Razvrstavanje objekata u slici metodom usporedbe konture objekata

Pole, Ante

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RAZVRSTAVANJE OBJEKATA U SLICI METODOM USPOREDBE KONTURE OBJEKATA

Diplomski rad

Ante Pole

Osijek, 2016.

Ovdje treba umetnuti: Obrazac D1 – Obrazac za ocjenu završnog rada.

Ovdje treba umetnuti: Izjavu o originalnosti završnog rada.

SADRŽAJ

1. UVOD

Cilj rada nam je pomoću zadanih alata i programskih sučelja realizirati aplikaciju koja će obavljati razvrstavanje objekata s 2D slike metodom usporedbe konture objekata. Vrlo jednostavno, kontura je granična linija koja odvaja površinu nekog područja ili površinu objekta od prostora oko nje.[1] Dakle, naglasak se daje na rubove naših objekata. Za promatrane objekte izabrali smo matice s kojima se jako često susrećemo u svakodnevnom životu. Matica je mehanička naprava koja služi priščvrćivanju ili podešavanju. Ona izgleda kao komad nekog tvrđeg materijala s provrtom i navojem. Postoji mnoštvo različitih materijala za izradu matica, kao i mnoštvo raznih oblika matrica. Osnovni oblik matica je šesterokut, oblik koji je našao vrlo široku primjenu u praksi te oblik koji koristimo u našoj realizaciji rada. Kako duljina matica nije uvijek ista i ovisna je o promjeru njenog provrta te o vrsti materijala od kojeg se matica izrađuje, ona će nam poslužiti kao jako bitan element kod razvrstavanja istih.

Koristit ćemo se jednim od pristupa segmentaciji slike, segmentacija temeljena na ocjeni granica korištenjem detekcije rubova, provodi se nalaženjem slikovnih elemenata koji leže na granicama područja. Takvi se slikovni elementi nazivaju rubni elementi i mogu se odrediti ispitivanjem njima susjednih slikovnih elemenata. Rubni element u slici se nalazi na granici između susjednih područja pa se može pronaći na temelju razlike između susjednih slikovnih elemenata. Većina detektora rubova rubne elemente pronalazi isključivo na temelju značajki inteziteta slikovnih elemenata, dok drugi rubni detektori koriste teksturu i pokret.

Kako nam je zadatak razvrstavanje objekata u slici, od velike važnosti nam je kvaliteta tih slika. Da bi došlo do što boljih i točnijih rezultata prvi uvjet nam je kvaliteta slike. Za idealnu sliku (bez prisutnosti šuma) se očekuje da će segmentacija na područja i rubna segmentacija dati identičan rezultat te da će područja biti ograničena zatvorenom konturom. Međutim, u slučaju realnih slika ni segmentacija na područja niti rubna segmentacija ne daju savršen rezultat upravo zbog prisutnosti šuma.

1.1. Zadatak i organizacija rada

Zadatak rada je projektirati, izraditi i verificirati sustav za razvrstavanje objekata u 2D slici koristeći metodu opisa konture objekata i usporedbe karakteristika kontura s referentnim

konturama u bazi istih. Kako bi prisutnost različitih smetnji i šumova smanjilo točnost rezultata, prvi uvjet bio je izraditi fotografije visoke kvalitete.

Slika 1.1 *Bijela podloga se pokazala kao nezadovoljavajuća za naše potrebe*

Kroz rad razmotrit će se razne metode obrade slike, mogućnosti primjene pojedinih metoda, njihove prednosti i nedostaci kao i razlozi korištenja navedenih metoda.

2. 2D OBJEKTI U SLICI I KONTURE OBJEKATA

2D računalna grafika ili 2D grafika je prikaz neke slike predstavljen u dvije dimenzije (dužina i širina). S jedne strane, digitalna slika je isto što i bilo koji drugi računalni podatak – dugačak niz jedinica i nula. Kada biste digitalnu sliku vidjeli prikazanu u takvome kodu, ne biste je mogli razlikovati od koda kakve tablice ili pisma. Slika se definira kao dvodimenzionalna funkcija, odnosno funkcija dvije varijable, gdje vrijednosti funkcije *f(x,y)* predstavljaju svjetlinu bilo koje točke i mogu poprimiti bilo koje realne vrijednosti između, na primjer, 0.0 (crne) i 1.0 (bijele).

Da bi sliku bilo moguće obraditi na računalu potrebno ju je imati u digitalnom formatu. Većina fotoaparata danas automatski snima slike u standardizirani digitalni format (JPEG, PNG, TIFF), a ukoliko to nije slučaj, sliku je potrebno digitalizirati.

Ono što razlikuje podatak slike je način njegove organizacije. Čak i dokument iz programa za obradu teksta možete otvoriti kao sliku. To nas dovodi do nove dimenzije zbog čega je slika prikaz nečega – podatak predstavlja ono u što ste usmjerili kameru u trenutku fotografiranja. Postavlja se pitanje kako je slika zapisana u podatku. Kod dokumenta napisanog u programu za obradu teksta svako slovo ima svoj kod, a njihova veličina i mjesto na stranici svoj, nešto poput kataloških brojeva koje softver može interpretirati. Kod slika radi se o slikovnim elementima (ili pikselima) umjesto o slovima. [2]

Temelj našeg promatranja su rubovi objekata (matica) s fotografija. Kako bih nam bilo što lakše realizirati sustav za razvrstavanje matica kvaliteta slike nam je od velike važnosti. Naša fotografija je pravilno napravljana tek kad se uvjerimo da je svaki objekt s fotografije ispravan te kada smo sigurno da su rubovi objekata jasno vidljivi. Ispitivanje svojstava rubova je također jedan od mnoštva parametara promatranih tokom procesa vizualnog ispitivanja. Rubna nepravilnost je jedan od prvih parametara koji trebaju biti ispitani. Ako promatrani objekt ima značajne nepravilnosti ruba u odnosu na referentni model ruba, objekt je potrebno ukloniti te su daljne analize potpuno nepotrebne. Ako su nepravilnosti relativno male i prihvatljive, tada objekt ide na daljne analize ali u tom slučaju ga se tretira kao objekt niže klase.

Dosada je razvijeno mnogo manje ili više uspješnih pristupa i metoda za ispitivanje rubova i površina objekata. U osnovi, sve metode provjera rubova su nam došle od metoda traženja rubova. Jedna od najčešće korištenih metoda je zasnovana na dobro poznatoj Hough-ovoj transformaciji. Hough-ova transformacija pronalazi prave linije, njihove kuteve i poziciju

objekta. Korištenjem spomenute transformacije moguće je pronaći točne rubne granične linije jedino ako su pravokutnog oblika. Svi drugi oblici s razno raznim krivuljama nisu pogodni.[6]

2.1. Segmentacija slike

Pri analizi objekata na slici neophodno je razgraničiti ″objekte od interesa″ od ostatka slike. Tehnika koja izdvaja objekte od interesa se naziva segmentacija. Metode segmentacije mogu se razdijeliti u tri grupe: metoda segmentacije zasnovana na određivanju praga, metoda segmentacije zasnovana na rubovima i metoda segmentacije zasnovana na regijama. Ne postoji univerzalna tehnika segmentacije koja će raditi na svim slikama niti je i jedna tehnika segmentacije savršena. [3]

$$R = \bigcup_{i=1}^{S} Ri\,, Ri \bigcap Rj = \emptyset, \;\; i \neq j. \tag{2.1}$$

Slika 2.1 *Potpuna segmentacije slike razdjeljene na konačan broj regija*

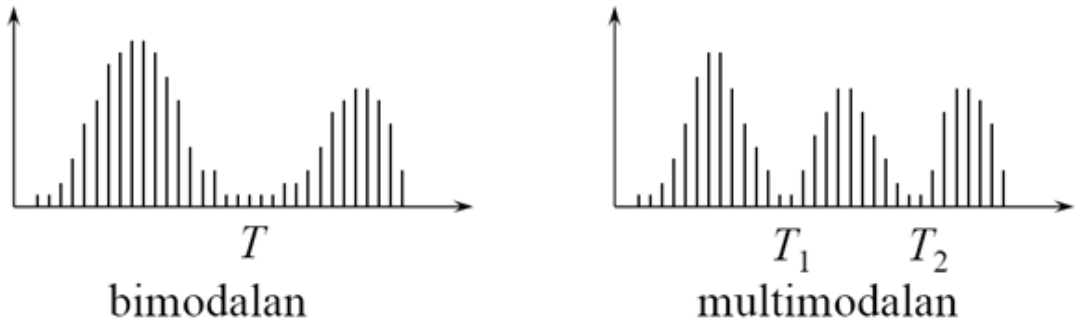
2.1.1. Metoda segmentacije zasnovana na određivanju praga (metoda binarizacije)

Metoda segmentacije koja se bavi određivanjem praga, izdvajanjem cijelog objekta od pozadine. Metoda segmentacije zasnovana na određivanju praga je najjednostavniji segmentacijski proces. Mnogi objekti ili regije slike imaju karakterističnu konstantnu refleksiju ili apsorpciju svjetla na njihovim površinama, to nam omogućuje da odredimo neki konstantni iznos tj. prag koji će izdvajati objekte od pozadine. Određivanje praga je računski nezahtjevna i jednostavna metoda, to je najstarija metoda koja još ima široku primjenu za jednostavnije zadatke. Vrlo se jednostavno izvodi u realnom vremenu uz pomoć adekvatne opreme. Osnovni algoritam: provjeriti sve piksele originalne slike. Algoritam transformira ulaznu sliku u binarnu (segmentiranu) sliku i to na način:

$$\begin{aligned} g(i,j) &= 1 \quad for\ f(i,j) \geq T; \\ &= 0 \quad for\ f(i,j) < T. \end{aligned} \tag{2.2}$$

gdje je T prag, g(i, j) = 1 su objekti a g(i, j) = 0 pozadina ili obrnuto. Ako se objekti međusobno ne dodiruju i ako se jasno razlikuju od pozadine onda je ovo prigodna metoda. Vrlo je bitno odrediti dobar prag, samo u rijetkim slučajevima se događa da je jedan prag dobar za cijelu sliku (globalni prag) pošto već i kod dosta jednostavnih slika dolazi do varijacija. Takve varijacije mogu biti uzrokovane nejednakim osvjetljenjem i nizom drugih faktora.

Metode određivanja praga se koriste da bih se odredio prag automatski. Ako unaprijed znamo neku značajku slike zadatak određivanja je pojednostavljen jer se određuje prag koji zadovoljava tu značajku. Metode se baziraju na analizi oblika histograma, najčešće se koristi histogram prvog reda. Histogram prvog reda predstavlja relativnu frenkvenciju svjetlina točaka u slici. Može biti bimodalan ili multimodalan te se može izračunati globalno ili lokalno.



Slika 2.2 *Oblici histograma prvog reda [3]*

Jedan od mogućih problema je to što je određivanje minimuma teško zbog izlomljenosti krivulja histograma. Problem se može riješiti tako da se izgladi histogram ili izvrši interpolacija glatkom funkcijom. [3]

2.1.2. Metoda segmentacije zasnovana na rubovima

Segmentacija bazirana na rubovima se zasniva na rubovima pronađenim uz pomoć raznih detektora, ti rubovi označavaju lokacije diskontinuiteta između nijansi, boja, tekstova ili nečega drugoga. Najčešće se problemi kod ove segmentacije javljaju zbog šumova ili drugih vrsta loših informacija o slici a to su da se registrira rub gdje ne postoji ili da se ne registrira gdje on postoji. Granice objekata se mogu izdvajati metodama praćenja granice, interpolacijom krivulje i Hough-ovom transformacijom.

Kod metode praćenja granice detekcija rubova je važna u analizi slika zato što rubovi određuju granice objekata i zato su korisni za segmentaciju, registraciju i identifikaciju objekata na slici. Rubovi su mjesta naglih promjena u vrijednosti točaka slike te je zbog toga moguće koristiti gradijent funkcije za detekciju ruba.

Gradijent funkcije dviju varijabli je vektor koji pokazuje smjer najbrže promjene funkcije

$$grad\ f(x,y) = \left[\frac{\partial f(x,y)}{\partial x}\ \ \frac{\partial f(x,y)}{\partial y}\right] = [fx\ \ fy] \tag{2.3}$$

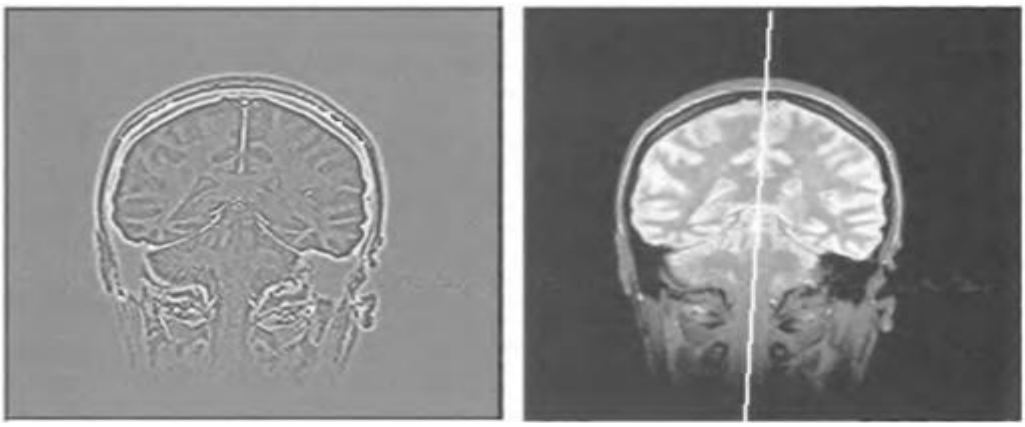
S obzirom na detekciju smjera ruba postoji podjela na gradijente (u dva ortogonalna smjera) i kompas (više smjerova) operatore.

Nekad je moguće točke rasporeda rubova (eng. *edge map*) povezati metodama interpolacije da bih se dobila zatvorena kontura koja definira regiju. Potrebno je razbiti konture u dijelove koji se interpoliraju. Za interpolaciju je moguće koristiti polinomske metode.

Hough-ova transformacije se koristi za detekciju linija na slici. Pravac se može opisati jednadžbom:

$$x\cos\theta + y\sin\theta = \rho \tag{2.4}$$

gdje je ρ udaljenost pravca od ishodišta a ϑ kut nagiba. Skup pravaca koji prolaze kroz jednu točku se preslikava u skup točaka koje leže na sinusoidi. [3]



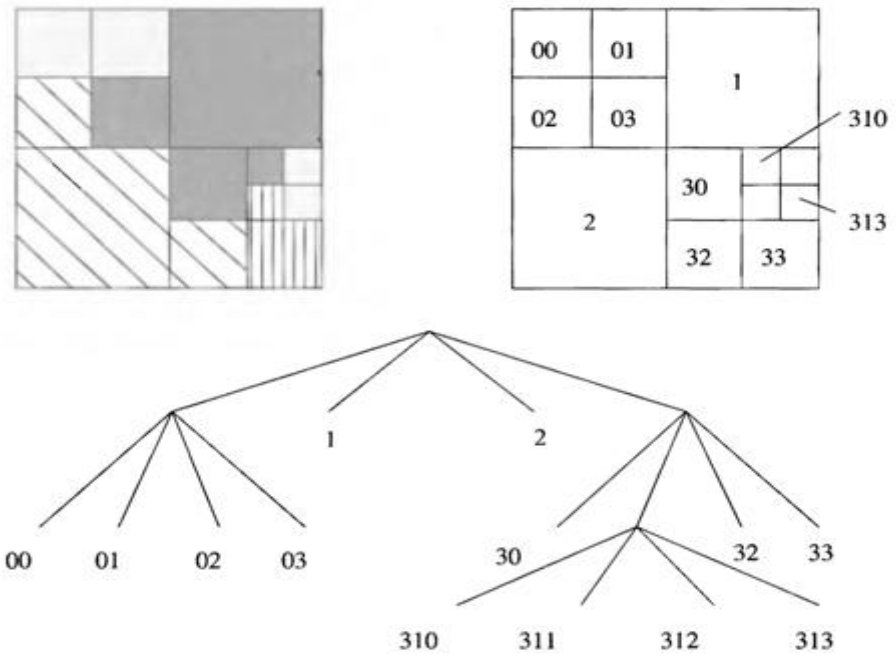
Slika 2.3 *Hough-ova transformacija za segmentaciju magnetske rezonance mozga [3]*

2.1.3. Metoda segmentacije zasnovana na regijama

Cilj ovih metoda je identificirati dijelove slike koje imaju slične značajke. Dvije karakteristične metode su: izrastanje područja te metoda dijeljenja i stapanja. Za razliku od prethodnih metoda u kojima su se tražili rubovi između regija ove metode direktno određuju regije. Ova metoda je pogodnija za slike koje imaju dosta šuma gdje je teško odrediti granice.

Problem segmentacije izrastanja područja sastoji se u određivanju uniformnih skupina točaka slike (regija). Algoritmni za izrastanje područja uspoređuju svojstva neklasificirane točke s dotad segmentiranom regijom da bi odlučili da li točka pripada regiji ili ne. Tehnike izrastanja područja mogu se podijeliti u tri grupe s obzirom na način uspoređivanja točaka: na osnovi sličnosti dvaju susjednih točaka, na osnovi sličnosti okolina dvaju susjednih točaka te na osnovi sličnosti točke i centroida regije.

Kod metode dijeljenja i stapanja koristi se prikaz slike pomoću kvartarnog stabla gdje se svaka grana dijeli u četiri grane. Inicijalna segmentirana regija je cijela slika. Ako je regija neuniformna onda se razbija u četiri podregije. Uniformnost regije se mjeri npr. pomoću razlike svjetline najtamnije i najsvjetlije točke u regiji. [3]



Slika 2.4 *Metoda dijeljenja i stapanja [3]*

2.2. Konturni opisnik i konturne karakteristike

Jedan od najkorisnijih svojstava konturnog opisnika jest da pod utjecajem različitih promjena ostaje identičan za zadani objekt. To svojstvo se naziva nepromjenjivost ili invarijacija (eng. *invariance*). Namjenski, invarijaci je definirana kao pratitelj. Objekt P je invarijacija za transformaciju T ako mjereni objekt P zamijenimo s transformacijom T:

$$P(T(I)) = T(P(I)) \tag{2.5}$$

gdje I predstavlja našu sliku. Drugim riječima, ako izvodimo neke promjene na slici prije mjerenja objekta, dobit ćemo identične rezultate kao kad bismo mjerili objekt bez bilo kakvih promjena. Jednostavno rečeno, invarijacija znači da vizualna svojstva ostaju nepromjenjena pod utjecajem raznih promjena.

Jedan od najjednostavnijih elemena kojim možemo opisati konturu objekta je njezina duljina. Dobivena duljina konture može nam poslužiti za brzi test eliminacije nekih oblika objekta. Nedostatak je što nam ona očito nije dovoljna za rekonstrukciju oblika, niti nam omogućava da kažemo nešto više o istom. Ako izračunamo kvadratnu krivulju kroz cijelu konturu, dobivamo energiju nagnuća (eng. *bending energy*) kao jedini opisnik. Kako energija nagnuća ne opisuje dovoljno opsežno kao funkcije krivulje te nam jako malo otkriva o oblicima, može nam poslužiti samo za sparivanje različitih oblika. U nastavku ćemo opisati neke od poznatih konturnih opisnika. [4]

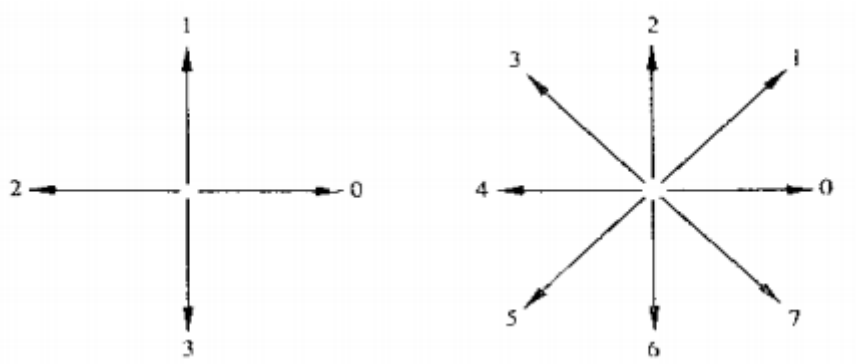
U osnovnim uvjetima pojam konture je definiran kao veza između piksela, tj. povezanost i susjedstvo. Piksel p na koordinati (x,y) ima četiri horizontalna i vertikalna susjeda, na koordinatama (x,y-1), (x-1,y), (x+1,y) i (x,y+1). Ta četiri horizontalna i vertikalna susjeda označavamo $N_4(p)$. Skup četiri dijagonalna susjeda $N_D(p)$, mogu biti definirana koordinatama (x-1,y-1), (x+1,y-1), (x-1,y+1) i (x+1,y+1). Sjedinjenjem tih piksela formira se skup od osam susjeda, $N_8(p)$. Skup piksela je povezan ako su susjedni, na način prethodno opisan i ako dijele iste kriterije sličnosti. Nakon segmentacije, trebali bih pripadati istome objektu. Prije traženja kontura sliga treba biti segmentirana te svaki piksel mora biti dodijeljen dijelovima objekta ili dijelovima pozadine. Nakon toga svi konturni pikseli mogu biti pronađeni jednim prijelazom preko slike.

Algoritam je nerekurzivni i ne podrazumijeva proceduru traženja svake konture, piksel po piksel. Svaki piksel je označen kao dio konture odgovarajućeg objekta ako:

1) Pripada objektu

2) Sadrži barem jedan od četiri susjedna piksela koji pripada pozadini

Sljedeći konturni opisnik koji ćemo detaljnije objasniti koristi se za lančane kodove. Lančani kodovi se koriste za predstavljanje kontura kroz povezane nizove vektora specifičnog smjera i duljine. Predstavljanje se bazira na povezanosti četiri ili osam segmenta, smjer svakog segmenta je predstavljen kodom opisanog preko Freedman-a (Slika 2.5).



Slika 2.5 *Smjer segmenta po Freedman-u [5]*

Kako su slike na ortogonalnoj koordinatnoj mreži, sa istim udjelom na obje osi, prvi pristup problemu bio bih praćenje konture, piksel po piksel. Ovaj direktan pristup nije praktičan, u jednu ruku zbog toga što lanac stvoren na ovaj način će biti predugačak, a u drugu ruku, šum i smetnje koje se pojavljuju duž konture, uslijed nesavršene segmentacije, dovest će do koda koji ne odraža pravilne karakteristike oblika objekta. Jedno od rješenje je ponovno traženje uzoraka konture koristeći koordinatnu mrežu veću od prethodne. Prateći konturu piksel po piksel, novi član u lancu, biti će dodan kada se te dvije mreže presijeku. Lančani kod ovisi i o početnoj točki.[5]

2.3. Usporedbe kontura

Kako bih došlo do nekog od oblika usporedbe kontura, prvi uvjet nam je određenim algoritmom dobiti rubove promatranih objekata. Postoje tri najčešće korištena algoritma praćenja rubova. To su: metoda središnjeg piksela, metoda kutnog piksela te metoda praćenja rubnih točaka. Ti algoritmi postoje u mnogim drugim modificiranim oblicima. U osnovi, oni traže rubne oblike na binarnoj fotografiji što je rezultat primjene filtera koji proizvodi oblik opisnika i time predstavlja oblik rubova. Ukratko ćemo opisati modificiranu metodu središnjeg piksela s direktnim pretraživanjem na primjeru keramičke pločice. Ovaj primjer će nam biti od velike pomoći za kasnije rješavanje našeg zadatka te će nas približiti metodi praćenja rubova .

Kao prvo, potrebno je pripremiti snimljenu fotografiju *I(x, y)* za prvi korak analize tj. za otkrivanje rubova pločice. Za točnije otkrivanje rubova naglašavamo pločicu u odnosu na njezinu podlogu koristeći jednostavnu metodu binarizacije koja je prethodno opisana, gdje je *B(x, y)* naglašena slika pločice a *T(I)* razina binarizacije.

$$B(x,y) = \begin{cases} 0 & if \quad I(x,y) \leq T(I), \\ 255 & if \quad I(x,y) > T(I) \end{cases} \tag{2.6}$$

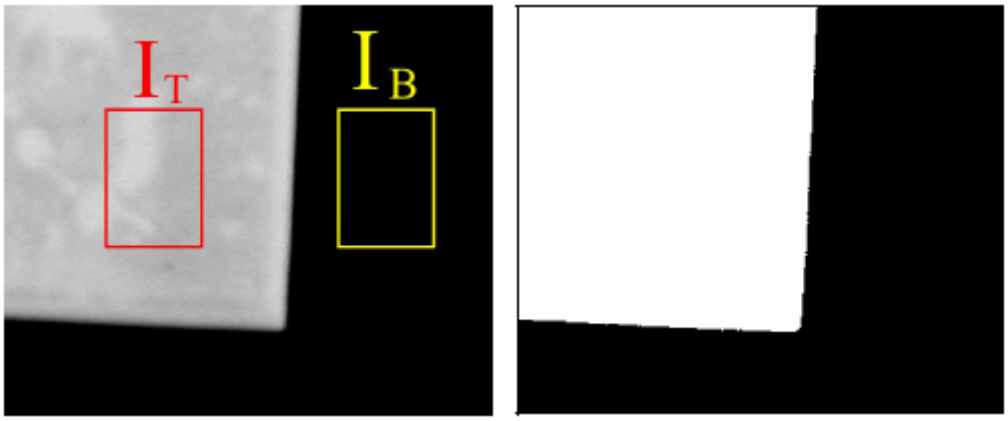
Korištena metoda binarizacije sadrži analizu histograma dijela pozadine fotografije, I_B, i analizu histograma dijela slike gdje se nalazi sama pločica, I_T. Nakon određivanja histograma, jednostavno oduzmemo ta dva histograma kako bi odredili novi histogram *H(I)* koji nam je potreban kod metode binarizacije.

$$H(I) = H_1(I_T) - H_2(I_B) \tag{2.7}$$

Kako bih odredili razinu binarizacije *T(I)*, pomoću novog histograma *H(I)* izračunamo njegovu srednju vrijednost *m(I)*, i standardnu devijaciju *s(I)*. Korištenjem sljedeće formule dobivamo vrijednost razine binarizacije :

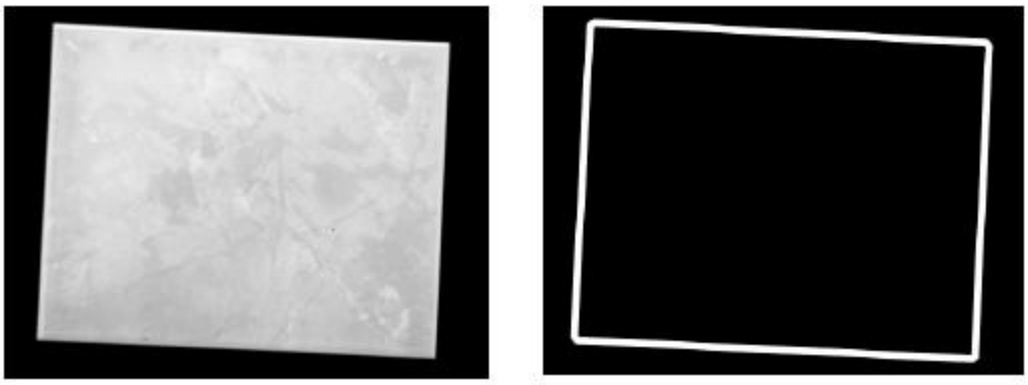
$$T(I) = m(I) - k*s(I) \tag{2.8}$$

gdje je *k* obično vrijednost od 0.8 do 1.2 za najbolje rezultate. Rezultat metode može se vidjeti na slici 2.5.



Slika 2.5 *Rezultat metode središnjeg piksela s direktnim pretraživanjem [6]*

U ovome radu nismo davala preveliku pažnju nepravilnim oblicima kontura, svu pažnju smo posvetili pravilnim oblicima kao što su i konture naših matica. Sljedeći korak kod modificirane metode središnjeg piksela s direktnim pretraživanjem je analiza rubova keramičke pločice. Za otkrivanje rubova obično se koristi *Canny* algoritam. *Canny* algoritam otkrivanja rubova daje nam prikaz konture keramičke pločice.[6]



Slika 2.6 *Otkrivanje rubova objekata korištenjem Canny algoritma [6]*

Metoda izravnog praćenja kontura se bazira na traženju rubnih piksela za koju su potrebne dodatne informacije o njihovom smjeru za pronalazak puta. Metoda se sastoji od dva dijela, tj. potrage za bilo kojim rubnim pikselom konture, *e(x,y)*, zajedno sa prethodno definiranim pretraživanjem smjera kuta, φ, na slici *E(x,y)*. Drugi dio predstavlja upisivanje rubnih piksela konture i njihovog smjera naspram njihove putanje.

Dva parametra su nam potrebna za direktno traženje i pronalaženje. Jedan je kut traženja smjera od prethodnog piksela konture (*step k-1*), φ *(k-1)* a drugi je prozor s koordinatama trenutnog piksela, *S(k)*, za traženje sljedećeg. Prozor za pretraživanja, *S(k)*, je matica s pet novo izračunatih koordinata piksela u koraku *k* naspram prijašnje lociranog rubnog piksela, *e(x,y)* u korak *k-1*. Matrica *S(k)* je dobivena iz formule:

$$S(k) = S(k-1)+D(k) \tag{2.9}$$

gdje je *D(k)* matrica sa izračunatim poželjnim smjerovima rubnih piksela.

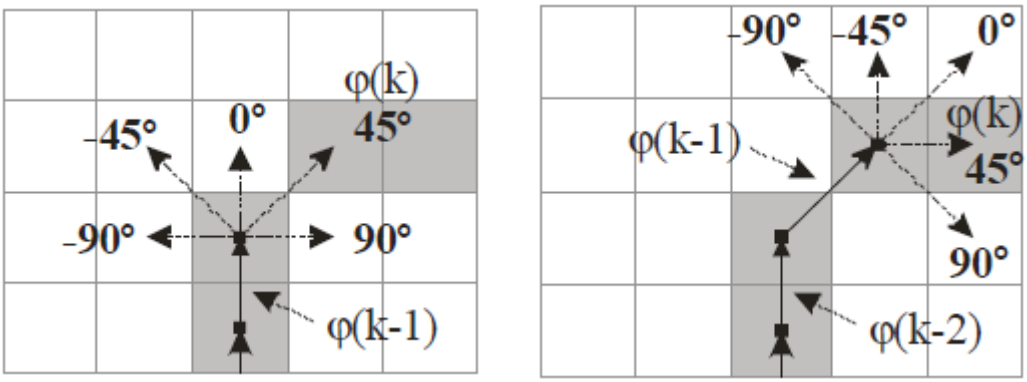
$$D(k) = \begin{bmatrix} \cos(\varphi(k-1)) & \sin(\varphi(k-1)) \\ \cos(\varphi(k-1)+\pi/4) & \sin(\varphi(k-1)+\pi/4) \\ \cos(\varphi(k-1)-\pi/4) & \sin(\varphi(k-1)-\pi/4) \\ \cos(\varphi(k-1)+\pi/2) & \sin(\varphi(k-1)+\pi/2) \\ \cos(\varphi(k-1)-\pi/2) & \sin(\varphi(k-1)-\pi/2) \end{bmatrix} \tag{2.10}$$

Poželjno traženje smjerova upisano je u matricu *D(k)*. Najpoželjniji smjerovi, prikazani su na slici 2.7.

Direction	Weight	Searching Order
0 °	0 – High Priority	First
45 °	1	Second
-45 °	2	Third
90 °	3	Fourth
-90°	4 – Least Priority	Fifth

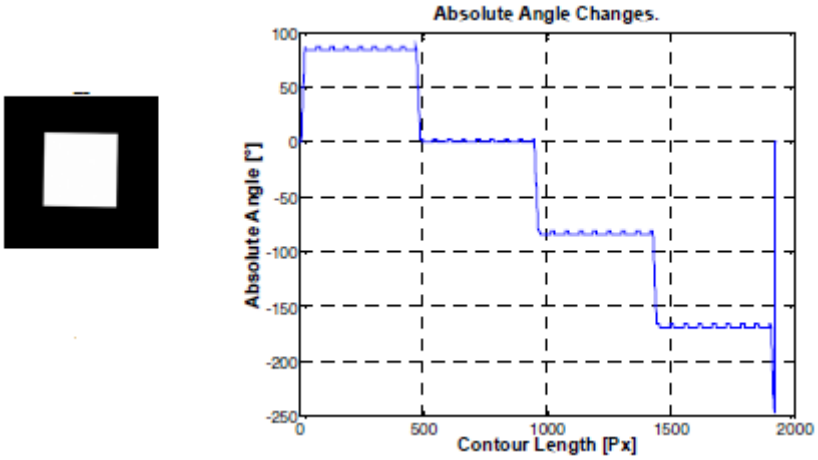
Slika 2.7 *Uvid u najpoželjnije smjerove rubnih piksela [6]*

Kada je *S(k)* izračunat, tada algoritam ispituje pet novih susjednih piksela. Algoritam ispituje jesu li ti pikseli rubni i ako jesu, označava ih kao rubne. Nakon pronalaska rubnih piksela, algoritam prihvaća novi smjer φ*(k)*, kao i φ*(k-1)* te nastavlja potragu. Na početku algoritma tretira prvog pronađenog rubnog piksela kao prvi piksel konture.

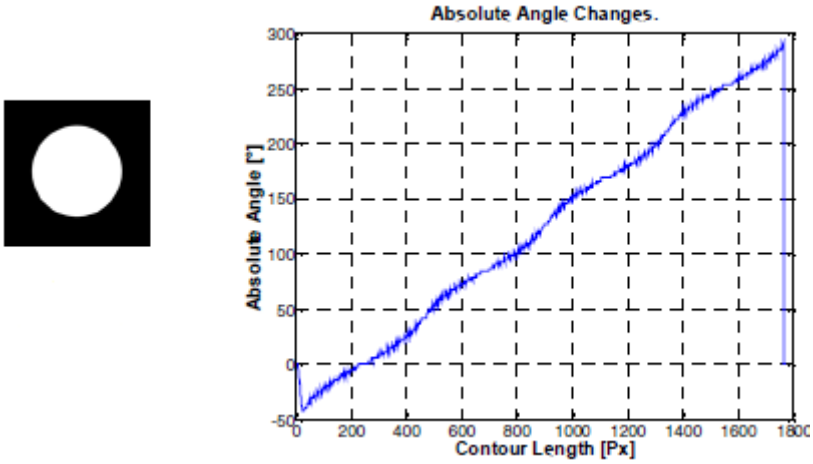


Slika 2.8 *Metoda direktnog pronalaska konture [6]*

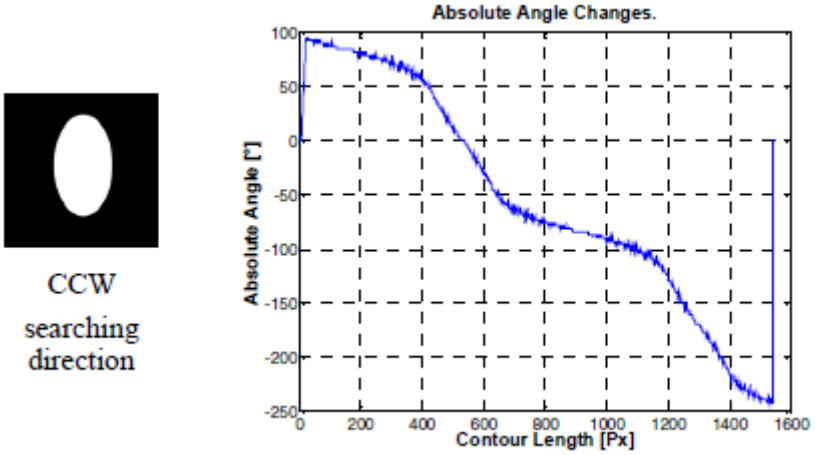
Algoritam zaustavlja potragu kada je došao do prvog piksela konture ili ako nema više piksela za ispitivanje u susjedstvu. Potencijalni problem predstavlja ako oblik konture sadrži rupe. Kako bih izbjegli ovaj problem, bilo bih dobro ispitati konturnu neprekidnost, te ako ima rupa, popuniti ih kako bi zatvorili konturu. U svakom koraku metode pronalaska konture za svaki locirani piksel dobijemo njezin smjer u odnosu na prethodni piksel. Kao rezultat, dobivamo vektor smjerova φ*(n)*, od *n* elemenata. Duljina vektora, *n*, odgovara duljini konture u pikselima i ta varijabla ovisi o veličini oblika konture i rezoluciji slike. Sljedeće slike prikazuju rezultate metode pretrage dobivene od nekoliko tipova oblika kontura. Svaki analizirani oblik ima različite rezultate od očekivanih rezultata koje smo mislili dobiti ovom metodom.



Slika 2.9 *Promjena kuta za kvadratni oblik [6]*



Slika 2.10 *Promjena kuta za oblik kruga [6]*

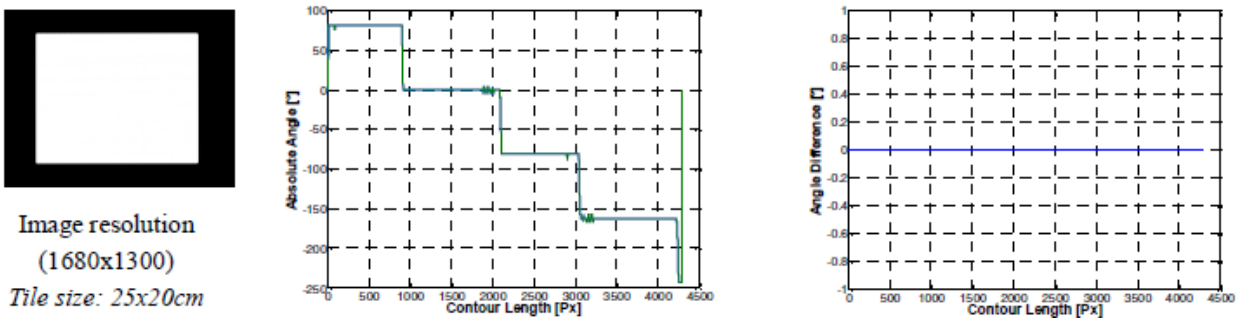


Slika 2.11 *Promjena kuta za oblik elipse [6]*

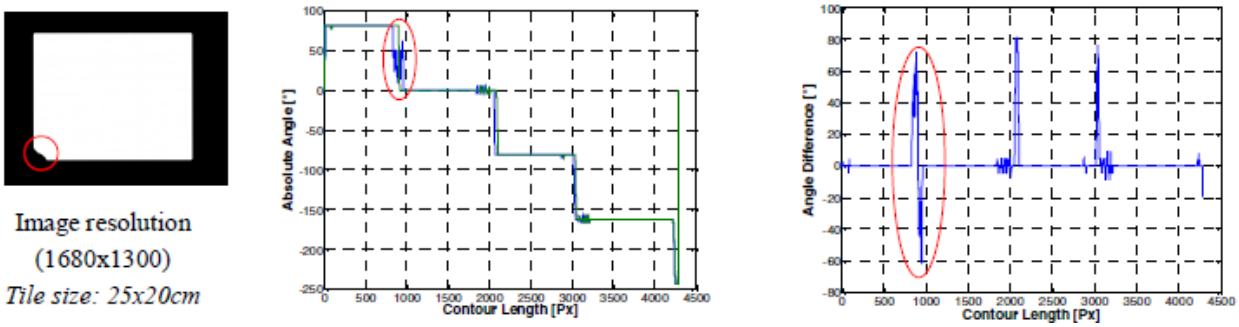
Rezultat praćenja konture, vektor s promjenom kuta φ, jedinstveno opisuje konturu oblika kutem u intervalu od 0 do 2π, te predstavlja konturni opisnik:

$$max(\varphi) - min(\varphi) = 2\pi \tag{2.11}$$

Nagib rezultata ovisi o početnoj točki praćenja smjera. Prvi uvjet za bilo kakvo uspoređivanje kontura je postojanje konturnog opisnika referentnog oblika koji nema nepravilnosti u konturi, φ_{ref}. Zbog toga što analizirani oblik može biti različite veličine potrebno je izračunati i duljinu referetnog oblika, L_{ref}. Nakon izračuna φ_{ref} i L_{ref} trenutno analizirani oblici mogu se uspoređivati s referentnim oblikom. Najlakši način pronalaska pogreške je da analizirani oblik usporedimo s referentnim veličinama, φ_{ref} i L_{ref} .Bilo kakvo odstupanje bih nam otkrilo da se oblici ne podudaraju. Možemo sa sigurnošću reći da oblici nisu identični te tu svaka daljna usporedba nije potrebna.



Slika 2.12 *Referentni model [6]*



Slika 2.13 *Analizirani model [6]*

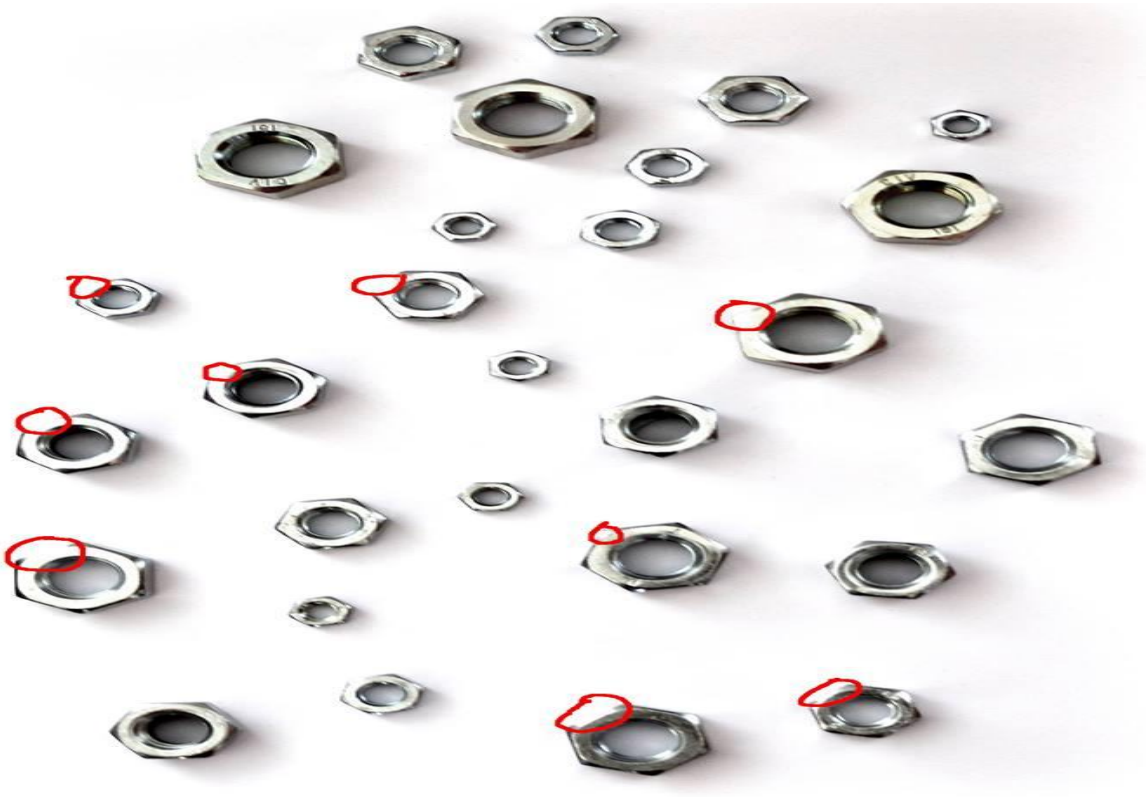
Na priloženim fotografijama vidi se jasna odstupanja veličina analiziranog modela u odnosu na referentne veličine. Ovdje daljna usporedba staje te se analizirani model odbacuje jer ne ispunjava tražene uvjete.[6]

3. REALIZACIJA SUSTAVA ZA RAZVRSTAVANJE OBJEKATA

Kako bih se program za razvrstavanje matica sa slike realizirao, potrebno je cjelokupni problem podijeliti u što više cjelina te zatim navedene cjeline detaljno razraditi. Osnovni algoritam za rješavanje ovog problema može se podijeliti u sljedeće cjeline:

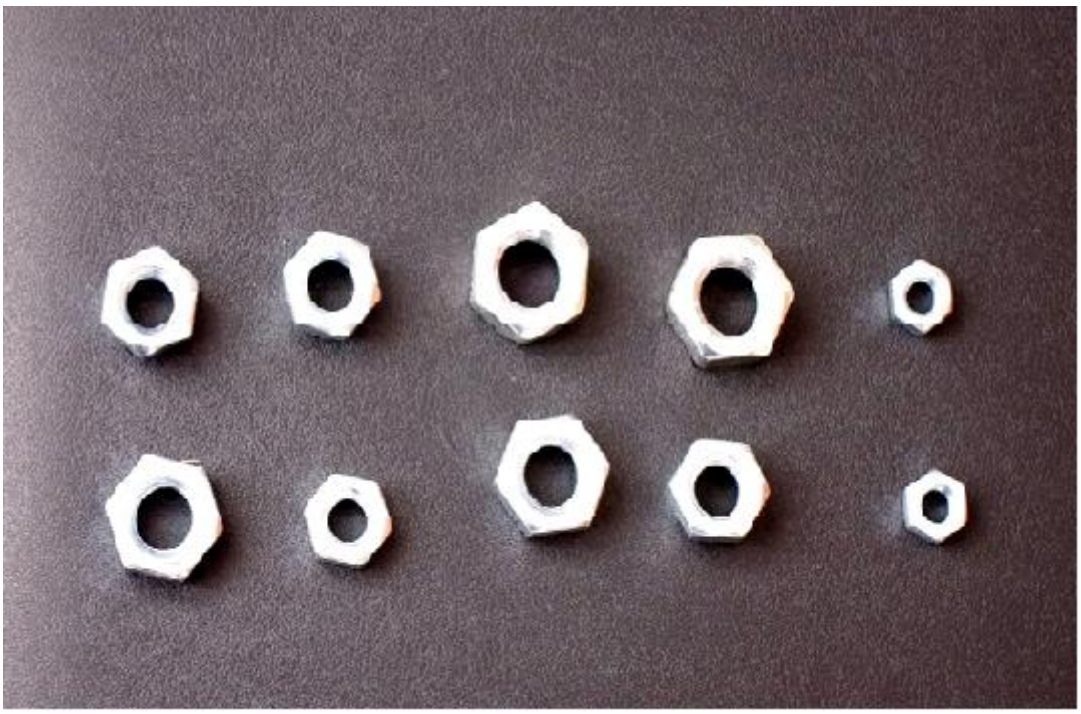
1. Snimanje fotografija visoke kvalitete
2. Priprema fotografija za obradu
3. Obrada fotografija
4. Prebrojavanje i grupiranje matica

Glavni problem, kao kod svake analize slike, predstavljala nam je kvaliteta fotografije. Kako bi fotografije bile što upotrebljive za naš program, visoka kvaliteta je jedan od najpotrebnijih uvjeta. Čak i kod posebnih uvjeta slikanja nailazili smo na manje oblike šuma. Šum je jedan od najvećih problema koji se javlja kod digitalne obrade fotografije i jako ga je teško u potpunosti ukloniti. Prvo smo kod slikanja koristili bijelu boju kao podlogu za naše matice. Shvatili smo da je od velike važnosti da nam boja podloge bude tamnija u odnosu na matice. Korištenjem svjetlijih podloga ne možemo izbjeći prekide u konturama zbog kojih program neće moći funkcionirati.



Slika 3.1 *Prekidi kontura uzrokovani zbog bijele podloge*

Jedan od uvjeta koji smo trebali zadovoljiti prije početka realizacije aplikacije jest da boja podloge bude različita ili suprotna tj. kontrastna od boje matica.



Slika 3.2 *Uspostavljen kontrast boje podloge i matica*

3.1. Korišteni alati i programska sučelja

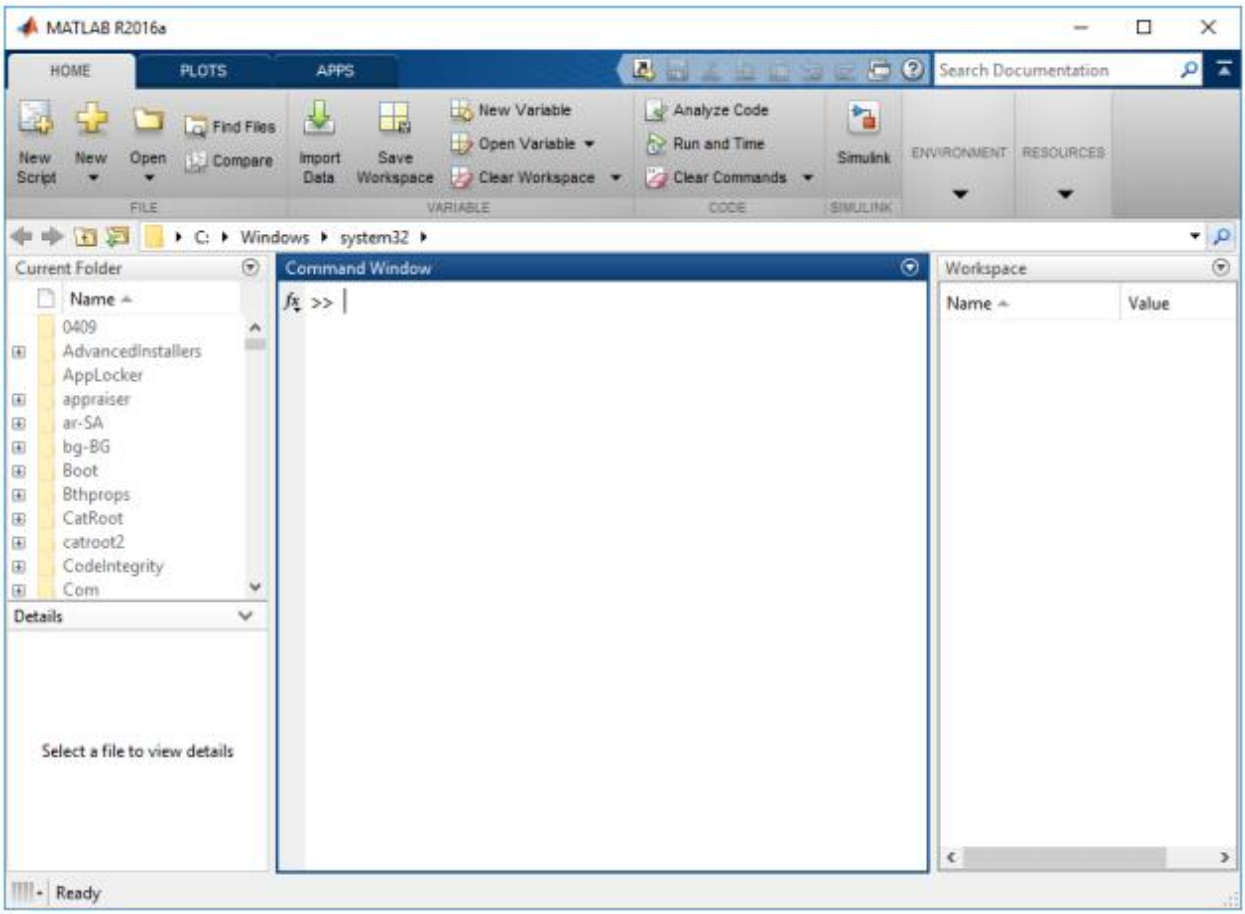
MATLAB (eng. **mat**rix **lab**oratory) je programski jezik četvrte generacije razvijen od strane američke korporacije *MathWorks* koja se specijalizira za izradu programske podrške za matematičke proračune. MATLAB omogućava manipulaciju matricama, grafički prikaz funkcija i podataka, implementaciju algoritama, izradu vlastitog sučelja, mogućnost obrade slike i brojne druge mogućnosti. Kratki vodič kroz osnove programiranja i obrade slike u MATLAB programskom okruženju možete pronaći u [7].

MATLAB skriptni jezik često se uspoređuje s C++, Java i Python programskim jezicima. MATLAB je skriptni jezik koji je izrađen isključivo s ciljem olakšavanja i ubrzavanja matematičkih i drugih proračuna dok su C++ i Python programski jezici opće namjene iako u današnje vrijeme postoje razne biblioteke koje omogućavaju slične ili iste mogućnosti. Prednosti MATLAB skriptnog jezika u odnosu na druge programske jezike uključuje efikasnost proračuna s matricama i poljima, jednostavan način prikaza i vizualizacije podataka, proširivost (dostupnost raznih alata – *toolbox*), jednostavan način upravljanja datotekama, jednostavan način obrade nizova, nema potrebe za deklariranjem varijabli – sve što je potrebno je njihova inicijalizacija.

Iako ima brojne prednosti, MATLAB skripti jezik ima i nekoliko iznimno bitnih nedostataka u odnosu na ostale objektno – orjentirane jezike. Vjerojatno najbitniji nedostatak odnosi se na brzinu izvođenja MATLAB skripti. Može se pokazati da je za rješavanje istog problema MATLAB i do nekoliko puta sporiji u odnosu na druge programske jezike (prema [8]). Kako bi se ovaj nedostatak izbjegao može se provesti paralelizacija koda ili je pak moguće pribjeći korištenju GPU biblioteka (CUDA, OpenCV, arrayFire i dr.) te izvršavati kod na grafičkoj kartici. U tom slučaju je potrebno dodati nešto više linija koda budući da se sve varijable i podaci, nad kojima se vrši obrada, moraju prebaciti na grafičku karticu te zatim nakon završetka obrade i obavljenih proračuna natrag u radni prostor MATLAB-a.

Na slici 3.3 prikazan je osnovni izgleb MATLAB programskog okružja. Najbitniji dijelovi grafičkog sučelja su komadni prozor (eng. *command window*) i radni prostor (eng. *workspace*). Komadni prozor jedan je od glavnih alata za unos podataka, izvršavanje MATLAB funkcija i M–datoteka kao i za prikaz dobivenih matematičkih rezultata. Grafički rezultati (grafovi, histografi i sl.) prikazuju se u odgovarajućem skočnom prozoru. Radni prostor omogućava prikaz i manipulaciju svim kreiranim varijablama koje su direktno postavljene ili izračunate tijekom MATLAB sesije. Od ostalih bitnih dijelova MATLAB grafičkog sučelja može se istaknuti i

povijest naredbi (eng. *command history*) koja omogućava brzi pristup prethodno pozivanim naredbama i linijama koda (do najviše 25 000) te trenutni direktorij (eng. *current directory*) koji omogućava brzi pristup datotekama u trenutnom radnom direktoriju MATLAB-a.



Slika 3.3 *MATLAB programsko okruženje [7]*

Također, treba napomenuti da je jedna od velikih prednosti MATLAB-a kompatibilnost verzija. Osim standardne kompatibilnosti, odnosno mogućnosti pokretanja programa kreiranih u nekoj od starijih verzija u novijoj verziji, MATLAB omogućava i pokretanja programa kreiranog u novijoj verziji u bilo kojoj starijoj verziji bez potrebe za dodatnim modifikacijama koda kao što je to primjerice potrebno pri korištenju *MS Visual Studio* programskog okruženja gdje je potrebno u zaglavlju glavne programske datoteke promijeniti verziju u kojoj se program pokreće. Naravno, postoje slučajevi gdje se određene funkcije neće moći pokrenuti ako su uvedene isključivo u novijoj verziji što se lako može popraviti dodavanjem M-datoteka u direktorij MATLAB-a

ukoliko su datoteke, kojima se definirane navedene funkcije, dostupne za besplatno preuzimanje putem službenih stranica. U suprotnom potrebno je izvršiti nadogradnju sustava.

3.2. Aplikacija

Kompletan izvorni kod MATLAB aplikacije nalazi se u prilogu dokumenta.

Prilikom obrade slike najčešće se koristi binarna slika. Binarna slika omogućuje izdvajanje dijelova slike ili piksela od interesa korištenjem samo dvije boje. Boje koje se koriste su najčešće crna i bijela, iako je moguće koristiti bilo koje dvije boje. Objekti na slici su u njoj prikazani jednom bojom koju nazivamo prednja boja (engl. *foreground)*, a ostali dio slike je prikazan drugom bojom koju nazivamo pozadinska boja (engl. *background)*. Boja objekta je binarno zapisana kao 1, a pozadinska boja kao 0.

Kako bi orginalnu sliku pretvorili u binarnu potrebno je prvo od orginalne slike stvoriti sivu sliku (engl. g*rayscale image*). Za pretvorbu slike korištena je ugrađena MATLAB funkcija *rgb2gray* koja kao argument prima naziv orginalne slike, i vraća stvorenu sivu sliku. Funkcija za stvaranje binarne slike *im2bw* koja je ugrađena u MATLAB prvo orginalnu sliku prebacuju u sivu, te iz nje stvara binarnu sliku. Ukoliko se koristi ova funkcija, nije potrebno dodavati posebnu naredbu za stvaranje sive slike, odnosno nije potrebno prvo stvarati sivu sliku te onda nju koristiti za stvaranje binarne slike.

Funkcija *im2bw* prima dva argumenta. Prvi argument je slika koju želimo pretvoriti u binarnu sliku, a druga je prag. Prag je vrijednost koja je granica između 0 i 1, odnosno između dviju boja na binarnoj slici. Odrediti granicu između dviju boja je potrebno, jer neki dijelovi slike svojim intenzitetom se mogu nalaziti između dviju boja, odnosno granica između objekta i pozadine na slici nije potpuno oštra. Prag omogućuje da se dijelovima slike koji nisu prepoznati kao 0 ili 1, pridruži jedna od tih vrijednosti. Ukoliko je na primjer prag 0.5, tada svi dijelovi čija vrijednost je veća ili jednaka od 0.5 će biti prikazani s 1, a ispod te vrijednosti će biti prikazani s 0.

```
im2bw(image,0.7);
```

Vrijednost praga moguće je ručno odrediti metodom isprobavanja više pragova što zahtijeva dosta vremena. Zbog toga se razvijaju metode koje pomažu prilikom određivanja pragova pa se na primjer vrijednost praga kod programa vezanih uz računalni vid i obradu slike najčešće

određuje Otsu metodom. Ova metoda se temelji na pretpostavci da se na slici koja se pretvara u binarnu sliku nalaze dvije vrste piksela (objekti i pozadina). Koristeći tu pretpostavku optimalni prag razdvajanja se određuje tako da se varijanca između dviju boja minimizira. U MATLABU za određivanje praga Otsovom metodom se koristi *graythresh* metoda. Iako, ova metoda za određivanje praga daje dobre rezultate, potrebno ju je koristiti s oprezom jer njezini rezultati značajno ovise o osvjetljenju i količini na slici prisutnih svijetlih i tamnih objekata. Zbog toga je ovu metodu dobro koristiti na slikama kojima je prethodno popravljen kontrast.

Prilikom stvaranja binarne slike na njoj se mogu pojaviti dijelovi koji su prikazani bojom 1, a koji ne pripadaju objektu slike. Do takvih pojava dolazi zbog različitih šumova i nedovoljne kvalitete slike. S binarne slike je potrebno ukloniti što više takvih pojava. U ovoj aplikaciji se takvi objekti uklanjaju MATLAB-ovom funkcijom *bwareaopen* koja uklanja povezane objekte koji imaju manji broj piksela nego što je broj koji je predan kao argument funkciji. Osim broja koji određuje veličinu objekta koji će biti uklonjen, ovoj funkciji je potrebno predati i binarnu sliku.

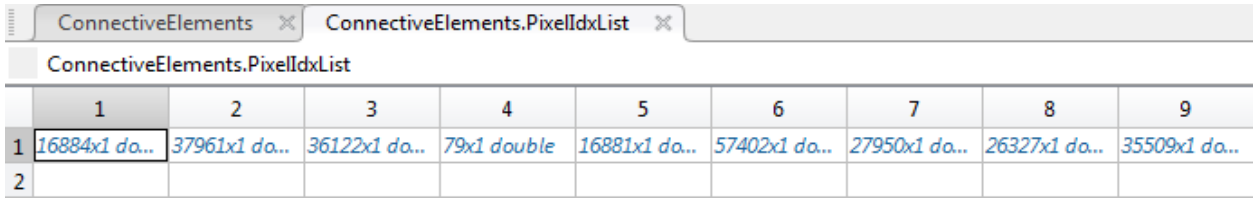
```
bwareaopen(bwImage, 70);
```

S obzirom da se obrada temelji na slici na kojoj se nalaze matice, a matice imaju praznu sredinu, potrebno je popuniti taj prostor, kako bi se dobio poligon koji će onda moći biti prepoznat i izbrojen. Za popunjavanje sredine matice korištena je MATLAB-ova funkcija *imfill* kojoj je kao argument predana binarna slika, a drugi argument je postavljen na *'holes'*. Ova funkcija popunjava 'rupe' unutar okvira prepoznatih objekta. Rupa je definirana kao skup pozadinskih piksela kojima se ne može pristupiti popunjavanjem pozadine krenuvši od ruba slike.

Nakon popunjavanja sredine matice, korištenjem funkcije *bwconncomp* sa binarne slike dobijamo sve povezane komponente. Funkcija radi na način da provjerava osam susjednih piksela od određenog piksela. Susjednim pikselima pridodaje vrijednosti 0 i 1 te na taj način dobijemo točnu veličinu određene matice izraženu u pikselima. Nakon iščitavanja svih povezanih komponenti, potrebno je još jednom provjeriti veličinu tih komponenti te ukoliko su one manje od određenog broja piksela tada ih se ne prepoznaje kao matica, odnosno ne broji.

Slika 3.4 *Rezultat pozivanja MATLAB funkcije bwconncomp*

Nakon realizacije i određivanja ukupnog broja matica preostaje navedene detektirane matice razvrstati i prikazati odgovarajućim grafom. Sortiranjem matica po veličini vrši se korištenjem polja prikaznih na slici 3.4. Naime, pristupom polje *PixelIdxList* unutar strukture *ConnectiveElements* možemo pristupiti veličini svake detektirane matrice. Iako je slika filtrirana i otklonjena je većina smetnji koje su se pojavile, i dalje postoji mogućnost lažnog detektiranja matica. Konkretno za navedeni primjer prikazan na slici 3.4 iz polja *PixelIdxList* može se vidjeti da je detektirano devet matica iako ih je na slici bilo osam.



Slika 3.5 *Sadržaj polja PixelIdxList*

Kako bih se ovakav problem rješio potrebno je postaviti granični broj povezanih elemenata (veličinu matice) za koje se podrazumijeva da jesu,odnosno nisu, matice. Teoretski za graničnu vrijednost može se postaviti bilo koji broj sve dok je taj broj manji od veličine najmanje matice no u tom slučaju program ne bih bio univerzalan te bih prepoznao samo određene veličine matica te je stoga najbolje za graničnu vrijednost postaviti vrijednost od 0 do 1000 (konkretno u ovome radu postavljena je vrijednost 400).

```
If length(ConnectiveElements.PixelIdxList{i}) >400
    connectObject_Numbers=connectObject_Numbers+1;
```

Provođenjem ovog postupka dolazi do osam matica. Na kraju preostaje navedene matice grupirati prema veličini na način da se matice približno istih dimenzija grupiraju zajedno. Najjednostavniji način provođenja ove metode je korištenje funkcije *pdist()* pomoću koje računamo razliku u veličini svih osam matica.

Nakon izračuna razlika u veličini, matice grupiramo korištenjem naredbi *linkage()* i *cluster()*. Bitno je napomenuti da je prilikom korištenja naredbe *cluster()* potrebno i specifirati prag koji određuje koliko će grupa matica postojati. Primjerice u radu je korištena vrijednost praga 0.71 na temelju kojeg dobijemo četiri grupe matica što je i korektno ukoliko pogledamo njihove veličine. Ukoliko bismo prag smanjili, matice bi se grupirale u više grupa i obrnuto.

```
Cluster(Z,'cutoff',0.71);
```

Način rada programa i dobiveni rezultati prikazani su u nastavku.

4. TESTIRANJE I REZULTATI

4.1. Metodologija testiranja

Testiranje softvera je proces verifikacije programa ili sustava s ciljem otkrivanja i ispravljanja grešaka te prestavlja integralni dio razvoja softvera. Primjenjuje se u svakoj fazi razvojnog ciklusa, a obuhvaća vise od 50% vremena potrebnog za razvoj softvera. Ciljevi testiranja softvera su: verifikacija i potvrđivanje softvera, poboljšanje kvalitete softvera i procjena pouzdanosti. Metodologija testiranja softverskog proizvoda podrazumijeva ciklus koji se sastoji od sljedećih faza: otklanjanje neispravnosti, testiranje ispravnosti, testiranje performansi i testiranje pouzdanosti.

Otklanjanje neispravnosti podrazumijeva otklanjanje sintaksičkih i logičkih grešaka tokom razvoja aplikacije.

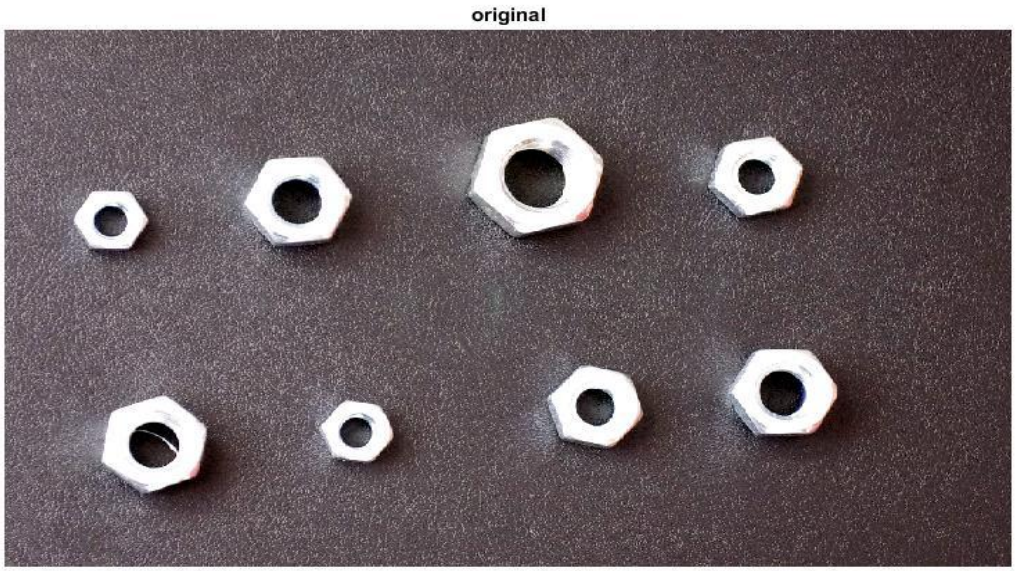
Testiranje ispravnosti obuhvaća sljedeće metode: funkcionalno testiranje i testiranje dizajna softvera. Kod funkcionalnog testiranja izvršava se samo analiziranje specificiranih funkcija i vrši se provjera ulaznih i izlaznih podataka. U ovim testovima se ne vrši analiza izvornog koda. Skoro 30% svih grešaka u kodu posljedica su problema nepotpunih ili neodređenih specifikacija. Testiranje dizajna softvera provjerava i analizira izvorni kod i zahtjeva dobro poznavanje programiranja, odgovarajućeg programskog jezika, kao i dizajna konkretnog softverskog proizvoda. Plan testiranja se određuje na osnovi elemenata implementacije softvera, kao što su programski jezik, logika i stilovi. Testovi se izvode na osnovu strukture programa. Kod ove metode postoji mogućnost provjere skoro cjelokupnog koda, na primjer provjerom da li se svaka linija koda izvršava barem jednom ili provjerom svih funkcija.

Testiranje performansi obuhvaća pronalaženje i otklanjanje koji smanjuju performanse softvera. Ispitivanje performansi najčešće obuhvaća: iskorištenost procesorskih resursa, protok podataka i vrijeme odziva. Tipični resursi koji se provjeravaju su: propusni opseg, brzina procesora, iskorištenost memorije, zauzetost prostora na disku. Testiranje pouzdanosti određuje vjerojatnost funkcioniranja softvera bez grešaka i bez stajanja. Testiranje robusnosti i testiranje opterećenjem su primjeri testova pouzdanosti, koji pokušavaju da izazovu kvar softvera u određenim situacijama. [9]

Pisanje našeg koda je počelo tek nakon izrade dovoljno kvalitetnih fotografija koje mogu biti iskorištene za dobivanje potrebnih rezultata. Također i originalna slika je morala proći neku vrstu obrade slike. Prvotno smo originalnu sliku s maticama posivili iliti potamnili. Tek nakon toga smo mogli pretvoriti posivljenu sliku u binarnu sliku s kojom program može raditi. Uklonili smo sve šumove i smetnje sa slike te smo pristupili punjenju praznine matica kako bismo dobili cjelovit i pun oblik. Matlab funkcijom *bwconncomp* smo povezali sve elemente s binarne slike što nam je kasnije omogućilo da dobijemo broj matica koje se nalaze na slici. Nakon početnih problema napokon je izrada aplikacije pokazala neki napredak. Dobijanjem broja elemenata sa slike preostalo nam je samo nekako grupirati elemente slike. Matice smo grupirali po njihovoj veličini te se našem radu počeo nadzirati kraj. Shvatilo smo da bilo koje mjerenje ili bilo koja usporedba objekata sa slika ne može nikada biti savršena, možemo samo pokušati i potruditi se da bude što točnije.

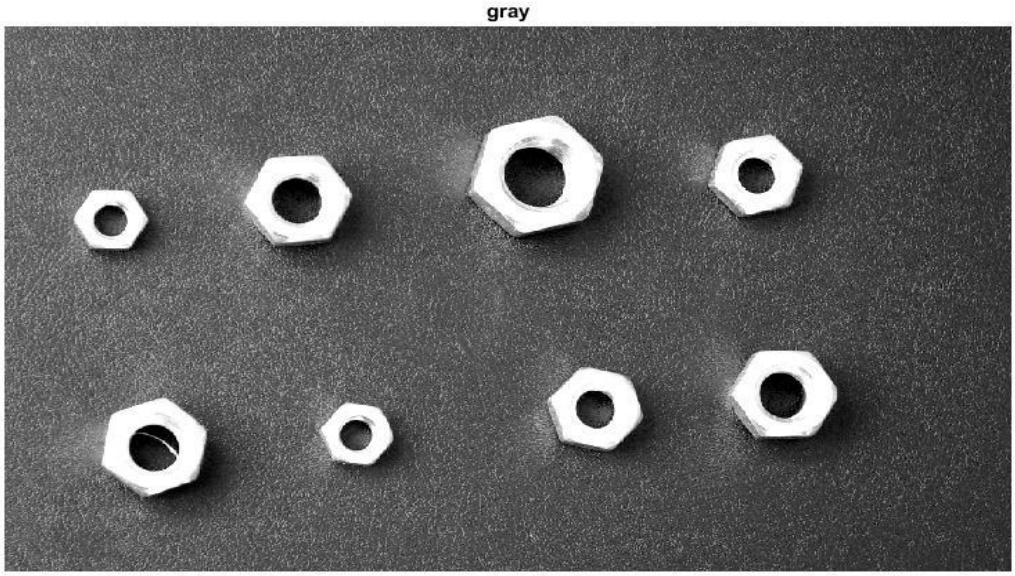
4.2. Rezultati testiranja

Rezultati dobiveni primjenom prethodno opisane metode direktno ovise o uvjetima u kojima su fotografije stvorene. Poželjno je da se snimanje fotografija provodi u kontroliranim uvjetima, gdje je iznimno dobro osvjetljenje kako bi se otklonio utjecaj sjene. Fotografije korištene za potrebe našeg rada snimljene su rezolucijom 2180*x*1296. Na slici 4.1 možemo primjetiti pojavu refleksije svjetlosti koja nam nije stvarala veće probleme u radu.



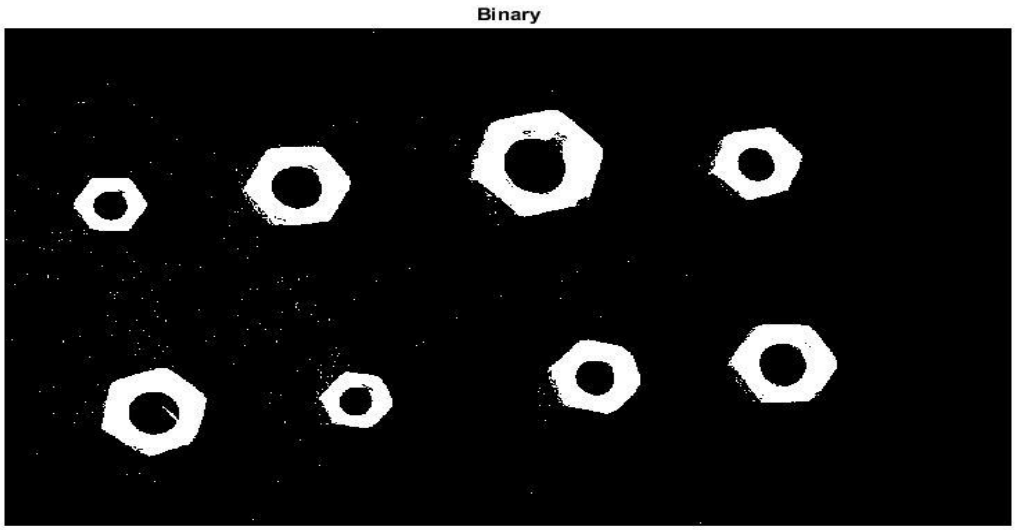
Slika 4.1 *Originalna slika*

S ciljem pretvaranja originalne slike u binarnu pristupili smo prvo stvaranju sive slike. To je za nas obavila ugrađena MATLAB funkcija *rgb2gray* koja kao argument prima naziv originalne slike te je zatim vraća u sivoj boji.

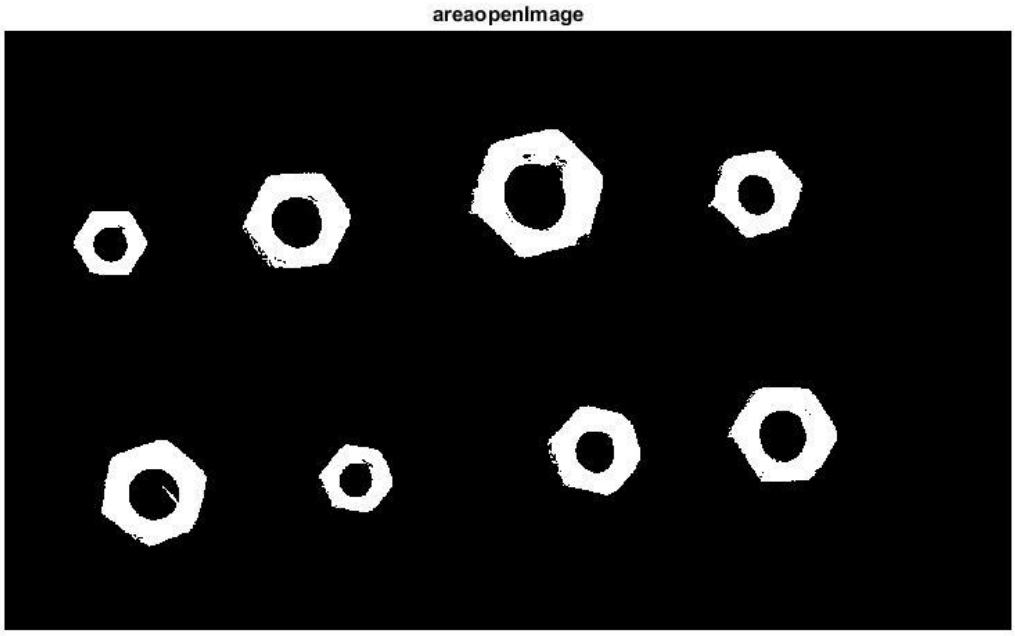


Slika 4.2 *Posivljena slika*

Binarnu sliku smo dobili pomoću ugrađene MATLAB funkcije *im2bw* koja je primila prethodno posivljenu sliku a vratila binarnu. Prag smo stavili na 0.7.

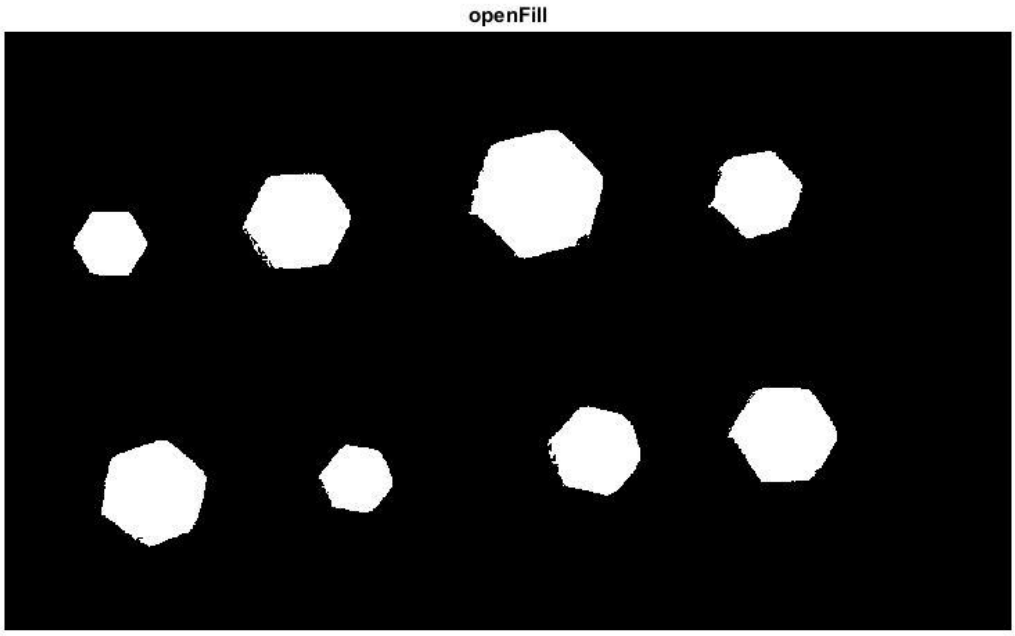


Slika 4.3 *Binarna slika*



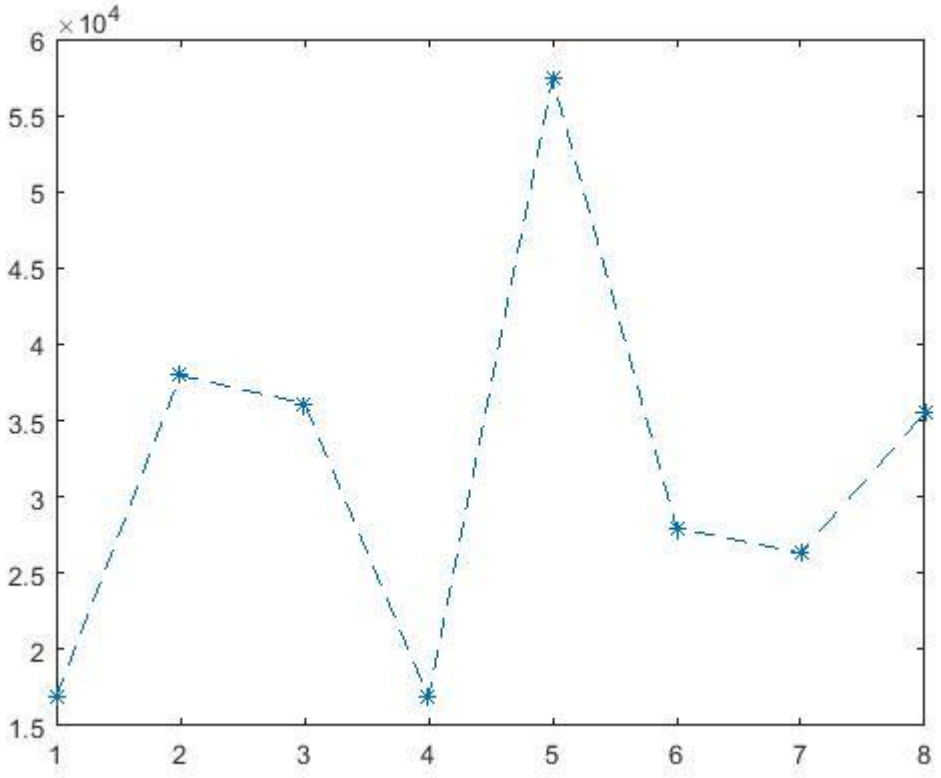
Slika 4.4 *Prikaz slike s uklonjenim smetnjama*

Uklonili smo sve nepotrebne pojave s binarne slike MATLAB-ov funkcijom *bwareaopen*. Funkcija uklanja povezane objekte koji imaju manje piksela od postavljenog praga (70).



Slika 4.5 *Ispunjene praznine matica*

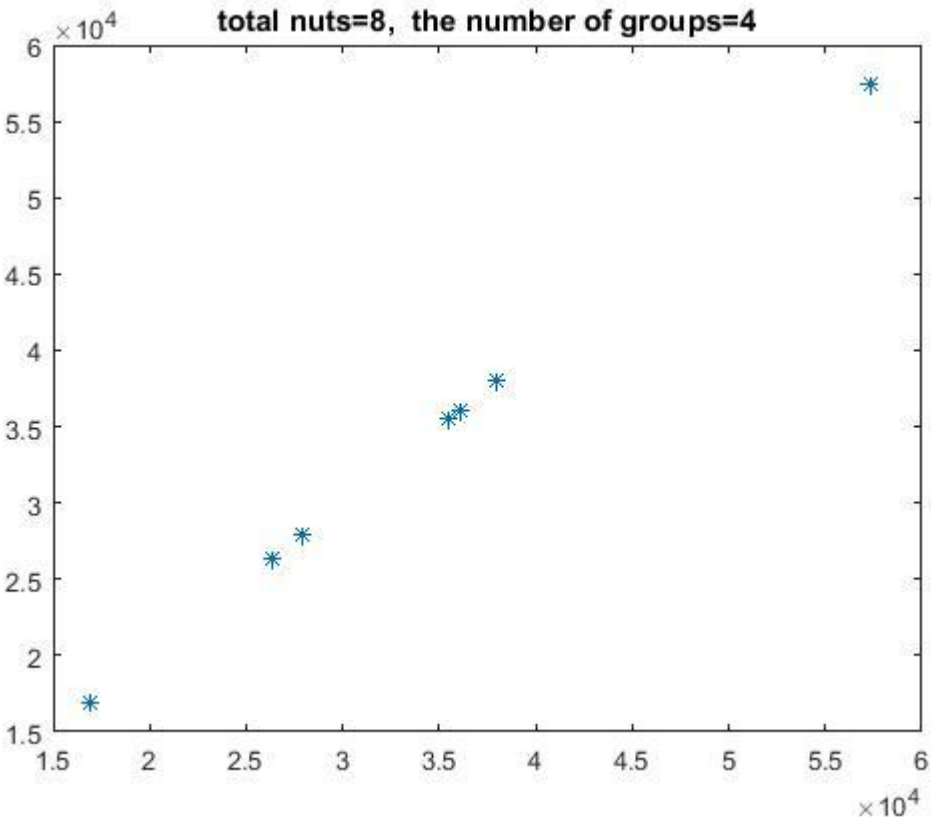
S obzirom da matice imaju praznu sredinu, bilo je potrebno popuniti taj prostor, kako bi se dobio poligon koji će onda moći biti prepoznat i izbrojen. Korištena je MATLAB funkcija *imfill*.



Slika 4.6 *Graf ovisnosti broja matica o njihovim veličinama u pikselima*

Svaka ″zvjezdica″ predstavlja jednu maticu. Spajanjem matica dobili smo graf u ovisnosti matica s njihovom veličinom u pikselima. Os apscisa daje nam uvid u točan broj matica dok nam os ordinata daje uvid u njihovu veličinu izraženu u pikselima.

MATLAB funkcija *repmat()* pomogla nam je da dobijemo graf sa slike 4.7. Stvorene su dvije matrice s identičnim elementima pomoću kojih smo dobili prikazano. I os apscisa i os ordinata prikazuju veličinu matica izraženu u pikselima. Na grafu se jasno može vidjeti točan broj matica kao i grupe matica.



Slika 4.7 *Grupirane matice po veličini izraženoj u pikselima*

5. ZAKLJUČAK

U ovome radu u prvi plan je stavljena obrada fotografije. Današnja obrada fotografije u općenitom smislu ima dosta mjesta za napredak. Od velike je važnosti kvaliteta fotografije bez koje nema smisla počinjati ikakvu detaljniju obradu slike. Treba biti realan i shvatiti da nikada, čak ni u posebnim uvjetima naslikavanja fotografija neće biti savršena. Koliko god se trudili uvijek će postojati prisutnost ako ne velikih, onda malih smetnji i šumova. Nakon početnih problema izrada rada je polako krenula i sve je počelo imati smisla. Upoznao sam se detaljnije s MATLABOM te shvatio kolike mogućnosti nam pruža to programsko okruženje. S maticama smo uglavnom svi upoznati, one su bile temelj našeg rada. Sam program omogućava pretraživanje njezinih oblika ali jako male promjene u kodu mogu poslužiti za pronalaženje i drugih, raznih oblika. Rad mi je pomogao da se upoznam puno detaljnije s obradom slike nego što sam ikada ranije bio u mogućnosti. Nakon završetka rada upoznat sam tek s malim dijelom obrade slike, ali koje će mi svakako pomoći u razvijanju daljnih programerskih vještina.

LITERATURA

[1] ″Kontura – Vidljiva granica objekata u prostoru, www.crtanje-i-slike.com/kontura.html, 25.8.2016.″

[2] Ang, Tom; Digitalna fotografija priručnik, Znanje d.d., Zagreb, Hrvatska, 2007.

[3] Jain, R., Kasturi, R., Schunck, B.G. Machine Vision 1995.

[4] Bryan S. Morse, ″Snape Description (Contours)″, Brigham Young University 1998.-2000.

[5] Zingaretti, P., Gasparroni, M., Vece, L., ″Fast Chain Coding of Region Boundaries″. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol.20, No.4, April 1998.

[6] Ž.Hocenski, T.Keser; Failure Detection and Isolation in Ceramic Tile Edges Based on Contour Descriptor Analysis, University J.J. Strossmayer, Faculty of Electrical Engineering, Osijek, Croatia

[7] ″Image processing with MATLAB, http://goo.gl/zV5cde, 10.9.2016.″

[8] T. Andrews, Computation Time Comparison Between Matlab and C++ Using Launch Windows, California Polytechnic State University San Luis Obisbo, SAD, rujan 2016.

[9] W.E.Perry, Quality Assurence for Information Systems: Methods,Tools and Techniques, QED Information Sciences, Inc, Wellesley, MA 1991.

SAŽETAK

U ovome radu zadatak je bio izraditi aplikaciju u MATLAB okruženju. Aplikacija je trebala prebrojati matice sa slike i grupirati ih po njihovim veličinama. Do njihovih veličina smo došli preko kontura matica. Matice su slikane na tamnoj podlozi kako bi došli do što kvalitetnijih rezultata. Veliku ulogu u radu je igrala kvaliteta izrađenih fotografija.

Ključne riječi: MATLAB, matice, konture, fotografija

ABSTRACT

The aim of this thesis was to design the application in MATLAB environment. The application suppose to count the nuts, shown in a certain photo and sort them by their size. The size of nuts was indetified on the base of their contour. Nuts were photographed on a dark background, with the aim of improving results quality. The quality of photos played an important role in this task.

Key words: MATLAB, nuts, contour, photo

ŽIVOTOPIS

Ante Pole rođen je u Zagrebu, Republika Hrvatska. Odrastao je u Mohovu, završio Osnovnu školu Šarengrad u Šarengradu s odličnim uspjehom. Potom upisuje Opću gimnaziju u Iloku koju je također završio s odličnim uspjehom. U slobodno vrijeme bavi se nogometom. 2010. godine upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku.

PRILOZI

Kompletan izvorni kod MATLAB aplikacije, kao i svi dodatni materijale nalaze se na CD-u priloženom uz ovaj rad. U nastavku slijedi kod programa.

```
clc;
clear;
close all;


image = imread('22.jpg');
figure;imshow(image);  title('original')


grayscaleImage = rgb2gray(image);
figure;imshow(grayscaleImage);title('gray')

bwImage=im2bw(image,0.7);
figure;imshow(bwImage);title('Binary')

areaopenImage = bwareaopen(bwImage, 70);
figure;imshow(areaopenImage);title('areaopenImage')



openFillImage=imfill(areaopenImage,'holes');
figure; imshow(openFillImage);title('openFill')

ConnectiveElements = bwconncomp(openFillImage);

    connectObject_Numbers=ConnectiveElements.NumObjects ;
    imRow=ConnectiveElements.ImageSize(1);
    imCol=ConnectiveElements.ImageSize(2);
    eachObjectSize=zeros(1,connectObject_Numbers) ;

    connectObject_Numbers=0;
    Area=[];
    for i=1:ConnectiveElements.NumObjects
        if length(ConnectiveElements.PixelIdxList{i}) >400
            connectObject_Numbers=connectObject_Numbers+1;
            Area=[Area length(ConnectiveElements.PixelIdxList{i})];
        end
    end


    figure;plot(Area,'*--');
    doubleArea=repmat(Area,2,1);
    T =
clusterdata(doubleArea,'maxclust',7,'linkage','ward','savememory','on');
```

```
    kkk=1;

    figure;plot(Area,Area,'*');


    areaDist = pdist(doubleArea');
    Z = linkage(areaDist);
    cc = cluster(Z,'cutoff',0.71);

    groupTotalNumber=max(cc);

    groupNumberPerNutSize=zeros(1,max(cc));
    for i=1:groupTotalNumber
        groupNumberPerNutSize(i)=length(find(cc==i));
    end


    TotalNumberOfNuts=length(Area)
    groupNumber_Per_NutSize=groupNumberPerNutSize

    title(strcat('total nuts=',num2str(TotalNumberOfNuts),',  the number of
groups=',num2str(groupTotalNumber)));
```