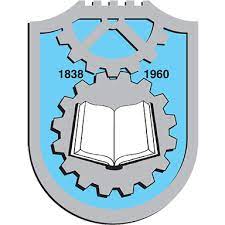
Univerzitet u Kragujevcu

Fakultet inženjerskih nauka

****

SEMINARSKI RAD IZ PREDMETA

SOFTVERSKOG INZENJERINGA

- PREPOZNAVANJE POVRŠINE NA MEDICINSKIM SLIKAMA –

Student: Predmetni nastavnik:

Ilija Todorović 71/2020 Prof. Nenad Filipović

Kragujevac, 2023

SADRŽAJ

[1. POSTAVKA ZADATKA 2](#_Toc142757720)

[2. OPIS DELOVA PROGRAMA SA IZVORNIM KODOM 2](#_Toc142757721)

[2.1 Korišćene python biblioteke 2](#_Toc142757722)

[2.2 Kreiranje prozora 3](#_Toc142757723)

[2.3 Klasa GUI 3](#_Toc142757724)

[3. UML DIJAGRAMI 7](#_Toc142757725)

[3.1 Use case dijagram 7](#_Toc142757726)

[3.2 Dijagram sekvenci 8](#_Toc142757727)

[3.3 Dijagram aktivnosti 9](#_Toc142757728)

[3.4 Dijagram stanja 9](#_Toc142757729)

[3.5 Dijagram klasa 10](#_Toc142757730)

[4. ZAKLJUČAK 11](#_Toc142757731)

[5. LITERATURA 12](#_Toc142757732)

# POSTAVKA ZADATKA

Potrebno je napraviti aplikaciju koja obavlja prepoznavanje površine sa 2D snimaka DICOM formata medicinskih slika. Potrebno je učitati date DICOM formate I na osnovu optimalnog threshholda izvršiti prepoznavanje kostiju. Kao rezultat dobijaju se nove slike sa segmentisanom oblašću od interesa. Takođe je potrebno vratiti I površinu prepoznatih kostiju.

# OPIS DELOVA PROGRAMA SA IZVORNIM KODOM

# Korišćene python biblioteke

Na samom početku programa potrebno je definisati biblioteke funkcija koje ćemo koristiti.

**import tkinter as tk**

**from tkinter import filedialog**

Koristili smo bioblioteku tkinter za kreiranje GUI okruženja, i iz te biblioteke smo dodatno importovali funkciju filedialog koju smo kasnije koristili za učitavanje DICOM snimka sa računara, tako što bismo sačuvali put do slike kao promenljivu. Više o tome u nastavku.

**from PIL import Image ,ImageTk**

Iz pillow biblioteke smo importovali Image I ImageTk koje ćemo kasnije koristiti za rad sa slikom kada budemo trebali da je prikažemo.

**import cv2**

Zatim smo učitali cv2 koja je u suštini velika open source biblioteka koja će nam kasnije služiti za threshold, izdvajanje kontura I površinu.

**import pydicom.data**

**import pydicom**

**from pydicom.pixel\_data\_handlers import apply\_voi\_lut**

Posle toga učitavamo pydicom koja je biblioteka za rad sa DICOM snimcima.

**import numpy as np**

I numpy koja je biblioteka koja se koristi za rad sa nizovima. To će nam trebati kasnije za učitavanje i prikazivanje DICOM snimka u aplikaciji, pošto je cilj da ga pretvorimo u niz pa onda iz niza u sliku.

**from matplotlib import cm**

Biblioteka matplotlib služi za kreiranje statičkih, animiranih i interaktivnih vizualizacija u pajtonu.

# Kreiranje prozora

Nakon što smo uvezli biblioteke koje će nam trebati potreno je da napravimo prozor (formu/dijalog) naše aplikacije. Sledeći kod nam služi da kreiramo prozor i da definišemo njegove dimenzije tj. geometriju i samo ime(title) prozora.

**root=tk.Tk()**

**root.geometry("800x600")**

**root.title("MedicinskeSlike")**

Na kraju svakog python koda u kome kreiramo neku formu/dijalog potrebno je staviti sledeći kod kako bi se forma pokrenula.

**root.mainloop()**

# Klasa GUI

Klasu smo nazvali GUI I u njoj smo upakovali tri metode jednu za kreiranje GUI okruženja, jednu za učitavanje DICOM slike I jednu za odredjivanje kontrure I površine.

Prva metoda je nazvana \_init\_ iz razloga da bi se pozivala odmah čim se program otvori, u njoj smo definisali promenljivu filepath koju ćemo kasnije koristiti da u njoj čuvamo put do DICOM slike, definisali smo frejm na levoj strani u koji ćemo spakovati dva dugmeta. Jedno koje će pritiskom korisnika pozvati I izvršiti metodu za učitavanje DICOM slike I jedno koje će pritiskom korisnika prikazati konture kostiju I izracunati njihovu povrsinu. Takođe smo definisali I jedan canvas u koji ćemo smeštati slike. Sam izgled prozora je uradjen po uzoru na: youtube/Code First with Hala/Tkinter Image Editing Tool tutorial – Python GUI project for begginers [1]

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

**self.file\_path = ""**

**self.left\_frame= tk.Frame(root, width=200, height=600, bg="white")**

**self.left\_frame.pack(side="left", fill="y")**

**self.canvas = tk.Canvas(root, width=700, height=600)**

**self.canvas.pack(padx= 30, pady=40)**

**button = tk.Button(self.left\_frame, text="Import DICOM", font=('Arial', 12) ,command = self.import\_image)**

**button.pack(padx=20, pady=20)**

**button = tk.Button(self.left\_frame, text="IZRACUNAJ", font=('Arial', 12) , command = self.racunaj\_konture)**

**button.pack(padx=20, pady=40)**

**self.text=tk.Label(self.left\_frame, text=f"")**

**self.text.pack(padx=20, pady=20)**

Ovde smo takođe u zagradama definisali veličine frejma, canvasa,label-a i dugmića, njihovu poziciju u prozoru, text koji će biti ispisan na dugmićima, kao i font i veličinu slova.

Nakon što smo kreirali naše okruženje u prozoru potrebno je da osposobimo dugmiće. Potrebno je deifnisati metodu(import\_image) za učitavanje DICOM slike u program.

**def import\_image(self):**

Prvo i osnovno što smo želeli jeste da kada pritisnemo na dugme da nam se otvori prozor u kojem ćemo mi moći da selektujemo sliku koju želimo da importujemo u aplikaciju. Zbog toga smo na početku importovali filedialog iz tkintera. Njega ćemo upotrebiti da sačuvamo put do slike u promenljivu.

**self.file\_path = filedialog.askopenfilename()**

Nakon učitavanja slike moramo proveriti da li je korisnik učitao sliku DICOM formata, ako nije moramo ga obavestiti da to uradi. Ako je slika jpg ili nekog drugog formata u labelu, koji smo stavili sa strane, ispisaće se upozorenje, ako je format dobar nastaviće se izvršavanje koda.

**if not self.file\_path.endswith('.dcm'):**

**self.text.config(text="slika mora da bude dcm formata")**

**return**

Pošto preko pillow biblioteke rad sa slikama ne omogućava rad sa DICOM slikama ideja je bila da se učitan DICOM prebaci u niz pa iz niza ponovo u sliku i da se tako smesti u canvas[2]. To smo uradili uz pomoć sledećeg koda.

**dicom\_image= pydicom.read\_file(self.file\_path)**

**slika\_u\_niz = apply\_voi\_lut(dicom\_image.pixel\_array, dicom\_image)**

**slika\_u\_niz = slika\_u\_niz / slika\_u\_niz.max()**

**self.save\_for\_cont = slika\_u\_niz**

**image = Image.fromarray(np.uint8(cm.gist\_earth(slika\_u\_niz)\*255))**

Ovde smo učitali i pročitali fajl na adresi. Pretvorili ga u niz, izvršili normalizaciju zato što nam je potrebno da nam pixeli budu od 0 do 1, za kasniji rad sa konturama tj. thresholdom. Sačuvali smo takav niz u posebnu promenljivu koju ćemo koristiti kasnije u sledećoj metodi i onda na kraju prebacili niz u sliku. Kako bismo je učitali u canvas.

**self.canvas.config(width=image.width, height=image.height)**

**image= ImageTk.PhotoImage(image)**

**self.canvas.image= image**

**self.canvas.create\_image(0,0, image=image, anchor="nw")**

Dobijenu sliku učitavamo u kanvas i konfigurišemo dimenzije kanvasa da se promene u skladu sa veličinom slike.

Sada kada smo učitali sliku potrebno je da odredimo konture kostiju i da izracunamo površinu.(Tehnički nismo ni morali da učitavamo sliku jer sve što nam je potrebno za sledeću metodu jeste promenljiva save\_for\_cont u kojoj se nalazi naša slika, ali radi lepšeg doživljaja korisnika aplikacije učitali smo sliku prvo pre izbacivanja nove slike kontura.) Prvo je potrebno definisati metodu koja će biti pozvana pritiskom na dugme IZRACUNAJ. Ime metode je racunaj\_konture.

**def racunaj\_konture(self):**

U slučaju da korisnik pritisne dugme IZRACUNAJ pre nego što je slika učitana, moramo ga obavestiti da mora pre svega da je učita.

**if self.file\_path == "":**

**self.text.config(text="moras prvo ucitati sliku ! ")**

**return**

Nakon toga podešavamo threshold. Dve najbitnije stavke ovde su ova dva broja 0.55 i 1 jer oni predstavljeju sam opseg thresholda tj. piksele koje svetline će on da uzima. S obzirom da su kosti najsvetliji delovi slike njih će kupiti. Igrajući se sa opsegom došli smo do zaključka da je 0.55 najoptimalniji za prepoznavanje[3].

**\_, t1 = cv2.threshold(self.save\_for\_cont,0.55,1,cv2.THRESH\_BINARY)**

Zatim je potrebno dati array prebaciti u int8 da bismo našli konture izračunali površinu.[4][5].

**t1 = t1.astype(np.uint8)**

**contours, \_  = cv2.findContours(t1,1,2)**

**area = cv2.contourArea(contours[0])**

Nakon toga niz vraćamo u float32 da bismo kreirali sliku koju ćemo smestiti u canvas kao u prošloj metodi.

**t1 = t1.astype(np.float32)**

**image = Image.fromarray(np.uint8(cm.gist\_earth(t1)\*255))**

**image= ImageTk.PhotoImage(image)**

**self.canvas.image= image**

**self.canvas.create\_image(0,0 ,image=image, anchor="nw")**

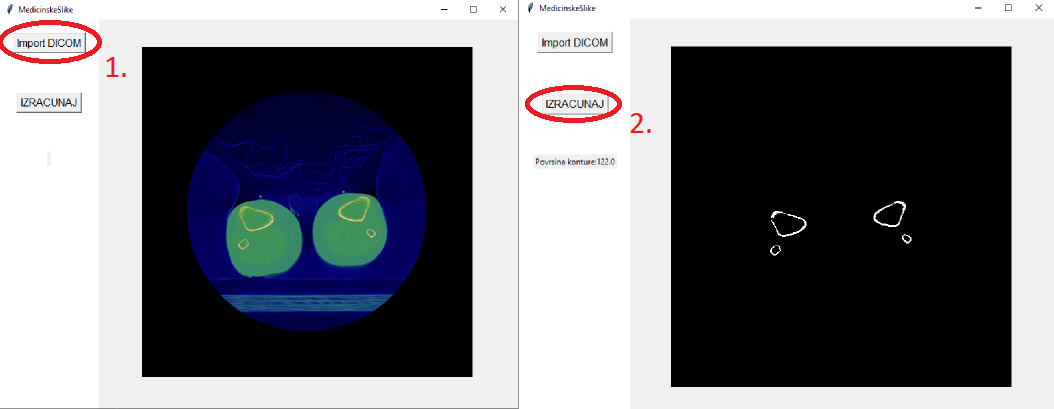
Na kraju ćemo samo konfigurisati naš lable da u njemu piše površina.

**self.text.config(text=f"Povrsina konture:{area}")**

I sve što ostaje na kraju jeste da pozovemo klasu u glavnom programu.

**gui = GUI()**

Aplikaciju koristimo tako što pritisnemo na dugme Import DICOM(slika 1, pozicija 1) i odaberemo sliku .dcm formata koju ćemo učitati. Zatim pritisnemo na dugme IZRAČUNAJ(slika 1, pozicija 2) da bismo dobili novu sliku sa izdvojenim konturama i ispisanom površinom.

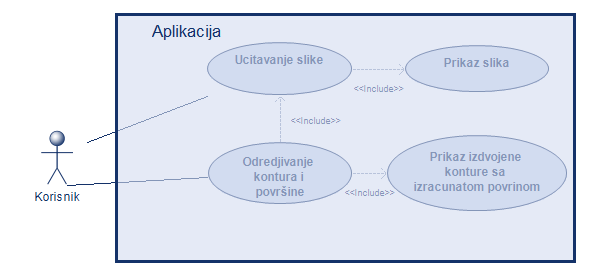


Slika 1 - Prikaz rada aplikacije

# UML DIJAGRAMI

# Use case dijagram

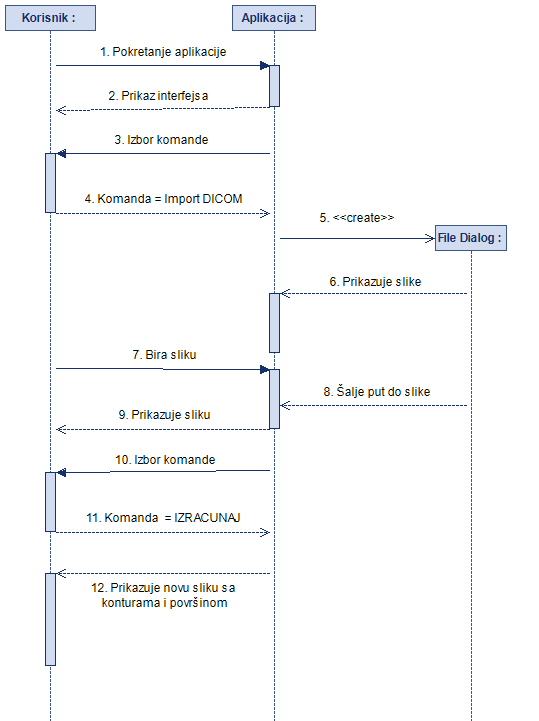
Use case dijagram prikazuje funkcionalnost aplikacije na najvišem nivou(slika 2).



Slika 2 - Use case dijagram

# Dijagram sekvenci

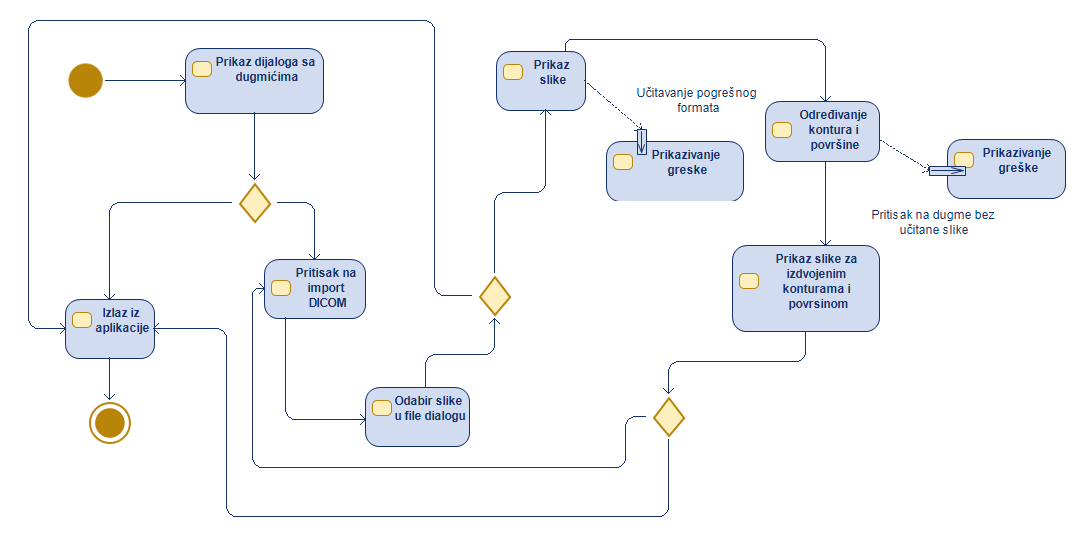
Ova vrsta dijagrama daje vremenski model dinamičkog aspekta sistema. Na slici 3 prikazan je dijagram sekvenci za slčajeve “Učitavanje slike” I “Odredjivanje kontura I površine”.



Slika 3 - Dijagram sekvenci za slučajeve "Učitavanje slike" i "Odredjivanje kontura i površine"

# Dijagram aktivnosti

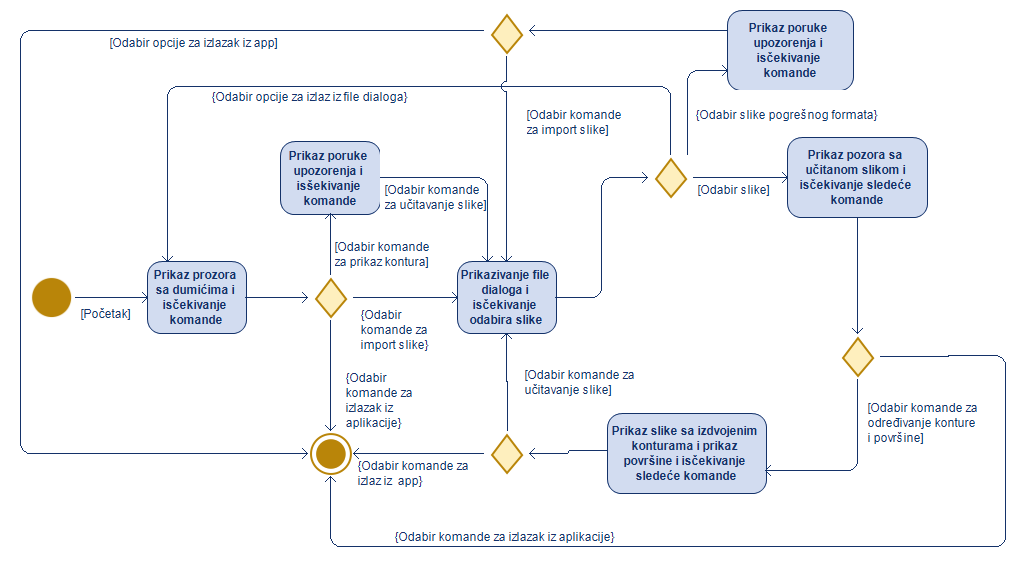
Dijagram aktivnosti prikazuje tok akcija u aplikaciji (slika 4).



Slika 4 - Dijagram aktivnosti

# Dijagram stanja

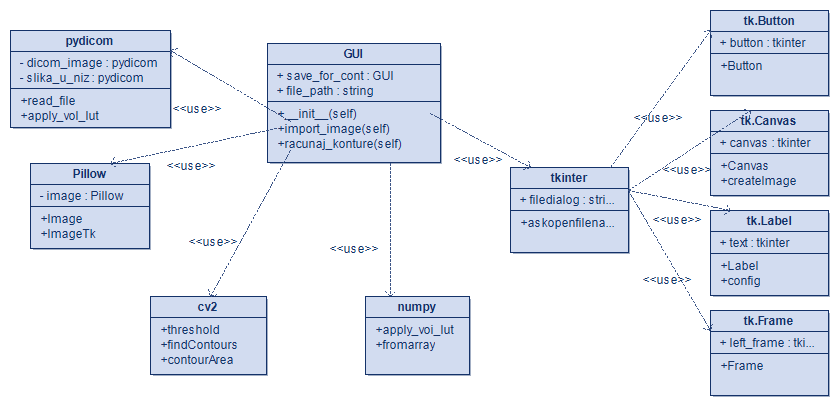
Dijagram stanja prikazuje različita stanja u kojoj se aplikacija nalazi pri interakciji sa korisnikom (slika5).



Slika 5 - Dijagram stanja

# Dijagram klasa

Dijagram klasa predstavlja klase koje čine aplikaciju kao I njene attribute I metode (slika 6).



Slika 6 - Dijagram klasa

# ZAKLJUČAK

Ovu aplikaciju je najefikasine bilo odraditi u python-u, s obzirom da već postoje bilblioteke koje su nam omogućile da sve odradimo efikasno I spakujemo u što manje linija koda. Ideja je bila da se program napravi tako da bude jednostavan za upotrebu da svako može da ga koristi, zbog toga ima samo dve opcije I sve je jasno određeno. Program uspešno učitava sliku I nakon toga veoma jasno prikazuje konture konstiju, što može biti od velike koristi samom korisniku u zavisnosti od toga zašta mu je potreban pregled kostiju. Sam rad na ovom projektu je bio jako dobra praksa za dalji rad u oblasti bioinženjeringa.

# LITERATURA

[]- OpenCV - Pristupljeno: 30.07.2023. - <https://docs.opencv.org/3.4/d6/d00/tutorial_py_root.html>

[1] – Image Editing Tool - Code First with Hala – Pristupljeno: 30.07.2023. -<https://www.youtube.com/watch?v=5V_cPy2dtTc&list=PLs3IFJPw3G9IiHm9PEP1UaMtuvACmxVMj&index=12>

[2] – StackOverflow – From array to image - Pristupljeno: 30.07.2023. -<https://stackoverflow.com/questions/10965417/how-to-convert-a-numpy-array-to-pil-image-applying-matplotlib-colormap>

[3] – OpenCV – threshold - Pristupljeno: 30.07.2023. -<https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>

[4] – LearnOpenCV.com - Find Contours - Pristupljeno: 30.07.2023. - <https://learnopencv.com/contour-detection-using-opencv-python-c/>

[5] – OpenCV – contourArea - Pristupljeno: 30.07.2023. -<https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html>