

**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I
INFORMACIJSKIH TEHNOLOGIJA OSIJEK**

Sveučilišni diplomski studij

**CRTANJE PUTEM WEB KAMERE I DETEKCIJA
NACRTANIH OBLIKA**

Obrada slike i računalni vid

Karla Fehir

Osijek, 2023.

SADRŽAJ

1. UVOD	1
2. PREGLED PODRUČJA I PROBLEMATIKE	2
2.1. Računalni vid.....	2
2.1. Korištene tehnologije i alati.....	3
2.1.1. Python.....	3
2.1.2. NumPy.....	3
2.1.3. OpenCV.....	3
3. OPIS ZADATKA S DOBIVENIM REZULTATIMA	5
3.1. Opis zadatka s dobivenim rezultatima.....	5
3.1.1. Pronalaženje HSV vrijednosti ciljanog markera	5
3.1.2. Crtanje pomoću definiranih HSV vrijednosti.....	6
3.1.3. Detekcija nacrtanih oblika.....	9
3.2. Evaluacija zadatka	10
3.2.1. Pronalaženje vlastite HSV vrijednosti markera.....	10
3.2.2. Crtanje preko Web kamere pomoću vlastitih vrijednosti	12
3.2.3. Detekcija nacrtanih oblika.....	12
4. ZAKLJUČAK	14
5. LITERATURA	15

1. UVOD

U ovom projektnom zadatku izrađen je „Air Canvas“ koji na sebi može crtati bilo što tako da kamerom detektiramo kretanje markera u boji te detektirati nacrtani oblik. Za realizaciju virtualnog markera koristit ćemo programski jezik Python te njegove biblioteke OpenCV i NumPy. Preferirani jezik je Python zbog njegovih biblioteka i sintakse koja je jednostavna za korištenje, ali uz razumijevanje osnova može se implementirati u bilo koji jezik koji podržava OpenCV.

Ovdje se koriste detekcija boja i praćenje kako bi se postigao cilj. Oznaka u boji se detektira i proizvodi se maska. Uključuje daljnje korake morfoloških operacija na proizvedenoj maski, a to su erozija i dilatacija. Erozija smanjuje nečistoće prisutne u maski, a dilatacija dodatno obnavlja erodiranu glavnu masku.

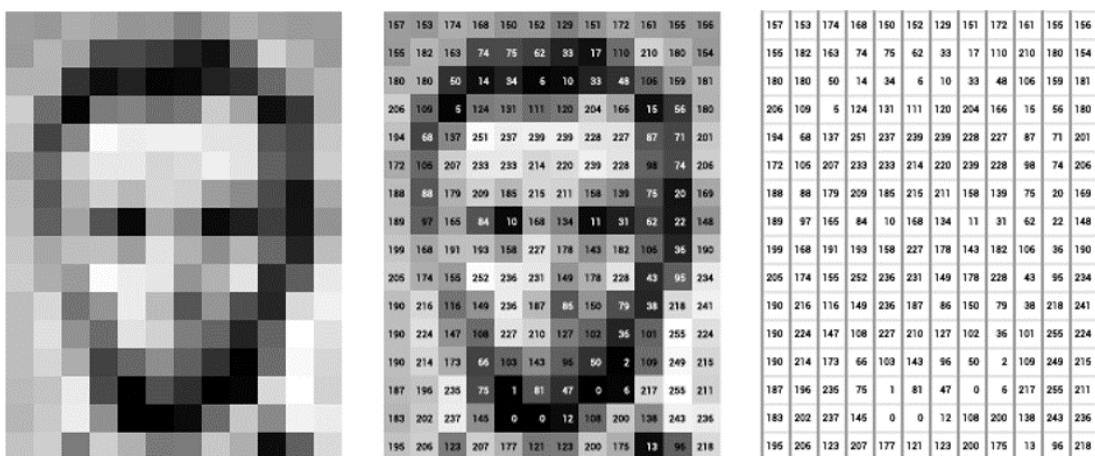
2. PREGLED PODRUČJA I PROBLEMATIKE

2.1. Računalni vid

Računalni vid [1] je područje računarstva u kojem se analizom i obradom slike može doći do informacija koje mogu pomoći donositi odluke o ponašanju tehničkog sustava. Na primjer, može dati odgovore na pitanja kao što su: nalazi li se na slici čovjek ili automobil ili koja je pozicija automobila. Pod obradom slike podrazumijeva se obrada digitalnih podataka u obliku slika i videa iz kojih je moguće dobiti informaciju o obliku, poziciji, orijentaciji, veličini, dimenziji i boji. Cilj je doći do informacija kao što su detekcija objekta, udaljenost objekta, boja, pozicija i slično. Obrada slike lak je vizualni zadatak za ljudski vid, međutim ovaj zadatak nije lak i smatra se izazovom za sustav koji se temelji na računalnom vidu jer ima visok stupanj varijabilnosti u svom izgledu. Između ostalog, kamera kao senzor daleko je lošija od ljudskog oka jer nema automatski fokus i prilagodbu osvjetljenju te često generira šum (eng. noise), a može imati i distorziju.

Na određenoj razini računalni vid je sve o prepoznavanju uzorka. Dakle, jedan od načina da naučite računalo kako razumjeti vizuelne podatke je da mu date slike, puno slika na tisuće, ako je moguće milijune koje su označene, a zatim ih podvrgnete različitim softverskim tehnikama ili algoritmima koji omogućuju računalu da ih pronađe uzorce u svim elementima koji se odnose na te oznake.

Na primjer, ispod je slika Abrahama Lincolna. Svjetlina svakog piksela na ovoj slici predstavljena je jednim 8-bitnim brojem u rasponu od 0 (crno) do 255 (bijelo). Ove brojeve softver vidi kada unesete sliku. Ti se podaci daju kao ulaz u algoritam računalnog vida koji će biti odgovoran za daljnju analizu i donošenje odluka.



Slika 1.1. Prikaz vrijednosti piksela slike

2.1. Korištene tehnologije i alati

2.1.1. Python

Python [2] je vrlo popularan programski jezik visoke razine, opće namjene. Programski jezik Python (najnoviji Python 3) koristi se u web razvoju, aplikacijama strojnog učenja, zajedno sa svim najnovijim tehnologijama u softverskoj industriji. Programski jezik Python vrlo je prikladan za početnike, također i za iskusne programere s drugim programskim jezicima kao što su C++ i Java.

Glavni skupovi alata [3] za obradu slika u pythonu su OpenCV, scikit-image i Pillow. Najopćenitije Python biblioteke (Numpy i Scipy) također pružaju neke alate za obradu slika. Sve te biblioteke mogu jednostavno komunicirati jedna s drugom zbog zajedničke upotrebe NumPy nizova za pohranu slike. Slika u sivim tonovima obično se pohranjuje u 2-dimenzionalnom nizu cijelih brojeva ili stvarnih vrijednosti Numpy s H redaka i W stupaca (W=širina, H=visina). Slika u boji pohranjuje se u 3-dimenzionalnom Numpy polju.

Projekt je izrađen u programskom jeziku Python. Za implementaciju virtualnog markera koriste se biblioteke OpenCV i NumPy koje su vrlo poznate u području računalnog vida.

2.1.2. NumPy

Numpy [4] je paket za obradu polja opće namjene. Pruža višedimenzionalni objekt niza visokih performansi i alate za rad s tim nizovima. To je temeljni paket za znanstveno računalstvo s Pythonom. Osim očite znanstvene upotrebe, Numpy se također može koristiti kao učinkovit višedimenzionalni spremnik generičkih podataka.

Polje u NumPy-ju je tablica elemenata (obično brojeva), svi iste vrste, indeksiranih nizom pozitivnih cijelih brojeva. U NumPy-ju, broj dimenzija niza naziva se rang polja. Korak cijelih brojeva koji daju veličinu niza duž svake dimenzije poznat je kao oblik niza. Klasa polja u NumPy-ju naziva se ndarray. Elementima u nizovima NumPy pristupa se pomoću uglatih zagrada i mogu se inicijalizirati pomoću ugniježđenih Python popisa.

2.1.3. OpenCV

OpenCV [5] je biblioteka funkcija koje se koriste za računalni vid, strojno učenje i obradu slika, a sada igra veliku ulogu u obradi u stvarnom vremenu, što je vrlo važno u današnjim sustavima. Njime se mogu obraditi slike i video za prepoznavanje predmeta, lica ili čak rukopisa čovjeka. Kad se integrira u razne biblioteke, poput Numpy, Python je sposoban

obraditi OpenCV strukturu za analizu. Za prepoznavanje uzorka slike i njenih različitih značajki preko OpenCV koristimo vektorski prostor i izvodimo matematičke operacije na tim značajkama. OpenCV je izdan pod BSD licencom i stoga je besplatan za akademsku i komercijalnu upotrebu. Postoje sučelja C++, C, Python, Java i podržava Windows, Linux, Mac OS, iOS i Android. Kada je dizajniran OpenCV, glavni fokus su bile aplikacije u realnom vremenu za računalnu učinkovitost. Kompletna arhitektura je napisana na optimiziranom C / C++ programskom jeziku kako bi se iskoristila prednost multi-core obrade.

3. OPIS ZADATKA S DOBIVENIM REZULTATIMA

3.1. Opis zadatka s dobivenim rezultatima

U ovom projektu koristimo Python biblioteku OpenCV za crtanje po ekranu pomoću virtualne olovke, tj. bilo koji marker se može koristiti za crtanje/brisanje koristeći tehniku otkrivanja kontura na temelju maske ciljnog markera željene boje.

3.1.1. Pronalaženje HSV vrijednosti ciljanog markera

Kao prvi korak moramo pronaći HSV raspon ciljanog markera te spremiti navedene vrijednosti u .npy datoteku. Prvo ćemo koristiti maskiranje boja kako bismo dobili masku naše obojene olovke/flomastera koristeći gornji HSV raspon. Zatim, pomoću otkrivanja kontura otkrivamo i pratimo lokaciju te olovke, tj. dobivamo x,y koordinate. Zatim crtamo liniju spajanjem x,y koordinata prethodne lokacije olovke (lokacije u prethodnom okviru) s novim x,y točkama. Također moramo definirati uređaj za snimanje. Ovdje koristimo kameru prijenosnog računala (0).

```
cap=cv2.VideoCapture(0)
```

Slika 3.1. Definiranje uređaja za snimanje

Kreiramo šest traka za praćenje kako bismo postavili vrijednosti gornje i donje granice H, S, V unutar novog prozora pod nazivom "HSV Trackbar"

```
cv2.namedWindow("HSV Trackbar")
cv2.createTrackbar("L-H","HSV Trackbar",0,179,nothing)
cv2.createTrackbar("L-S","HSV Trackbar",0,255,nothing)
cv2.createTrackbar("L-V","HSV Trackbar",0,255,nothing)
cv2.createTrackbar("U-H","HSV Trackbar",179,179,nothing)
cv2.createTrackbar("U-S","HSV Trackbar",255,255,nothing)
cv2.createTrackbar("U-V","HSV Trackbar",255,255,nothing)
```

Slika 3.2. Kreiranje trake za postavljanje HSV vrijednosti

Nadalje, čitamo okvire unutar beskonačne petlje u kojoj dohvaćamo ažurirane vrijednosti trake s HSV vrijednostima, izrađujemo masku pomoću ovog raspona i na kraju, izvodimo bitwise operaciju i prikazujemo rezultat.

```

while True:
    _, frame = cap.read()
    frame = cv2.resize(frame, (712, 400))
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    l_h=cv2.getTrackbarPos("L-H","HSV Trackbar")
    l_s=cv2.getTrackbarPos("L-S","HSV Trackbar")
    l_v=cv2.getTrackbarPos("L-V","HSV Trackbar")
    h_h=cv2.getTrackbarPos("U-H","HSV Trackbar")
    h_s=cv2.getTrackbarPos("U-S","HSV Trackbar")
    h_v=cv2.getTrackbarPos("U-V","HSV Trackbar")

    low=np.array([l_h,l_s,l_v])
    high=np.array([h_h,h_s,h_v])

    mask=cv2.inRange(hsv,low,high)
    result=cv2.bitwise_and(frame,frame,mask=mask)
    cv2.imshow("result",result)

```

Slika 3.3. Dohvaćanje HSV vrijednosti s trake i kreiranje maske

Pritiskom tipke 's' na tipkovnici spremamo trenutne postavke HSV vrijednosti raspona u datoteku marker_values.npy i prekidamo beskonačnu petlju. Ovo će stvoriti novu datoteku marker_values.npy i izaći iz programa. Na kraju gasimo sve otvorene prozore.

```

key = cv2.waitKey(1)
if key == ord('s'):
    thearray = [[l_h,l_s,l_v],[h_h, h_s, h_v]]
    np.save('marker_values',thearray)
    break
if key == 27:
    break

cap.release()
cv2.destroyAllWindows()

```

Slika 3.4. Spremanje HSV vrijednosti u .npy datoteku

3.1.2. Crtanje pomoću definiranih HSV vrijednosti

Kao drugi korak koristiti ćemo definirane HSV vrijednosti iz datoteke marker_values.npy za crtanjem markera.

Pravimo novu klasu pod nazivom drawingCanvas u kojoj ćemo definirati sve potrebne funkcije. Prvo definiramo `__init__()` i inicijaliziramo potrebne varijable. U ovom koraku pozivamo i funkciju `draw()`.

```
class drawingCanvas():
    def __init__(self):
        self.marker_values = np.load('marker_values.npy')
        self.cap = cv2.VideoCapture(0)
        self.canvas = None
        self.x1, self.y1=0,0
        self.val=1
        self.draw()
```

Slika 3.5. Inicijalizacija potrebnih varijabli

Definiramo funkciju `draw()` u kojoj čitamo okvir s kamere unutar beskonačne petlje i okrećemo okvir vodoravno. Zatim kreiramo masku pozivanjem funkcije `CreateMask()` te otkrivamo konture pomoću maske. Crtamo po platnu i prikazujemo rezultate. Čitamo unos s tipkovnice, pohranjujemo ga u `k` i pozivamo funkciju `takeAction()`. Pritiskom tipke 'Esc' događa se prekid i izlaz iz programa.

```
def draw(self):
    while True:
        _, self.frame = self.cap.read()
        self.frame = cv2.flip( self.frame, 1 )
        self.frame = cv2.resize(self.frame, (712, 400))
        if self.canvas is None:
            self.canvas = np.zeros_like(self.frame)

        mask=self.CreateMask()
        contours=self.Contours(mask)
        self.drawLine(contours)
        self.shapes = self.ShapeDetection(self.canvas, self.frame)
        self.display()

        k = cv2.waitKey(1) & 0xFF
        self.takeAction(k)
        if k == 27:
            break
```

Slika 3.6. Prikaz funkcije `draw()`

Definiramo funkcije `CreateMask()` i `ContourDetect()` koje su prikazane na slici 3.7.

```

def CreateMask(self):
    hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)
    lower_range = self.marker_values[0]
    upper_range = self.marker_values[1]
    mask = cv2.inRange(hsv, lower_range, upper_range)
    return mask

def Contours(self,mask):
    contours, h = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    return contours

```

Slika 3.7. Prikaz funkcija CreateMask() i ContourDetect()

Zatim definiramo funkciju drawLine() u kojoj na temelju trenutne i prethodne pozicije markera crtamo liniju spajanjem dviju koordinata. Također definiramo centar detektiranog markera i označujemo ga s kružnicom.

```

def drawLine(self,contours):
    if contours and cv2.contourArea(max(contours, key = cv2.contourArea)) > 100:
        c = max(contours, key = cv2.contourArea)
        x2,y2,w,h = cv2.boundingRect(c)

        # Find center point of the contour
        M = cv2.moments(c)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        cv2.circle(self.frame, (cX, cY), 10, (0, 0, 255), 2)

        if self.x1 == 0 and self.y1 == 0:
            self.x1,self.y1= x2,y2
        else:
            self.canvas = cv2.line(self.canvas, (self.x1,self.y1),(x2,y2), [255*self.val,255,255], 10)
            self.x1,self.y1= x2,y2
    else:
        self.x1,self.y1 = 0,0

```

Slika 3.8. Prikaz funkcije drawLine()

Definiramo funkciju display() u kojoj se prikazuje platno i okvir s linijama.

```

def display(self):
    self.frame = cv2.add(self.frame,self.canvas)
    cv2.imshow('frame',self.frame)
    cv2.imshow('canvas',self.canvas)
    cv2.imshow('shapes',self.shapes)

```

Slika 3.9. Prikaz funkcije display()

Definiramo takeAction() za izvođenje različitih operacija na temelju unosa s tipkovnice. Pritiskom na tipku 'e' prebacujemo se u način rada gumice za brisanje. Ovo će crtati crnom

bojom kada pomaknemo marker i na neki način može izvesti radnju brisanja na platnu. Također možemo očistiti platno pritiskom tipke 'c'.

```
def takeAction(self,k):
    # When c is pressed clear the entire canvas
    if k == ord('c'):
        self.canvas = None
    #press e to change between eraser mode and writing mode
    if k==ord('e'):
        self.val= int(not self.val)
```

Slika 3.10. Prikaz funkcije takeAction()

To je sve što nam treba da napravimo vlastiti virtualni marker za zaslon koristeći OpenCV i bilo koji marker/objekt u boji koji leži uokolo.

3.1.3. Detekcija nacrtanih oblika

Kako bismo detektirali oblik nacrtan markerom, koristit ćemo aproksimaciju konture. Aproksimacija kontura [6] je algoritam za smanjenje broja točaka u krivulji, općenito poznat kao Ramer-Douglas-Peuckerov algoritam ili algoritam split-and-merge. Aproksimacija konture temelji se na pretpostavci da se krivulja može aproksimirati nizom kratkih segmenata linije. To dovodi do rezultirajuće aproksimirane krivulje koja se sastoji od podskupa točaka koje su definirane izvornom krivuljom. Aproksimacija konture zapravo je već implementirana u OpenCV putem metode cv2.approxPolyDP. Kako bismo izvršili aproksimaciju konture, prvo izračunavamo opseg konture, a zatim konstruiramo stvarnu aproksimaciju konture. Uobičajene vrijednosti za drugi parametar za cv2.approxPolyDP obično su u rasponu od 1-5% opsega izvorne konture, u ovom slučaju koristimo 6% kako bi nacrtani oblici bili točno detektirani. Važno je razumjeti da se kontura sastoji od popisa vrhova. Na primjer, ako aproksimirana kontura ima tri vrha, detektiran je trokut, a ako ima četiri vrha onda je detektiran pravokutnik.

Nadalje računamo središte konture, nakon čega slijedi otkrivanje oblika i označavanje. Na kraju, crtamo konture i označeni oblik. Funkcija za detekciju oblika prikazana je na slici 3.11.

```

def ShapeDetection(self, image, frame):
    thresh_image = cv2.inRange(image, np.array([250, 250, 250]), np.array([255, 255, 255]))
    thresh_image = cv2.GaussianBlur(thresh_image, (5, 5), 0)
    contours, _ = cv2.findContours(thresh_image.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    result_image = np.zeros_like(image)

    for i, contour in enumerate(contours):
        epsilon = 0.06*cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, epsilon, True)

        x, y, w, h= cv2.boundingRect(approx)
        x_mid = int(x + (w/3))
        y_mid = int(y + (h/1.5))
        coords = (x_mid, y_mid)
        colour = (255, 0, 0)
        font = cv2.FONT_HERSHEY_DUPLEX

        if len(approx) == 3:
            cv2.putText(result_image, "Triangle", coords, font, 1, colour, 1)
        elif len(approx) == 4:
            cv2.putText(result_image, "Rectangle", coords, font, 1, colour, 1)

    return result_image

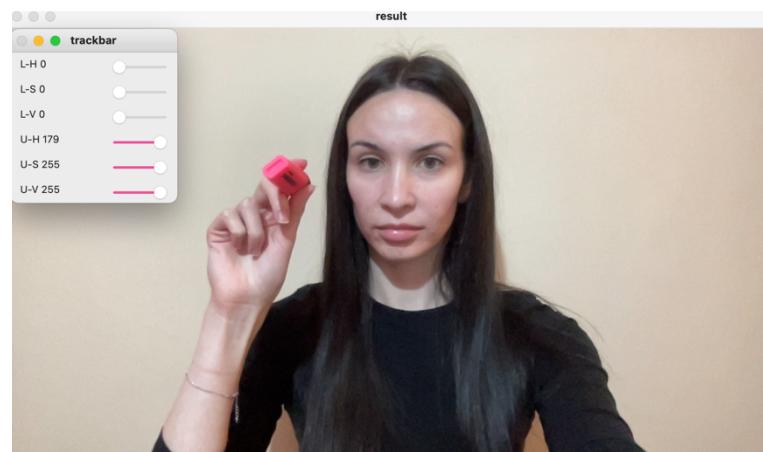
```

Slika 3.11. Prikaz funkcije ShapeDetection()

3.2. Evaluacija zadatka

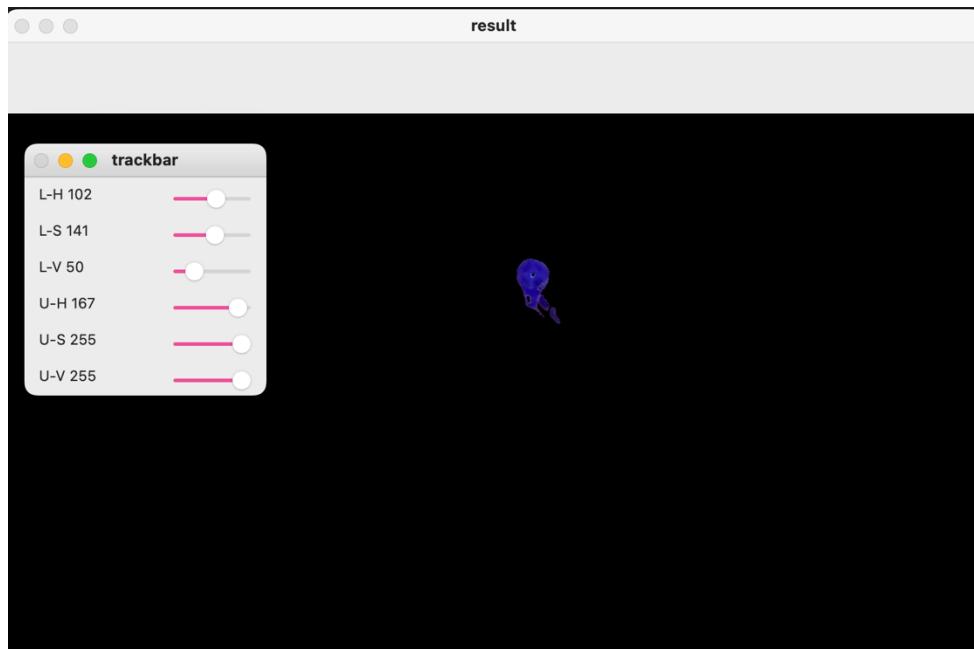
3.2.1. Pronalaženje vlastite HSV vrijednosti markera

Pokretanjem prvog Python programa SetMarkerHSV.py otvaraju se dva prozora, jedan sa snimkom Web kamere te drugi sa HSV trakom. Na slici 3.11 prikazano je početno stanje bez prilagođenih HSV vrijednosti markera.

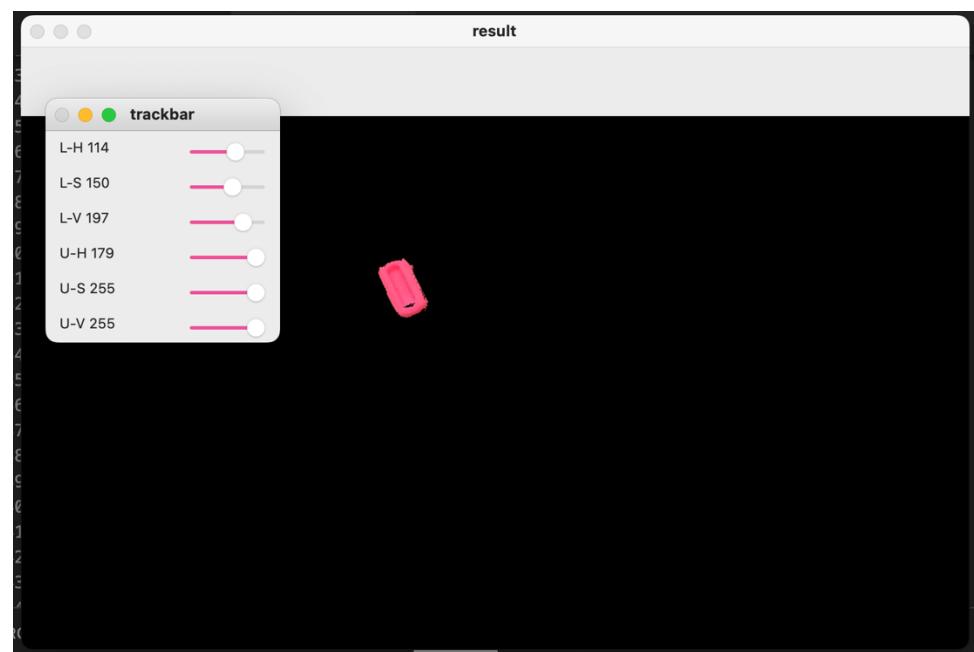


Slika 3.11. Početno stanje bez prilagođenih HSV vrijednosti markera

Na slikama 3.12. i 3.13. nalazi se prikaz Web kamere nakon prilagođenih HSV vrijednosti markera. Na prvoj slici smo koristili marker plave boje, a na drugoj marker roze boje.



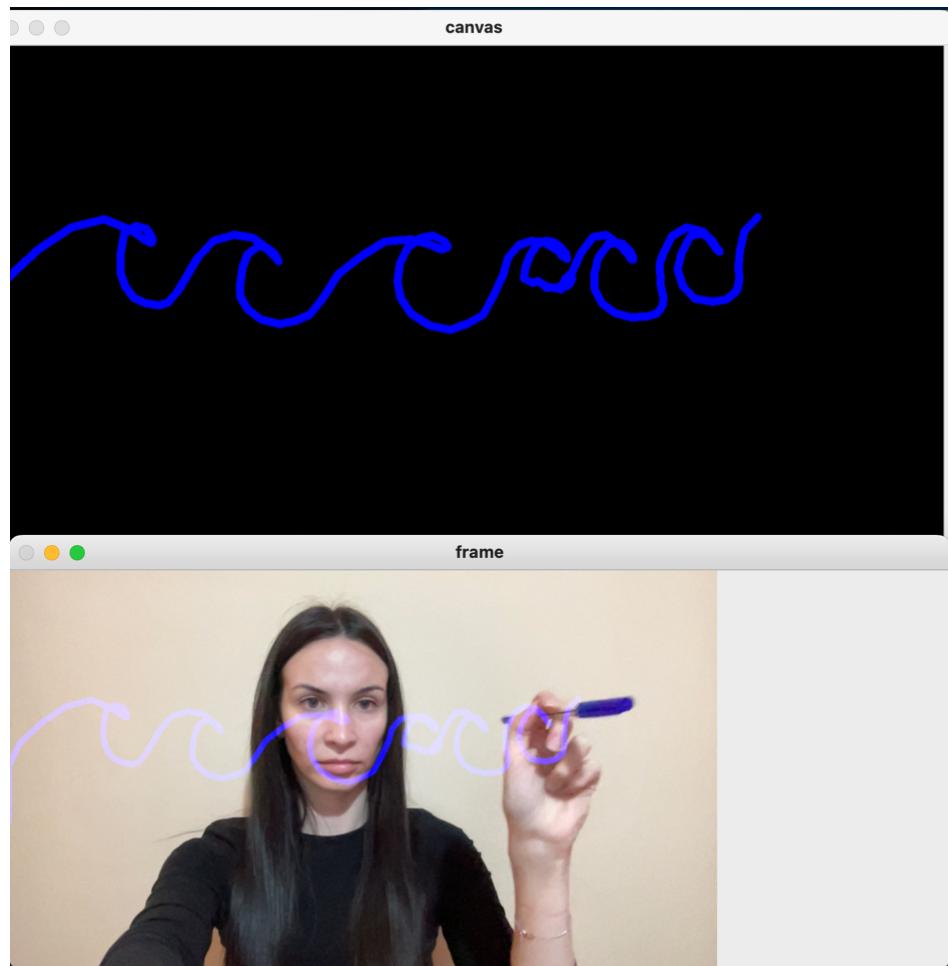
Slika 3.12. HSV vrijednosti plavog markera



Slika 3.13. HSV vrijednosti rozog markera

3.2.2. Crtanje preko Web kamere pomoću vlastitih vrijednosti

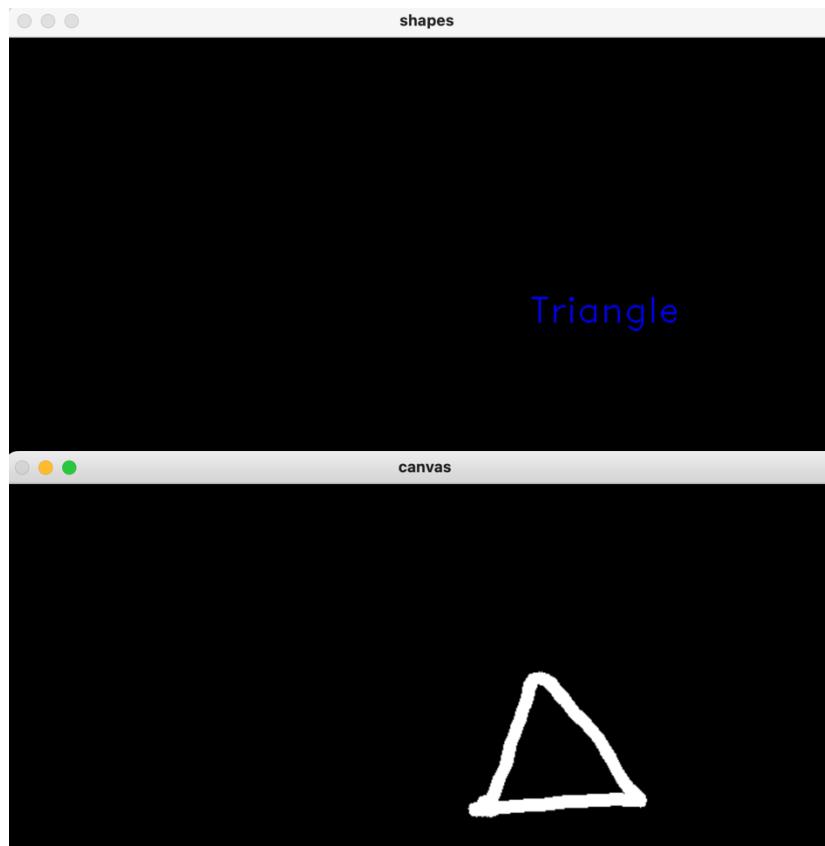
Nakon što smo spremili HSV vrijednosti naših markera u penrange.npy datoteku, pokrećemo program DetectShapes.py. Također se otvaraju dva prozora, jedan s prikazom Web kamere te drugi s prikazom platna s crtežom markera.



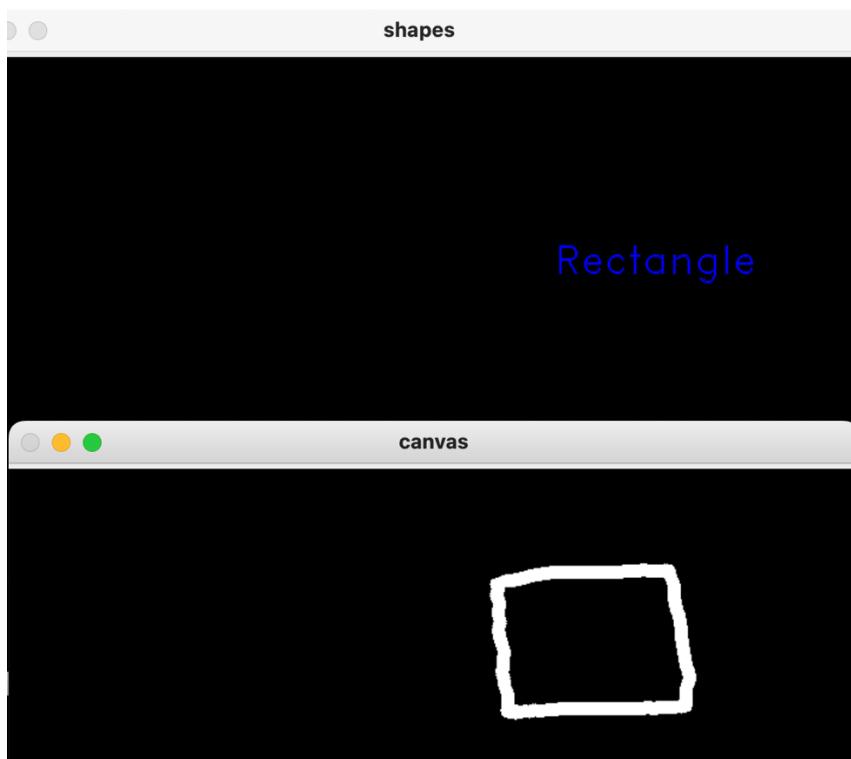
Slika 3.14. Prikaz crteža na platnu

3.2.3. Detekcija nacrtanih oblika

Pokretanjem programa DetectShapes.py otvara se i treći prozor 'shapes'. Ukoliko nacrtamo geometrijski oblik kao što je pravokutnik ili trokut, u prozoru 'shapes' biti će ispisani detektirani oblik. Na slikama 3.15 i 3.16 prikazani su detektirani oblici.



Slika 3.15 detekcija trokuta



Slika 3.16 detekcija pravokutnika

4. ZAKLJUČAK

Cilj ovog projektnog zadatka bio je upoznavanje s principom rada biblioteka OpenCV i NumPy te programskim jezikom Python. Kroz projekt susreli smo se s najpoznatijim metodama u području računalnog vida kako bi ostvarili crtanje preko Web kamere pomoću markera u boji.

5. LITERATURA

- [1] <https://www.ibm.com/topics/computer-vision>
- [2] <https://www.geeksforgeeks.org/python-programming-language/?ref=gcse>
- [3] https://virgilio0.github.io/Virgilio/inferno/computer-vision/introduction-to-computer-vision.html#_4-computer-vision-systems
- [4] <https://www.geeksforgeeks.org/python-numpy/?ref=gcse>
- [5] <https://www.geeksforgeeks.org/opencv-overview/>
- [6] <https://pyimagesearch.com/2016/02/08/opencv-shape-detection/>