Stohastička optimizacija u obradi slike

Benković, Marko Domagoj

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Marko Domagoj Benković

STOHASTIČKA OPTIMIZACIJA U OBRADI SLIKE

Diplomski rad

Voditelj rada:
prof. dr. sc. Luka Grubišić

Zagreb, studeni, 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Svojoj obitelji, djevojci, prijateljima, mentoru i svima drugima koji su na bilo koji način pomogli u ispunjenju ovog cilja.

Sadržaj

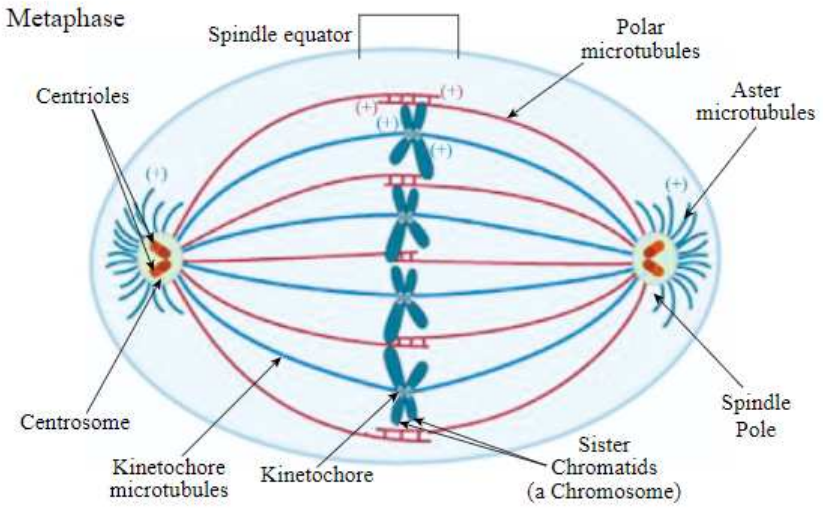
Uvod

Tema ovog diplomskog rada je algoritamska obrada slike dobivene korištenjem elektronskog mikroskopa. U ovom kontekstu, sliku poistovjećujemo s funkcijom $I : \{0, ..., n_x\} \times \{0, ..., n_y\} \rightarrow \{0, ..., 255\}$ koja točki (i, j) pridružuje intenzitet signala $I(i, j) = I_{ij}$, a više o digitalnim slikama i njihovoj obradi slijedi u idućem poglavlju. U daljnjoj obradi slike potrebno je identificirati povezana područja sličnog intenziteta i odrediti njihov parametarski zapis koji će biti pogodan za daljnju obradu. Automatska obrada slike nužna je zbog sljedeća dva razloga - objektivizacija rezultata (neovisnih o analitičaru) te mogućnost obrade familije slika (npr. film ili CT snimke). Kao prototipni problem za ovu studiju promatrat ćemo slike stanične diobe s ciljem objektivnog praćenja dinamike procesa.

U staničnoj biologiji, diobeno vreteno predstavlja citoskeletnu strukturu eukariotskih stanica koja se formira tijekom stanične diobe, s ciljem razdvajanja sestrinskih kromatida prema suprotnim polovima stanice. Jedan od izazova računalnog vida jest automatska detekcija centrosoma, koji su zaslužni za određivanje položaja mnogih organela unutar stanice. U prvom poglavlju ovog rada se predlaže i obrađuje jedna od metoda lociranja centrosoma, a u drugom poglavlju računamo kut otvora diobenog vretena na polovima.

Razlikujemo mitotičko vreteno tijekom mitoze, procesa koji proizvodi genetski identične stanice, i mejotičko vreteno tijekom mejoze, procesa koji proizvodi gamete s dvostruko manjim brojem kromosoma od matične stanice. Dakle, uloga diobenog vretena je u mitozi i mejozi, ali ovdje je fokus primarno na mitotičkom vretenu. Na slici 0.1 nalazi se ilustracija strukture mitotičkog vretena za vrijeme treće faze mitoze - metafaze.

Nastavljamo s kratkim pregledom i uvodnim opisom svih pet faza mitoze. Kod životinjskih organizama, mitotičko vreteno nastaje u profazi, prvoj fazi mitoze. Preciznije, za vrijeme profaze, mitotičko diobeno vreteno stvara se iz centrosoma, a tvore ga mikrotubule koje, osim što oblikuju vreteno, također služe za prijenos raznih proteina te sudjeluju u staničnom i unutarstaničnom gibanju. Vreteno se formira između dva centrosoma koji se nalaze na staničnim polovima te se sastoji od dva poluvretena. Oba poluvretena se pružaju od svog centrosoma prema sredini stanice. To nas, između ostalog, motivira na promatranje kuta otvora vretena na polovima. U drugoj fazi mitoze, prometafazi, za niti mitotičkog vretena na mjestu centromera vežu se raspršeni kromosomi, a kad se proces vezivanja



Slika 0.1: Struktura mitotičkog vretena za vrijeme metafaze

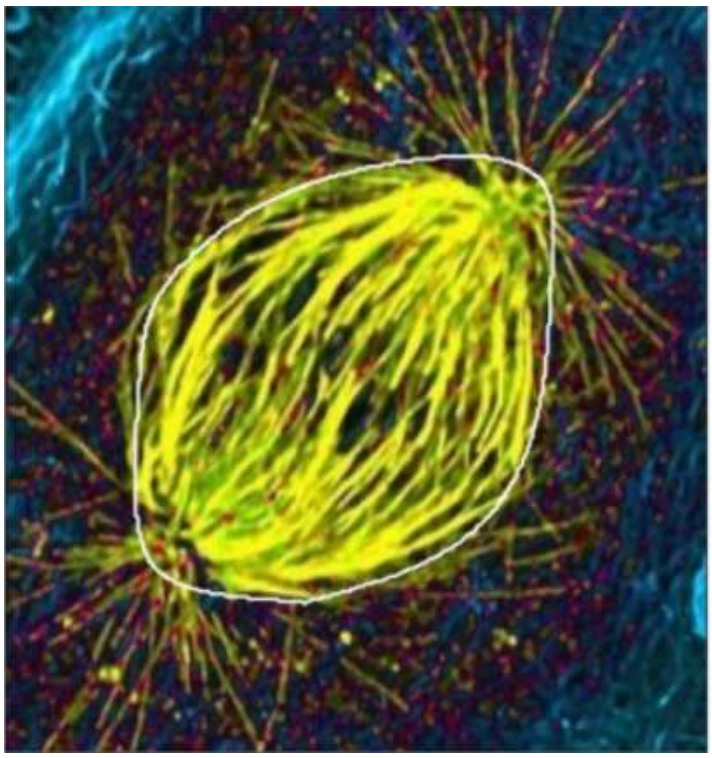
završi, počinje treća faza - metafaza. Za vrijeme metafaze se raspršeni kromosomi poredaju u središnjoj ravnini stanice u smjeru okomitom na položaj mikrotubula. U idućoj fazi, anafazi, samostalni kromosomi putuju prema polovima vretena, na način da se skraćuju mikrotubule mitotičkog vretena. Konačno, vreteno se raspada u završnoj fazi mitoze - telofazi. Više o diobenim vretenima može se pronaći u [13] i [11].

Nadalje, na zadanoj slici vretena želimo odrediti i formulom reprezentirati krivulju koja aproksimira konturu vretena. Također, provodimo i dva optimizacijska postupka - kvadratna optimizacija s ciljem pronalaska elipse koja najbolje opisuje aproksimiranu konturu te pronalazak optimalne konture u ovisnosti o pragu kojim segmentiramo sliku. Preciznije, želimo pronaći optimalni $\mathbf{a}^*$ za koji funkcija greške

$$\epsilon^2(\mathbf{a}) = \sum_{i=1}^{n} \delta(C(\mathbf{a}), \mathbf{x}_i)$$

postiže globalni minimum, pri čemu je $C(\mathbf{a})$ familija krivulja parametriziranih vektorom $\mathbf{a}$ te δ metrika koja mjeri udaljenost točke $\mathbf{x}$ od krivulje $C(\mathbf{a})$.
S druge strane, za $i \in \{p-t_1, ..., p, ..., p+t_2\}$, pri čemu je p Otsuov prag te t_1 i t_2 proizvoljni, dobivamo niz elipsi E_i koje najbolje opisuju konture ovisne o pragu i. Zatim, formiramo niz duljina velikih osi v_i te računamo $\overline{v}$. Konačno, tražimo $\arg\min_i |v_i - \overline{v}|$ koji određuje optimalnu konturu.

Slijedi primjer 0.2 koji prikazuje parametriziranu krivulju na konkretnoj slici mitotičkog vretena.

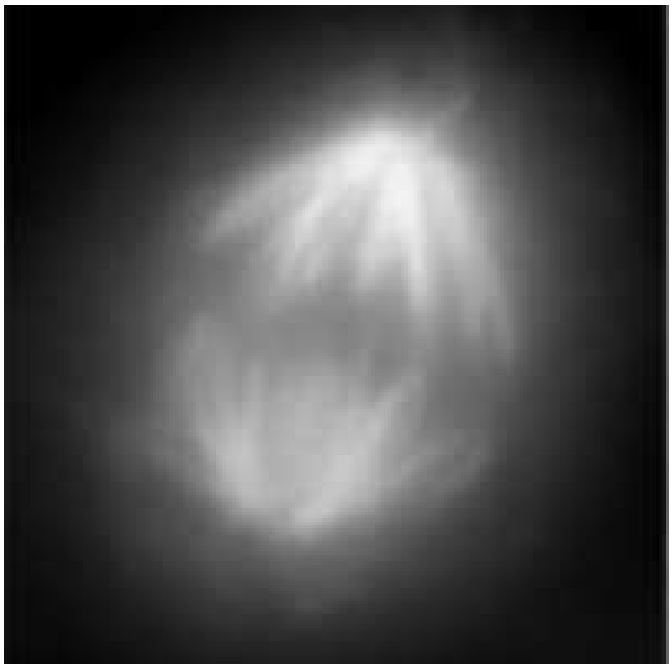


Slika 0.2: Parametrizirana krivulja koja aproksimira konturu mitotičkog vretena

Poglavlje 1

Određivanje položaja centrosoma

Kao što je već spomenuto u uvodnom dijelu, važan dio stanične strukture su centrosomi, a samim tim i određivanje njihove lokacije. U ovom poglavlju predlažemo metodu računalnog vida koja određuje položaj centrosoma na digitalnim slikama mitotičkog vretena. Na sljedećoj slici nalazi se jedan takav primjerak dobiven elektronskim mikroskopom, kojeg koristimo za daljnji razvoj metode.



Slika 1.1: Mitotičko vreteno pod mikroskopom[1]

1.1 Digitalna slika

Najprije definiramo pojam digitalne slike - osnovni pojam koji predstavlja jedan element iz skupa podataka. Digitalna slika sastoji se od konačno mnogo elemenata koje nazivamo pikselima. Pri tome, svaki piksel ima određenu vrijednost na određenoj lokaciji. Preciznije, digitalna slika je numerički zapis slike organiziran u višedimenzionalni niz. Vrijednosti tog niza definirane su funkcijom dvije varijable x i y, koje zovemo prostornim koordinatama, a vrijednost funkcije u bilo kojem paru koordinata (x, y) naziva se intenzitetom slike u toj točki.

Također, u ovisnosti o vrijednostima piksela razlikujemo tri tipa digitalnih slika. To su crno-bijele slike, slike sivih tonova te slike u boji. Ako navedena funkcija poprima vrijednosti iz skupa $\{0, 1\}$, onda govorimo o crno-bijeloj slici. Ako je riječ o vrijednostima iz skupa $\{0, 1, ..., 255\} =: S$, radi se o slici sivih tonova, pri čemu 0 označava crnu, a 255 bijelu boju. Konačno, ako su vrijednosti iz skupa $\{(r, g, b) \mid r, g, b \in S\}$, onda govorimo o slici u boji. U tom slučaju, sliku u boji još zovemo i RGB slikom, iz razloga što r predstavlja intenzitet crvene, g zelene i b plave boje. U ovom radu koristimo prvenstveno slike sivih tonova.

Uz digitalne slike je usko vezan i pojam njihove obrade. Naime, da bismo računalom obradili digitalnu sliku te tako iz nje izvukli korisne informacije ili joj poboljšali kvalitetu, koristimo određene algoritme za obradu slike. Sav programski kod napisan je u Python-u, iz razloga što se radi o programskom jeziku koji ima dobru podršku za algoritme u području računalnog vida.
Prvi korak je učitavanje slike pomoću nekog programskog alata. U tu svrhu koristimo *OpenCV* biblioteku ([7]) i *cv2.imread* metodu. Polazni skup podataka na kojem testiramo razvijene metode sastoji se od nekoliko digitalnih slika sivih tonova, raznih dimenzija. Jedan primjerak odgovara prethodnoj slici 1.1.

1.2 Tehnike obrade slike

Podsjetimo se, zadatak nam je locirati centrosome na zadanim slikama. Da bismo to postigli, najprije na slici pronalazimo rub pripadnog mitotičkog vretena. Jednom kad odredimo rub, možemo se fokusirati na manji dio slike i tamo tražiti centrosome, umjesto da to činimo na kompletnoj slici.

[1] Slika je dobivena s poveznice `https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/mitotic-spindle-apparatus`

Normalno zamućenje (Gaussian blur)

Nakon što smo učitali sliku, najprije koristimo tzv. tehniku zamućenja (en. *blurring*). Ukratko, zamućivanje čini sliku manje "oštrom" i na taj način može, između ostalog, ukloniti šum sa slike. Budući da je, u našem slučaju, digitalna slika dvodimenzionalni niz (matrica) vrijednosti iz skupa $\{0, 1, ..., 255\}$, to se postiže zaglađivanjem naglih promjena vrijednosti intenziteta piksela. Upravo je tehnika zamućivanja slike gotovo uvijek prvi korak pri detekciji rubova na slici.

Postoje dva tipa metode zamućivanja slike, ovisno o tipu filtera - linearni (npr. *mean* i *Gaussian*) i nelinearni (npr. *median*, *max* i *min*). Ovdje opisujemo korištenu metodu Gaussovog zamućenja. Međutim, prije toga trebamo definirati pojam diskretne konvolucije ([5], [3]).

Definicija 1.2.1. *Neka su $f, g : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$. Diskretna konvolucija funkcija f i g, u oznaci $f * g$, je operacija definirana sa:*

$$(f * g)(x, y) := \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j)g(x - i, y - j).$$

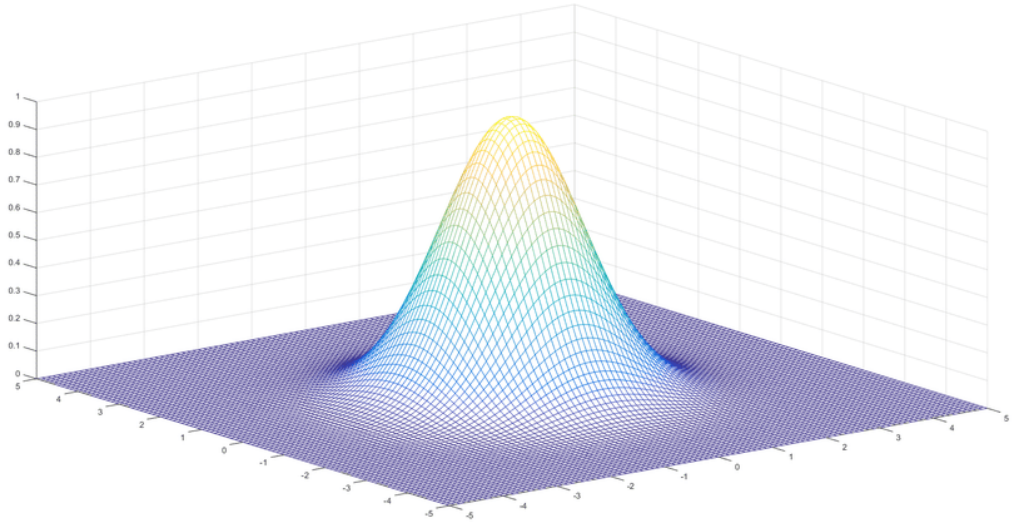
Također, potrebna nam je i Gaussova funkcija dvije varijable:

$$G(x, y) := \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

pri čemu σ predstavlja standardnu devijaciju dvodimenzionalne normalne distribucije.

Dakle, Gaussovo izglađivanje je metoda koja koristi Gaussovu funkciju za transformaciju pojedinog piksela na slici. U tu svrhu se primijenjuje operacija diskretne konvolucije specijalizirane matrice (filter ili en. *kernel*) na matricu digitalne slike. Ideja je koristiti vrijednosti iz dvodimenzionalne normalne distribucije za konstrukciju filtera neparne dimenzije (najčešće su to 3×3, 5×5 ili 7×7). Konvolucija djeluje tako da se nova vrijednost pojedinog piksela na digitalnoj slici postavlja na težinski prosjek susjedstva tog piksela, pri čemu se piksel koji se u tom trenutku obrađuje preklapa sa središnjim pikselom filtera te tako matrica dobivena konvolucijom ima jednake dimenzije kao i matrica digitalne slike (ilustracija se može vidjeti u primjeru 1.2.2). Kod konvolucije Gaussovim filterom, vrijednost izvornog piksela dobiva najveću težinu (ima najveću Gaussovu vrijednost), a susjedni pikseli dobivaju manje težine s povećanjem udaljenosti od izvornog piksela.

Slijedi pojednostavljeni primjer koji prikazuje rezultat konvolucije dvije matrice. Matrica I predstavlja digitalnu sliku, dok matrica K predstavlja filter, odnosno *kernel*.



Slika 1.2: Dvodimenzionalna normalna distribucija

Primjer 1.2.2. *Neka su*

$$I = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 2 & 3 & 0 & 2 \\ 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}, \quad K = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

dvije matrice dimenzija 4×4 *i* 3×3*. Tada je matrica dobivena konvolucijom matrica* I *i* K *jednaka:*

$$I * K = \begin{bmatrix} \frac{11}{16} & \frac{7}{8} & \frac{15}{16} & \frac{3}{4} \\ \frac{5}{4} & \frac{25}{16} & \frac{19}{16} & \frac{13}{16} \\ \frac{5}{4} & \frac{3}{2} & \frac{15}{16} & \frac{1}{2} \\ \frac{7}{8} & \frac{7}{8} & \frac{1}{2} & \frac{5}{16} \end{bmatrix}$$

U navedenom primjeru nailazimo na tipičan problem konvolucije dvije matrice: kad navedeni postupak provodimo na, primjerice, elementu matrice I na poziciji $(0, 0)$, taj element se poklapa sa središnjim elementom matrice K te tako filter "ispada" preko ruba matrice I. Odnosno, prvi (nulti) redak matrice K bismo točkovno množili sa "-1." retkom matrice I koji, jasno, ne postoji. Taj problem se može riješiti na nekoliko načina, a jedan od njih je

korišten u danom primjeru - proširenje matrice I konstantom vrijednošću izvan definiranog područja, npr. proširenje nulom. Tako dobivamo novu 6×6 matricu:

$$\tilde{I} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 1 & 0 \\ 0 & 2 & 3 & 0 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

na koju se primijenjuje konvolucija filterom K te dobivamo novu matricu $I * K$ dimenzija 4×4. Dakle, proširenje matrice I nulom omogućuje da se središnji element filtera K efektivno preklopi sa svakim elementom matrice I prilikom procesa konvolucije $\tilde{I}$ i K te se tako sačuva dimenzija. Kad to ne bismo radili, nego jednom provukli filter kroz matricu I (središnji element filtera se u tom slučaju ne preklopi sa svakim elementom od I), onda bismo konvolucijom dobili matricu manje dimenzije te tako izgubili dio informacija sa rubova polazne slike. Međutim, to je česta praksa kod konvolucijskih neuronskih mreža, gdje se operacijom konvolucije dvije matrice (tenzora) u konvolucijskom sloju dobiva nova matrica (tenzor) manjih dimenzija koja čuva informacije o pojedinim dijelovima slike (tenzora).

Obzirom da je slika, tj. matrica piksela, sastavljena od diskretnog skupa podataka, potrebna nam je i diskretna aproksimacija Gaussove funkcije za konstrukciju filtera. *OpenCV* biblioteka sadrži razne funkcije pogodne za obradu digitalnih slika. U ovom slučaju, funkcija *cv2.getGaussianKernel* određuje koeficijente filtera. Funkcija prima tri parametra:

- *ksize* - veličina filtera, neparan prirodan broj
- *sigma* - standardna devijacija, ukoliko se specificira nepozitivna vrijednost, računa se po formuli: $sigma = 0.3 \cdot ((ksize - 1) \cdot 0.5 - 1) + 0.8$
- *ktype* - tip vrijednosti koeficijenata, CV_32F ili CV_64F

te računa $ksize \times 1$ vektor koeficijenata G po formuli:

$$G_i = \alpha \cdot e^{-\frac{(i-(ksize-1)/2)^2}{2 \cdot sigma^2}}, \quad \forall i \in \{0, 1, ..., ksize - 1\},$$

pri čemu je α normalizacijski skalar takav da je $\sum_i G_i = 1$.

Gaussov filter je linearno separabilan pa je, primjerice:

$$K = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

To svojstvo omogućava da se, nakon što se odredi $ksize \times 1$ vektor G, iskoristi funkcija *cv2.sepFilter2D* koja primujenjuje separabilan filter na sliku tako što prvo primijeni G na svaki redak dane slike, a zatim primijeni isti filter G na svaki stupac dobivenog rezultata.

Međutim, cijeli postupak je implementiran u jednoj metodi *cv2.GaussianBlur*, kojom dobivamo zamućenje slike. Parametri funkcije su sljedeći:

- *src* - ulazna digitalna slika
- *ksize* - veličina Gaussovog filtera; uređen par neparnih prirodnih brojeva (mogu biti različiti) ili $(0, 0)$ (u tom slučaju se veličina računa na temelju st. devijacije)
- *sigmaX* - st. devijacija u x smjeru
- *sigmaY* - st. devijacija u y smjeru; ako iznosi 0 ili se ne postavi vrijednost, onda je jednaka *sigmaX*. Ako su i *sigmaX* i *sigmaY* jednaki 0, onda se računaju iz *ksize* koordinata
- *borderType* - metoda ekstrapolacije piksela

U daljnjem korištenju metode *GaussianBlur* veličinu filtera *ksize* mijenjamo ovisno o dimenzijama slike. Odnosno, uzimamo 10% srednje vrijednosti dimenzija slike.

Više detalja o navedenim funkcijama dostupno je u [6].

Metoda odsjecanja (thresholding)

Nakon zamućivanja slike Gaussovim filterom, idući korak je korištenje metode odsjecanja (en. *thresholding*). Radi se o vrsti segmentacije slike, pri čemu se mijenjaju intenziteti piksela s ciljem lakšeg analiziranja slike. Time se slika pretvara iz slike u boji ili slike sivih tonova u binarnu sliku. Najčešće se pragovi (en. *threshold*) koriste za odabir interesnih područja na slici, dok se dijelovi koji nas ne zanimaju zanemaruju.

Razlikujemo nekoliko vrsta tehnika odsjecanja, a u ovom odjeljku opisujemo jednostavno, prilagodljivo i Otsuovo određivanje praga kao tri osnovne metode.

Jednostavno odsjecanje

Radi se o osnovnoj metodi odsjecanja - jednostavno odsjecanje. Metoda je jednostavna iz razloga što se za svaki piksel primjenjuje ista vrijednost zadanog praga. Ako je vrijednost piksela manja od praga, postavlja se na 0, inače se postavlja na najveću vrijednost.

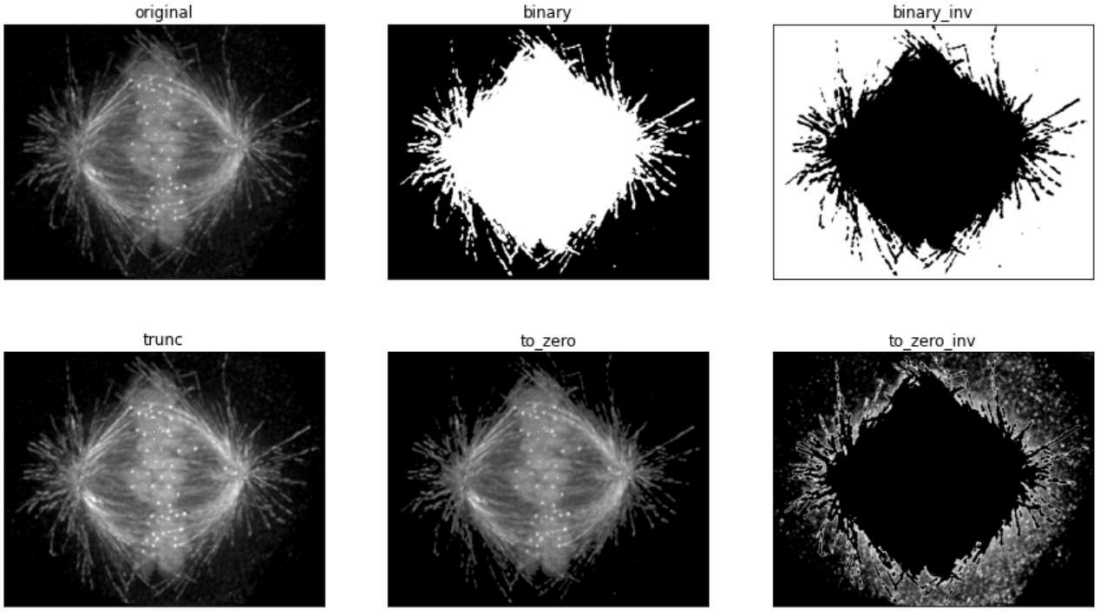
Metoda se programski realizira korištenjem funkcije *cv2.threshold*. Prvi argument funkcije je izvorna slika u sivim tonovima. Drugi argument je vrijednost praga. Treći argument je

najveća vrijednost koja se dodjeljuje vrijednostima piksela iznad praga. *OpenCV* pruža pet različitih tipova jednostavnog odsjecanja, a jednog od njih specificiramo kao četvrti parametar funkcije:

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TO_ZERO
- cv.THRESH_TO_ZERO_INV

Metoda vraća dvije vrijednosti - vrijednost praga i promijenjenu sliku.

Na sljedećoj slici 1.3 mogu se vidjeti rezultati metode u ovisnosti o odabiru vrijednosti zadnjeg parametra. Za sve odabrane vrijednosti korišten je prag s vrijednošću 30, osim kod *cv.THRESH_TRUNC* opcije, gdje je prag iznosio 150.



Slika 1.3: Metoda jednostavnog odsjecanja

Prilagodljivo odsjecanje

Kod jednostavnog odsjecanja koristi se jedna fiksna globalna vrijednost za prag, što u nekim slučajevima nije ono što želimo (npr. ako slika ima različite uvjete osvjetljenja na različitim područjima). U tom slučaju može biti korisno tzv. prilagodljivo određivanje praga (en. *adaptive thresholding*).

Algoritam određuje prag za pojedini piksel na temelju njegove okoline. Dakle, dobivamo različite pragove za različita područja iste slike, što daje bolje rezultate za slike s različitim osvjetljenjem. Ova metoda se programski realizira korištenjem funkcije *cv2.adaptiveThreshold*, koja ima iste parametre kao i *cv2.threshold* funkcija, samo što ovdje nemamo parametar koji predstavlja prag (određuje se algoritmom) te uz to imamo još tri dodatna parametra:

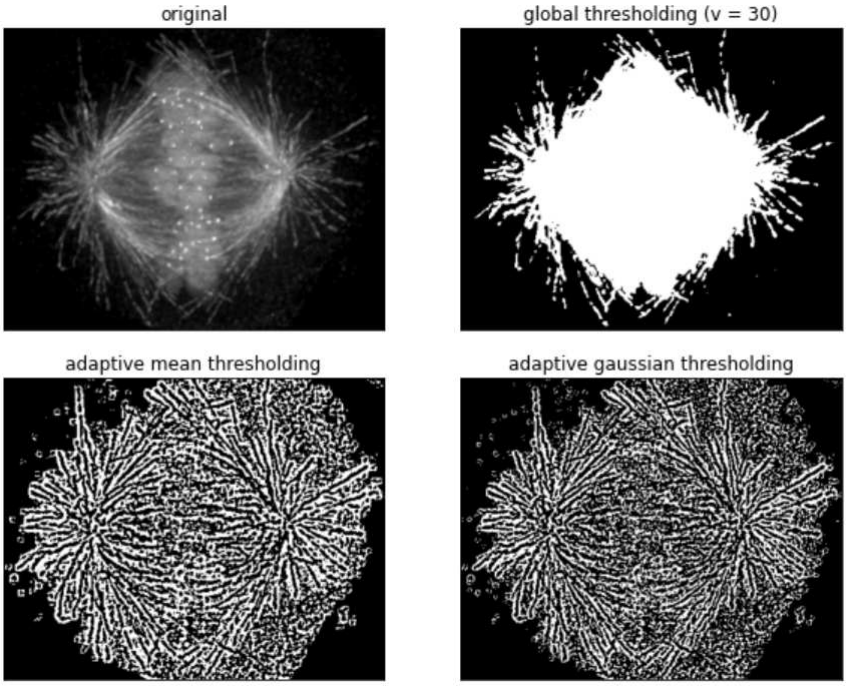
- *adaptiveMethod* - metoda računanja praga, razlikujemo dvije:
  - *cv2.ADAPTIVE_THRESH_MEAN_C* - prag je srednja vrijednost piksela *blockSize* $\times$ *blockSize* okoline umanjena za konstantu *C*
  - *cv2.ADAPTIVE_THRESH_GAUSSIAN_C* - prag je Gaussova težinska srednja vrijednost piksela *blockSize* $\times$ *blockSize* okoline umanjena za konstantu *C*
- *blockSize* - veličina okoline piksela koja određuje dimenzije kernel-a; neparan prirodan broj
- *C* - konstanta koja se oduzima od izračunate srednje vrijednosti intenziteta piksela; u pravilu pozitivna vrijednost, ali može biti i nepozitivna

Metoda vraća promijenjenu sliku nakon odsjecanja.

Na sljedećoj slici 1.4 prikazana je usporedba dvije do sad navedene metode. Kod jednostavnog odsjecanja korišten je prag s vrijednošću 30 te obje metode računanja praga kod prilagodljivog odsjecanja.
Na slikama se jasno vidi da je jednostavna metoda dala bolje rezultate u odnosu na prilagodljivu metodu. Međutim, u daljnjem radu koristimo - *Otsu's binarization* metodu.

Otsuovo odsjecanje (Otsu's binarization)

Kod jednostavne metode odsjecanja biramo proizvoljnu globalnu vrijednost praga. Nasuprot tome, Otsuova metoda izbjegava ručni odabir globalne vrijednosti i određuje ju automatski. Razmotrimo sliku koja ima samo dvije različite vrijednosti piksela (bimodalna slika). Dakle, histogram vrijednosti piksela sastojao bi se samo od dva vrha. Dobar globalni prag bio bi u sredini te dvije vrijednosti. Slično, Otsuova metoda određuje optimalnu vrijednost globalnog praga iz histograma vrijednosti piksela slike sivih tonova.
Prije formalnijeg opisa metode, navodimo definicije dva osnovna pojma iz statistike.



Slika 1.4: Usporedba jednostavne i prilagodljive metode odsjecanja

Definicija 1.2.3. *Neka je $X = \{x_1, ..., x_n\}$ zadani skup podataka. Varijanca, u oznaci σ^2, je statistička mjera disperzije podataka, a računa se kao*

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 ,$$

pri čemu je

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

srednja vrijednost podataka.

Konkretno, neka je V skup svih vrijednosti intenziteta piksela na slici I. Najprije se kreira normalizirani histogram od $L = |V|$ razreda kao reprezentacija distribucije intenziteta piksela slike I dimenzije $M \times N$. Svaki razred odgovaru određenom intenzitetu piksela zadane slike. Pri tome su normalizirane vrijednosti histograma jednake

$$p_i = \frac{n_i}{M \cdot N}, \quad \sum_{i \in V} p_i = 1, \quad p_i \geq 0, \quad \forall i \in V.$$

n_i predstavlja frekvenciju intenziteta i.
Zatim, u ovisnosti o vrijednosti praga t možemo definirati sljedeće klase:

$\forall t \in \{\min(V), ..., \max(V) - 1\}$,

$$C_{t,1} := \{(x, y) : I(x, y) \leq t\}, \quad C_{t,2} := \{(x, y) : I(x, y) > t\}.$$

Klase $C_{t,1}$ i $C_{t,2}$ predstavljaju segmentaciju slike, s ciljem da se svim pikselima klase $C_{t,1}$ intenzitet postavi na 0, a pikselima klase $C_{t,2}$ na 255.
Tad je vjerojatnost da neki piksel pripada klasi $C_{t,1}$ jednaka:

$$P_{t,1} = \sum_{i \in V,\, i \leq t} p_i =: w_{t,1} \ .$$

Očito je onda $P_{t,2} = 1 - P_{t,1} =: w_{t,2}$ te vrijedi:

$$w_{t,1} + w_{t,2} = 1 \ .$$

Također, definiramo i srednje vrijednosti intenziteta u pojedinim klasama:

$$\mu_{t,1} = \frac{\sum\limits_{i \in V,\, i \leq t} p_i \cdot i}{\sum\limits_{i \in V,\, i \leq t} p_i}, \quad \mu_{t,2} = \frac{\sum\limits_{i \in V,\, i > t} p_i \cdot i}{\sum\limits_{i \in V,\, i > t} p_i},$$

pri čemu i predstavlja intenzitet piksela s odgovarajućom frekvencijom p_i, te srednju vrijednost svih intenziteta na slici

$$\mu = \sum_{i \in V} p_i \cdot i = w_{t,1}\mu_{t,1} + w_{t,2}\mu_{t,2} \ .$$

Cilj je odrediti optimalni t^* koji minimizira varijancu među intenzitetima piksela iste klase (en. *intra-class/within-class variance*), definiranu kao

$$\sigma^2_{t,W} := w_{t,1}\sigma^2_{t,1} + w_{t,2}\sigma^2_{t,2} \ .$$

Dakle, radi se o težinskoj sumi varijanca obje klase. Intuitivno, globalni prag je bolji ukoliko je varijanca intenziteta piksela unutar klase manja.
Alternativno, ekvivalentan uvjet je maksimizirati varijancu između klasa (en. *inter-class/between class variance*):

$$\sigma^2_{t,B} = \sigma^2 - \sigma^2_{t,W} \ ,$$

pri čemu je σ^2 varijanca intenziteta piksela slike I:

$$\sigma^2 = \frac{\sum\limits_{i \in V} n_i (i - \mu)^2}{M \cdot N} \ .$$

Za varijancu između klasa može se pokazati da vrijedi jednakost:

$$\sigma^2_{t,B} = w_{t,1} (\mu_{t,1} - \mu)^2 + w_{t,2} (\mu_{t,2} - \mu)^2$$

Dokaz.

$$\begin{aligned}
\sigma^2_{t,B} &= \sigma^2 - \sigma^2_{t,W} \\
&= \frac{\sum\limits_{i\in V} n_i\,(i-\mu)^2}{M\cdot N} - \left(w_{t,1}\sigma^2_{t,1} + w_{t,2}\sigma^2_{t,2}\right) \\
&= \sum_{i\in V} p_i\,(i-\mu)^2 - \left(w_{t,1}\frac{\sum\limits_{i\in V,\, i\le t} n_i\,(i-\mu_{t,1})^2}{M\cdot N\cdot w_{t,1}} + w_{t,2}\frac{\sum\limits_{i\in V,\, i>t} n_i\,(i-\mu_{t,2})^2}{M\cdot N\cdot w_{t,2}}\right) \\
&= \sum_{i\in V} p_i\,(i-\mu)^2 - \sum_{i\in V,\, i\le t} p_i\,(i-\mu_{t,1})^2 - \sum_{i\in V,\, i>t} p_i\,(i-\mu_{t,2})^2 \\
&= \sum_{i\in V,\, i\le t} p_i\,(i-\mu)^2 - \sum_{i\in V,\, i\le t} p_i\,(i-\mu_{t,1})^2 + \sum_{i\in V,\, i>t} p_i\,(i-\mu)^2 - \sum_{i\in V,\, i>t} p_i\,(i-\mu_{t,2})^2 \\
&= (*)
\end{aligned}$$

Za prve dvije sume dalje vrijedi:

$$\begin{aligned}
\sum_{i\in V,\, i\le t} p_i\,(i-\mu)^2 - \sum_{i\in V,\, i\le t} p_i\,(i-\mu_{t,1})^2 &= \sum_{i\in V,\, i\le t} p_i\left((i-\mu)^2 - (i-\mu_{t,1})^2\right) \\
&= \sum_{i\in V,\, i\le t} p_i\left(\mu^2 - \mu^2_{t,1} + i\,(2\mu_{t,1} - 2\mu)\right) \\
&= \left(\mu^2 - \mu^2_{t,1}\right)\cdot \sum_{i\in V,\, i\le t} p_i + (2\mu_{t,1} - 2\mu)\cdot \sum_{i\in V,\, i\le t} p_i\cdot i \\
&= \left(\mu^2 - \mu^2_{t,1}\right)\cdot w_{t,1} + (2\mu_{t,1} - 2\mu)\cdot \frac{\sum\limits_{i\in V,\, i\le t} p_i\cdot i}{\sum\limits_{i\in V,\, i\le t} p_i}\cdot \sum_{i\in V,\, i\le t} p_i \\
&= \left(\mu^2 - \mu^2_{t,1}\right)\cdot w_{t,1} + (2\mu_{t,1} - 2\mu)\cdot \mu_{t,1} w_{t,1} \\
&= w_{t,1}\left(\mu^2 - \mu^2_{t,1} + 2\mu^2_{t,1} - 2\mu\mu_{t,1}\right) \\
&= w_{t,1}\,(\mu - \mu_{t,1})^2 \\
&= w_{t,1}\,(\mu_{t,1} - \mu)^2
\end{aligned}$$

Analogno dobivamo za preostale dvije sume:

$$\sum_{i \in V,\ i>t} p_i \left(i-\mu\right)^2 - \sum_{i \in V,\ i>t} p_i \left(i-\mu_{t,2}\right)^2 = w_{t,2}\left(\mu_{t,2}-\mu\right)^2$$

Konačno, zbrajanjem tih dvaju rezultata dobivamo:

$$(*) = w_{t,1}\left(\mu_{t,1}-\mu\right)^2 + w_{t,2}\left(\mu_{t,2}-\mu\right)^2$$

□

Dalje dobivamo:

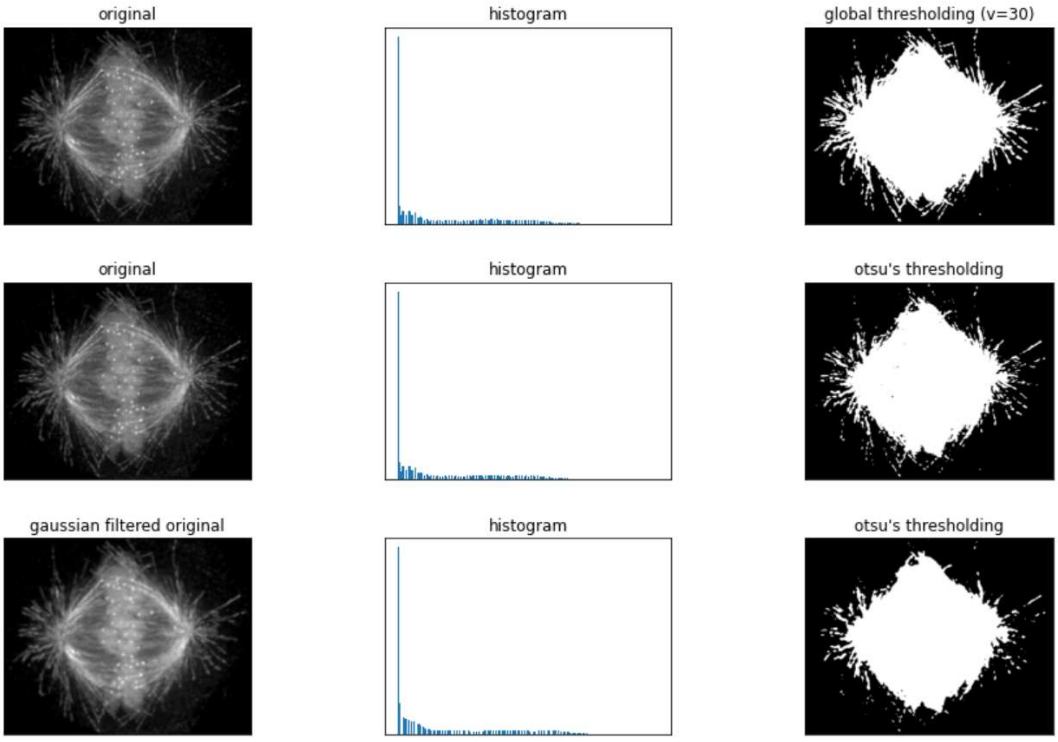
$$\begin{aligned}
\sigma_{t,B}^2 &= w_{t,1}\left(\mu_{t,1}-\mu\right)^2 + w_{t,2}\left(\mu_{t,2}-\mu\right)^2 \\
&= w_{t,1}\mu_{t,1}^2 - 2w_{t,1}\mu_{t,1}\mu + w_{t,1}\mu^2 + w_{t,2}\mu_{t,2}^2 - 2w_{t,2}\mu_{t,2}\mu + w_{t,2}\mu^2 \\
&= w_{t,1}\mu_{t,1}^2 + w_{t,2}\mu_{t,2}^2 + \left(w_{t,1}+w_{t,2}\right)\mu^2 - 2\left(w_{t,1}\mu_{t,1} + w_{t,2}\mu_{t,2}\right)\mu \\
&= w_{t,1}\mu_{t,1}^2 + w_{t,2}\mu_{t,2}^2 - \mu^2 \\
&= w_{t,1}\mu_{t,1}^2 + w_{t,2}\mu_{t,2}^2 - w_{t,1}^2\mu_{t,1}^2 - w_{t,2}^2\mu_{t,2}^2 - 2w_{t,1}\mu_{t,1}w_{t,2}\mu_{t,2} \\
&= w_{t,1}\mu_{t,1}^2\left(1-w_{t,1}\right) + w_{t,2}\mu_{t,2}^2\left(1-w_{t,2}\right) - 2w_{t,1}\mu_{t,1}w_{t,2}\mu_{t,2} \\
&= w_{t,1}w_{t,2}\mu_{t,1}^2 + w_{t,1}w_{t,2}\mu_{t,2}^2 - 2w_{t,1}\mu_{t,1}w_{t,2}\mu_{t,2} \\
&= w_{t,1}w_{t,2}\left(\mu_{t,1}^2 - 2\mu_{t,1}\mu_{t,2} + \mu_{t,2}^2\right) \\
&= w_{t,1}w_{t,2}\left(\mu_{t,1}-\mu_{t,2}\right)^2
\end{aligned}$$

Dakle, efikasnije je tražiti optimalni t^* koji maksimizira varijancu između klasa. Postupak je sljedeći:

1. Odredi normalizirani histogram
2. Za inicijalni $t = \min(V)$ izračunaj $w_{t,j}$, $\mu_{t,j}$ i $\sigma_{t,B}^2$, $\forall j \in \{1,2\}$
3. $\forall t \in \{\min(V)+1, ..., \max(V)-1\}$:
   a) Ažuriraj $w_{t,j}$ i $\mu_{t,j}$, $\forall j \in \{1,2\}$
   b) Izračunaj $\sigma_{t,B}^2$
4. Optimalni t^* je $\max\limits_t \sigma_{t,B}^2$

Programska realizacija ostvaruje se korištenjem funkcije *cv2.threshold*, uz dodatak zastavice (en. *flag*) *cv2.THRESH_OTSU*. Također, moguće je i ručno specificirati vrijednost globalnog praga, međutim, algoritam pronalazi globalnu optimalnu vrijednost praga i vraća ju kao prvu povratnu vrijednost. Druga povratna vrijednost je, slično kao i kod jednostavne metode, promijenjena slika.

Na idućoj slici 1.5 nalaze se rezultati jednostavne te Otsuove metode odsjecanja. Također, može se vidjeti i kako uklanjanje šuma Gaussovim filterom ponekad može pomoći pri određivanju bolje segmentacije slike.



Slika 1.5: Usporedba Otsuove i jednostavne metode odsjecanja

Detaljnije o *OpenCV thresholding* metodama može se pronaći u [8] i [20].

1.3 Određivanje konture mitotičkog vretena

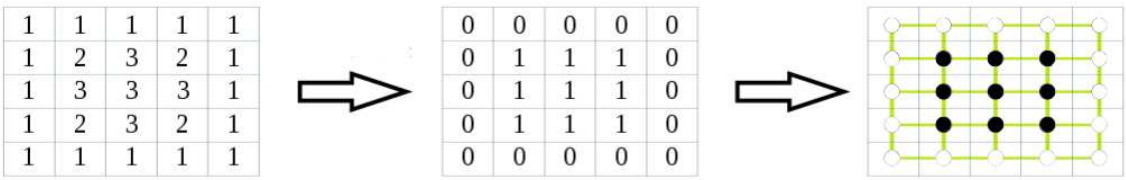
U prošlom potpoglavlju smo zamutili sliku sivih tonova, u određenoj mjeri, Gaussovim filterom te nakon toga primijenili Otsuovu metodu odsjecanja. Tako smo sliku segmentirali

na dva dijela - područje bijele boje koje odgovara mitotičkom vretenu te pozadinu crne boje. To olakšava detekciju i iscrtavanje konture (vanjskog ruba) mitotičkog vretena na slici. U tu svrhu koristimo funkciju *find_contours* ([2]) iz paketa *skimage.measure*, koja je implementirana pomoću *marching squares* algoritma.

Marching squares algoritam

Radi se o algoritmu računalne grafike koji generira konture za dvodimenzionalno polje skalara (”pravokutni” *array* numeričkih vrijednosti).

Algoritam ([10]) najprije primjenjuje metodu odsjecanja na ulaznu sliku I dimenzija $M \times N$ te iz nje kreira binarnu. Dakle, vrijednosti intenziteta piksela ispod zadanog praga postavljaju se na 0, odnosno na 1 onim pikselima čija je vrijednost intenziteta iznad praga. Zatim, svaki 2×2 blok matrice piksela I čini jednu ćeliju konture (obojano zelenim bridovima na slici 1.6). Tako dobivamo $(M-1) \times (N-1)$ reprezentaciju polazne slike pomoću ćelija konture. Svaki vrh (čvor) ćelije možemo obojati dvjema bojama, a u ovom slučaju, radi ilustracije, crna boja ispune sugerira da vrh pripada unutar konture. Također, dva vrha zvat ćemo *bipolarnima* ako su obojani različitim bojama.

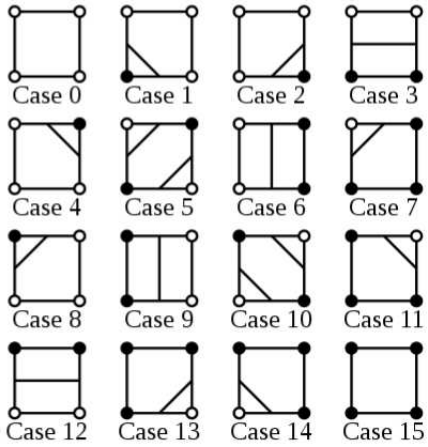


Slika 1.6: Ilustracija konstrukcije ćelija iz polazne slike - *marching squares*

Nakon toga, za svaku ćeliju dobivene reprezentacije provodi se sljedeći postupak:

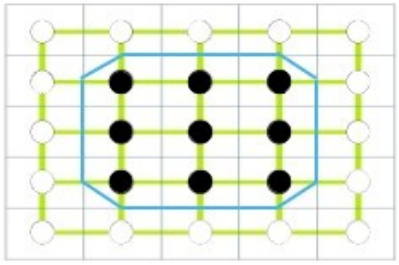
1. Generiramo niz od 4 bita obilaskom vrhova ćelije u smjeru obrnutom kazaljke na satu, počevši od gornjeg lijevog vrha ćelije. Svaki vrh ćelije reprezentira jedan piksel binarne slike te mu odgovara točno jedan bit. Primjerice, na slici 1.6, prvoj ćeliji (gornji lijevi kut) odgovara niz 0010. Ukupno postoji $2^4 = 16$ mogućih binarnih nizova duljine 4 jer za svaku poziciju u nizu možemo odabrati dvije vrijednosti.

2. Dobiveni binarni zapis pretvorimo u dekadski te u bazi pod tim rednim brojem pogledamo brid koji reprezentira ćeliju. Dakle, u bazi čuvamo 16 mogućih pozicija bridova prikazanih na slici 1.7.



Slika 1.7: Numerirane pozicije bridova

Zatim, u svaku ćeliju ucrtamo odgovarajući brid, tako što ga spojimo sa dva polovišta stranica ćelije koje presjeca. Odnosno, krajnje točke brida su polovišta stranica između bipolarnih čvorova.



Slika 1.8: Dobivena kontura nakon prva dva koraka

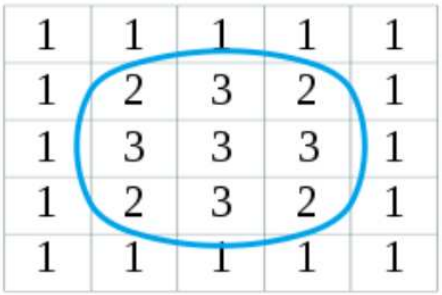
3. Koristimo linearnu interpolaciju s ciljem pronalaska prikladnijih krajnjih točaka bridova koje nisu nužno na polovištima stranica ćelije. Efektivno se dobije finija, zaglađenija kontura. Dakle, za dva bipolarna čvora P_1 i P_2 s odgovarajućim intenzitetima v_1 i v_2, računamo krajnju točku brida P kao težinski prosjek

$$P = (1 - t)\, P_1 + tP_2 \ ,$$

pri čemu je

$$t = \frac{threshold - \min(v1, v2)}{\max(v1, v2) - \min(v1, v2)} \ .$$

Slijedi konačni rezultat:



Slika 1.9: Dobivena kontura nakon linearne interpolacije

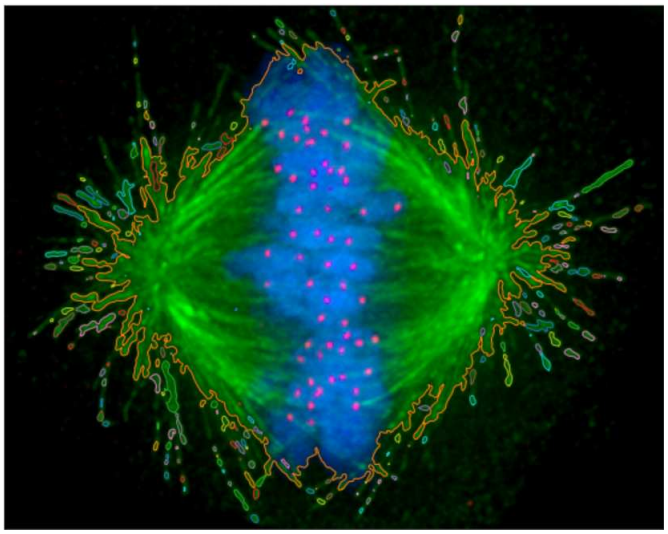
Međutim, na slici 1.7 može se primijetiti problem višeznačnosti za slučajeve 5 i 10. Radi se o tzv. sedlastim točkama. Odnosno, nije točno određeno kako ucrtati paralelene bridove jer kad odredimo jednu krajnju točku jednog brida (nalazi se između dva bipolarna čvora), onda imamo dvije opcije za drugu krajnju točku istog brida. Tako možemo bridove iz slučaja 10 ucrtati u ćeliju slučaja 5 i obrnuto. Problem se može riješiti dodavanjem dodatne informacije - u centar ćelije dodamo novi čvor tako što izračunamo prosječnu vrijednost intenziteta preostala četiri vrha te mu odredimo $0/1$ aktivaciju pomoću *thresholda*. Tad dobivamo točno jednu dijagonalu sa sva tri vrha iste boje koja onda određuje smjer bridova.

Programska realizacija algoritma je već spomenuta, koristimo funkciju *find_contours* koja prima pet parametra:

- *image* - ulazna slika; *ndarray*
- *level* - *threshold*; *float*; opcinalno
- *fully_connected* - rješava problem višeznačnosti, tj. proslijeđena vrijednost naznačuje koje vrijednosti (ispod ili iznad *thresholda*) su povezane dijagonalom; {”low”, ”high”} - *string*; opcinalno
- *positive_orientation* - naznačuje oko kojih vrijednosti (ispod ili iznad *thresholda*) se u smjeru kontra kazaljke na satu opisuje kontura; {”low”, ”high”} - *string*; opcinalno
- *mask* - *bool* maska, *True* naznačuje sudjeluje li točka u crtanju konture; *2D ndarray boole*-ova ili *None*; opcinalno

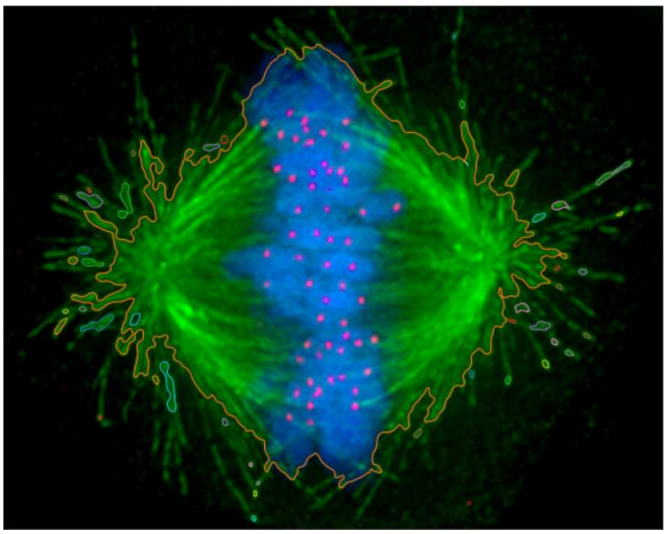
Povratna vrijednost funkcije je lista svih pronađenih kontura, pri čemu je svaka kontura tipa $(n, 2)$ – *ndarray* od n točaka (parova koordinata).

Pozivom funkcije *find_contours* nad slikom sivih tonova, uz Otsuovu vrijednost praga, dobivamo sljedeće konture:



Slika 1.10: Pronađene konture na slici vretena - *marching squares*

Na idućoj slici može se vidjeti kako zamućivanje Gaussovim filterom pomaže i u pronalasku relevantnijih kontura. Koristi se zamućena slika s odgovarajućim Otsuovim pragom.



Slika 1.11: Konture vretena nakon zamućenja - *marching squares*

Podešavanjem parametara funkcije *GaussianBlur*, točnije, govorimo o veličini filtera i standardnoj devijaciji, dobit ćemo i različite konture nakon primjene *marching squares* algoritma.

Površina unutar konture

Iako *marching squares* daje više od jedne konture, nas zanima samo vanjska kontura najveće površine, odnosno ona koja najbolje ocrtava rubove vretena. U tu svrhu koristimo *cv2.contourArea* funkciju te biramo konturu najveće površine. Spomenuta funkcija prima dva parametra - *contour* (*ndarray* koordinata odgovarajuće konture) i *oriented* (*boolean* koji naznačuje je li kontura orijentirana). Očekivano, funkcija vraća površinu unutar konture. Implementacija je inspirirana Greenovim teoremom koji omogućuje računanje površine unutar određene konture koristeći krivuljni integral umjesto plošnog integrala po području unutar konture ([16]).

Teorem 1.3.1. *(Greenov teorem)*
Neka je C pozitivno orijentirana, jednostavna, zatvorena krivulja u ravnini i neka je D područje ograničeno krivuljom C. Ako L i M imaju neprekidne parcijalne derivacije na otvorenoj domeni koja sadrži D, onda vrijedi:

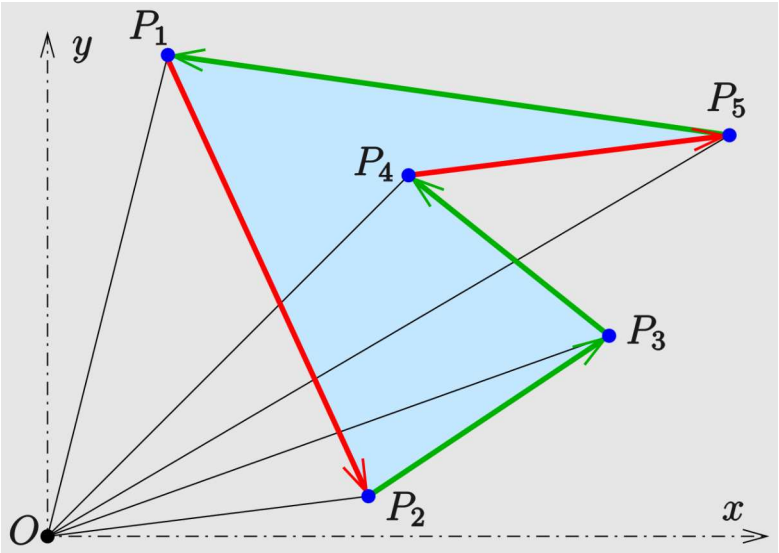
$$\int_C L\,dx + M\,dy = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right) dA\ .$$

Međutim, digitalna slika je konačna funkcija na diskretnoj domeni, stoga se koristi diskretna verzija formule ([12]) za računanje površine unutar konture zadane konačnim nizom koordinata:

$$A = \frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1}y_i).$$

Naime, za zadani konačni niz točaka $P_1(x_1, y_1), ..., P_n(x_n, y_n)$ te ishodište $O(0, 0)$, uz dogovor da je $P_{n+1} = P_1$, površinu unutar poligona određenog točkama P_i možemo dobiti kao sumu orijentiranih površina trokuta $\triangle_{OP_iP_{i+1}}$. Površina trokuta ima pozitivan predznak ako je pozitivno orijentirana (obrnuto smjeru kazaljke na satu), a u suprotnom ima negativan predznak. Na slici 1.12 nalazi se ilustracija, gdje zeleni bridovi označavaju da se radi o pozitivnoj površini pripadnog trokuta, a crveni bridovi negativnoj. Obzirom da je površina A_i pojedinog trokuta jednaka polovini površine paralelograma kojeg čine vektori $\overrightarrow{OP_i}$ i $\overrightarrow{OP_{i+1}}$, dobivamo da je

$$A_i = \frac{1}{2}\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix} = \frac{1}{2}\left(x_i y_{i+1} - y_i x_{i+1}\right),$$



Slika 1.12: Orijentirane površine

budući da površinu paralelograma možemo dobiti koristeći vektorski produkt odgovarajućih vektora, odnosno determinantu sačinjenu od koordinata vektora.
Općenito, za dva vektora $\mathbf{a} = (a_1, a_2, a_3)$, $\mathbf{b} = (b_1, b_2, b_3) \in \mathbb{R}^3$ možemo zapisati vektorski produkt kao

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}.$$

Stavimo $a_3 = b_3 = 0$ pa se izraz pojednostavljuje:

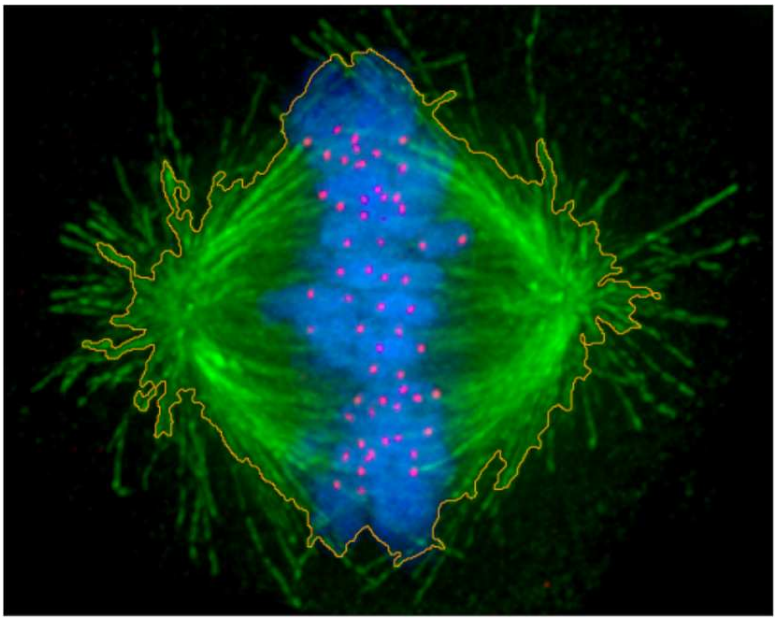
$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \mathbf{k}\,.$$

Površina paralelograma tad je jednaka normi vektorskog produkta, odnosno apsolutnoj vrijednosti dobivene determinante.

Nakon što imamo izračunatu orijentiranu površinu svakog trokuta, dalje jednostavno dobivamo traženu formulu za ukupnu površinu:

$$A = \sum_{i=1}^{n} A_i.$$

Rezultat korištenja *contourArea* funkcije prikazan je na slici 1.13.



Slika 1.13: Vanjska kontura vretena

1.4 Aproksimacija konture B-splajn krivuljom

Dobivenu vanjsku konturu vretena možemo ”zagladiti” tako da ju aproksimiramo nekom funkcijom. Time očekujemo dobiti preciznije rezultate lociranja centrosoma i računanja kuta otvora vretena. Obzirom da nemamo neku jasnu teorijsku osnovu za odabir konkretne aproksimacijske funkcije, možemo iskoristiti splajnove kao opću familiju aproksimacijskih funkcija. Odnosno, koristit ćemo B-splajn interpolaciju konture ([19]).

B-splajn interpolacija

Definicija 1.4.1. *Neka su zadane točke* $x_0, x_1, ..., x_m$ *u rastućem poretku. Splajn funkcija* φ *reda n je po dijelovima polinomna funkcija stupnja* $n-1$*, tj.* $\varphi|_{[x_{k-1},\, x_k]} = p_k,\ k \in \{1, ..., m\}$*. Pri tome su, najčešće,* p_k *polinomi stupnjeva* $1, 2, 3$ *ili* 5*. Točke* $x_0, x_1, ..., x_m$ *u kojima se dijelovi spajaju zovemo čvorovima interpolacije.*

Definicija 1.4.2. *Za čvorove* $t_0, t_1, ..., t_m$*, B-splajn reda* 1 *definiramo kao funkciju*

$$B_{i,1}(x) := \begin{cases} 1, & t_i \le x < t_{i+1} \\ 0, & \textit{inače} \end{cases}$$

za koju vrijedi

$$\sum_i B_{i,1}(x) = 1, \quad \forall x.$$

B-splajn reda $k+1$ *definiramo rekurzivno:*

$$B_{i,k+1}(x) := w_{i,k}(x)B_{i,k}(x) + \left(1 - w_{i+1,k}(x)\right) B_{i+1,k}(x),$$

gdje su

$$w_{i,k}(x) := \begin{cases} \dfrac{x - t_i}{t_{i+k} - t_i}, & t_{i+k} \neq t_i \\ 0, & \textit{inače}. \end{cases}$$

Važno svojstvo B-splajnova je uloga baznih funkcija na prostoru splajnova, odnosno svaki splajn reda n na danoj mreži čvorova može se prikazati kao linearna kombinacija baznih funkcija.

Programska realizacija interpolacije zadane konture kao diskretnog skupa točaka ostvaruje se korištenjem funkcije *splprep* programske biblioteke *scipy.interpolate* ([1]). Funkcija pronalazi parametriziranu krivulju, odnosno B-splajn reprezentaciju konture. Postoji nekoliko parametara, a nama su najvažnija sljedeća dva:

- x - niz točaka koji predstavlja konturu; *array*
- s - parametar glatkoće parametrizirane krivulje; ako se eksplicitno ne specifira, uzima se $s = m - \sqrt{2m}$, pri čemu je m broj točaka konture; *float*

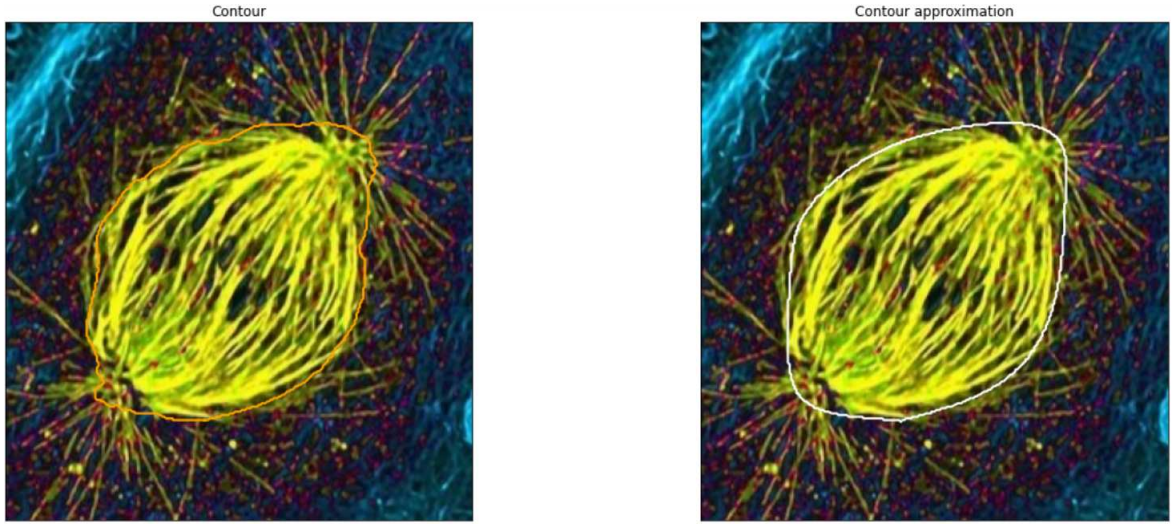
Također, funkcija vraća nekoliko povratnih vrijednosti, od kojih su nam najbitnije:

- *tck* - uređena trojka čvorova, koeficijenata B-splajna i stupanj splajna; *tuple*
- u - niz vrijednosti parametara u segmentu $[0, 1]$; *array*

Za evaluaciju dobivenog B-splajna koristimo funkciju *splev* iz iste biblioteke. Također postoji nekoliko parametara, a zanimaju nas:

- x - niz točaka u kojima želimo evaluirati B-splajn, obično je to povratna vrijednost u funkcije *splprep*; *array*
- *tck* - uređena trojka dobivena iz funkcije *splprep*; *tuple*

Na idućoj slici 1.14 nalazi se primjerak vanjske konture dobivene pomoću funkcije *find_contours*, uz korištenje optimalnog globalnog Otsuovog praga, i pripadne B-splajn aproksimacije. Da bismo postigli veću zaglađenost aproksimacijske krivulje, korišten je parametar $s = 2.5m$.



Slika 1.14: Lijevo: vanjska kontura, desno: B-splajn reprezentacija konture

1.5 Elipsa kao model konture vretena

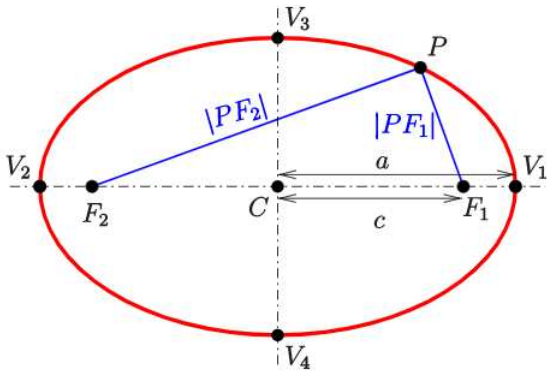
Podsjetimo se najprije jednostavne definicije elipse.

Definicija 1.5.1. *Neka su F_1 i F_2 dvije fiksirane točke u $\mathbb{R}^2$, koje zovemo fokusima (žarištima), te $a \in \mathbb{R}$ za koji vrijedi $2a > |F_1F_2|$. Elipsu definiramo kao skup točaka:*

$$E = \{P \in \mathbb{R}^2 \ : \ |PF_1| + |PF_2| = 2a\} \ .$$

Dužinu koja prolazi kroz fokuse zovemo velika os elipse, dok dužina koja je okomita na veliku os, a prolazi centrom elipse, zovemo mala os elipse. Centar je točka koja je polovište male i velike osi, a vrijednost $c = \sqrt{a^2 - b^2}$ zovemo žarišna udaljenost.

Nakon aproksimacije konture B-splajnom, vrijeme je da odredimo lokacije centrosoma. Grubo govoreći, dvodimenzionalni prikaz vretena na digitalnoj slici ima, u određenoj mjeri, oblik elipse. Stoga, kao jedna od opcija nameće se pronalazak elipse koja najbolje opisuje vreteno. Odnosno, cilj je pronaći elipsu koja najbolje opisuje aproksimiranu konturu iz prošlog potpoglavlja. Također, u tom kontekstu za očekivati je da tjemena elipse uz veliku os odgovaraju polovima vretena. Samim tim se velika os nameće kao dobar izbor pozicioniranja centrosoma. Točnije, centrosome ćemo tražiti na velikoj osi dobivene elipse, počevši od oba tjemena prema centru.



Slika 1.15: Elipsa

fitEllipse algoritam

Za pronalazak tražene elipse, koristimo funkciju *cv2.fitEllipse* koja kao parametar prima *ndarray* $2D$ točaka te vraća rotirani pravokutnik u kojem je upisana elipsa ([4]). Odnosno, raspakiranjem povratne vrijednosti možemo dobiti:

- Koordinate centra elipse
- Duljine velike i male osi elipse
- Kut rotacije elipse u stupnjevima

Prije nego navedemo opis algoritma, potrebno nam je nekoliko definicija ([14], [17]).

Definicija 1.5.2. *Neka je V vektorski prostor nad poljem $\mathbb{F}$. Skalarni produkt na V je preslikavanje $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{F}$ koje ima sljedeća svojstva:*

1. $\langle x, x \rangle \geq 0, \ \forall x \in V$
2. $\langle x, x \rangle = 0 \Longleftrightarrow x = 0$
3. $\langle x_1 + x_2, y \rangle = \langle x_1, y \rangle + \langle x_2, y \rangle, \ \forall x_1, x_2, y \in V$
4. $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle, \ \forall \alpha \in \mathbb{F}, \ \forall x, y \in V$
5. $\langle x, y \rangle = \overline{\langle y, x \rangle}, \ \forall x, y \in V.$

Definicija 1.5.3. *Vektorski prostor na kojemu je definiran skalarni produkt zovemo unitaran vektorski prostor. Neka je V unitaran prostor. Norma na V je funkcija $\| \cdot \| : V \to \mathbb{R}$ definirana s $\|x\| = \sqrt{\langle x, x \rangle}$.*

Definicija 1.5.4. *Neka je V vektorski prostor. Preslikavanje $d : V \times V \to \mathbb{R}$ zadano s $d(x, y) = \|x - y\|$ i svojstvima:*

1. $d(x, y) \geq 0, \ \forall x, y \in V$
2. $d(x, y) = 0 \Longleftrightarrow x = y$
3. $d(x, y) = d(y, x), \ \forall x, y \in V$
4. $d(x, y) \leq d(x, z) + d(z, y), \ \forall x, y, z \in V$

zovemo metrika na V.

Definicija 1.5.5. *Algebarske ravninske krivulje drugoga reda nastale presjekom ravnine i kružne dvostruke stožaste plohe. To su kružnica, elipsa, parabola, hiperbola te njihove degeneracije: par ukrštenih pravaca (ako presječna ravnina prolazi kroz vrh stošca), odnosno par usporednih pravaca (ako se vrh stošca pomakne u beskonačnost). U pravokutnom Kartezijevu koordinatnom sustavu konike su određene općom jednadžbom:*

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

Definicija 1.5.6. *Neka je V vektorski prostor nad poljem $\mathbb{F}$ i $A \in L(V)$. Kažemo da je skalar $\lambda_0 \in \mathbb{F}$ svojstvena vrijednost operatora A ako postoji vektor $x \in V, \ x \neq 0$, takav da je $Ax = \lambda_0 x$. Vektor x naziva se svojstveni vektor pridružen svojstvenoj vrijednosti λ_0.*

Uz navedene definicije potrebna nam je i tzv. metoda Lagrangeovih multiplikatora. Radi se o metodi pomoću koje možemo pronaći lokalne ekstreme funkcije uz ograničenja (restrikcija na domenu funkcije čije ekstreme tražimo). Preciznije, da bismo pronašli ekstreme funkcije $f(\mathbf{x})$ uz dodatni uvjet $g(\mathbf{x}) = 0$, formiramo Lagrangeovu funkciju

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

i pronalazimo stacionarne točke od $\mathcal{L}$ u ovisnosti o $\mathbf{x}$ i λ.

Primjer 1.5.7. *Jednostavan primjer korištenja Lagrangeove metode.*

$$\left.\begin{array}{lll} x^2 + y^2 & \to & \textit{ekstr.} \\ x + y = 1 & & \end{array}\right\}$$

$$F(x, y, \lambda) := x^2 + y^2 + \lambda(x + y - 1)$$

$$\left.\begin{aligned}\frac{\partial F}{\partial x} &= 2x+\lambda = 0\\ \frac{\partial F}{\partial y} &= 2y+\lambda = 0\\ \frac{\partial F}{\partial \lambda} &= x+y-1 = 0\end{aligned}\right\} \quad \Rightarrow \quad x=y=\frac{-\lambda}{2} \quad \Rightarrow \quad x=y=\frac{1}{2} \quad \Rightarrow \quad \lambda=-1$$

$$\Rightarrow \quad F(x,y,-1) = x^2+y^2-x-y+1$$

$$\frac{\partial^2 F}{\partial x^2}=2, \quad \frac{\partial^2 F}{\partial y^2}=2, \quad \frac{\partial^2 F}{\partial x \partial y}=0; \quad D=\begin{vmatrix}2 & 0\\ 0 & 2\end{vmatrix}=4>0$$

$D>0,\ \frac{\partial^2 F}{\partial x^2}>0 \quad \Rightarrow \quad$ *točka* $M_0\left(\frac{1}{2},\frac{1}{2}\right)$ *je točka lokalnog minimuma.*

Slijedi formulacija problema pronalaska tražene elipse ([15]). Neka su:

- $P=\{\mathbf{x}_1,...,\mathbf{x}_n\},\ \mathbf{x}_i=(x_i,y_i)\,,\ \forall i\in\{1,...,n\}$
- Familija krivulja $C(\mathbf{a})$ parametriziranih vektorom $\mathbf{a}$
- Metrika $\delta(C(\mathbf{a}),\mathbf{x})$, koja mjeri udaljenost točke $\mathbf{x}$ od krivulje $C(\mathbf{a})$

Želimo pronaći optimalni $\mathbf{a}^*=\mathbf{a}_{\min}$ za koji funkcija greške

$$\epsilon^2(\mathbf{a}) = \sum_{i=1}^{n} \delta(C(\mathbf{a}),\mathbf{x}_i)$$

postiže globalni minimum. Tad krivulja $C(\mathbf{a}_{\min})$ najbolje opisuje dane podatke.
Navedena familija krivulja predstavlja familiju konika zadanih u implicitnoj formi

$$C(\mathbf{a}) = \{\mathbf{x} \mid F(\mathbf{a};\mathbf{x})=0\}$$

pri čemu je

$$F(\mathbf{a};\mathbf{x}) = A_{xx}x^2+A_{xy}xy+A_{yy}y^2+A_x x+A_y y+A_0\ .$$

Alternativno, možemo koristiti i kompaktni zapis:

$$F(\mathbf{a};\mathbf{x}_i) = [x_i^2\ x_iy_i\ y_i^2\ x_i\ y_i\ 1]\cdot[A_{xx}\ A_{xy}\ A_{yy}\ A_x\ A_y\ A_0\ ]^T = \mathbf{X}_i\cdot\mathbf{a}^T\ .$$

Funkcija *fitEllipse* je implementirana pomoću tzv. algoritma algebarske udaljenosti ([15]), koji minimizira funkciju greške

$$\epsilon^2(\mathbf{a}) = \sum_{i=1}^{n} F(\mathbf{a},\mathbf{x}_i)^2 = \|D\mathbf{a}\|^2\ ,$$

uz uvjet $\|\mathbf{a}\|^2 = 1$. D zovemo *matrica dizajna*, dimenzije su joj $n \times 6$ te svaki od n redaka čini jedan vektor $\mathbf{X}_i$. Zapravo nam je cilj analitički minimizirati funkciju

$$E(\mathbf{a}) = \|D\mathbf{a}\|^2 - \lambda(\|\mathbf{a}\| - 1) = \mathbf{a}^T D^T D\mathbf{a} - \lambda(\mathbf{a}^T\mathbf{a} - 1)$$

tako što rješavamo problem svojstvenih vektora:

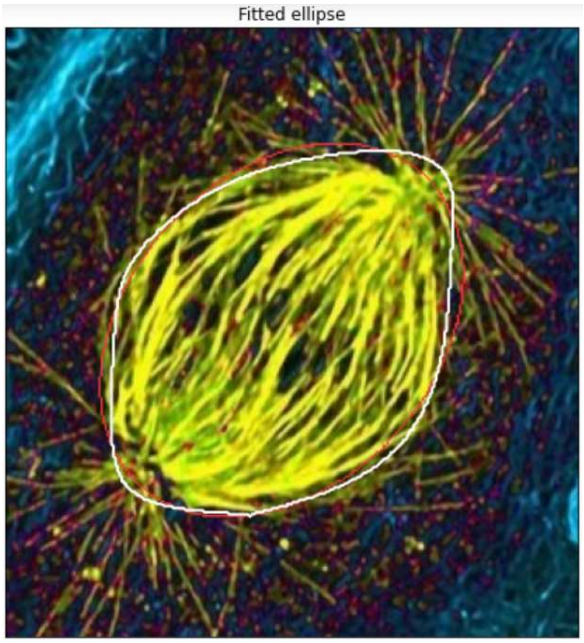
$$\begin{aligned} \nabla_{\mathbf{a}} E = 0 &\iff 2D^T D\mathbf{a} - 2\lambda\mathbf{a} = 0 \\ &\iff D^T D\mathbf{a} = \lambda\mathbf{a} \end{aligned}$$

pri čemu je λ Lagrangeov multiplikator. Konačno, dobivamo da je $\mathbf{a}_{\min}$ svojstveni vektor od $D^T D$ koji odgovara najmanjoj svojstvenoj vrijednosti.

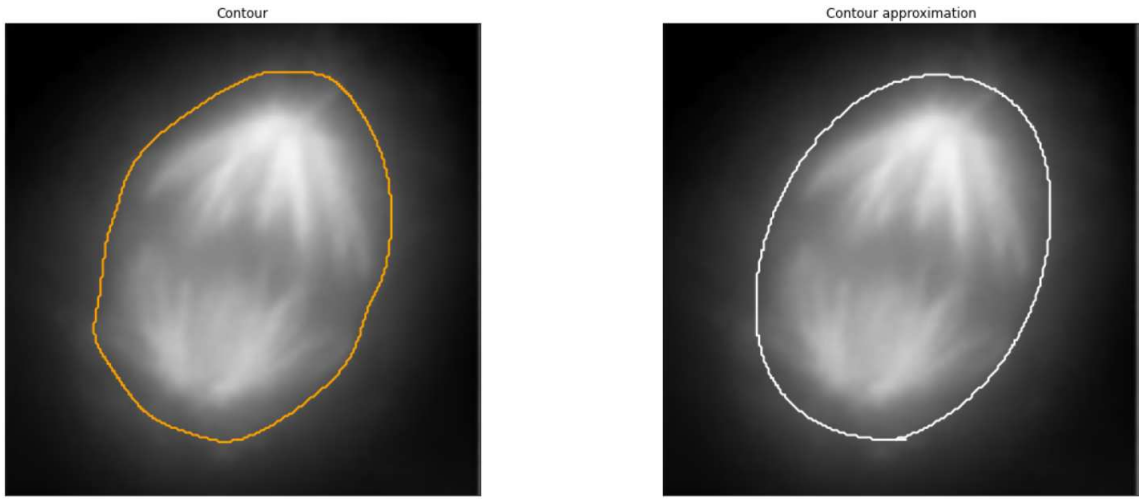
Onog trenutka kad nađemo odgovarajuću elipsu, možemo ju prikazati na slici pomoću funkcije *cv2.Ellipse*. Funkcija kao parametre prima sliku, rotirani pravokutnik (u ovom slučaju povratna vrijednost *fitEllipse* funkcije), boju elipse, debljinu krivulje (u slučaju negativne vrijednosti bojom se ispuni i područje unutar elipse) te tip linije kojom se crta krivulja.

Uz dobivenu elipsu, želimo iscrtati i veliku os jer na njoj tražimo centrosome. U tu svrhu računamo koordinate tjemena na početku i kraju velike osi. To lako pronalazimo korištenjem polarnih koordinata, uz dane koordinate centra elipse, duljinu velike poluosi te kut rotacije. Tada možemo iskoristiti funkciju *cv2.line* koja prima iste parametre kao i *cv2.Ellipse*, osim što umjesto rotiranog pravokutnika, u ovom slučaju, kao parametre šaljemo koordinate izračunatih tjemena elipse. Na slici 1.16 prikazana je elipsa koja najbolje opisuje B-splajn krivulju.

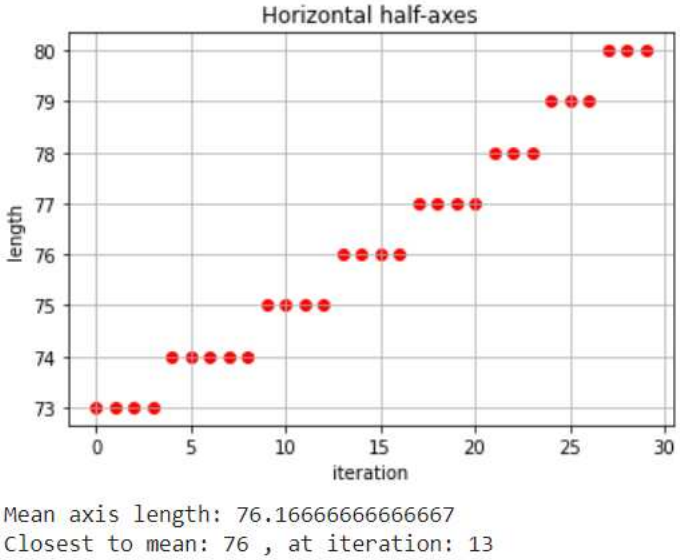
Distribucija piksela kod nekih slika vretena ponekad nije idealna za primjenu Otsuove metode jer pripadni histogram ne prikazuje dvije očite dominantne vrijednosti intenziteta, nego samo jednu. Najčešće je to mala vrijednost bliža nuli, budući da je većina slike tamno područje. U tom slučaju dobiveni globalni Otsuov prag ne određuje idealnu konturu kakvu bismo željeli. U tu svrhu variramo dobiveni prag p, odnosno isprobavamo cjelobrojne vrijednosti iz nekog manjeg intervala $[p - t_1, p + t_2]$. Za svaku vrijednost novog praga dobivamo drugačije konture (veće ili manje) pa samim tim i drugačije duljine velikih osi elipsa koje ih najbolje opisuju. Tako možemo promatrati niz duljina velikih osi i uzeti srednju vrijednost te konačno odrediti konturu čija je duljina velike osi najbliža izračunatoj srednjoj vrijednosti. Time automatiziramo pronalazak kontura vretena te izbjegavamo ručno podešavanje optimalnog praga za neke slučajeve, ali ponekad na uštrb preciznosti. Na slikama 1.17 i 1.18 priloženi su primjeri rezultata navedene metode.



Slika 1.16: Elipsa (crveno) koja najbolje opisuje B-splajn reprezentaciju konture (bijelo)



Slika 1.17: Lijevo: vanjska kontura, desno: B-splajn reprezentacija konture, nakon optimizacije praga



Slika 1.18: Duljine velikih osi elipsa za 30 različitih vrijednosti praga

Pomoćni algoritmi za lociranje centrosoma

Nadalje, na velikoj osi dobivene elipse želimo locirati centrosome. Obzirom da su centrosomi stanična tjelešca iz kojih izlaze mikrotubule (na slici sivih tonova su prikazane većim intenzitetom piksela od pozadinskog intenziteta), razumno je pretpostaviti da će okolina centrosoma imati veću prosječnu vrijednost intenziteta od ostalih područja uzduž velike osi. Okolinu centrosoma identificiramo sa podmatricom P dimenzije $K \times K$, pri čemu je u centru matrice P centralni piksel centrosoma. Pod "ostala područja na velikoj osi" također mislimo na matrice dimnezija $K \times K$, kojima se središnji piksel nalazi na velikoj osi. Dakle, ideja je pogledati matrice dimenzija $K \times K$ sa središtima na velikoj osi elipse te odrediti točku (središte) u kojoj je prosječan intenzitet pripadnog područja maksimalan. Da bismo to postigli, implementiramo sljedeće algoritme.

Prosječan intenzitet okoline

Funkcija *kernel_mean_intensity* najprije određuje veličinu filtera u ovisnosti o veličini slike *img*. Zatim uzima podmatricu dane slike s centrom u *kernel_center*, koja odgovara filteru te računa prosječnu vrijednost svih piksela dobivenog filtera, koristeći metodu *mean* iz biblioteke *NumPy*. Algoritam implementiran u Pythonu prikazan je na slici 1.19.

```python
def kernel_mean_intensity(img, kernel_center, ksize=5):
    """finds the mean pixel intensity in the obtained kernel."""
    y, x = kernel_center

    square_shape = (img.shape[0] + img.shape[1]) // 2
    ksize = int(0.05 * square_shape // 2)

    kernel_img = img[x-ksize:x+ksize, y-ksize:y+ksize]

    return np.mean(kernel_img)
```

Slika 1.19: Implementacija funkcije *kernel_mean_intensity*

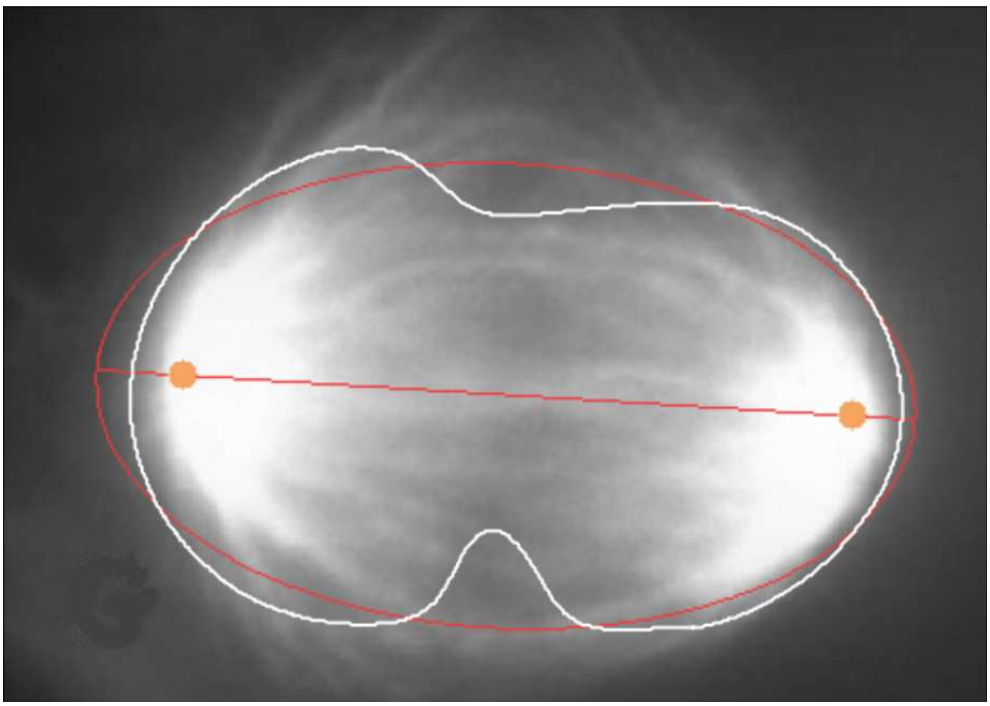
Računanje koordinata centrosoma

```python
def find_centrosome_position(img, start_point, a, rotation_angle, option):
    """finds centrosome coordinates along the horizontal axis of the fitted ellipse;
       option = 1 -> left centrosome, option = 2 -> right centrosome"""
    centrosome = start_point
    x, y = start_point
    max_intensity = -1

    for t in np.arange(0, 0.4, 0.01):
        if rotation_angle <= 90:
            if option == 1:
                kernel_center = (int(x + t * a * math.cos(math.radians(rotation_angle))),
                                 int(y - t * a * math.sin(math.radians(rotation_angle))))
            else:
                kernel_center = (int(x + t * a * math.cos(math.radians(rotation_angle + 180))),
                                 int(y + t * a * math.sin(math.radians(180 - rotation_angle))))
        else:
            if option == 1:
                kernel_center = (int(x + t * a * math.cos(math.radians(rotation_angle + 180))),
                                 int(y + t * a * math.sin(math.radians(180 - rotation_angle))))
            else:
                kernel_center = (int(x + t * a * math.cos(math.radians(rotation_angle))),
                                 int(y - t * a * math.sin(math.radians(rotation_angle))))

        kernel_intensity = kernel_mean_intensity(img, kernel_center)
        kernel_intensity /= 255 # normalization step
        kernel_intensity = kernel_intensity * math.exp(-t * 2.5) # penalize intesities which
                                                                 # are more distant from pole
        if kernel_intensity > max_intensity:
            max_intensity = kernel_intensity
            centrosome = kernel_center

    return centrosome
```

Slika 1.20: Implementacija funkcije *find_centrosome_position*

Funkcija *find_centrosome_position* računa koordinate jednog centrosoma. Parametar *start_point* odgovara koordinatama jednog tjemena elipse na kraju/početku velike osi, a to je upravo početna središnja točka spomenutog filtera. Jednostavno ju određujemo pomoću polarnih koordinata. Iterativnim postupkom krećemo se po velikoj poluosi te u svakoj iteraciji određujemo novu točku središta filtera, također koristeći polarne koordinate. Međutim, nije potrebno provoditi postupak do centra elipse, odnosno uzduž cijele poluosi jer su centrosomi obično bliže tjemenu, nego centru elipse. Za svaku izračunatu središnju točku filtera računamo prosječnu vrijednost intenziteta piksela pripadne okoline pomoću *kernel_mean_intensity* funkcije. Dobivene vrijednosti skaliramo na interval $[0, 1]$. Također, penaliziramo udaljenost središta filtera od tjemena elipse, tj. pola vretena. Odnosno, ako dvije okoline imaju sličnu prosječnu vrijednost intenziteta piksela, želimo odabrati onu koja je bliža tjemenu, iz razloga što su centrosomi u pravilu bliže polu nego središtu vretena. U tu svrhu dobivenu vrijednost još množimo s e^{-x}, pri čemu je $x \in [0, 1]$ i ima ulogu udaljenosti od tjemena. Konačno, funkcija vraća odgovarajuću točku koja sugerira položaj centrosoma. Implementacija algoritma u Pythonu prikazana je na slici 1.20.

Slijedi primjer lociranih centrosoma na slici 1.21.



Slika 1.21: Locirani centrosomi (narančasto)

Poglavlje 2

Računanje kuta otvora mitotičkog vretena na polovima

Nakon lociranja centrosoma na zadanim slikama, idući zadatak je izračunati kut otvora vretena na polovima. U ovom poglavlju navodimo dvije moguće metode za određivanje kuta te konačne rezultate za obje metode - lokacije centrosoma te vrijednosti kutova.

2.1 Metoda I - presjek aproksimirane konture i ortogonalne elipse

U prvoj metodi polove vretena poistovjećujemo sa tjemenima elipse na krajevima velike osi. Osim elipse E_1 koja najbolje opisuje aproksimiranu konturu, tj. njezinu B-splajn reprezentaciju, promatramo i elipsu E_2 koja je ortogonalna na E_1. Odnosno, kut između njihovih velikih osi iznosi $90°$. Također, centar elipse E_2 jednak je centru elipse E_1, duljine velikih osi su im jednake, a mala os elipse E_2 određena je tako da elipsa E_2 prolazi kroz centrosome. Odnosno, duljina male osi elipse E_2 jednaka je udaljenosti između centrosoma. Primjer se može vidjeti na slici 2.1.

Konstruiranjem ortogonalne elipse E_2 dobivamo četiri točke presjeka između elipsi E_1 i E_2 te četiri točke presjeka između aproksimirane konture i elipse E_2. Neka su točke P_1 i P_2 dvije točke presjeka E_2 i aproksimirane konture, u blizini jednog centrosoma. S P_3 i P_4 označimo točke presjeka aproksimirane konture i E_2 u blizini drugog centrosoma. Također, označimo s T_1 i T_2 tjemena elipse E_1. Traženi kut otvora vretena na polu T_1 definiramo kao kut $|\measuredangle P_1T_1P_2|$. Analogno definiramo kut otvora vretena na polu T_2, uz pripadne točke P_3 i P_4.

Obzirom da je aproksimirana kontura C diskretan skup točaka, nije nužno da elipsa E_2 prolazi (nekom) točkom konture C. Doduše, na slici se vide konkretni presjeci, ali samo

Slika 2.1: Elipse E_1 i E_2

zato što funkcija koja iscrtava konturu C spaja linijom njezine susjedne točke. Dakle, da bismo odredili je li neka točka $P \in C$ "dovoljno blizu" elipse E_2, dovoljno je provjeriti relaksirani uvjet definicije 1.5.1. Prilagodit ćemo uvjet tako što umjesto stroge jednakosti dozvoljavamo blago odstupanje za $\epsilon > 0$:

$$||PF_1| + |PF_2| - 2a| < \epsilon \,, \tag{2.1}$$

pri čemu su F_1 i F_2 fokusi, a $2a$ duljina velike osi elipse E_2. Tim uvjetom pronalazimo točke na konturi C koje su dovoljno blizu elipse E_2. Obzirom da je s lijeve strane nejednakosti apsolutna vrijednost, te točke mogu biti i unutar i van elipse.

Jednostavan algoritam koji to ostvaruje prikazan je na slici 2.2.
Primijetimo parametar *foci* funkcije *is_point_on_ellipse*. Radi se uređenom paru fokusa F_1 i F_2 ortogonalne elipse E_2. Koordinate tih točaka računamo isto kao što smo računali koordinate tjemena elipse E_1 ili središnje točke filtera kojeg smo pomicali uzduž velike osi elipse E_1, dakle, pomoću polarnih koordinata. Jedino novo što nam još treba je formula za računanje žarišne udaljenosti c:

$$c = \sqrt{a^2 - b^2} \,,$$

```python
def is_point_on_ellipse(point, foci, a_ortho):
    epsilon = 4  # tolerance
    focus1, focus2 = foci

    l1, l2 = math.dist(focus1, point), math.dist(focus2, point)

    return abs(l1 + l2 - 2 * a_ortho) < epsilon
```

Slika 2.2: Implementacija funkcije *is_point_on_ellipse*

pri čemu su a i b velika poluos i mala poluos elipse E_2.

Pozivom funkcije *is_point_on_ellipse* za svaku točku aproksimirane konture C, dobivamo nekoliko točaka koje zadovoljavaju relaksirani uvjet 2.1, odnosno nalaze se dovoljno blizu elipse E_2. Stoga, za svaku od četiri točke "presjeka" imamo nekoliko kandidata, a potreban nam je samo jedan. Dakle, nakon što pronađemo sve točke iz C koje zadovoljavaju uvjet, prirodno ih možemo rasporediti u četiri grupe tako da su točke unutar iste grupe međusobno bliže, nego točke iz različitih grupa. Zatim, biramo reprezentanta svake grupe kojeg ćemo proglasiti traženim presjekom. To postižemo sljedećim algoritmom.

k-means algoritam

Radi se o jednostavnom algoritmu klasteriranja (grupiranja, en. *clustering*). Naime, klasteriranje je jedna od najčešćih tehnika korištenih za dobivanje intuicije o strukturi podataka. Ideja je pronaći homogene podskupine (grupe, klastere) podataka takve da podaci u svakom klasteru budu što sličniji, obzirom na korištenu mjeru sličnosti. Jasno, odabir mjere sličnosti ovisi o samom zadatku.

k-means je iterativni algoritam ([18]) kojim pokušavamo podijeliti skup podataka u disjunktne klastere, gdje svaki podatak pripada samo jednom klasteru. Kao što je već naglašeno u prošlom odlomku, cilj je da podaci unutar istog klastera budu što sličniji, a podaci između različitih klastera što različitiji. Algoritam dodjeljuje točku klasteru tako da je zbroj kvadrata udaljenosti između točaka i središta klastera - centroida (aritmetička sredina svih točaka koje pripadaju tom klasteru) minimalan. Što je manje odstupanja unutar klastera, to su točke homogenije (sličnije) unutar istog klastera. Slijedi formalniji opis algoritma.

Definicija 2.1.1. *Neka je $A \neq \emptyset$ proizvoljan skup. Kažemo da je familija skupova $\mathcal{F}$ particija skupa A ako vrijedi:*

- $\forall x \in \mathcal{F},\ x \subseteq A$

- $\forall x \in \mathcal{F},\ x \neq \emptyset$
- $\forall x, y \in \mathcal{F},\ x \neq y$, *vrijedi* $x \cap y\emptyset$
- $\bigcup_{x \in \mathcal{F}} x = A$

Neka je $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ zadani skup podataka te $\mathbf{x}_i \in \mathbb{R}^d,\ \forall i \in \{1, ..., n\}$. Želimo particionirati skup X na $k \leq n$ podskupova, tj. odrediti particiju $S = \{S_1, ..., S_k\}$ koja minimizira sumu:

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|^2 = \sum_{i=1}^{k} |S_i|\ \mathrm{Var}\,(S_i)$$

pri čemu je μ_i srednja vrijednost točaka $i-$tog klastera S_i, a $\mathrm{Var}\,(S_i)$ varijanca $i-$tog klastera. Algoritam provodimo iterativno na način:

1. Inicializiraj centroide $\mu_1, ..., \mu_k\ \in \mathbb{R}^d$ na slučajan način.
2. Ponavljaj dok nije ispunjen uvjet konvergencije:
   a) $\forall i \in$ {1,...,n} odredi klaster $\mathrm{c}^{(\mathrm{i})}$ kojem pripada točka $\mathbf{x}_i$:

$$c^{(\mathrm{i})} := \arg\min_j \|\mathbf{x}_i - \mu_j\|^2$$

   b) $\forall$j $\in \{1, ..., k\}$:

$$\mu_j := \frac{\sum_{i=1}^{n} 1\{c^{(\mathrm{i})} = j\} \cdot \mathbf{x}_i}{\sum_{i=1}^{n} 1\{c^{(\mathrm{i})} = j\}} = \frac{1}{|S_j|} \cdot \sum_{\mathbf{x}_i \in S_j} \mathbf{x}_i$$

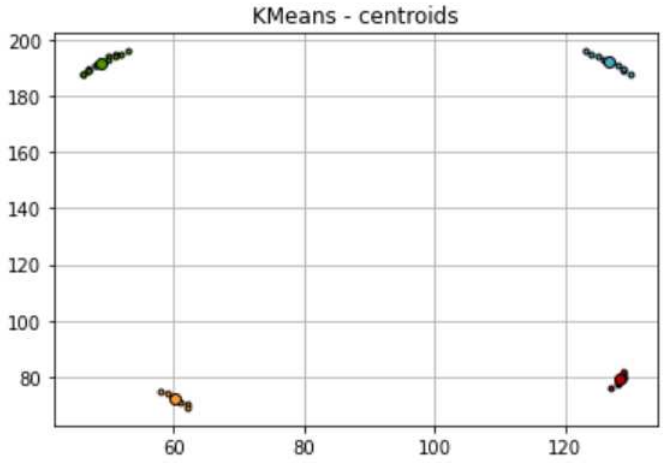
U drugom koraku algoritma, petlja se izvršava dok se ne ispuni tzv. uvjet konvergencije. Odnosno, algoritam staje u trenutku kad više nema promjena pri dodjeljivanju klastera točkama.

Na slici 2.3 prikazana je programska realizacija algoritma koji pronalazi centroide klastera. Funkcija *find_clusters* kao prvi parametar prima listu točaka koje su dovoljno blizu elipsi E_2, a drugi parametar predstavlja broj klastera. Zatim koristimo klasu KMeans paketa *sklearn* s već gotovom implementacijom algoritma klasteriranja. Funkcija *find_clusters* vraća četiri centroida pronađenih klastera. Obzirom da centroidi nisu nužno točke iz skupa podataka, ideja je iskoristiti centroide za pronalazak četiri točke svakog klastera koje su najbliže pripadnom centroidu. To je ilustrirano na slici 2.4 koja prikazuje graf s lokacijama centroida, za sliku 2.1. Također, navedene su i optimalne četiri točke presjeka.

Više o Kmeans algoritmu može se pronaći u [9].

```python
def find_clusters(intersections, n_clusters=4):
    """groups intersection points of the contour and the orthogonal ellipse in order to approximate 4
    intersection points"""
    k_means = KMeans(init="k-means++", n_clusters=n_clusters)
    k_means.fit(intersections)
    k_means_labels = k_means.labels_
    centroids = k_means.cluster_centers_
    k_means_labels_unique = np.unique(k_means_labels)

    intersections = np.array(intersections)
    colors = ["#4EACC5", "#FF9C34", "#4E9A06", "#B30000"]
    plt.figure()
    for k, col in zip(range(n_clusters), colors):
        labels = k_means_labels == k
        cluster_center = centroids[k]
        plt.plot(intersections[labels, 0], intersections[labels, 1], "o",
                 markerfacecolor=col, markeredgecolor="k", markersize=3)
        plt.plot(cluster_center[0], cluster_center[1], "o", markerfacecolor=col,
                 markeredgecolor="k", markersize=6)
    plt.title("KMeans - centroids")
    plt.grid(True)
    plt.show()

    return centroids
```

Slika 2.3: Implementacija funkcije *find_clusters*




```
Centroids of intersections: [(127, 192), (60, 72), (49, 192), (128, 79)]
Optimal intersections: [(127, 192), (60, 72), (49, 192), (128, 78)]
```

Slika 2.4: Centroidi četiri klastera i optimalne točke presjeka

Računanje kuta i završni rezultati

Konačno, odabirom četiri reprezentanta svakog klastera, pronašli smo točke potrebne za računanje kuta otvora vretena na polovima. Jedan od načina za računanje traženog kuta je

kosinusov poučak:

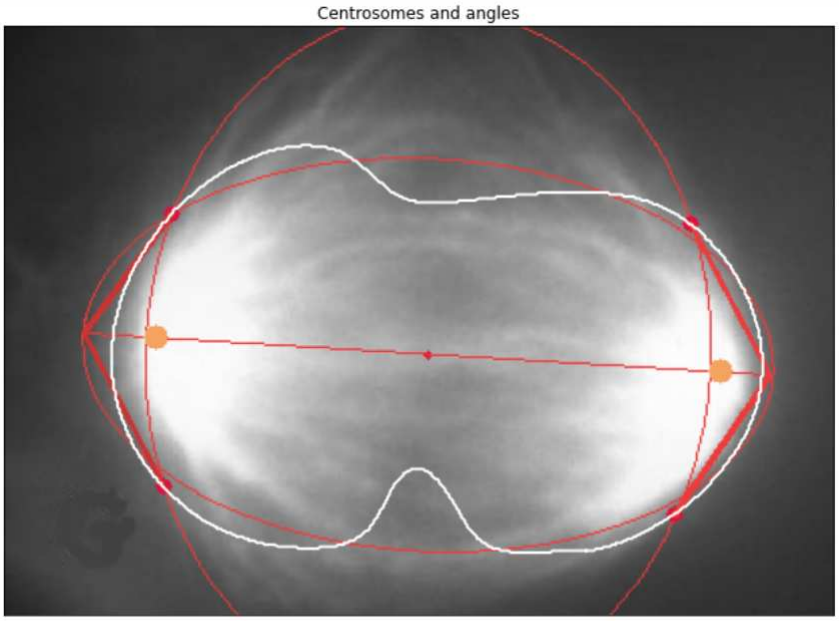
$$c^2 = a^2 + b^2 - 2ab\cos\gamma\ ,$$

pri čemu su a, b i c stranice trokuta, a γ kut nasuprot stranice c. Slijedi jednostavna implementacija prikazana na slici 2.5.

```
def find_angle(endpoint, points):
    """finds the angle between 2 lines, given coordinates of 3 points"""
    point1, point2 = points[0], points[1]
    u, v, w = math.dist(endpoint, point1), math.dist(endpoint, point2), math.dist(point1, point2)

    angle = (u**2 + v**2 - w**2)/(2*u*v)
    return math.degrees(math.acos(angle))
```

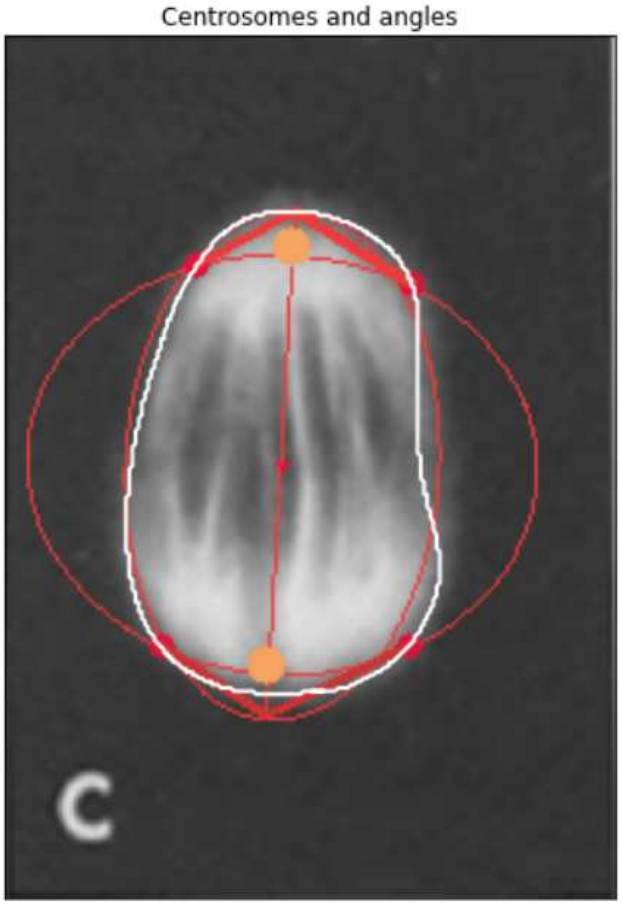
Slika 2.5: Implementacija funkcije *find_angle*

Prikazujemo i konačne rezultate za nekoliko primjeraka mitotičkih vretena. Na slikama se nalaze locirani centrosomi i vizualizacija kutova, a ispod njih veličine optimalnih kutova otvora vretena.

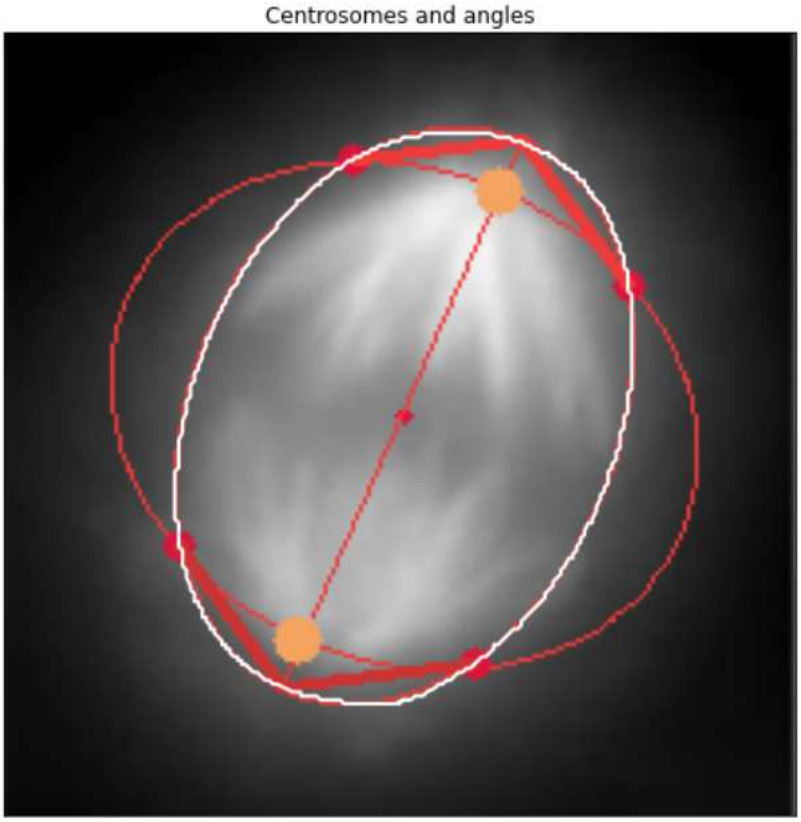



```
Left angle: 116.3°
Right angle: 117.8°
```

Slika 2.6: Metoda I - primjer završnog rezultata (1)

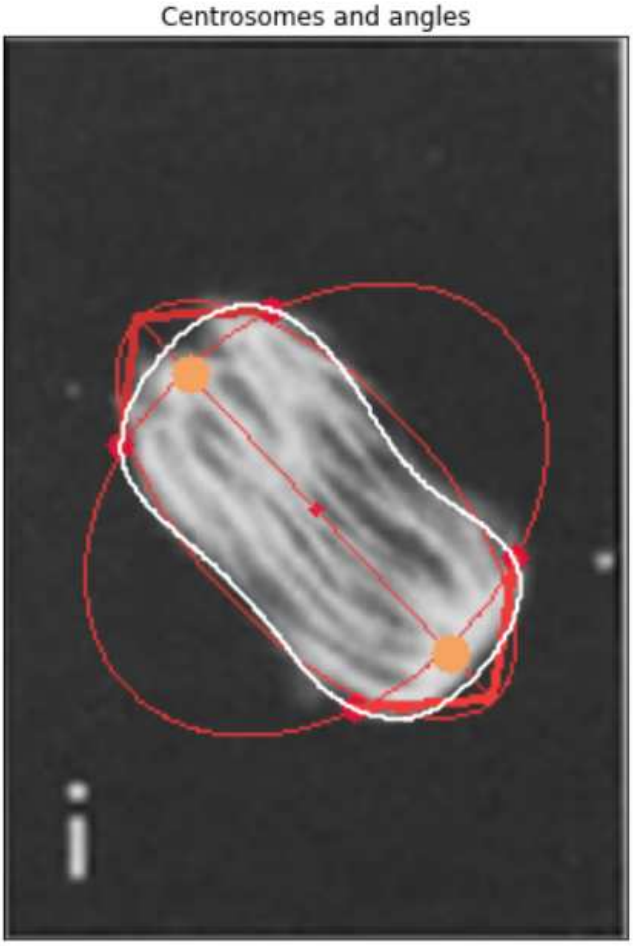


Slika 2.7: Metoda I - primjer završnog rezultata (2)

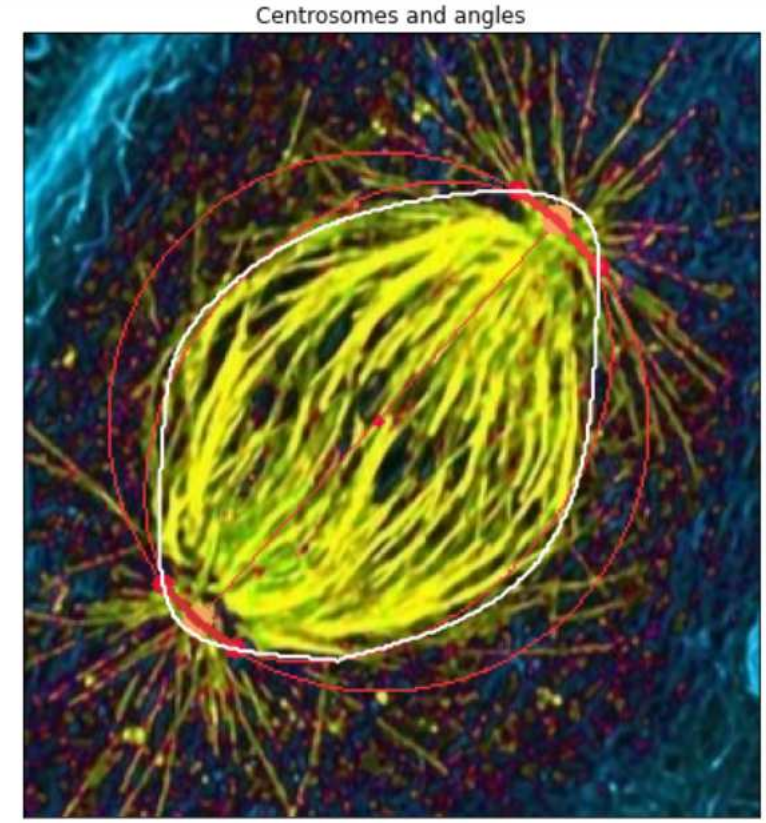


Left angle: 118.86°
Right angle: 119.49°

Slika 2.8: Metoda I - primjer završnog rezultata (3)



Slika 2.9: Metoda I - primjer završnog rezultata (4)



Slika 2.10: Metoda I - primjer završnog rezultata (5)

Prethodna slika 2.10 pokazuje situaciju u kojoj prva metoda nije dala očekivane rezultate. Naime, centrosomi su locirani na samim tjemenima elipse E_1 pa su pripadne četiri točke presjeka vrlo blizu polovima. Zbog toga dobivamo puno veće kutove otvora, nego što bismo to očekivali.

2.2 Metoda II - odmak od pola po parametriziranoj krivulji

Druga metoda računanja kuta otvora vretena temelji se na jednostavnom principu. Slijede koraci metode:

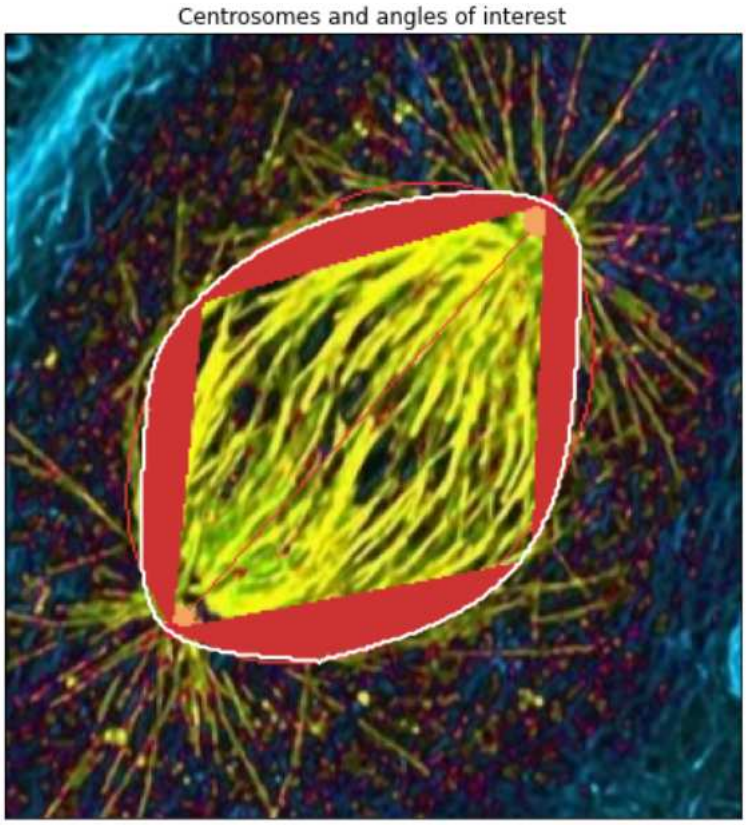
1. Odredi polove vretena P_1 i P_2, tj. tjemena elipse E_1
2. Odredi dvije točke $P_1^{'}$ i $P_2^{'}$ na parametriziranoj krivulji B (B-splajn reprezentaciji konture) koje su najbliže pripadnim polovima
3. Za pronađenu točku $P_1^{'} = B(u_1)$ i parametar $u_1 \in [0, 1]$:
   a) Na krivulji B odredi nove dvije točke $T_1 = B(u_1 - u)$ i $T_2 = B(u_1 + u)$ za sve $u \in [a..b] \subset [0, 1]$
   b) Izračunaj veličinu kuta $|\measuredangle T_1 P_1^{'} T_2|$
   c) Izračunaj μ, srednju vrijednost dobivenih kutova za sve parametre u
   d) Odaberi točke T_1 i T_2 koje zatvaraju kut θ, veličine najbliže srednjem kutu μ
   e) Kut θ je optimalni kut
4. Ponovi korak 3. za točku P_2'

U koraku 3.a) zapravo se želimo po krivulji pomaknuti ”lijevo i desno” za jednake udaljenosti, odnosno težimo da dobiveni trokut $\triangle T_1 P_1^{'} T_2$ bude jednakokračan (idealan slučaj). Početni odmak od točke $P_1^{'}$, u oznaci a, biramo tako da nije preblizu nuli. U protivnom, dobili bismo nekoliko točaka koje su relativno blizu točki $P_1^{'}$, a samim tim i kutove koji iznose gotovo 180°. U praksi, a biramo tako da preskočimo prvih 5% točaka na obje strane točke $P_1^{'}$. Također, vrijednost b biramo tako da točke $T_1^{'}$ i $T_2^{'}$ ne prelaze na drugu polovicu vretena, s obzirom na malu os elipse E_1.

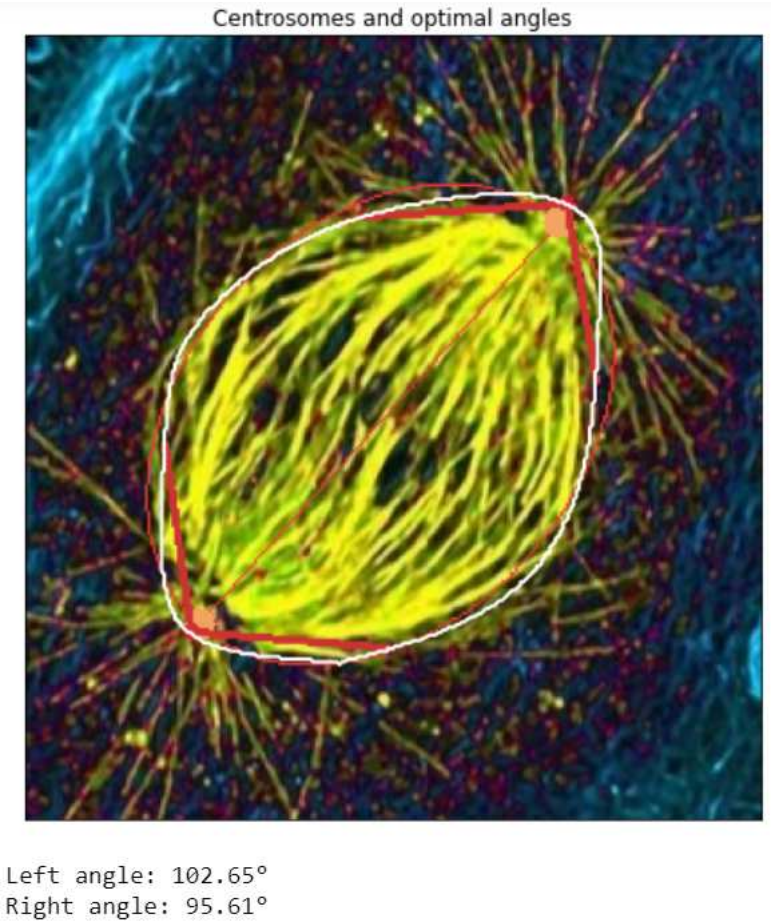
Kut otvora računamo jednako kao i kod prve metode.

Završni rezultati

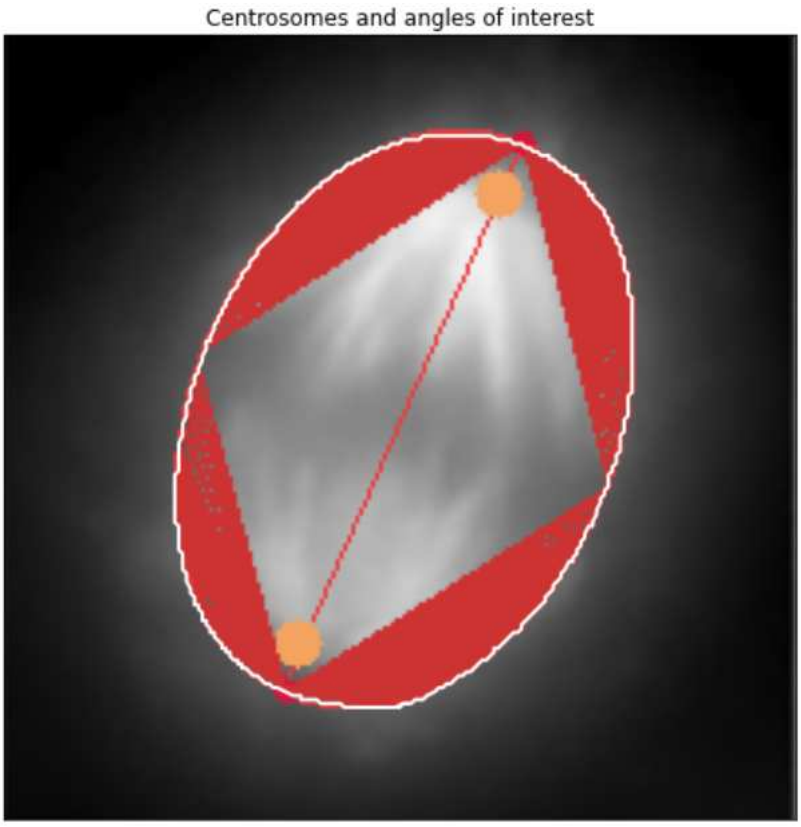
Slijedi nekoliko primjera završnih rezultata druge metode, s prikazanim centrosomima i optimalnim kutovima. Na slikama se može vidjeti da metoda rješava problematični slučaj prikazan na slici 2.10.



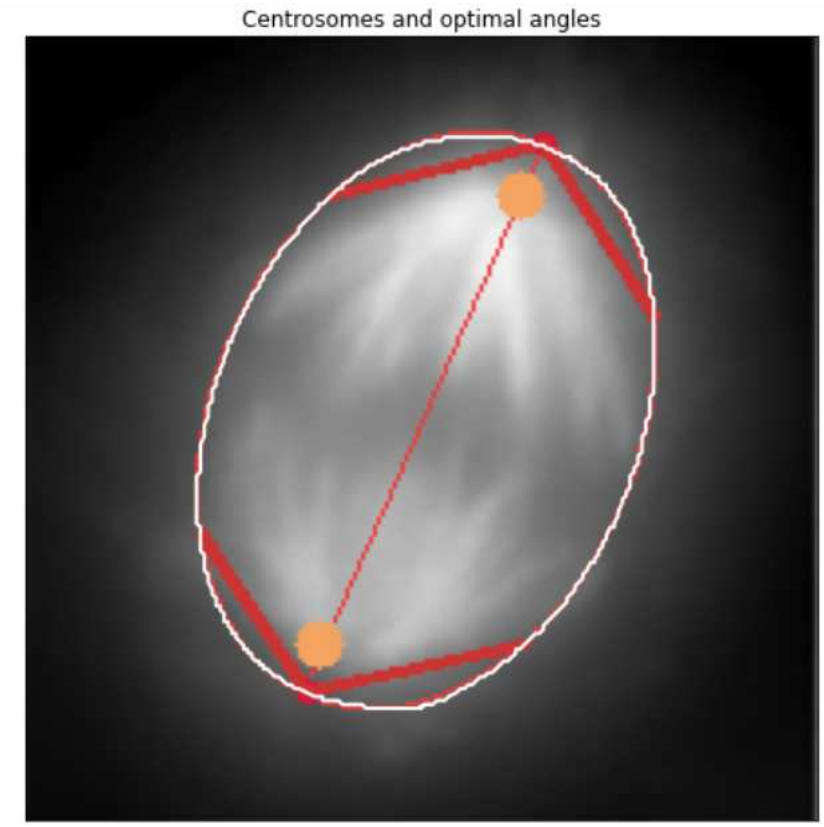
Slika 2.11: Metoda II - centrosomi i potencijalni kutovi (1)



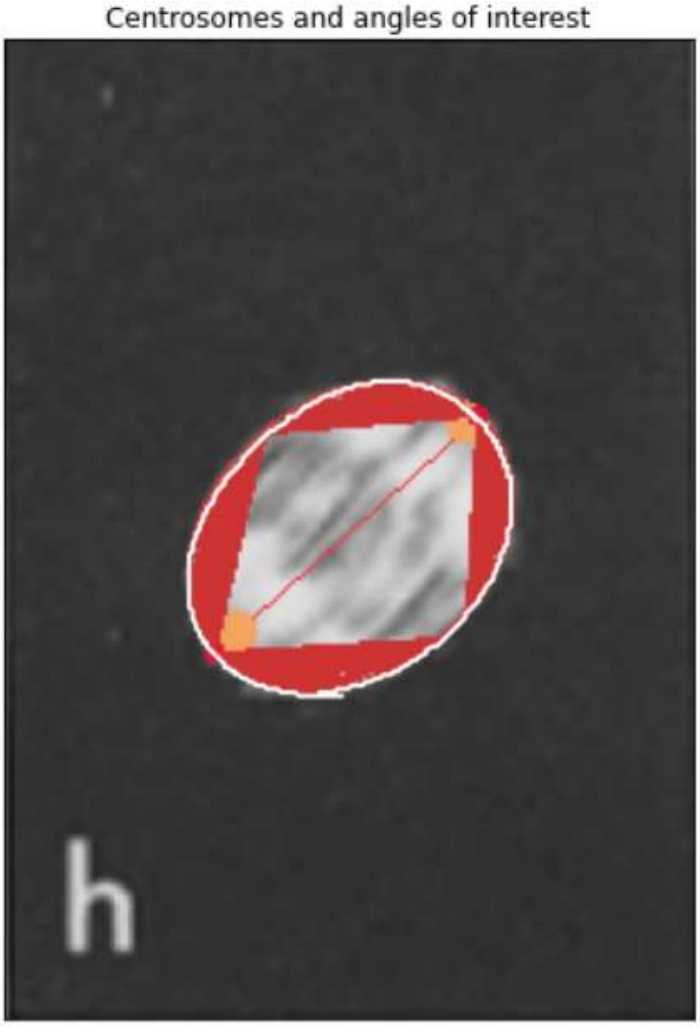
Slika 2.12: Metoda II - primjer završnog rezultata (1)



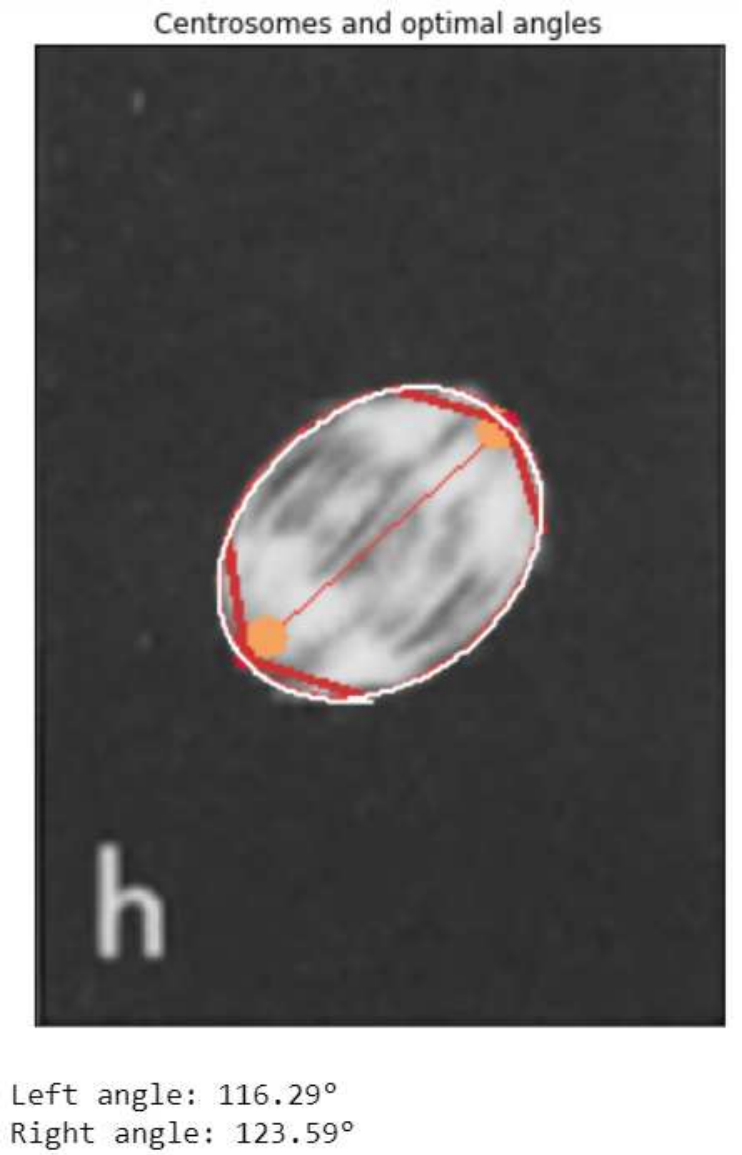
Slika 2.13: Metoda II - centrosomi i potencijalni kutovi (2)



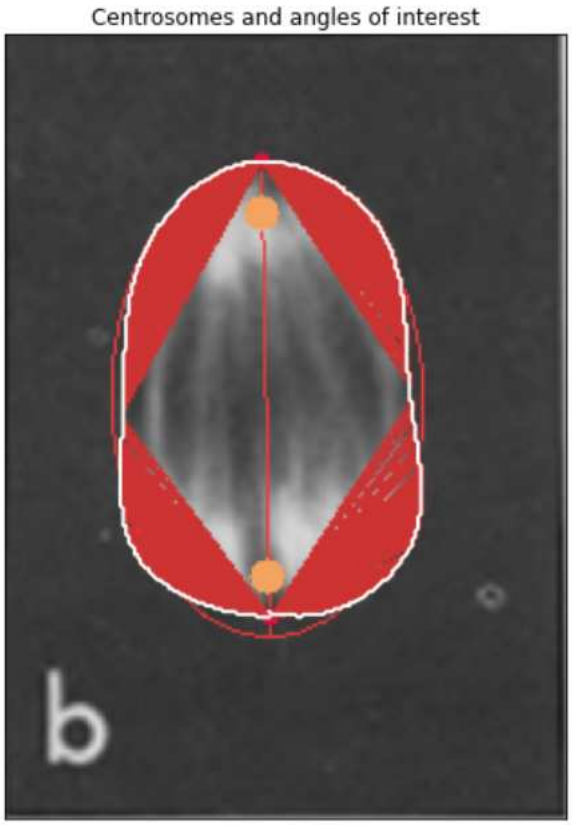
Slika 2.14: Metoda II - primjer završnog rezultata (2)



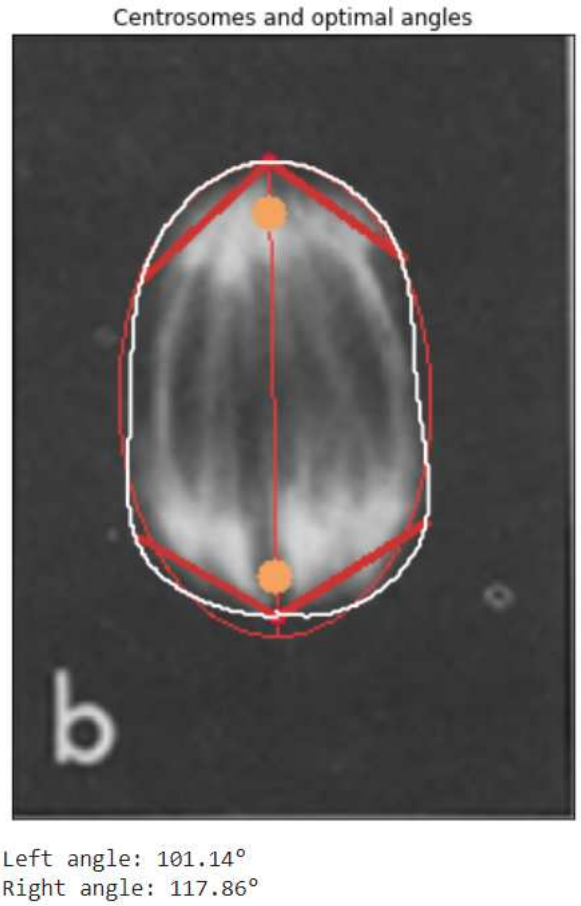
Slika 2.15: Metoda II - centrosomi i potencijalni kutovi (3)



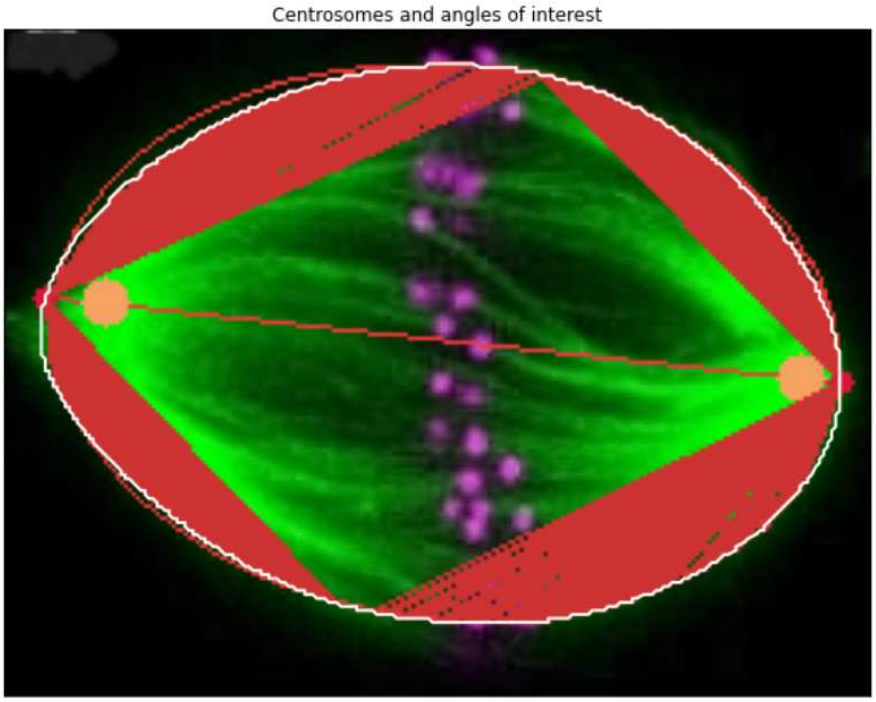
Slika 2.16: Metoda II - primjer završnog rezultata (3)



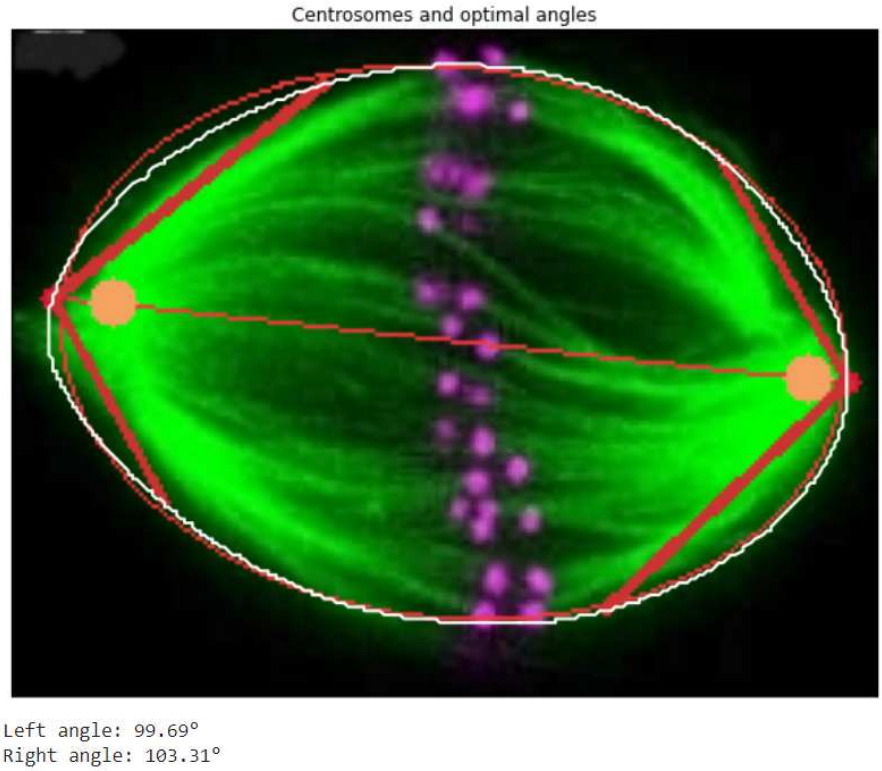
Slika 2.17: Metoda II - centrosomi i potencijalni kutovi (4)



Slika 2.18: Metoda II - primjer završnog rezultata (4)



Slika 2.19: Metoda II - centrosomi i potencijalni kutovi (5)



Slika 2.20: Metoda II - primjer završnog rezultata (5)

Poglavlje 3

Zaključak

Glavna motivacija ovog rada je automatska obrada slike s ciljem praćenja dinamike određenog procesa te objektivizacije rezultata. Time omogućavamo obradu familije slika ovisnih o vremenu ili lokaciji (npr. CT) te praćenje brzine promjene procesa.

Konkretno, prikazana je metoda automatske obrade digitalne slike kojom na slici mitotičkog vretena, uz određene optimizacijske postupke, određujemo položaje centrosoma te računamo kutove otvora vretena na polovima. Metodu ugrubo možemo podijeliti na sljedeće korake:

1. Određivanje konture vretena
2. Određivanje lokacije centrosoma i polova
3. Računanje kuta otvora na polovima

Za prvi korak metode isprobane su tri tehnike odsjecanja koje daju segmentiranu sliku, a Otsuova metoda pronalaženja optimalnog globalnog praga pokazala se najuspješnijom na našem skupu podataka. Postupak određivanja lokacije centrosoma generirao je relativno uspješne rezultate, korištenjem kvadratne optimizacije za pronalazak elipse koja najbolje opisuje konturu vretena povezanog područja. Međutim, ima prostora za napredak kod metode određivanja kuta otvora vretena. Naveli smo dvije metode - prva koristi presjek dviju elipsi za određivanje ”krajnjih točaka kuta”, ali vidi se na konkretnom primjeru 2.10 gdje metoda ne daje željene rezultate. Druga metoda je robusnija i rješava taj problem. Međutim, vrijednost kuta ovisi o dobivenoj parametriziranoj krivulji i odabiru parametra glatkoće. Dakle, idući korak za eventualno poboljšanje metode bio bi isprobavanje alternativnih metoda detekcije rubova na slici. Tako potencijalno možemo dobiti interpretabilnije konture vretena te preciznije rezultate pri računanju kutova.

Na kraju, ovim putem zahvaljujem i prof. dr. sc. Nenadu Pavinu te prof. dr. sc Ivi Tolić na poticaju i na motivaciji za bavljenje navedenim problemom.

Bibliografija

[1] *B-spline representation of an N-D curve*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splprep.html, posjećena 2. studenoga 2022.

[2] *Contour finding*, https://scikit-image.org/docs/dev/auto_examples/edges/plot_contours.html, posjećena 1. listopada 2022.

[3] *Convolution*, https://en.wikipedia.org/wiki/Convolution, posjećena 23. rujna 2022.

[4] *Creating Bounding rotated boxes and ellipses for contours*, https://docs.opencv.org/3.4/de/d62/tutorial_bounding_rotated_ellipses.html, posjećena 18. listopada 2022.

[5] *Gaussian Smoothing*, https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm, posjećena 23. rujna 2022.

[6] *Image Filtering*, https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html, posjećena 25. rujna 2022.

[7] *Image Processing*, https://docs.opencv.org/4.x/d7/dbd/group__imgproc.html, posjećena 23. rujna 2022.

[8] *Image Thresholding*, https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html, posjećena 28. rujna 2022.

[9] *K-Means clustering*, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, posjećena 15. listopada 2022.

[10] *Marching squares*, https://en.wikipedia.org/wiki/Marching_squares, posjećena 5. listopada 2022.

[11] *Mitotic spindle*, https://www.nature.com/articles/nrm.2016.162, posjećena 20. rujna 2022.

[12] *Shoelace formula*, https://en.wikipedia.org/wiki/Shoelace_formula, posjećena 15. listopada 2022.

[13] *Spindle apparatus*, https://en.wikipedia.org/wiki/Spindle_apparatus, posjećena 20. rujna 2022.

[14] Damir Bakić, *Linearna algebra*, 2008.

[15] Andrew W. Fitzgibbon i Robert B. Fisher, *A Buyer's Guide to Conic Fitting*, (1970), https://homepages.inf.ed.ac.uk/rbf/MY_DAI_OLD_FTP/rp810.pdf.

[16] Ilja Gogić, Pavle Pandžić i Josip Tambača, *Integrali funkcija više varijabli*, 2020.

[17] ———, *Diferencijalni račun funkcija više varijabli*, 2021.

[18] Chris Piech, *K-Means*, (2012), https://stanford.edu/~cpiech/cs221/handouts/kmeans.html.

[19] Richard F. Riesenfeld, *Applications Of B-Spline Approximation To Geometric Problems Of Computer-Aided Design*, (1973), https://www.researchgate.net/publication/47761241_Applications_Of_B-Spline_Approximation_To_Geometric_Problems_Of_Computer-Aided_Design.

[20] Jamileh Yousefi, *Image Binarization using Otsu Thresholding Algorithm*, (2011), https://www.researchgate.net/publication/277076039_Image_Binarization_using_Otsu_Thresholding_Algorithm.

Sažetak

U ovom radu predstavljen je postupak automatske obrade slike s ciljem praćenja dinamike određenog procesa te objektivizacije rezultata. Kao prototip problema koriste se slike stanične diobe za vrijeme mitoze. Također, uvodi se pojam centrosoma i predlaže metoda računalnog vida koja određuje njihov položaj na digitalnim slika. Navodimo sve korištene algoritme potrebne za implementaciju metode te ih opisujemo u formalnom smislu. Uz to, predlažu se i dvije metode za računanje kuta otvora vretena na polovima te prikazujemo implementaciju pripadnih algoritama. Konačno, za svaku metodu navodimo završne rezultate.

Summary

This thesis presents the procedure of automatic image processing with the aim of monitoring the dynamics of a certain process and objectifying the results. Images of cell division during mitosis are used as a prototype of the problem. Also, the term centrosome is introduced and a computer vision method that determines their position on digital images is proposed. The algorithms needed to implement the method are listed and described in a formal manner. In addition, two methods for finding the opening angle of the spindle at the poles are proposed, along with the implementation of the corresponding algorithms. At the end, the final results for each method are shown.

Životopis

Rođen sam 15. srpnja 1995. godine u Bjelovaru, a školovanje započinjem upisom u Osnovnu školu Trnovitica, u Velikoj Trnovitici. Tamo stječem prvi interes za matematiku te upisujem Prirodoslovno-matematičku gimnaziju u Bjelovaru, 2010. godine.
Obrazovanje nastavljam 2014. godine na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu, a 2020. stječem zvanje Sveučilišnog prvostupnika matematike te, na istom fakultetu, upisujem Diplomski studij Računarstva i matematike.
Za vrijeme studiranja obavljam dvije studentske prakse u firmama Sedam IT i Memgraph te radim kao student *developer* u Ericsson Nikola Testla d.d. i MMK Systems d.o.o. Također, u suradnji sa dvojicom kolega sa Fakulteta elektrotehnike i računarstva u Zagrebu, 2022. godine ostvarujem treće mjesto na LUMEN Data Science studentskom natjecanju. Trenutno radim kao pripravnik na Fakultetu elektrotehnike i računarstva u skloptu *TakeLab* grupe istraživača na području obrade prirodnog jezika.