Akademia Górniczo-Hutnicza

Wydział Informatyki

Projekt nr 2
Z przedmiotu Informatyka Medyczna

"Segmentacja medyczna obrazu"

Autor: *Jacek Tyszkiewicz*
Kierunek studiów: *Informatyka*

Kraków, 2025

Spis treści

1. Segmentacja obrazów medycznych

1.1 Cel projektu:

Celem projektu było zapoznanie się z przetwarzaniem i analizą obrazów medycznych w formacie DICOM. Należało zaimplementować interaktywne narzędzia do wizualizacji oraz analizy danych obrazowych. Podczas realizacji tego zadania wykonano:

1.2 Zadanie 1

Zadanie 1a (1 pkt)

- Odczytaj z pliku DICOM wartości *WindowWidth* oraz *WindowCenter*
- Wykonaj transformację danych obrazowych zgodnie z wartościami okna
- Wyświetl obraz po transformacji

Zadanie 1b (1 pkt)

- Dodaj funkcjonalność dynamicznej zmiany parametrów okna przy ruchu myszką

Zadanie 1c (1 pkt)

- Dodaj funkcjonalność pomiarową:
  - Rysowanie odcinka na obrazku
  - Wyznaczenie długości odcinka
  - Przeliczenie na jednostki odczytane z metadanych (*PixelSpacing*)
  - Wynik pomiaru wypisać na konsolę

```
# AGH UST Medical Informatics 03.2021
# Lab 2 : DICOM
import PIL
print(PIL.__version__)
import pydicom
from tkinter import *
from PIL import Image, ImageTk
from PIL.Image import Resampling

class MainWindow():

    ds = pydicom.dcmread("head.dcm")
    data = ds.pixel_array

    def __init__(self, main):
        # print patient name
        print(self.ds.PatientName)

        #todo: from ds get windowWidth and windowCenter

        self.winCenter = int(self.ds.WindowCenter) if type(self.
            ds.WindowCenter) != pydicom.multival.MultiValue else
            int(self.ds.WindowCenter[0])
        self.winWidth = int(self.ds.WindowWidth) if type(self.ds.
            WindowWidth) != pydicom.multival.MultiValue else int(
            self.ds.WindowWidth[0])

        # prepare canvas
        self.canvas = Canvas(main, width=512, height=512)
        self.canvas.grid(row=0, column=0)
        self.canvas.bind("<Button-1>", self.init_window)
        self.canvas.bind("<B1-Motion>", self.update_window)
        self.canvas.bind("<Button-3>", self.init_measurement)
        self.canvas.bind("<B3-Motion>", self.update_measurement)
        self.canvas.bind("<ButtonRelease-3>", self.
            finish_measurement)

        self.line = None
        self.measure_start = None
```

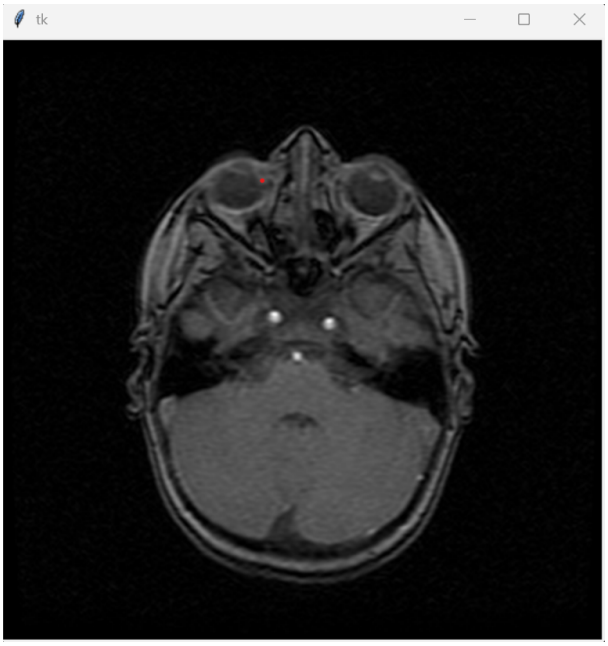
```
        # load image
        # todo: apply transform
        #self.array = self.transform_data(self.data, self.
            winWidth, self.winCenter)
        self.array = self.transform_data(self.data, self.winWidth
            , self.winCenter)
        self.image = Image.fromarray(self.array)
        self.image = self.image.resize((512, 512), Resampling.
            LANCZOS)
        self.img = ImageTk.PhotoImage(image=self.image, master=
            root)
        self.image_on_canvas = self.canvas.create_image(0, 0,
            anchor=NW, image=self.img)

    def transform_data(self, data, window_width, window_center):
        # todo: transform data (apply window width and center)

        img = data.astype('int16')

        lower_bound = window_center - (window_width / 2)
        upper_bound = window_center + (window_width / 2)

        # Przycinanie wartości
        img[img < lower_bound] = lower_bound
        img[img > upper_bound] = upper_bound

        # Normalizacja do zakresu 0-255
        img = ((img - lower_bound) / window_width) * 255.0
        img = img.astype('uint8')
        return img

    def init_window(self, event):
        # todo: save mouse position
        self.start_x = event.x
        self.start_y = event.y
        #print("x: " + str(event.x) + " y: " + str(event.y))
        print("Punkt początkowy     x:", self.start_x, "y:", self
            .start_y)

    def update_window(self, event):
```

```
        # todo: modify window width and center
        dx = event.x - self.start_x
        dy = event.y - self.start_y

        sensitivity_width = 0.5
        sensitivity_center = 0.5

        self.winWidth += int(dx / sensitivity_width)
        self.winCenter += int(dy / sensitivity_center)

        # Ograniczenia
        self.winWidth = max(1, min(self.winWidth, 5000))  # np.
            max 4000
        self.winCenter = max(0, min(self.winCenter, self.data.max
            ()))  # np. w zakresie obrazu

        self.start_x = event.x
        self.start_y = event.y

        print(f"dx: {dx}, dy: {dy} => WW: {self.winWidth}, WC: {
            self.winCenter}")

        self.array2 = self.transform_data(self.data, self.
            winWidth, self.winCenter)
        self.image2 = Image.fromarray(self.array2)
        self.image2 = self.image2.resize((512, 512), Resampling.
            LANCZOS)
        self.img2 = ImageTk.PhotoImage(image=self.image2, master=
            root)
        self.canvas.itemconfig(self.image_on_canvas, image=self.
            img2)

    def init_measurement(self, event):
        # todo: save mouse position
        # todo: create line
        # hint: self.canvas.create_line(...)
        #print("x: " + str(event.x) + " y: " + str(event.y))
        self.measure_start = (event.x, event.y)
        self.line = self.canvas.create_line(event.x, event.y,
            event.x, event.y, fill="red", width=2)
        print(f"Punkt początkowy pomiaru: {self.measure_start}")
```

```
    def update_measurement(self, event):
        # todo: update line
        # hint: self.canvas.coords(...)
        #print("x: " + str(event.x) + " y: " + str(event.y))
        if self.line and self.measure_start:
            self.canvas.coords(self.line, self.measure_start[0],
               self.measure_start[1], event.x, event.y)

    def finish_measurement(self, event):
        # todo: print measured length in mm
        #print("x: " + str(event.x) + " y: " + str(event.y))
        if self.line and self.measure_start:
            x0, y0 = self.measure_start
            x1, y1 = event.x, event.y

            dx = x1 - x0
            dy = y1 - y0
            pixel_distance = (dx**2 + dy**2) ** 0.5

            # Odczyt PixelSpacing z DICOM
            spacing = self.ds.PixelSpacing  # [mm/pixel]
            spacing_x = float(spacing[0])
            spacing_y = float(spacing[1])

            # Przeliczenie długości na mm
            real_distance = ((dx * spacing_x)**2 + (dy *
               spacing_y)**2) ** 0.5

            print(f"Długość odcinka: {pixel_distance:.2f} px")
            print(f"Długość rzeczywista: {real_distance:.2f} mm")

            # Reset zmiennych
            self.line = None
            self.measure_start = None


root = Tk()
MainWindow(root)
root.mainloop()
```

Opis kodu - zadanie 1a

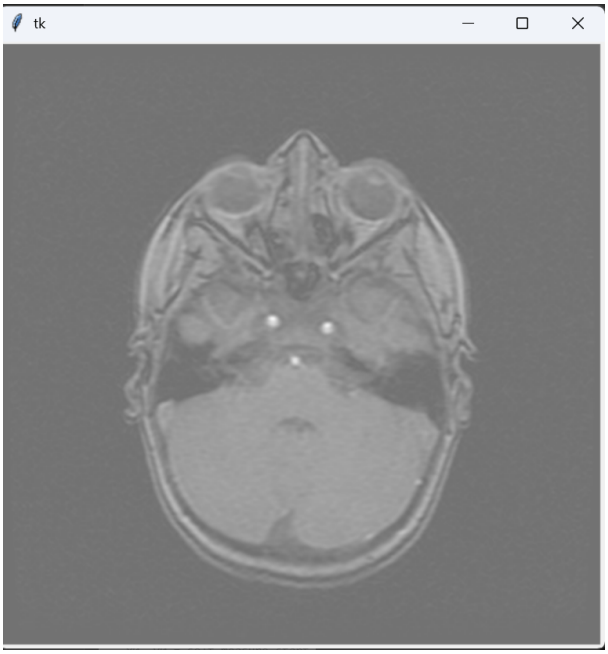
W konstruktorze odczytujemy wartości WindowWidth i WindowCenter z pliku DICOM, następnie zapisujemy self.winCenter, self.winWidth. Następnie transformacja danych wykonywana jest w funkcji transform_data tzn. transformujemy dane do int16, wyznaczamy zakres okna. Następnie w celu zwiększenia kontrastu, niektóre pixele są przycinane tzn. muszą mieścić się w zakresie (lower_bound, upper_bound), a na koniec są normalizowane do zakresu 0-255 i konwertowane do uint8. Obraz jest następnie wyświetlany za pomocą funkcji update_window.



Rys. 1.1. wynik zadania 1a

Opis kodu - zadanie 1b

Funkcjionalność dynamicznego zmiany okna przy pomocy myszki, została zaimplementowana w funkcji update_window. Kontrast można zmieniać na podstawie ruchów w poziomie, natomiast jasność na podstawie ruchów w pionie. W kodzie obliczamy zmianę położenia myszki dx, dy, a następnie aktualizujemy Window_Width (kontrast) i Window_Center (jasność), uwzględniamy czułość, którą ustawiłem na 0.5, aby zmiany w jasności i kontraście zachodziły wolniej.



Rys. 1.2. wynik zadania 1b

Opis kodu - zadanie 1c

```
def init_measurement(self, event):
    self.measure_start = (event.x, event.y)
    self.line = self.canvas.create_line(event.x, event.y, event.x
       , event.y, fill="red", width=2)
```

Opis: Tworzy nową linię od punktu początkowego po kliknięciu prawego przycisku myszy.

```
def update_measurement(self, event):
    if self.line and self.measure_start:
        self.canvas.coords(self.line, self.measure_start[0], self
           .measure_start[1], event.x, event.y)
```

Opis: Dynamicznie aktualizuje współrzędne końca linii podczas przeciągania myszą.

```
def finish_measurement(self, event):
    x0, y0 = self.measure_start
    x1, y1 = event.x, event.y

    dx = x1 - x0
    dy = y1 - y0
    pixel_distance = (dx**2 + dy**2) ** 0.5
```

Opis: Liczy długość odcinka w pikselach za pomocą twierdzenia Pitagorasa.

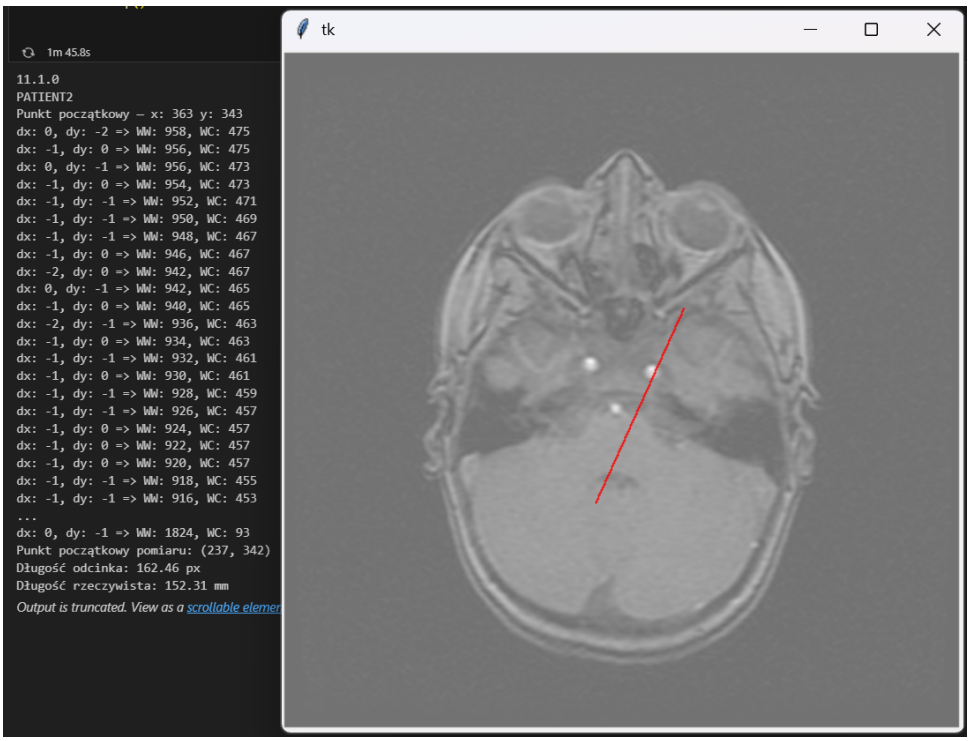
W tej samej metodzie:

```
spacing = self.ds.PixelSpacing
spacing_x = float(spacing[0])
spacing_y = float(spacing[1])

real_distance = ((dx * spacing_x)**2 + (dy * spacing_y)**2)
   ** 0.5
```

Opis:

– `PixelSpacing` pochodzi z metadanych DICOM i określa fizyczny rozmiar piksela w mm.

– Długość odcinka w pikselach jest przeliczana na milimetry z uwzględnieniem różnic w rozdzielczości w osiach X i Y.



Rys. 1.3. wynik zad 1c

Zadanie 2 (2 punkty)

- Zaimplementuj metodę segmentacji przez rozrastanie obszarów
- Punkt startowy wybierany myszką
- Efektem ma być obrys wydzielonego obszaru
- Dane: plik `abdomen.png`
- Plik z kodem do zadania (`lab2_segmentation.py`)

```
# AGH UST Medical Informatics 03.2021
# Lab 2 : Segmentation

import cv2 as cv
import numpy as np

# Wczytanie obrazu
im = cv.imread('abdomen.png')
im = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
w, h = im.shape[1], im.shape[0]
mask = np.zeros((im.shape), np.uint8)

def region_growing(image, seed_point, threshold=10):
    h, w = image.shape
    visited = np.zeros((h, w), dtype=np.bool_)
    region = np.zeros((h, w), dtype=np.uint8)

    x0, y0 = seed_point
    seed_value = image[y0, x0]

    stack = [(x0, y0)]

    while stack:
        x, y = stack.pop()
        if visited[y, x]:
            continue

        visited[y, x] = True
        intensity = image[y, x]
        if abs(int(intensity) - int(seed_value)) <= threshold:
            region[y, x] = 255
```

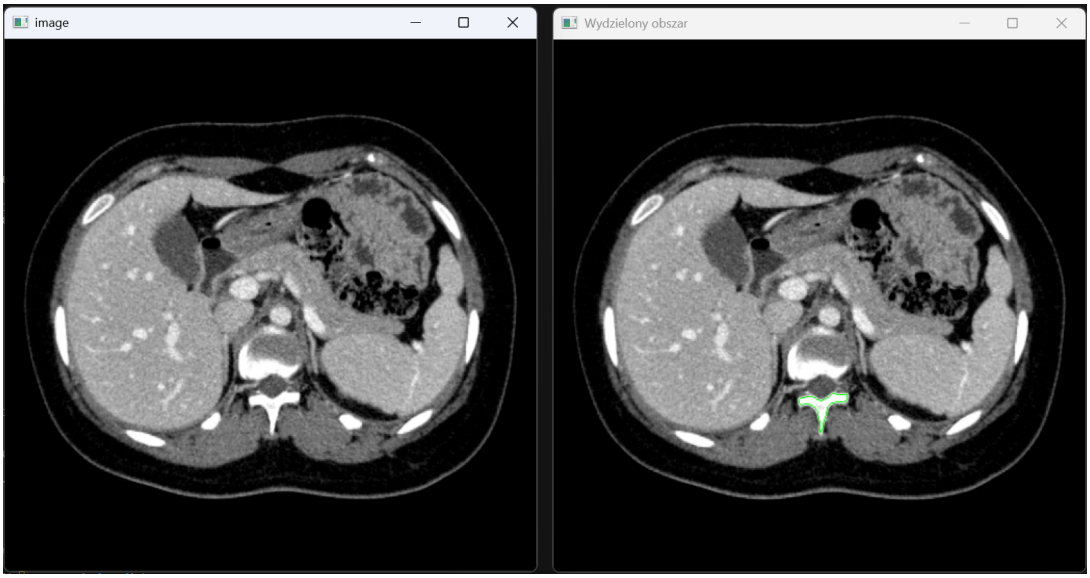
```
            for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
                nx, ny = x + dx, y + dy
                if 0 <= nx < w and 0 <= ny < h and not visited[ny
                    , nx]:
                    stack.append((nx, ny))

    return region

def mouse_callback(event, x, y, flags, params):
    if event == cv.EVENT_LBUTTONDOWN:
        print(f"Kliknieto w punkt: ({x}, {y}) wartość: {im[y, x]}
            ")
        region = region_growing(im, (x, y), threshold=10)


        contours, _ = cv.findContours(region, cv.RETR_EXTERNAL,
            cv.CHAIN_APPROX_SIMPLE)
        output = cv.cvtColor(im, cv.COLOR_GRAY2BGR)
        cv.drawContours(output, contours, -1, (0, 255, 0), 1)

        cv.imshow("Wydzielony obszar", output)

cv.imshow('image', im)
cv.setMouseCallback('image', mouse_callback)
cv.waitKey()
cv.destroyAllWindows()
```

1.2.1 Opis kodu - zadanie 2

Algorytm segmentacji działa w następujący sposób:
W pierwszej kolejności program wczytuje plik abdomen.png i konwertuje go do skali szarości (ponieważ segmentacja jest oparta na jasności pikseli). Następnie, tworzona jest pusta maska (do ewentualnego przechowywania segmentacji) oraz zapisywany jest rozmiar obrazu (wysokość, szerokość). Kolejno w kodzie jest zdefiniowana funkcja region_growing, która jest odpowiedzialna za segmentację. Przyjmuje ona obraz, punkt startowy oraz próg tolerancji.Następnie znajduje wszystkie piksele podobne intensywnością do punktu startowego (z dokładnością threshold i oznacza je jako część segmentowanego obszaru. Ustawiana jest funkcja mouse_callback, która reaguje na kliknięcie lewym przyciskiem

myszy. Gdy użytkownik kliknie: pobierany jest punkt kliknięcia. Wywoływana jest segmentacja z tego punktu. Na wynikowej masce (czarno-biały obszar) szukane są kontury. Ostatecznie na oryginalnym obrazie rysowany jest zielony obrys segmentowanego obszaru.

1.2.2 Wynik działania programu z zadania 2



Rys. 1.4. wynik zadania 2

Widzimy obrys, segmentacja się powiodła.