UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Inženirski pristop k načrtovanju in implementaciji reševalca Sudoku za mobilne naprave

(Engineering approach to Sudoku solver design and implementation)

Ime in priimek: Marko Tavčar
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Jernej Vičič
Somentor: doc. dr. Branko Kavšek

Koper, september 2015

Ključna dokumentacijska informacija

Ime in PRIIMEK: Marko TAVČAR

Naslov zaključne naloge: Inženirski pristop k načrtovanju in implementaciji reševalca sudoku za mobilne naprave

Kraj: Koper

Leto: 2015

Število listov: 36 število slik: 12 število tabel: 2

Število prilog: 1 število strani prilog: 1 število referenc: 31

Mentor: doc. dr. Jernej Vičič

Somentor: doc. dr. Branko Kavšek

Ključne besede: Sudoku, Android, OpenCV, prepoznavanje številk

Izvleček: Zaključna naloga predstavlja inženirski pristop k načrtovanju in implementaciji reševalca sudoku za mobilne naprave. Aplikacija omogoča optično zaznavanje številk kot metodo vnosa. Naloga nudi pregled najbolj znanih algoritmov za reševanje igre sudoku in pregled metod ter orodij za optično zaznavanje številk.

Key words documentation

Name and SURNAME: Marko TAVČAR

Title of final project paper: Engineering approach to sudoku solver design and implementation

Place: Koper

Year: 2015

Number of pages: 36 Number of figures: 12 Number of tables: 2

Number of appendices: 1 Number of appendix pages: 1 Number of references: 31

Mentor: Assist. Prof. Jernej Vičič, PhD

Co-Mentor: Assist. Prof. Branko Kavšek, PhD

Keywords: Sudoku, Andrid, OpenCV, digit recognition

Abstract: In this article we describe the process of design and implementation of sudoku solver for mobile devices. The application enables optical digit recognition as an input method. Article gives an overview of the best known algorithms for solving sudoku and overview of methods and tools used for optical digit recognition.

Zahvala

Zahvaljujem se mentorju, doc. dr. Jerneju Vičiču in somentorju doc. dr. Branku Kavšku , za strokovno pomoč pri izdelavi zaključne naloge.

Zahvaljujem se tudi družini in prijateljem za podporo in pomoč med študijem.

Kazalo vsebine

Seznam tabel

Seznam slik

Seznam prilog

Seznam kratic

API	Application programming interface
ART	Android runtime
CV	Computer Vision
DVM	Dalvik virtual machine
HOG	Histogram of oriented gradients
$k-NN$	k-Nearest Neighbors
NDK	Native Development Kit
OpenCV	Open Source Computer Vision
SVM	Support vector machine

1 Uvod

Zaključna projektna naloga predstavlja inženirski pristop k načrtovanju in implementaciji aplikacije za mobilne naprave ki rešuje sudoku. Aplikacija omogoča uporabo kamere in optičnega prepoznavanja številk kot način vnosa.

V drugem poglavju so opisana pravila igre sudoku in načini reševanja. Podrobneje so predstavljena orodja ki se lahko uporabijo za prepoznavanje črk, kot so na primer knjižnica Tesseract ter razvrstitveni metodi *k najbližjih sosedov* (KNN) in *metoda podpornih vektorjev*(SVM).

Tretje poglavje vsebuje metodologijo oziroma opisuje kaj smo vse potrebovali za razvoj aplikacije.

V četrtem poglavju so podstavljeni rezultati meritev hitrosti izbranega algoritma za reševanje igre sudoku ter evalvacija točnosti in hitrosti optičnega zaznavanja števil.

Zaključna projektna naloga vsebuje tudi podatke o licencah uporabljenih knjižnic kot tudi podatke o licenci razvite aplikacije.

1.1 Motivacija

Večina sodobnih mobilnih naprav omogoča nameščanje aplikacij tretje osebe. Proizvajalci kot tudi razvijalci aplikaciji za mobilne naprave imajo koristi od tega, saj razvijalci neposredno tržijo aplikacije, ki dodajajo nove funkcionalnosti obstoječim napravam. Razvoj aplikacij je danes lažji kot kdajkoli prej, saj na spletu obstaja veliko računalniških programov, knjižnic in navodil, ki nam pomagajo pri razvoju.

Odločili smo se razviti aplikacijo, ki bo znala reševati sudoku igre. Večina obstoječih aplikacij, ki rešujejo sudoku zahtevajo ročni vnos številk. Naša aplikacija bo imela možnost samodejnega vnosa številk iz zajete slike, kot tudi možnost ročnega vnosa.

2 Predstavitev domene

2.1 Sudoku

Sudoku je logična igra, v kateri igralec poskuša zapolniti kvadratno mrežo dimenzije 9 x 9 s števili od 1 do 9 [10]. Mreža je sestavljena iz 9ih polj, vsako polje vsebuje podmrežo velikosti 3x3. Vsako polje, kot tudi vsaka vrstica in stolpec celotne mreže mora vsebovati vsa števila med 1 in 9, pri čemer se vsako število pojavi enkrat [10] [21]. Primer sudoku mreže lahko vidimo v sliki 2.1.

V nadaljevanju bomo predstavili nekaj metod za reševanje sudokuja.

2.1.1 Sestopanje

Sestopanje ali vračanje (ang. backtracking) je pristop v programiranju, ki postopoma zgradi možne rešitve. Algoritem zavrne vsako delno rešitev, takoj ko ugotovi da le ta ne izpolnjuje pogojev, oziroma da jo ni mogoče dopolniti do veljavne rešitve [15].

Zgoraj navedeni pristop lahko uporabimo da popolnimo mrežo sudokuja. To naredimo, tako da se sprehodimo skozi mrežo od leve proti desni ter od zgoraj navzdol dokler ne najdemo prazno polje. V to polje poskušamo vstaviti število, tako da število ni v nasprotju z nobenim od sudoku pravil. Lahko se zgodi da imamo več takih števil. V tem primeru izberemo enega in nadaljujemo. Če ne moremo najti tako število to pomeni, da smo v enem od prejšnjih korakov izbrali napačno številko. Algoritem se v tem primeru vrne na prejšnji korak in izbere novo številko, s katero bo poizkušal popolniti sudoku [5].

Algoritem konča, ko je cela sudoku mreža zapolnjena.

2.1.2 Algoritem X

Za dano matriko, ki je sestavljena samo iz enk in ničel, algoritem x najde množico vrstic v kateri se enka pojavi natanko enkrat za vsaki stolpec [12].

	6	5		3				7
1		7			5			
		8						1
			2		1			
		6		8		3		
			5		3			
5						6		
			8			4		3
4				7		2	1	

Slika 2.1: Primer sudoku mreže [29]: Na začetku je mreža deloma izpolnjena.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \tag{2.1}$$

Na primer, matrika 2.1 ima takšno množico vrstic $\{1, 4, 5\}$. Temu se bolj strokovno na področju matematike, ter računalniške znanosti reče *exact cover*.

Glede na kompleksnost algoritma X, ne bomo v podrobnosti predstavili korake do rezultata, saj to ni namen diplomske naloge.

Sudoku lahko rešimo z algoritmom x če ga predstavimo kot redko *matriko* enk in ničel. To lahko naredimo na naslednji način:

Stolpci *matrike* predstavljajo pravila. V slučaju igre sudoku, imamo 4 pravila:

- pravilo celice: samo eno število n zapišemo v celico sudoku mreže
- pravilo vrstice: številka n se lahko samo enkrat pojavi znotraj ene vrstice
- pravilo stolpca: številka n se lahko samo enkrat pojvati znotraj enega stolpca
- pravilo polja: številka n se lahko samo enkrat pojavi znotraj enega polja dimenzije 3x3

Vsaka številka n v sudoku mreži ima svoja pravila. Zaradi tega bo *matrica* imela $9 * 9 * 4 = 324$ stolpcev.Vrstice predstavljajo vsako možno pozicijo za vsako številko, torej bo *matrica* imela $9 * 9 * 9 = 729$ vrstic [7].

2.1.3 Ostali

Constraint programming

Uporaba constraint programming modela za reševanje igre sudoku je najbolj podobna tehniki reševanja, ki jo ljudje uporabljajo [9] [27].

Stohastično iskanje

Stohastično oziroma učnnakljo iskanje je ena od optimizacijskih metod ki pride do rešitve tako da [19] [18]:

1. naključno vpiše številke v sudoku mrežo
2. prešteje napačna števila
3. poskuša optimizirati število napačnih števil

2.2 Optično prepoznavanje znakov

Optično prepoznavanje znakov (ang. OCR) nam omogoča pretvorbo fotografij v besedilo ki ga lahko urejamo. OCR se uporablja v različnih vejah znanosti kot so: umetna inteligenca, računalniški vid, in prepoznavanje vzorcev [23].

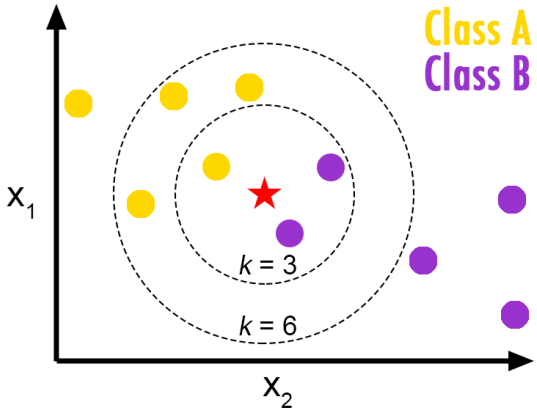
Recimo da imamo dokument v papirnati obliki in ga hočemo popraviti oziroma spremeniti. Ročno tipkanje tega dokumenta v Word obliko bi zahtevalo veliko truda in časa. Obstaja programska oprema ki naredi točno to. Dokument v papirni obliki skeniramo in sliko tega dokumenta z uporabo OCR programa prevedemo v tekstualno oziroma Word obliko.

Obstajajo tudi brezplačne in odprtokodne različice programske opreme za OCR, kot so recimo Tesseract in GOCR [28] [14].

V nadaljevanju si bomo prvo ogledali Tesseract, potem pa si bomo ogledali načine prepoznavanja znakov z uporabo klasifikacijskih algoritmov znotraj OpenCV knjižnice.

2.2.1 Tesseract

Tesseract je odprtokodni OCR program, dostopen pod Apache 2.0 licenco. Dostopen je za večino operacijskih sistemov, kot so Linux, Mac OS, Windows. Napisan je v programskem jeziku C++. Najnovejša različica je 3.02 [28].



Slika 2.2: Primer k-NN algoritma [11]: pri $k = 3$, zvezda je klasificirana kod razred B, pri $k = 6$ zvezda je klasificirana kot razred A.

Tesseract nima grafičnega vmesnika ampak obstaja več drugih programov ki mu dodajajo to funkcionalnost [16]. Lahko ga uporabljamo direktno, prek ukazne vrstice ali iz drugega programa z uporabo programskega vmesnika(API).

2.2.2 OpenCV

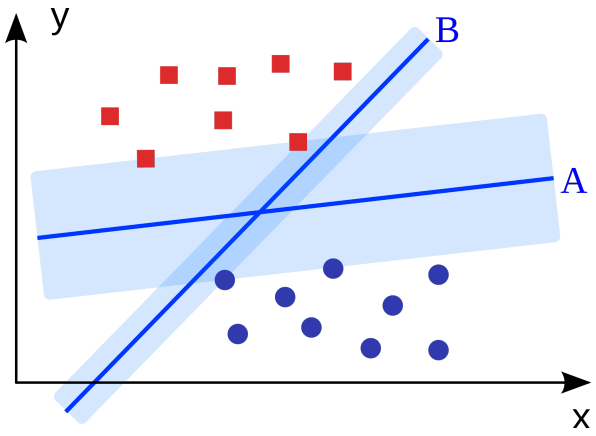
OpenCV je odprtokodna knjižnica, ki se uporablja za računalniški vid, brezplačno je dostopna pod BSD licenco [8]. Računalniški vid je področje računalništva, ki se ukvarja z obdelavo, analizo in dojemanjem slik, ali v splošnem večdimenzionalnih podatkov [31].

k-NN

K najbližjih sosedov, po angleško k Nearest Neighbours(k-NN), je algoritem za razvrščanje in regresijsko analizo [20]. Uporablja se pri strojnem učenju in rudarjenju podatkov. k-NN je eden izmed najbolj enostavnih algoritmov za strojno učenje.

Učni primeri tega algoritma so večdimenzionalni vektorji skupaj z informacijo katerem razredu ta vektor pripada. Proces učenja tega algoritma je v bistvu samo shranjevanje učnih primerov [31]. Razvrščanje poteka tako da se algoritmu poda večdimenzionalni vektor in naravno število k. Algoritem vrne ime razreda kateremu pripada večina izmed k najbližjih sosedov vhodnega vektorja [31].

Slika 2.2 prikazuje primer razvrščanja dvodimenzionalnega vektorja. Imamo pet učnih primerov iz razreda A, in pet učnih primerov iz razreda B. Hočemo razvrstiti nov vhodni vektor v eden izmed dva razreda. V primeru da imamo $k = 3$, dva od tri najbližja soseda sodijo v razred B, zato bomo nov vektor razvrstiti v razred B. V primeru $k = 6$, štiri od šest najbližjih sosedov sodijo razredu A, v tem primeru bomo vhodni vektor razvrstili v razred A [11].



Slika 2.3: Primer SVM klasifikatorja [30]: hiperravnina A optimalno razdeli prostor, saj je ločitvena meja najširša v tem primeru.

Največja slabost tega algoritma je računska zahtevnost, saj je potrebno izračunati oddaljenost vhodnega vektorja do vseh ostalih [25].

SVM

Metoda podpornih vektorjev, po angleško Support Vector Machine (SVM), je skupina metod za razvrščanje in regresijsko analizo. Uporablja se pri strojnem učenju in rudarjenju podatkov [6] [22]. Za razliko od k-NN, SVM poskuša analizirati učno množico v procesu učenja. Medtem kot k-NN išče razdaljo SVM poskuša definirati hiperravnino, ki razdeli prostor, tako da so vsi primeri enega razreda učnih primerov na eni strani in vsi primeri drugega razreda na drugi strani hiperravnine. Poleg tega je hiperravnina enako oziroma maksimalno oddaljena od vseh podpornih vektorjev, kot v primeru na sliki 2.3. Podporni vektorji so učni primeri ki so najbližji hiperravnini in s tem največ vplivajo na lego hiperravnine [24].

Prednost SVM razvrščanja je v tem da enkrat ko imamo definirano ločitveno funkcijo oziroma hiperravnino lahko zelo hitro razvrščamo nove vektorje [26].

Slabost je v tem da je SVM v bistvu binarni klasifikator. Če hočemo razvrščati več razredov, moramo imeti poseben klasifikator za vsak razred.

3 Metodologija

Poglavje predstavlja natančen opis programske opreme in algoritmov uporabljenih pri razvoju mobilne aplikacije.

3.1 Programiranje aplikacij za mobilne operacijske sisteme

Mobilnve naprae, predvsem mobilni telefoni, imajo veliko potenciala za razvoj aplikaciji, saj sodobni mobilni telefoni imajo veliko število vgrajenih senzorjev, ki omogočajo razvijalcem več prostora za inovativnost [4].

V letu 2014 je prodano več kot milijardo pametnih mobilnih naprav. Najbolj priljubljeni mobilni operacijski sistem v tem letu je bil Android s tržnim deležem od 80,7%, drugi po vrsti je iOS ki drži 18,2% trga [13].

Iz tega razloga smo se odločili, da bomo našo aplikacijo razvili za Android.

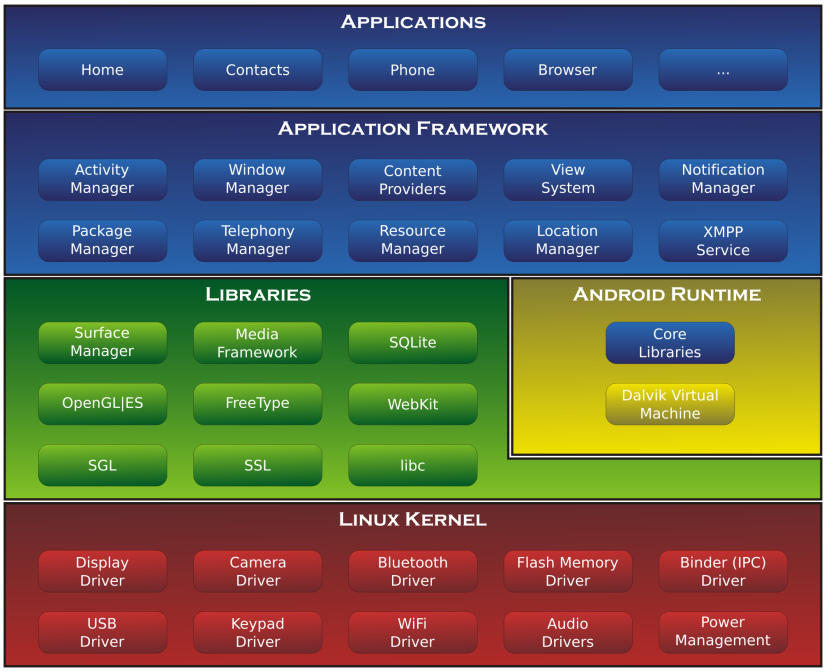
3.1.1 Arhitektura operacijskega sistema Android

Osnova Android sistema je Linux jedro, prikazano rdeče na sliki 3.1. Jedro upravlja s procesi, pomnilnikom, omrežjem in vsebuje gonilnike. Jedro je v bistvu vmesnik med strojno opremo in operacijskim sistemom [3].

Nad Linux jedrom ležijo knjižnice. Programska knjižnica je zbirka datotek, programov, metod in procedur, ki so na voljo drugim aplikacijam.

Nad Linux jedrom leži tudi izvajalno okolje, ki vsebuje osnovne Java knjižnice in Dalvik virtualni stroj(DVM). Te knjižnice so posebna implementacija standardnih Java knjižnic optimiziranih za mobilne naprave. DVM deluje kot Java virtualni stroj s tem da je optimiziran da porabi relativno malo pomnilnika. Znotraj Dalvik virtualnega stoja se izvajajo Java aplikacije. V novejših različicah Androida, DVM je zamenjan z Android Runtime(ART) [17].

Aplikacijsko ogrodje je množica osnovnih programskih vmesnikov, ki so potrebni za razvoj aplikacij.



Slika 3.1: Arhitektura Android operacijskega sistema [3]

3.2 Algoritem za reševanje Sudoku iger

Od vseh algoritmov, ki smo jih opisali v razdelku 2.1, smo se odločili za sestopanje (backtracking). Algoritem sestopanja lahko zelo enostavno in elegantno implementiramo z uporabo rekurzije. Čeprav ni najhitrejši, je dovolj hiter za naše potrebe.

Koda spodaj je implementacija algoritma v programskem jeziku Java; *sudoku* je matrica v kateri ničle predstavljajo prazne celice. Števila x in y so indeksi vrstic in stolpcev. Pomožna funkcija $legal(x, y, i)$, ki sprejme kot parametra indeksa x in y, in število i nam pove če lahko i vstavimo v $sudoku[x][y]$.

```
private boolean solve (int x, int y) {
   if (y >= 9) {
      y = 0;
      x++;
   }
   if (x >= 9) {
      return true;
   }
   if (sudoku[x][y] != 0) {
      return solve(x,y+1);
   }
   for (int i = 1; i <= 9; i++) {
      if (legal(x, y, i)) {
```

```
            sudoku[x][y] = i;
            if ( solve(x,y + 1))
               return true;
         }
      }
      sudoku[x][y] = 0;
      return false;
}
```

3.3 Obdelava slik

Poglavje opisuje način, kako smo z uporabo kamere zaznali sudoku mrežo in jo pripravili za optično prepoznavanje številk.

Za to smo uporabili OpenCV knjižnico za računalniški vid. Z uradne spletne strani smo pobrali različico 2.4.9 izvorne kode knjižnice za Android. Knjižnica je napisana v programskem jeziku C++, ampak vsebuje tudi Java vmesnik, ki nam omogoča, da iz našega programa kličemo C++ funkcije. Da lahko dodamo to knjižnico v Eclipse jo moramo prvo prevesti z uporabo Android NDK. Z uradne spletne strani smo pobrali Android-ndk-r9d. Prevedeno knjižnico smo kot projekt uvezli v Eclipse in jo dodali našem projektu.

Predpostavili smo da bo sudoku imel največji obris na sliki ki jo zajamemo. OpenCV ima funkcijo ki najde vse obrise v binarni sliki [8]. Ta funkcija sprejeme pet parametrov in sicer: vhodno binarno sliko, seznam kontur v katerega bo funkcija vpisala vse konture, hierarhično matriko ki nam pove če je ena kontura znotraj druge, parameter ki pove funkciji v kakšni obliki bi radi imeli zapisano hierarhijo in na koncu parameter ki pove če želimo aproksimirati obris.

```
void Imgproc.findContours(Mat image, List<MatOfPoint> contours,
                          Mat hierarchy, int mode, int method)
```

Ko posnamemo fotografijo Android kamera nam vrne barvno sliko. Da bi lahko uporabili zgornjo funkcijo moramo to barvno sliko pretvoriti v binarno sliko. Binarna slika je sestavljena samo iz črnih in belih slikovnih pik. Koda ki nam pretvori barvno sliko v binarno je prikazano spodaj:

```
Imgproc.cvtColor(mRgba, mBw, Imgproc.COLOR_BGR2GRAY);
Imgproc.GaussianBlur(mBw, mBw, new org.opencv.core.Size(5, 5), 0);
Imgproc.adaptiveThreshold(mBw, mBw, 255,
     Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C,
```

```
    Imgproc.THRESH_BINARY_INV, 11, 2);
```

Funkcija cvtColor pretvori barvno sliko mRgba v sivinsko sliko mBw. GaussianBlur zamegli sliko, kot parametre dobi vhodno in izhodno sliko, velikost Gausovega jedra in standardni odklon. Sliko zameglimo, da bi se znebili šuma na katerega je funkcija adaptiveThreshold občutljiva.

Thresholding je proces segmentacije slike z sivimi odtenki [8]. Slikovna pika (i, j) izhodne slike bo imela vrednost 255 samo če vhodna pika ima na istem indeksu slikovno piko z vrednostjo večjo od x. Globalni thresholding, kot samo ime pove, uporablja globalno vrednost parametra x. To ima za posledico, da je ta algoritem zelo občutljiv na razliko v osvetlitvi vhodne slike, kot je razvidno iz slike 3.2.b)

$$dst(i,j) = \begin{cases} 255 & ; src(i,j) > x \\ 0 & ; sicer \end{cases}$$

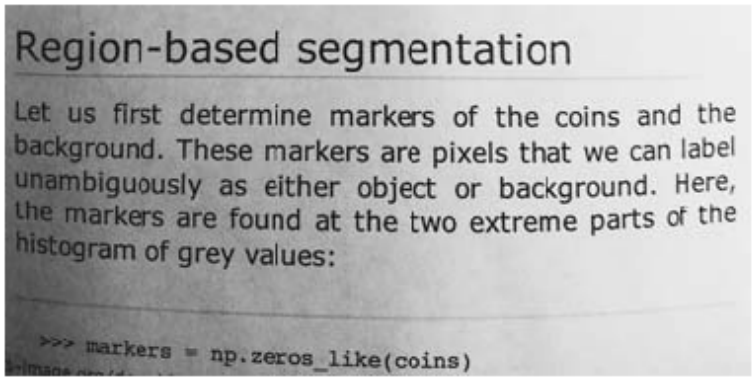
Prilagodljivi prag (threshold) določi lokalno vrednost parametra x kot uteženo (ponderirano) vsoto bližnjih vrednosti slikovnih pik [8]. Ker je x lokalen, razlike v osvetlitvi vhodne slike imajo zanemarljiv vpliv na izhodno sliko, kot je razvidno iz slike 3.2.c).

V seznamu zaprtih kontur poiščemo konturo, ki ima največjo površino in ima 4 oglišča. Kontura je v bistvu vektor koordinat, zato moramo narediti približek če želimo preveriti število oglišč. Približek naredimo s funkcijo $Imgproc.approxPolyDP$ ki sprejeme vhodno konturo, izhodno spremenljivko, parameter, ki pove natančnost približka in parameter, ki pove če hočemo da je približek zaprta ali odprta konturo. To naredimo z naslednjo kodo:

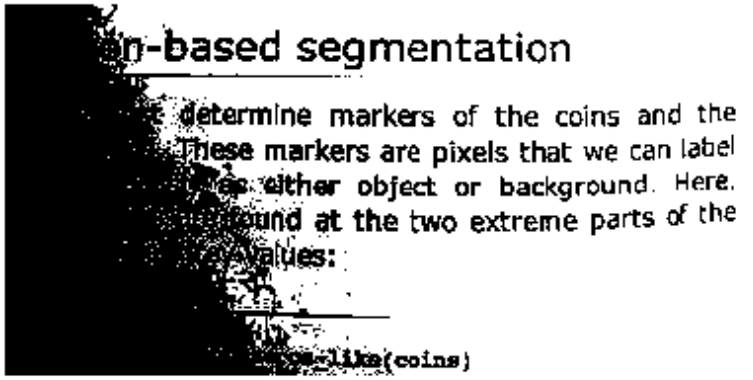
```
MatOfPoint2f curveF,approx,biggest;
double maxArea = 0.0;
for(int i = 0; i < cntrs.size(); i++){
   double area = Imgproc.contourArea(cntrs.get(i));
   curveF = new MatOfPoint2f(cntrs.get(i).toArray());
   approx = new MatOfPoint2f();
   double eps = Imgproc.arcLength(curveF, true);
   Imgproc.approxPolyDP(curveF, approx, 0.01*eps, true);
   if(area > maxArea && approx.size().height == 4 ){
      maxArea = area;
      biggest = approx;
      maxInd = i;
   }
}
```

Zdaj vemo točno kje na sliki se nahaja sudoku. Ta podatek je shranjen v vektorju

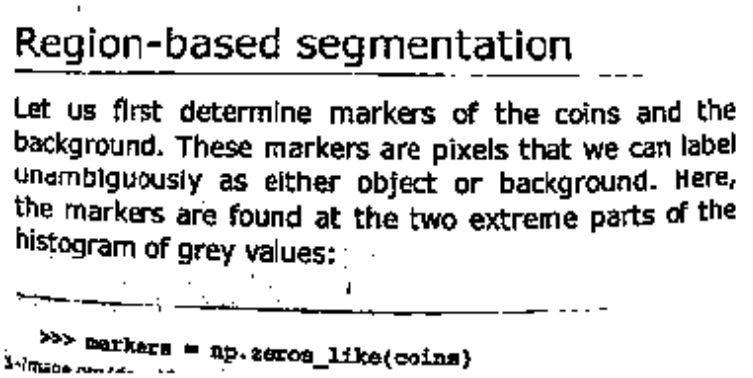
a)



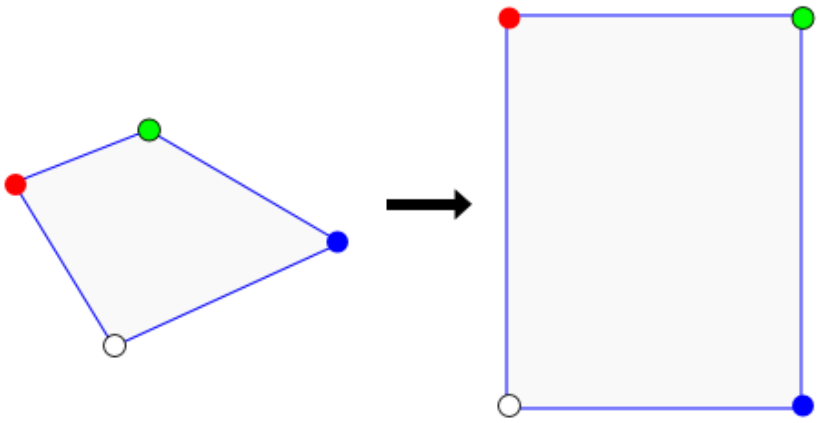
b)



c)



Slika 3.2: Razlika med globalnim in prilagodljivim thresholdom [1]: a) originalna slika, b) globalni threshold, c) prilagodljivi threshold



Slika 3.3: Perspektivna transformacija nam omogoča navpičen pogled na sliko, ki je zajeta pod kotom [2]

biggest. Problem je da je sudoku, ko ga poslikamo je skoraj zagotovo izkrivljen in nagnjen. Iz tega razloga ga moramo projicirati oziroma narediti perspektivno transformacijo kot na sliki 3.3. Sudoku bomo projicirali v novo sliko, ki bo dimenziji $700x700$ slikovnih pik. Funkcija getPerspectiveTransform nam vrne transformacijsko matriko oziroma matriko linearne preslikave iz biggest v endM, kjer je endM matrika ki vsebuje oglišča nove slike oziroma ciljnega prostora. WarpPerspective projicira sudoku v novo sliko z uporabo transformacijske matrike [8].

```
Mat pt = Imgproc.getPerspectiveTransform(sort(biggest), endM);
Imgproc.warpPerspective(mBw, mSud, pt,
     new Size(mSudSize,mSudSize));
```

Ko imamo predelano sliko sudoku mreže jo lahko ročno razdelimo na celice. In za vsako celico pogledamo če prazna ali vsebuje številko. To naredimo tako da preverimo če je dovolj velik obris znotraj celice. Če najdemo tak obris, ga izrežemo in shranimo v matriko slik dimenzij 9x9, če ne, na tem indeksu ostane null vrednost.

Tako smo sliko pripravili za naslednji korak kateri je prepoznavanje številk.

3.4 Prepoznavanje številk

V tem poglavju bomo pokazali dve metodi za prepoznavanje številk iz slik, ki smo jih pripravili v poglavju 3.3.

3.4.1 Tesseract

V poglavju 2.2.1 smo povedali kaj je Tesseract, zdaj pa bomo pokazali kako smo ga namesti in kako ga uporabljamo.

Izvorno kodo smo pobrali z naslova: *https* : *//github.com/rmtheis/tess − two*. Z uporabo Android-ndk-r9d smo prevedli izvorno kodo in potem jo uvozili v Eclipse kot nov projekt. Uvoženi projekt smo označili kot knjižnico in jo dodali v naš projekt. Tesseract uporablja dodatno datoteko, za vsaki jezik ki lahko prepozna, v katerem je shranjen model za prepoznavanje znakov. Iz naslova: *https* : *//github.com/tesseract − ocr/tessdata* smo pobrali datoteko za angleški jezik in ga dodali v projekt.

Tesseract inicializiramo z naslednjo kodo:

```
TessBaseAPI T = new TessBaseAPI();
T.init(MainActivity.TESSBASE_PATH, MainActivity.DEFAULT_LANGUAGE);
T.setPageSegMode(TessBaseAPI.PageSegMode.PSM_SINGLE_CHAR);
T.setVariable(TessBaseAPI.VAR_CHAR_WHITELIST, "123456789");
```

Funkcija *init* sprejeme ime jezika in absolutno pot do direktorija v katerem se nahaja ta jezik. S funkcijo *setPageSegMode* smo mu povedali, da bodo slike vsebovale samo en znak, medtem *setVariable* omogoča nastavitev seznama dovoljenih znakov, v našem primeru "123456789"

Prepoznavanje znakov je zelo enostavno. Funkcija *setImage* vzame sliko v bitmap obliki, jo obdela in prepozna znake. *GetUTF8Text* pa nam vrne znake kot niz.

```
T.setImage(bmp);
String txt = T.getUTF8Text();
```

3.4.2 K najbližjih sosedov

Kako *k-NN* deluje smo pokazali v poglavju 2.2.2. V tem poglavju bomo razložili kako smo ta algoritem uporabili za prepoznavanje številk.

Prepoznavanje številk je lažje kot prepoznavanje znakov, saj imamo omejeno število številk katere moramo razlikovati. Z drugimi besedami, lahko imamo samo deset različnih številk. Ker imamo majhen nabor številk, za prepoznavanje lahko uporabimo algoritme za razvrščanje.

Algoritem *k-najbližjih sosedov* je vključen v OpenCV knjižnico. Postopek namestitve OpenCV knjižnice in njeno dodajanje v naš projekt je opisano v poglavju 3.3.

Naslednja koda inicializira in trenira *k-NN* klasifikator.

```
CvKNearest knn = new CvKNearest();
knn.train(mSamples, mResponses);
```

Metoda train vzame 2 parametra, seznam vektorjev in seznam razredov. Vektorji so binarne slike, medtem ko so razredi naravna števila od ena do devet.

Slika 3.4: Primer učne množice ki ima po en učni primer za vsaki razred.

Pomembno je omeniti da je treba binarne slike pretvoriti v vektorje. OpenCV ima funkcijo ki naredi točno to.

```
mSamples.push_back(image.reshape(1,1));
```

Reshape funkcija vzame dva parametra: število kanalov in število vrstic. V našem primeru je število kanalov ena, ker binarna slika ima samo en kanal dokler barvna slika ima tri (rdeči, zeleni in modri).

Novo sliko prepoznamo z naslednjo funkcijo:

```
public int recognize(Mat slika) {
   Mat res = new Mat();
   knn.find_nearest(slika.reshape(1,1), 3,
         res, new Mat(), new Mat());
   return (int) res.get(0,0)[0];
}
```

Za razliko od Tesseracta, k-NN nima shranjenega vzorca za prepoznavanje ampak ga moramo trenirati vsakič ko ga zaženemo. Tudi ko vnesemo učno množico v k-NN nimamo možnosti shraniti klasifikacijski model, saj ga k-NN ne uporablja. Vnesena učna množica se nahaja le v delovnem pomnilniku (RAM-u) dokler se aplikacija ne zapre. Ta problem rešimo tako da shranimo učno množico v pomnilnik mobilne naprave. Učna množica je v našem primeru, nabor črno belih slik kot na sliki 3.4.

Ko zaženemo aplikacijo se $k - NN$ inicializira samo v primeru da se učna množica nahaja v pomnilniku naprave. V nasprotnem primeru $k - NN$ treniramo ročno prvič ko posnamemo sudoku. Treniramo ga tako da ročno vnesemo številko za vsako sliko številke, ki jo zazna algoritem, ki je opisan v poglavju 3.3. Vsaka zaznana slika se shrani kot del učne množice v pomnilniku mobilne naprave.

4 Rezultati

V tem poglavju bomo prikazali rezultate testiranja posameznih komponent programa. Prvo bomo prikazali hitrost algoritma, ki smo ga uporabili za reševanje igre sudoku in potem bomo predstavili hitrost in pravilnost algoritmov za prepoznavanja številk.

4.1 Aplikacija

Aplikacija ki smo jo razvili je testirana na več različnih napravah; združljiva je z Android sistemi od različice 4.2 naprej.

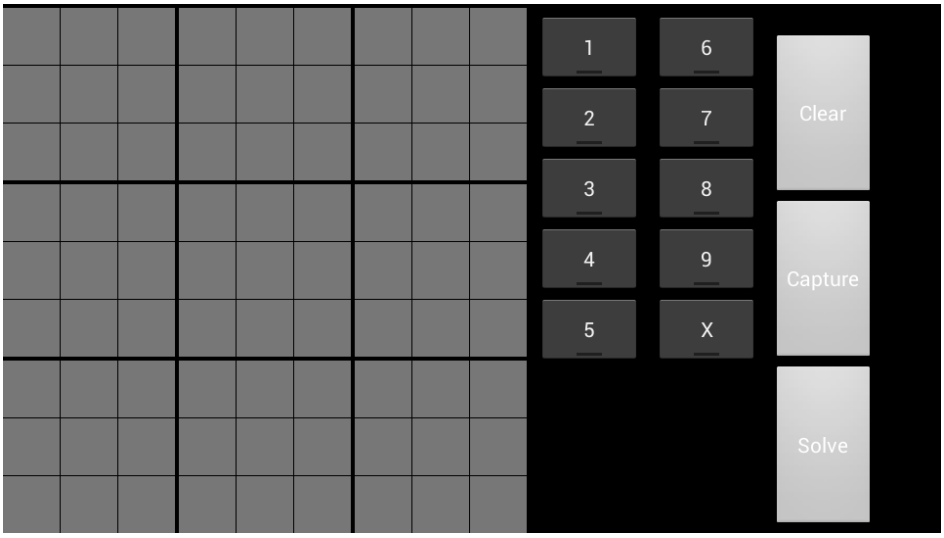
Naša aplikacija potrebuje dve dodatni dovoljenji, in sicer:

```
<uses-permission
   android:name="android.permission.CAMERA" />
<uses-permission
   android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

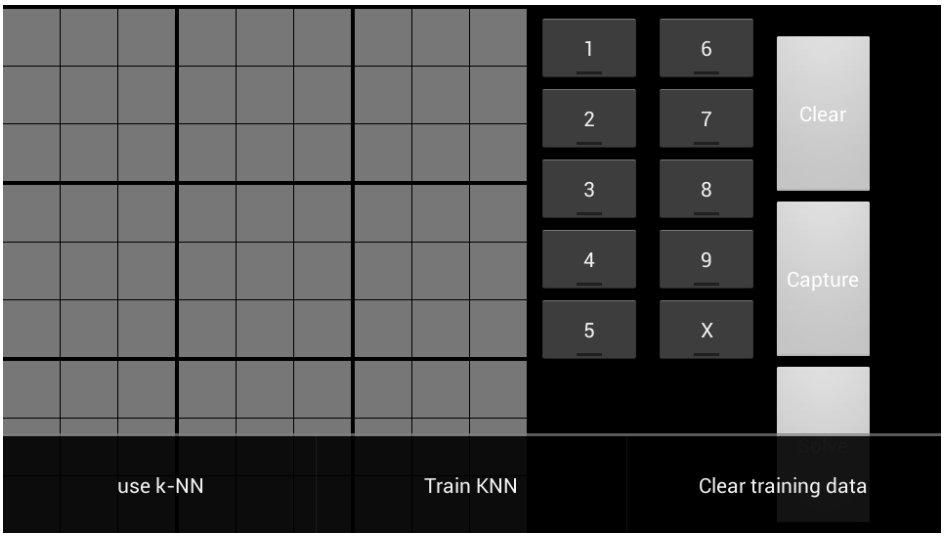
Dovoljenje za pisanje v pomnilnik potrebujemo zato ker naša aplikacija zapiše Tesseract traineddata datoteko v pomnilnik v primeru če že ne obstaja. Ta datoteka je bistvena za delovanje Tesseract knjižnice.

Aplikacija zasede 17,14 megabajtov pomnilnika pa še 20,8 megabajtov za Tesseract traineddata datoteko. Če ne bi uporabili Tesseract knjižnice bi naša aplikacija zasedla manj kot en megabajt.

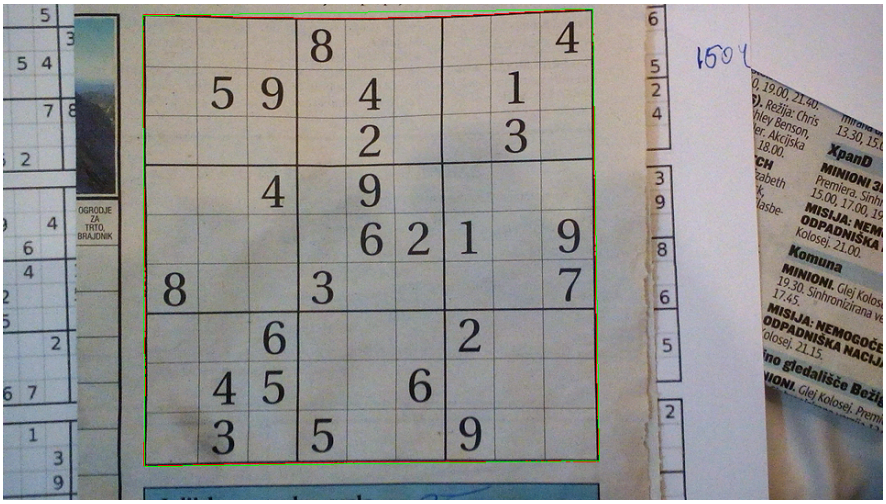
Spodaj so prikazani zaslonski posnetki razvite aplikacije.



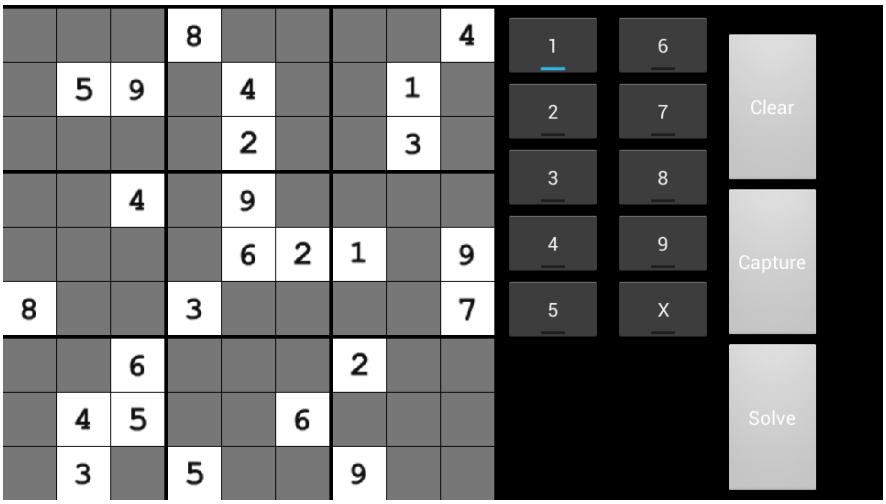
Slika 4.1: Zaslonski posnetek 1: osnovni pogled



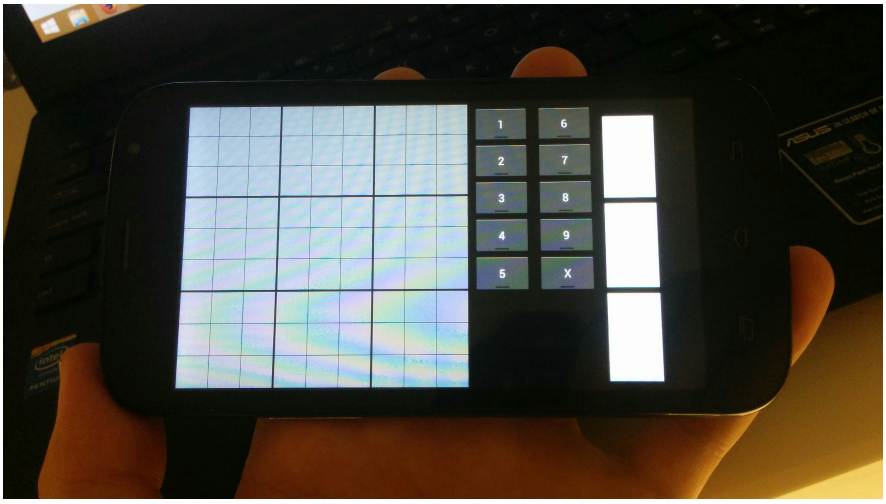
Slika 4.2: Zaslonski posnetek 2: izbira algoritma



Slika 4.3: Zaslonski posnetek 3: slikana igra iz dnevnika Pimorske novice



Slika 4.4: Zaslonski posnetek 4: vnešena igra iz slike 4.3



Slika 4.5: Delujoča aplikacija nameščena pri uporabniku na telefon Doogee Discovery DG500

4.2 Hitrost algoritma za reševanje igre sudoku

Čas izvajanja algoritma za reševanje je odvisen od težavnosti igre sudoku. Težje igre zahtevajo več časa pri izračunu rešitve.

Testiranje smo izvedli, tako da smo algoritem zagnali na deset težkih sudoku iger. Pri tem smo seveda merili koliko je milisekund preteklo od klica funkcije za reševanje do izrisa rešitve.

V tabeli 4.1 je prikazan čas izvajanja za vsako meritev.

Tabela 4.1: Meritve hitrosti algoritma v milisekundah

Meritev	1	2	3	4	5	6	7	8	9	10
Rezultat	52	30	56	156	29	146	74	408	33	154

4.3 Hitrost in pravilnost prepoznavanja številk

Pri oceni hitrosti prepoznavanja številk smo upoštevali tudi obdelavo slik; iz tega razloga je čas prepoznavanja odvisen od ločljivosti zajete slike. Meritve prikazane spodaj so zabeležene pri privzetih nastavitvah kamere oziroma pri ločljivosti $1600x1200$ slikovnih pik.

Pri meritvah smo uporabili iste igre sudoku, kot smo jih uporabili pri meritvah hitrosti algoritma. Pomembno je omeniti, da so vse igre sudoku uporabljale isto pisavo in da smo za treniranje k-NN algoritma uporabili učno množico ki je imela petdeset učnih primerov.

Pri oceni pravilnosti prepoznavanja smo upoštevali tudi pravilnost zaznavanja številk, kar pomeni, da smo napačno zaznane številke kot tudi številke, ki sploh niso bile zaznane obravnavali kot napako.

V tabeli 4.2 so predstavljeni rezultati meritev. V drugem in tretjem stolpcu je prikazan čas prepoznavanja številk v milisekundah. V četrtem in petem stolpcu je prikazan odstotek pravilno prepoznanih številk.

Tabela 4.2: Meritve hitrosti in pravilnosti prepoznavanja številk.

Meritev	Tesseract	k-NN	Tesseract	k-NN
1	2942ms	2640ms	100%	100%
2	3527ms	2085ms	100%	100%
3	3939ms	2450ms	100%	100%
4	4106ms	2631ms	96%	100%
5	4602ms	1877ms	100%	100%
6	4100ms	2561ms	97%	100%
7	4224ms	2170ms	100%	100%
8	2804ms	3097ms	96%	96%
9	2420ms	3363ms	96%	100%
10	2597ms	4309ms	97%	100%

5 Dostopnost

V tem poglavju so opisane licence knjižnic, ki smo jih uporabili. Opisali smo tudi dostopnost izvorne kode.

5.1 Licenca izvorne kode

V našem projektu smo uporabili dve odprtokodni knjižnici. Čeprav sta knjižnici odprtokodni in brezplačni se moramo držati pravil in predpisov njunih licenc. Licence nam povejo v katere namene jih lahko uporabimo.

OpenCV ima BSD (3-clause BSD licence) licenco, katero si lahko ogledate na naslovu `http://opencv.org/license.html`.

Tesseract uporablja Apache licenco 2.0, ki je dostopna na naslovu `http://www.apache.org/licenses/LICENSE-2.0`

Za naš projekt smo izbrali licenco BSD (Berkley Software Distribution). Ta licenca ni preveč zahtevna, saj ima samo 3 klavzule:

- Vse distribucije izvorne kode morajo vsebovati obvestilo o licenci
- Vse distribucije izvršilne datoteke morajo v dokumentaciji vsebovati obvestilo o licenci
- Imena avtorjev ali oseb ki so prispevale razvoju projekta se ne smejo uporabljati pri oglaševanju produktov izpeljanih iz tega projekta brez posebnega pisanega dovoljenja.

5.2 Dostopnost programa izvorne kode

Izvršilna datoteka in izvorna koda se lahko pobere iz spletne strani gitlab. Spletna povezava do tega projekta je: `https://gitlab.com/Tavcar/sudocrku`.

Gitlab smo izbrali zaradi dveh razlogov: brezplačen je za odprtokodne projekte in izvorna koda je dostopna vsem.

Če želite naprej razvijati svojo različico aplikacije potrebno je klonirati repozitorij z orodjem git na sledeči način:

```
git clone https://gitlab.com/Tavcar/sudocrku.git
```

6 Zaključek

V zaključni nalogi smo najprej predstavili najbolj znane algoritme za reševanje igre sudoku, nato smo si ogledali nekatere od pristopov za optično prepoznavanje znakov. Za tem smo pokazali, kako smo implementirali algoritem sestopanja za reševanje igre sudoku ter kako smo obdelali sliko in zaznali številke v sudoku mreži. V nadaljevanju smo razložili kako smo implementirali prepoznavanje številk z uporabo knjižnice *Tesseract* tako kot z uporabo algoritma za razvrščanje *k najbližjih sosedov*. Na koncu smo si ogledali hitrost in pravilnost delovanja pilotnega programa in njegovo licenco.

Pokazali smo, da je možno narediti program, ki z uporabo kamere dokaj hitro in pravilno vnese številke. Rezultati so pokazali, da *Tesseract*, čeprav je zelo priročen, ni vedno najboljša izbira, kadar gre samo za prepoznavanje številk.

Trenutna implementacija programa, čeprav ni popolna, se lahko uporabi v resničnem svetu. Izboljšave na področju obdelave slik in zaznavanja števil bi veliko izboljšale uporabnost programa.

7 Literatura in viri

[1] Adaptive Thresholding. http://scikit-image.org/docs/dev/auto_examples/plot_threshold_adaptive.html. *Stran dostopana: 30.8.2015. (Citirano na straneh VIII in 11.)*

[2] N. Amin. Automatic perspective correction for quadrilateral objects. *http://opencv-code.com/tutorials/automatic-perspective-correction-for-quadrilateral-objects/,Stran dostopana: 30.8.2015*, 2013. *(Citirano na straneh VIII in 12.)*

[3] Android. Android Anatomy and Physiology. *http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf, Stran dostopana: 30.8.2015*, 2012. *(Citirano na straneh VIII, 7 in 8.)*

[4] Y. Arase, F. Ren, and X. Xie. User activity understanding from mobile phone sensors. *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing - Ubicomp '10*, page 391, 2010. *(Citirano na strani 7.)*

[5] J. R. Bitner and E. M. Reingold. Backtrack programming techniques, 1975. *(Citirano na strani 2.)*

[6] Chih-Wei Hsu, Chih-Chung Chang and C.-J. Lin. A Practical Guide to Support Vector Classification. *BJU international*, 101(1):1396–400, 2008. *(Citirano na strani 6.)*

[7] J. Chu. A Sudoku Solver in Java implementing Knuth's Dancing Links Algorithm. 2006. *(Citirano na strani 4.)*

[8] O. Community. *The OpenCV Reference Manual*. 2010. *(Citirano na straneh 5, 9, 10 in 12.)*

[9] B. Crawford, M. Aranda, C. Castro, and E. Monfroy. Using Constraint Programming to solve Sudoku puzzles. In *Proceedings - 3rd International Conference on Convergence and Hybrid Information Technology, ICCIT 2008*, volume 2, pages 926–931, 2008. *(Citirano na strani 4.)*

[10] T. Davis. The Mathematics of Sudoku. *Strategies*, pages 1–26, 2008. *(Citirano na strani 2.)*

[11] B. DeWilde. Classification of Hand-written Digits, 2012. *(Citirano na straneh VIII in 5.)*

[12] Donald E. Knuth. Dancing links. *Millenial Perspectives in Computer Science, 2000, 187–214*, 2000. *(Citirano na strani 2.)*

[13] Gartner Says Smartphone Sales Surpassed One Billion Units in 2014. http://www.gartner.com/newsroom/id/2996817. *Stran dostopana: 30.8.2015*, 2015. *(Citirano na strani 7.)*

[14] GOCR. http://jocr.sourceforge.net/. *Stran dostopana: 30.8.2015. (Citirano na strani 4.)*

[15] S. W. Golomb and L. D. Baumert. Backtrack Programming, 1965. *(Citirano na strani 2.)*

[16] GUIs and Other Projects using Tesseract OCR. https://code.google.com/p/tesseract-ocr/wiki/3rdParty. *Stran dostopana: 30.8.2015. (Citirano na strani 5.)*

[17] S. P. John. Android Architecture. *http://www.eazytutz.com/android/android-architecture/, Stran dostopana: 30.8.2015. (Citirano na strani 7.)*

[18] N. E. C. Labs, I. Way, and P. Nj. Stochastic Learning. Technical report, MLSS, 2003. *(Citirano na strani 4.)*

[19] R. Lewis. Metaheuristics can solve sudoku puzzles. *Journal of Heuristics*, 13(4):387–401, 2007. *(Citirano na strani 4.)*

[20] T. Liu, A. Moore, and A. Gray. Efficient exact k-NN and nonparametric classification in high dimensions. *Advances in Neural . . .*, (c), 2003. *(Citirano na strani 5.)*

[21] N. Y. Louis Lee, G. P. Goodwin, and P. N. Johnson-Laird. The psychological puzzle of Sudoku, 2008. *(Citirano na strani 2.)*

[22] S. Maji and J. Malik. Fast and Accurate Digit Classification, 2009. *(Citirano na strani 6.)*

[23] S. Mori, C. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7), 1992. *(Citirano na strani 4.)*

[24] W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006. *(Citirano na strani 6.)*

[25] P. Piro, R. Nock, F. Nielsen, and M. Barlaud. Leveraging k-NN for generic classification boosting. *Neurocomputing*, 80:3–9, 2012. *(Citirano na strani 6.)*

[26] S. Shalev-Shwartz and N. Srebro. SVM optimization. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 928–935. ACM Press, 2008. *(Citirano na strani 6.)*

[27] H. Simonis. Sudoku as a constraint problem. *IC-Parc*, page 15, 2005. *(Citirano na strani 4.)*

[28] R. Smith. An overview of the tesseract OCR engine. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, volume 2, pages 629–633, 2007. *(Citirano na strani 4.)*

[29] T. Stellmach. Sudoku, 2006. *(Citirano na straneh VIII in 3.)*

[30] SVM. https://commons.wikimedia.org/wiki/File:Svm_intro.svg. *Stran dostopana: 30.8.2015*, 2010. *(Citirano na straneh VIII in 6.)*

[31] R. Szeliski. Computer Vision : Algorithms and Applications. *Computer*, 5:832, 2010. *(Citirano na strani 5.)*

Priloge

Priloga A: Pilotni program (zgoščenka)