



UNIVERZITET U SARAJEVU  
ELEKTROTEHNIČKI FAKULTET  
ODSJEK ZA AUTOMATIKU I ELEKTRONIKU

---

## **Sistem mašinske vizije za automatsko rješavanje 2D puzzli**

---

SEMINARSKI RAD

PREDMET: UVOD U DIGITALNU OBRADU SIGNALA

**Autori:**  
**Isam Vrce (1997/18447)**  
**Armin Žunić (1936/18301)**

**Odgovorni profesor:**  
**Vanr.prof.dr Emir Sokić**

Sarajevo,  
septembar 2022.

# Sadržaj

<b>1</b>	<b>Uvod</b>	<b>1</b>
<b>2</b>	<b>Idealne puzzle</b>	<b>2</b>
2.1	Algoritam . . . . .	2
<b>3</b>	<b>Realne puzzle</b>	<b>7</b>
3.1	Akvizicija puzzli . . . . .	7
3.1.1	Odabir uređaja za akviziciju . . . . .	7
3.2	Izdvajanje kontura puzzli . . . . .	10
3.2.1	Binarizacija ulazne slike . . . . .	10
3.2.2	Pronalazak čoškova . . . . .	13
3.2.3	Pronalazak ivica puzzle . . . . .	16
3.3	Algoritam . . . . .	18
3.3.1	Template matching . . . . .	18
3.3.2	Izdvajanje potencijalnih puzzli . . . . .	19
<b>4</b>	<b>Formiranje konačne slike</b>	<b>22</b>
<b>5</b>	<b>Eksperimentalni rezultati</b>	<b>24</b>
<b>6</b>	<b>Zaključak</b>	<b>26</b>

# Uvod

Svi smo jednom u životu, bar kao djeca, slagali puzzle. Ova, "igra" koja bi trebala biti opuštajuća često može biti frustrirajuća. Često se desi da je potrebno mnogo vremena da pronađemo puzzlu koja se uklapa na pravo mjesto. Vođeni iskustvima iz djetinjstva, autori su odlučili da na neki način automatiziraju proces slaganja puzzli. Odlučili smo napraviti program koji će izvršavati proces slaganja puzzli. Odnosno program koji će "ubiti" zabavu ili možda "rodit" veću.

Kombinacije, mogućnosti, strategije, sve su ovo realni problemi s kojima smo se susreli slagajući puzzle. Okvir puzzle je možda lahko složiti, međutim, kako nastaviti dalje. Da li gledati boje, oblike, slike svake puzzle ili možda ipak samo konture puzzli. Ovo su samo neka od pitanja koja smo sebi postavili kao autori ovog rada. Dugo smo razmišljali kako početi, na čemu zasnovati temu, na koji način slagati puzzle. Došli smo do zaključka da bi najbolji način bio napraviti program koji će slagati puzzle baš onako kako to i čovjek radi. Pod ovim smo podrazumijevali nekoliko glavnih stavki, koje će kroz nekoliko narednih sekcija biti detaljno objašnjene.

Prvi zadatak je bio složiti puzzle koje su idealne, odnosno one generirane programski na računaru. Ovaj zadatak predstavlja uvod za slaganje realnih puzzli, koje su obrađene u narednom poglavlju. U ovom poglavlju će biti obrađeni i načini digitalizacije fizičkih puzzli tako da se iste mogu obrađivati u programu. Nakon dobijenog redoslijeda kojim slagalice trebaju biti poredane na slici, izvršen je i prikaz konačne složene puzzle. Pored ovoga prikazana su i vremena potreban da se slagalica složi do kraja.

U procesu kreiranja programa koji bi trebao složiti puzzlu nekih 100 do 200 puta brže od čovjeka, smo vjerovatno mogli složiti stotinu različitih puzzli. Međutim put koji nas je doveo do konačnog programa je bio zanimljiv kao i slaganje istih. Jer putovati je nekad bolje nego stići.

# Idealne puzzle

U ovom poglavlju ćemo pažnju posvetiti "idealnim" puzzlama. Pod idealnim puzzlama ćemo smatrati one koje su dobijene nekim online alatom za pravljenje puzzli. U takvom slučaju nijedna puzzla nije zašumljena, te se jasno razlikuje pozadina od bitnog dijela svake slike. Primjer takve puzzle je dat na slici (slika 2.1).



Slika 2.1: Prikaz idealne puzzle

Sljedeće što karakteriše ovakve puzzle, dobijene online putem, jeste što je gotovo svaka puzzla u potpunosti različita od drugih, i to sa svih strana. Ovo je vrlo bitno napomenuti, jer u mnogome pojednostavljuje algoritam za slaganje ovakvih puzzli. Ipak, povećanjem broja puzzli, vjerovatno bi i ovakav online alat generirao slične puzzle. U ovom dijelu smo slagali 70 puzzli, a razlog tome je što smo u ovom dijelu željeli pokazati jednostavnost algoritma ukoliko se radi o idealnom slučaju. U nastavku rada, kada budemo slagali realne puzzle to neće biti slučaj.

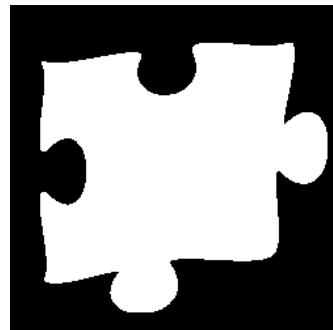
## 2.1 Algoritam

Zamisao algoritma jeste da se iz svake puzzle ekstrahuje nekolicina tačaka od značaja, te da se na osnovu njih formira matematički zakon koji će biti osnova nekog kriterija minimizacije. Ovo bi kao posljedicu trebalo dati tačno onu puzzlu koja se treba uklopiti u neku prije uklopljenu puzzlu. Traženje specifičnih tačaka na svakoj puzzli se bazira na nekoliko koraka koji će biti u nastavku obrađeni.

Nakon što su napravljeni snimci ekrana svake od puzzli zasebno, takve slike su snimljene u jedan folder i moguće je krenuti u njihovu dalju obradu. Na početku je za svaku sliku moguće odrediti masku, što u ovakvim "idealnim uslovima" neće biti nikakav problem. Jednostavnim izdvajanjem potpuno bijelih piksela od ostalih formira se jasna razlika između pozadine i puzzle na slici. Primer dobijene maske se nalazi na slici (slika 2.3) dok je originalna puzzla prikazana na slici (slika 2.2).

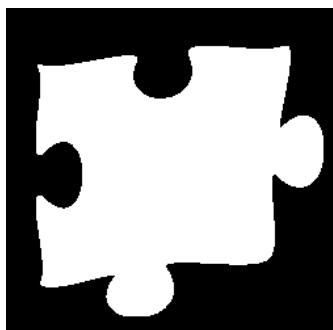


Slika 2.2: Orginalna slika jedne puzzle

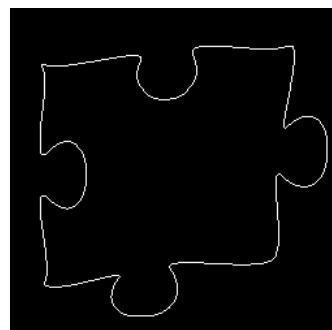


Slika 2.3: Maska prethodne orginalne slike

Nakon ovoga je moguće naći najveću konturu na slici, što će značiti izdvajanje vanjske konture puzzle. Ukoliko njenu debljinu postavimo na jedan piksel, dobićemo tačno vanjsku ivicu puzzle, što je prikazano na slici (slika 2.5).

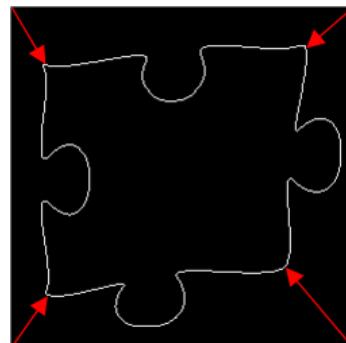


Slika 2.4: Maska (binarna slika) jedne puzzle



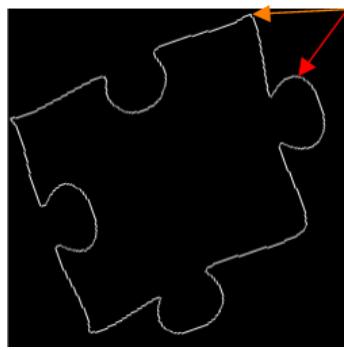
Slika 2.5: Izdvojena kontura jedne puzzle

Dalje bi bilo poželjno ispravno odrediti četiri čoška svake puzzle. Obzirom da su puzzle "idealne", to još podrazumijeva i da su one na pojedinačnim slikama rotirane tačno onako kako bi se trebale uklapati jedna u drugu. To dalje znači da bismo čoškove mogli odrediti na vrlo jednostavan način. Tačka čoška ima najkraću udaljenost od odgovarajućeg čoška slike, među svim tačkama konture puzzle. Slikoviti prikaz prethodno rečenog je dat ispod na slici (slika 2.6).



Slika 2.6: Prikaz udaljenosti čoškova slike od čoškova puzzli

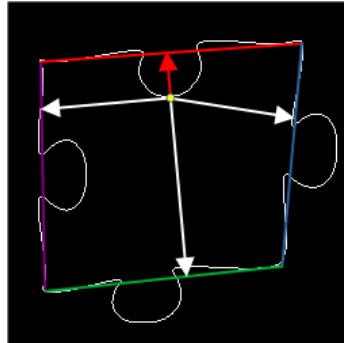
Međutim, da su kojim slučajem ispuštenja nekih puzzli bliže samim čoškovima puzzle ili da su puzzle malo zarotirane ili da su slike imalo zašumljene ovakav način ne bi davao očekivane rezultate, te su autori u slučaju realnih puzzli morali osmisliti nešto drugo. Ipak, prikaz ovog problema za slučaj idealnih puzzli je dat na slici (slika 2.7).



Slika 2.7: Prikaz problema određivanja čoška jedne puzzle

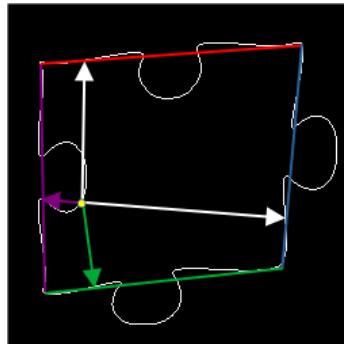
Kako god, za dva dataset-a nad kojima su autori testirali ovaj algoritam, određivanje čoškova sa opisanom procedurom nije stvaralo probleme, te se moglo krenuti u dalju izradu algoritma.

Dvije tačke po jednoj strani puzzle nisu bile dovoljne za uspješnost algoritma, te je zbog toga određena i treća specifična tačka po svakoj strani puzzle. Izbor je pao na tačku vrha svakog ispuštenja/udubljenja na puzzli. Međutim, da bi se odredile ove tačke bilo je potrebno znati koja tačka konture pripada kojoj strani puzzle. To se uradilo na veoma intuitivan način. Između svaka dva čoška je povučena duž i za svaku tačku konture se računala najkraća udaljenost do svake od četiri duži. Ondje gdje je ta udaljenost najmanja, ta tačka konture pripada strani puzzle između čija dva čoška je provučena duž. Jasnije objašnjenje je dato kroz sliku (slika 2.8).



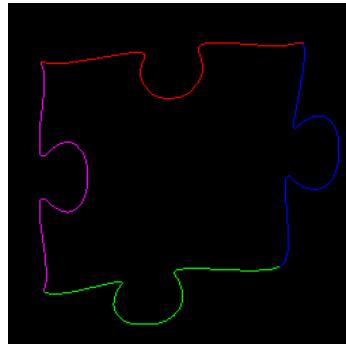
Slika 2.8: Određivanje pripadnosti tačke konture jednoj od četiri strane puzzle

Opet potencijalni problem, koji je riješen sa realnim puzzlama, je taj ako se udubljenje nalazi veoma blizu čoškova puzzle. U tom slučaju će ovakav način bar dio udubljenja prepisati konturi kojih u stvarnosti ne pripada. Također, prikaz ovog problema je dat na slici (slika 2.9).



Slika 2.9: Mogući problem određivanja pripadnosti tačke konture jednoj od četiri strane puzzle

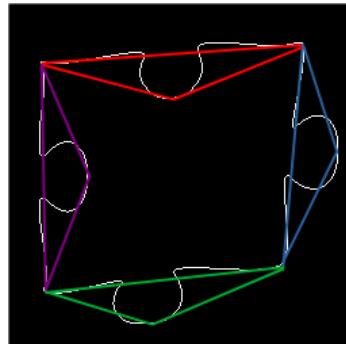
Bilo kako bilo, u slučaju "idealnih" puzzli ovo nije stvaralo nikakav problem, te se moglo nastaviti sa traženjem treće tačke svake strane puzzle. Rezultat pridruživanja svake tačke konture određenoj strani puzzle je prikazan na slici (slika 2.10).



Slika 2.10: Tačke konture u bojama zavisno od strane na kojoj se nalaze

Sa slike je jasno da je određivanje vrhova udubljenja/ispupčenja prilično jednostavno, jer za svaku stranu možemo jednostavno naći minimalnu/maksimalnu vrijednost x ili y koordinate od tačaka te strane, te jednostavno dobiti tražene vrhove. Uz ovaj dio, u posmatranom algoritmu se usput odredi gdje svaka puzzla ima udubljene, ispuštenje ili pravac. Vrijedi napomenuti da bi imalo zašumljena slika veoma uticala na određivanje ovih vrhova, te u konačnici algoritam ne bi radio kako treba.

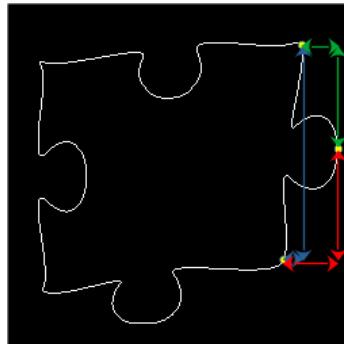
Nakon što za svaku puzzlu imamo 12 specifičnim tačaka, moguće je odrediti za svaku stranu puzzle njene specifične dužine. Po svakoj strani puzzle postoje 3 specifične dužine, a to su dužine između svake od 3 specifične tačke. Prikaz ovih dužina je dat na slici (slika 2.11).



Slika 2.11: Prikaz specifičnih dužina svake strane jedne puzzle

Puzzle koje se uklapaju jedna u drugu bi trebale imati iste ili približno iste sve tri dužine. Upravo je ovo kriterij kojim će se voditi algoritam, koji će za svaku puzzlu računati sve razlike dužina i nakon toga će susjednu izabrati na način da nađe minimalnu razliku ovih dužina za neku od puzzli.

Međutim, primjećeno je da algoritam sa samo ovakvim načinom ne daje očekivane rezultate, jer se čak i u "idealnim" uslovima dešavalo da puzzla koja ne treba biti na tom mjestu daje najbolju vrijednost kriterija. Pa nakon samo jedne pogrešno spojene puzzle, ostatak algoritma je beskoristan. Rješenje se moralo pronaći u nekom dodatnom poređenju, pored standardnih Euklidskih dužina. Zbog toga se moralo uvesti dodatno poređenje koordinata tačaka, gdje se mjerila samo njihova absolutna razlika po x i y osi. Grafički prikaz ove razlike je dat na slici (slika 2.12).



Slika 2.12: Prikaz dodatnih apsolutnih razlika na jednoj strani puzzle

Nakon uvođenja dodatnog kriterija slaganje puzzli je izvršeno uspješno na jednom dataset-u od 70 puzzli. Međutim, na drugom dataset-u od isto toliko puzzli se desilo da i ovaj dodatni kriterij nije mogao pomoći pri slaganju puzzli, te bi algoritam dao pogrešno rješenje. To pogrešno rješenje se ogledalo u tome da bi algoritam nakon složenih 30-tak puzzli došao do posljednje puzzle (npr. donji desni čošak), te nakon toga više teoretski nije mogao vezivati preostale puzzle. Upravo je ovo bila odlična indikacija da se desio problem, u slaganju. Ovo znači da algoritam treba krenuti slaganje puzzli ispočetka, uz napomenu da se neke puzzle (koje su u prethodnoj iteraciji dale veoma blisku vrijednost puzzlama koje su se tada uklapale) trebaju zamijeniti drugim. Ovo je na neki način bilo pamćenje puzzli iz prethodne iteracije, te se može reći da u ovom dijelu algoritam radi sa određenom memorijom. Poslije uvođenja ovog dodatnog koraka, algoritam je uspješno savladao i drugi dataset.

# Realne puzzle

Nakon idealnih, na red je došlo slaganje realnih puzzli. Svi problemi koji nisu postojali prilikom slike generirane online programom postoje prilikom pokušaja akvizicije realne puzzle. Pa je prvi problem odrediti način, odnosno sredstvo akvizicije.

## 3.1 Akvizicija puzzli

### 3.1.1 Odabir uređaja za akviziciju

#### Korištenje fotoaparata

Najbolji način akvizicije bio bi uzeti fotoaparat i fotografirati puzzle razbacane po radnoj podlozi koja može biti tkanina, puna neravnina i sličnih nesavršenosti, iz bilo kojeg ugla i udaljenosti od radne površine. Jasno je da ovakav način uzimanja uzoraka unosi mnogo promjenjivih, koje je gotovo nemoguće otkloniti. U nastavku su pobrojani neki od problema koji mogu nastati:

- Prvi, i najočigledniji je da sve puzzle trebaju biti "istih" dimenzija. Ukoliko se stranica puzzle treba uklopiti u drugu, na akviziranoj fotografiji one trebaju biti identičnih dimenzija.
- Problem pozadine: Ukoliko je pozadina neravna onda će puzzle biti zakrenute pod određenim uglom, pa se ponovno pojavljuje problem da puzzle trebaju biti istih dimenzija.
- Problem osvjetljenja: Potrebno je kreirati osvjetljenje koje će osigurati da su sve puzzle osvijetljene podjednako.
- Problem udaljenosti fotoaparata od podloge: Potrebno je na neki način osigurati istu udaljenost od podloge.
- Poznato je da fotoaparati prilikom fotografiranja uvode distorziju koja je sve očiglednija što su objekti na slici udaljeniji od centra. Ovaj problem je rješiv i moguće je različitim transformacijama umanjiti (u idealnim slučajevima u potpunosti otkloniti) efekte distorzije. Međutim, ovdje nije riječ o idealnim uslovima i efekti distorzije u konačnici mogu uticati na moguće uklapanja dvije puzzle jedne u drugu.
- Veći problem od distorzije predstavlja broj puzzli koji se mogu obuhvatiti na jednoj fotografiji. Sa povećanjem broja puzzli na jednoj slici neujednačenost uslova osvjetljenja postaje sve izraženija. Puzzle koje su na samim krajevima su manje osvijetljene od onih koje se nalaze u sredini fotografije.
- Količina memorije potreba da se pohrane sve puzzle: Fotografije dobijene fotoaparatom za pohranu zahtijevaju relativno puno memorije.
- Svi prije navedeni problemi, su do neke mjere rješivi, ali jedan ostaje nerješiv, a to je činjenica da su puzzle trodimenzionalni objekti. Ovaj problem nije primjetan na puzzlama

koje se nalaze u sredini fotografije. Međutim, puzzle koje se nalaze na ivicama fotografije pokazuju svoju osobinu trodimenzionalnosti i gotovo je nemoguće odrediti tačne konture puzzle. Ovo je u direktnoj vezi sa problemom broja puzzli, jer je onda potrebno smanjiti broj puzzli na samo nekoliko, ali i sa nekoliko puzzli ovaj problem ostaje prisutan.



Slika 3.1: Akvizicija fotoaparatom

Da bi prije pobrojani problemi bili jasniji data je slika (slika 3.1). Moguće je osigurati donekle iste uvjete fotografiranja, te se može smatrati da je udaljenost od podloge ista za sve slike. Međutim koliko god se trudili nismo bili u mogućnosti osigurati ujednačeno osvjetljenje na cijeloj slici, pa se tako može vidjeti da su slagalice gore-ljevo osvjetljene od onih dole-desno. Također pozadina nije "isto bijela" na cijeloj fotografiji. Ovo možda ne predstavlja problem za izdvajanje konture ali može predstavljati problem sa pogleda kasnijeg eventualnog slaganja puzzli.

Problem broja puzzli je očigledan. Na slici se nalazi samo 9 puzzli (koje su istini za volju mogli biti gušće raspoređene). Pa tako ako bi se na slici nalazilo po 15 puzzli bilo bi potrebno 11 fotografija da bi se akviziralo 160 puzzli. Obzirom na činjenicu da konkretna fotografija "zauzima" 3.9 MB memorije. Zaključuje se da bi bilo potrebno oko 40MB memorije samo da bi se pohranile sve puzzle. Iako su ovdje pobrojani svi problemi koji nastaju, autori se nisu zadržavali na rješavanju istih. jer problem koji nastaje zbog trodimenzionalnosti puzzli nije rješiv. Na slici (slika 3.2) je izdvojena gornja-ljeva slagalica (onako kako je program izdvaja sa slike, vidimo da je rotirana u odnosu na stvarnu):



Slika 3.2: Akvizicija fotoaparatom, problem trodimenzionalnosti puzzle

Na slici (slika 3.2) se jasno vidi da je izdvajanje stvarne puzzle gotovo nemoguće. Žutom linijom je prikazana stvarna kontura puzzle, kontura koja bi se trebala izdvojiti. Crvenom isprekidanom linijom je prikazana kontura koja se zapravo izdvoji. Jasno je da ove dvije konture odstupaju jedna od druge i da trodimenzionalnost ima veliki uticaj na konačni ishod u ovom slučaju, ali i u slučaju svih puzzli koje se nalaze dalje od centra fotografije.

Problem je rješiv na način da se fotografira puzzle po puzzla i da se pri tome koristi telecentrično sočivo. Korištenjem telecentričnog sočiva dobija se efekat fotografiranja puzzle "iz beskonačnosti". Na ovaj način se problemi trodimenzionalnosti gotovo u potpunosti otklanaju jer se puzzle na slici vidi kao 2D lik, a ne kao 3D objekat. Na slici (slika 3.2) bi i dalje ostao problem sjene, ali taj se problem može riješiti boljim osvjetljenjem. Međutim, rješavanjem jednog problema dobili smo dva nova. Potrebno je fotografirati puzzlu po puzzlu, što oduzima mnogo vremena. Pored toga je potrebno kod sebe imati fotoaparat i telecentrično sočivo da bi se fotografirale puzzle. Autori nisu bili zadovoljni ovim rješenjem te su pokušali naći neko "bolje".

### Korištenje skenera

Izbor je pao na korištenje skenera. Skener rješava gotovo sve pobrojane probleme, a pristupačan je širim narodnim masama. Problem osvjetljenja su riješili proizvođači skenera jer je svaki dio slike jednakos osvijetljen. Problem istih dimenzija i jednakih udaljenosti je riješen jer se puzzle postavljaju na staklenu površinu koja je uvijek jednakod udaljena od senzora koji akviziraju sliku. Problem trodimenzionalnosti je riješen zbog načina rada samog skenera koji se kreće po jednoj od osa da bi akvizirao sliku. Te konačno korištenjem skenera je moguće "posložiti" zaista mnogo slagalica na radnu površinu i skenirati ih odjednom, te nije potrebno fotografirati puzzlu po puzzlu. Pored svega navedenog problem memorije je lako rješiv, jer se samo u nekoliko klikova može podesiti rezolucija uzorkovane slike koja je u direktnoj vezi sa zauzećem memorije.



Slika 3.3: Akvizirana slika korištenjem skenera sa crnom pozadinom

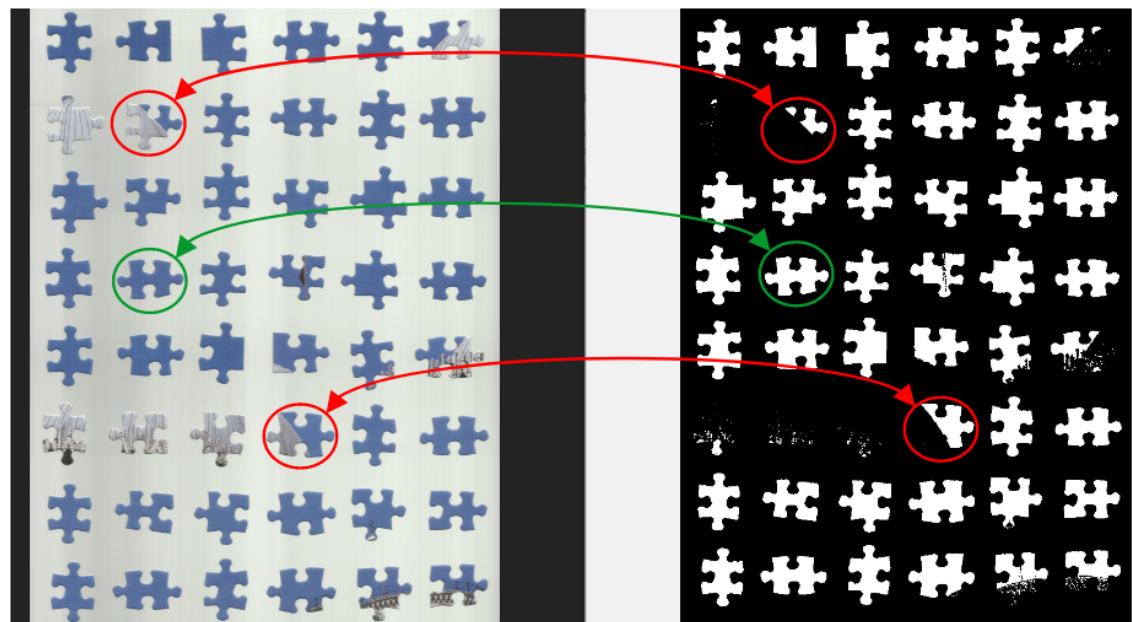
Slika (slika 3.3) pokazuje da je svaka puzzla zaista dvodimenzionalna, te da su svi prije pomenuti problemi riješeni. Na slici se nalazi 35 puzzli, a slika zauzima 465 KB memorije. Ovo je veliko unaprijeđenje u odnosu na akviziciju korištenjem fotoaparata gdje slika sa 15-tak puzzli zauzima 3900 KB memorije. Rezolucija akvizirane slike je: 2338 x 1700 pixela. Odnosno korišena je rezolucija od 200 Dots Per Inch (DPI) na skeneru. Međutim kao što je napomenuto ranije samo u par klikova se može dobiti skenirana slika druge rezolucije. Ukoliko se ista slika skenira rezolucijom 75 DPI odnosno 637 x 876 piksela, onda ista zauzima samo 78.5 KB memorije. Za savremene računare memorija naizgled ne predstavlja velik problem. Smatrali smo da je svejedno da li je ulazna slika teška 0.5 MB, 1 MB ili 10 MB. Međutim, ubrzo smo shvatili zašto slike koje zauzimaju mnogo prostora predstavljaju problem. Bilo je potrebno pronaći adekvatnu rezoluciju kojom se akvizira slika. Veća rezolucija povećava preciznost akvizicije, posljeđično zauzima više memorije na računaru. Nakon što je odabran uređaj za akviziciju potrebno je osmisliti način za dobijanje kontura sa slike.

## 3.2 Izdvajanje kontura puzzli

Izdvajanje kontura realnih puzzli je složeniji problem od onog u slučaju "idealnih" puzzli. Čitav proces dobijanja konture se može podijeliti u nekoliko faza, pri čemu će se u nastavku svakoj zasebno posvetiti pažnja.

### 3.2.1 Binarizacija ulazne slike

Binarizacija idealnih puzzli je bila jednostavan zadatak. Međutim, kod realnih puzzli imamo dosta suptilnih problema koje je potrebno riješiti. Prvi od njih je što pikseli na skeniranoj slici više nisu potpuno bijeli. Takvi pikseli gotovo da ne postoje. Odnosno mora se uvesti opseg vrijednosti koji odgovara bijelim pikselima. Uvođenjem opsega vrijednosti postoji mogućnost da se veliki broj piksela koji su pozadina detektiraju kao dio puzzle ili obratno, kao što je to slučaj na slici (slika 3.4).

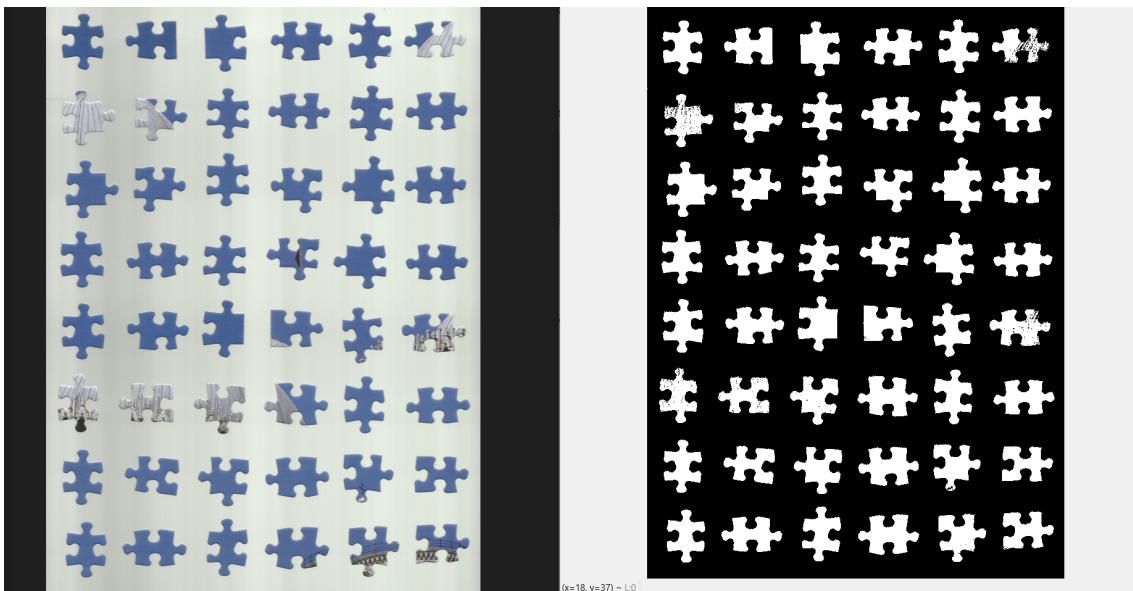


Slika 3.4: Problem binarizacije slike u slučaju realnih puzli

Na slici (slika 3.4) su vidljive skenirane puzzle, ali i binarizirana slika koja je dobijena tako što je orginalna slika konvertirana u HSV prostor boja i u tom prostoru boja je odabran opseg vrijednosti koji odgovara pozadini. Granica je postavljena provjerom većeg broja pozadinskih piksela.

Ali kao što je prije naglašeno problem predstavljaju puzzle čiji je sadržaj iste boje kao pozadina, označeno crvenom bojom na slici (slika 3.4). Vidljivo je da su bijeli dijelovi potpuno "odsječeni" od puzzle. Naravno da su ovi podaci neupotrebljivi i pristup je potrebno odbaciti. Međutim, puzzle na kojima nema bijele boje su zaista dobro binarizirane. Primjer je puzzla označena zelenom bojom na istoj slici.

Zbog prije navedenih problema, bilo je potrebno osmisliti drugi način binarizacije ulazne slike. Jedan od prijedloga bio je transformirati ulaznu skeniranu sliku u neki drugi prostor boja, i tu tražiti karakteristike koje će izdvojiti puzzlu od pozadine. Autori su testirali gotovo **sve** transformacije iz BGR prostora boja koje su podržane u OpenCV-u. Neke od njih, kao što su transformacija iz BGR u XYZ prostor boja te transformacija iz BGR u YCrCb prostor boja, su dale dobre rezultate. Rezultati su dati na slici (slika 3.5).



Slika 3.5: Binarizacija slike u slučaju realnih puzzli transformacijom iz BGR u YCrCb prostor boja

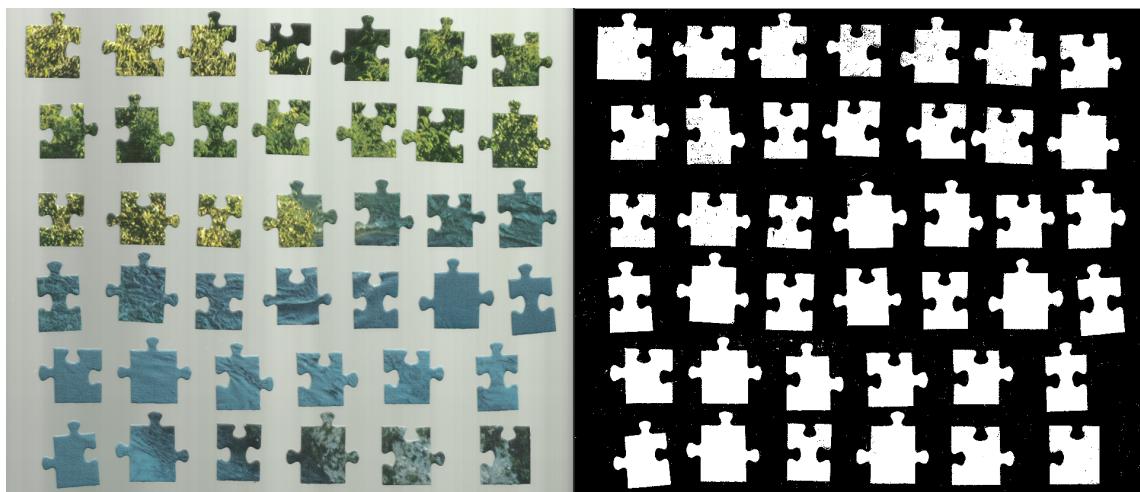
Na slici (slika 3.5) primjećuje se da je problem sa postojanjem bijele boje na slagalici riješen. Međutim, sada problem predstavlja oština ivica, odnosno njen nedostatak. Ivice nisu oštare nego su zašumljene što bi predstavljalo problem prilikom slaganja puzzle, te je i ovaj pristup odbačen.

U potrazi za boljim rješenjem autori su se našli u slijepoj ulici. Tako pokušavajući naći rješenje jedan od autora počeo je zumirati puzzle skenirane rezolucijom od 1200 DPI, što je za dimenzije prostora za skeniranje veličine a4 papira ekvivalentno rezoluciji od 10200 x 14028 piksela, što je apsolutni maksimum koji komercijalni, lako dostupni skeneri mogu podržati. Na ovaj način slika veličine a4 formata teži nevjerovatnih 17 MB. Kasnije će se pokazati da je ova rezolucija bespotrebna, ali u potrazi za rješenjem nekada je potrebno iscrpiti sve izvore da bi se rješenje pronašlo. Put ka rješenju nije nužno onaj najbrži, ali rješenje uvijek postoji, samo ga je potrebno pronaći. Nakon što se dovoljno zumira slika sa ovako mnogo detalja (slika 3.6) mogu se primijetiti "kružići" koji nastaju prilikom printanja samih puzzli.



Slika 3.6: Puzzla skenirala rezolucijom 1200 DPI

Pozadina nema mnogo detalja bez obzira na rezoluciju kojom skeniramo slagalice. Ova činjenica rezultira time da je pronađak dijelova skenirane slike sa puno detalja ekvivalentan izdvajaju puzzle od pozadine. Da bi se detektirali detalji na slici, koriste se algoritmi za detekciju ivica (Canny i Laplacian). Na slici (slika 3.7) se mogu vidjeti rezultati:



Slika 3.7: Binarizacija slike skenirane velikom rezolucijom korištenjem Laplacian-a

Ovako binarizirana slika je korištena za izdvajanje kontura, te kasnije za slaganje puzzli. Zbog velike rezolucije izdvajanje binariziranih puzzli tako da se na jednoj slici nalazi jedna puzzla traje dugo vremena. Pa tako za 40 puzzli na slici (slika 3.7) postupak traje u prosjeku 7.7s (izvršavanje programa na procesoru "Ryzen 5 5500U"). Odnosno za izdvajanje binarne slike za jednu puzzlu je potrebno u prosjeku 0.1925s. Nakon prvih testiranja algoritma za slaganje (koji će biti pojašnjen u poglavljima u nastavku) zaključeno je da je slaganje nedovoljno brzo. Te je i ovaj pristup privremeno odbačen. Međutim, autori su znali da se mogu vratiti ovom pristupu ukoliko za to bude potrebe. Nastavljeno je sa potragom za boljim načinom binarizacije koji bi omogućio binarizaciju slike manje rezolucije (75 DPI, 100 DPI, 200 DPI, ...).

Obazrivi čitaoci su do sada mogli primijetiti da su neke skenirane slike imale crnu a neke bijelu pozadinu. Pa tako skenirana slika (slika 3.3) ima crnu pozadinu, a skenirana slika (slika 3.5) ima bijelu pozadinu. Razlog se objašnjava činjenicom da se prilikom skeniranja puzzli **poklopac skenera** drži otvorenim. Pa ukoliko je prostorija u kojoj se vrši skeniranje tamna, pozadina će biti crna, a ukoliko je pozadina svijetla, pozadina će biti bijela. Odavdje se može zaključiti da je

moguće mijenjati boju pozadine. Ova činjenica je omogućila da se na jednostavan način mogu binarizirati slagalice na slici, jer je moguće postaviti boju pozadine na proizvoljnu (kasnije će u tekstu biti govora na koji način je ovo moguće uraditi). Pošto se može postaviti proizvoljna boja pozadine, moguće je postaviti baš onu boju pozadine koje nema na puzzlama. Ovo radimo da izbjegnemo problem kao na slici (slika 3.4). Te ukoliko se ovo postigne izdvajanje puzzli možemo raditi na način sličan kao sa "idealnim" puzzlama (uz postavljanje opsega vrijednosti za pozadinu), ali bez bojazni da će se na slagalicu pojaviti boja kao na pozadini.

Način na koji se osiguravamo da smo odabrali boju pozadine koja je nepostojeća na slagalicu jeste da na skeniranoj slici sa proizvoljnom bojom pozadine (najbolje crne) odredimo najmanji broj piksela određene boje u nekom prostoru boja (najbolje HSV). Pa tako ukoliko je to crvena boja, onda kao pozadinu postavimo crvenu boju, ako je to ljubičasta, onda kao pozadinu postavimo ljubičastu.

Da bi se omogućilo proizvoljno postavljanje boje pozadine, koristi se RBG LED traka koja se zalijepiti na reflektirajuću pozadinu dimenzija a4 formata (jer je te dimenzije korišteni skener), najbolje tanki limeni okvir. Zatim je potrebno osigurati da svjetlost koju odašilju RBG LED diode bude ravnomjerno raspršena. Ovo se radi da skenirana slika ne bi bila "istačkana" na mjestima gdje se nalaze LED diode. Nije potrebno uvijek izmišljati "toplju vodu". Moguće je koristiti gotovo rješenje koje provjerovalo radi. Rješenje na koje se misli jeste pozadinsko osvjetljenje televizora. Srećom imali smo jedan pokvaren LCD televizor koji smo rastavili i iskoristili nekoliko slojeva tankih plastičnih listova koji za cilj upravo imaju raspršiti pozadinsku svjetlost LED dioda u televizoru. Ovdje je važno napomenuti da autori nisu imali dovoljno vremena da naprave ovaj hardeverki uređaj, iako smo rastavili televizor i izdvojili 3 komponente potrebne da se rasprši svjetlost. Ovo ostaje nekome ko bi nastavio unaprijeđivati postojeće algoritme. Zato smo kao puzzle nad kojim ćemo testirati algoritam izdvajanja kontura, izdvajanje čoškova, te u konačnici slaganja svih puzzli, odabrali one koje nemaju crne boje na sebi. A kao pozadinu smo odabrali odabrali crnu pozadinu. Upravo su to puzzle sa slike (slika 3.3). Rezolucija skenera je postavljena na 200 DPI.

Nekoliko paragrafa iznad pokazano je da je potrebno 0.1925s da bi se iz slike sa svim puzzlama izdvojila maska tako da se na jednoj slici nalazi maska jedne puzzle (slika skenirana rezolucijom 1200 DPI). Ukoliko je slika skenirana rezolucijom 200 DPI i korištenjem pristupa koji je posljednji pokazan ovo vrijeme je sniženo na 0.0041s u istim uslovima i izvršavanju na laptopu sa procesorom Ryzen 5 5500U. Ovo je poboljšanje od oko **47 puta**, samo na izdvajanje pojedinačnih puzzli. Upravo je ovo razlog zašto je korištenje slike visoke rezolucije odbačeno.

### 3.2.2 Pronalazak čoškova

Da bi puzzle bilo moguće složiti, potrebno je imati ivice, a da bismo dobili ivice, potrebno je imati krajnje tačke svake ivice, odnosno četiri čoška svake puzzle. Određivanje čoškova predstavlja veliki zadatak, posebno zbog činjenice da puzzle na skeniranoj slici mogu biti proizvoljno rotirane. Ali prije svega potrebno je izdvojiti sve konture puzzli na skeniranoj slici, a za to koristimo binariziranu sliku.

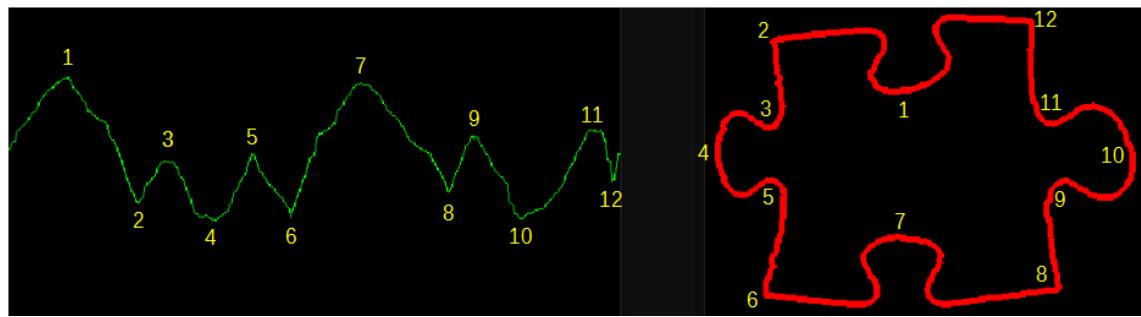
#### Pronalazak konutra

Da bi se pronašle konture na skeniranoj slici koristi se funkcija iz OpenCV biblioteke "findContours". Ukoliko se ovoj funkciji kao treći parametar proslijedi "*RETR\_TREE*", kao rezultat se dobijaju sve konture koje algoritam pronađe. Ovdje spadaju i različita oštećenja, prašina te eventualne greške prilikom binarizacije slike. A da bismo se osigurali da je pronađena kontura zaista puzzle, uvodimo ograničenje da obim te puzzle mora biti u nekom opsegu. Ovdje se može uvesti

dio koji će u ovisnosti od rezolucije podešavati koji je obim konture pogodan da bude puzzle.

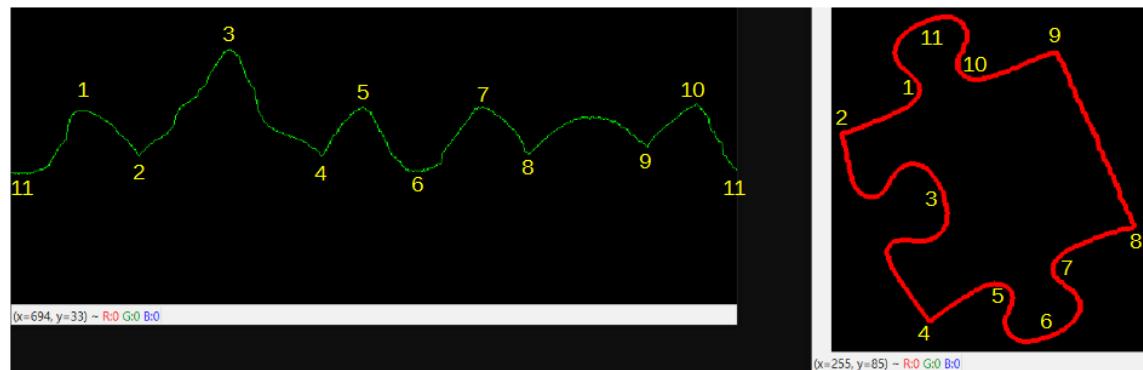
Nakon pronađenih kontura, i dalje ostaje zadatak pronaći čoškove puzzle. Ovdje je pravo vrijeme za naglasiti da ćemo se u ostatku rada na svakom koraku boriti sa problemom proizvoljne **rotacije puzzli** i pronalaskom algoritama koji će biti invarijantni na rotaciju. Prvi problem je odrediti referentnu tačku u odnosu na koju će se gledati kontura, jer rezultat koji vratí funkcija "findContours", je samo niz tačaka. Bilo je potrebno pronaći neku tačku koja je invarijantna na rotaciju puzzle. Kada se ovako stvari postave, jasno je da je ta tačka centar puzzle. Nakon pronalaska središnje tačke (na način da se pronađe pravougaonik koji opisuje slagalicu, te se onda pronađe težiste tog pravougaonika) moguće je izračunati udaljenosti svake tačke na konturi od središnje.

Prethodno rečeno se može vidjeti na slici (slika 3.8), gdje je sa lijeve strane grafik udaljenosti tačaka konture od centra, a sa lijeve strane sama kontura. Dodatno žutim brojevima su označene tačke na konturi koje odgovaraju tačkama na grafiku udaljenosti. Bitno je još za napomenuti da je koordinatni početak tačka gore-lijevo na grafiku udaljenosti.



Slika 3.8: Udaljenost tačaka konture od centra puzzle

Ono što se može primjetiti na slici (slika 3.8) jeste da su čoškovi oni dijelovi grafika koji su lokalni minimumi, a u svojoj okolini imaju brzu promjenu prvog izvoda. Odnosno udaljenost se povećava do neke tačke, pa se naglo počne smanjivati. Tačka gdje se nalazi ovaj skok je čošak slagalice. Ovo zvuči jednostavno i intuitivno, i jasno je zašto je invarijantno na rotaciju. Primjer je slika (slika 3.9).



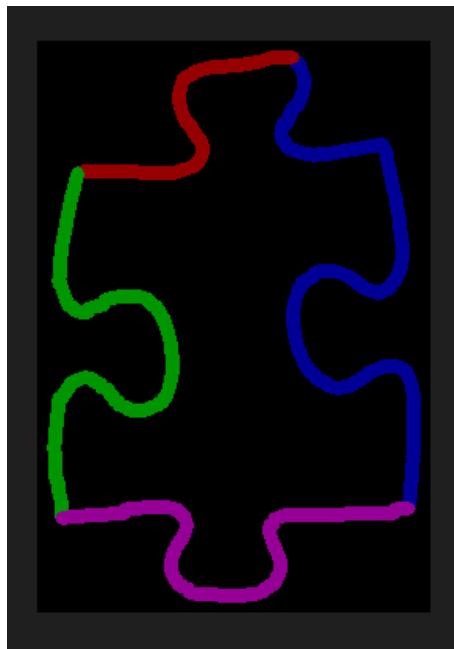
Slika 3.9: Udaljenost tačaka konture od centra rotirane puzzle

### Algoritam za pronalazak čoškova

Izazovno je napisati kôd koji će uspješno pronaći sve čoškove. Postoje puzzle za koje autori nisu mogli ispravno odrediti čoškove sa grafika udaljenosti a program je ipak uspješno radio. Ukratko će biti opisan postupak određivanja čoška:

- provjeriti da li je tačka lokalni minimum, zbog postojanja šuma mnogo tačaka su lokalni minimumi. Pokušaj zaglađivanja zašumljenog signala, dovodi do toga se se izglađuju i čoškovi što dodatno otežava njihovo određivanje. Neka se u nastavku razmatranja tačke lokalnog minimuma označene sa "A".
- uzeti nekoliko simetrično raspodjeljenih tačaka od tačke "A". Broj i udaljenost ovih tačaka od tačke "A" je promjenjiv. Neka su u nastavku ovo tačke B1L, B1D, B2L, B2D,... (oznaka "L" odgovara da se ta tačka nalazi lijevo od tačke "A", analogno za "D")
- sabratи sve relativne udaljenosti tačaka B1L, B1D, B2L, B2D,... od centra puzzle i tačke A od centra puzzle. Ovdje je moguće uvesti neke težinske faktore. Tačka koja je udaljena 10 piksela od čoška nema isti uticaj kao ona koja je udaljena 3 piksela od čoška. Težinski faktori su još jedna promjenjiva koja se podešavala u programu da se postigne što je moguće veća tačnost.
- odabrati 4 najbolja kandidata, ti kandidati predstavljaju čoškove
- dodatno je čoškove potrebno sortirati na način da se zna koji čošak odgovara gornjem-desnom, koji gornjem-ljevom, i tako dalje, jer je ovo od krucijalnog značaja za pravilno određivanje kontura koje slijedi.

Ipak uspješnost pronašlaska čoškova nije 100 %. Nad testnim skupom od 160 slagalica nad kojim se vršilo slaganje, pogrešno su određena tri čoška, što daje preciznost od  $3/(160*4) = 99.53\%$ . Primjer čoška koji nije dobro određen je dat na slici (slika 3.10).

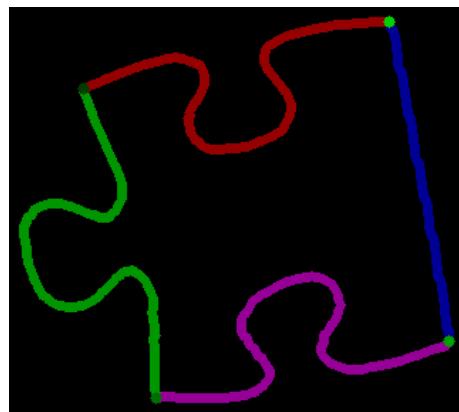


Slika 3.10: Primjer puzzle na kojoj čoškovi nisu dobro određeni

Na slici (slika 3.10), kontura je obojena, što je naredni dio o kojem će biti govora, ali na granicama konture se nalaze čoškovi. Primjeti se da jedan od čoškova nije dobro određen, ali tačka za koju program smatra da je čošak, zaista "liči" na čošak. Imamo gornji dio koji je prilično ravan, a zatim naglu promjenu pravca pod uglom koji je blizak ugлу od 90 stepeni. Čoškovi koji nisu dobro određeni su hardkodirani. I to su uradili autori naknadno kada je otkriveno da je neki čošak pogrešno određen. Također, nekada se desi da čošak nije tačno na čošku nego je nekoliko piksela pomjeren u stranu. Ovo nije predstavlja problem što se tiče slaganja puzzli. Predstavljalo je mali problem kod vizualizacije, ali o tom nešto više riječi na samom kraju rada.

### 3.2.3 Pronalazak ivica puzzle

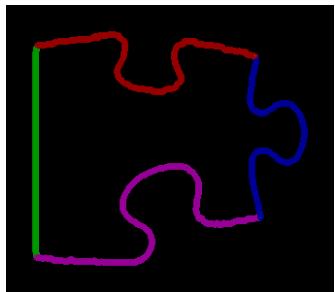
Nakon što su čoškovi pronađeni i sortirani u smjeru suprotnom kazaljci na satu počevši od gornjeg-ljevog čoška, konture se pronalaze na način da se sve tačke koje se nalaze između dva čoška oboje nekom unaprijed zadanim bojom za tu stranicu puzzle. Pa se tako tačke (odnosno duži između svih tačaka, koje se nalaze između gornje-ljeve i donje-ljeve tačke) boje zelenom bojom, itd. Raspored boja je vidljiv na slici (slika 3.11). Dodatno se na konturama iscrtava i tačka koja odgovara čošku. Ovo se radi da bi kasnije bilo lako pronaći čošak. Vjerovatno bi bilo lakše da se svi podaci spreme u variable (kasnije u programu je napravljena velika struktura podataka koja sadrži sve ove podatke), ali pošto je program pisan postepeno i nadograđivao se, odlučeno je da se svaki korak spasi da se može iskoristiti za ubuduće. Svi čoškovi su zelene boje, uvijek je gornji-ljevi čošak najtamniji i u smjeru suprotnom kazaljci na satu se povećava svjetlina zelene boje. Na slici (slika 3.11) su krugovi većeg prečnika nego na puzzlama koje se slažu (tamo su prečnika 1 piksel), ali je ovdje ostavljen veći prečnik da se bolje vide čoškovi.



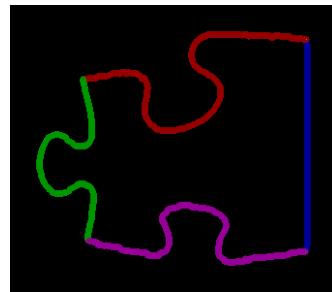
Slika 3.11: Primjer puzzle na kojoj su određeni čoškovi i ivice

Još jedna **važna napomena**: Napisan je kôd koji će rotirane slagalice kao na slici (slika 3.11) rotirati tako da budu približno "vodoravne". Ali taj kôd nije implementiran u konačni program, ovo ostaje dodati nekome ko bude unaprijeđivao postojeće algoritme.

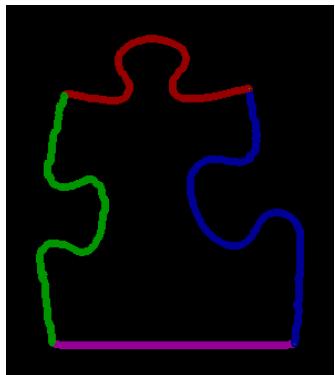
Nakon što je program uspješno izdvojio pojedinačne puzzle, te uspješno pronašao čoškove i formirao obojene konture po svakoj strani puzzle, može se krenuti u njihovu dalju obradu. Kako smo na početku istakli, orientacija pojedinih puzzli na skeniranoj slici je proizvoljna, pa da bi algoritam pri provjeri uklapanja uspješno provjerio svaku stranu puzzle, najjednostavnije je od jedne puzzle dobiti četiri, tako da algoritam umjesto  $N$  puzzli posmatra  $4*N$  puzzli, gdje je broj  $N$  u slučaju puzzli obrađenih u ovom radu jednak 160. Potrebno je voditi računa da se svakom rotacijom rotiraju i označeni čoškovi, kao i obojene strane puzzle. Ovo nije poželjno, te je ovo potrebno ispraviti. Poseban dio kôda se bavi tom tematikom, a kao rezultat od jedne puzzle dobijamo njih četiri rotirane po 90 stepeni, uz očuvanje oznaka čoškova i boje strana puzzli. Primjer takve četiri puzzle je dat na četiri slike ispod (slike 3.12, 3.13, 3.14, 3.15).



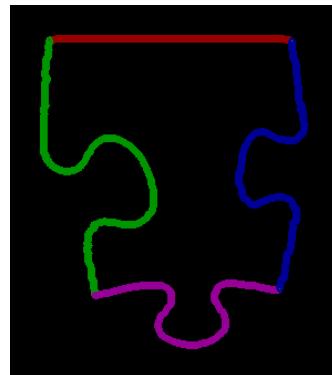
Slika 3.12: Kontura orginalne izdvojene puzzle



Slika 3.13: Kontura puzzle rotirane za  $180^\circ$



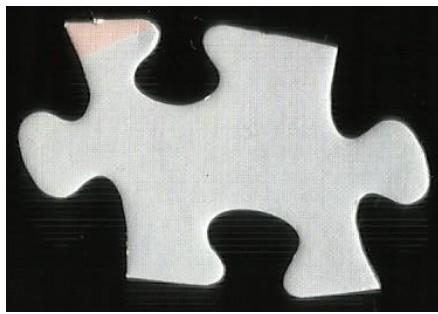
Slika 3.14: Kontura puzzle rotirane za  $90^\circ$



Slika 3.15: Kontura puzzle rotirane za  $270^\circ$

Nakon ovoga je za svaku od  $4^*N$  puzzli moguće odrediti, po tri specifične dužine po svakoj strani puzzle (kao i kod "idealnih" puzzli). Prije ovoga se određuju vršne tačke, ispupčenja/udubljenja svake puzzle, kako bismo za svaku puzzlu imali 12 specifičnih tačaka.

U slučaju idealnih puzzli se posmatrala i apsolutna razlika  $x$  i  $y$  koordinata specifičnih tačaka, što u realnom slučaju nema smisla. Obzirom da je svaka puzzla pri skeniranju proizvoljno zarotirana, ove razlike ne bi davale apsolutno nikakvu korisnu informaciju. Primjer puzzli koje se uklapaju jedna u drugu, a nisu isto zarotirane su date ispod na slikama (slika 3.16 i 3.17). Desnu puzzlu je potrebno zarotirati u smijeru suprotnom kazaljci na satu da bi se ista mogla uklopliti u lijevu puzzlu.



Slika 3.16: Puzzla sa lijeve strane



Slika 3.17: Puzzla sa desne strane

Pored posmatranih dužina, od krucijalne važnosti za realne puzzle će biti template matching između pojedinih strana puzzli. Zbog toga je potrebno iz svake puzzle ekstraktovati četiri ivice od značaja. Upravo zbog ovog su se strane kontura bojile u različite boje. Primjer izdvajanja četiri konture iz jedne puzzle je dat na slikama (slike 3.18, 3.19, 3.20, 3.21).



Slika 3.18: Izdvojena gornja ivica puzzle 3.16 Slika 3.19: Izdvojena donja ivica puzzle 3.16



Slika 3.20: Izdvojena lijeva ivica puzzle 3.16 Slika 3.21: Izdvojena desna ivica puzzle 3.16

### 3.3 Algoritam

Na početku algoritma se određuje početna puzzla od koje počinje glavni dio algoritma (slaganje puzzle). Ova puzzla mora zadovoljavati osobinu da su njene dvije susjedne strane ravne (bez ispupčenja/udubljenja), jer se takva puzzla sigurno nalazi u jednom od čoškova konačne slike puzzle. Nije bitno o kojem čošku je riječ jer algoritam će raditi uspješno bilo da slaganje puzzli radi odozdo ili odozgo. Bitno je napomenuti da trenutna verzija algoritma treba poznavati veličinu matrice konačne puzzle (broj redova i kolona), te da prva puzzla koju smo već spomenuli pri skeniranju treba biti okrenuta tačno onako kako se nalazi na konačnoj slici, jer bi se u suprotnom moglo desiti da broj redova konačne slike odgovara broju kolona i obratno. Izbjegavanje ovih stvari bi zahtijevalo nekoliko izmjena u postojećem kôdu, što može biti zadatak za naredne verzije istog.

Nakon što smo pronašli početnu puzzlu, nju dalje nećemo uzimati u obzir, tj. svaku narednu puzzlu koju pronađemo, odmah izbacujemo iz daljeg razmatranja. Nije moguće imati jednu te istu puzzlu na više od jednog mjestu u konačnoj matrici. Dalje se nastavlja sa slaganjem puzzli i to na način da se slaže cijeli prvi ili posljednji red puzzle, zavisno od toga koja puzzla je prva pronađena. U slučaju slaganja prvog, odnosno posljednjeg reda ili kolone u obzir dolaze samo puzzle koje imaju jednu ravnu ivicu, čime se veliki broj puzzli izbacuje iz promatranja.

#### 3.3.1 Template matching

Traženje ispravnih puzzli u prvom ili posljednjem redu se vrši samo poređenjem svih "mogućih ivica" već pomenutim template matchingom. "Moguće ivice" su one koje se mogu teoretski uklopiti jedna u drugu, a to je slučaj kada jedna ivica ima udubljenje, a druga ispupčenje ili obratno. Nema smisla provjeravati druge slučajeve, jer bi to dovelo do bespotrebnog gubljenja vremena.

U samom template matchingu jedna ivica je stacionarna, dok se druga ivica rotira za određeni ugao u razmaku od -10 do 10 stepeni. Ovo radimo jer akvizirane puzzle nisu tačno rotirane tako da se mogu uklopiti jedna u drugu. Da to možemo osigurati, slaganje bi bilo trivijalno. Korak kojim se mijenja ovaj ugao je veoma bitan, jer bi mali korak značio mnogo bespotrebnih "template matching-a" odnosno dosta uzaludno potrošenog vremena, dok bi preveliki korak mogao dati pogrešne rezultate, jer bismo mogli preskočiti idealno (najbolje) poklapanje dvije ivice puzzli. Zbog toga je zadani korak izabran na 1 stepen, dok u samoj petlji imamo određeni broj uslova koji uko-

liko su ispunjeni na neki način manipulišu ovim korakom ili čak u potpunosti zaustavljaju daljnje template poređenje. Taksativno pobrojani uslovi su dati u nastavku: ("koeficijent" predstavlja rezultat "template matching-a" dvije strane puzzle koje se upoređuju i on bi trebao biti veći ukoliko je poklapanje veće i obratno; njegova početna vrijednost je 0 i kreće se u opsegu od 0 do 1)

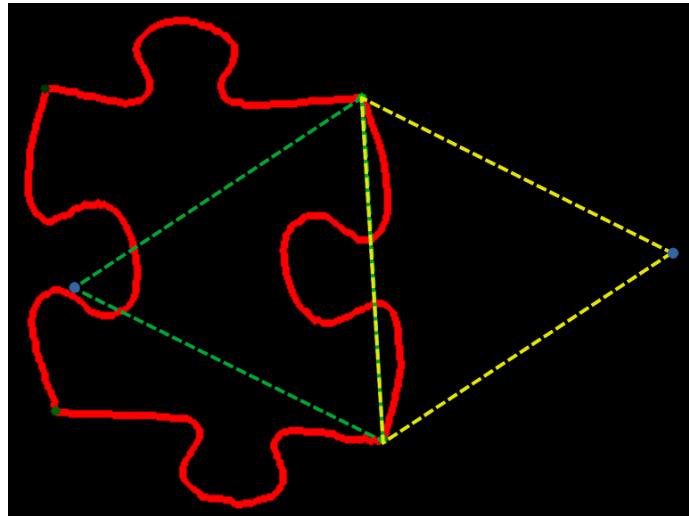
- Ukoliko je trenutni koeficijent veći od prethodnog (znači da se radi o boljem poklapanju) i u varijablu zapisujemo trenutni koeficijent
- Ukoliko je koeficijent poklapanja prešao vrijednost 0.75, tada je riječ o veoma dobrom poklapanju, te kada koeficijent dostigne svoj maksimum iznad vrijednosti 0.75, tada smo sigurni da se on poslije neće i dalje povećavati i poređenje se tu može zaustaviti
- Ukoliko je koeficijent manji od 0.4 odmah povećamo korak za 3 stepena. Kao što smo već rekli, koeficijent veći od 0.75 znači dobro poklapanje. Međutim, da bismo ga dostigli bilo bi potrebno rotirati puzzlu nekoliko stepeni. Stoga smo odmah povećali korak da bismo uštedili na vremenu izvršenja. Ukoliko će poklapanje biti loše, onda nam nije nikakav problem i preskočiti neko bolje poklapanje u svrhu uštede vremena.
- Ukoliko je koeficijent manji od 0.5 odmah povećati korak za 2 stepena, zbog istih razloga kao u prethodnom slučaju.

### 3.3.2 Izdvajanje potencijalnih puzzli

Slaganje krajnjih ivica je relativno brzo jer je potrebno provjeriti manji broj puzzli, ali i manji broj ivica puzzli, jer ih je moguće rotirati samo na jednu stranu. Problem s vremenom potrebnim za slaganje nastaje prilikom slaganja redova i kolona koji nisu ivice. U ovom slučaju je potrebno provjeriti mnogo više puzzli, ali i sve moguće rotacije svake puzzle (u narednom poglavljju će biti govora o tome koliko tačno vremena traje slaganje ukoliko se iskoriste određena ubrzanja). Zbog toga je bilo potrebno osmisliti neki brz i efikasan način koji će suziti izbor sa 100 puzzli na samo nekoliko nad kojima će se kasnije vršiti detaljna mogućnost uklapanja korištenjem template matching-a. Dodatno, bilo je potrebno osigurati invarijantnost na rotaciju.

Procesor veoma brzo može vršiti osnovne algebarske operacije kao što su sabiranje, oduzimanje, množenje i dijeljenje. Jasno i template matching nije ništa drugo nego korištenje osnovnih algebarskih operacija. Međutim, računanje kroskorelacijske dvije slike zahtjeva veliki broj operacija. Dakle cilj nam je potencijalno uklapanje dvije puzzle svesti na što je manji broj računanja i osigurati invarijantnost na rotaciju.

Odlučeno je da će se za svaku stranicu puzzle odrediti dvije tačke koje će zajedno sa dva susjedna čoška tvoriti dva jednakoststranična trougla. Jedan jednakoststranični trougao se nalazi "unutar" puzzle, a drugi "van" puzzle. Da bi bilo jasnije o čemu je ovdje riječ prikazana je slika (slika 3.22), na kojoj su vidljive dvije plave tačke koje zajedno sa dva susjedna čoška tvore jednakoststranični trougao. Na istoj slici su prikazani "unutrašnji" i "vanjski" trouglovi obojeni zelenom i žutom bojom respektivno. Dalje se za obje tačke računa udaljenost do svake tačke na konturi i ove udaljenosti se spremanju u vektor za tu specifičnu tačku (unutrašnju ili vanjsku). Na slici su prikazni trouglovi samo za desnu ivicu, slično se dobijaju za ostale ivice. Dakle za svaku slagalicu imamo 8 karakterističnih tačaka, poslijedictvo 8 vektora koji se sastoje od dužina svake tačke do ivice koju "opisuju".



Slika 3.22: Prikaz karakterističnih tačaka koje će biti korištene za određivanje potencijalnog uklapanja dvije puzzle

Jednom kada imamo podatke za sve tačke, moguće je krenuti u određivanje potencijalnih puzzli koje se mogu uklopiti. Ukoliko bi ivica svake puzzle bila savršeno određena, onda bi vektori udaljenosti unutrašnje tačke prve puzzle i vanjske tačke druge puzzle bili identični. Postojanje šuma ima veliki uticaj. Zbog toga se kao mjeru "potencijalnog uklapanja" puzzli uzima kvadrat greške. Usvaja se da se puzzle "potencijalno mogu uklopiti" ako je ova greška manja od nekog broja. Ovaj broj je usvojen iskustveno tako da je broj puzzli relativno mali, ali da se među tim potencijalnim puzzlama uvijek nalazi ona koja se zaista treba uklopiti (čiju će detaljnu mogućnost uklapanja provjeriti template matching).

Jasno je zašto je ova metoda invariantna na rotaciju, ali i dalje postoji jedan problem. Ukoliko ivice puzzli koje se trebaju uklopiti nemaju isti obim (jedna ivica je duža od druge zbog nesavršenosti proizvodnog procesa), onda će unutrašnja i vanjska tačka biti različito udaljene, pa će samim tim i greška biti velika, iako puzzle možda oblikom dobro prate jedna drugu. Ovo je riješeno programski. Ovdje nemamo prostora da to detaljno opišemo, ali je vrijedilo istaći ovu manu.

Na kraju vrijedi napomenuti da se za svaku moguću puzzlu, pri procesu traženja odgovarajuće, u određeni vektor upisuje koeficijent. Taj koeficijent je direktno posljedica template matchinga ukoliko se došlo do tog koraka, dok ukoliko to nije slučaj u vektor se upisuje neki veliki broj ( $>1000$ ). Razlog tome jeste što se iz ovog vektora poslije pređenih svih mogućih puzzli u konačnu matricu puzzle upisuje index vektora sa najmanjih koeficijentom, što u prevodu znači sa najboljim poklapanjem. U nastavku će biti pobrojani svi uslovi (samim tim i koeficijenti koji se upisuju u vektor) koje puzzla mora proći da bi se došlo do dijela koji iziskuje najviše vremena, template matching:

- Ukoliko se sklapa prvi red, ako puzzla nema ravnu gornju ivicu vraća se koeficijent 1000
- Ukoliko se sklapa posljednji red, ako puzzla nema ravnu donju ivicu vraća se koeficijent 10000
- Ukoliko se sklapa posljednja kolona, ako puzzla nema ravnu desnu ivicu vraća se koeficijent 100000
- Ukoliko se sklapa prva kolona, ako puzzla nema ravnu lijevu ivicu vraća se koeficijent 1000000

- Ukoliko se sklapa puzzla iz unutrašnjosti (ne na već pomenutim ivicama), ukoliko se ona ne može uklopiti u obje susjedne puzzle (lijevo i gore ili desno i dolje) vraća se koeficijent 20000
- Ako puzzla ima veću razliku dužina od 55 vraća se koeficijent 9000
- Ukoliko nije zadovoljen nijedan prethodni uslov ide se na template matching

Na kraju svega se formira konačna matrica u kojoj su smješteni brojevi svih puzzla izdvojenih u jedan zaseban folder, te su iste spremne za formiranje konačne slike posmatrane puzzle.

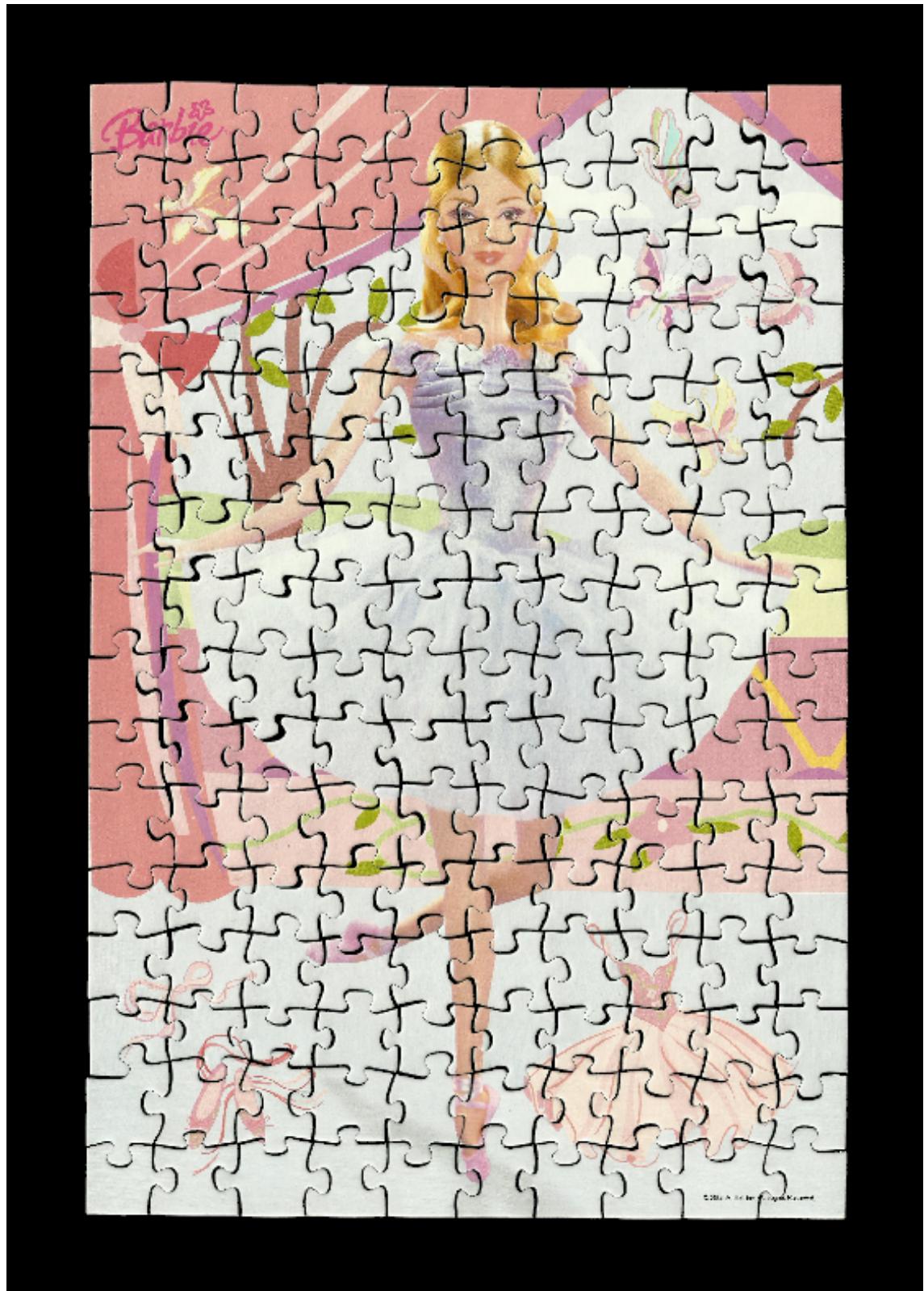
# Formiranje konačne slike

Kao posebna cjelina ovog rada se može izdvojiti formiranje konačne slike iz matrice dobijene prethodno opisanim algoritmom. Autori su vjerovatno proveli više vremena nego što sebi žele priznati pokušavajući iz dijelova puzzli prikazati konačan sliku.

Naizgled jednostavan zadatak (posložiti redne brojeve slagalica na praznu sliku i dobiti konačan izgled) kada se kreće sa implementacijom nije toliko lagan. Problemi koji se pojavljuju su dati ispod:

- određivanje čoškova na puzzlama (riješeno ranije u kôdu)
- male rotacije prave veliki problem. Neka se gornji desni čošak jedne puzzle poklopi se gornjim lijevim čoškom puzzle koja se treba uklopiti u nju. Ukoliko postoji mala rotacija, onda se donji desni čošak prve puzzle neće poklopiti s donjim lijevim čoškom druge puzzle. Ukoliko se desi i najmanje odstupanje od samo nekoliko piksela, ovaj problem se akumulira i nastane velika greška na kraju puzzle.
- puzzla nije pravougaonik, ima udubljenja i ispuštenja. Ovo se odnosi na način slaganja. Udubljenje prve puzzle treba biti popunjeno ispuštenjem druge.
- nije moguće slagati puzzle samo red po red. Ukoliko pogriješimo samo jedan pixel prilikom rotacije, na 10 puzzli greška je 10 piksela, odnosno cijela slagalica će izgledati kao da je "uvrnuta" u jednu stranu. Potrebno je konstantno vršiti kompenzaciju eventualno nastale greške (iako ne znamo da li se greška uopšte desila).
- postojanje puzzli koje kad se uklope nemaju istu dužinu ivice. Ovaj problem nije primjetan na početku, jer razlika u dužinama puzzli nije velika, otprilike od 5 do 15 piksela. Međutim kada program prikaže peti red, greška se zbog svega prethodnog počne nekontrolirano nagomilavati. Pa program treba voditi računa i o ovome i vršiti potencijalne kompenzacije.

Mogli bismo posvetiti desetak stranica o načinu prikaza složene puzzle i specifičnostima implementacije. Obzirom da nam to nije tema rada, reći ćemo samo da smo većinski riješili sve gore pobrojane probleme i dobili konačnu sliku. Slika konačne puzzle je data na slici (slika 4.1). Slaganje puzzle je vršeno odozdo prema gore, pa zato prvi red nije "savršen".



Slika 4.1: Konačni izgled puzzle

# Eksperimentalni rezultati

U procesu formiranja završnog kôda, isti je prošao kroz nekoliko faza, pri čemu se iz svake faze mogu izvući određeni zaključci. U nastavku će biti navedene verzije kôda sa svim svojim osobinama, pri čemu svaku verziju odlikuje osobina da je konačna matrica puzzli uspješno složena. Procesor na kojem se izvršavao kôd je Ryzen 5 3500U. Program je izvršavan na jednoj jezgri procesora. Frekvencija procesora je: 2.1Ghz 8GB RAM-a.

- Prva verzija kôda se samo bazirala na template matchingu. Izvršene su sve moguće provjere uklapanja puzzli (naravno, nisu provjeravani slučajevi udubljenje-udubljenje ili ispučenje-ispučenje), bez bilo kakve optimizacije u samom template matchingu ili bilo kakvog izdvajanja potencijalnih puzzli. Također, testirana je samo jedna rotacija puzzli i ukupno vrijeme slaganja je iznosilo nešto malo iznad 40 minuta.
- U sljedećoj verziji, dodani su određeni uslovi u funkciju template matchinga, čime se postupak znatno ubrzao. Također, izbačeno je poređenje puzzli koje se ne mogu uklopiti sa obje strane. I u ovom slučaju je u toku testiranja korištena samo jedna rotacija puzzli, što u našem slučaju znači da je korišteno 160 puzzli i vrijeme izvršavanja ovog kôda je 3 minute i 25 sekundi.
- U narednoj verziji pretjeranih izmjena u kôdu nije bilo, osim što su se ovaj put gledale sve četiri rotacije pojedinačnih puzzli. Prvi put možemo reći da su uzeti u obzir svi slučajevi. Kako nije bilo izmjena u kôdu očekivati je da vrijeme slaganja u ovom slučaju bude približno vremenu prethodne verzije pomnoženo sa četiri. I stvarno, vrijeme slaganja u ovom slučaju iznosi 12 minuta i 10 sekundi.
- Uz dodavanje dijela za izdvajanje potencijalnih puzzli, nova verzija je dala veoma dobre rezultate. Pooštreni su još neki zahtjevi, te je izvršenje kôda sa svim rotacijama (640 puzzli) smanjeno na 3 minute i 20 sekundi.
- U posljednjoj verziji, gdje je dodano sve pomenuto u radu, postignuti su sljedeći rezultati.
  - Utrošeno vrijeme za akviziciju puzzli 42s
  - Vrijeme formiranja matrice konačne puzzle 54s
  - Vrijeme formiranja konačne slike puzzle 11s

Za kraj, predstavljamo vrijeme izvršavanja na procesoru koji je dvije generacije mlađi, Ryzen 5 5500U, 2.1Ghz 16GB RAM-a

- Utrošeno vrijeme za akviziciju puzzli 27.5s
- Vrijeme formiranja matrice konačne puzzle 34s
- Vrijeme formiranja konačne slike puzzle 8.6s

Iako je riječ o procesorima namijenjenim laptopima, autori su mišljenja da je izvršavanja (slaganje puzzli) relativno brzo. Primjera radi u radu koji su autori pronašli, a koji se bavi istom tematikom, slaganje 200 puzzli, skeniranih rezolucijom 100 DPI, dakle duplo manjom rezolucijom, na procesoru Intel Core2 Duo 2.4 GHz i 4 GB RAM-a, je trajalo 9 minuta. U poređenju sa našim od samo 34 ili 54 sekunde, poboljšanje je zaista veliko. Potrebno je uzeti u obzir i činjenicu da je riječ o nekoliko generacija mlađim procesorima, koji su istini za volju namijenjeni za laptape, a Intel Core2 Duo je desktop procesor. Da bismo na neki način pokušali "izjednačiti" snage, na internet stranici gadgetversus.com smo pronašli podatak da procesor "Intel Core 2 Duo E8400" (korišteći jednu jezgru) na programu koji služi za mjerjenje performansi procesora "Cinebench R23 single thread" ostvaruje rezultat: 450, a procesor "Ryzen 5 5500U" (također na jednoj jezgri) ostvaruje rezultat: 1155. Ukoliko se kao program za testiranje performansi procesora koristi "PassMark single thread", onda procesor "Intel Core 2 Duo E8400" ostvaruje rezultat: 1233, a procesor "Ryzen 5 5500U": 2462. zaključuje se da je procesor: "Ryzen 5 5500U" 2.47 puta brži u odnosu na procesor "Intel Core 2 Duo E8400" (Cinebench R23 single thread), te 2 puta brži ukoliko se koristi PassMark single thread.

Imajući ovo u vidu, te ukoliko saberemo sva vremena izvršavanja našeg programa ( $27.5 + 34 + 8.6$ s) i uvažavanje činjenice da je naš procesor brži oko 2.5 puta, onda bi izvršavanja našeg programa na istom (približno istom) hardveru, kao u radu [1] bilo otprilike 3 minute. Bitno je nglasiti da su se neke implementacijske pojedinosti vjerovatno promijenile (poboljšale) u proteklih 11 godina (kada je pisan rad [1]), i to da se u radu slagalo 200 puzzli skeniranih rezolucijom 100 dpi, a ne 160 puzzli skeniranih rezolucijom 200 dpi kao što je to slučaj kod nas.

Prilikom rada programa, program interno za sebe formira veliki broj slika. Ove slike su konture različitih debljina, binarizirane slike, puzzle u različitim rotacijama. Obzirom da znamo da ovo sve zauzima prostor, nakon završetka rada, program se sam pobrine da svi nepotrebno kreirani folderi i fajlovi budu izbrisani. Naravno potrebno je za vrijeme izvršavanja programa osigurati najmanje 200MB prostora na mašini na kojoj se program izvršava da bi program mogao formirati sve slike koje su mu potrebne.

# Zaključak

U prvom dijelu ovog rada je pokazan efikasan i veoma brz način slaganja idealnih puzzli. Pоказало се да идеалности таквих puzzli u mnogome olakšavaju posao. Prošlo se kroz jednostavno određivanje maske, konture i čoškova svake puzzle. Kao rezultat oba dataset-a jeste formiranje konačne matrice za svega nekoliko sekundi.

Sve neidealnosti su obrađene u dijelu sa realnim puzzlama, te se istim posvetilo dosta pažnje. Poseban akcenat je stavljen na načine akvizicije puzzli, što je osnova daljeg rada sa istim. U programskom dijelu su u detalje objašnjene sve nesavršenosti, te kako se izborilo sa njima. Na jasan način je prezentiran i algoritam slaganja puzzli. Na kraju svega je naveden i dio za formiranje konačne slike.

Stvari koje je potrebno poboljšati, promijeniti ili u potpunosti iznova osmisliti:

- lijepo napisati kôd: Trenutno napisani kôd radi i funkcionalan je. Međutim, obzirom da je kôd pisan u etapama, određene stvari su redundantne. Pa tako npr, nije potrebno spašavati konture na kojima su tačke obojene različitim nijansama zelene boje da bi se negdje drugo u kôdu mogli detektirati čoškovi puzzle kada su svi čoškovi spašeni u neku varijablu u kôdu. Ovakvih primjera redundancije ima na još nekoliko mjestu. Vjerujemo da bi ovo prilično ubrzalo izvršavanje programa, a i njegovu čitljivost.
- napraviti pozadinsko osvjetljenje: U radu je opisan postupak koji je potrebno uraditi da bi se napravilo pozadinsko osvjetljenje. Na ovaj način bi se mogle slagati puzzle proizvoljene boje. Trenutno smo u mogućnosti složiti puzzle koje nemaju mnogo crne ili bijele boje na sebi.
- testirati program za različite puzzle: Da bi ovo bilo moguće prije svega je potrebno napraviti pozadinsko osvjetljenje. Zatim je potrebno u kôd dodati već implementirani dio koji rotira proizvoljno rotirane puzzle tako da budu "vodoravne", odnosno onako rotirane kako su trenutno rotirane ulazne skenirane puzzle. Potrebno je uvesti parametar koji će na osnovu rezolucije skenirane slike određivati koji opseg obima pronađenih kontura na skeniranoj slici odgovara konturama puzzli (da izbjegnemo konture koje pripadaju nekim oštećenjima ili su prašina). Također, bitno je omogućiti da program sam može odrediti broj redova i kolona konačne matrice puzzli. Ovo vjerujemo da nije teško, ali je potrebno promijeniti dosta kôda da bi se omogućilo.
- potencijalno dodavanje neke vrste mašinskog učenja za detekciju čoškova.
- paralelizacija, te korištenje GPU zarad ubrzanja izvršavanja kôda. Zadatak slaganja puzzli je tipičan primjer gdje paralelizacija omogućava značajna ubrzanja. Trenutno napisan kôd se može krenuti slagati iz bilo kojeg čočka. Pa je tako moguće slagalicu krenuti slagati iz sva četiri čoška istovremeno. Gdje bi se ova 4 procesa izvršavala na 4 različita jezgra procesora. Dodatno, čim je jedna puzzla složena, moguće je odmah pristupiti vizualizaciji

iste. Također, sa povećanjem broja puzzli, potencijalne greške su moguće. Pa bi jedan dio procesora mogao "pregledati" da li je sve do tada složeno uredu.

- napraviti robota koji zaista slaže puzzle. Ovo je konačna želja autora, a to je da neko u budućnosti napravi robota (robotski manipulator, ili 2D CNC mašinu) koja bi puzzle rasute po površini sama skenirala i složila.

Kada se pogleda prethodni dio teksta, neko bi mogao doći do zaključka da je ostalo puno toga za implementirati. Ali u tome i jeste čar dolaska do novih znanja. Nadamo se da smo otškrinuli vrata koja će neko u budućnosti otvoriti. Nadamo se da će budući istraživači uživati u završetku ovog projekta kao što smo i mi. Više od dvijehiljadešestotina linija kôda koje smo napisali su ispunile protekla tri mjeseca izrade ovog zadatka.

# **Literatura**

- [1] Peter Ondrúška. “Automatic assembly of jigsaw puzzles from digital images”. Disertacija. Charles University in Prague, 2011.