UNIVERZITET U ZENICI

Politehnički fakultet

Softversko inženjerstvo

Računarska grafika

DIPLOMSKI RAD

RAZVOJ SOFTVERA ZA ZAMUĆIVANJE LICA U DIGITALNIM VIDEOZAPISIMA

Samed Mujkanović

Mentor: Prof. dr. Samir Lemeš

Zenica, 2025. godine

BIBLIOGRAFSKA KARTICA RADA

NAUČNO PODRUČJE RADA: Tehničke nauke

NAUČNO POLJE RADA: Elektrotehnika, elektronika i informaciono inženjerstvo

NAUČNA GRANA RADA: Grafika i vizualizacija

NAZIV RADA: Razvoj softvera za zamućivanje lica u digitalnim videozapisima

USTANOVA U KOJOJ JE IZRAĐEN RAD: Univerzitet u Zenici, Politehnički fakultet u Zenici

MENTOR RADA: Prof. dr. Samir Lemeš

DATUM ODBRANE RADA: 28.06.2025

ČLANOVI KOMISIJE ZA ODBRANU:

1. V.prof. dr. Nevzudin Buzađija,
2. Prof. dr. Samir Lemeš,
3. Prof. dr. Edin Berberović.

IZJAVA

Izjavljujem da sam diplomski rad pod naslovom „Razvoj softvera za zamućivanje lica u digitalnim videozapisima“ izradio samostalno pod mentorstvom Prof. dr. Samir Lemeš. U radu sam koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, zaključke, stavove, teorije i zakonitosti koje sam izravno ili parafrazirajući ih naveo u diplomskom radu na uobičajan, standardan način, povezao sam sa fusnotama i korištenim bibliografskim jedinicama.

Student:
Samed Mujkanović

SAŽETAK

Računarski vid sve više dobiva na značaju u savremenim softverskim rješenjima, naročito kada je riječ o zaštiti privatnosti u digitalnim medijima. Ovaj diplomski rad fokusira se na razvoj softverske aplikacije za automatsko zamućivanje lica u videozapisima u stvarnom vremenu. Aplikacija je implementirana u programskom jeziku Python koristeći OpenCV biblioteku. U okviru rada istražene su i upoređene različite metode prepoznavanja lica, poput Haar Cascade klasifikatora i DNN modela. Analizirane su i različite metode za zamućivanje slike (Gaussian Blur, Median Blur, Pixelation). Softver omogućava učitavanje videozapisa sa kamere ili iz video datoteke, automatsko prepoznavanje lica te primjenu zamućivanja isključivo na te regije, uz mogućnost podešavanja parametara putem korisničkog interfejsa. Dokumentiran je cijeli razvojni proces, uključujući dizajn arhitekture, implementaciju i testiranje performansi na različitim video rezolucijama.

Ključne riječi: računalni vid, zamućivanje lica, Python, OpenCV, prepoznavanje lica, video obrada.

ABSTRACT

Computer vision is gaining increasing importance in modern software solutions, particularly in the context of privacy protection in digital media. This thesis focuses on the development of a software application for real-time face blurring in video footage. The application is implemented in the Python programming language using the OpenCV library. Various face detection methods, including the Haar Cascade classifier and DNN-based models, were explored and compared. In addition, multiple image blurring techniques (Gaussian Blur, Median Blur, and Pixelation) have been analyzed and implemented. The software supports video input from both live camera feeds and video files, performs automatic face detection, and applies blurring exclusively to the identified regions, with adjustable parameters via a user interface. The entire development process is documented, including software architecture design, implementation, and performance testing across different video resolutions.

Keywords: computer vision, face blurring, Python, OpenCV, face detection, video processing.

SADRŽAJ

1. UVOD

U savremenom digitalnom društvu, videozapisi predstavljaju ključni medij za komunikaciju, nadzor, dokumentiranje događaja i dijeljenje sadržaja na internetu. Njihova sveprisutnost u svakodnevnom životu donosi brojne prednosti, ali istovremeno otvara ozbiljna pitanja u vezi sa zaštitom privatnosti pojedinaca. Snimci sa nadzornih kamera, društvenih mreža, javnih događaja i medijskih izvještaja često sadrže lica osoba koje nisu dale saglasnost za javno prikazivanje, što može predstavljati kršenje prava na privatnost.

Jedna od najčešće korištenih i najjednostavnijih metoda za zaštitu identiteta u videozapisima jeste zamućivanje lica. Ova tehnika omogućava skrivanje prepoznatljivih karakteristika lica bez značajnog narušavanja konteksta scene. Korištenje algoritama za automatsko prepoznavanje lica u kombinaciji sa tehnikama zamućivanja postaje sve relevantnije u oblastima kao što su video nadzor, forenzika, novinarstvo, zaštita maloljetnika u medijima, kao i u softverima za obradu i uređivanje videozapisa.

Cilj ovog diplomskog rada je razvoj softverske aplikacije koja omogućava automatsko prepoznavanje i zamućivanje lica u digitalnim videozapisima u stvarnom vremenu. Aplikacija je implementirana u programskom jeziku Python, koristeći biblioteku OpenCV, koja predstavlja jedan od najraširenijih alata za računalni vid. Softver će podržavati obradu videozapisa uživo (web kamera) ili iz postojećih video datoteka, prepoznavanje lica korištenjem OpenCV-ovih ugrađenih metoda kao što su Haar Cascade klasifikator i DNN (Deep Neural Network) modeli, te primjenu različitih efekata zamućivanja kao što su Gaussian Blur, Median Blur i Pixelation.

Također, korisniku je omogućeno podešavanje ključnih parametara zamućivanja kao što su intenzitet, veličina područja i vrsta filtera, putem jednostavnog komandnog interfejsa ili grafičkog korisničkog okruženja. U okviru rada biće obrađena i arhitektura aplikacije, performanse sistema pri različitim video rezolucijama i uslovima, te mogućnosti proširenja funkcionalnosti u budućnosti.

U kontekstu prethodnih istraživanja i dostupne literature, razvoj ove aplikacije oslanja se na postojeće metode i alate. OpenCV zvanična dokumentacija i tutorijali (1, 2) bili su osnovni izvor za razumijevanje rada sa slikama i video streamovima u Pythonu. Implementacija Haar Cascade metode (3) direktno se temelji na vodiču za prepoznavanje lica koji koristi unaprijed

istrenirane modele, dok je praktični tutorijal za zamućivanje lica u stvarnom vremenu (4) poslužio kao korisna smjernica za kombinovanje prepoznavanja i obrade. Dodatno, naučni rad Nukala i saradnika (5) ispituje primjenu dubokog učenja za prepoznavanje lica u zamućenim slikama, naglašavajući potencijal takvih tehnika u kompleksnijim scenarijima. Rad Imrana i kolega (6) prikazuje okvir za napredno prepoznavanje lica u video nadzoru, što potvrđuje potrebu za alatima poput ovog rada u kontekstu zaštite privatnosti. Ovi izvori su dali značajan doprinos kako u oblikovanju aplikacije, tako i u razumijevanju šireg konteksta problema prepoznavanja i anonimizacije lica.

2. KORIŠTENE TEHNOLOGIJE

U ovom diplomskom radu korišten je skup savremenih tehnologija i biblioteka koje omogućavaju obradu slike i videozapisa, prepoznavanja objekata te razvoj funkcionalne aplikacije u programskom jeziku Python. U nastavku su detaljno opisane najvažnije korištene tehnologije:

2.1 Python

Python je interpretirani, višenamjenski programski jezik visokog nivoa poznat po svojoj jednostavnoj sintaksi i čitljivosti. Zahvaljujući velikoj zajednici korisnika i brojnim dostupnim bibliotekama, Python je postao standard u mnogim područjima, uključujući računarski vid (computer vision)**,** obradu podataka**,** mašinsko učenje **i** automatizaciju zadataka**.**

U ovom radu Python je korišten kao osnovni jezik za razvoj aplikacije zbog svoje kompatibilnosti s OpenCV bibliotekom, mogućnosti rada s realnim vremenom i podrške za rad s videom i slikom.

2.2 OpenCV

OpenCV (Open Source Computer Vision Library) je otvorena biblioteka namijenjena obradi slike i videozapisa, te razvoju aplikacija iz oblasti računarskog vida. Biblioteka je razvijena u C++, ali ima punu podršku i za Python, što je čini izuzetno pogodnom za brzo prototipiranje i razvoj.

OpenCV nudi brojne funkcionalnosti kao što su:

- Prepoznavanje lica i objekata,
- Obrada i transformacija slika,
- Zamućivanje, filtriranje i segmentacija slike,
- Praćenje objekata u videozapisu,
- Rad sa kamerama i video datotekama.

U okviru ovog rada, OpenCV se koristi za učitavanje i obradu video snimka, prepoznavanje lica pomoću Haar Cascade i DNN modela, te za primjenu različitih tehnika zamućivanja slike.

2.3 Haar Cascade Classifier

Haar Cascade je metoda za prepoznavanje objekata zasnovana na Haar značajkama, koju je razvio Paul Viola i Michael Jones. Ova tehnika koristi unaprijed trenirane klasifikatore za brzo prepoznavanje objekata, naročito lica, na slici ili u videozapisu.

Prednosti Haar Cascade metode:

- Brzo prepoznavanje u stvarnom vremenu,
- Laka integracija putem OpenCV biblioteke,
- Dostupni gotovi modeli za prepoznavanje lica, očiju, profila itd.

Iako je Haar Cascade brz i jednostavan za upotrebu, ima određena ograničenja kada je riječ o prepoznavanju pod različitim uglovima, slabom osvjetljenju i zaklonjenim licima.

2.4 DNN (Deep Neural Network) Modul iz OpenCV-a

Za naprednije i tačnije prepoznavanje lica, koristi se **DNN modul** iz OpenCV biblioteke koji omogućava učitavanje i izvođenje unaprijed treniranih konvolucijskih neuronskih mreža (CNN). OpenCV podržava različite formate modela, uključujući Caffe, TensorFlow, ONNX i Torch.

U ovom radu koristi se unaprijed trenirani model za prepoznavanje lica baziran na *ResNet-10 arhitekturi* optimiziranoj za brzinu i efikasnost. Ova metoda omogućava bolje prepoznavanje pri različitim pozicijama glave, kvalitetnijim rezultatima u slabijim uslovima osvjetljenja i većoj robusnosti u odnosu na Haar Cascade.

2.5 Algoritmi za zamućivanje slike

U aplikaciji se koriste tri različite tehnike zamućivanja kako bi se omogućila fleksibilnost i poređenje rezultata:

- **Gaussian Blur**: koristi gausovsku funkciju za izračunavanje težine susjednih piksela i stvara gladak, prirodan efekat zamućenja. Pogodan je za većinu slučajeva jer zadržava ivice i konture.

- **Median Blur**: zamjenjuje svaki piksel median vrijednošću iz susjedstva. Efikasan je kod uklanjanja šuma i često daje bolje rezultate kod zamućivanja detalja poput lica.
- **Pixelation (blokovsko zamućivanje)**: smanjuje rezoluciju regije, čime se stvara efekat „pikselizacije“. Koristi se često za anonimnost u medijskim sadržajima jer potpuno uklanja prepoznatljive karakteristike lica.

2.6 Video Input (Kamera i Video datoteka)

Aplikacija omogućava obradu videozapisa iz dva izvora:

- Uživo (web kamera) – omogućava korisniku da vidi rezultate u stvarnom vremenu,
- Video datoteka (.mp4, .avi, itd.) – korisnik može obraditi već postojeće snimke.

OpenCV-ova funkcija cv2.VideoCapture() koristi se za oba slučaja, a izlaz se prikazuje u prozoru koristeći cv2.imshow().

2.7 Korisnički interfejs (CLI i/ili GUI)

U prvoj verziji aplikacije koristi **se** komandna linija **(CLI)** za unos parametara:

- Izbor izvora videozapisa,
- Izbor algoritma za zamućivanje,
- Jačina i veličina filtera.

Po potrebi, može se proširiti i jednostavnim grafičkim korisničkim interfejsom (GUI) koristeći **Tkinter** ili **PyQt**.

3. PREPOZNAVANJE LICA I METODE ZAMUĆIVANJA

3.1 Metode prepoznavanja lica

Prepoznavanje lica predstavlja osnovu svakog sistema koji želi vršiti analizu ili obradu video sadržaja u kojem su prisutni ljudi. Identifikacija regije slike u kojoj se nalazi lice korisnika omogućava selektivnu obradu – u ovom slučaju, zamućivanje. U okviru ovog rada razmatrane su dvije najčešće korištene metode prepoznavanja lica u biblioteci OpenCV: Haar Cascade i DNN (Deep Neural Network).

3.1.1 Haar Cascade Classifier

Haar Cascade klasifikator predstavlja jednu od najranijih i najraširenijih metoda za prepoznavanje lica i objekata u stvarnom vremenu. Razvijen od strane Paula Violle i Michaela Jonesa (2001), ovaj algoritam koristi tzv. **Haar-like karakteristika** – pravougaone obrasce koji predstavljaju razlike između svijetlih i tamnih regija slike.

Princip rada uključuje prolazak kroz veliki broj malih prozora unutar slike i evaluaciju svake regije pomoću unaprijed definisanih Haar značajki. Klasifikator je treniran pomoću **AdaBoost** algoritma, koji selektuje najrelevantnije značajke iz skupa od nekoliko hiljada i kombinuje ih u kaskadne nivoe – niz klasifikatora koji postepeno filtriraju negativne regije.

Prednosti:

- **Efikasnost:** Omogućava veoma brzo prepoznavanje lica i objekata, pogodna za rad u stvarnom vremenu.
- **Jednostavna integracija:** Lako se koristi preko OpenCV biblioteke uz već dostupne XML datoteke s treniranim modelima.
- **Niski hardverski zahtjevi:** Ne zahtijeva GPU podršku, može raditi i na slabijim uređajima.

Nedostaci:

- **Ograničena preciznost:** Osjetljiv je na promjene osvjetljenja, položaja lica i zaklonjenost (npr. djelimično prekriveno lice).

- **Visok broj lažno pozitivnih prepoznavanja:** Može pogrešno prepoznati druge objekte kao lice.
- **Nerobustan na varijacije:** Ne prepoznaje dobro rotirana lica i profile.

Uprkos svojim ograničenjima, Haar Cascade se i dalje koristi u aplikacijama gdje su brzina i jednostavnost važnije od visoke preciznosti.

3.1.2 DNN (Deep Neural Network) metoda

DNN pristup prepoznavanja lica koristi **konvolucijske neuronske mreže (CNN)**, koje omogućavaju mnogo dublje i apstraktnije razumijevanje vizualnih obrazaca. U ovom radu koristi se **ResNet-10** model treniran pomoću **Single Shot Detector (SSD)** algoritma, implementiran kroz cv2.dnn modul OpenCV biblioteke.

Model prihvata ulaznu sliku, skalira je na 300x300 piksela i prolazi kroz duboku mrežu slojeva koji generišu **bounding box** koordinate lica. Ova metoda značajno smanjuje broj lažnih prepoznavanja i nudi veću preciznost u različitim uslovima.

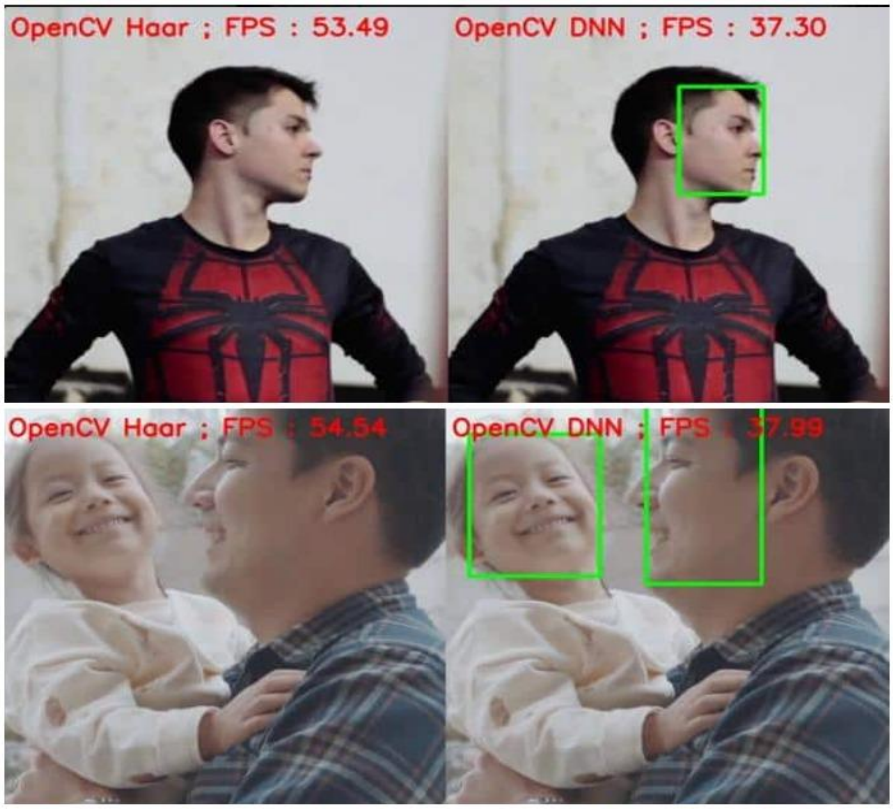
Prednosti:

- **Visoka tačnost i preciznost:** Pruža pouzdane rezultate čak i kod zaklonjenih ili djelimično osvijetljenih lica.
- **Robusnost:** Efikasno detektuje više lica odjednom, različitih veličina i orijentacija.
- **Fleksibilnost:** Pogodan za različite primjene – od nadzora do interaktivnih sistema.

Nedostaci:

- **Performanse:** Zahtijeva više resursa, a brzina obrade može biti ograničena na sistemima bez GPU podrške.
- **Kompleksnost implementacije:** Zahtijeva dodatne datoteke modela (.prototxt, .caffemodel) i više memorije.

DNN modeli su trenirani na velikim skupovima slika kao što su WIDER FACE dataset, što ih čini otpornim na različite izazove u realnim okruženjima.



Slika 1. Razlika između Haar i DNN modela.

3.2 Algoritmi za zamućivanje slike

Zamućivanje slike (engl. blurring) je jedna od osnovnih tehnika obrade slike koja omogućava smanjenje nivoa detalja u određenoj regiji. U kontekstu zaštite privatnosti, posebno kod videozapisa i nadzornih snimaka, cilj zamućivanja je prikrivanje prepoznatljivih osobina lica kako bi se spriječila identifikacija. U ovom radu implementirane su tri metode zamućivanja: **Gaussian Blur**, **Median Blur** i **Pixelation**, koje se razlikuju po načinu rada, vizuelnim efektima i stepenu efikasnosti u sakrivanju identiteta.

3.2.1 Gaussian Blur

Gaussian Blur koristi matematičku funkciju poznatu kao Gausova raspodjela (normalna raspodjela) kako bi se izvršilo ponderisano srednjene piksel vrijednosti u okolini svake tačke slike. U praksi, ovo se ostvaruje primjenom konvolucije između ulazne slike i Gausovog kernela (matrice koeficijenata) definisanog standardnom devijacijom (sigma).

Tehnička specifikacija:

- Kernel je uvijek neparne dimenzije (npr. 3x3, 5x5, 15x15).
- Svaki piksel se zamjenjuje prosjekom svojih susjeda, gdje pikseli bliži centru imaju veću težinu.
- U OpenCV-u se koristi cv2.GaussianBlur() funkcija.

Prednosti:

- Vizuelno najprirodnije zamućenje.
- Zadržava osnovne konture lica i pozadine.
- Idealno za situacije kada nije potrebna potpuna anonimizacija već samo redukcija detalja.

Nedostaci:

- Ako je kernel mali, lice može ostati djelimično prepoznatljivo.
- Za veću zaštitu mora se koristiti veći kernel, što povećava procesorsko opterećenje i smanjuje performanse u stvarnom vremenu.
- Nije efikasan u uklanjanju šuma jer zadržava određeni nivo detalja.

Upotreba u praksi:

- Često se koristi u medijima kada je potrebno "ublaženo" prikrivanje identiteta.
- Prikladan za niskorizične slučajeve kada nije nužna potpuna cenzura.

3.2.2 Median Blur

Median Blur zamjenjuje vrijednost svakog piksela medijanom vrijednosti iz njegove okoline (npr. 3x3 ili 5x5 piksel blokova). Za razliku od Gaussian Blur-a koji koristi prosjek, medijan je otporniji na ekstremne vrijednosti i šum.

Tehnička specifikacija:

- Koristi se cv2.medianBlur() funkcija u OpenCV-u.
- Kernel mora biti neparan broj (npr. 3, 5, 7).

- Ne dolazi do izračuna ponderisanih vrijednosti, već se vrši sortiranje i pronalazak srednje vrijednosti.

Prednosti:

- Izuzetno efikasan protiv "salt and pepper" šuma.
- Očuvanje ivica bolje nego kod Gaussian Blur.
- Pogodniji za male objekte i visoke kontraste unutar lica (npr. oči, obrve, usne).

Nedostaci:

- Može proizvesti grublji efekat zamućivanja, što može izgledati manje estetski.
- Ne daje glatke prelaze, pa regije zamućenja mogu izgledati "isjeckano".
- Nije pogodan za slike visoke rezolucije kada je potrebna fina kontrola.

Upotreba u praksi:

- Koristi se u forenzičkoj analizi i digitalnoj obradi slike gdje je potrebno uklanjanje šuma bez gubitka ivica.
- Pogodan za zaštitu identiteta u zatamnjenim ili šumovitim snimcima.

3.2.3 Pixelation (Pikselizacija)

Pixelation je tehnika kojom se određena regija slike smanjuje na vrlo nisku rezoluciju (npr. 10x10 piksela), a zatim se ta ista regija ponovo skalira na izvornu veličinu. Efekat koji se dobiva jeste izgled velikih "blokova", čime se potpuno brišu detalji.

Tehnička specifikacija:

- Regija se prvo "resize-a" na manju dimenziju pomoću `cv2.resize()` uz `INTER_LINEAR` ili `INTER_NEAREST`.
- Zatim se ista regija ponovo proširi na originalnu dimenziju.
- Najčešće se koristi dimenzije poput 10x10, 20x20 kao osnovna redukcija.

Prednosti:

- Najefikasnija metoda za potpunu anonimizaciju lica.
- Potpuno uklanja sve osobine lica, nemoguće za prepoznavanje algoritmima.
- Jasno ukazuje korisniku da je došlo do namjerne cenzure.

Nedostaci:

- Efekat može izgledati grubo i narušiti vizuelnu estetiku snimka.
- Nema mogućnosti "mekog prijelaza" između regije i ostatka slike.
- Teže se implementira ako regije lica nisu pravilnog oblika (potrebno prethodno prilagoditi bounding box).

Upotreba u praksi:

- Standardna tehnika u tabloidnim medijima, YouTube videozapisima i nadzornim sistemima gdje je potrebna čvrsta zaštita identiteta.
- Preporučena za upotrebu u visoko osjetljivim sadržajima (maloljetnici, svjedoci, žrtve)

3.2.4 Detaljna analiza svih metoda zamućivanja

Tabela usporedbe metoda za zamućivanje lica

Karakteristika	Gaussian Blur	Median Blur	Pixelation
Osnovni princip	Konvolucija slike sa Gaussovom funkcijom radi ponderisanog prosjeka piksela	Zamjena vrijednosti piksela sa medijanom piksela iz okoline	Smanjenje rezolucije regije i njeno ponovno skaliranje uz interpolaciju
Vizuelni rezultat	Glatki prelazi, mehko zamućenje koje zadržava osnovne konture	Grublje zamućenje, često očuvanje ivica i obrisa	Jasan „blokovski” efekat s potpunim gubitkom detalja
Estetika	Estetski ugodna, prirodan izgled	Umjereno prijatan izgled, djelimično „grub”	Vizuelno jako uočljivo i manje estetski

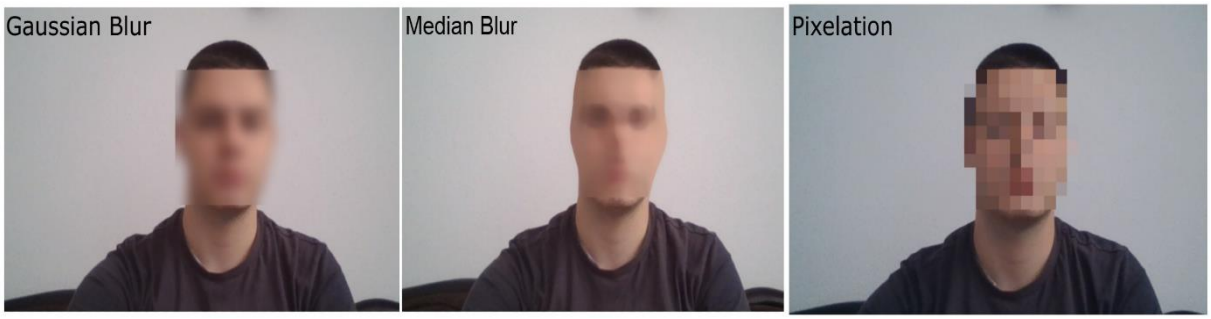
Karakteristika	Gaussian Blur	Median Blur	Pixelation
Nivo anonimnosti	Nizak do srednji – moguće prepoznavanje pri manjem kernelu	Srednji – bolje skriva karakteristike od Gaussian metode	Visok – gotovo potpuna zaštita identiteta
Otpornost na šum	Ograničena – prosjek može biti osjetljiv na šum	Vrlo dobra – efikasno uklanja „salt-and-pepper" šum	Nije direktno namijenjen uklanjanju šuma, ali degradira sve informacije
Zadržavanje ivica	Gubi ivice zbog izravnavanja	Dobro zadržava ivice	Potpuno gubi ivice zbog blokova
Brzina izvođenja	Brza uz mali kernel; usporava s većim kernelima	Srednje spora (sortiranje u svakom prozoru)	Vrlo brza, jednostavna za implementaciju
Kontrola intenziteta	Podešava se veličinom kernela (standardna devijacija)	Podešava se samo veličinom prozora	Podešava se faktorom skaliranja (npr. 0.1x, 0.2x)
Primjene	Softversko zamućivanje, neagresivna cenzura, estetski filteri	Uklanjanje šuma, obrada medicinskih slika, prepoznavanje ivica	Potpuna cenzura u medijima, novinarstvu, nadzoru
Pogodnost za stvarno vrijeme	Uglavnom da, ali slabija s velikim kernelima	Ograničeno stvarno vrijeme (ovisno o veličini slike)	Odlična za stvarno zbog jednostavne strukture
GPU podrška	Dostupna i optimizirana u većini biblioteka (npr. OpenCV)	Lošije skalira s GPU jer sort algoritmi nisu trivijalni za paralelizaciju	Vrlo jednostavno implementirati na GPU

3.2.5 Zaključna poređenja i preporuke

- Ako je cilj postizanje balansa između zamućenja i očuvanja estetike**,** Gaussian Blur je najprikladniji. Daje uglađen izgled i koristi se tamo gdje prepoznavanje ne mora biti potpuno spriječeno, kao u slučajevima blagog cenzurisanja.

- Ako je cilj zadržati ključne strukture lica, ali ukloniti detalje i šum**,** Median Blur nudi kompromis. Pogodan je za slike sa visokim kontrastom, niskom rezolucijom i izraženim šumom. Efikasno zamagljuje bez uništavanja ivica.

- Ako je cilj potpuna zaštita identiteta**,** Pixelation je najsigurnija metoda. Ona u potpunosti uklanja vizuelne informacije iz regije i ne ostavlja nikakav prepoznatljiv trag. Koristi se u medijima, nadzornim sistemima, pravnoj dokumentaciji i zaštiti osjetljivih subjekata.



Slika 2. Prikaz algoritama za zamućivanje u realnoj upotrebi

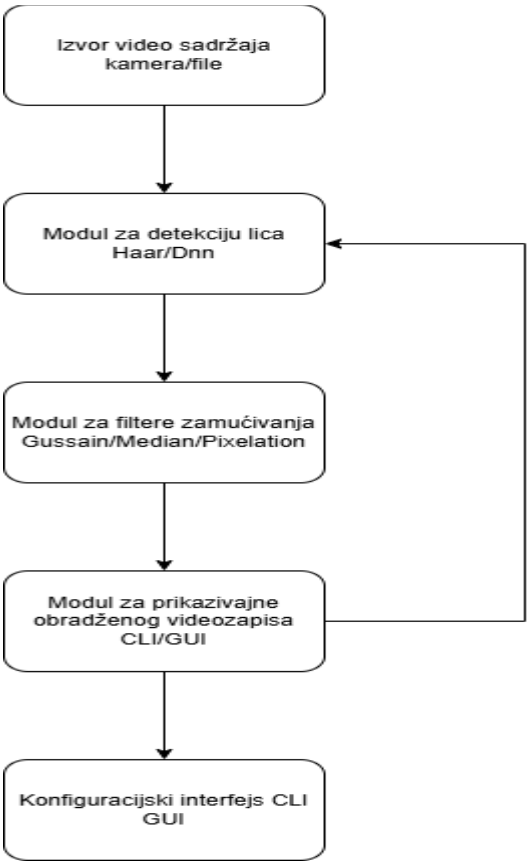
4. ARHITEKTURA SOFTVERSKOG RJEŠENJA

U cilju efikasne, modularne i proširive implementacije aplikacije za zamućivanje lica u videozapisima, dizajnirana je softverska arhitektura zasnovana na principima dobre organizacije koda, ponovne upotrebljivosti i jednostavne nadogradnje. Aplikacija je razvijena korištenjem programskog jezika **Python** uz pomoć **OpenCV biblioteke**, te dodatnih modula za rad sa interfejsom, argumentima komandne linije i konfiguracijom.

4.1 Pregled arhitekture

Aplikacija je podijeljena u sljedeće logičke komponente:

- **Modul za učitavanje videozapisa** (kamera ili datoteka),
- **Modul za prepoznavanje lica** (Haar Cascade ili DNN),
- **Modul za zamućivanje detektovanih regija** (Gaussian, Median, Pixelation),
- **Modul za prikaz i upis obrađenog videozapisa**,
- **Korisnički interfejs (CLI ili GUI)** za upravljanje parametrima aplikacije.



Slika 3. Dijagram arhitekture

4.2 Opis komponenti

4.2.1 Učitavanje video streama

Koriste se OpenCV metode cv2.VideoCapture() koje omogućavaju otvaranje kamere (0) ili video datoteke (.mp4, .avi, itd.). Aplikacija automatski prepoznaje izvor na osnovu korisničkog unosa.

4.2.2 Prepoznavanje lica

Ova komponenta nudi dvije mogućnosti:

- **Haar Cascade Classifier**: Brza, ali manje tačna metoda bazirana na pravougaonim obrascima.
- **DNN model**: Koristi unaprijed treniranu mrežu (res10_300x300_ssd_iter_140000 caffemodel) u kombinaciji sa deploy.prototxt.
- Detektovana lica se vraćaju u obliku koordinata pravougaonika koji definiše područje za dalju obradu.

4.2.3 Zamućivanje regija

Na svako detektovano lice primjenjuje se algoritam koji korisnik odabere:

- GaussianBlur – koristi cv2.GaussianBlur(),
- MedianBlur – koristi cv2.medianBlur(),
- Pixelation – ručno skalira lice na manju rezoluciju, pa ga vraća u originalni prostor.

4.2.4 Prikaz i snimanje videozapisa

OpenCV-ova funkcija cv2.imshow() koristi se za prikaz rezultata u stvarnom vremenu, dok se cv2.VideoWriter() može koristiti za snimanje obrađenog izlaza u datoteku.

4.2.5 Konfiguracija i korisnički interfejs

Aplikacija omogućava jednostavan unos parametara putem komandne linije (argparse) ili alternativno pomoću GUI-a razvijenog uz pomoć Tkinter biblioteke (ili PyQt ukoliko se želi napredniji interfejs).

Parametri koje korisnik može odrediti uključuju:

- Izbor izvora videozapisa (kamera/datoteka),
- Izbor metode prepoznavanja (Haar/DNN),
- Vrstu zamućenja,
- Veličinu kernela,
- Uključivanje snimanja rezultata.

5. IMPLEMENTACIJA SOFTVERA ZA ZAMUĆIVANJE LICA

5.1 Kratak opis aplikacije

Ovaj rad se bavi razvojem desktop aplikacije koja omogućava automatsko zamućenje lica u stvarnom vremenu ili prilikom obrade video datoteka. Aplikacija je razvijena koristeći programski jezik Python, uz korištenje biblioteke OpenCV za obradu videozapisa, te Tkinter biblioteke za izradu jednostavnog grafičkog korisničkog interfejsa (GUI).

Nakon pokretanja aplikacije, korisnika dočekuje intuitivno osmišljeni interfejs, koji nudi nekoliko osnovnih funkcionalnosti jasno raspoređenih kroz dugmad i menije:

Real-Time obrada – Korisnik može pokrenuti kameru uređaja i posmatrati kako se prepoznavanje i zamućenje lica odvija u stvarnom vremenu. Također, korisnik može mijenjati odabrani model zamućivanja te model prepoznavanja samog lica. Također, ti modeli se mogu mijenjati i automatski prikazivati promjene. Korisnik može snimiti videozapis preko kamere uređaja te spremiti snimak na uređaj jednim klikom. Ova funkcionalnost je pogodna za situacije u kojima je neophodna trenutna anonimnost ili nadzor sa zaštitom privatnosti.

Obrada videozapisa – Omogućena je selekcija postojećeg videozapisa sa lokalnog diska, nakon čega aplikacija automatski detektuje lica u svakom kadru videozapisa i vrši njihovo zamućenje. Također, kao i kod stvarne obrade korisnik može odabrati koju vrstu zamućivanja želi i metodu prepoznavanja lica. Proces se odvija kadar po kadar, nakon čega korisnik dobiva nov videozapis sa svim zamućenim licima.

Prepoznavanje više lica – Sistem je u stanju da detektuje više lica istovremeno, bez obzira na njihovu poziciju u kadru.

Pregled rezultata – Nakon završene obrade, korisniku se automatski prikazuje rezultat (video), kako bi mogao da vizualno potvrdi efikasnost obrade.

Jednostavan interfejs – Zahvaljujući Tkinter biblioteci, aplikacija je jednostavna za upotrebu, bez potrebe za naprednim tehničkim znanjem korisnika.

Ova aplikacija ima za cilj da posluži kao osnovna alatka za zaštitu privatnosti u digitalnim medijima, posebno u kontekstu sve češćih zahteva za anonimizacijom u medijima, obrazovanju, istraživačkim projektima i socijalnim mrežama. Korišćenje biblioteke OpenCV omogućilo je visoku efikasnost prepoznavanja i obrade, dok je upotreba Tkinter biblioteke olakšala razvoj preglednog i funkcionalnog korisničkog interfejsa.

Ovaj rad će u nastavku detaljno predstaviti teorijsku osnovu tehnologija koje stoje iza ove aplikacije, kao i samu implementaciju, izazove, testiranje i moguće pravce daljeg razvoja.

5.2 Struktura koda aplikacije

Aplikacija za prepoznavanje i zamagljivanje lica implementirana je u programskom jeziku Python i organizovana u dva odvojena Python datoteka: video_processor.py i gui.py. Ova modularna organizacija omogućava bolju čitljivost, održavanje i proširivost koda. U nastavku se daje pregled svake komponente aplikacije.

5.2.1 video_processor.py

Ova datoteka sadrži sav logički kod odgovoran za obradu videozapisa. Uključuje funkcije za:

- Prepoznavanje lica pomoću Haar i DNN metode.
- Primjenu efekta zamućenja (engl. blur) na prepoznata lica u svakom kadru videozapisa.
- Obradu videozapisa u stvarnom vremenu s kamere ili iz prethodno učitane video datoteke.

Glavne funkcije u ovoj datoteci uključuju:

- detect_and_blur_faces(frame): funkcija koja prima jedan video kadar, detektuje lica i primjenjuje zamućenje.
- process_video(source): funkcija koja otvara datu video struju i prosljeđuje kadarove na obradu.
- stop_processing(): zaustavlja obradu i oslobađa resurse.

Ova datoteka predstavlja "srce" aplikacije u smislu funkcionalnosti.

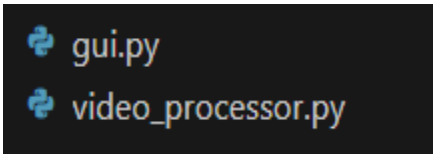
5.2.2 gui.py

Datoteka gui.py definiše grafički korisnički interfejs koristeći Tkinter biblioteku. U okviru GUI-ja korisniku su dostupne sljedeće komande:

- Pokretanje kamere (obrada u stvarnom vremenu).
- Učitavanje video datoteke sa računara.
- Zaustavljanje obrade.
- Vizuelni prikaz trenutnog kadrova obrade u Tkinter Canvas komponenti.

Ova datoteka sadrži klasu (npr. FaceBlurApp) koja povezuje dugmad, platna i ostale GUI elemente sa funkcijama iz video_processor.py. Interakcija korisnika sa GUI-jem automatski pokreće odgovarajuće funkcije za prepoznavanje i zamućivanje lica.

5.3 Veze između modula



Slika 4. Prikaz modula

U aplikaciji za prepoznavanje i zamućivanje lica, moduli gui.py i video_processor.py međusobno sarađuju kako bi omogućili obradu video datoteka i prikaz rezultata korisniku putem grafičkog korisničkog interfejsa (GUI). Svaki modul ima specifičnu ulogu u sistemu, a njihove veze su sljedeće:

5.3.1 Modul video_processor.py

- Namjena: Ovaj modul je odgovoran za tehnički dio obrade. To uključuje:
- Prepoznavanjae lica u videozapisima i s kamere (haar i dnn metode)
- Zamućenje (blur) lica različitim metodama
- Procesiranje video datoteka i kombinovanje obrađenog videozapisa sa originalnim zvukom
- Prikaz s kamere u stvarnom vremenu
- Generisanje obrađenih kadrova za prikaz u GUI

5.3.2 Modul gui.py

Namjena: Predstavlja korisnički interfejs aplikacije. Korisniku omogućava:

- Odabir između obrade video datoteke ili kamere,
- Izbor metode za prepoznavanje lica,
- Vizuelni prikaz rezultata prepoznavanja u stvarnom vremenu,
- Pokretanje i nadgledanje obrade video datoteke.

Stoga, modul video_processor.py se ponaša kao servis koji obrađuje video, dok gui.py omogućava interakciju korisnika s aplikacijom. Veza između njih ostvaruje se jednostavnim importovanjem i pozivanjem funkcije process_video, čime se postiže čista i modularna arhitektura aplikacije.

5.4 Kreiranje video_processor.py datoteke

Prije svega, potrebno je da uvezemo potrebne biblioteke za obradu videozapisa, kao i ostale biblioteke za korištenje vanjskih komadni poput ffmpeg.

```
import cv2
import os
import tempfile
import numpy as np
import subprocess
```

Kod 5. Uvoz biblioteka i priprema modela

- cv2: OpenCV, za rad sa slikama i videozapisima
- os, tempfile: rad sa datotekama i privremenim folderima
- numpy: numeričke operacije (koristi se u DNN prepoznavanju)
- subprocess: pokretanje vanjskih komandi (npr. ffmpeg)

Nakon dodavanja biblioteka, učitavamo model za HAAR prepoznavanje lica kao i DNN model.

```
haar_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

dnn_path = os.path.join(os.path.dirname(__file__), "dnn")
dnn_model = cv2.dnn.readNetFromCaffe(
    os.path.join(dnn_path, "deploy.prototxt"),
    os.path.join(dnn_path, "res10_300x300_ssd_iter_140000.caffemodel")
)
```

Kod 6. Učitavanje modela za prepoznavanje lica

Dakle, pomoću OpenCV-a učitavamo Haar Cascade model za prepoznavanje lica, a DNN model za prepoznavanje lica na osnovu Caffe frameworka. Također, datoteke moraju biti u dnn folderu uz projekat da bi sve funkcionisalo.

Sada je potrebno da importujemo funkciju prepoznavanja lica.

```
def detect_faces(frame, method="haar"):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = []

    if method == "haar":
        detected = haar_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
        faces = [(x, y, w, h) for (x, y, w, h) in detected]

    elif method == "dnn":
        h, w = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
                                     (300, 300), (104.0, 177.0, 123.0))
        dnn_model.setInput(blob)
        detections = dnn_model.forward()

        for i in range(detections.shape[2]):
            confidence = detections[0, 0, i, 2]
            if confidence > 0.5:
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (x1, y1, x2, y2) = box.astype("int")
                faces.append((x1, y1, x2 - x1, y2 - y1))

    return faces
```

Kod 7. Implementacija funkcija za prepoznavanje

Ova funkcija služi za prepoznavanje lica na datoj slici korištenjem jedne od dvije metode: Haar Cascade Classifier ili DNN (Deep Neural Network).

Parametri:

- frame: Ulazna slika (kadar iz videozapisa).
- method: String koji definiše metodu prepoznavanja. Podržane vrijednosti su "haar" i

"dnn".

Povratna vrijednost:

- Lista detektovanih lica u formatu (x, y, w, h) — pozicija i veličina svakog lica.

Opis funkcije:

1. Slika se konvertuje u grayscale za potrebe Haar metode.
2. Ukoliko je metoda "haar":
    - Koristi se detectMultiScale za prepoznavanje lica bazirano na šablonima.
3. Ukoliko je metoda "dnn":
    - Slika se preprocesira u blob, zatim se prolazi kroz neuralnu mrežu.
    - Uzima se samo rezultat gdje je vjerovatnoća (confidence) veća od 50%.
    - Rezultat se skalira nazad na dimenzije originalne slike.

Kada smo implementirali samu funkciju prepoznavanja lica, potrebno je da sada importujemo efekte za zamućivanje lica.

```
def apply_blur(frame, faces, method="gaussian"):
    h_frame, w_frame = frame.shape[:2]
    for (x, y, w, h) in faces:
        x1 = max(0, x)
        y1 = max(0, y)
        x2 = min(w_frame, x + w)
        y2 = min(h_frame, y + h)

        roi = frame[y1:y2, x1:x2]
        if roi.size == 0:
            continue

        if method == "gaussian":
            blurred = cv2.GaussianBlur(roi, (99, 99), 30)
        elif method == "median":
            blurred = cv2.medianBlur(roi, 35)
        elif method == "pixelation":
            small = cv2.resize(roi, (10, 10), interpolation=cv2.INTER_LINEAR)
            blurred = cv2.resize(small, (x2 - x1, y2 - y1), interpolation=cv2.INTER_NEAREST)
        else:
            blurred = roi

        frame[y1:y2, x1:x2] = blurred
    return frame
```

Kod 8. Implementacija efekata za zamućivanja

Ova funkcija primjenjuje efekt zamućenja na detektovana lica unutar datog video kadra (frame-a). Efekti zamućenja se mogu mijenjati, a podržane metode uključuju gaussian blur, median blur i pixelaciju.

Parametri:

- frame: Ulazni kadar slike (u formatu numpy.ndarray) na kojem se nalazi jedno ili više detektovanih lica.
- faces: Lista koordinata detektovanih lica, gdje je svako lice predstavljeno kao (x, y, w, h) — gornji lijevi ugao i dimenzije prozora.
- method: Metoda zamućenja koja se koristi. Podržane vrijednosti su:
    - "gaussian" – koristi Gaussovu funkciju za zamućenje.
    - "median" – koristi srednju vrijednost piksela.
    - "pixelation" – koristi tehniku sažimanja i ponovnog skaliranja za efekat pikselizacije.

Gaussovo zamućenje koristi dvodimenzionalnu normalnu distribuciju kako bi ublažilo detalje u slici.
Median filter zamjenjuje svaki piksel srednjom vrijednošću njegovih susjeda. Efikasan je za uklanjanje šuma.
Pikselizacija se postiže tako što se slika prvo smanji na manju dimenziju (npr. 10x10), a zatim se ponovo proširi na originalnu veličinu.

Funkcija apply_blur je ključna za zaštitu privatnosti jer omogućava skrivanje identiteta pojedinaca na videozapisima. Takođe, njena fleksibilnost u izboru metode zamućenja omogućava korisniku da izabere najprikladniji vizuelni efekat prema specifičnim potrebama.

Nakon implementacije efekata za zamućivanje potrebno je da se sve to obradi. Krecemo u implementaciju funkcije za kompletnu obradu video datoteka: otvara ulazni video, detektuje lica u svakom kadru, primjenjuje zamućenje nad tim licima i na kraju snima novi video sačuvavši izvorni zvuk. Glavna svrha ove funkcije jeste omogućiti automatsku zaštitu privatnosti na videozapisima.

```python
def process_video_file(input_path, output_path, detection_method="haar", blur_method="gaussian"):
    cap = cv2.VideoCapture(input_path)
    if not cap.isOpened():
        raise Exception("Ne mogu otvoriti ulazni video.")

    fps = cap.get(cv2.CAP_PROP_FPS)
    width  = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    temp_fd, temp_output_path = tempfile.mkstemp(suffix=".mp4")
    os.close(temp_fd)

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(temp_output_path, fourcc, fps, (width, height))

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        faces = detect_faces(frame, detection_method)
        frame = apply_blur(frame, faces, blur_method)
        out.write(frame)

    cap.release()
    out.release()

    combine_audio(input_path, temp_output_path, output_path)
    os.remove(temp_output_path)
```

Kod 9. Implementacija obrade video datoteke

Parametri:

- input_path: Putanja do ulazne (neobrađenog) video datoteke.
- output_path: Putanja na koju će se sačuvati konačna video datoteka sa zamućenim licima i originalnim zvukom.
- detection_method: Metoda prepoznavanja lica – može biti "haar" (klasična) ili "dnn" (neuronska mreža).
- blur_method: Metoda zamućenja – "gaussian", "median" ili "pixelation".

Objašnjenje funkcije:

- Na početku funkcije, otvara se video. Otvara se video datoteka pomoću OpenCV-a.
- Ako video ne može biti otvoren (npr. datoteka ne postoji ili je oštećena), baca se greška.
- Nakon toga, dobavljaju se osnovne informacije o videozapisu: broj sličica po sekundi (FPS), širina i visina.
- Kreira se privremena datoteka za zapisivanje obrađenog videozapisa. Ova datoteka se koristi jer se u početku obrađuje samo video bez zvuka.
- Konfiguriše se VideoWriter objekat koji omogućava upisivanje kadrova u novu video datoteku, koristeći MP4 kodek.

Glavna petlja funkcije:

- Svaki kadar se čita pojedinačno iz videozapisa. Kada se dođe do kraja datoteke, petlja se prekida.
- Na svakom kadru se:

    1. Detektuju lica korištenjem izabrane metode (haar ili dnn).
    2. Primjenjuje se odgovarajuća metoda zamućenja.
    3. Obrađeni kadar se snima u novu datoteku.

Na kraju funkije imamo oslobađanje memorijskih resursa vezani za ulaznu i izlaznu video datoteku, te funkciju combine_audio koristi FFmpeg kako bi izvukla originalni audio iz ulazne datoteke i dodala ga u obrađeni video i nakon toga, se briše privremena datoteka koji je sadržavala video bez zvuka.

Funkcija process_video_file predstavlja centralni mehanizam za obradu video datoteka u aplikaciji. Ona obezbjeđuje potpuno automatizovan proces prepoznavanja i zamućenja lica, čuvajući pritom originalni audio zapis.

Nakon implementacije obrade video datoteke krećemo sa implementacijom funkcije za zvuk na obrađeni videozapis. Ovu funkciju sam dodao jer mislim da znatno doprinosi boljem korisničkom iskustvu. Kao rezultat toga, funkcija combine_audio služi za spajanje obrađenog videozapisa sa originalnim zvukom iz ulazne video datoteke. Nakon što je video obrađen (lica zamućena), originalni zvuk se ponovno pridružuje kako bi se dobila konačna video datoteka koja izgleda kao original – ali sa primijenjenom zaštitom privatnosti.

```
def combine_audio(original_video, processed_video, output_path):
    command = [
        "ffmpeg",
        "-y",
        "-i", processed_video,
        "-i", original_video,
        "-c:v", "copy",
        "-map", "0:v:0",
        "-map", "1:a:0?",
        "-shortest",
        output_path
    ]
    try:
        subprocess.run(command, check=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
    except subprocess.CalledProcessError as e:
        print("Greška pri kombinovanju zvuka:", e)
```

Kod 10. Implementacija funkcije za dodavanje zvuka na obrađeni videozapis

Parametri:

- original_video: Putanja do originalne (neobrađene) video datoteke koja sadrži i video i zvuk.
- processed_video: Putanja do videozapisa koji je već obrađen, ali nema zvuk.
- output_path: Putanja na koju će se sačuvati novi video sa obradom slike i originalnim zvukom.

Ovdje se formira lista argumenata za pozivanje FFmpeg alata iz komandne linije. Ključne opcije su:

- "ffmpeg" – Poziva program FFmpeg.
- "-y" – Automatski potvrđuje zamjenu datoteka ako već postoji.
- "-i", processed_video – Učitava video koji je već obrađen (nema zvuk).
- "-i", original_video – Učitava originalni video kako bi se iz njega izvukao zvuk.
- "-c:v", "copy" – Video se ne re-enkodira, već se kopira (štedi vrijeme i čuva kvalitet).
- "-map", "0:v:0" – Uzima videozapis iz prvog ulaza (processed_video).
- "-map", "1:a:0?" – Pokušava da uzme audio zapis iz drugog ulaza (original_video). Znak ? audio je opcion.
- "-shortest" – Obustavlja finalnu datoteku čim istekne najkraći stream (audio ili video).
- output_path – Putanja gdje se čuva spojeni video.

Na samom kraju funkcije imamo komandu koja se pokreće pomoću subprocess.run(). Ako dođe do greške u izvršavanju (npr. FFmpeg nije instaliran ili su datoteke neispravne), prikazuje se poruka o grešci i standardni izlaz i greške se preusmjeravaju u DEVNULL kako bi se spriječilo ispisivanje nepotrebnih poruka u konzolu.

Funkcija combine_audio omogućava da se nakon video obrade sačuva originalni audio zapis, čime se dobiva profesionalan rezultat: Video sa zamućenim licima i neizmijenjenim zvukom. Korištenje FFmpeg alata osigurava visoku efikasnost i kompatibilnost sa raznim formatima. Ova funkcija je ključna za korisničko iskustvo jer omogućava da krajnji rezultat ne izgleda "umjetno" ili tehnički degradirano.

Sada slijedi implementacija funkcije process_camera_stream koja omogućava obradu u stvarnom vremenu videozapisa sa web kamere. Ova funkcija se koristi za prepoznavanje lica i njihovo trenutno zamućivanje tokom video streama, pružajući korisniku uvid u to kako bi

zamućenje izgledalo uživo.

```
def process_camera_stream(detection_method="haar", blur_method="gaussian"):
    cap = cv2.VideoCapture(0)
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        faces = detect_faces(frame, detection_method)
        frame = apply_blur(frame, faces, blur_method)
        cv2.imshow("Real-time obrada (pritisni 'q' za izlaz)", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Kod 11. Implementacija funkcije za real time obradu

Parametri:

- detection_method: Metoda za prepoznavanje lica ("haar" ili "dnn"). Po zadanim postavkama koristi se "haar".
- blur_method: Tip metode zamućenja ("gaussian", "median" ili "pixelation"). Po zadanim postavkama koristi se "gaussian".

Opis funkcije:

- Na početku funkcije se otvara video prikaz sa zadanim postavkama web kamere. Parametar 0 označava prvu kameru na sistemu.
- Zatim ulazi se u beskonačnu petlju za čitanje kadrova iz kamere, ako kamera ne može da pročita kadar (ret == False), petlja se prekida.
- Nakon toga, na svaki pročitan kadar primjenjuje se prepoznavanje lica pomoću izabrane metode (haar ili dnn), a zatim se na prepoznata lica primjenjuje odabrana metoda zamućivanja (gaussian, median, pixelation).
- Zatim obrađeni kadar se prikazuje u prozoru sa nazivom *Real-time obrada (pritisni 'q' za izlaz)* i pritiskom na taster q korisnik može zatvoriti prozor i prekinuti obradu u stvarnom vremenu.
- Na kraju funkcije kada se završi video stream, resursi kamere se oslobađaju, a svi otvoreni prozori se zatvaraju.

Funkcija process_camera_stream je idealna za testiranje i demonstraciju rada algoritama za prepoznavanje i zaštitu identiteta u stvarnom vremenu. Ona pokazuje kako algoritmi reaguju na pokret, osvjetljenje i više lica istovremeno. Time se omogućava ne samo validacija algoritma već i uvid u njegovu korisničku upotrebljivost.

Kada smo završili sa obradom videozapisa sa web kamere, slijedi funkcija get_processed_frame predstavlja jednostavnu pomoćnu funkciju koja iz otvorenog video streama (npr. kamere) čita jedan kadar, zatim na njega primjenjuje prepoznavanje i zamućenje lica, te vraća tako obrađen kadar.

```python
def get_processed_frame(cap, detection_method="haar", blur_method="gaussian"):

    if cap is None or not cap.isOpened():
        return None

    ret, frame = cap.read()
    if not ret:
        return None

    faces = detect_faces(frame, detection_method)
    frame = apply_blur(frame, faces, blur_method)

    return frame
```

Kod 12. Implementacija pomoćne funkcije za obradu kadrova

Parametri:

- cap: Objekat tipa cv2.VideoCapture, koji predstavlja izvor video signala (npr. kamera).
- detection_method: Metoda za prepoznavanje lica, "haar" ili "dnn". Po zadanim postavkama koristi "haar".
- blur_method: Metoda za zamućivanje detektovanih lica, može biti "gaussian", "median" ili "pixelation". Po zadanim postavkama koristi "gaussian".

Objašnjenje funkcije:
Funkcija odmah provjerava da li je kamera uspješno otvorena. Ako nije, vraća None – nije moguće dobiti kadar.
Zatim se pokušava pročitati jedan kadar iz kamere. Ako čitanje nije uspješno (ret == False), funkcija opet vraća None.

Ako je sve uredu i kada je kadar uspješno pročitan, primjenjuje se: Prepoznavanje lica putem detect_faces i zamućenje detektovanih lica putem apply_blur.
Na kraju funkcije, funkcija vraća obrađeni kadar sa primijenjenim zamućenjima.

Funkcija get_processed_frame je dizajnirana kao modularni alat koji omogućava da se u bilo kojem trenutku iz video toka izdvoji i obradi jedan kadar. Ova fleksibilnost je korisna u GUI aplikacijama, gdje je potrebno često prikazivati novi obrađeni kadar bez zadržavanja tokova ili blokiranja glavne petlje.

5.5 Kreiranje datoteke gui.py

Na samom početku gui.py datoteke nalaze se importi svih potrebnih modula i biblioteka koje omogućavaju grafički prikaz, obradu slika i video sadržaja, upravljanje datotekama, kao i izvođenje svih operacija.

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import cv2
import threading
import os
import shutil
import time
from video_processor import detect_faces, apply_blur, process_video_file
```

Kod 13. Pozivanje biblioteka i druge datoteke

Objašnjenje biblioteka:

tkinter (standardna biblioteka za GUI):

- tkinter predstavlja osnovu za kreiranje grafičkog korisničkog interfejsa.
- filedialog: omogućava korisniku da odabere datoteku iz sistema putem dijaloga.
- messagebox: koristi se za prikaz informacija, grešaka i upozorenja u zasebnim prozorima.

PIL (Python Imaging Library – Pillow):

- Image: omogućava otvaranje i manipulaciju slikama.
- ImageTk: omogućava konverziju PIL slika u format koji Tkinter može prikazati (PhotoImage).

cv2 (OpenCV):

- Koristi se za učitavanje i obradu video kadrova, konverziju u različite formate (npr. BGR u RGB), i za rad sa kamerom.

threading:

- Omogućava izvršavanje funkcija u pozadini (background threads) kako GUI ne bi "zaledio" tokom dužih operacija poput obrade videozapisa.

os, shutil i time:

- os: za rad sa putanjama, datotekama i direktorijima.
- shutil: omogućava kopiranje i premještanje datoteka i foldera.
- time: koristi se za dodavanje vremenskih oznaka ili kašnjenja (sleep).

video_processor.py:

- detect_faces: funkcija za pronalazak lica na kadru.
- apply_blur: funkcija za zamućivanje detektovanih lica.
- process_video_file: kompletna obrada videozapisa (kadar po kadar) sa primjenom zamućenja i naknadnim spajanjem audio zapisa.

Nakon dodavanja biblioteka i modula, idemo na implementaciju klase FaceAppBlur. Ova klasa predstavlja glavnu komponentu GUI aplikacije za pronalazak i zamućivanje lica. Sadrži sve potrebne atribute i inicijalne postavke za rad korisničkog interfejsa i video obrade.

```
class FaceBlurApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Softver za zamućivanje lica")
        self.root.geometry("900x700")

        self.video_path = None
        self.detection_method = tk.StringVar(value="haar")
        self.blur_method = tk.StringVar(value="gaussian")

        self.recording = False
        self.video_writer = None
        self.temp_recording_path = "temp_recording.mp4"

        self.capture_fps = 7.5
        self.display_fps = 60.0

        self.cap = None

        self.create_main_screen()
```

Kod 14. Implementacija klase FaceAppBlur

Objašnjenje klase:

self.root

- Referenca na glavni Tkinter prozor. Postavlja se naslov prozora i njegove dimenzije (900x700 piksela).

self.video_path

- Čuva putanju do odabrane video datoteke koje korisnik želi da obradi.

self.detection_method i self.blur_method

- Dvije Tkinter varijable koje čuvaju trenutno odabranu metodu pronalaska lica (haar ili dnn) i metodu zamućenja (gaussian, median, pixelation).
- Ove vrijednosti se kasnije koriste u menijima (radio buttons ili dropdown) i prosljeđuju se funkcijama obrade videozapisa.

self.recording i self.video_writer

- self.recording: boolean varijabla koja pokazuje da je snimanje aktivno.
- self.video_writer: OpenCV objekt za pisanje video datoteke, koristi se kada korisnik snima uživo kameru.

self.temp_recording_path, self.capture_fps, self.display_fps

- temp_recording_path: naziv privremene datoteke u koji se snimaju video kadrovi prije konačnog izvoza.
- capture_fps: brzina kojom se kadrovi hvataju tokom snimanja (radi optimizacije performansi).
- display_fps: učestalost osvježavanja prikaza uživo u GUI prozoru.

self.cap

- OpenCV VideoCapture objekt – koristi se za pristup kameri ili video datoteci.

self.create_main_screen()

- Poziva se funkcija koja inicijalno kreira grafičke komponente glavnog ekrana aplikacije: dugmad, padajuće liste, prikaz videozapisa, itd.

Zaključak:

Zbog toga, konstruktor klase FaceBlurApp postavlja sve ključne atribute koji su potrebni za rad aplikacije, uključujući:

- GUI konfiguraciju,
- početne metode obrade,
- podešavanja za video snimanje i prikaz.

Ova struktura omogućava modularan i lako proširiv dizajn aplikacije.

Nakon dodavanja klase idemo na implementaciju metode clear_screen(self). Ova metoda se koristi za čišćenje svih trenutnih grafičkih elemenata sa Tkinter GUI prozora. Koristi se kada je potrebno preći na novi ekran ili resetovati sadržaj prozora.

```
def clear_screen(self):
    for widget in self.root.winfo_children():
        widget.destroy()
```

Kod 15. Implementacija metode clear_screen(self)

Objašnjenje metode:
self.root.winfo_children()

- Metoda winfo_children() vraća listu svih widget-a (dugmad, okviri, oznake itd.) koji su trenutno djeca glavnog Tkinter prozora (self.root).

widget.destroy()

- Svakom od tih widget-a se poziva metoda destroy() kako bi bio uklonjen iz GUI-a.

Zaključak:
Na taj način se osigurava da je ekran potpuno "čist" prije nego što se doda novi sadržaj (npr. kada korisnik pređe sa početnog menija na ekran za pregled videozapisa). Ova metoda se koristi za dinamičko upravljanje prikazom u GUI aplikaciji – omogućava prelaz između različitih "ekrana" unutar iste Tkinter aplikacije, bez potrebe za otvaranjem novih prozora.

Zatim prelazimo na implementaciju metode stop_camera_feed(self). Ova metoda služi za zaustavljanje video prenosa s kamere i oslobađanje svih resursa povezanih s video snimanjem. Posebno je korisna kada korisnik prekine snimanje uživo ili zatvori aplikaciju.

```
def stop_camera_feed(self):
    self.recording = False
    if self.video_writer:
        self.video_writer.release()
        self.video_writer = None

    if self.cap:
        if self.cap.isOpened():
            self.cap.release()
        self.cap = None
```

Kod 16. Implementacija metode za stop_camera_feed(self)

Objašnjenje metode:

self.recording = False

- Zaustavlja indikaciju da se trenutno vrši snimanje. Ova vrijednost se koristi u petlji za ažuriranje snimka.

if self.video_writer:

- Provjerava da li objekt za zapisivanje videozapisa (cv2.VideoWriter) postoji i aktivan je.

self.video_writer.release()

- Oslobađa objekt za zapisivanje – zatvara datoteku u koji se upisivao video.

self.video_writer = None

- Resetira referencu na objekt kako bi se izbjegla greška prilikom budućih snimanja.

if self.cap:

- Provjerava da li postoji VideoCapture objekt koji predstavlja konekciju s kamerom.

if self.cap.isOpened(): self.cap.release()

- Ako je kamera otvorena, zatvara je i oslobađa povezane resurse.

self.cap = None

- Briše referencu na objekt kamere, osiguravajući da ne ostane neiskorišten memorijski resurs.

Zaključak:

Stoga, ova metoda osigurava da se svi resursi vezani za snimanje i kameru pravilno zatvore nakon korištenja, čime se izbjegavaju greške poput: Zaključane kamere, nemogućnosti ponovnog pokretanja snimanja i gubitka memorije

Pravilno upravljanje resursima u aplikacijama koje koriste kameru je od suštinskog značaja za stabilnost i pouzdanost softverskog rješenja.

Nakon dodavanja metode za zaustavljanje videoprenosa, idemo na Metodu create_main_screen(self). Ova metoda je odgovorna za prikaz glavnog ekrana aplikacije. Ona se koristi za inicijalni prikaz korisničkog interfejsa kada se aplikacija pokrene ili kada korisnik želi da se vrati na početni meni. Metoda kombinuje funkcionalnosti čišćenja prethodnog prikaza, kreiranja naslova i dugmadi za izbor daljih koraka.

```
def create_main_screen(self):
    self.stop_camera_feed()
    self.clear_screen()

    title = tk.Label(self.root, text="Softver za zamućivanje lica", font=("Helvetica", 24))
    title.pack(pady=50)

    video_btn = tk.Button(self.root, text="Obrada videozapisa", command=self.video_processing_screen, width=25, height=2)
    video_btn.pack(pady=20)

    realtime_btn = tk.Button(self.root, text="Real time obrada", command=self.realtime_processing_screen, width=25, height=2)
    realtime_btn.pack(pady=20)

    exit_btn = tk.Button(self.root, text="Exit", command=self.root.quit)
    exit_btn.place(relx=1.0, rely=1.0, anchor='se', x=-10, y=-10)
```

Kod 17. Implementacija metode za glavni prikaz aplikacije

Objašnjenje metode:

self.stop_camera_feed() i self.clear_screen()

- Ove dvije metode se pozivaju kako bi se osiguralo da se prethodna aktivnost zaustavi (npr. Kamera u stvarnom vremenu ili neki drugi prikaz), te da se svi elementi uklone iz prozora aplikacije kako bi se kreirao novi sadržaj.

title = tk.Label

- Kreira se Label widget koji prikazuje naziv softvera na vrhu ekrana. Koristi se veći font za vizuelni akcenat, a dodatni razmak (pady=50) poboljšava raspored.

video_btn

- Kreira dugme koje korisnika vodi na ekran za obradu prethodno snimljenog videozapisa. Pritiskom na dugme poziva se metoda video_processing_screen().

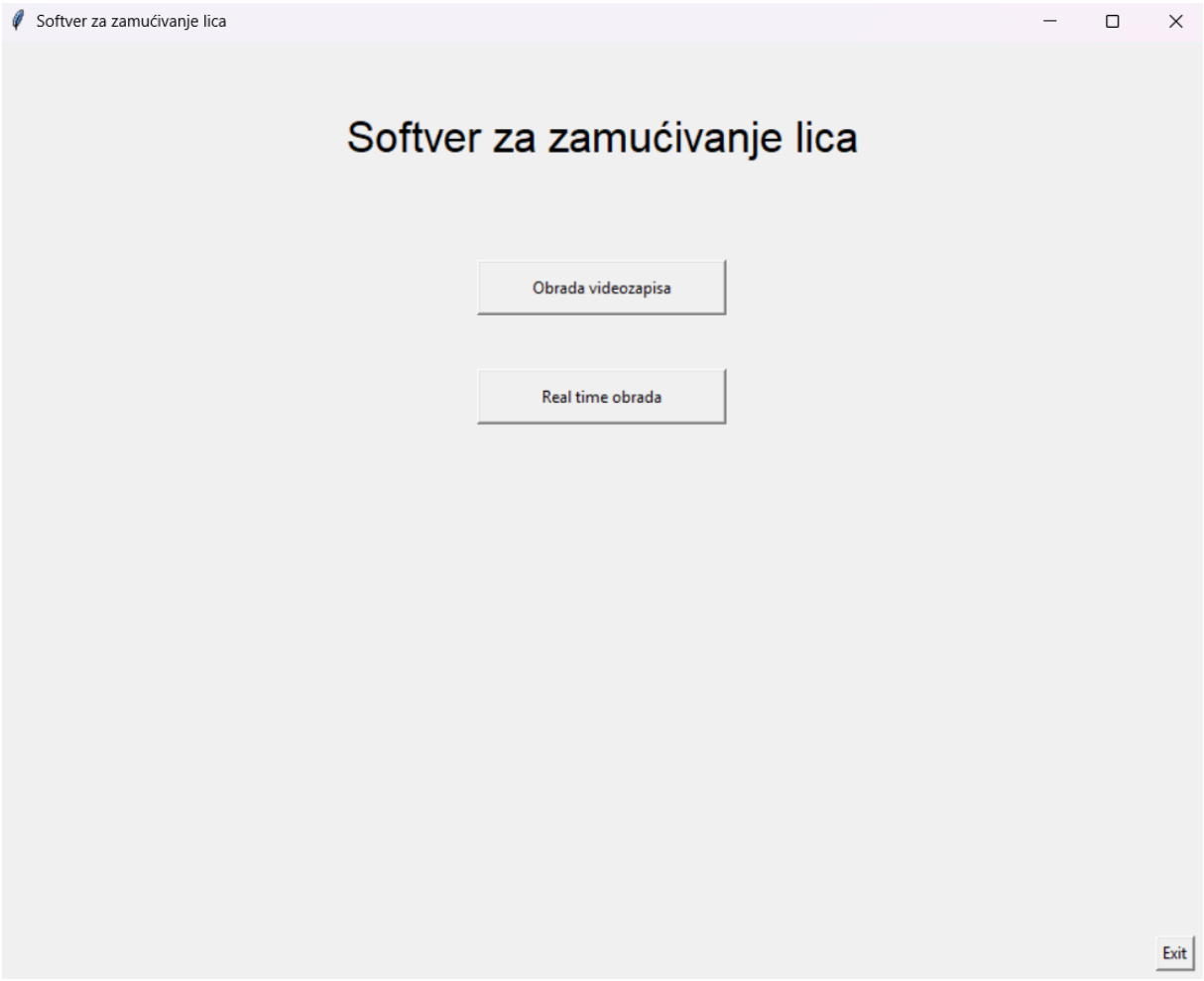
realtime_btn

- Kreira dugme koje omogućava korisniku da pokrene kameru i izvrši obradu lica u stvarnom vremenu. Aktivira metodu realtime_processing_screen().

exit_btn

- Dugme za zatvaranje aplikacije postavljeno je u donji desni ugao prozora pomoću place() metode. Parametri relx=1.0, rely=1.0, anchor='se' i pomaci x=-10, y=-10 osiguravaju preciznu poziciju dugmeta uz ivicu prozora.

Metoda create_main_screen omogućava jednostavno i intuitivno upravljanje aplikacijom kreiranjem centralnog interfejsa s ključnim opcijama za korisnika. Ova metoda igra važnu ulogu u korisničkom iskustvu jer definiše ulaznu tačku iz koje korisnik bira naredne korake, obradu snimljenog videozapisa ili obradu u stvarnom vremenu.



Slika 18. Prikaz glavnog ekrana aplikacije

Aplikacija je vrlo jednostavna za korištenje kao što se vidi iz priloženog. Stoga, kada korisnik pokrene aplikaciju, dočeka ga prvo glavni ekran aplikacije. Korisnik može birati koji način obrade zamučivanja želi, kao rezultat toga, „Obrada videozapisa“ služi za obradu tj.

zamućivanje već postojeće video datoteke, a Real time obrada služi kao što sam naslov kaže obradu u stvarnom vremenu koristeći web kameru uređaja.

Nakon implementacije glavne stranice aplikacije, slijedi Metoda video_processing_screen (self). Ova metoda definiše grafički korisnički interfejs (GUI) za rad sa snimljenim videozapisima, omogućavajući korisniku da odabere metode pronalaska lica i zamućivanja, učita video datoteku, te započne proces obrade. Ključna je komponenta aplikacije kada korisnik želi da izvrši obradu lica na postojećem videozapisu.

```
def video_processing_screen(self):
    self.stop_camera_feed()
    self.clear_screen()

    tk.Label(self.root, text="Metoda detekcije lica:", font=("Helvetica", 12)).pack(pady=5)
    frame1 = tk.Frame(self.root)
    frame1.pack(pady=5)
    tk.Radiobutton(frame1, text="Haar", variable=self.detection_method, value="haar").pack(side=tk.LEFT, padx=10)
    tk.Radiobutton(frame1, text="DNN", variable=self.detection_method, value="dnn").pack(side=tk.LEFT, padx=10)

    tk.Label(self.root, text="Metoda zamućivanja:", font=("Helvetica", 12)).pack(pady=5)
    frame2 = tk.Frame(self.root)
    frame2.pack(pady=5)
    tk.Radiobutton(frame2, text="Gaussian blur", variable=self.blur_method, value="gaussian").pack(side=tk.LEFT, padx=10)
    tk.Radiobutton(frame2, text="Median blur", variable=self.blur_method, value="median").pack(side=tk.LEFT, padx=10)
    tk.Radiobutton(frame2, text="Pixelation", variable=self.blur_method, value="pixelation").pack(side=tk.LEFT, padx=10)

    file_btn = tk.Button(self.root, text="Otvori videozapis", command=self.load_video_file)
    file_btn.pack(pady=10)

    self.video_label = tk.Label(self.root)
    self.video_label.pack(pady=10)

    self.process_btn = tk.Button(self.root, text="Obradi videozapis", command=self.process_video_file)
    self.process_btn.pack(pady=10)
    self.process_btn.config(state="disabled")

    tk.Button(self.root, text="Nazad", command=self.create_main_screen).place(relx=0.0, rely=1.0, anchor='sw', x=10, y=-10)
    tk.Button(self.root, text="Exit", command=self.root.quit).place(relx=1.0, rely=1.0, anchor='se', x=-10, y=-10)
```

Kod 19. Implementacija ekrana za obradu videozapisa

Objašnjenje metode:

self.stop_camera_feed() i self.clear_screen()

- Zaustavlja prethodne procese i briše sve prethodne elemente s ekrana, čime se priprema novi prikaz.

tk.Label (Metoda prepoznavanja lica)

- Prikazuje natpis za odabir metode, a zatim se prikazuju dvije opcije:

tk.Radiobutton

- Haar koristi tradicionalni Haar Cascade algoritam, dok DNN koristi duboke neuronske mreže (Deep Neural Networks). Korišten je Radiobutton kako bi korisnik mogao odabrati samo jednu opciju.

tk.Label (Metoda zamućivanja)

- Prikazuje izbor između tri metode: Gaussian blur – koristi Gaussian filter za zamućenje, median blur – koristi srednju vrijednost piksela za zamućenje i pixelation – koristi smanjenje rezolucije za efekat "pikselizacije".

file_btn = tk.Button

- Dugme omogućava korisniku da otvori datoteku sa lokalnog diska putem dijaloga.

self.video_label = tk.Label(self.root) i self.video_label.pack(pady=10)
Label u koji se prikazuje pregled videozapisa nakon učitavanja.
self.process_btn

- Dugme za pokretanje obrade koje je inicijalno deaktivirano i postaje aktivno tek nakon što korisnik učita video.

tk.Button (Nazad i Exit)

- Dugme „Nazad" vraća korisnika na glavni meni i dugme „Exit" zatvara aplikaciju.

Zaključak
Stoga, metoda video_processing_screen omogućava korisniku da kroz jasan i funkcionalan grafički interfejs odabere algoritme i metode obrade, učita videozapis i započne obradu. Time se korisničko iskustvo podiže na viši nivo jer korisnik ima kontrolu nad ključnim parametrima aplikacije.

Nakon implementacije ekrana za obradu videozapisa, prelazimo na Metodu load_video_file(self). Ova metoda omogućava korisniku da izabere i učita videozapis iz datoteka sistema putem dijaloga. Nakon uspješnog učitavanja, omogućava dugme za obradu i pokreće prikaz videozapisa u aplikaciji.

```
def load_video_file(self):
    path = filedialog.askopenfilename(filetypes=[("Video files", "*.mp4 *.avi")])
    if path:
        self.video_path = path
        self.process_btn.config(state="normal")
        self.play_video_preview(path)
```

Kod 20. Implementacija učitavanja video datoteke iz sistema

Objašnjenje metode:

path = filedialog.askopenfilename

- Otvara dijalog za biranje datoteke putem standardnog Tkinter filedialog modula. Korisniku su ponuđene samo video datoteke s ekstenzijama .mp4 i .avi.

if path:, self.video_path = path

- Provjerava da li je korisnik odabrao validnu datoteku, ako jeste, čuva putanju do datoteke u varijablu self.video_path, koja se koristi kasnije tokom obrade.

self.process_btn.config(state="normal")

- Aktivira dugme za obradu videozapisa (self.process_btn) koje je prethodno bilo onemogućeno. Time se korisniku omogućava prelazak na sljedeći korak – pokretanje obrade videozapisa.

self.play_video_preview(path)

- Poziva pomoćnu metodu za prikazivanje pregleda videozapisa unutar GUI-ja. Na taj način korisnik može odmah vidjeti koji video je odabran, što poboljšava upotrebljivost i preglednost.

Zaključak:

Metoda load_video_file predstavlja ključni korak u interakciji korisnika sa softverom, omogućujući učitavanje videozapisa koji će se obraditi. Nakon učitavanja, aplikacija automatski omogućava obradu i prikazuje pregled videozapisa, čime se postiže intuitivan i efikasan tok rada.

Nakon metode load_video_file, uključujemo metodu play_video_preview(self, path) i funkciju update (). U metodi play_video_preview, koristi se ugnježđena funkcija update koja služi za prikazivanje videozapisa u Tkinter GUI prozoru. Funkcija update se rekurzivno poziva pomoću Tkinter-ove metode after(), čime omogućava glatki prikaz svakog kadra videozapisa u labeli. Funkcija je definirana lokalno unutar metode i nije dostupna van njenog konteksta. Ova metoda služi za prikazivanje kratkog video pregleda u GUI interfejsu aplikacije nakon što korisnik odabere videozapis. Pregled omogućava korisniku da potvrdi da je učitana ispravna datoteka prije pokretanja obrade. Funkcija update() koristi se za dinamički prikaz videozapisa u grafičkom korisničkom interfejsu (GUI) aplikacije. Omogućava kontinuirano čitanje kadrova iz učitanog videozapisa i njihovo prikazivanje u Tkinter prozoru, čime se korisniku nudi

realističan pregled video sadržaja.

```
def play_video_preview(self, path):
    cap = cv2.VideoCapture(path)

    def update():
        if not cap.isOpened():
            return
        ret, frame = cap.read()
        if ret:
            frame = cv2.resize(frame, (600, 400))
            rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(rgb)
            imgtk = ImageTk.PhotoImage(image=img)
            if hasattr(self, 'video_label') and self.video_label.winfo_exists():
                self.video_label.imgtk = imgtk
                self.video_label.configure(image=imgtk)
                self.root.after(30, update)
        else:
            cap.release()
    update()
```

Kod 21. Implementacija metode i funkcije za prikaz odabrane datoteke

Objašnjenje metode:

cap = cv2.VideoCapture(path)

- Otvara videozapis sa zadate putanje pomoću cv2.VideoCapture. Kreira objekat cap koji omogućava čitanje kadrova iz video datoteke.

Objašnjenje funkcije:

if not cap.isOpened(): i return

- Provjerava da li je video datoteka uspješno otvorena. Ako nije, funkcija odmah završava izvršavanje.

ret, frame = cap.read()

- Pokušava pročitati jedan kadar iz video datoteke, dok ret označava da li je čitanje bilo uspješno, a frame sadrži video kadar.

if ret:

- Ako je kadar uspješno učitan, veličina slike se prilagođava (600x400 piksela) radi boljeg prikaza u GUI prozoru. Boje se konvertuju iz BGR (OpenCV format) u RGB (format koji koristi Tkinter). Slika se pretvara u format koji može prikazati Tkinter koristeći PIL.Image i ImageTk.PhotoImage.

if hasattr

- Provjerava da li video_label postoji i da li je još uvijek aktivan u GUI-ju. Ažurira sadržaj video_label sa novom slikom (imgtk). Postavlja after metodu da se funkcija update() ponovo pozove za 30 milisekundi, čime se kreira efekat "video prikaza u stvarnom vremenu".

else: cap.release()

- Ako nije moguće pročitati više kadrova (kraj videozapisa), oslobađa se resurs cap.

Zaključak:
Funkcija update() omogućava neprekidno učitavanje i prikazivanje video kadrova u Tkinter GUI-ju, simulirajući pregled videozapisa u stvarnom vremenu. Njena rekurzivna priroda pomoću after() metode omogućava nesmetan prikaz bez blokiranja glavnog GUI procesa.

Na samom kraju ekrana za obradu videozapisa uključujemo metode process_video_file i _run_processing_thread.

```
def process_video_file(self):
    if not self.video_path:
        messagebox.showwarning("Upozorenje", "Niste odabrali video fajl.")
        return
    save_path = filedialog.asksaveasfilename(defaultextension=".mp4", filetypes=[("MP4 files", "*.mp4")])
    if save_path:
        messagebox.showinfo("Obrada u toku", "Molimo sačekajte dok se video obrađuje.")
        threading.Thread(
            target=self._run_processing_thread,
            args=(self.video_path, save_path, self.detection_method.get(), self.blur_method.get()),
            daemon=True
        ).start()

def _run_processing_thread(self, video_path, output_path, detection_method, blur_method):
    try:
        process_video_file(video_path, output_path, detection_method, blur_method)
        messagebox.showinfo("Uspjeh", "Video uspješno obrađen i sačuvan!")
    except Exception as e:
        messagebox.showerror("Greška", f"Došlo je do greške: {str(e)}")
```

Kod 22. Implementacija metoda process_video_file i _run_processing_thread

Metoda process_video_file(self) pokreće proces obrade videozapisa koji je korisnik prethodno odabrao putem grafičkog interfejsa aplikacije. U slučaju da korisnik nije odabrao video, metoda će prikazati upozorenje koristeći Tkinter-ovu funkciju messagebox.showwarning.
Ako korisnik odredi putanju za spremanje obrađenog videozapisa, korisniku se prikazuje poruka da obrada traje. Obrada se zatim pokreće u posebnoj niti koristeći threading.Thread kako

bi se izbjeglo blokiranje glavnog GUI interfejsa tokom procesiranja. Nit poziva internu pomoćnu metodu _run_processing_thread.

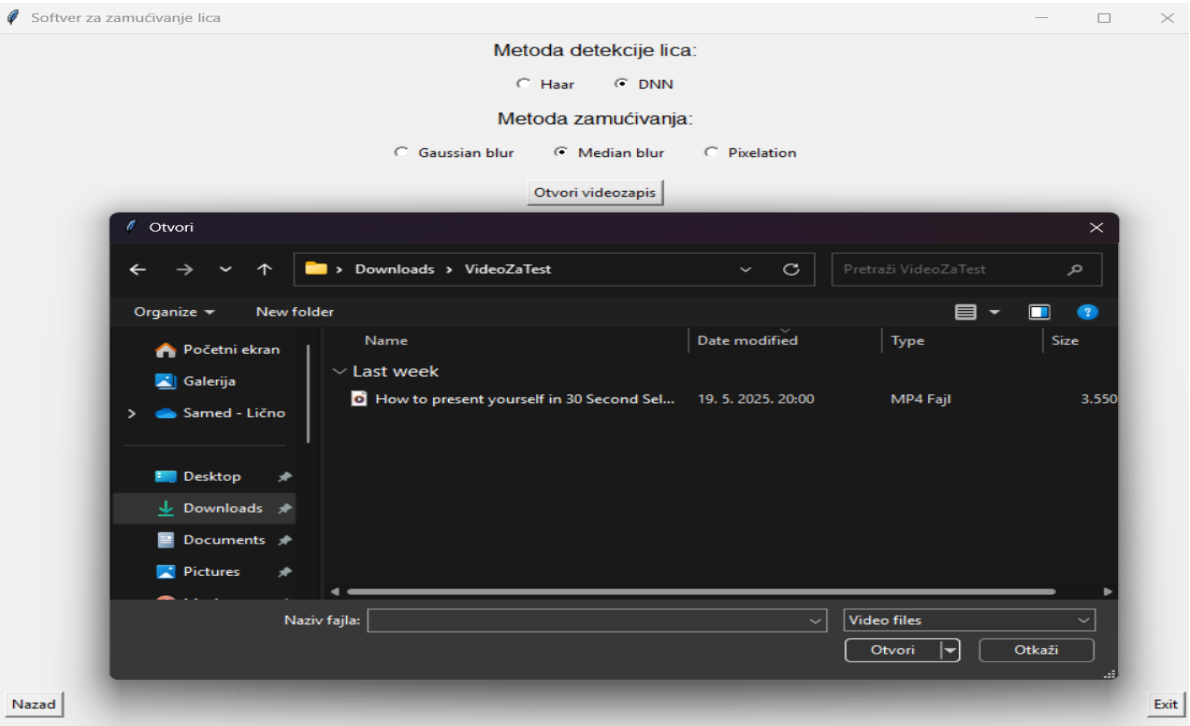
Zaključak:
Metoda _run_processing_thread(self, video_path, output_path, detection_method, blur_method) izvršava stvarnu logiku obrade videozapisa. Ova metoda koristi funkciju process_video_file iz video_processor modula, koja provodi prepoznavanje lica i zamućivanje kadrova na osnovu odabranih metoda.
U slučaju da je obrada uspješno završena, korisniku se prikazuje poruka o uspjehu putem messagebox.showinfo. Ako dođe do greške tokom procesiranja, korisnik je obaviješten putem messagebox.showerror sa detaljima greške.

Nakon dodavanja metoda process_video_file i _run_processing_thread sada ćemo prikazati kako aplikacija radi u praksi.

Kada korisnik odabere obradu videozapisa, dočeka ga ovaj ekran. Stoga, iz priloženog vidimo kako je korisniku data mogućnost da sam odabere metodu prepoznavanja lica, metodu zamućivanja, mogućnost biranja videozapisa sa uređaja, te da jednim klikom obradi cijeli videozapis. Po zadanim postavkama je odabrana Haar metoda prepoznavanja i Gaussian blur metoda.

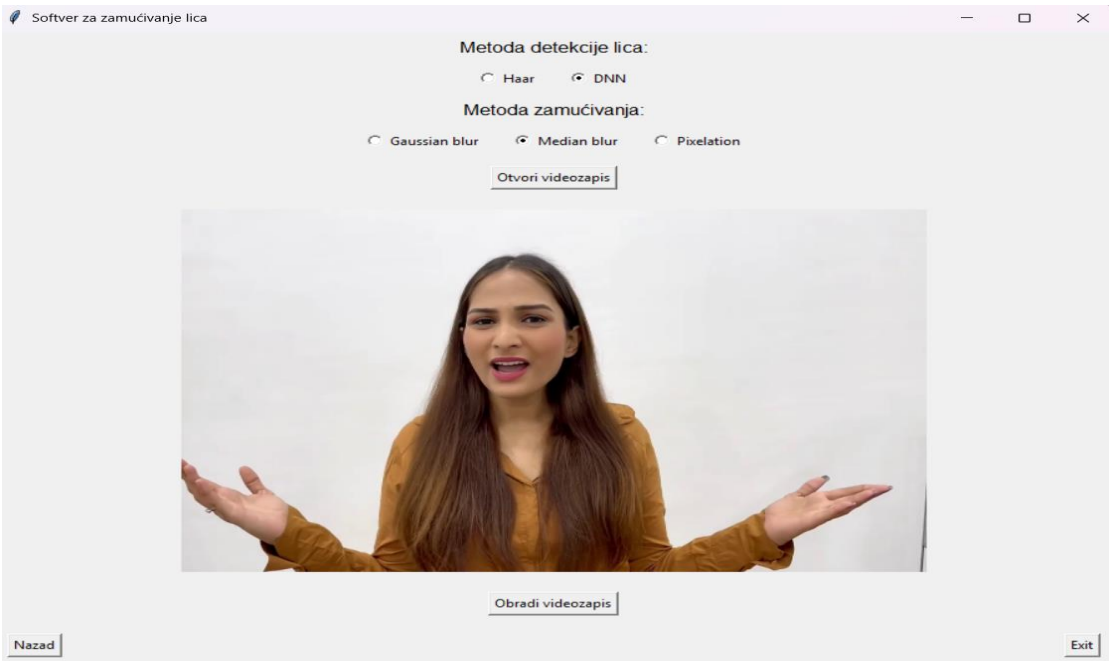
Slika 23. Prikaz ekrana obrada videozapisa

Kada korisnik odabere metodu prepoznavanja lica i metodu zamućivanja lica (za test smo odabrali DNN metodu prepoznavanja lica i median blur metodu zamućivanja) korisnik može da učita datoteku sa uređaja. Korisnik može da unese .avi ili .mp4 format.



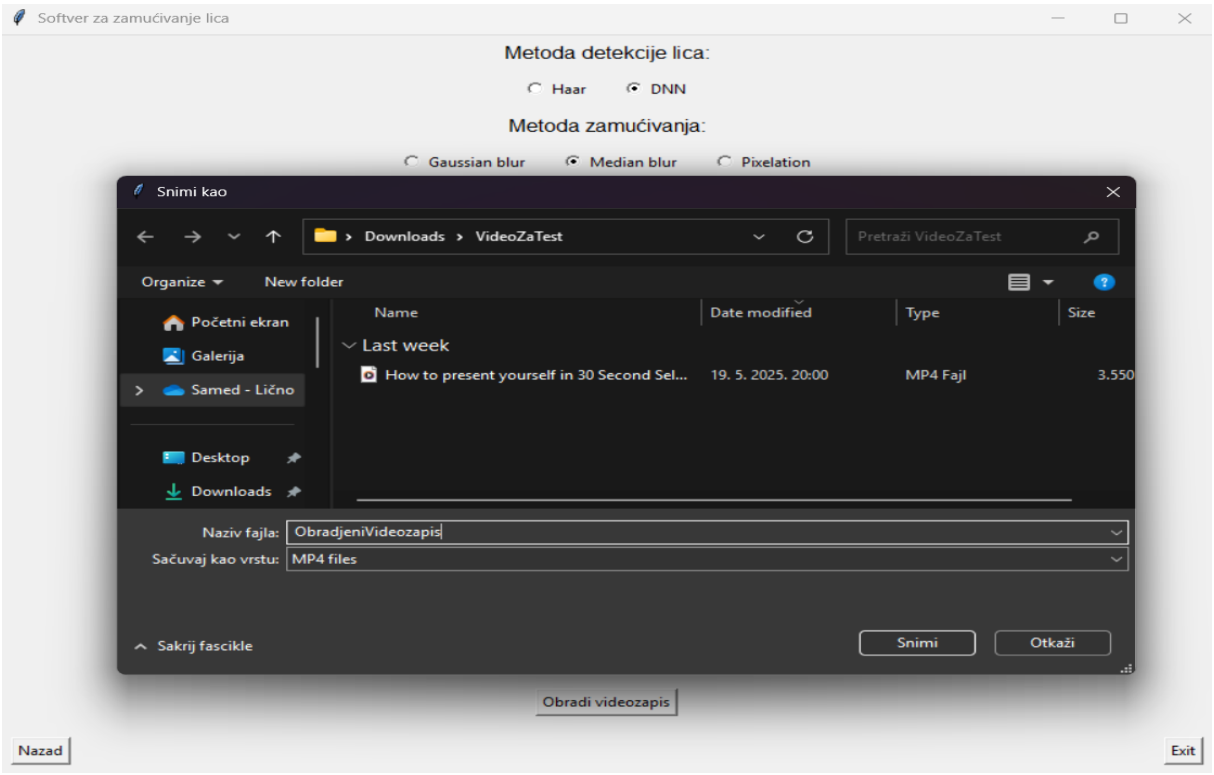
Slika 24. Prikaz odabirane video datoteke

Kada korisnik odabere datoteku, videozapis kreće sa prikazivanjem u stvarnom vremenu.

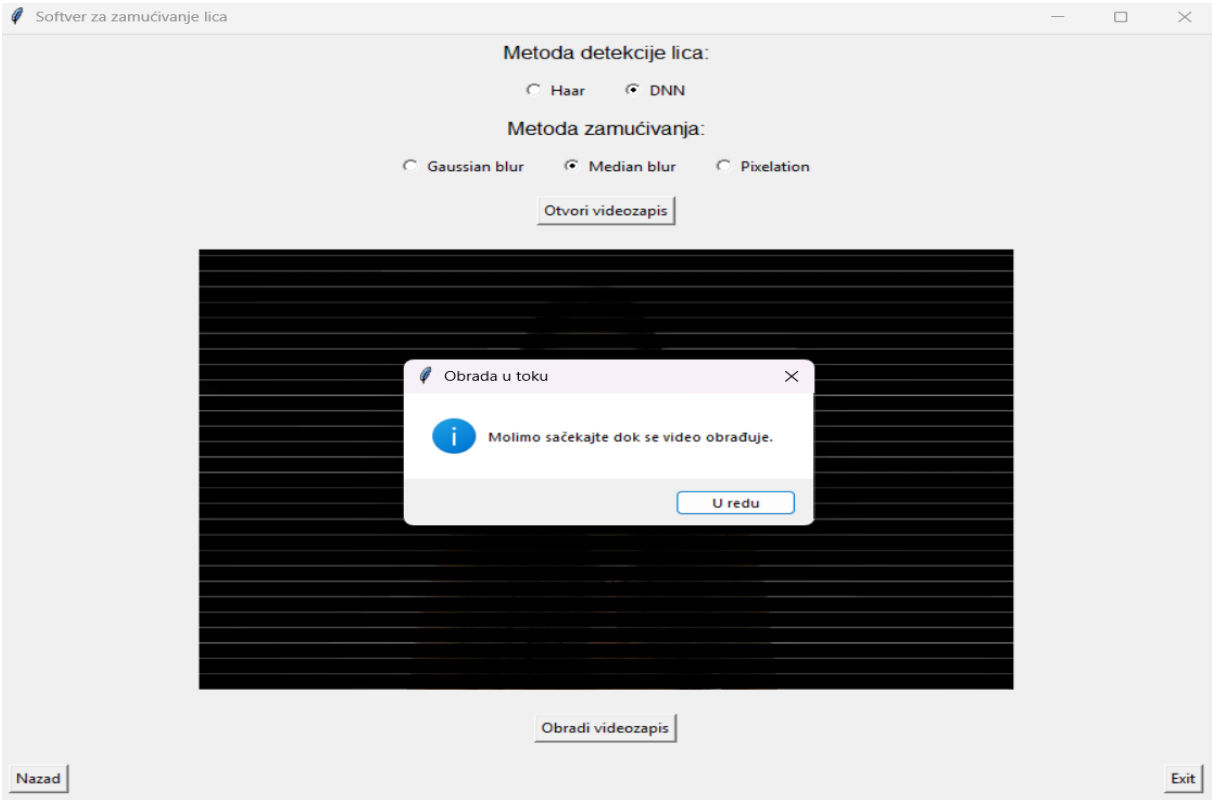


Slika 25. Prikaz odabranog videozapisa

Kada korisnik klikne na „Obradi videozapis“, korisniku izađe prozor da odabere gdje želi da sačuva obrađeni videozapis i kako želi da ga nazove.

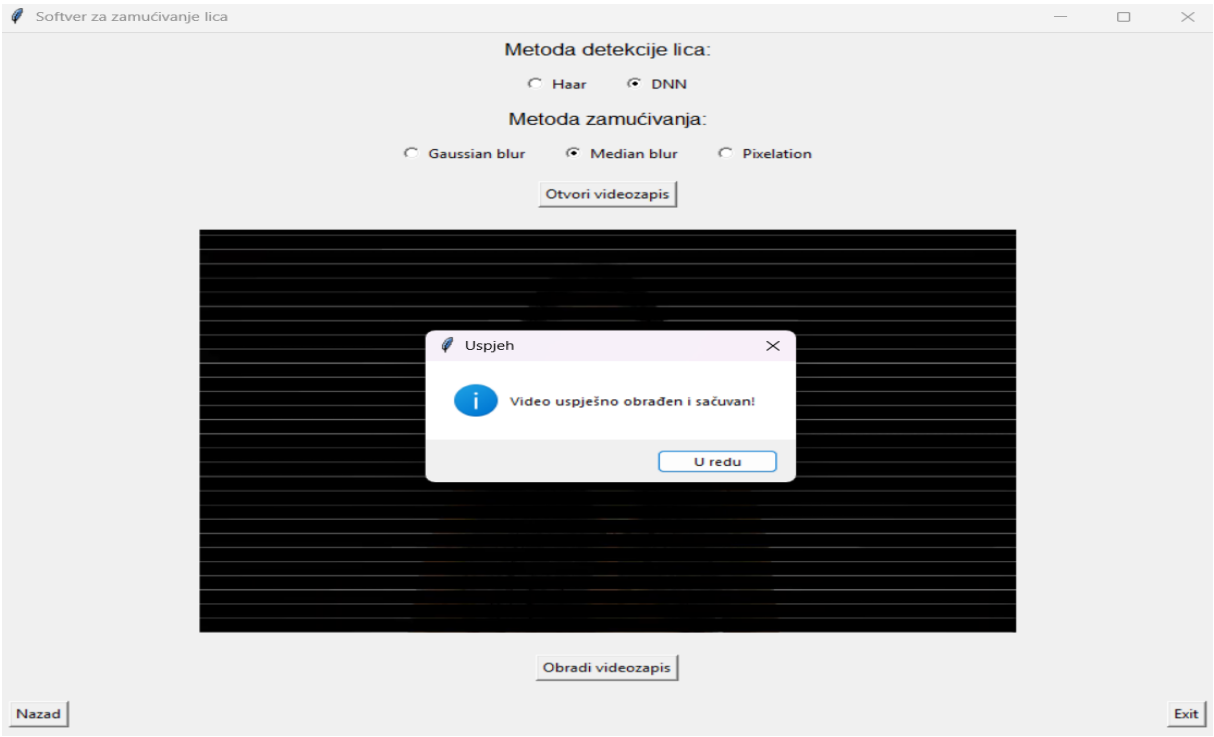


Slika 26. Prikaz spremanja obrađenog videozapisa



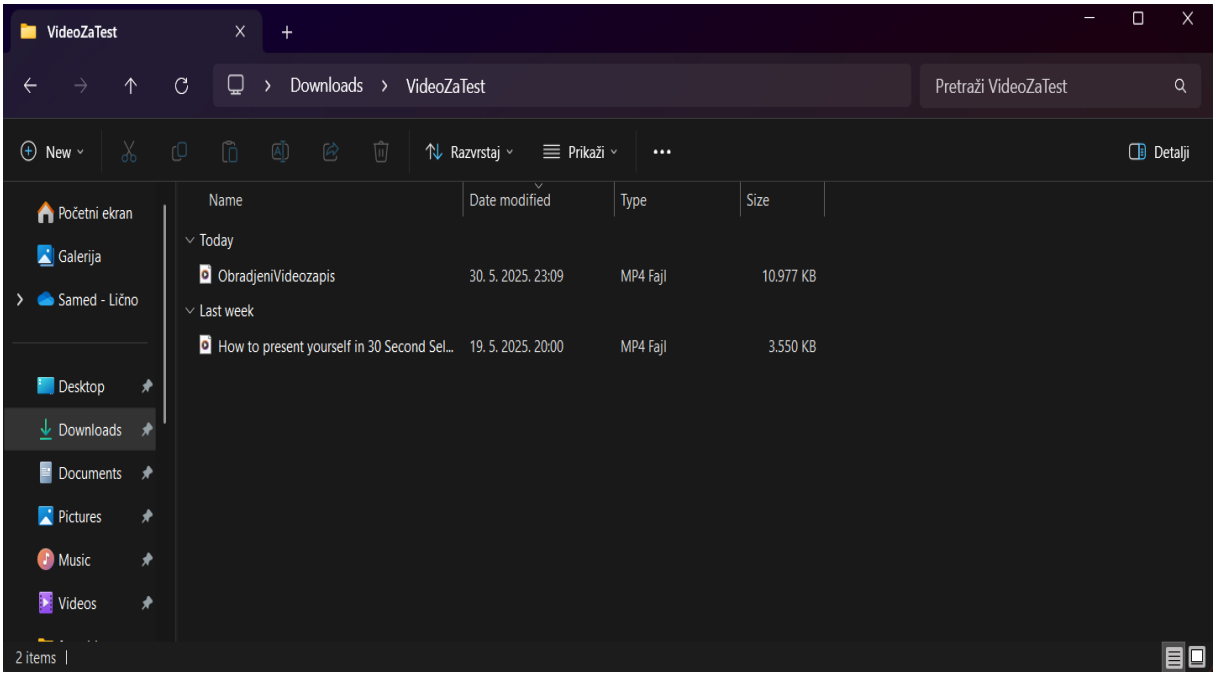
Slika 27. Prikaz ispisane poruke da se video obrađuje

Nakon što korisnik sačeka par sekundi, video je uspješno obrađen. Također, korisniku se prikaže poruka „Video uspješno obrađen i sačuvan!“

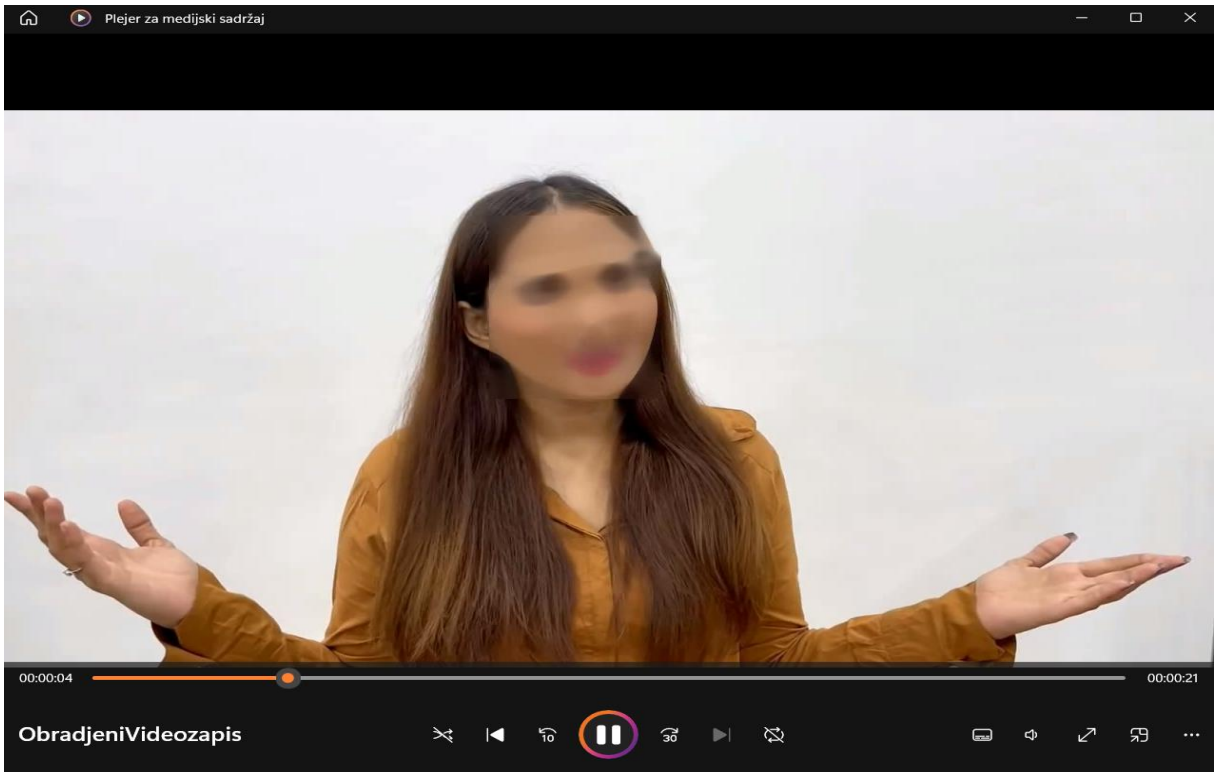


Slika 28. Prikaz ispisane poruke za uspješno obrađen video

Kada odemo u folder gdje smo spremili obrađen videozapis, dočeka nas uspješno spremljen i obrađen videozapis.



Slika 29. Prikaz sačuvanog i obrađenog videozapisa



Slika 30 Prikaz samog rezultata i uspješne obrade videozapisa

Video je spreman za upotrebu. Uspješno obrađen u skladu s prethodno odabranim metodama u aplikaciji metodu prepoznavanja i metodu zamućivanja, te je video sačuvan za dalje korištenje i upotrebljavanje.

Nakon implementiranja kompletnog koda za obradu videozapisa, prelazimo na implementaciju koda za obradu u stvarnom vremenu, tj. Real time obrada. Sada implementiramo metodu realtime_processing_screen(self). Ova metoda omogućava korisniku da koristi kameru u stvarnom vremenu, bira metodu za prepoznavanje lica i metodu zamućivanja, te da snimi i spremi video sa zamagljenim licima. Ovo je ključna funkcionalnost aplikacije kada je u pitanju live stream obrada slike direktno s kamere.

```python
def realtime_processing_screen(self):
    self.stop_camera_feed()
    self.clear_screen()

    tk.Label(self.root, text="Metoda detekcije lica:", font=("Helvetica", 12)).pack(pady=5)
    frame1 = tk.Frame(self.root)
    frame1.pack(pady=5)
    tk.Radiobutton(frame1, text="Haar", variable=self.detection_method, value="haar").pack(side=tk.LEFT, padx=10)
    tk.Radiobutton(frame1, text="DNN", variable=self.detection_method, value="dnn").pack(side=tk.LEFT, padx=10)

    tk.Label(self.root, text="Metoda zamućivanja:", font=("Helvetica", 12)).pack(pady=5)
    frame2 = tk.Frame(self.root)
    frame2.pack(pady=5)
    tk.Radiobutton(frame2, text="Gaussian blur", variable=self.blur_method, value="gaussian").pack(side=tk.LEFT, padx=10)
    tk.Radiobutton(frame2, text="Median blur", variable=self.blur_method, value="median").pack(side=tk.LEFT, padx=10)
    tk.Radiobutton(frame2, text="Pixelation", variable=self.blur_method, value="pixelation").pack(side=tk.LEFT, padx=10)

    tk.Button(self.root, text="Pokreni kameru", command=self.start_camera_feed).pack(pady=10)

    self.video_label = tk.Label(self.root)
    self.video_label.pack(pady=10)

    self.record_btn = tk.Button(self.root, text="Snimi videozapis", command=self.start_recording, state="disabled")
    self.record_btn.pack(pady=5)

    self.stop_btn = tk.Button(self.root, text="Zaustavi snimanje", command=self.stop_recording, state="disabled")
    self.stop_btn.pack(pady=5)

    self.save_btn = tk.Button(self.root, text="Spremi videozapis", command=self.save_recording, state="disabled")
    self.save_btn.pack(pady=5)

    tk.Button(self.root, text="Nazad", command=self.create_main_screen).place(relx=0.0, rely=1.0, anchor='sw', x=10, y=-10)
    tk.Button(self.root, text="Exit", command=self.root.quit).place(relx=1.0, rely=1.0, anchor='se', x=-10, y=-10)
```

Kod 31. Implementacija metode za real time obradu

Objašnjenje metode:

self.stop_camera_feed()

- Zaustavlja trenutno aktivni feed s kamere ako postoji, kako bi se izbjeglo pokretanje više instanci kamere.

self.clear_screen()

- Briše sve postojeće grafičke elemente sa ekrana (widgete) i priprema prozor za prikaz novih kontrola.

tk.Label(...) i tk.Radiobutton(...)

- Prikazuje opcije za izbor metode prepoznavanja lica. Korisnik može birati između Haar Cascade ("haar") i DNN ("dnn").
- Svaka grupa dugmadi je smještena unutar Frame widgeta kako bi raspored bio pregledniji.

tk.Label(...) i tk.Radiobutton(...) (drugi dio)

- Omogućava korisniku da izabere metodu zamućivanja: Gaussian blur, Median blur ili Pixelation. Ove metode će se primijeniti na detektovana lica u stvarnom vremenu.

tk.Button(..., command=self.start_camera_feed)

- Dugme za pokretanje kamere. Nakon što korisnik odabere metode, klikom na ovo dugme pokreće se prikaz kamere unutar aplikacije i vrši se prepoznavanje i zamućivanje lica u stvarnom vremenu.

self.video_label = tk.Label(self.root)

- Rezerviše prostor u GUI-ju gdje će se prikazivati video feed s kamere.

self.record_btn, self.stop_btn, self.save_btn

- Ova tri dugmeta služe za kontrolu snimanja: Snimi videozapis - pokreće snimanje videozapisa, zaustavi snimanje - završava snimanje i spremi videozapis omogućava spremanje snimljenog videozapisa u datoteke. Na početku su onemogućeni (state="disabled"), a postaju aktivni tokom i nakon pokretanja video snimanja.

tk.Button(..., command=self.create_main_screen)

- Dugme "Nazad" vraća korisnika na glavni meni aplikacije.

tk.Button(..., command=self.root.quit)

- Dugme "Exit" omogućava korisniku da zatvori aplikaciju.

Zaključak:
Metoda realtime_processing_screen predstavlja centralno mjesto za upravljanje funkcijama obrade videozapisa u stvarnom vremenu. Korisniku omogućava visoku razinu kontrole nad metodama prepoznavanja i zamućivanja, a dodatno nudi funkcionalnosti snimanja i spremanja videozapisa sa zamućenim licima. Intuitivan raspored kontrole i logičan tok koraka omogućavaju jednostavno korištenje čak i za korisnike bez tehničkog predznanja.

Nakon postavljanja opcija za prepoznavanje i zamućivanje lica, korisnik klikom na „Pokreni kameru“ aktivira metodu start_camera_feed(self). Ova metoda inicijalizuje kameru, provjerava njenu dostupnost i započinje prikaz slike u stvarnom vremenu.

```
def start_camera_feed(self):
    if self.cap and self.cap.isOpened():
        return
    self.cap = cv2.VideoCapture(0)
    if not self.cap.isOpened():
        messagebox.showerror("Greška", "Ne mogu da otvorim kameru.")
        self.cap = None
        return
    self.record_btn.config(state="normal")
    self.recording = False
    self.last_write_time = 0
    self.update_camera_feed()
```

Kod 32. Implementacija metode za pokretanje kamere

Objašnjenje metode:

if self.cap and self.cap.isOpened():

- Provjerava da li je kamera već aktivna i otvorena. Ako jeste, metoda se odmah prekida kako bi se spriječilo ponovno otvaranje već aktivne kamere.

self.cap = cv2.VideoCapture(0)

- Pokušava otvoriti primarnu kameru na uređaju (obično je to ugrađena web kamera ili prva dostupna USB kamera).

if not self.cap.isOpened():

- Ako otvaranje kamere ne uspije (npr. kamera je zauzeta ili ne postoji), prikazuje se poruka o grešci korisniku putem messagebox.showerror, i vrijednost self.cap se resetuje na None.

self.record_btn.config(state="normal")

- Omogućava dugme za snimanje videozapisa koje je inicijalno bilo onemogućeno. Time se korisniku pruža mogućnost da odmah započne snimanje kada poželi.

self.recording = False

- Postavlja status snimanja na False, snimanje još nije počelo. Ovo se koristi u logici aplikacije za upravljanje tokom snimanja.

self.last_write_time = 0

- Resetuje internu vremensku varijablu koja prati kada je posljednji kadar upisan tokom snimanja. Ovo je važno za kontrolu brzine snimanja videozapisa.

self.update_camera_feed()

- Poziva pomoćnu metodu koja pokreće kontinuirani prikaz slike s kamere u GUI prozoru. U toj metodi se također, vrši prepoznavanje lica i primjenjuje odabrana metoda zamućivanja u stvarnom vremenu.

Zaključak:
Metoda start_camera_feed je ključna za pokretanje live prikaza kamere u aplikaciji. Provjerava dostupnost kamere, inicijalizuje sve potrebne varijable i omogućava osnovne kontrole snimanja. Kroz nju počinje cijeli proces obrade u stvarnom vremenu zamućivanja lica, čineći je fundamentalnim korakom u toku rada sa kamerom.

Nakon implementacije metode za pokretanje kamere, prelazimo na metodu za automatsko mijenjanje metode zamućivanja lica ili metode prepoznavanja lica tj. update_camera_feed.

```
def update_camera_feed(self):
    if not (self.cap and self.cap.isOpened()):
        return
    if not hasattr(self, 'video_label') or not self.video_label.winfo_exists():
        return
    ret, frame = self.cap.read()
    if ret:
        faces = detect_faces(frame, self.detection_method.get())
        frame = apply_blur(frame, faces, self.blur_method.get())

        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(rgb)
        imgtk = ImageTk.PhotoImage(image=img)

        self.video_label.imgtk = imgtk
        self.video_label.configure(image=imgtk)

        if self.recording and self.video_writer:
            current_time = time.time()
            if (current_time - self.last_write_time) >= 1.0 / self.capture_fps:
                self.video_writer.write(frame)
                self.last_write_time = current_time

    if self.cap and self.cap.isOpened() and self.video_label.winfo_exists():
        self.root.after(int(1000 / self.display_fps), self.update_camera_feed)
```

Kod 33. Implemetacija metode za update_camera_feed

Objašnjenje metode:
if not (self.cap and self.cap.isOpened()): return

- Provjerava da li je kamera otvorena. Ako nije, metoda se prekida jer nema dostupnog video feeda za obradu.

if not hasattr(self, 'video_label') or not self.video_label.winfo_exists(): return

- Provjerava da li video_label postoji i da li je još uvijek aktivan u GUI-ju. Ako nije, zaustavlja dalju obradu kako bi se izbjegle greške pri prikazu slike.

ret, frame = self.cap.read()

- Čita jedan kadar s kamere. Ako je čitanje uspješno (ret=True), nastavlja se obrada slike.

faces = detect_faces(frame, self.detection_method.get())

- Na pročitani kadar primjenjuje se odabrana metoda prepoznavanja lica (haar ili dnn). Rezultat je lista koordinata detektovanih lica.

frame = apply_blur(frame, faces, self.blur_method.get())

- Na detektovana lica se primjenjuje odabrana metoda zamućivanja (gaussian, median, ili pixelation), čime se lica u kadaru zamućuju.

rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

- Pretvara kadar iz BGR (OpenCV standard) u RGB format koji je kompatibilan s Tkinter-ovim prikazom slike.

img = Image.fromarray(rgb)

- Konvertuje NumPy array (sliku) u PIL objekt slike.

imgtk = ImageTk.PhotoImage(image=img)

- Konvertuje PIL sliku u format koji Tkinter može prikazati putem Label widgeta.

self.video_label.imgtk = imgtk

- Čuva referencu na sliku kako bi se izbjeglo da Python automatski obriše sliku iz memorije.

self.video_label.configure(image=imgtk)

- Ažurira prikaz slike u GUI-ju. Korisnik sada vidi live video feed sa zamućenim licima.

if self.recording and self.video_writer:

- Ako je snimanje aktivno i video_writer je inicijalizovan, nastavlja se proces snimanja.

current_time = time.time()

- Dobiva trenutno vrijeme (u sekundama) kako bi se upravljalo vremenom između pisanja kadarova u video.

if (current_time - self.last_write_time) >= 1.0 / self.capture_fps:

- Provjerava je li prošlo dovoljno vremena od posljednjeg upisa, prema definisanom FPS-u snimanja (self.capture_fps).

self.video_writer.write(frame)

- Upisuje trenutni kadar u snimljeni video.

self.last_write_time = current_time

- Ažurira vrijeme posljednjeg snimanja kadara.

if self.cap and self.cap.isOpened() and self.video_label.winfo_exists():

- Na kraju se metoda sama ponovno poziva kroz self.root.after, čime se osigurava kontinuirano ažuriranje slike s kamere u definisanom intervalu (60 fps u GUI-ju).

Zaključak:
Metoda update_camera_feed je srž obrade u stvarnom vremenu video obrade u aplikaciji. Pruža prikaz slike s kamere u GUI-ju, detektuje lica, primjenjuje zamućivanje i omogućava snimanje. Efikasno koristi Tkinter-ovu funkciju after za neprekidno ažuriranje prikaza, čime se postiže glatko i responzivno korisničko iskustvo.

Nakon obrade prikaza kamere u stvarnom vremenu prelazimo na metodu start_recording(self). Ova metoda omogućava korisniku da započne snimanje videozapisa s kamere u aplikaciji. Video se snima lokalno u privremenu datoteku, uz kontrolu kvalitete i kadrova. Ključni je dio funkcionalnosti kada korisnik želi sačuvati zamućeni video feed za kasnije.

```
def start_recording(self):
    if not (self.cap and self.cap.isOpened()):
        messagebox.showwarning("Upozorenje", "Kamera nije pokrenuta.")
        return
    if self.recording:
        return

    fourcc = cv2.VideoWriter_fourcc(*"mp4v")
    width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))


    if os.path.exists(self.temp_recording_path):
        os.remove(self.temp_recording_path)

    self.video_writer = cv2.VideoWriter(self.temp_recording_path, fourcc, self.capture_fps, (width, height))
    self.recording = True

    self.record_btn.config(state="disabled")
    self.stop_btn.config(state="normal")
    self.save_btn.config(state="disabled")
```

Kod 34. Implementacija metode za početak snimanja u stvarnom vremenu pomocu web kamere

Objašnjenje metode:
if not (self.cap and self.cap.isOpened()):

- Provjerava da li je kamera aktivna. Ako nije, prikazuje se upozorenje korisniku pomoću messagebox.showwarning, a metoda se prekida.

if self.recording:

- Ako je snimanje već aktivno, metoda se odmah zaustavlja kako ne bi došlo do dvostrukog pokretanja.

fourcc = cv2.VideoWriter_fourcc(*"mp4v")

- Postavlja video codec za snimanje. "mp4v" je standardni codec za .mp4 videozapise.

width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

- Dohvata širinu i visinu kadarova iz trenutnog video feeda kako bi se postavila odgovarajuća veličina izlaznog videozapisa.

if os.path.exists(self.temp_recording_path): os.remove(self.temp_recording_path)

- Ako već postoji privremena datoteka aza snimanje (od prethodnih sesija), ona se briše kako bi se izbjegli konflikti i osigurala čista nova snimka.

self.video_writer = cv2.VideoWriter(...)

- Inicijalizuje VideoWriter objekt koji će upisivati kadrove u datoteku. Koristi prethodno definisane dimenzije, codec i FPS.

self.recording = True

- Postavlja indikator da je snimanje aktivno. Koristi se u drugim dijelovima aplikacije za upravljanje snimanjem (npr. u update_camera_feed).

self.record_btn.config(state="disabled")

- Onemogućava dugme "Snimi videozapis" kako korisnik ne bi mogao pokrenuti snimanje više puta istovremeno.

self.stop_btn.config(state="normal")

- Omogućava dugme "Zaustavi snimanje", čime korisniku daje kontrolu da prekine snimanje kada poželi.

self.save_btn.config(state="disabled")

- Onemogućava dugme "Spremi videozapis" dok snimanje nije završeno. Bit će omogućeno tek nakon zaustavljanja.

Zaključak:

Metoda start_recording postavlja sve potrebne parametre i inicijalizuje mehanizam za snimanje videozapisa iz kamere. Efikasno sprečava višestruko pokretanje snimanja, priprema datoteku sistem i omogućava korisniku jednostavnu kontrolu. Ova metoda igra ključnu ulogu u procesu snimanja i osigurava da snimljeni videozapis bude tehnički ispravan i spreman za kasnije spremanje.

Nakon pokretanja funkcionalnosti za snimanje videozapisa, prelazimo na metodu stop_recording(self). Ova metoda omogućava korisniku da zaustavi trenutno snimanje videozapisa koje je prethodno pokrenuto pomoću start_recording. Ovdje se zaustavlja pisanje kadrova u privremenu datoteku i omogućava se opcija za njegovo spremanje.

```
def stop_recording(self):
    if not self.recording:
        return
    self.recording = False
    if self.video_writer:
        self.video_writer.release()
        self.video_writer = None

    self.record_btn.config(state="normal")
    self.stop_btn.config(state="disabled")
    self.save_btn.config(state="normal")
```

Kod 35. Implementacija metode za stopiranje snimanja

Objašnjenje metode

if not self.recording:

- Provjerava da li je snimanje aktivno. Ako nije (npr. korisnik klikne "Zaustavi" bez da je pokrenuo snimanje), metoda se odmah prekida.

self.recording = False

- Postavlja indikator recording na False, čime se efektivno zaustavlja proces snimanja.

if self.video_writer:

- Ako je instanca video_writer aktivna, kadrovi zapisivali u datoteku:

self.video_writer.release()

- Zatvara datoteku za pisanje i finalizira videozapis.

self.video_writer = None

- Poništava objekt video_writer kako bi se očistio prostor i spriječilo ponovno korištenje.

self.record_btn.config(state="normal")

- Ponovno omogućava dugme "Snimi videozapis", čime korisniku daje mogućnost da pokrene novo snimanje ako želi.

self.stop_btn.config(state="disabled")

- Onemogućava dugme "Zaustavi snimanje", jer više nema aktivnog snimanja koje bi trebalo zaustaviti.

self.save_btn.config(state="normal")

- Omogućava dugme "Spremi videozapis", što korisniku daje opciju da snimi prethodno zabilježeni video u trajnu datoteku.

Zaključak:
Metoda stop_recording je ključna za završetak procesa snimanja videozapisa. Osigurava da se resursi oslobode, video datoteka pravilno zatvori, i da korisnik dobije kontrolu nad sljedećim korakom — spremanjem snimljenog sadržaja. Time se omogućava uredan i stabilan tok rada unutar aplikacije.

Nakon zaustavljanja snimanja, prelazimo na metodu save_recording(self). Ova metoda omogućava korisniku da trajno spremi prethodno snimljeni videozapis na željenu lokaciju na disku. Aktivira se klikom na dugme "Spremi videozapis" nakon što je snimanje završeno.

```
def save_recording(self):
    if not os.path.exists(self.temp_recording_path):
        messagebox.showwarning("Upozorenje", "Nema snimljenog videozapisa za spremanje.")
        return

    save_path = filedialog.asksaveasfilename(defaultextension=".mp4", filetypes=[("MP4 files", "*.mp4")])
    if save_path:
        shutil.move(self.temp_recording_path, save_path)
        messagebox.showinfo("Uspjeh", "Videozapis je spremljen.")
        self.save_btn.config(state="disabled")
```

Kod 36. Implementacija metode za spremanje snimljenog videozapisa

Objašnjenje metode:
if not os.path.exists(self.temp_recording_path):

- Provjerava da li privremena snimljena datoteka zapravo postoji. Ako ne postoji (npr. korisnik nije pokrenuo snimanje), prikazuje se upozorenje.

messagebox.showwarning(...)

- Obavještava korisnika da ne postoji nijedan snimljeni video koji bi mogao biti spremljen.

save_path = filedialog.asksaveasfilename(...)

- Otvara dijalog pomoću kojeg korisnik može odabrati naziv i lokaciju gdje želi da spremi videozapis. Podrazumijevani format za spremanje je .mp4.

if save_path:

- Provjerava da li je korisnik unio validnu putanju i nije zatvorio dijalog bez odabira lokacije.

shutil.move(self.temp_recording_path, save_path)

- Premješta snimljeni privremeni videozapis na odabranu putanju. Time se videozapis trajno čuva.

messagebox.showinfo(...)

- Informiše korisnika da je videozapis uspješno spremljen.

self.save_btn.config(state="disabled")

- Onemogućava dugme "Spremi videozapis", jer se video već jednom spremio, čime se izbjegava višestruko spremanje istih datoteka.

Zaključak:
Metoda save_recording finalizira proces snimanja u stvarnom vremenu. Omogućava korisniku da sačuva rezultat svog rada, osiguravajući jednostavno i intuitivno spremanje bez potrebe za ručnim upravljanjem datoteka. Kombinacija automatske provjere postojanja snimka i korisničkog unosa lokacije čini ovaj korak pouzdanim i korisnički orijentisanim.

Nakon završetka svih glavnih funkcionalnosti aplikacije, prelazimo na metodu on_closing(self). Ova metoda se poziva kada korisnik zatvori prozor aplikacije (npr. klikom na "X" u gornjem desnom uglu). Njena svrha je da osigura pravilno zatvaranje svih aktivnih procesa prije gašenja aplikacije.

```
def on_closing(self):
    self.stop_camera_feed()
    self.root.destroy()
```

Kod 37. Implementacija metode za gašenje aplikacije

Objašnjenje metode:

self.stop_camera_feed()

- Poziva metodu zaustavljanja prikaza kamere (ako je aktivna). Na taj način se osigurava da se video stream s kamere ispravno zatvori i da se resursi oslobode. Ovaj korak je ključan kako bi se izbjeglo "zaključavanje" kamere ili rušenje programa prilikom izlaska.

self.root.destroy()

- Zatvara glavni Tkinter prozor i prekida glavnu petlju aplikacije. Time se program u potpunosti završava.

Zaključak:

Metoda on_closing omogućava sigurno i kontrolisano zatvaranje aplikacije. Osigurava da se svi procesi (poput kamere) isključe prije gašenja, čime se sprječavaju potencijalni problemi sa sistemskim resursima ili neželjeno ponašanje pri ponovnom pokretanju aplikacije. Implementacija ove metode je jednostavna, ali izuzetno važna za stabilnost softvera.

Na samom kraju datoteke gui.py nalazi se posebni kontrolni blok koji omogućava pokretanje aplikacije kao samostalnog programa. Ovaj blok nije metoda unutar klase, već posebna Python konstrukcija koja služi kao ulazna tačka kada se skripta pokrene direktno. Radi se o standardnoj praksi u Python projektima za GUI ili druge aplikacije koje se mogu pokretati kao glavni program.

```
if __name__ == "__main__":
    root = tk.Tk()
    app = FaceBlurApp(root)
    root.protocol("WM_DELETE_WINDOW", app.on_closing)
    root.mainloop()
```

Kod 38. Implementacija kontrolnog bloka

Objašnjenje bloka:

if __name__ == "__main__":

- Ova linija osigurava da se sadržaj unutar nje izvršava samo kada se datoteka gui.py pokrene direktno, a ne kada se uveze kao modul u nekoj drugoj skripti. Time se omogućava fleksibilnost pri korištenju.

root = tk.Tk()

- Kreira se glavni Tkinter prozor (root), koji predstavlja osnovu za sve GUI elemente u aplikaciji.

app = FaceBlurApp(root)

- Instancira se glavna klasa aplikacije FaceBlurApp i veže se za glavni prozor. Ova linija pokreće cjelokupnu logiku i prikazuje početni meni aplikacije.

root.protocol("WM_DELETE_WINDOW", app.on_closing)

- Povezuje događaj zatvaranja prozora ("X" dugme) sa metodom on_closing, kako bi se osiguralo pravilno gašenje aplikacije i zatvaranje svih resursa (npr. kamera).

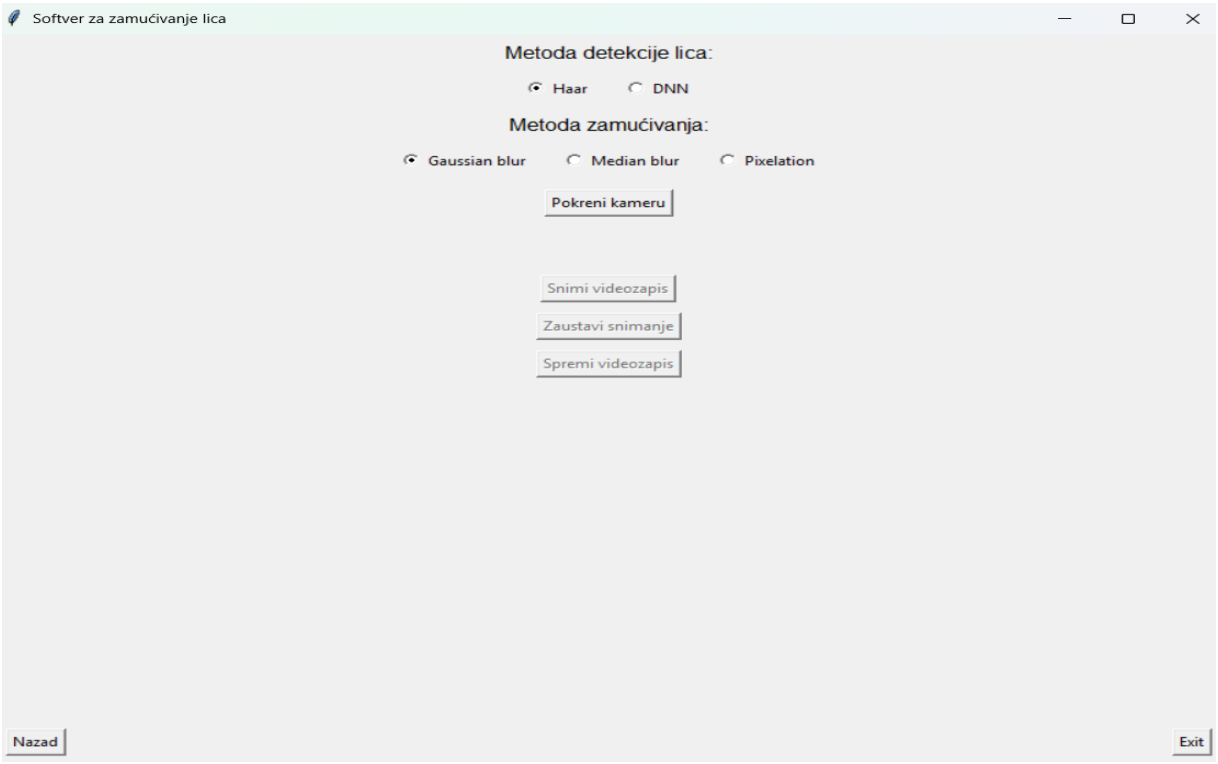
root.mainloop()

- Pokreće glavnu petlju aplikacije. Bez ove linije, GUI se ne bi prikazao niti bi odgovarao na korisničke akcije.

Zaključak:

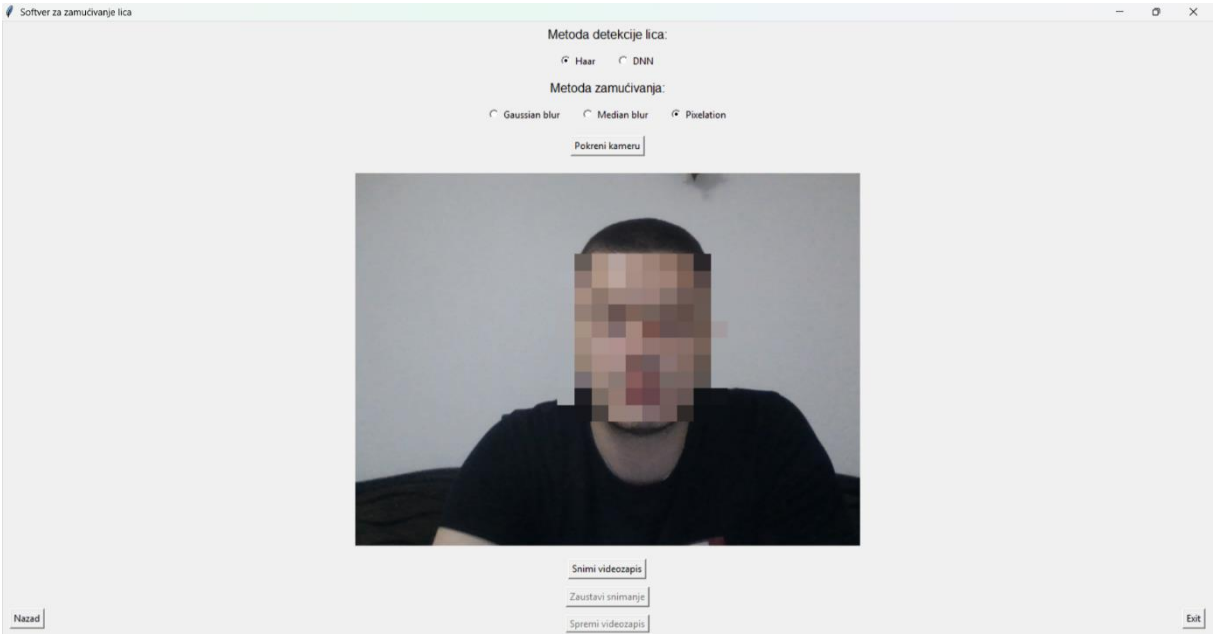
Ovaj komandni blok osigurava da se GUI pravilno inicijalizuje, upravlja životnim ciklusom prozora i zatvara resurse na odgovarajući način. Njegovo postojanje čini aplikaciju samostalnom i spremnom za korištenje bez dodatnih konfiguracija.

Kada smo sve funkcije i metode implementirali, aplikacija je spremna za upotrebu. U nastavku je prikazano korištenje obrade u stvarnom vremenu zamućivanja lica. Kao rezultat toga, kada korisnik pokrene aplikaciju dočeka ga početna stranica gdje bira: Obrada videozapisa i Real time obrada. Kada korisnik klikne „Real time obrada“ dočeka ga novi prozor. Po zadanim postavkama su odabrani Haar metoda prepoznavanja lica i gaussian blur metoda zamucćivanja lica.



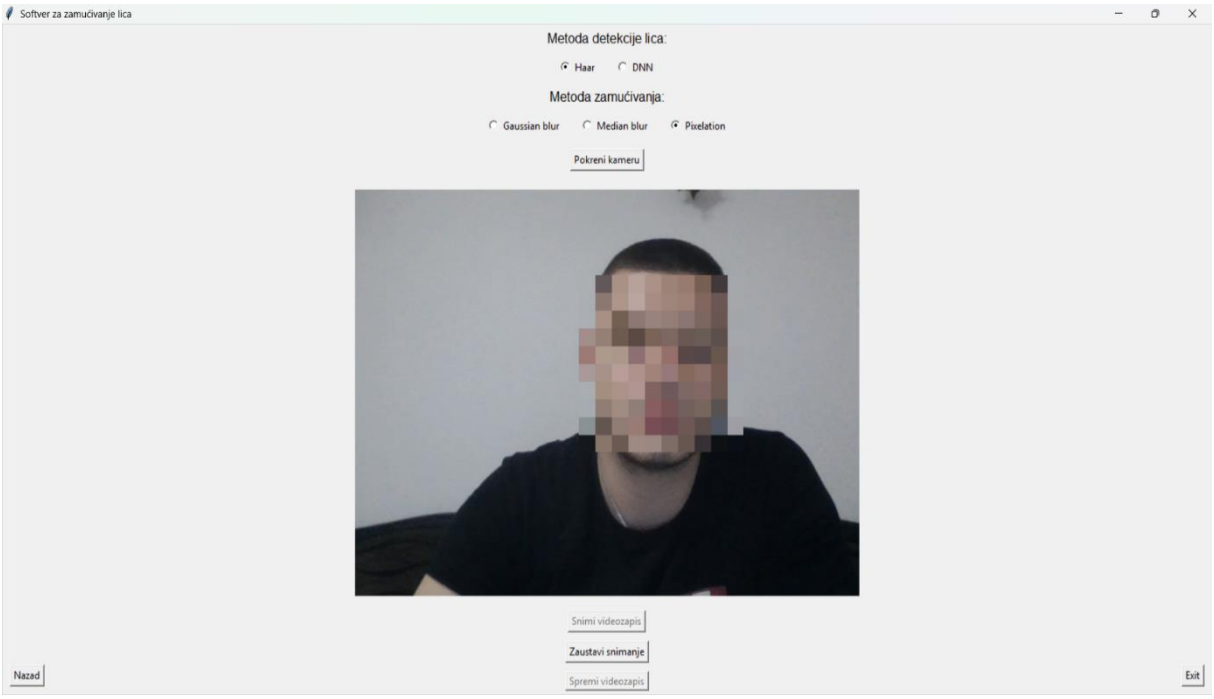
Slika 39. Prikaz novog ekrana za real time obradu

Kada korisnik izabere metodu prepoznavanja lica i metodu zamućivanja i klikne na pokreni kameru, web kamera se odmah aktivira i prikaže u prozoru. Korisnik može vidjeti kako aplikacija uspješno obavlja svoj posao tj. kako radi prepoznavanje i zamućivanje lica.



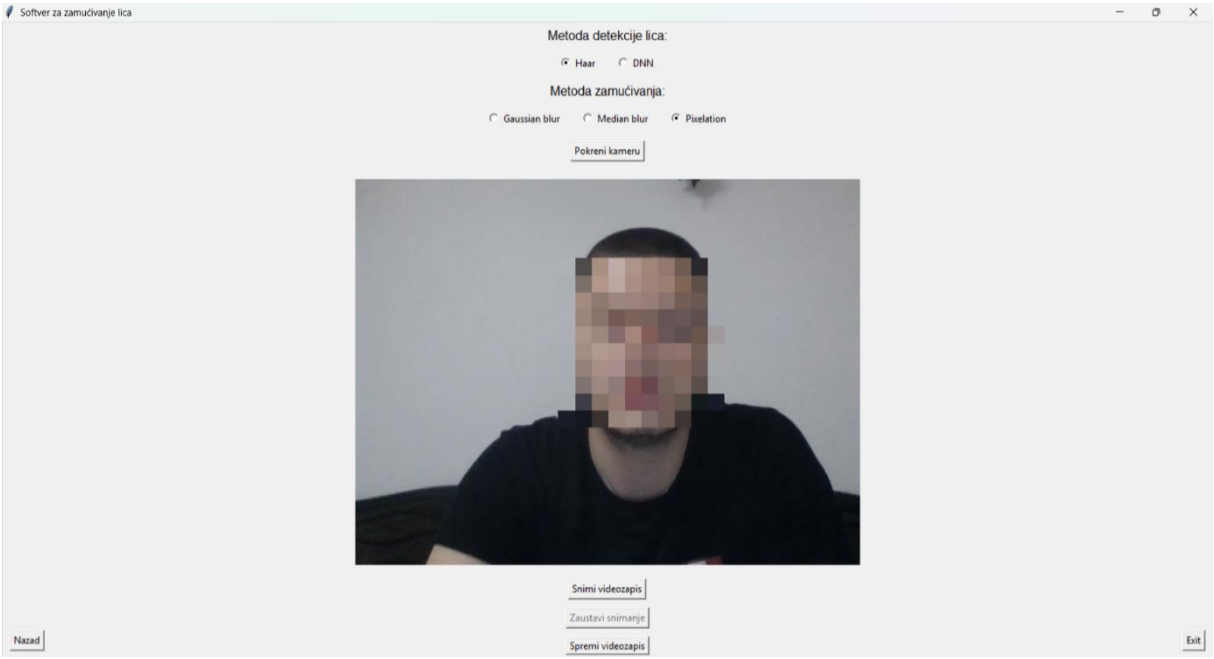
Slika 40. Prikaz pokretanja web kamere i rada aplikacije

Kada je korisnik pokrenuo kameru, korisnik može da snimi videozapis pomoću kamere. Dakle, kada se pokrene snimanje, aktivira se odmah dugme za zaustavljanje snimanja.

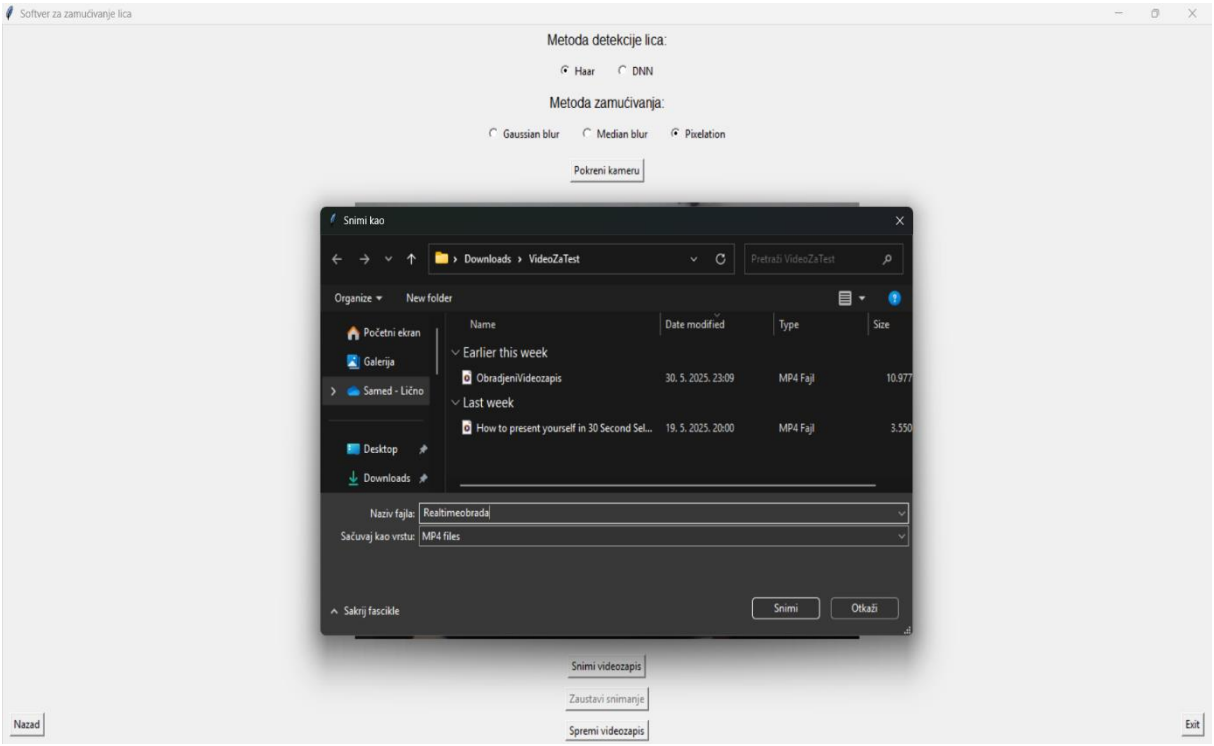


Slika 41. Prikaz pokretanja samog snimanja

Kada korisnik snimi videozapis i klikne zaustavi snimanje, aktivira se dugme za spremanje videozapisa gdje je korisniku omogućeno da bira gdje želi u datotekama da sačuva videozapis i kako želi da ga imenuje.

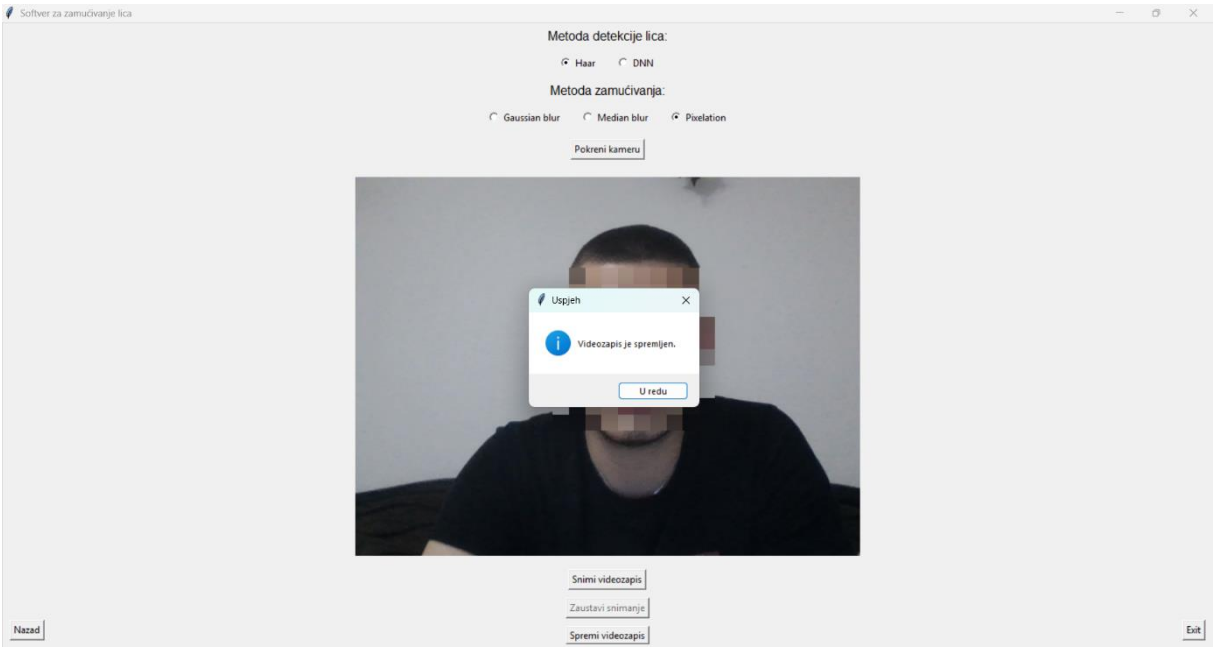


Slika 42. Prikaz zaustavljanja snimanja



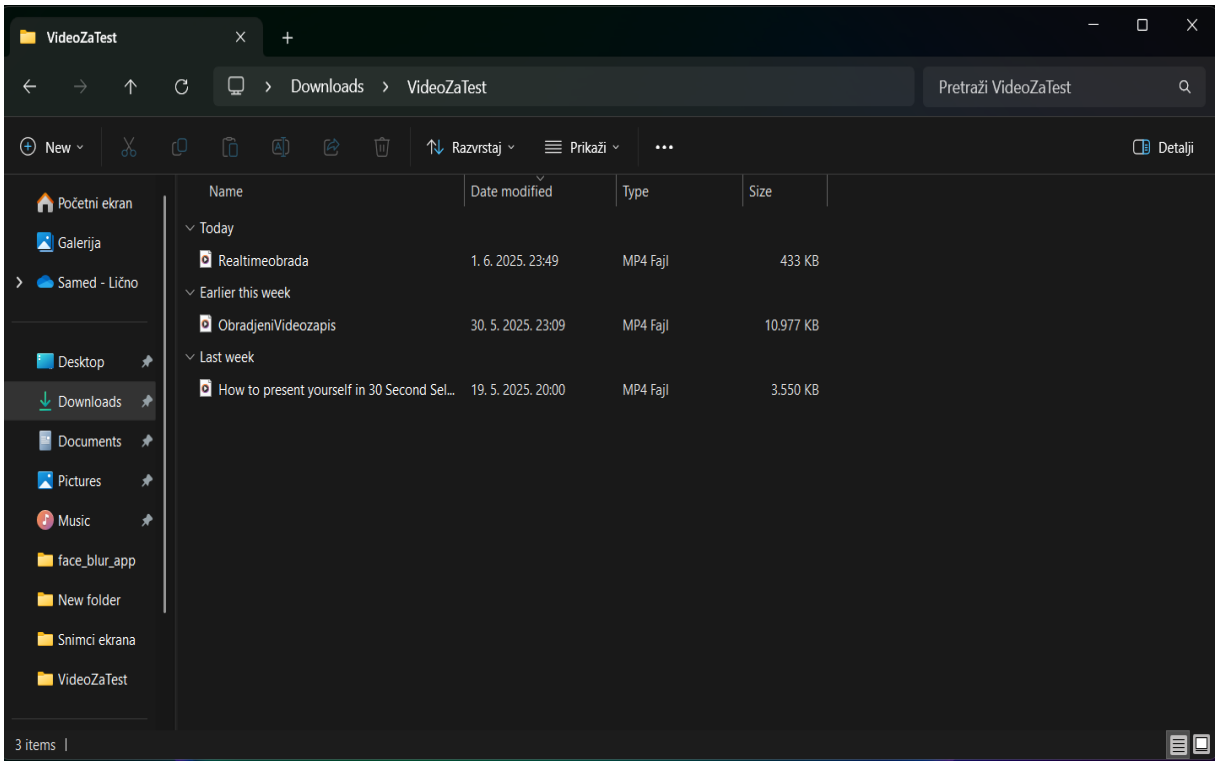
Slika 43. Prikaz spremanja videozapisa

Kada se videozapis uspješno spremi korisniku izađe poruka da je videozapis spremljen.

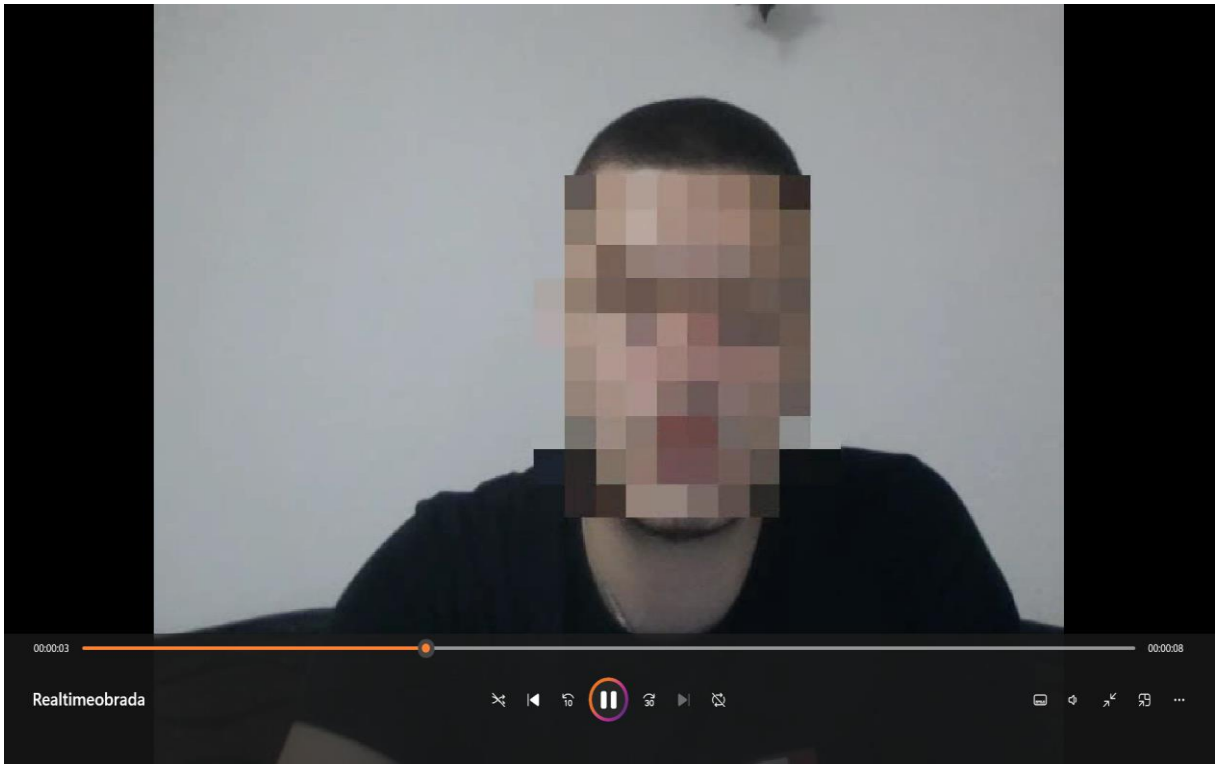


Slika 44. Prikaz uspješno spremljenog videozapisa

Kada korisnik ode do foldera gdje je spremio videozapis i pokrene ga, kao što se i očekuje od aplikacije, aplikacija je uspješno obavila svoj posao.



Slika 45. Prikaz spremljenog videozapisa u folderu



Slika 46. Prikaz uspješno obavljenog zadatka aplikacije

Zaključak aplikacije

Aplikacija za prepoznavanje i zamućivanje lica predstavlja kompletno desktop rješenje izrađeno pomoću Tkinter GUI biblioteke, OpenCV-a za obradu slike i videozapisa, te

standardnih Python alata za upravljanje datotekama i interakciju s korisnikom. Glavni cilj aplikacije je da korisnicima omogući intuitivno, jednostavno i efikasno upravljanje procesom zaštite privatnosti u videozapisima kroz automatsko prepoznavanje i zamućivanje lica.

Aplikacija je modularno strukturirana:

- video_processor.py implementira ključne funkcije za obradu videozapisa – uključujući prepoznavanje lica i primjenu različitih metoda zamućivanja (Gaussian, Median, Pixelation).
- gui.py čini interfejs korisničkog dijela aplikacije – omogućavajući rad u stvarnom vremenu s kamerom, kao i obradu već postojećih videozapisa.

Ključne funkcionalnosti uključuju:

- Prepoznavanje lica u stvarnom vremenu ili iz video datoteke pomoću Haar i DNN metoda.
- Zamućivanje detektovanih lica uz izbor između Gaussian blur-a, Median blur-a i Pixelation metode.
- Snimanje i spremanje videozapisa direktno iz kamere.
- Jednostavan pregled i obradu video datoteke iz sistema korisnika.
- Intuitivan i pregledan korisnički interfejs pogodan i za tehnički manje iskusne korisnike.

Aplikacija omogućava da se zahtjevne operacije i obrade izvode u pozadini, bez zamrzavanja GUI-ja. Također, pravilno rukovanje resursima poput kamere i datoteka osigurava stabilnost i pouzdanost tokom korištenja.
Ova aplikacija može služiti kao osnovni alat za zaštitu privatnosti u snimljenim materijalima, ali i kao temelj za dalji razvoj u oblasti video nadzora, zaštite podataka i računalnog vida. Moguće nadogradnje uključuju integraciju sa bazama podataka, automatsko prepoznavanje identiteta ili čak online obradu putem web servera.

Zaključak:
Aplikacija uspješno kombinuje napredne mogućnosti OpenCV biblioteke sa jednostavnošću Tkinter GUI-ja, pružajući korisnicima snažan alat za anonimnost i zaštitu identiteta. Implementacija je pažljivo izvedena kako bi zadovoljila potrebe korisnika, istovremeno poštujući principe dobre organizacije koda, modularnosti i održivosti softverskog rješenja.

6. TESTIRANJE PERFORMANSI APLIKACIJE

Kako bi se procijenila efikasnost i upotrebljivost aplikacije u stvarnim uslovima, izvršeno je testiranje performansi u zavisnosti od različitih video rezolucija i metoda zamućivanja. Testovi su provedeni na računaru srednje klase bez GPU akceleracije, koristeći dvije metode prepoznavanja lica (Haar i DNN), te tri metode zamućivanja: **Gaussian Blur**, **Median Blur** i **Pixelation**.

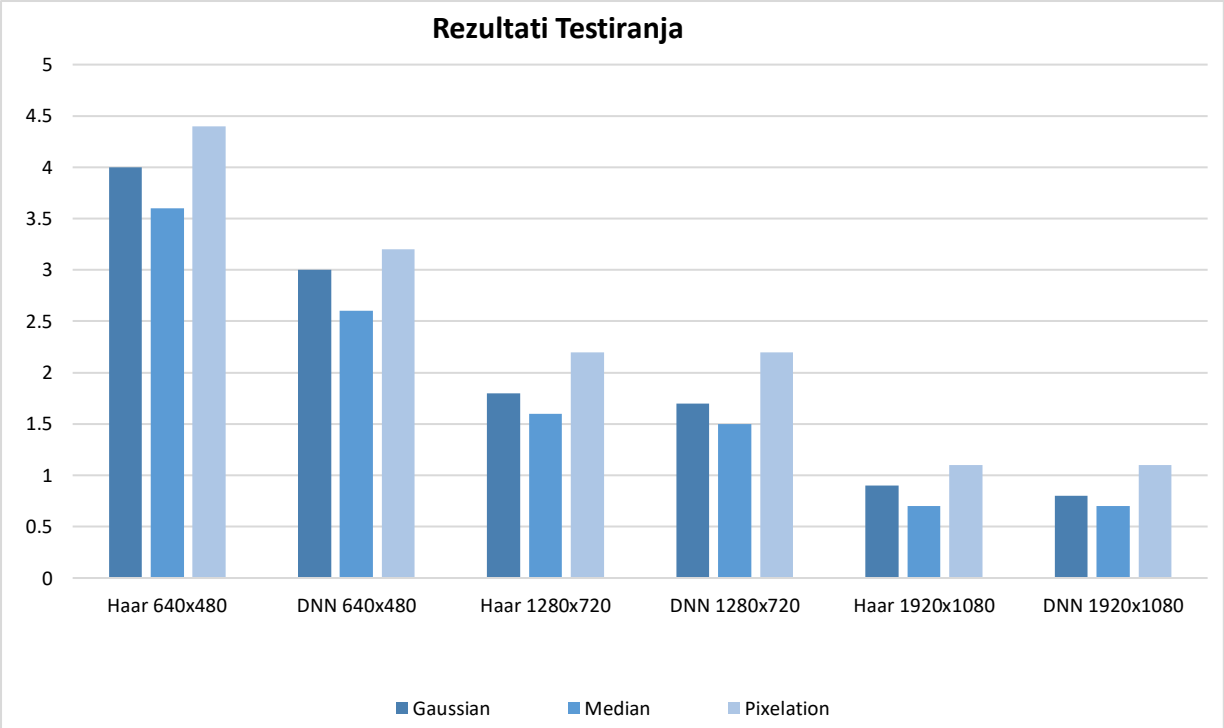
6.1 Metodologija testiranja

- Ulazni video signali generisani su iz postojeće video datoteke i web kamere.
- Rezolucije korištene u testiranju su: **640x480**, **1280x720**, i **1920x1080**.
- Mjerena je prosječna brzina izvođenja (FPS – frames per second) za svaku kombinaciju algoritma i rezolucije.
- Testiranja su izvršena za svaku metodu zamućivanja posebno.

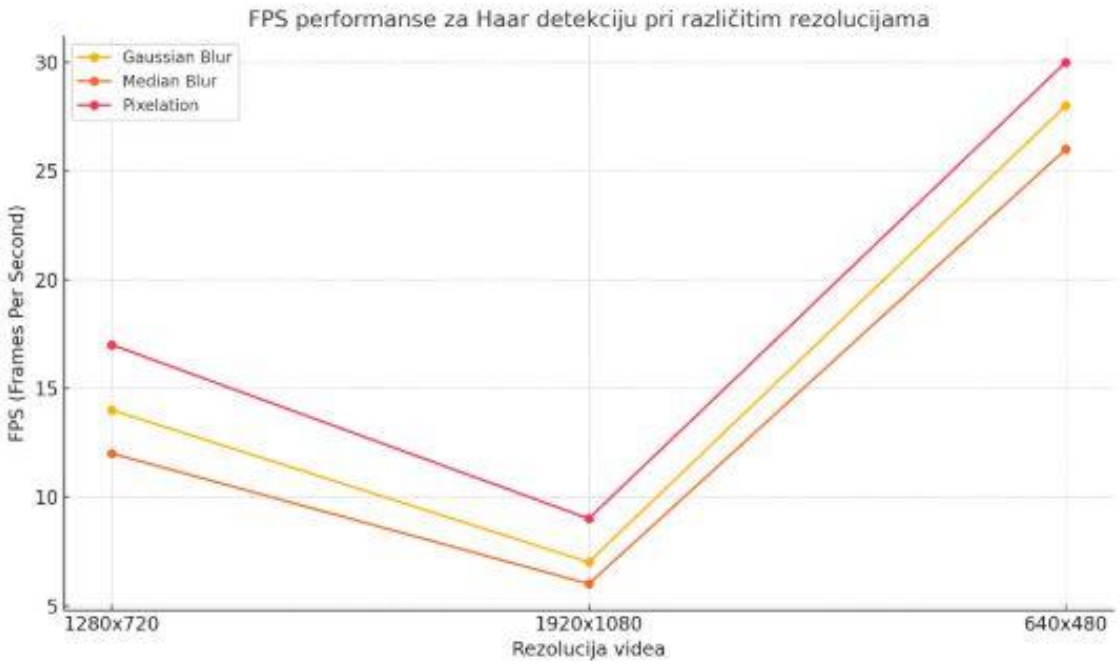
6.2 Rezultati testiranja

Tabela poređenja svih metoda prepoznavanja lica i metoda zamućivanja lica i usporedba performansi na različitim rezolucijama

Rezolucija	Metoda prepoznavanja	Gaussian Blur (FPS)	Median Blur (FPS)	Pixelation (FPS)
640x480	Haar	28	26	30
640x480	DNN	21	19	24
1280x720	Haar	14	12	17
1280x720	DNN	12	10	17
1920x1080	Haar	7	6	9
1920x1080	DNN	6	6	9



Grafik 47. Prikaz linijskog grafa performansi



Grafik 48. Prikaz linijskog grafa performansi

6.3 Analiza performansi

- **Gaussian Blur** i **Median Blur** imaju slične performanse, s time da Median Blur blago zaostaje zbog nešto veće složenosti operacije sortiranja susjednih piksela.

- **Pixelation** je najbrža metoda zamućivanja jer koristi jednostavne operacije skaliranja i bez matematičkih funkcija težinskog prosjeka.
- **Haar se** pokazala značajno bržom u svim rezolucijama u poređenju s DNN-om, naročito pri visokoj rezoluciji (Full HD), gdje DNN znatno usporava rad aplikacije.
- S porastom rezolucije FPS linearno opada kod svih metoda, što je očekivano zbog veće količine podataka za obradu po kadru.

6.4 Preporuke

- Za uređaje sa slabijim hardverom i gdje je brzina bitna, preporučuje se kombinacija Haar + Pixelation.
- Za kvalitetnije i preciznije rezultate u kontrolisanim uslovima, moguća je upotreba DNN + Gaussian Blur ili Median Blur, uz žrtvovanje brzine.

7. KONAČAN ZAKLJUČAK

Ovaj diplomski rad predstavlja sveobuhvatno rješenje za izazov zaštite privatnosti u digitalnim videozapisima putem automatskog prepoznavanja i zamućivanja lica. Kroz temeljitu analizu problema, teorijsku obradu algoritama i praktičnu implementaciju softverske aplikacije, uspješno je demonstriran način na koji se tehnologije računalnog vida mogu primijeniti za efikasnu anonimnost osoba u video sadržaju.

Aplikacija je razvijena u programskom jeziku Python koristeći biblioteku OpenCV, a posebna pažnja je posvećena modularnosti i jasnoći arhitekture. Korištene su dvije ključne metode za prepoznavanje lica: klasični Haar Cascade Classifier i moderni DNN model baziran na dubokom učenju. Time se korisniku omogućava izbor između jednostavnijeg, bržeg pristupa i naprednijeg, ali zahtjevnijeg pristupa, u zavisnosti od scenarija i raspoloživih resursa.

Za zamućivanje detektovanih regija implementirane su tri metode: Gaussian Blur, Median Blur i Pixelation. Svaka metoda je analizirana u kontekstu nivoa anonimnosti, estetskog izgleda, efikasnosti i otpornosti na šum, čime se korisniku omogućava fleksibilan pristup obradi video sadržaja u skladu sa specifičnim potrebama – bilo da je riječ o blagoj cenzuri ili potpunom skrivanju identiteta.

Poseban kvalitet aplikacije jeste integracija u stvarnom vremenu i offline obrade – korisnik može koristiti kameru za zamućivanje lica uživo ili učitati već postojeći videozapis. Implementirani grafički korisnički interfejs (GUI) razvijen pomoću Tkinter biblioteke omogućava intuitivnu interakciju, čineći aplikaciju pogodnom i za korisnike bez tehničkog predznanja.

Testiranjem performansi na različitim video rezolucijama i u različitim uslovima pokazano je da aplikacija nudi zadovoljavajuću efikasnost, posebno kada se koristi DNN model za prepoznavanje i pixelation za zamućivanje, što pruža visok nivo zaštite identiteta.

Softver razvijen u ovom radu demonstrira kako se uz relativno jednostavne alate i algoritme može ostvariti značajan doprinos u oblasti zaštite privatnosti. Aplikacija se može dalje razvijati i nadograđivati u pravcu prepoznavanja emocija, višeklasne klasifikacije objekata, automatskog sačuvavanja zamućenih videozapisa u cloud sistemima ili čak korištenja naprednih tehnika poput YOLO i FaceNet. U vremenu kada je digitalna privatnost sve više ugrožena, ovakva

rješenja postaju nužan alat u borbi za očuvanje anonimnosti i etičke upotrebe video sadržaja.

8. LITERATURA

1. OpenCV službena dokumentacija. Dostupno na: https://docs.opencv.org/
2. OpenCV Python Tutorials. Dostupno na: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
3. Haar Cascade Classifier OpenCV Guide. Dostupno na: https://docs.opencv.org/4.x/d7/d8b/tutorial_py_face_detection.html
4. Real-Time Face Blurring with OpenCV Python. Toolify.ai. Dostupno na: https://www.toolify.ai/ai-news/realtime-face-blurring-with-opencv-python-1119539
5. Nukala, S., Yuan, X., Roy, K., & Odeyomi, O. T. (2024). *Face recognition for blurry images using deep learning*. In 2024 4th International Conference on Computer Communication and Artificial Intelligence (CCAI), IEEE, str. 46–52.
6. Imran, A., Ahmed, R., Hasan, M. M., Ahmed, M. H. U., Azad, A. K. M., & Alyami, S. A. (2024). *FaceEngine: A Tracking-Based Framework for Real-Time Face Recognition in Video Surveillance System*. SN Computer Science, 5(5), 609.
7. GIMP Documentation – Gaussian Blur. Dostupno na: https://docs.gimp.org/3.0/en/gimp-filter-gaussian-blur.html
8. GIMP Documentation – Median Blur. Dostupno na: https://docs.gimp.org/3.0/en/gimp-filter-median-blur.html
9. Cloudinary: *Pixelization*. Dostupno na: https://cloudinary.com/glossary/pixelization
10. Bradski, G. (2000). *The OpenCV Library*. Dr. Dobb's Journal of Software Tools.
11. Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer.
12. Patel, V. M., Gopalan, R., Li, R., & Chellappa, R. (2015). *Visual Domain Adaptation: A Survey of Recent Advances*. IEEE Signal Processing Magazine, 32(3), 53–69.
13. Viola, P., & Jones, M. J. (2004). *Robust Real-Time Face Detection*. International Journal of Computer Vision, 57(2), 137–154.
14. Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). *Image Quality Assessment: From Error Visibility to Structural Similarity*. IEEE Transactions on Image Processing, 13(4), 600–612.
15. Dalal, N., & Triggs, B. (2005). *Histograms of Oriented Gradients for Human Detection*. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).