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

Sveučilišni diplomski studij

Određivanje zbroja bačenih kockica

Seminarski rad

Josip Rizner

Osijek, 2022.

Sadržaj

1. UVOD

Kada čovjek pogleda sliku on odmah prepozna koji objekt se nalazi na slici, koje je boje i ima predodžbu o veličini objekta. S druge strane, računalo ne može tako lako prepoznati objekt sa slike ili prepoznati njegove atribute. Računalo obrađuje sliku i primjenjuje razne algoritme kako bi se izdvojile korisne informacije koje se nalaze na pojedinoj slici. Značajnim poboljšanjem u procesorskoj snazi modernih računala koja su u stanju rješavati zadatke obrade slike u stvarnom vremenu, napredovao je i računalni vid. Danas se primjene računalnog vida mogu pronaći u raznim poljima poput medicine, prehrambene industrije, vojne industrije, kontrole kvalitete u proizvodnji…

Cilj seminarskog rada je razviti algoritam koji će omogućiti unos slike s bačenim kockicama te obraditi sliku kako bi se dobio zbroj bačenih kockica. Programsko rješenje će se ispitati i analizirati za odgovarajuće ulazne podatke.

U drugom poglavlju se ulazi u područje i problematiku teme seminarskog rada. U trećem poglavlju opisan je zadata i dobiveni rezultati. Dodatno, u trećem poglavlju opisuju se korištene funkcije, objašnjava se implementacija u programskom jeziku te se ispituje ispravnost programskog rješenja.

1.1. Zadatak seminarskog rada

Razviti algoritam za čitanje zbroja brojeva na slici standardne kockice sa šest strana (koja ima točkice za brojeve). Na slici može biti više od jedne kockice i može biti slikana pod kutom, tako da se u isto vrijeme vidi više strana kockice. Potrebno je detektirati koja je gornja strana kockice i koliki je ukupni zbroj. Ispitati na vlastitim slikama i prikazati rezultate.

2. OPIS PODRUČJA I PROBLEMATIKE

Detekcija značajki vrlo je zahtjevan zadatak u području obrade slike i računalnog vida. Računala ne mogu razumijevati vizualne informacije kao i ljudi, koji će uočavati detalje i raspoznavati različite elemente na slikama. Iz tog razloga, računala obrađuju sliku pomoću raznih algoritama kako bi se izvukle potrebne značajke. U [1], [2] i [3] je navedeno kako je jedan od prvih koraka pretvaranje slike u nijanse sive boje (eng. Grayscale) te uklanjanje šuma kako bi se dobili bolji rezultati. Kada je slika pretvorena u sive tonove i kada je uklonjen šum, može se raditi detekcija rubova (eng. Edge detection) i thresholding, odnosno binarizacija slike. Nakon toga se može se puno lakše doći do željenih značajki.

3. OPIS ZADATKA I DOBIVENIH REZULTATA

U ovom poglavlju će prvo biti opisana *OpenCV* biblioteka i korištene funkcije, nakon toga će biti opisan rad algoritma i na kraju će biti prikazano par testnih slučajeva sa rezultatima.

3.1. OpenCV biblioteka

Kako je navedeno u [4], OpenCV (engl. Open Source Computer Vision) je biblioteka otvorenog koda koja sadrži programske funkcije koje su korištene u računalnom vidu i strojnom učenju. Usmjerena je na aplikacije koje obavljaju svoje funkcije u stvarnom vremenu. Napisana je u C i C++ programskim jezicima. Također, podržava korištenje na više operacijskih sustava, a to su: MS Windows, Linux, Mac OS i Android. Sadrži sučelja za C++, Python i MATLAB programske jezike. OpenCV biblioteka također sadrži i veliki broj optimiziranih algoritama, među kojima su algoritmi korišteni u računalnom vidu i strojnom učenju. Navedeni algoritmi korišteni su za detekciju objekata, praćenje detektiranih objekata, praćenje pokreta, spajanje više slika u jednu, itd.

3.2. Pretvorba prostora boja

Kako je opisano u [5], boje prisutne na bilo kojoj slici predstavljene su prostorima boja u OpenCV-u. Postoji nekoliko prostora boja od kojih svaki ima svoju važnost. Neki od osnovnih prostora boja su RGB (engl. Red, Green, Blue), CMYK (engl. Cyan, Magenta, Yellow, Key) i BGR (engl. Blue, Green, Red). Kada god postoji potreba za pretvaranje jednog prostora boja u drugi prostor boja, tada se koristi funkcija *cvtColor*. U OpenCV biblioteci postoji više od 150 kodova za pretvorbu prostora boja. Pretvorba prostora boja vrlo je korisna za rješavanje problema u području računalnog vida. Deklaracija funkcije *cvtColor* može se vidjeti na slici 3.1.

```
void cv::cvtColor ( InputArray  src,
                    OutputArray dst,
                    int         code,
                    int         dstCn = 0
                  )
```

Slika 3.1. *Deklaracija funkcije cvtColor, preuzeto s [5]*

3.3. Uklanjanje šuma

Digitalne slike sklone su raznim vrstama šuma. Šum je rezultat pogrešaka u procesu dobivanja slike, koje rezultiraju vrijednostima piksela koji ne odražavaju pravi intenzitet stvarne slike. Pri izradi ovog projektnog zadatka korištena je funkcija *GaussianBlur()*. U [7] se navodi kako je funkcija *GaussianBlur()* vrlo efikasno uklanja Gaussov šum. Pri pozivu ove funkcije potrebno je odrediti širinu i visinu kernela koja bi trebala biti pozitivna i neparna. Isto tako, treba odrediti standardnu devijaciju u smjeru X i Y pomoću parametara sigmaX i sigmaY. U slučaju kada je naveden samo sigmaX, sigmaY se uzima kao sigmaX. Na slici 3.2. može se vidjeti deklaracija funkcije *GaussianBlur* s opisom parametara.

Python: `cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst`

Parameters:
- **src** – input image; the image can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – Gaussian kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from `sigma*`.
- **sigmaX** – Gaussian kernel standard deviation in X direction.
- **sigmaY** – Gaussian kernel standard deviation in Y direction; if `sigmaY` is zero, it is set to be equal to `sigmaX`, if both sigmas are zeros, they are computed from `ksize.width` and `ksize.height`, respectively (see `getGaussianKernel()` for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of `ksize`, `sigmaX`, and `sigmaY`.
- **borderType** – pixel extrapolation method (see `borderInterpolate()` for details).

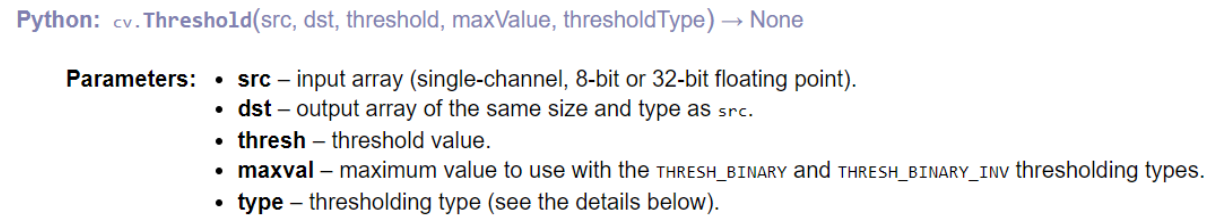
Slika 3.2. *Deklaracija funkcije GaussianBlur, preuzeto s [8]*

3.4. Binarizacija slike

Postupak binarizacije, odnosno pretvaranja slike u crno bijelu, relativno je jednostavan. Potrebno je odabrati prag binarizacije s kojim će se uspoređivati svaki piksel te sukladno rezultatu usporedbe pretvoriti u crni ili bijeli piksel. Za binarizacije slike korištene su dvije funkcije:

- *cv.threshold* s Otsu binarizacijom
- *cv.adaptive_threshold*

Prema [9], funkcija cv.threshold uspoređuje svaki piksel s vrijednosti praga binarizacije. Ako je vrijednost piksela manja od praga, postavlja se na 0, inače se postavlja na maksimalnu vrijednost. Prvi parametar koji funkcija prima je izvorna slika koja bi trebala biti slika u sivim tonovima. Drugi parametar je vrijednost praga koja se koristi za klasifikaciju vrijednosti piksela. Treći parametar je maksimalna vrijednost koja se dodjeljuje vrijednostima piksela koje prelaze prag. Četvrti predani parametar određuje tip binarizacije. Osnovni tipovi binarizacije su *cv.THRESH_BINARY, cv.THRESH_BINARY_INV*. Funkcija vraća dvije vrijednosti, prva je korišteni prag, a druga je crno bijela slika. Na slici 3.3 može se vidjeti deklaracija funkcije *threshold* s opisanim parametrima.



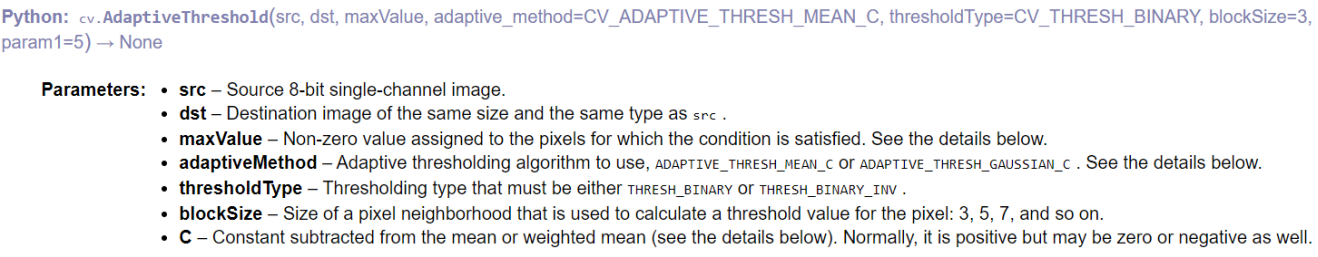
Slika 3.3. *Deklaracija funkcije Threshold, preuzeto s [10]*

Pri globalnom određivanju praga pomoću *Threshold* funkcije koriste se proizvoljno odabrana vrijednost kao prag. Nasuprot tome, Otsuova metoda izbjegava odabir vrijednosti i određuje je automatski. Otsuova metoda binarizacije određuje optimalnu globalnu vrijednost praga iz histograma slike. Otsuova metoda binarizacije koristi se pozivom funkcije *Threshold* i predajom *cv.THRESH_OTSU* kao dodatnog parametra.

Kada postavljanje globalne vrijednosti kao prag ne daje dobre rezultate, može se koristiti adaptivni *Threshold*. U [9] je navedeno kako funkcija *adaptiveThreshold* određuje prag za piksel na temelju vrijednosti piksela oko njega. Tako se dobiju različiti pragovi za različite dijelove iste slike što daje bolje rezultate za slike s različitim osvjetljenjem. Koriste se dvije metode za računanje praga:

- *cv.ADAPTIVE_THRESH_MEAN_C*
- *cv.ADAPTIVE_THRESH_GAUSSIAN_C*

Na slici 3.4. može se vidjeti deklaracija funkcije *adaptiveThreshold* s opisanim parametrima.



Slika 3.4. *Deklaracija funkcije adaptiveThreshold, preuzeto s [10]*

3.5. Pronalaženje kontura na slici

Konture na slici mogu se pronaći korištenjem funkcije *findContours*. Funkcija *findContours* otkriva promjenu boje na slici te promjenu označava kao konturu. Preporučeno je korištenje binariziranih slika kako bi se dobili najbolji rezultati. Na slici 3.5. može se vidjeti deklaracija funkcije *findContours* s opisanim parametrima.

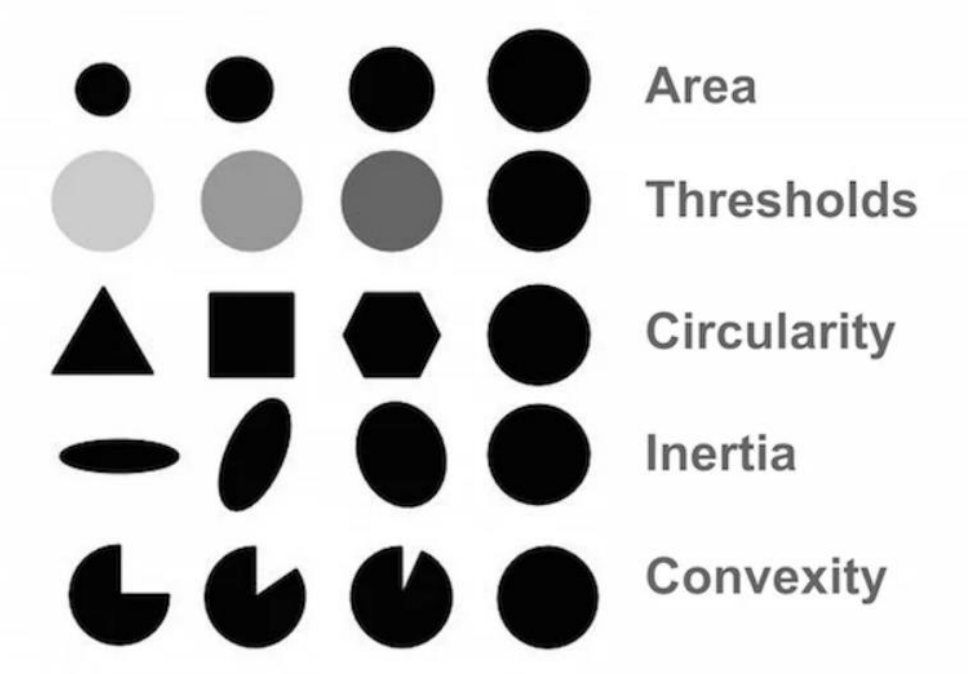
Python: `cv.FindContours(image, storage, mode=CV_RETR_LIST, method=CV_CHAIN_APPROX_SIMPLE, offset=(0, 0)) → contours`

Parameters:
- **image** – Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as `binary`. You can use `compare()`, `inRange()`, `threshold()`, `adaptiveThreshold()`, `Canny()`, and others to create a binary image out of a grayscale or color one. The function modifies the `image` while extracting the contours. If mode equals to `CV_RETR_CCOMP` or `CV_RETR_FLOODFILL`, the input can also be a 32-bit integer image of labels (`CV_32SC1`).
- **contours** – Detected contours. Each contour is stored as a vector of points.
- **hierarchy** – Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contour `contours[i]`, the elements `hierarchy[i][0]`, `hiearchy[i][1]`, `hiearchy[i][2]`, and `hiearchy[i][3]` are set to 0-based indices in `contours` of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour `i` there are no next, previous, parent, or nested contours, the corresponding elements of `hierarchy[i]` will be negative.
- **mode** –

Slika 3.5. *Deklaracija funkcije findContours, preuzeto s [10]*

3.6. Pronalaženje krugova na slici

Jedan od načina pronalaska krugova, odnosno mrlja (engl. blob) čiji oblik podsjeća na krug, je korištenjem *SimpleBlobDetector*-om. Prema [11], postavljanjem parametara *SimpleBlobDetector-a* može se jednostavno kontrolirati koje mrlje će biti pronađene, odnosno klasificirane kao mrlja. Osnovni parametri su površina, *threshold*, pravilnost kruga, inercija i konveksnost. Prikaz parametara te što se njima filtrira može se vidjeti na slici 3.6.



Slika 3.6. *Parametri SimpleBlobDetector-a, preuzeto s [11]*

3.7. Programsko rješenje

Prvi korak je učitavanje slike na kojoj se nalazi bačene igrače kockice. Slika se učitava pomoću *imread* funkcije. Funkcija prima putanju do mjesta gdje se slika nalazi. Na slici 3.7. može se vidjeti izvorna slika učitana za ovaj primjer.

Slika 3.7. *Izvorna slika*

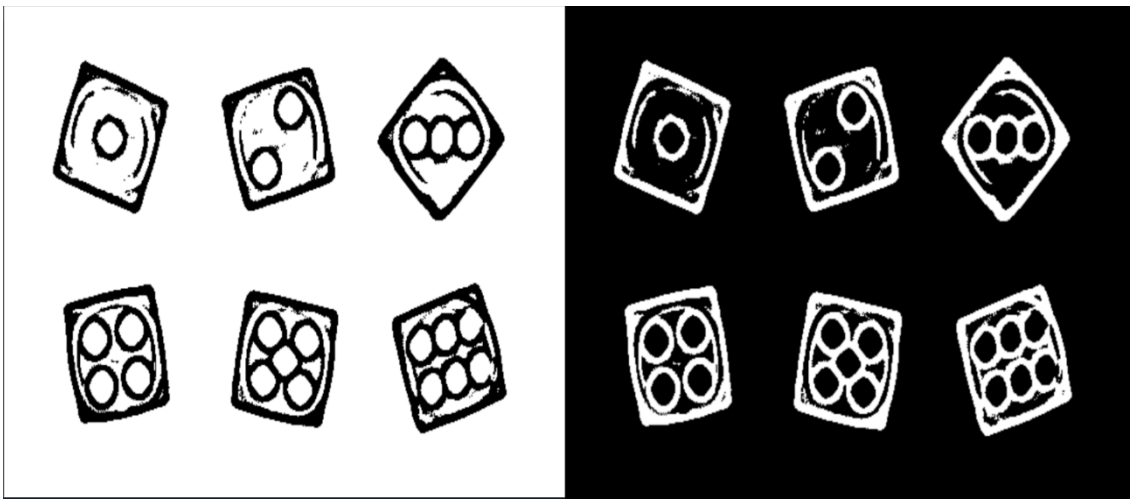
Nakon što je slika učitana, slika se pretvara iz BGR (engl. Blue, Green, Red) prostora boja u sive tonove pomoću *cvtColor* funkcije. Slika u sivim tonovima može se vidjeti na slici 3.8.

Slika 3.8. *Izvorna slika pretvorena u sive tonove*

Sada kada je slika u sivim tonovima, treba se zamutiti koristeći funkciju *GaussianBlur*. Zamućenjem slike uklonit će se neželjen šum i ublažiti će se svi „oštri“ rubovi na slici kako se kasnije ne bi dobili lažno pozitivni rezultati kada se budu tražile konture na slici. Na slici 3.8. može se vidjeti slika nakon primjene funkcije *GaussianBlur*.

Slika 3.9. *Izvorna slika nakon pretvorbe u sive tonove i zamućivanja*

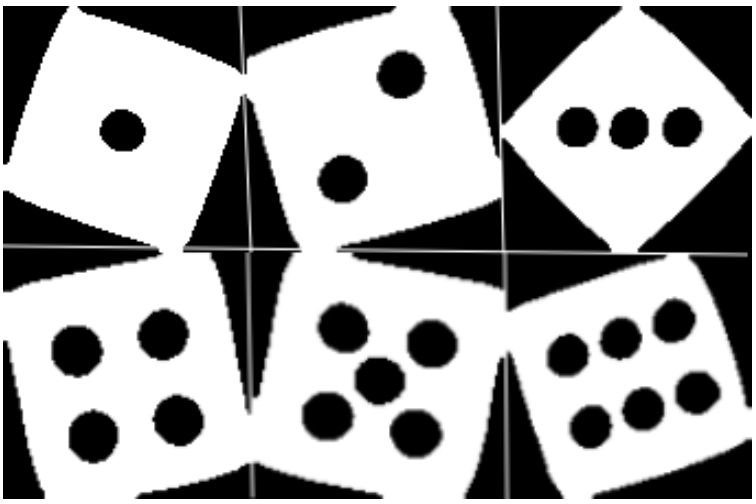
Sada kada je sa slike uklonjen šum, može se binarizirati. Korištenjem funkcije *adaptiveThreshold* dobiti će se binarna slika na kojoj će se jasno vidjeti obrubi kockica te kružići koji označavaju koliko je bačeno. Kada je dobivena binarna slika, provjerava se ako je crna boja dominanta, tj. ako su obrubi kockice i kružići bijele boje. U slučaju da nije, bijeli pikseli se mijenjaju crnim pikselima i obrnuto. Ovaj korak je vrlo bitan kako bi se kasnije mogli detektirati rubovi, odnosno konture svake kockice koje bi trebale biti bijele boje. Binarna slika gdje je bijela dominantna boja (lijevo) i binarna slika gdje je crna dominantna boja (desno), može se vidjeti na slici 3.10.



Slika 3.10. *Slika nakon primjene adaptivnog Thresholda*

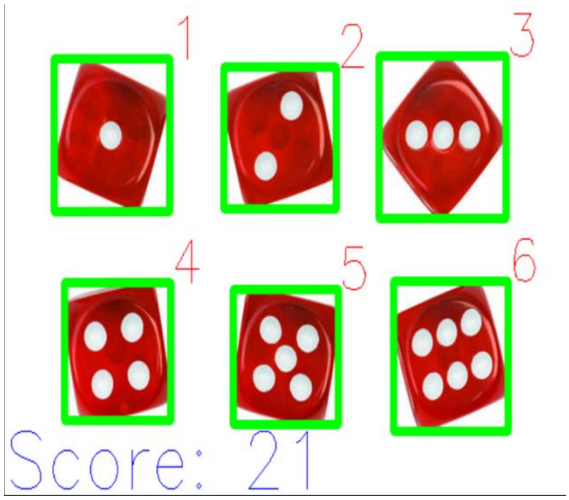
Binarna slika prosljeđuje se funkciji *getDiceContours.* Uz binarnu sliku, još se prosljeđuje izvorna sliku, kopiju izvorne slike na kojoj će se iscrtavati rezultati te instanca *simpleBlobDetector-a. getDiceContours* pronalazi kockice na slici. Provjerava se površina potencijalne kockice kako ne bi došlo do lažno pozitivnog rezultata gdje je zapravo pronađen šum. Kada je kockica pronađena,

na slici crta pravokutnik oko kockice te se taj dio slike unutar pravokutnika prosljeđuje funkciji *getPips* na daljnju obradu. Funkcija *getPips* zapravo dobiva dio izvorne slike na kojem se nalazi kockica. Taj dio slike ponovno se pretvara u sive tonove te se uklanja šum, ali različito od prvog puta, sada se slika binarizira Otsuovom metodom. Kada se dobije binarna slika, provjerava se ako je crna dominantna boja. Pretpostavka je da će kockica na slici biti bijele boje te sukladno tome, dominantna boja bi trebala biti bijela. U slučaju da to nije slučaj, crni pikseli se mijenjaju bijelima i obrnuto. Ovaj korak je bitan kako bi *simpleBlobDetector* mogao naći kružiće na slici koji bi trebali biti crne boje. Na izvornoj slici može se vidjeti šest kockica, prema tome, funkcija *getPips* trebala bi dobiti i obraditi šest različitih dijelova izvorne slike. Slike obrađene *getPips* funkcijom mogu se vidjeti na slici 3.11.



Slika 3.11. *Slike obrađene getPips funkcijom*

Kada je dobivena binarna slika, mogu se pronaći i prebrojati kružići pomoću *simpleBlobDetector-a.* Broj pronađenih kružića zapisuje se pored pravokutnika za svaku kockicu. Na kraju, na sliku s rezultati zapisuje se ukupni zbroj bačenih kockica. Slika s rezultatima bacanja može se vidjeti na slici 3.12.



Slika 3.12. *Slika s rezultatima bacanja*

Na slikama 3.13. i 3.14. može se vidjeti programski kod.

```python
import cv2
import numpy as np

path = "images/img9.jpg"

def showImg(img):
    img = cv2.resize(img, (800, 700))
    cv2.imshow("dice", img)
    cv2.waitKey(0)

def getDiceContours(thresh_img, original_img, result_image, pipDetector):
    pipCount = 0
    contours, hierarchy = cv2.findContours(thresh_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    for contour in contours:
        par = cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, 0.02*par, True)
        x, y, w, h = cv2.boundingRect(approx)

        if((w)*(h) > thresh_img.shape[0]*thresh_img.shape[1]*0.01):
            cv2.rectangle(result_image, (x, y), (x + w, y + h), (0, 255, 0), 5)
            pipCount += getPips(original_img[y:y+h, x:x+w], y, x+w, pipDetector, result_image)

    return pipCount



def getPips(dice_img, y, x, pipDetector, result_image):
    dice_img_gray = cv2.cvtColor(dice_img, cv2.COLOR_BGR2GRAY)
    dice_img_blur = cv2.GaussianBlur(dice_img_gray, (7,7), 5)
    _, dice_img_thresh = cv2.threshold(dice_img_blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    #Blob Detector detects only black pips/blobs
    if(isBlackDominantColorInTheBinaryImage(dice_img_thresh)==True):
            dice_img_thresh = 255 - dice_img_thresh
    cv2.imshow("Dice img", dice_img_thresh)
    cv2.waitKey(0)
    keypoints = pipDetector.detect(dice_img_thresh)
    numberOfKeypoitns = len(keypoints)

    #For cases when there is a lot of background on picture parts with dice
    if(numberOfKeypoitns==0):
        dice_img_thresh = 255 - dice_img_thresh
        keypoints = pipDetector.detect(dice_img_thresh)
        numberOfKeypoitns = len(keypoints)

    cv2.putText(result_image, str(numberOfKeypoitns), (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 0, 255))
    return numberOfKeypoitns
```

Slika 3.13. *Programski kod [1/2]*

```
48
49  def getPipDetector():
50      params = cv2.SimpleBlobDetector_Params()
51      params.minThreshold = 150
52      params.maxThreshold = 200
53      params.filterByCircularity = True
54      params.minCircularity = 0.1
55      params.filterByInertia = True
56      params.minInertiaRatio = 0.5
57      params.filterByConvexity = True
58      params.minConvexity = 0
59      return cv2.SimpleBlobDetector_create(params)
60
61  def isBlackDominantColorInTheBinaryImage(img):
62      white_pixels = np.count_nonzero(img > 0)
63      black_pixels = np.count_nonzero(img == 0)
64      if(white_pixels > black_pixels):
65          return False
66      return True
67
68  def isWhiteDiceOnWhiteBackground(img):
69      white_pixels = np.count_nonzero(img>230)
70      if(white_pixels > img.shape[0]*img.shape[1]*0.8):
71          return True
72      return False
73
74
75  original_img = cv2.imread(path)
76  showImg(original_img)
77  result_image = original_img.copy()
78  img_gray = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
79
80
81  #Check if the pictures contains white dice on white background
82  if(isWhiteDiceOnWhiteBackground(img_gray)):
83      img_blur = cv2.GaussianBlur(img_gray, (5,5), 2)
84      showImg(img_blur)
85      thresholded_img = cv2.adaptiveThreshold(img_blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 1)
86      showImg(thresholded_img)
87
88  else:
89      img_blur = cv2.GaussianBlur(img_gray, (7,7), 5)
90      showImg(img_blur)
91      thresholded_img = cv2.adaptiveThreshold(img_blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
92      showImg(thresholded_img)
93
94
95  if(isBlackDominantColorInTheBinaryImage(thresholded_img)==False):
96      thresholded_img = 255 - thresholded_img
97  showImg(thresholded_img)
98  pipDetector = getPipDetector()
99
100 pips = getDiceContours(thresholded_img, original_img, result_image, pipDetector)
101
102 cv2.putText(result_image, "Score: "+str(pips), (0,result_image.shape[0]-5), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0))
103
104 showImg(result_image)
105 print("Pips detected: ",pips)
106
```

Slika 3.14. *Programski kod [2/2]*

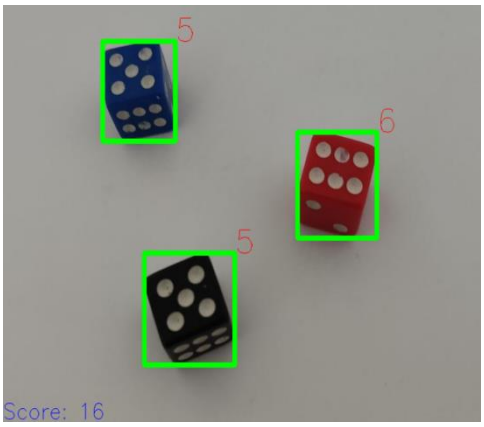
3.8. Testiranje programskog rješenja

Uz prikazani primjer u proplom poglavlju, programsko rješenje bit će dodatno testirano na 5 različitih ispitnih slučajeva. Svaki od testnih slučajeva predstavlja posebne slučajeve koji se mogu pojaviti.

3.8.1. Ispitni slučaj 1

U ovom ispitnom slučaju na slici se nalaze tri kockice različitih boja na svijetloj pozadini. Izvorna slika za ovaj ispitni slučaj može se vidjeti na slici 3.15., a rezultati za ovu sliku mogu se vidjeti na slici 3.16.

Slika 3.15. *Ispitni slučaj 1 - izvorna slika*

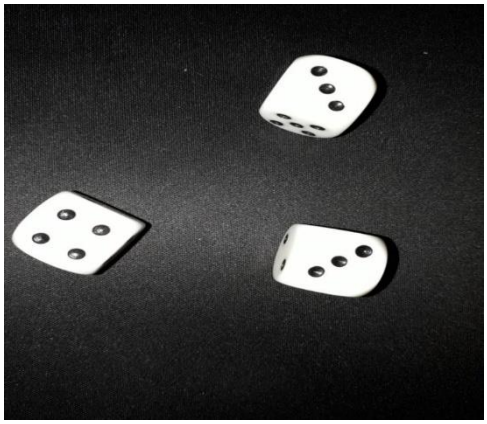


Slika 3.16. *Ispitni slučaj 1 - rezultat*

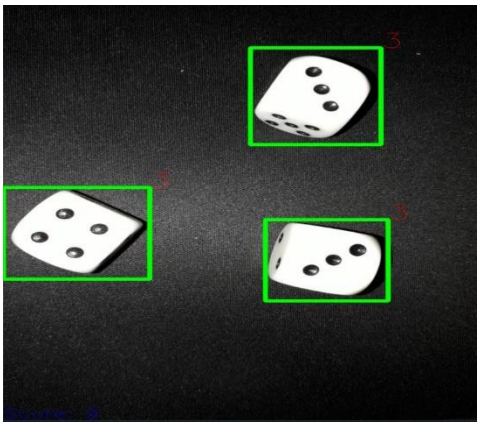
Promatranjem rezultata ovog ispitnog slučaja može se zaključiti kako algoritam nema problema s prepoznavanjem kockica različitih boja, tj. kockice će biti prepoznate bez obzira na boju.

3.8.2. Ispitni slučaj 2

U ovom ispitnom slučaju na slici se nalaze bijele kockice na tamnoj pozadini, ali se na slici jasno vidi odsjaj. Izvorna slika za ovaj ispitni slučaj može se vidjeti na slici 3.17., dok se slika s rezultatima može vidjeti na slici 3.18.



Slika 3.17. *Ispitni slučaj 2 - izvorna slika*



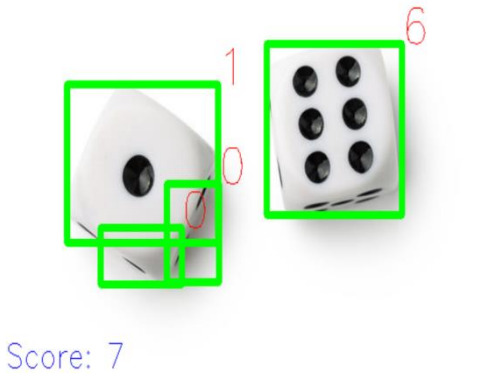
Slika 3.18. *Ispitni slučaj 2 – rezultat*

Promatranjem rezultata drugog ispitnog slučaja može se zaključiti kako manje količine odsjaja na slici ne stvaraju problem algoritmu. Ovo je također testirano i na slikama s bijelom pozadinom što algoritmu također ne stvara problem.

3.8.3. Ispitni slučaj 3

U ovom ispitnom slučaju na slici se nalaze bijele kockice na bijeloj pozadini. Kada se detektira ovakva slika koriste se drugačiji parametri pri korištenju funkcija *GaussianBlur* i *adaptiveThreshold* kako bi se dobili željeni rezultati. Izvorna slika za ovaj ispitni slučaj može se vidjeti na slici 3.19., dok se slika s rezultatima može vidjeti na slici 3.20.

Slika 3.19. *Ispitni slučaj 3 - izvorna slika*

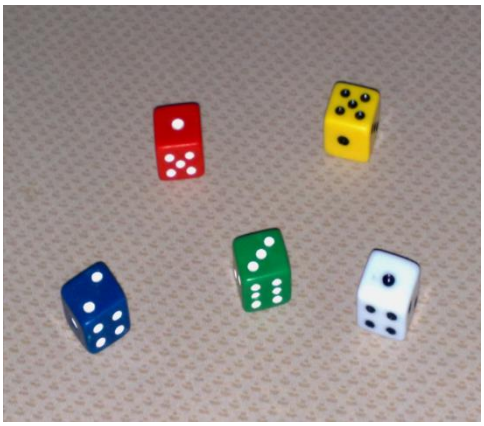


Slika 3.20. *Ispitni slučaj 3 – rezultat*

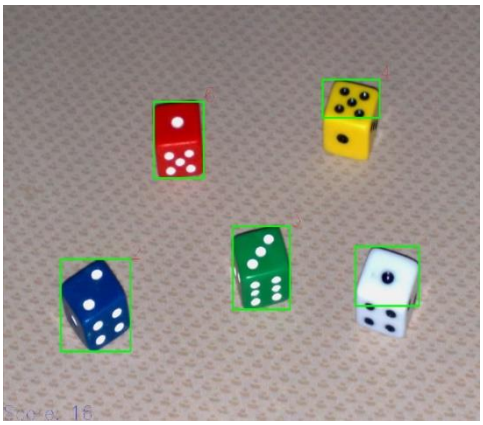
Slike sa bijelim kockicama na bijeloj pozadini predstavljaju poseban izazov. Promatranjem slike 3.20. može se vidjeti kako je algoritam dobro prebrojao kružiće na slici. S druge strane, može se vidjeti kako su neke sjene označene kao kockice što je nedostatak ovog algoritma i može prouzročiti ozbiljnije probleme na slikama ovog tipa.

3.8.4. Ispitni slučaj 4

U ovom ispitnom slučaju kockice će biti različitih boja i slikane pod većim kutom. Dodatno, pozadina neće jednobojna. Izvorna slika za ovaj ispitni slučaj može se vidjeti na slici 3.21., dok se slika s rezultatima može vidjeti na slici 3.22.

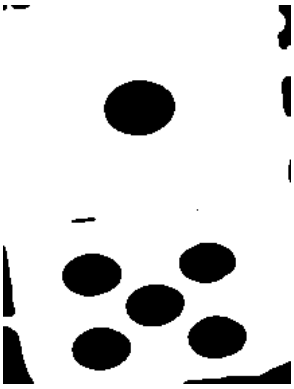


Slika 3.21. *Ispitni slučaj 4 - izvorna slika*



Slika 3.22. *Ispitni slučaj 4 – rezultat*

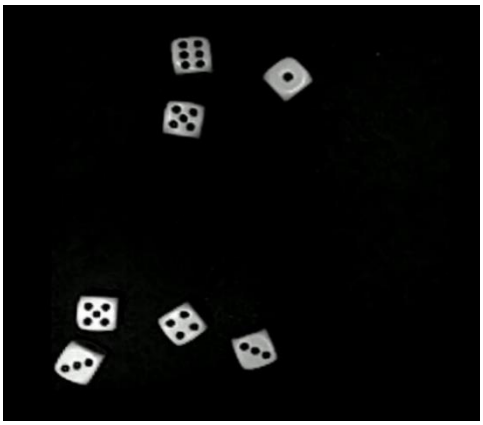
Promatranjem rezultata četvrtog ispitnog slučaja može se zaključiti kako blagi uzorak u pozadini ne predstavlja problem, ali kockice slikane pod velikim kutom mogu algoritmu stvoriti problem. Na slici se može vidjeti kako je crvenom kockicom bačeno 1, ali je na slici označeno da je bačeno 6. Detaljnijom analizom ovog slučaja i provjerom dijela slike s crvenom kockicom koji je predan funkciji *getPips,* može se vidjeti zašto je rezultat za crvenu kockicu šest. Na slici 3.23. može se vidjeti dio slike koji je poslan funkciji *getPips*.



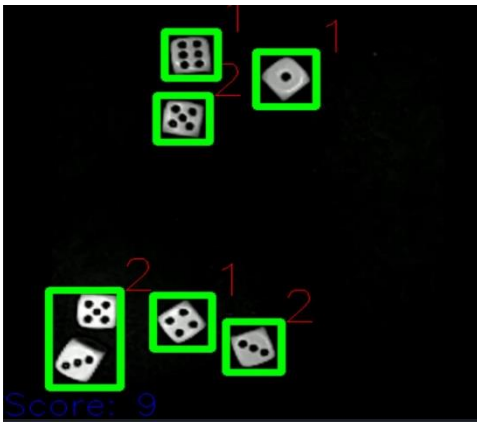
Slika 3.23. *Slika crvene kockice obrađene funkcijom getPips*

3.8.5. Ispitni slučaj 5

U zadnjem ispitnom slučaju koristiti će se slika na kojoj su bijele kockice na tamnoj pozadini, ali slika je male rezolucije i loše kvalitete. Izvorna slika za ovaj ispitni slučaj može se vidjeti na slici 3.24., dok se slika s rezultatima može vidjeti na slici 3.25.



Slika 3.24. *Ispitni slučaj 5 - izvorna slika*



Slika 3.25. *Ispitni slučaj 5 – rezultat*

Promatranjem petog ispitnog slučaja možemo zaključiti kako algoritam ne radi dobro kada je izvorna slika vrlo niske rezolucije. Na slici se može vidjeti kako su konture kockica prepoznate, ali korištenjem *simpleBlobDetector-a* prepoznat je samo dio kružića.

4. ZAKLJUČAK

Algoritam je dosta uspješan u pronalaženju zbroja bačenih kockica. Prednost algoritma je što brzo pronalazi sve kockice na slici te ih pronalazi bez obzira na boju i uzorak pozadine i bez obzira na boju samih kockica. Do problema može doći u iznimnim slučajevima kada su na slici bijele kockice na bijeloj pozadini. Dodatno, algoritam je dosta otporan na manje količine odsjaja koji se može pojaviti na slikama. S druge strane, algoritmu stvaraju probleme slike koje su niske rezolucije i slike na kojima su kockice slikane pod jako velikim kutom. Najbolji rezultati se dobivaju na slikama gdje su bijele kockice slikane s malim kutom na tamnoj pozadini.

Naravno, algoritam ima mnogo prostora za poboljšanja, kao što je korištenje drugog načina za pronalaženje kružića kako bi algoritam radio i na slikama niže rezolucije te bolje prepoznavanje gornje strane kockice kako se ne bi zbrajali i kružići koje se nalaze na bočnim stranama kockice.

5. LITERATURA

[1] E. Omeragic and E. Sokic, "Counting rectangular objects on conveyors using machine vision," *2020 28th Telecommunications Forum (TELFOR)*, 2020, pp. 1-4, doi: 10.1109/TELFOR51502.2020.9306530.

[2] N. A. OTHMAN, M. U. SALUR, M. KARAKOSE and I. AYDIN, "An Embedded Real-Time Object Detection and Measurement of its Size," *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, 2018, pp. 1-4, doi: 10.1109/IDAP.2018.8620812.

[3] Z. Shuaishuai and P. Chen, "Research on License Plate Recognition Algorithm Based on OpenCV," *2019 Chinese Automation Congress (CAC)*, 2019, pp. 68-72, doi: 10.1109/CAC48633.2019.8996599.

[4] About OpenCV, https://opencv.org/about.html

[5] OpenCV cvtColor, https://www.educba.com/opencv-cvtcolor/

[6] OpenCV- Color Space Conversion, https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html

[7] Smoothing Images, https://docs.opencv.org/3.4/d4/d13/tutorial_py_filtering.html

[8] OpenCV – Image fitering, https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#gaussianblur

[9] Image Thresholding, https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[10] Miscellaneous Image Transformation, https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?#cv2.threshold

[11] Blob detection using OpenCV, https://learnopencv.com/blob-detection-using-opencv-python-c/

6. POPIS SLIKA