Elektrotehnički fakultet
Univerzitet u Beogradu

Kompjuterska vizija

Uklanjanje pozadine na videu i menjanje slike

Profesor:
Prof. dr Veljko Papić

Student:
Aleksa Jakovljević 3170/2023

Beograd, jul 2024.

Sadržaj

Uvod

U današnjem digitalnom dobu veoma često se srećemo sa metodom obrade slike kojom se bavimo u ovom projektnom zadatku, a to je uklanjanje pozadine sa video snimka u realnom vremenu. Ovakvu metodu najčešće srećemo prilikom korišćenja alata za online komunikaciju kao što je Microsoft Teams, Google Meet, Zoom i ostali, gde imamo mogućnost da prilikom online sastanka zamenimo trenutnu pozadinu, tj. ambijent u kome se nalazimo sa nekom slikom (npr. lepo uredjena kancelarija, plaža, park i slično). Ovaj feature ne pridodaje preterano samom kvalitetu komunikacije ali unapredjuje korisničko iskustvo i bitan je zbog zaštite privatnosti korisnika. Baš zbog svoje praktičnosti ova metoda kompjuterske vizije je neizostavan deo bilo koje komunikacione mreže.

Ovaj rad istražuje metode za uklanjanje pozadine iz snimaka sa web kamere laptopa i zamenu te pozadine unapred definisanom slikom. Da bi postigli željeni efekat koristimo biblioteku OpenCV koja je open-source i najveća je biblioteka za kompjutersku viziju.

U prvom delu rada ćemo obraditi ručno napisan algoritam koji na osnovu uslikane referentne pozadine pravi razliku izmedju nje i trenutne slike koju kamera beleži. Pomoću te dve slike kreiramo masku za izdvajanje prednjeg plana koji kombinujemo sa unapred definisanom slikom koja služi kao nova pozadina. Objasnićemo ceo proces detekcije prednjeg plana i to kako on zapravo radi.

U poslednjem delu ćemo analizirati naš algoritam tako što ćemo diskutovati o njegovim dobrim i lošim stranama i na samom kraju ćemo govoriti o potencijalnim unapredjenjima.

Teorija iza uklanjanja pozadine

U ovom delu rada ćemo reći nešto više o teoriji koja stoji iza uklanjanja pozadine sa snimka i ukratko ćemo obraditi nekoliko algoritama koji se bave ovim problemom.

Uklanjanje pozadine sa snimka je problem koji rešava tehnika koja se u literaturi često naziva detekcija prednjeg plana tj. na engleskom **Foreground detection** kako ćemo i oslovljavati ovaj pojam u nastavku rada s obzirom na nedostatak srpske reči koja bi bila elegantan prevod reči foreground. Uopšteno govoreći ova tehnika rešava problem tako što sa snimka izdvaja prednji plan i pozadinu i time dobijamo dve komponente snimka koje koristimo u daljem radu shodno zahtevima našeg problema.

Funkcionisanje forground detekcije se zasniva na uočavanju promena u sekvenci slika tj. snimku. To zapravo znači da želimo da zabeležimo kretanje nekog tela u odnosu na statičnu pozadinu i tako možemo najlakše napraviti distinkciju izmedju pozadine i foreground-a.

Neke od osnovnih tehnika za detekciju foreground-a su diferenciranje frejmova, Mean filter tehnika, miks Gausovskih modela i sl. Tehnike koje smo naveli su respektivno kompleksnije jedna od druge i u sledećem delu ćemo čitaoca uputiti u osnove svake.

Mean filtering tehnika radi tako što kroz snimanje održavamo modelovanu pozadinu i tokom vremena je ažuriramo. Usvajamo inicijalnu pozadinu tako što usrednjimo prvih nekoliko frejmova i nakon toga računamo foreground na sličan način kao i kod diferenciranja, a to je oduzimanje trenutne slike od usvojene pozadine. Tokom izvršavanja algoritma ažuriramo pozadinu da bi uračunali promene u pozadini koje se vremenom dešavaju.

Miks Gausovih modela je sofisticiranija tehnika od prethodno navedene zbog toga što se mnogo bolje ponaša u slučajevima kada imamo kompleksniju i dinamičniju pozadinu. MoG pristup modelira svaki piksel snimka kao miks nekoliko gausovskih raspodela . To nam dozvoljava da prezentujemo različite elemente pozadine ali i različita stanje i pozicije koje elementi pozadine mogu da zauzmu. Stoga je jasno zašto se ova tehnika mnogo bolje ponaša u slučajevima kada je pozadina dinamičnija.

Prva navedena tehnika nije opisana na početku jer se naš algoritam upravo zasniva na njoj. Zbog toga ćemo u daljem delu rada imati poseban deo gde objašnjavamo teoriju iza diferenciranja frejmova.

Za razliku od klasičnog problema koji se svodi na izdvajanje objekata koji se kreću u odnosu na statične naš problem je malo drugačiji. U našem slučaju izdvajamo pozadinu na real-time snimku u kome čovek sedi ispred laptopa što donosi razlike u odnosu na klasičan problem uklanjanja pozadine. Korisnik ovog algoritma je čovek koji sedi ispred laptopa i on je dosta statičniji i pokreti se uglavnom svode na pravljenje facijalnih ekspresija i blago pomeranje u okviru kadra. S toga uvodimo neke modifikacije u odnosu na klasične algoritme za izdvajanje pozadine i njih ćemo obraditi u daljem izlaganju.

Diferencijacija frame-ova (frame differencing)

Tehnika detekcije pozadine koju smo koristili u ručno pisanom algoritmu se oslanja na tehniku diferencijacije frame-ova. Ova tehnika segmentira foreground od pozadine tako što uzmemo pozadinsku sliku u jednom trenutku, a zatim svaki frejm koji dobijamo u trenutku **t** (označavamo sa **I(t)**) poredimo sa tom pozadinom. Kada kažemo poredjenje slika mislim na korišćenje tehnike oduzimanja slika u kompjuterskoj viziji gde za svaki piksel u I(t) uzimamo vrednost tog piksela koju označavamo sa **P(I(t))** i oduzimamo je sa odgovarajućom vrednosti piksela na istoj poziciji na pozadini, koju označavamo sa **P(B)**.

Matematički to možemo napisati kao:

$$P[F(t)] = P[I(t)] - P[B]$$

Ako imamo pozadinu koja nije potpuno statična (što nije naš slučaj) uvodi se prag (treshold) sa kojim poredimo razliku izmedju dva frejma.

Sada kada smo objasnili osnovni princip na osnovu koga dobijamo foreground, tj. razliku izmedju trenutnog frejma i pozadine možemo da krenemo da obradjujemo naš algoritam koji se razlikuje u nekim segmentima da bi poboljšali njegove performanse.

Algoritam za uklanjanje pozadine

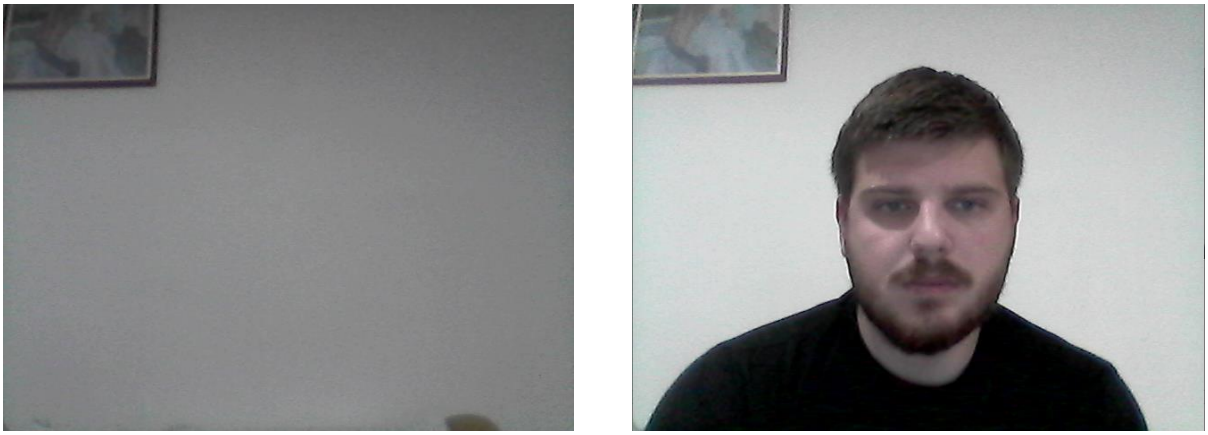
U ovom delu rada ćemo proći kroz kod i objasniti kako radi naš algoritam. Može se reći da je sam algoritam podeljen u dva dela, prvi, inicijalni deo u kome konfigurišemo sve što nam treba i drugi, glavna petlja u kome snimamo i obradjujemo svaki frejm koji kamera snimi.

Algoritam radi tako što poredi statičnu pozadinu koja se nalazi iza korisnika i trenutni kadar. Da bi ovo izveli potrebno je da imamo jasno uslikanu sliku pozadine što naš algoritam zapravo i radi. Da bi uspešno uslikali pozadinu, prilikom pokretanja programa, pre nego što kliknete taster ’e’ za uslikavanje pozadine, trebalo bi da se pomerite iz kadra. Nakon klika na taster ’e’ korisnik se može vratiti u kadar i uspešno koristiti naš program. Jednostavno rečeno, na početku pokretanja programa prvi kadar mora biti kadar bez korisnika u njemu gde se jasno vidi pozadina i ambijent u kome se on nalazi.

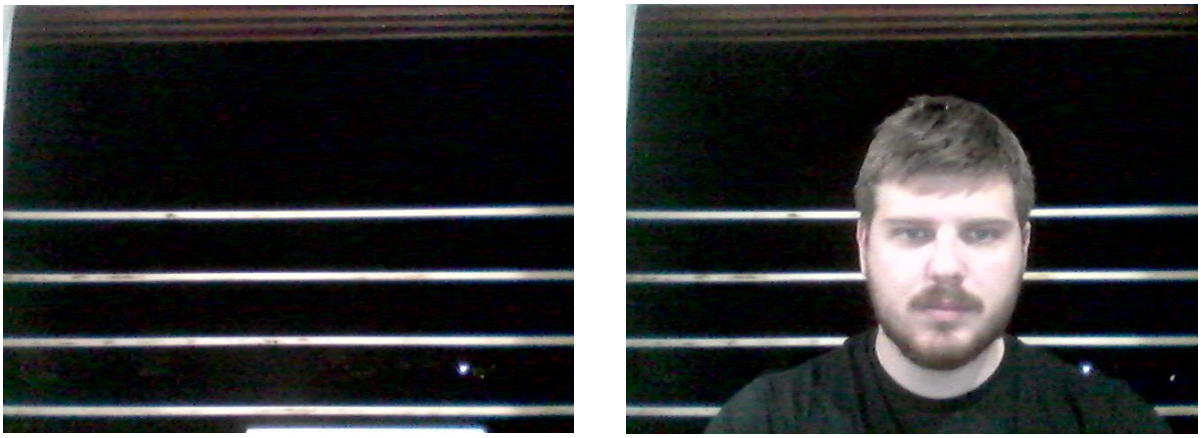
Na slikama 1, 2 i 3 prikazan je prvi kadar sa leve strane, koji uzimamo kao referentnu pozadinu, a sa desne je kadar u kome se nalazi korisnik. Vidimo da postoje tri reda slika, u tri različita ambijenta i kroz ceo rad ćemo pratiti kako se algoritam ponaša u ova tri ambijenta jer smo procenili da oni reprezentuju granične slučajeve u odredjenoj meri. Na slici 1 vidimo da pozadina ima bež boju i ovakav ton pozadine može biti nezgodan jer je sličan boji kože, pa se prilikom diferenciranja frejmova može desiti da algoritam ne može uspešno da diferencira kožu od pozadine. Baš iz ovih razloga izabrali smo ovaj slučaj jer ga smatramo dovoljno ekstremnim da reprezentuje ovakvu situaciju u praksi. Slučaj kada je pozadina bele boje (Slika 2) ili crne boje (Slika 3) je uzeta u razmatranje jer je to ekstremni slučaj što se tiče svetline i kontrasta pozadine od foreground-a.

Posmatranjem ova tri slučaja cilj nam je da čitalac rada vidi kako se ovaj algoritam ponaša u graničnim slučajevima i da pokažemo da algoritam radi za različite ambijente.

Slika 1



Slika 2



Slika 3

Inicijacija

Najpre inicijalizujemo video snimanje preko web kamere laptopa i učitavamo sliku koju smo predodredili kao zamenu pozadine. Za ove operacije koristimo ugradjene funkcije iz biblioteke OpenCV. Pokretanje snimka prikazano je na slici 4:

```
import cv2 as cv
import numpy as np
import cvzone

# Iniciranje video zapisivanja
video = cv.VideoCapture(0, cv.CAP_DSHOW)

# Ucitavanje nove pozadinske slike
newBgImage = cv.imread("summer-beach-background.jpg")
if newBgImage is None:
    print("Error: Could not load background image.")
    exit()
```

Slika 4

Nakon toga beležimo pozadinu i to radimo tako što uzimamo prvi frejm sa početka snimanja (Slika 5):

```
# Zabeležavanje početnog frame-a kao referente pozadine
ret, bgReference = video.read()
```

Slika 5

U inicijalnom delu koda se nalazi i funkcija za promenu veličine slike (Slika 6) koja je zamena za našu pozadinu. Njena uloga je da prilagodi novu pozadinu širini i visini prozora koji se otvori kada pokrenemo snimanje i dobijamo odgovarajuće dimenzije nove pozadine.

```
def resize(dst, img):
    width = img.shape[1]
    height = img.shape[0]
    dim = (width, height)
    resized = cv.resize(dst, dim, interpolation=cv.INTER_AREA)
    return resized
```

Slika 6

Takodje, pre ulaska u glavnu petlju definisana je funkcija koja služi da poništi efekat prilagodjavanja boje kod web kamere (Slika 7). Tokom testiranja koda primetio sam da kada se iza mene nalazi pozadina koja je svetlija, njena svetlina se promeni prilikom mog ulaska u kadar. To se dešava zbog toga što kamera sama prilagodjava boju u svakom trenutku da bi slika izgledala što bolje za različite nivoe osvetljenja prostorije. Ovaj efekat se odlično vidi na slici 2, gde prilikom uslikavanja pozadine vidimo da je ona gotovo siva, ali mojim ulaskom u kadar ona postaje bela. Efekat je primetan i na slici 1. Tu nastaje problem jer mi u ovom algoritmu efektivno poredimo boju referentne pozadine i trenutnog kadra i na osnovu toga zaključujemo šta je foreground, a šta pozadina (više o tome ćemo pričati u narednom delu rada). Samim tim što se boja pozadine menja iako to nije slučaj u realnosti, jer je ona statična, imamo problem jer naš algoritam može da pomeša efekat prilagodjavanja boje sa novim objektom koji je ušao u kadar i koji moramo segmentirati.

```python
def adjust_brightness(image, brightness_increment=50, threshold=110):
    # Kreiramo masku koja pamti vrednosti piksela koji su veci od praga
    mask = image > threshold

    brightness_matrix = np.full(image.shape, brightness_increment, dtype=np.uint8)

    # Povecavamo vrednosti piksela za one piksele za koje je maska jednaka True
    brightened_image = np.where(mask, cv.add(image, brightness_matrix), image)

    return brightened_image
```

Slika 7

Kao što sam ranije naveo, efekat je najizraženiji kada iza imamo veoma svetlu pozadinu (beli zid i slično) pa je ideja iza ove funkcije da povećamo vrednosti piksela čije vrednosti prelaze empirijski odredjen prag 110. Povećanje od 50 je takodje empirijski odredjeno tako da boja zida izgleda što približnije onoj boji koju imamo kada se pojavimo u kadru.

Glavna petlja

Inicijalizacija snimanja

Sada kada smo prošli inicijalni deo koda, u kome učitavamo podatke potrebne za dalji rad programa, možemo preći na glavnu petlju. Kao što iz imena možemo naslutiti naš kod se izvršava u While beskonačnoj petlji. To radimo da bi obezbedili kontinualno snimanje video zapisa jer funkcija iz biblioteke OpenCV koju pozivamo `video.read()`, gde je `video` promenljiva kojoj smo dodelili `cv.VideoCapture` u

inicijalnom delu koda, beleži samo jedan frejm sa web kamere laptopa. Konstantnim vrtenjem u petlji zapravo dobijamo kontinualno snimanje i u okviru petlje izvršavamo obradu trenutnog frejma i ažuriramo prikaz tako što prikazujemo svaki frejm pa korisnik dobija osećaj kontinualnog video snimanja.

Nakon pozivanja funkcije koja snima trenutni frejm pozivamo i funkciju definisanu u inicijalnom delu za promenu veličine slike. Na samom dnu slike vidimo da se poziva funkcija za prilagodjavanje svetline pozadine onoj koju imamo kada udjemo u kadar. Sam početak glavne petlje izgleda ovako (Slika 8):

```
while True:
    ret, img = video.read()
    if not ret:
        print("Error: Could not read frame from video source.")
        break

    # Resize nove pozadine koja odgovara veličini video frejma
    if newBgImage.shape[:2] != bgReference.shape[:2]:
        bg = resize(newBgImage, bgReference)
    else:
        bg = newBgImage

    if takeBgImage == 0:
        bgReference = img
        bgReference = adjust_brightness(bgReference)
```

Slika 8

Na slici 9 prikazna je svetlija pozadina koja odgovara pozadini koju vidimo na levoj slici slike 2. Vidimo da je sada belina zida mnogo sličnija belini na desnoj slici 2.

Slika 9

Kreiranje maske oduzimanjem frejmova

Sledeći deo petlje jeste sama obrada trenutne slike koju dobijamo sa kamere kako bi napravili masku kojom ćemo ekstrahovati foreground od pozadine. Kreiranje maske započinjemo sledećim setom naredbi (Slika 10):

```
# Kreiranje maske
diff1 = cv.subtract(img, bgReference)
diff2 = cv.subtract(bgReference, img)

cv.imshow("diff1", diff1)
cv.imshow("diff2", diff2)
```

Slika 10

U ovom delu koda oduzimamo trenutni frejm `img` od referentne pozadine (pozadina koja nam služi za zamenu) i obrnuto. Oduzimanje dve slike se vrši preko funkcije `subtract` iz OpenCV biblioteke i ona radi tako što oduzima odgovarajuće vrednosti piksela slike koja je druga prosledjena funkciji od prve. Pored toga što računa razliku izmedju vrednosti dva piksela ova funkcija obezbedjuje da je rezultat razlike koju dobijemo u odgovarajućem opsegu što znači da negativne vrednosti postavlja na nulu tj. piksel postaje crn. Za slike koje imaju više kanala, kao što je naš frejm sa kamere, koji je RGB slika, razlika izmedju piksela se računa za svaki od kanala.

Ako tražimo razliku izmedju trenutnog frejma i referentne pozadine postavlja se pitanje zašto razliku računamo dva puta tako da dobijamo promenljive **diff1** i **diff2**.

diff1 reprezentuje razliku '`img - bgReference`' čime naglašavamo delove slike `img` koji su svetliji od referentne pozadine. Prostije rečeno, kada oduzmemo piksel pozadine koji je svetliji od piksela trenutnog frejma rezultujući piksel bi trebalo da ima negativnu vrednost, tj. nultu (znamo na osnovu prethodno pomenutih informacija o tome kako `cv.subtract` radi). Takodje, većina piksela će imati iste vrednosti jer je veći deo trenutnog frejma isti sa referentnom pozadinom (jedina razlika je u čoveku koji se trenutno nalazi u kadru, pozadina ostaje ista, statična) pa očekujemo da njihova razlika bude bliska nuli. To znači da će jedini svetli delovi slike diff1 biti oni za koje su vrednosti piksela `img` veći od `bgReference`.

Na slikama 11, 12 i 13 su prikazane slike diff1 za različite boje pozadine. Na slikama 11 i 12 vidimo samo crnu siluetu korisnika jer je pozadina svetlija (može se videti na slikama

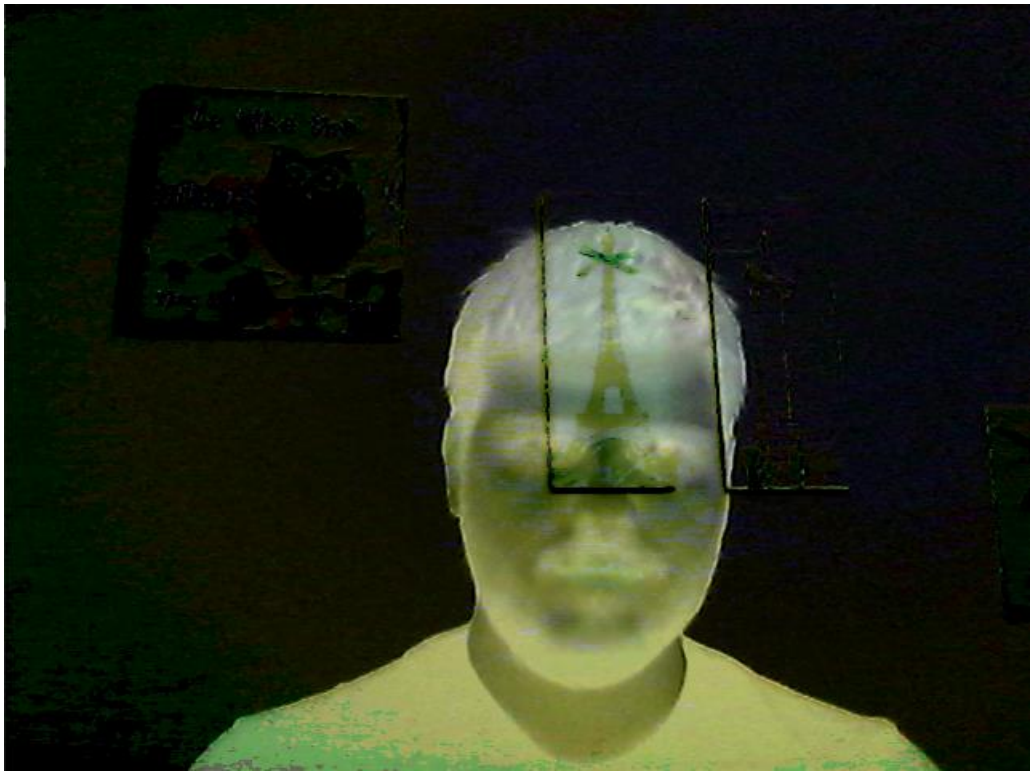
1 i 2) od njega pa je ovo rezultat koji očekujemo. Na slici 13 jasno vidimo korisnika izdvojenog iz pozadine pošto je ona gotovo crna pa je ovo rezultat koji očekujemo. Takodje, na slici 13 je dobro oslikano šta dobijamo za diff1 jer su rešetke koje su potpuno bele boje i koje pripadaju pozadini sada crne jer smo veoma visoke vrednosti piksela koje predstavljaju rešetke oduzeli od manjih i kao rezultat dobili piksele čije su vrednosti nula ili bliske nuli.

Slika 11

Slika 12

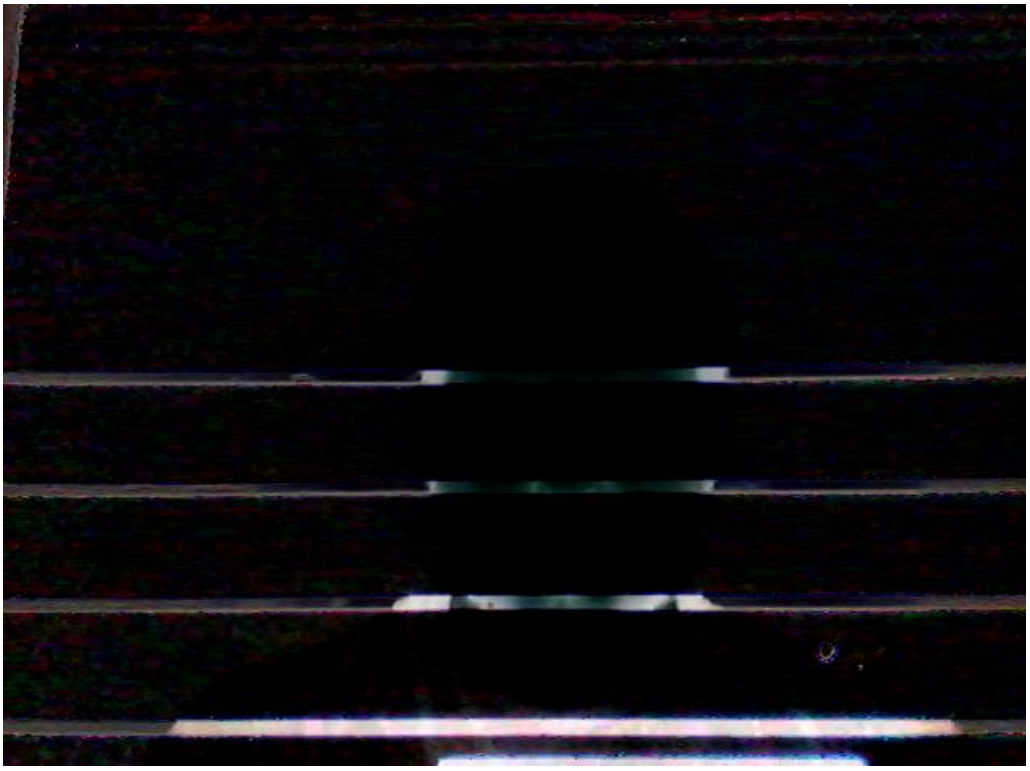
Slika 13

diff2 reprezentuje razliku ‘`bgReference - img`’ čime naglašavamo delove slike img koji su tamniji od referentne pozadine.



Slika 14

Slika 15



Slika 16

Jasno se vidi da su sada naglašeni delovi trenutne slike koji su tamniji od pozadine, a to se najbolje vidi na osnovu majice, brade, brkova, kose i zenica koje u originalu imaju veoma male vrednosti piksela. Takodje, na slici 16, u delovima gde su rešetke, jasno vidimo obrise korisnika jer su to delovi gde su pikseli tamniji od rešetki.

Računanjem **diff1** i **diff2** efektivno radimo detekciju foreground-a jer ako ne postoji razlika izmedju trenutnog frejma i referentne pozadine to zapravo znači da se ništa novo nije pojavilo tj. da ne postoji foreground koji moramo detektovati.

Uklanjanje šuma nakon oduzimanja

```
diff = cv.add(diff1, diff2)
diff[abs(diff) < 60.0] = 0

cv.imshow("diff", diff)
```

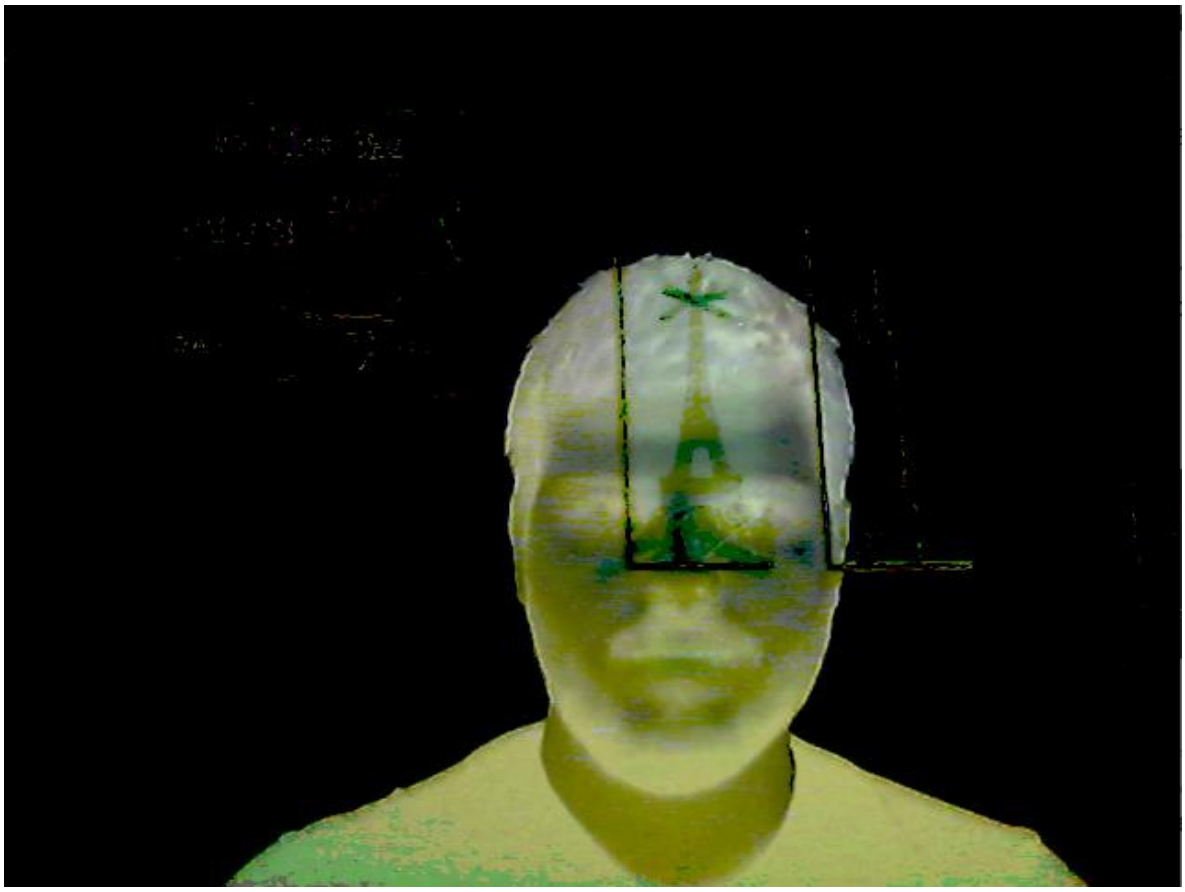
Slika 17

Poslednje što treba uraditi je da iskombinujemo obe razlike tako što ih saberemo i dobijemo **diff** i time zapravo enkapsuliramo informacije iz oba smera, mesta na kojima su pikseli `img` svetliji od piksela pozadine i mesta na kojima su tamniji. Ovime dobijamo konačnu masku koja ima sveobuhvatniju informaciju o razlici izmedju foreground-a i background-a i time nam je omogućena lakša detekcija prednjeg plana sa snimka.

Na slici 17 vidimo da se vrednosti piksela koje su manje od 60 postvaljaju na nulu. To radimo da bi uklonili male razlike u svetlini izmedju dva kadra. Često se desi da kada udjemo u kadar nakon snimanja pozadine, tj. nakon samoj početka snimanja, web kamera lap topa odradi korekciju boje celokupnog frejma zbog novog objekta koji je ušao u kadar. Ovo naročito može da se primeti kod lošijih kamera koje, da bi održale bilo kakav kvalitet, imaju mnogo veću fluktuaciju u boji prilikom prolaska novih objekata (novih boja) kroz kadar.

Jasno je da kada korisnik udje u kadar i za nijansu promeni ton boja celog frejma blago se promeni i boja pozadine što će izazvati da se pojavi razlika u boji izmedju pozadine trenutnog frejma i referentne pozadine za koju znamo da ne postoji jer je naša pozadina statična. Imamo sreće da su razlike u boji koje unosi korekcija boje same kamere dovoljno male da prostim ograničavanjem, gde svaki piksel manji od vrednosti 60 postavljamo na nulu, možemo lako ukloniti ovaj efekat.

Na slikama 18, 19 i 20 vidimo konačne maske za različite ambijente u kojima se korisnik nalazi:



Slika 18

Slika 19

Slika 20

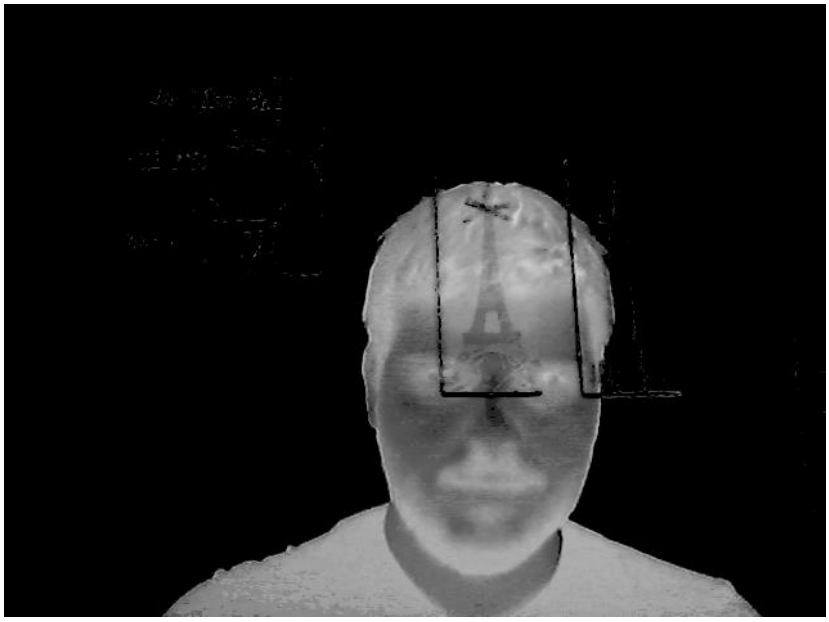
Konverzija boje maske u grayscale

Nakon što smo kreirali masku tako što smo oduzimali trenutni frejm od referentne pozadine sada konvertujemo boju te maske u grayscale koristeći funkciju `cv.cvtColor()`. Nakon konverzije radimo blago uklanjanje šuma tako što za sve vrednosti piksela koji su manji od 10 kažemo da su sada jednaki nuli. Objašnjen kod prikazan je na slici 21.

```
gray = cv.cvtColor(diff, cv.COLOR_BGR2GRAY)
gray[np.abs(gray) < 10] = 0
fgMask = gray
```

Slika 21

Postavlja se pitanje zašto radimo konverziju iz RGB palete u grayscale. Glavni razlog je pojednostavljivanje problema jer grayscale boja u sebi sadrži samo informaciju o intenzitetu, u opsegu od crne do bele, što pojednostavljuje dalje procesiranje slike koje ćemo imati u odnosu na sliku u boji sa tri kanala (RGB). Za rad sa jednim kanalom nam je potrebno manje kompjuterskih resursa što je jako bitno jer nad ekstrahovanom maskom vršimo mnoštvo operacija u realnom vremenu i radimo u Python-u koji je jedan od sporijih programskih jezika. Jako je bitno da svu obradu slike možemo da izvršimo dovoljno brzo tako da korisnik našeg programa kao izlaz dobije snimak koji je u realnom vremenu kontinualan, bez nekih preskakanja i kočenja. Još jedan razlog zašto prelazimo na grayscale boje je zato što ćemo prilikom obrade koristiti neke operacije koje je pogodnije koristiti nad grayscale slikama, kao što je bitsko ne ili bitsko i. Na kraju ovog dela koda dobijamo foreground masku `fgMask` preko koje ćemo kasnije dobiti jasnu segmentaciju izmedju foregrounda i pozadine.



Slika 22

Uklanjanje šuma metodama erode i dilate

```
# Uklanjanje suma
kernel = np.ones((3, 3), np.uint8)
fgMask = cv.erode(fgMask, kernel, iterations=2)
fgMask = cv.dilate(fgMask, kernel, iterations=2)
```

Slika 23

Sledeći set naredbi (Slika 23) koje se javljaju u našoj petlji služi da uklonimo šum sa naše slike, a to radimo pomoću morfoloških funkcija **erode** i **dilate**. Najpre napravimo kernel 3x3 (veličina izabrana empirijski) kojim ćemo 'klizati' preko slike i primenjivati morfološke operacije koje smo pomenuli.

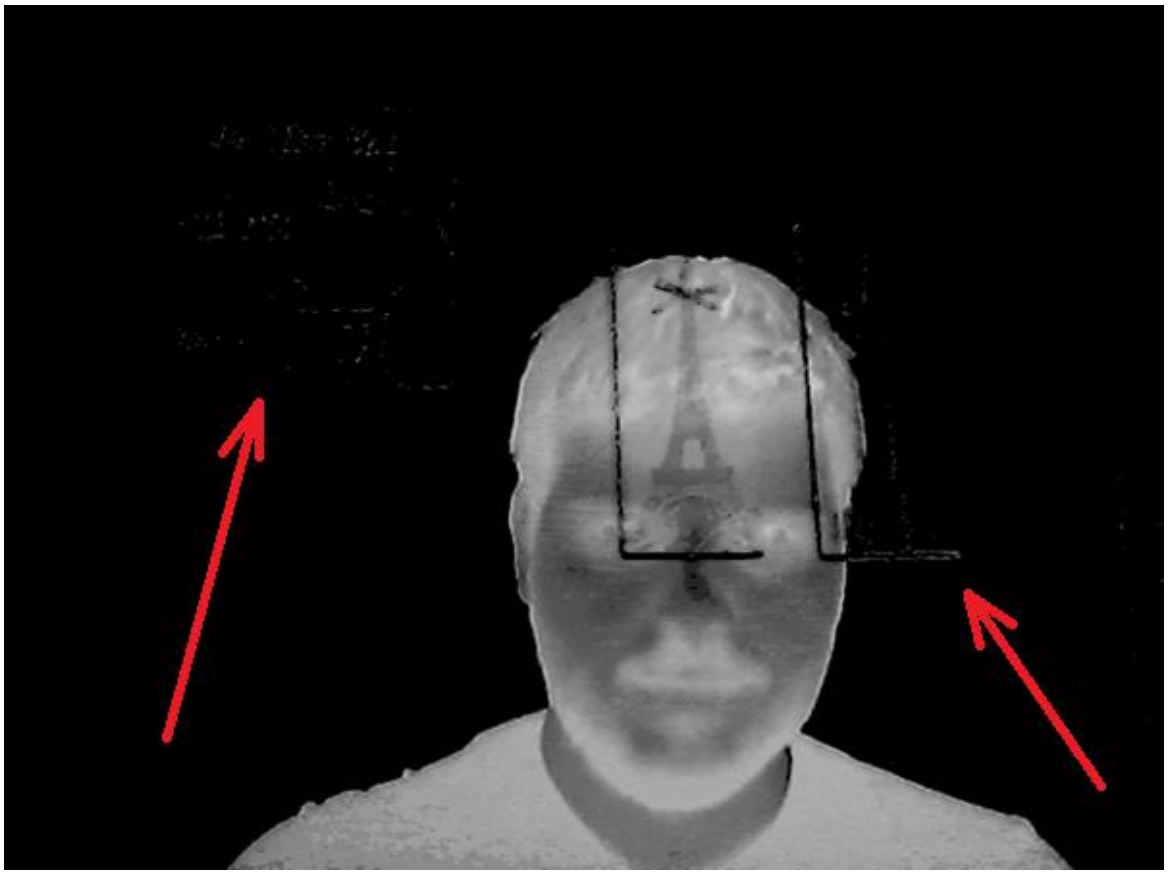
Erosion je morfološka operacija koja uklanja svetle piksele na ivicama objekata. Radi tako što postavljamo kernel preko svakog piksela slike i postavljamo vrednost tog piksela na minimalnu vrednost koja je zahvaćena kernelom. Erozija nam služi da uklonimo mali beli šum na slici (beli kao zašumljenost slike pojedinim belim pikselima) i da učinimo da mali beli objekti nestanu. Konkretno u našem slučaju erozija je pogodna za uklanjanje malih regija belih piksela koji su se pojavili u pozadini. Ako metodu posmatramo kroz primer, zamislimo da se nalazimo u prirodi i u našoj pozadini se nalazi drvo koje se njiše zbog vetra. Ranije smo objasnili da segmentaciju foreground-a radimo tako što poredimo svetlinu/tamnoću piksela izmedju referentne pozadine i trenutnog frejma. Samim njihanjem drveta naš algoritam će uočiti promene svetline odredjenih piksela i to će se izraziti na našu foreground masku pojavom belih piksela na njoj. Pošto su ta pomeranja mala takvi beli pikseli će stvoriti bele regione na našoj masci koji su mali po površini pa ćemo erozijom to neželjeno pomeranje da uklonimo sa maske. Konkretnije, na primeru korisnika koji sedi u zatvorenom prostoru, što je slučaj koji najčešće očekujemo, prilagodjavanje boje kamere može izazvati da pojedini delovi slike posvetle i to će se na našoj masci odraziti kao mali region belih piksela. Nama je u interesu da tu regiju uklonimo jer je ona efektivno deo pozadine pa to radimo tehnikom erozije.

Dilation je inverzna tehnika tehnici erozije. Ovde prolazimo kroz svaki piksel kernelom i vrednost piksela postavljamo na maksimalnu vrednost koju smo zahvatili kernelom. Ovim povećavamo bele regije na binarnoj slici. Dilacija služi da povratimo površinu belih regija koje smo smanjili erozijom.

Dve tehnike koristimo zajedno jer erozija ima za cilj da ukloni male bele regije (bele tačkice na slici) koje možemo proglasiti šumom, ali njenim korišćenjem smanjujemo i bele površine koje su nam od interesa da segmentiramo (regije gde se pojavio čovek u

kadru). Da bi povratili površine koje su nam od interesa sledeći korak je da upotrebimo dilaciju koja radi operaciju inverznu eroziji. Kombinacijom erozije i dilacije efektivno gubimo šum u pozadini koji je prouzrokovan malim pomeranjima u pozadini ili korekcijom boje same kamere.

Efekat koji postižemo korišćenjem ove dve operacije za uklanjanje šuma se najbolje vidi ako uporedimo sliku 24 i sliku 25. Na slici 24 vidimo male bele tačkice koje potiču od slika na zidu koje se nalaze u pozadini i označene su strelicama. Kada primenimo eroziju i dilaciju dobijemo sliku 25 na kojoj se jasno vidi da takvih tačkica više nema i da je korisnik jasno predstavljen belom siluetom na tamnoj pozadini.



Slika 24

Slika 25

Morfološko zatvaranje

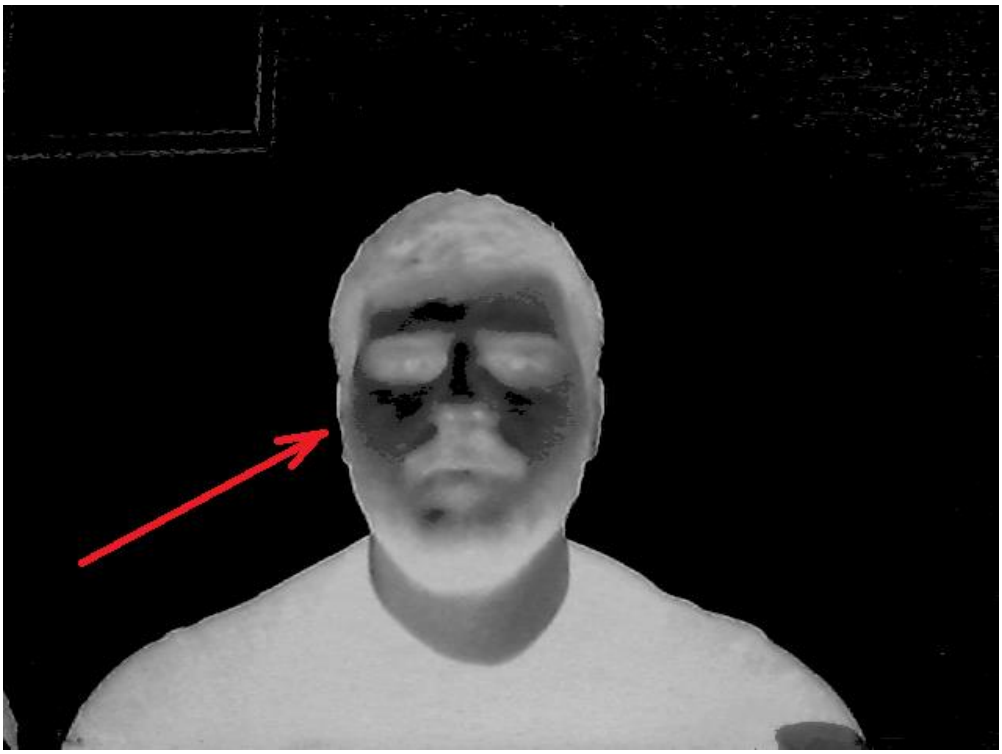
Sledeći korak u procesiranju slike tako da dobijemo što bolju segmentaciju jeste morfološko zatvaranje, koje ostvarujemo pozivanjem funkcije iz biblioteke OpenCV, `cv.morphologyEx` (Slika 26):

```
# Delimo sliku na tri dela: 25%, 50%, 25%
height, width = fgMask.shape
left_part = fgMask[:, :width//4]
middle_part = fgMask[:, width//4:3*width//4]
right_part = fgMask[:, 3*width//4:]

small_kernel = np.ones((3, 3), np.uint8)
large_kernel = np.ones((20, 20), np.uint8)

left_part = cv.morphologyEx(left_part, cv.MORPH_CLOSE, small_kernel)
middle_part = cv.morphologyEx(middle_part, cv.MORPH_CLOSE, large_kernel)
right_part = cv.morphologyEx(right_part, cv.MORPH_CLOSE, small_kernel)

# Spajamo nazad delove
fgMask[:, :width//4] = left_part
fgMask[:, width//4:3*width//4] = middle_part
fgMask[:, 3*width//4:] = right_part

fgMask[fgMask > 3] = 255
```

Slika 26

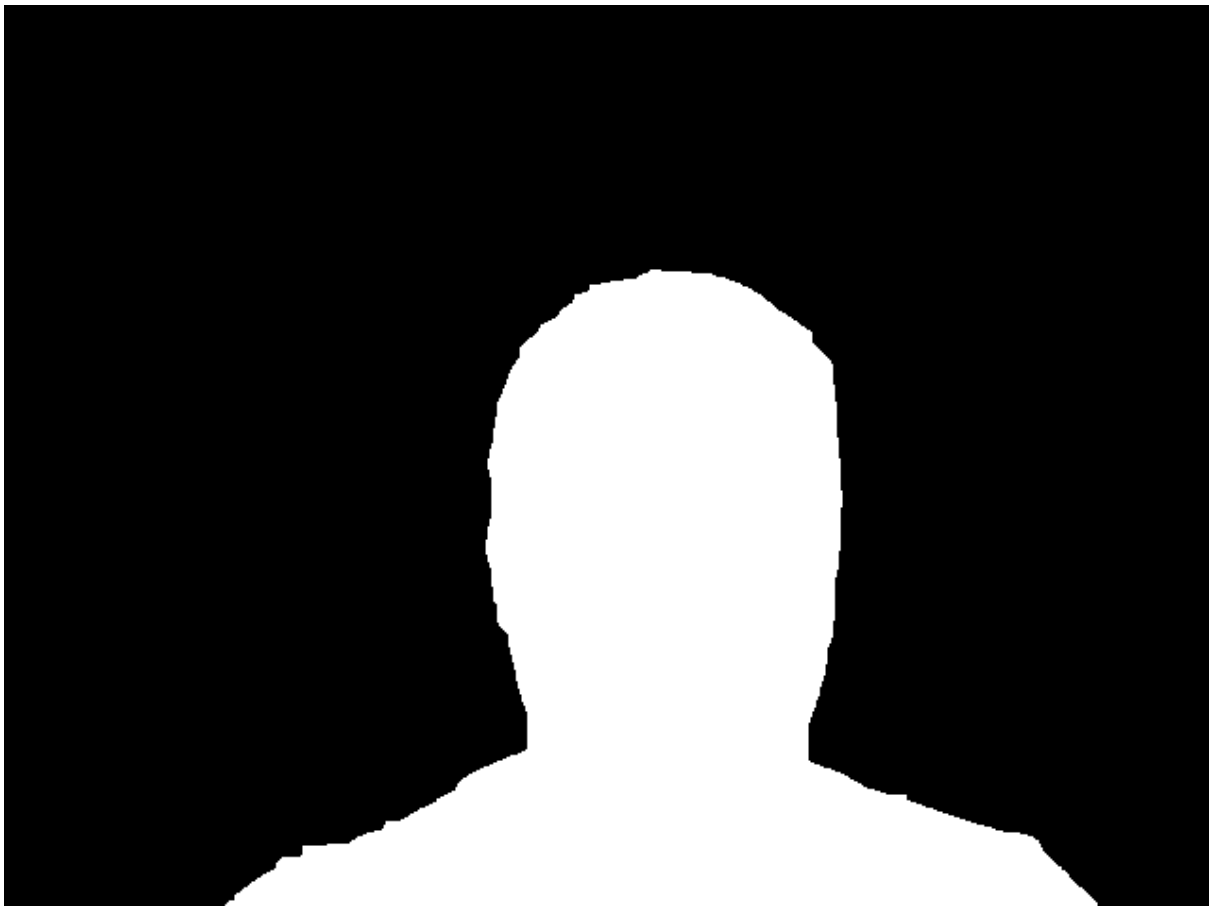
Morfološko zatvaranje smo prilagodili našem najčešćem slučaju, a to je da se korisnik našeg programa nalazi ispred kamere i da njegova glava zauzima centralnih 50% kadra. Zbog toga delimo frame na tri dela, sa strane po 25% i centralnih 50%. Ovo radimo da bi primenili morfološko zatvaranje sa različitom veličinom kernela za različite regije slike. U centralnom delu koristimo veći kernel 20x20 da bi potpuno zatvorili regije koje označavaju lice korisnika. Zbog početne pretpostavke da će se korisnik naći u centralnih 50% ovde koristimo veći kernel. Sa strane koristimo manji kernel jer je manja verovatnoća da se korisnik nadje u tom delu kadra, pa smanjujemo kernel da ne bi belu regiju koja predstavlja korisnikovo lice spojili sa nekom manjom regijom u pozadini koja predstavlja šum i koju treba zanemariti.

Postavlja se pitanje zašto uopšte radimo morfološko zatvaranje kada smo prethodno izdvojili masku i uklonili šum. Kroz testiranje algoritma, primetio sam da se često dešava da su vrlo loše detektovane promene svetline piksela u delovima lica kao što su jagodice i regija iznad obrva u odnosu na pozadinu. To je zato što na te delove lica pada najviše svetla i oni dobiju beličastu boju, pa kada se nalazimo ispred svetle pozadine, kao što je beli zid, algoritam zna da napravi grešku i da primeti veoma malu razliku izmedju referentne pozadine i trenutnog frejma. Prostije rečeno, pozadina je bela dok su naši najosvetljeniji delovi lica skoro beli, pa kada izvršimo oduzimanje koje smo obradili (Slika 10) razlika bude gotovo jednaka 0, što rezultuje crnim regijama na licu. Baš iz tih razloga radimo morfološko zatvaranje jer želimo da imamo foreground koji u sebi nema ’crne rupe’. Crne regije na licu koje se javljaju su prikazane na slici 27:



Slika 27

Nakon što izvršimo morfološka zatvaranja sa različitim kernelima u zavisnosti od dela kadra treba nazad sastaviti ovako tri izdvojena dela u foreground masku. Kada sastavimo nazad masku postavljamo sve piksele veće od 3 na vrednost 255 (bela boja) čime zapravo binarizujemo sliku što znači da imamo samo bele i crne delove slike (Slika 28). Ovako dobijena maska, binarizovana, je ključna za segmentiranje foreground-a od pozadine.



Slika 28

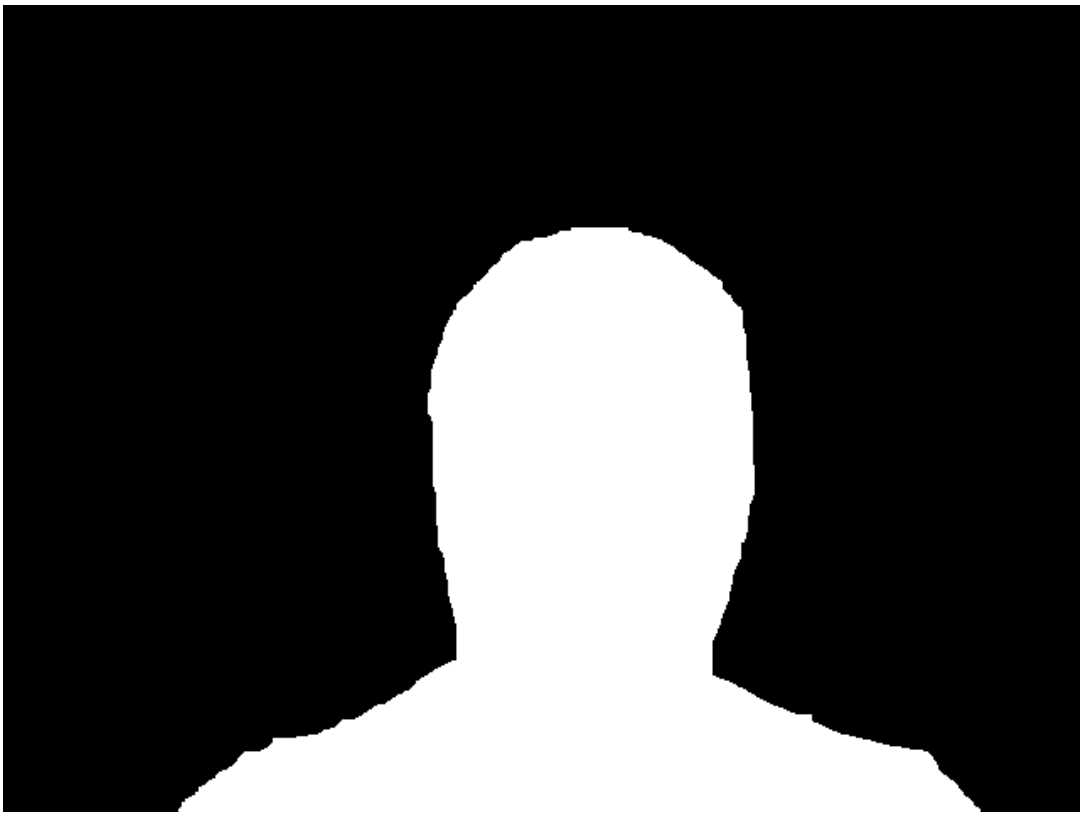
Detektovanje kontura

Sada kada imamo dobijenu binarizovanu foreground sliku, sledeći korak je detektovanje svi kontura na fgMask-u i čuvanje samo najveće. Može se desiti da smo posle svog procesiranja koje smo obradili na foreground-u našli nekoliko morfološki zatvorenih celina. Takodje, možemo usvojiti pretpostavku da će korisnik našeg programa, koji stoji ispred web kamere, zapravo biti najveći objekat koji se pojavio u odnosu na referentnu pozadinu, pritom za nas najbitniji da se dobro segmentira. Prema tome u sledećem delu algoritma ćemo detektovati sve konture koje se javljaju na prednjem delu slike i sačuvati samo najveću.

```
contours, _ = cv.findContours(fgMask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
if contours:
    largest_contour = max(contours, key=cv.contourArea)
    fgMask = np.zeros_like(fgMask)
    cv.drawContours(fgMask, [largest_contour], -1, (255), thickness=cv.FILLED)
```

Slika 29

Nakon primene ovih operacija nad trenutnim frejmom završili smo sa procesiranjem slike i kao izlaz dobijamo konačnu foreground masku, kao binarizovanu sliku, koja nam služi da izdvojimo prednji deo kadra od pozadine. Na slikama 30, 31 i 32 su prikazane konačne maske za sva tri ambijenta koje smo pominjali u radu, gde prvo ide soba sa bež zidovima, sa belim i na kraju crna pozadina:



Slika 30

Slika 31

Slika 32

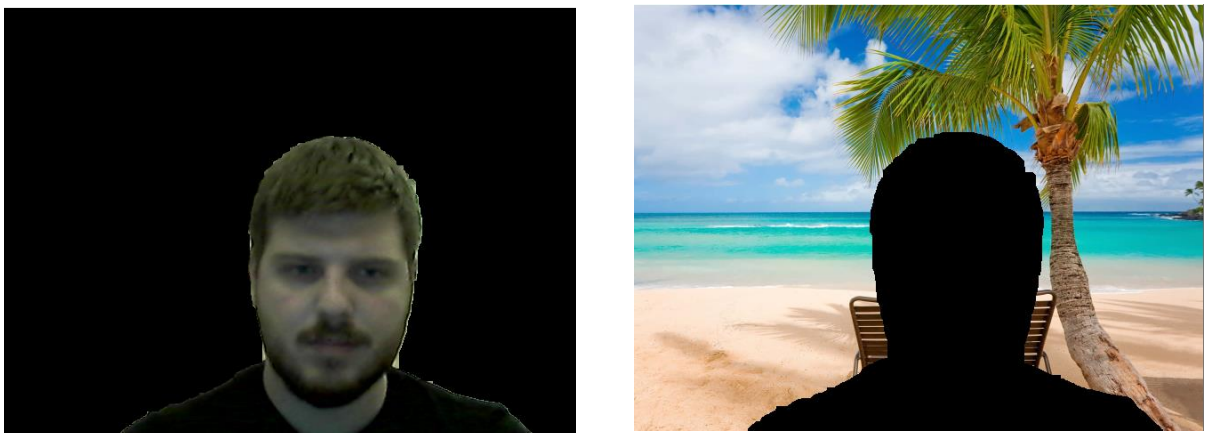
Kreiranje foreground i background maske

Sada smo prošli kroz sve korake procesiranja slike i dobili binarizovanu sliku prednjeg dela kadra koju ćemo sada iskoristiti za distinkciju foreground-a od pozadine kao i za segmentaciju. Ako posmatramo `fgMask` dobili smo binarizovanu sliku gde znamo da je to crno bela slika gde beli deo slike predstavlja foreground, a crni pozadinu. Ovo zapravo znači da vrlo lako možemo dobiti masku čije će bele površine označavati područja pozadine tako što na trenutnu masku primenimo operaciju bitsko ne koju pozivamo kao funkciju iz bibilioteke OpenCV kao `cv.bitwise_not()`. Tu masku nazivamo inverznom. Ovo znači da sada imamo masku za izdvajanje foreground-a i masku za izdvajanje background-a što nam govori da imamo dovoljno elemenata za izdvajanje prednjeg dela frejma od pozadine. Prelazimo iz grayscale boje nazad u RGB tako da dobijene maske možemo primeniti u sva tri kanala i pomoću operacije bitsko ne koju primenjujemo na originalnoj slici i na novoj, zamenskoj pozadini.

```
fgMask_inv = cv.bitwise_not(fgMask)
fgMask_inv = cv.cvtColor(fgMask_inv, cv.COLOR_GRAY2BGR)

fgMask = cv.cvtColor(fgMask, cv.COLOR_GRAY2BGR)

fgImage = cv.bitwise_and(img, fgMask)
bgImage = cv.bitwise_and(bg, fgMask_inv)

cv.imshow('fgImage', fgImage)
cv.imshow('bgImage', bgImage)
```

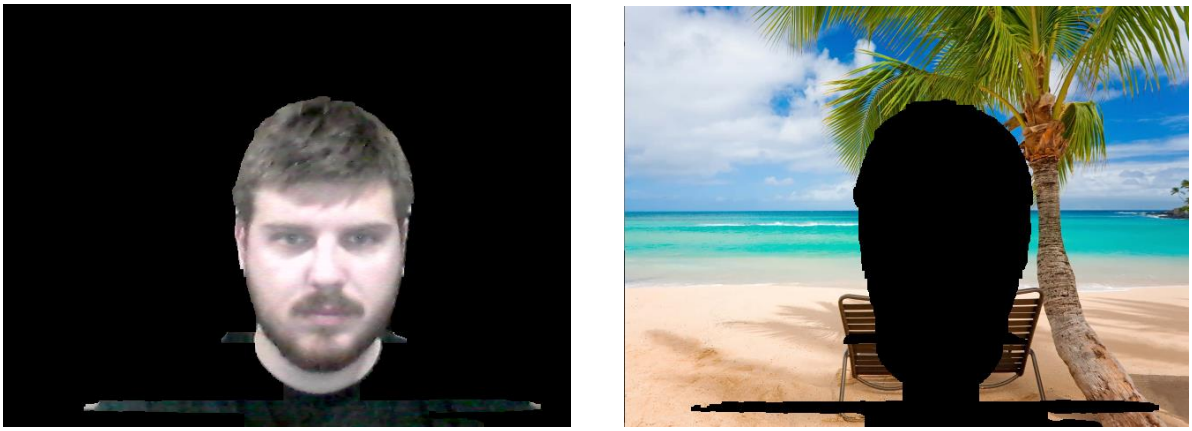
Slika 33

Na slikama 34, 35 i 36 prikazani su izdvojeni foreground i background za različite ambijente.



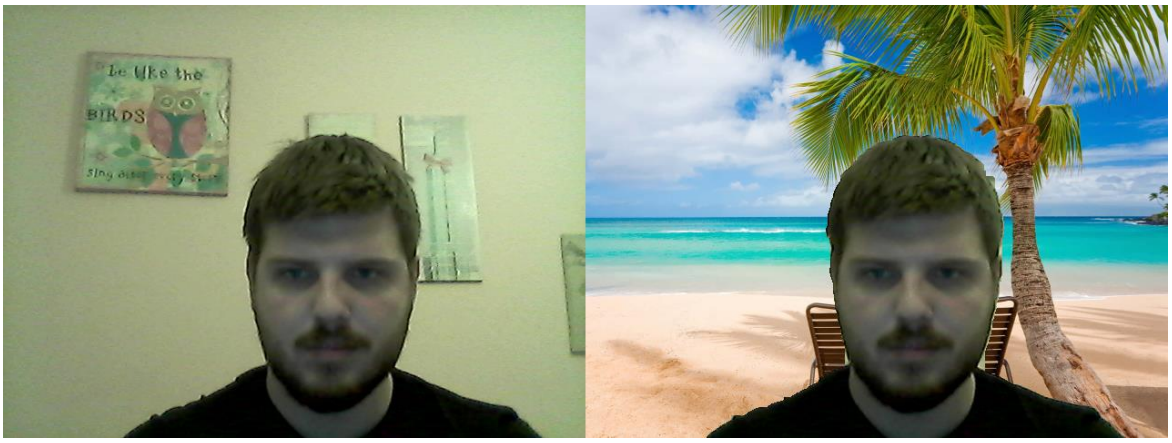
Slika 34

Slika 35



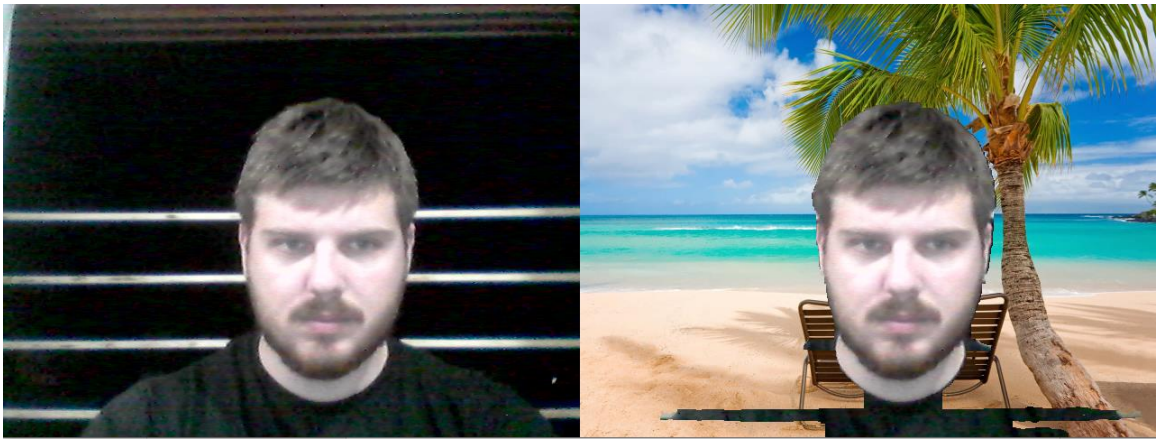
Slika 36

Sabiranjem segmentiranog prednjeg dela slike i segmentirane pozadine dobijamo krajnji rezultat koji sve vreme želimo. Vidimo da je uspešno izdvojen foreground od stvarne pozadine i da je ona zamenjena slikom plaže (Slika 37, 38, 39):



Slika 37

Slika 38



Slika 39

Upravljanje algoritmom

Na dnu same petlje imamo deo koda koji nam omogućava upravljanje programom tj. beleženje pozadine na klik tastera ’e’, ponovni pokušaj beleženja pozadine na taster ’r’ kao i prekid rada algoritma pritiskom na taster ’q’ (Slika 40):

```
key = cv.waitKey(5) & 0xFF
if ord('q') == key:
    break
elif ord('e') == key:
    takeBgImage = 1
    print("Background Captured")
elif ord('r') == key:
    takeBgImage = 0
    print("Ready to Capture new Background")
```

Slika 40

Ceo algoritam

```
import cv2 as cv
import numpy as np
import cvzone

# Iniciranje video zapisivanja
video = cv.VideoCapture(0, cv.CAP_DSHOW)

video.set(cv.CAP_PROP_AUTO_EXPOSURE, 0.25)
video.set(cv.CAP_PROP_EXPOSURE, -4)


# Ucitavanje nove pozadinske slike
newBgImage = cv.imread("summer-beach-background.jpg")
if newBgImage is None:
    print("Error: Could not load background image.")
    exit()

# Zabeležavanje početnog frame-a kao referente pozadine
ret, bgReference = video.read()

def resize(dst, img):
    width = img.shape[1]
    height = img.shape[0]
    dim = (width, height)
    resized = cv.resize(dst, dim, interpolation=cv.INTER_AREA)
    return resized

def adjust_brightness(image, brightness_increment=50, threshold=110):
    # Kreiramo masku koja pamti vrednosti piksela koji su veci od praga
    mask = image > threshold

    brightness_matrix = np.full(image.shape, brightness_increment,
dtype=np.uint8)

    # Povecavamo vrednosti piksela za one piksele za koje je maska jednaka
True
    brightened_image = np.where(mask, cv.add(image, brightness_matrix), image)

    return brightened_image


takeBgImage = 0

while True:
    ret, img = video.read()
    if not ret:
```

```
        print("Error: Could not read frame from video source.")
        break

    # Resize nove pozadine koja odgovara veličini video frejma
    if newBgImage.shape[:2] != bgReference.shape[:2]:
        bg = resize(newBgImage, bgReference)
    else:
        bg = newBgImage

    if takeBgImage == 0:
        bgReference = img
        bgReference = adjust_brightness(bgReference)


    cv.imshow("bgReference", bgReference)

    # Kreiranje maske
    diff1 = cv.subtract(img, bgReference)
    diff2 = cv.subtract(bgReference, img)

    cv.imshow("diff1", diff1)
    cv.imshow("diff2", diff2)

    diff = cv.add(diff1, diff2)
    diff[abs(diff) < 60.0] = 0

    cv.imshow("diff", diff)

    gray = cv.cvtColor(diff, cv.COLOR_BGR2GRAY)
    gray[np.abs(gray) < 10] = 0
    fgMask = gray

    cv.imshow("fgMask", fgMask)



    # Uklanjanje suma
    kernel = np.ones((3, 3), np.uint8)
    fgMask = cv.erode(fgMask, kernel, iterations=2)
    fgMask = cv.dilate(fgMask, kernel, iterations=2)

    cv.imshow("gray fgMask", fgMask)


    # Delimo sliku na tri dela: 25%, 50%, 25%
    height, width = fgMask.shape
    left_part = fgMask[:, :width//4]
    middle_part = fgMask[:, width//4:3*width//4]
    right_part = fgMask[:, 3*width//4:]
```

```
    small_kernel = np.ones((3, 3), np.uint8)
    large_kernel = np.ones((20, 20), np.uint8)

    left_part = cv.morphologyEx(left_part, cv.MORPH_CLOSE, small_kernel)
    middle_part = cv.morphologyEx(middle_part, cv.MORPH_CLOSE, large_kernel)
    right_part = cv.morphologyEx(right_part, cv.MORPH_CLOSE, small_kernel)

    # Spajamo nazad delove
    fgMask[:, :width//4] = left_part
    fgMask[:, width//4:3*width//4] = middle_part
    fgMask[:, 3*width//4:] = right_part

    fgMask[fgMask > 3] = 255

    cv.imshow("fgMask after morphologyEx", fgMask)


    contours, _ = cv.findContours(fgMask, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
    if contours:
        largest_contour = max(contours, key=cv.contourArea)
        fgMask = np.zeros_like(fgMask)
        cv.drawContours(fgMask, [largest_contour], -1, (255),
thickness=cv.FILLED)

    cv.imshow("Foreground Mask", fgMask)

    fgMask_inv = cv.bitwise_not(fgMask)
    fgMask_inv = cv.cvtColor(fgMask_inv, cv.COLOR_GRAY2BGR)

    fgMask = cv.cvtColor(fgMask, cv.COLOR_GRAY2BGR)

    fgImage = cv.bitwise_and(img, fgMask)
    bgImage = cv.bitwise_and(bg, fgMask_inv)

    cv.imshow('fgImage', fgImage)

    # Kombinovanje foreground-a i novog backgrounda-a
    bgSub = cv.add(bgImage, fgImage)

    imgStacked = cvzone.stackImages([img, bgSub], 2, 1)
    cv.imshow('BG Subtraction', imgStacked)
    cv.imshow('Background Removed', bgSub)
    cv.imshow('Original', img)

    key = cv.waitKey(5) & 0xFF
    if ord('q') == key:
        break
```

```
    elif ord('e') == key:
        takeBgImage = 1
        print("Background Captured")
    elif ord('r') == key:
        takeBgImage = 0
        print("Ready to Capture new Background")

cv.destroyAllWindows()
video.release()
```

Prednosti i mane algoritma

Jedna od glavnih prednosti našeg algoritma jeste njegova jednostavnost. Sama ideja na kojoj ceo algoritam počiva je krajnje intuitivna – poredimo kadar bez korisnika u njemu i kadar kada je on tu i na osnovu toga izolujemo razlike izmedju dve slike. Takodje struktura algoritma je jako jednostavna, ceo kod se nalazi u jednoj while beskonačnoj petlji što znači da neko ko ga analizira može lako da se snadje u njemu. Pored toga to znači da je lak za nadogradnju i unapredjenje po jedinstvenim potrebama koje korisnik ima na osnovu okruženja u kome se nalazi - osvetljenja koje ima, kvalitet kamere njegovog laptopa i slično.

Što se tiče mana algoritma jedna od glavnih je nemogućnost da se izdvoji referenta pozadina bez njenog eksplicitnog uslikavanja, a kao što znamo tako ne rade moderni algoritmi za uklanjanje pozadine. Ovo unosi dodatan korak pre samog početka rada algoritma gde se korisnik najpre mora skloniti iz kadra što definitivno nije praktično. Takodje, algoritam radi na osnovu boja i svetline piksela što uvodi odredjenu dozu relativiteta prilikom rada algoritma koja se tiče samog osvetljenja prostorije u kojoj se korisnik nalazi, kvalitet kamere koju ima na laptopu i slično. Trenutni parametri algoritma su podešeni za kameru mog laptop-a što ne znači da će sve raditi savršeno na drugom, koji ima bolju ili lošiju kameru. Ipak korisnik vrlo lako može prilagoditi parametare kao što su prag za prilagodjenje svetline pozadine (Slika 7), prag za uklanjanje šuma (Slika 17) i širine kernela za porfološko zatvaranje (Slika 26). Ovi parametri su bitni za kvalitet segmentacije i prilagodjavanje algoritma različitim okruženjima.

Potencijalna unapredjenja algoritma

Sada kada smo prošli kroz ceo algoritam i detaljno objasnili kako funkcioniše, jasno je da postoji prostora za unapredjivanje kako bi dobili bolje performanse i još bolju i primenjiviju segmentaciju pozadine. S obzirom na to da radimo segmentaciju na osnovu boja piksela dva frejma jasno je da smo u startu ograničeni što se performansi tiče jer mnogo zavisimo od boje, što znači da zavisimo od kvaliteta kamere i osvetljenja u sobi. Iako imamo ta ograničenja, algoritam se pokazao primenljivim, ali naravno da postoji prostora za unapredjivanje. Nekoliko tehnika koje možemo primeniti za unapredjivanje, a koje su mi pale na pamet, ćemo opisati u sledećem delu rada.

Najveća mana našeg algoritma je to što korisnik mora da uslika referentnu pozadinu pre nego što krene sa radom,a zapravo znamo da moderni algoritmi koji rade istu stvar ne

funkcionišu tako. Neke od naprednijih algoritama za dobijanje referentne pozadine su miks gausa i KNN. Ovi algoritmi već postoje u okviru biblioteke OpenCV i aktiviramo ih pozivom funkcija `cv.createBackgroundSubtractorMOG2` i `cv.createBackgroundSubtractorKNN`. Ovi algoritmi su pre svega namenjeni za subtrakciju pozadine u dinamičnim okruženjima pa bi njihova primena u našem slučaju bila znatno komplikovanije od samih poziva funkcija. Ovi algoritmi odlično detektuju promene u kadru ali problem koji se javlja jeste odredjena statičnost osobe koja stoji ispred kamere i oni nakon nekoliko sekundi počinju da tumače korisnika kao pozadinu zbog svoje trenutne statičnosti. Ipak testiranjem ovih algoritama primetio sam da se oni mogu upotrebiti za prikaz siluete čoveka ispred kamere pa me to saznanje dovodi do zaključa da bi se ovi algoritmi mogli iskoristiti za dinamičko odredjivanje referentne pozadine, bez njenog posebnog uslikavanja. Takodje algoritmi koji detektuju ivice na snimku daju slične rezultate kao pomenuta dva algoritma pa su i oni zanimljivi za razmatranje. Iako sam sve ove ideje i sam testirao i pokušavao da ih implementiram, nisam u tome do kraja uspeo, tj. nisam dobio prihvatljive rezultate, pa ovaj izazov ostavljam na razmatranje čitaocu rada.

Već smo pomenuli da je informacija o boji ta koja nam daje mogućnost za uklanjanje pozadine pa je dobro razmotriti uvodjenje algoritama i metoda koji će naš algoritam učiniti što otpornijim na varijable kao što su kvalitet kamere, osvetljenje u prostoriji, boja pozadine, ugradjeni mehanizmi za poboljšavanje slike kao što su prilagodjavanje boje i slično. Usavršavanjem ovog dela algoritma učinili bi da naš algoritam radi gotovo identično dobro u svim uslovima.

Reference

https://en.wikipedia.org/wiki/Foreground_detection - jun 2024.

https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html - jul 2024.

https://docs.opencv.org/4.x/dd/d4d/tutorial_js_image_arithmetics.html - jul 2024.