

        # --- Tlačítko HOTOVO ---
        btn_frame = tk.Frame(self.parent)
        btn_frame.pack(fill=tk.X, pady=5)

        btn_hotovo = tk.Button(
            btn_frame,
            text="Hotovo",
            command=self.ukoncit_vyber,
            bg="#4CAF50",

```

```

        fg="white",
        font=("Arial", 10, "bold"),
        padx=20,
        pady=5
    )

    btn_hotovo.pack(padx=10)

# Příprava prázdných dat pro tabulku
    self.table_data = []
    for i in range(self.max_bodu):
        self.table_data.append([str(i+1), "", ""]) # Číslo, X, Y

    col_labels = ["Bod č.", "X [px]", "Y [px]"]

# Vykreslení tabulky
    self.table = self.ax_table.table(
        cellText=self.table_data,
        colLabels=col_labels,
        loc='center',
        cellLoc='center',
        colWidths=[0.2, 0.4, 0.4]
    )
    self.table.auto_set_font_size(False)
    self.table.set_fontsize(10)
    self.table.scale(1, 1.5) # Roztáhneme řádky na výšku

# --- SLIDERY pod obrázkem ---
    ax_min = self.fig.add_axes([0.15, 0.08, 0.5, 0.03]) # [left, bottom, width,
    ax_max = self.fig.add_axes([0.15, 0.12, 0.5, 0.03])

    self.slider_min = Slider(ax_min, 'Černý', self.data_min, self.data_max, val
    self.slider_max = Slider(ax_max, 'Bílý', self.data_min, self.data_max, vali

    self.slider_min.on_changed(self.update_contrast)
    self.slider_max.on_changed(self.update_contrast)

# --- EVENTY ---
    self.cid = self.fig.canvas.mpl_connect('button_press_event', self.on_click)

def update_contrast(self, val):
    vmin = self.slider_min.val
    vmax = self.slider_max.val
    if vmin >= vmax: return
    self.im.set_clim(vmin=vmin, vmax=vmax)
    self.fig.canvas.draw_idle()

def update_table_content(self):
    # Projdeme všech 17 řádků
    for i in range(self.max_bodu):

        cell_x = self.table[(i+1, 1)] # sloupec X
        cell_y = self.table[(i+1, 2)] # sloupec Y

        if i < len(self.body):
            # Máme bod -> zapíšeme souřadnice

```

```

        x, y = self.body[i]
        cell_x.get_text().set_text(f"{x:.1f}")
        cell_y.get_text().set_text(f"{y:.1f}")
    else:
        # Nemáme bod -> vymažeme text
        cell_x.get_text().set_text("")
        cell_y.get_text().set_text("")

self.fig.canvas.draw_idle()

def on_click(self, event):
    if event.inaxes != self.ax_img: return

    # 1. PŘIDÁNÍ BODU (DVOJKLIK)
    if event.button == 1 and event.dblclick:
        if len(self.body) >= self.max_bodu:
            print("Dosažen maximální počet bodů!")
            return

        bod = (event.xdata, event.ydata)
        self.body.append(bod)

        line_object, = self.ax_img.plot(event.xdata, event.ydata, 'rx', markers)
        self.markers.append(line_object)

        self.update_table_content() # Aktualizace tabulky

        # Vykreslení grafu se stane uvnitř update_table_content, ale pro jistotu
        self.fig.canvas.draw()

    # 2. SMAZÁNÍ BODU (PRAVÉ TLAČÍTKO)
    elif event.button == 3:
        if self.body and self.markers:
            self.body.pop()
            odstraneny_marker = self.markers.pop()
            odstraneny_marker.remove()

            self.update_table_content() # Aktualizace tabulky
            self.fig.canvas.draw()

def ziskej_vysledky(self):
    return [(int(x + 0.5), int(y + 0.5)) for x, y in self.body]

def ukoncit_vyber(self):
    self.parent.destroy()

### GUI

class TORanalyser:
    def __init__(self, master):
        self.master = master
        master.title("TORanalyser")

    try:
        master.iconbitmap(resource_path("icon.ico"))
    except:
        # pro případ

```

```

        pass

        screen_w = master.winfo_screenwidth()
        screen_h = master.winfo_screenheight()
        width = int(screen_w * 0.7) # Trochu zvětšíme pro grafy
        height = int(screen_h * 0.7)
        master.geometry(f"{width}x{height}")
        master.minsize(800, 600)

        self.container = tk.Frame(master)
        self.container.pack(fill="both", expand=True)

        self.zobraz_uvodni_obrazovku()

    def _on_mousewheel(self, event):
        # funkční pro Windows / MacOS
        if event.delta:
            self.results_canvas.yview_scroll(int(-1 * (event.delta / 120)), "units")
        # funkční pro Linux
        elif event.num == 4:
            self.results_canvas.yview_scroll(-1, "units")
        elif event.num == 5:
            self.results_canvas.yview_scroll(1, "units")

    def clear_container(self):
        for widget in self.container.winfo_children():
            widget.destroy()

    def zobraz_uvodni_obrazovku(self):
        self.clear_container()
        self.main_title = tk.Label(self.container, text="Výběr DICOM souboru", font=
        self.main_title.pack(pady=40)
        btn = tk.Button(self.container, text="Vybrat soubor", command=self.vyber_so
        btn.pack()

    def vyber_soubor(self):
        filepath = filedialog.askopenfilename(filetypes=[("DICOM Files", "*.dcm"),
        if not filepath: return
        try:
            ds = pydicom.dcmread(filepath)
            self.zobraz_info_obrazovku(filepath, ds)
        except Exception as e:
            messagebox.showerror("Chyba", f"Nelze načíst soubor:\n{e}")

    def zobraz_info_obrazovku(self, filepath, ds):
        self.clear_container()
        self.filename = os.path.basename(filepath)

        self.main_title = tk.Label(self.container, text=f"Soubor: {self.filename}",
        self.main_title.pack(pady=10)

        content = tk.Frame(self.container)
        content.pack(fill="both", expand=True, padx=10)

        # KONFIGURACE sloupců: 0 - Info + Náhled, 1 - Grafy
        content.columnconfigure(0, weight=0, minsize=200)

```

```

content.columnconfigure(1, weight=1)
content.rowconfigure(1, weight=0)
content.rowconfigure(2, weight=1)

# LEVÝ BLOK

    ## informace o souboru
self.info_frame = tk.LabelFrame(content, text=" Informace o snímku ", padx=
self.info_frame.grid(row=1, column=0, sticky="nsew", padx=5)

table = tk.Frame(self.info_frame)
table.pack(anchor="w")

def add_row(r, lbl, val):
    tk.Label(table, text=lbl+":", font=("Arial", 9, "bold")).grid(row=r, co
    tk.Label(table, text=val, wraplength=350, justify="left").grid(row=r, c

tags = [
    ("Cesta k souboru", filepath),
    ("Pacient", getattr(ds, "PatientName", "N/A")),
    ("ID", getattr(ds, "PatientID", "N/A")),
    ("Datum", getattr(ds, "StudyDate", "N/A")),
    ("Modalita", getattr(ds, "Modality", "N/A")),
    ("kVp", getattr(ds, "KVP", "N/A")),
    ("mAs", getattr(ds, "Exposure", "N/A")),
    ("SID", getattr(ds, "DistanceSourceToDetector", 'N/A')),
]
for i, (l, v) in enumerate(tags):
    if l == "Datum":
        v = f"{v[6:8]}. {v[4:6]}. {v[0:4]}"
    add_row(i, l, v)

    ## náhled
self.preview_frame = tk.LabelFrame(content, text=" Náhled snímku ", padx=10
self.preview_frame.grid(row=2, column=0, sticky="nsew", padx=5)
self.preview_img_label = tk.Label(self.preview_frame)
self.preview_img_label.pack()

try:
    img = ds.pixel_array.astype(float)
    img = (img - img.min()) / (img.max() - img.min()) * 255
    pil_img = Image.fromarray(img.astype(np.uint8)).convert("L")
    pil_img.thumbnail((400, 400))
    self.preview_img = ImageTk.PhotoImage(pil_img)
    self.preview_img_label.configure(image=self.preview_img)
    self.preview_img_label.image = self.preview_img
except Exception as e:
    self.preview_img_label.configure(text=f"Náhled nelze zobrazit\n{e}")

# PRAVÝ BLOK
self.graph_container = tk.LabelFrame(content, text=" Analýza ")
self.graph_container.grid(row=1, column=1, rowspan=2, sticky="nsew", padx=5
self.graph_container.columnconfigure(0, weight=1)

    ## prostorové rozlišení
right_inner = tk.Frame(self.graph_container)

```

```

right_inner.pack(fill="both", expand=True, padx=5, pady=5)

right_inner.columnconfigure(0, weight=2)
right_inner.columnconfigure(1, weight=1)
right_inner.rowconfigure(0, weight=3)
right_inner.rowconfigure(1, weight=1)

self.resolution_frame = tk.LabelFrame(
    right_inner, text=" Rozlišení při vysokém kontrastu ", padx=5, pady=5
)
self.resolution_frame.grid(row=0, column=0, rowspan=2, sticky="nsew", padx=

self.results_canvas = tk.Canvas(self.resolution_frame, highlightthickness=0
self.results_scrollbar = tk.Scrollbar(self.resolution_frame, orient="vertic
self.results_frame = tk.Frame(self.results_canvas)

self.results_canvas.create_window((0, 0), window=self.results_frame, anchor
self.results_canvas.configure(yscrollcommand=self.results_scrollbar.set)

self.results_canvas.pack(side="left", fill="both", expand=True)
self.results_scrollbar.pack(side="right", fill="y")

    ## nastavení scrollování
self.results_frame.bind("<Configure>", lambda e: self.results_canvas.config
self.results_canvas.bind("<Enter>", lambda e: self.results_canvas.bind_all(
self.results_canvas.bind("<Leave>", lambda e: self.results_canvas.unbind_al

    ## kontrast
self.contrast_frame = tk.LabelFrame(
    right_inner, text=" Rozlišení při nízkém kontrastu ", padx=5, pady=5
)
self.contrast_frame.grid(row=0, column=1, sticky="nsew")

self.contrast_frame.rowconfigure(0, weight=1)
self.contrast_frame.rowconfigure(1, weight=0)
self.contrast_frame.rowconfigure(2, weight=0)
self.contrast_frame.columnconfigure(0, weight=1)

self.contrast_graph_container = tk.LabelFrame(self.contrast_frame)
self.contrast_graph_container.grid(row=0, column=0, sticky="nsew", pady=(0,

self.SNR_table = tk.LabelFrame(self.contrast_frame)
self.SNR_table.grid(row=1, column=0, sticky="ew")
self.SNR_table.columnconfigure(0, weight=0)
self.SNR_table.columnconfigure(1, weight=0)
self.SNR_table.columnconfigure(2, weight=1)
self.SNR_table.columnconfigure(3, weight=1)

tk.Label(self.SNR_table, text="Disk č.", font=("Arial", 9, "bold")).grid(ro
tk.Label(self.SNR_table, text="Tabulkový kontrast [%]", font=("Arial", 9, "
tk.Label(self.SNR_table, text="SNR", font=("Arial", 9, "bold")).grid(row=0,
tk.Label(self.SNR_table, text="CNR", font=("Arial", 9, "bold")).grid(row=0,

number_list = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12
contrast_list = ["7,60", "6,60", "5,50", "4,50", "3,90", "3,30", "2,70", "2
for i in range(1, 18):

```

```

        tk.Label(self.SNR_table, text=number_list[i-1]).grid(row=i, column=0, s
        tk.Label(self.SNR_table, text=contrast_list[i-1]).grid(row=i, column=1,

self.detail_table = tk.LabelFrame(right_inner, text=" Výsledky ", padx=5, p
self.detail_table.grid(row=1, column=1, sticky="nsew", pady=(5, 0))
self.detail_table.columnconfigure(0, weight=1)
self.detail_table.columnconfigure(1, weight=1)
self.detail_table.columnconfigure(2, weight=0, minsize=20) # mezera
self.detail_table.columnconfigure(3, weight=1)
self.detail_table.columnconfigure(4, weight=1)
self.detail_table.columnconfigure(5, weight=1)

    ## výsledky
    tk.Label(self.detail_table, text="Rozlišení při vysokém kontrastu", font=("
    tk.Label(self.detail_table, text=" ", font=("Arial", 9, "bold")).grid(row=
    tk.Label(self.detail_table, text="Rozlišení při nízkém kontrastu", font=("A

# TLAČÍTKA
btn_bar = tk.Frame(self.container)
btn_bar.pack(side="bottom", fill="x", pady=10)
btn_zpet = tk.Button(btn_bar, text="Výběr jiného souboru", command=self.zob
btn_zpet.pack(side="left", padx=20)
self.btn_analyze = tk.Button(btn_bar, text="Spustit analýzu", bg="#4CAF50",
                             command=lambda: self.spust_analyzu(filepath)
)
self.btn_analyze.pack(side="right", padx=20)

def spust_analyzu(self, filepath):

    try:
        self.main_title.config(
            text=f"Analýza DICOM souboru: {self.filename}"
        )

        image = pydicom.dcmread(filepath)
        image_value = image.pixel_array.astype(np.float32)

        output, list_S_obj, phantom_parameters, results = vysoky_kontrast(image

        win = tk.Toplevel(self.master)
        win.title("Výběr středů nízkokontrastních disků")

        Centers_App = UrceniStreduNKDisku(win, image)

        win.transient(self.master)
        win.grab_set()
        self.master.wait_window(win)

        seznam_stredu = Centers_App.ziskej_vysledky()

        output, SNRs, CNRs, fig_angle = nizky_kontrast(output, image_value, sez

        if isinstance(output, np.ndarray):
            img = Image.fromarray(output)
            img = img.resize((400, 400))
            img_tk = ImageTk.PhotoImage(img)

```

```

self.preview_img_label.configure(image=img_tk)
self.preview_img_label.image = img_tk

for plot in list_S_obj:
    if isinstance(plot, Figure):
        plot.tight_layout()

        canvas = FigureCanvasTkAgg(plot, master=self.results_frame)
        canvas.draw()
        canvas_widget = canvas.get_tk_widget()
        canvas_widget.pack(pady=5)

self.results_frame.update_idletasks()
self.results_canvas.configure(
    scrollregion=self.results_canvas.bbox("all"))

if isinstance(fig_angle, Figure):
    fig_angle.tight_layout()

    canvas_angle = FigureCanvasTkAgg(
        fig_angle,
        master=self.contrast_graph_container
    )
    canvas_angle.draw()

    canvas_widget = canvas_angle.get_tk_widget()
    canvas_widget.pack(fill="both", expand=True)

for i in range(0, len(SNRs)):
    tk.Label(self.SNR_table, text=str(int(round(SNRs[i], 0)))).grid(row=i, column=0, sticky="n")
    tk.Label(self.SNR_table, text=CNRs[i]).grid(row=i+1, column=3, sticky="n")

contrast_list = ["7,60", "6,60", "5,50", "4,50", "3,90", "3,30", "2,70"]
for i, j in enumerate(SNRs):
    if j >= 5:
        SNR_lim = SNRs[i]
        SNR_num = i+1
        contrast = contrast_list[i]
        continue
    else:
        SNR_lim = SNRs[i-1]
        SNR_num = i
        contrast = contrast_list[i-1]
        break

mtf30 = str(round(results[0], 2)) + " lp/mm"
mtf50 = str(round(results[1], 2)) + " lp/mm"

tk.Label(self.detail_table, text="MTF(30) =").grid(row=1, column=0, sticky="n")
tk.Label(self.detail_table, text=mtf30).grid(row=1, column=1, sticky="nw")
tk.Label(self.detail_table, text="MTF(50) =").grid(row=2, column=0, sticky="n")
tk.Label(self.detail_table, text=mtf50).grid(row=2, column=1, sticky="nw")

tk.Label(self.detail_table, text="Disk č. ").grid(row=1, column=4, sticky="n")
tk.Label(self.detail_table, text=SNR_num).grid(row=1, column=5, sticky="nw")

```

```

        tk.Label(self.detail_table, text="SNR = ").grid(row=2, column=4, sticky=
        tk.Label(self.detail_table, text=SNR_lim).grid(row=2, column=5, sticky=
        tk.Label(self.detail_table, text="Kontrast: ").grid(row=3, column=4, st
        tk.Label(self.detail_table, text=contrast).grid(row=3, column=5, sticky

    if hasattr(self, "btn_analyze"):
        self.btn_analyze.pack_forget()

    except Exception as e:
        messagebox.showerror("Chyba", str(e))

if __name__ == "__main__":
    root = tk.Tk()

    root.iconbitmap(resource_path("icon.ico"))
    root.title("TORanalyser")

    app = TORanalyser(root)

    menubar = tk.Menu(root)

    help_menu = tk.Menu(menubar, tearoff=0)
    help_menu.add_command(
        label="Návod",
        command=lambda: show_instructions(root)
    )
    help_menu.add_command(
        label="O programu",
        command=lambda: show_about(root)
    )

    menubar.add_cascade(label="Nápověda", menu=help_menu)
    root.config(menu=menubar)

    root.mainloop()

```

In []: