Implementacija skenera dokumenata korištenjem računalne obrade slike

Mandić, Bojan

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:463222

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-17**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

Implementacija skenera dokumenata korištenjem računalne obrade slike

Završni rad

Bojan Mandić

Osijek, godina 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 02.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Bojan Mandić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4452, 13.10.2019.
OIB Pristupnika:	95798810615
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Krešimir Romić
Sumentor iz tvrtke:	
Naslov završnog rada:	Implementacija skenera dokumenata korištenjem računalne obrade slike
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak završnog rad:	Razviti desktop aplikaciju koja omogućava skeniranje dokumenata iz učitane digitalne slike ili direktno s kamere. Potrebno je imlementirati automatsko ili poluautomatsko ispravljanje perspekitve i izrezivanje dokumenta. Istražiti i koristiti metode iz računalne obrade slike kako bi se postigao bolji kontrast i oštrina skeniranog dokumenta. Omogućiti spremanje i ispis skeniranog dokumenta. Sumentor: Krešimir Romić
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	02.09.2022.
Datum potvrde ocjene od strane Odbora:	07.09.2022.
Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 07.09.2022.

Ime i prezime studenta:	Bojan Mandić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4452, 13.10.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Implementacija skenera dokumenata korištenjem računalne obrade slike**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Krešimir Romić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Tema završnog rada izrada je desktop aplikacije za skeniranje dokumenata korištenjem računalne obrade slike. Ovakav oblik aplikacije koristan je osobama koje nemaju digitalni skener, a trebaju prenijeti sliku u digitalni oblik. Korisnik kamerom slika dokument ili učitava postojeću sliku dokumenta s računala i korištenjem aplikacije pretvara ju u format koji nalikuje dokumentu koji je skeniran s pravim digitalnim skenerom. Aplikacija izrezuje dokument iz pozadine i ispravlja perspektivu istoga te korisnik ima mogućnost poboljšavanja kvalitete obrađene slike korištenjem različitih filtera i alata. Konačnu sliku moguće je pohraniti na računalo ili ju isprintati.

Za izradu aplikacije korišten je programski jezik C# [1], uz pomoć EmguCV [2], DirectShow.NET [3] za izradu funkcionalnih rješenja te Windows Forms [4] biblioteke koja je korištena za izradu grafičkog sučelja aplikacije.

U drugom poglavlju su opisana postojeća rješenja i njihova usporedba s aplikacijom u završnom radu. Treće poglavlje obuhvaća tehnologije koje su korištene u izradi aplikacije. Četvrto poglavlje se sastoji od više poglavlja koja opisuju ulazne podatke, način detekcije papira, alate za ispravljanje perspektive, poboljšanje prikaza skenirane slike i izlazne podatke. Unutar petog poglavlja analizirana je uspješnost aplikacije i njezina brzina izvođenja te korištenje hardverskih resursa. U zadnjem poglavlju naveden je zaključak i osvrt na sveukupne rezultate.

2. POSTOJEĆA RJEŠENJA

U ovome poglavlju napravljen je pregled postojećih rješenja za skeniranje dokumenata. Postojeća rješenja se, prema platformi za koju su razvijene, mogu podijeliti na: desktop aplikacije, mobilne aplikacije i web aplikacije.

CamScanner [5] je mobilna aplikacija za skeniranje dokumenata korištenjem kamere. Pruža napredne funkcionalnosti za obradu slike, poluautomatsko ispravljanje perspektive i omogućava dohvaćanje teksta sa slike i njegovo prevođenje na druge jezike. Aplikacija je naprednija u odnosu na aplikaciju opisanu u ovome radu.

Adobe Scan: PDF Scanner, OCR [6] mobilna je aplikacija za skeniranje dokumenata korištenjem kamere. Aplikacija omogućuje korisniku napredne funkcionalnosti za obradu slike, poluautomatsko ispravljanje perspektive, dodavanje različitih filtera i spremanje rezultata na oblak. Aplikacija je naprednija u odnosu na aplikaciju opisanu u ovome radu jer ima veći broj filtera i različitih načina za obradu slike (npr. crtanje i brisanje po slici dokumenta, dodavanje teksta).

Evernote [7] je web i mobilna aplikacija za vođenje bilježaka. Dokument je moguće skenirati, ispraviti mu perspektivu i dodati razne filtere korištenjem mobilne aplikacije, a zatim dokument umetnuti u bilješku i sve zajedno pohraniti u oblak. Korištenjem web aplikacije, moguć je pristup pohranjenim bilješkama i njihovo uređivanje. Funkcionalnosti za skeniranje dokumenata i njihovu obradu podjednake su s funkcionalnostima koje obuhvaća aplikacija opisana u ovome radu. Razlikuju se jedino u tome što skeniranje dokumenata nije glavna namjena *Evernote* aplikacije, već je sporedna funkcionalnost za lakše vođenje bilježaka.

OnlineCamScanner [8] predstavlja web aplikaciju za skeniranje dokumenata. Aplikacija omogućava učitavanje slike dokumenta s računala, poluautomatsko ispravljanje perspektive i mogućnost dodavanja raznih efekata sa svrhom poboljšanja kvalitete slike. Gotovu sliku je zatim moguće preuzeti na računalo. Aplikacija obuhvaća podjednake funkcionalnosti kao i aplikacija opisana u ovome radu. Jedina prednost je što je dostupna bilo gdje putem interneta.

Najsličnija desktop aplikacija je *PaperScan 3* [9], a ona je namijenjena ispravljanju pogrešaka koje su nastale korištenjem računalnog skenera. Aplikacijom je moguće izravnati sliku, ukloniti margine i poboljšati kvalitetu korištenjem različitih filtera. Sliku je zatim moguće pohraniti na računalo. Aplikacija je slična po svojoj namjeni aplikaciji koja je opisana u ovome radu, a najviše se podudaraju po funkcionalnostima za poboljšanje konačnih rezultata.

3. KORIŠTENE TEHNOLOGIJE

U ovome poglavlju navedene su i opisane tehnologije koje su korištene za izradu aplikacije. Tehnologije možemo podijeliti na programske jezike (*C#*) i biblioteke (*Emgu CV*, *DirectShow.NET*, *Windows Forms*).

3.1. C#

Za izradu ove aplikacije u potpunosti je korišten programski jezik *C#*. *C#* je moderni, objektno-orijentirani programski jezik koji omogućava programerima razvijanje različitih tipova snažnih i sigurnih aplikacija koje se pokreću na *.NET* platformi [10]. Programski jezik ima korijene u obitelji *C* jezika i sličan je programskim jezicima *C*, *C++* i Javi. Neke od značajki jezika su sakupljač smeća (engl. *garbage collector*) koji vraća memoriju koju su zauzeli nedostupni i neiskorišteni objekti, *LINQ* (engl. *Language Integrated Query*) sintaksa koja je namijenjena radu s podatcima i lambda izrazi koji podržavaju funkcionalne programerske tehnike.

3.2. EMGU CV

Emgu CV je višeplatformski *.NET* omotač za *OpenCV* [11] biblioteku za računalnu obradu slike. On omogućava korištenje *OpenCv* funkcionalnosti u *.NET* okruženju. *OpenCV* biblioteka sadrži različite alate za računalnu obradu slike, poput detekcije lica, pokreta ruku i različitih objekata, stereo vid i alate za proširenu stvarnost. Za izradu ove aplikacije korišteni su alati za detekciju rubova, ispravljanje perspektive, rad sa slikama i različiti filteri.

3.3. DIRECTSHOW.NET

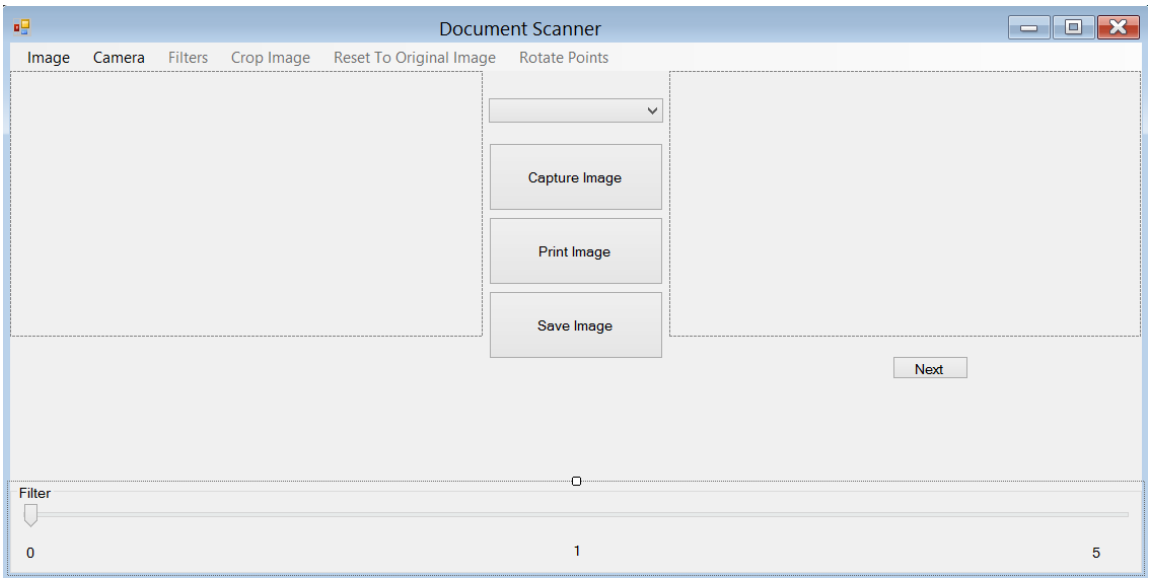
DirectShow.NET je biblioteka koja pruža *DirectShow* funkcionalnosti unutar *.NET* aplikacija. *DirectShow* programsko je sučelje za medijsku arhitekturu, osmišljeno u *Microsoft-u*. Korištenjem istoga, aplikacije mogu snimati ili puštati zvukove i videe u visokoj kvaliteti. U ovoj aplikaciji biblioteka se koristi za jednostavnije stvaranje i dohvaćanje liste dostupnih kamera, kao i upravljanje istom.

3.4. WINDOWS FORMS

Windows Forms Microsoft-ov je softverski okvir za izradu korisničkog sučelja. Platforma pruža širok spektar razvojnih značajki koje uključuju alate za razvoj kontrole, grafike i korisničkog unosa. Za potrebe ove aplikacije korišteni su alati za izradu jednostavnog korisničkog sučelja koje omogućava lakše korištenje aplikacije.

4. IMPLEMENTACIJA SKENERA DOKUMENATA

Unutar ovoga poglavlja opisan je proces izrade desktop aplikacije, kao i same funkcionalnosti koje aplikacija pruža.



Slika 4.1. Izgled sučelja unutar Microsoft Visual Studio-a

Grafičko sučelje izrađeno je korištenjem *Windows Forms* (Slika 4.1.) platforme i predstavljeno je klasom *Form1* koja nasljeđuje klasu *Form*. Klasa *Form1* sastoji se od atributa (Slika 4.2.) koji su potrebni za rad s kamerom (detaljnije opisano u potpoglavlju 4.1.), ispravljanje perspektive (detaljnije opisano u potpoglavlju 4.3.) i poboljšanje prikaza skenirane slike (detaljnije opisano u potpoglavlju 4.4.). Osim atributa, klasa *Form1* sadrži metode za rad sa sučeljem, kao i metode za obradu događaja unutar sučelja.

```
public partial class Form1 : Form
{
    VideoCapture capture;
    DocumentScanner documentScanner;
    Video_Device[] webCams;
    public Point mPoint1, mPoint2, mPoint3, mPoint4;
    int cameraDevice, mPointMoveInProgress, brightness, contrast;
    bool isImageCaptured, hasMoved, isCropOn;
```

Slika 4.2. Atributi klase *Form1*

4.1. ULAZNI PODATCI

Ulazni podatci aplikacije predstavljeni su slikom koja može biti dohvaćena s računala ili direktno s kamere. Prilikom pokretanja aplikacije (Slika 4.3.) stvara se lista dostupnih kamera koja se predaje korisniku na odabir putem propadajućeg izbornika. Za dohvaćanje liste svih dostupnih kamera koristi se statička metoda *DsDevice.GetDevicesOfCat* koja na temelju predanog jedinstvenog globalnog identifikatora *FilterCategory.VideoInputDevice*, filtrira i dohvaća sve dostupne kamere unutar sustava. Zatim se za svaku dohvaćenu kameru stvara objekt strukture *Video_Device* i dodaje se na listu *webCams* koja predstavlja listu svih dostupnih kamera unutar sustava pohranjenu u *Video_Device* formatu.

```
private void Form1_Load(object sender, EventArgs e)
{
    DsDevice[] systemCameras = DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice);
    webCams = new Video_Device[systemCameras.Length];

    for (int cameraIndex = 0; cameraIndex < systemCameras.Length; cameraIndex++)
    {
        webCams[cameraIndex] = new Video_Device(cameraIndex, systemCameras[cameraIndex].Name,
            systemCameras[cameraIndex].CLSID);

        cameraSelection.Items.Add(systemCameras[cameraIndex].Name);
    }

    if (cameraSelection.Items.Count > 0)
    {
        cameraSelection.SelectedIndex = 0;
    }
}
```

Slika 4.3. Metoda *Form1_Load* koja se poziva pri učitavanju aplikacije

Struktura *Video_Device* (Slika 4.4.) se sastoji od atributa *Device_Name* tipa *string* koji predstavlja naziv kamere, *Device_ID* tipa *int* koji predstavlja redni broj kamere u sustavu i *Identifier* tipa *Guid* koji predstavlja jedinstveni globalni identifikator kamere. Klasa sadrži i metodu *ToString* koja ne prima parametre, a vraća jednostavni zapis atributa objekta u *string* formatu.

```
struct Video_Device
{
    public string Device_Name;
    public int Device_ID;
    public Guid Identifier;

    1 reference
    public Video_Device(int ID, string Name, Guid Identity = new Guid())
    {
        Device_ID = ID;
        Device_Name = Name;
        Identifier = Identity;
    }

    – references
    public override string ToString()
    {
        return String.Format("[{0}] {1}: {2}", Device_ID, Device_Name, Identifier);
    }
}
```

Slika 4.4. Struktura *Video_Device*

Nakon što je odabrana kamera iz propadajućeg izbornika (Slika 4.5.), poziva se metoda *SetupCapture* (Slika 4.5.) koja prima redni broj kamere. Ona postavlja odabranu kameru.

```
private void cameraSelection_SelectedIndexChanged(object sender, EventArgs e)
{
    SetupCapture(cameraSelection.SelectedIndex);
}
2 references
private void SetupCapture(int cameraID)
{
    try
    {
        cameraDevice = cameraID;

        if (capture != null)
        {
            capture.Dispose();
            capture = null;
        }

        capture = new VideoCapture(cameraDevice, VideoCapture.API.DShow);
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
    }
}
```

Slika 4.5. Metode *cameraSelection_SelectedIndexChanged* i *SetupCapture*

Kamera se pokreće pritiskom na gumb Camera -> Start (Slika 4.6.). Nakon toga se aktiviraju gumbovi za kontrolu kamere korištenjem funkcije *ToggleCamerControls* i onemogućuju nepotrebni gumbovi. Postavlja se odabrana kamera i njezina rezolucija na maksimalnu moguću. Zatim započinje prijenos slike s kamere korištenjem svojstva *ImageGrabbed* kojemu se predaje metoda *Capture_ImageGrabbed* za obradu dohvaćene slike. Ako je kamera uspješno pokrenuta, omogućuje se gumb „Capture“ koji služi za slikanje, a atributi *mPoint1*, *mPoint2*, *mPoint3*, *mPoint4* tipa *Point*, koji će služiti za ispravljanje perspektive i izrezivanje slike, postavljaju se na svoje zadane vrijednosti.

```
private void startCapture_Click(object sender, EventArgs e)
{
    ToggleCameraControls();
    perspectiveButton.Visible = false;
    saveImage.Visible = false;
    printImage.Visible = false;
    filtersToolStripMenuItem.Enabled = false;
    cropImageToolStripMenuItem.Enabled = false;
    resetToOriginalImageToolStripMenuItem.Enabled = false;

    if (capture == null)
    {
        SetupCapture(cameraSelection.SelectedIndex);
    }
    capture.Set(CapProp.FrameWidth, 10000);
    capture.Set(CapProp.FrameHeight, 10000);

    capture.ImageGrabbed += Capture_ImageGrabbed;
    capture.Start();


    if (capture.IsOpened)
    {
        captureImage.Visible = true;
    }

    mPoint1 = new Point(0, 0);
    mPoint2 = new Point(0, resultBoxView.Height);
    mPoint3 = new Point(resultBoxView.Width, resultBoxView.Height);
    mPoint4 = new Point(resultBoxView.Width, 0);
    resultBoxView.Image = null;
}
```

Slika 4.6. Metoda *startCapture_Click* koja se poziva klikom na „Start“ gumb

Metoda *Capture_ImageGrabbed* (Slika 4.7.) dohvaća sliku iz kamere i postavlja ju u okvir za pregled slike kamere. Ujedno provjerava i je li pritisnut gumb „Capture“ za slikanje i ako je sprema sliku pomoću objekta *documentScanner* koji se nalazi unutar klase *Form1*. Poziva se metoda *detectPaper* nad objektom *documentScanner* koja služi za detekciju papira na slici. Funkcija *using* se koristi radi sprječavanja curenja memorije koje bi se događalo zbog gomilanja objekata *cameraImg* koji su korišteni za pohranu dohvaćenih slika tijekom aktivnog snimanja. Nakon završetka funkcije, memorija se automatski oslobađa. Za istu svrhu korištena je *Dispose()* metoda nad svojstvom *imageBoxView.Image* koji predstavlja sliku okvira za prikaz slike dohvaćene iz kamere.

```
private void Capture_ImageGrabbed(object sender, EventArgs e)
{
    try
    {
        using (Mat cameraImg = new Mat())
        {
            capture.Retrieve(cameraImg);

            if (imageBoxView.Image != null) imageBoxView.Image.Dispose();
            imageBoxView.Image = cameraImg.ToBitmap();

            if (isImageCaptured)
            {
                isImageCaptured = !isImageCaptured;
                documentScanner.CapturedImage = cameraImg.ToBitmap();
                documentScanner.ResultImage = cameraImg.ToBitmap();
                documentScanner.OriginalImage = cameraImg.ToBitmap();

                capture.Dispose();
                capture = null;

                documentScanner.detectPaper();
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
    }
}
```

Slika 4.7. Metoda *Capture_ImageGrabbed*

Klasa *DocumentScanner* (Slika 4.8.) predstavlja skener dokumenata i sve njegove funkcionalnosti. Sadrži svojstva u koja se pohranjuju dohvaćena slika (*CapturedImage*), konačna slika (*ResultImage*) i originalna slika poslije ispravljanja perspektive (*OriginalImage*). Osim ovih svojstava, klasa sadrži i objekt *MyForm* tipa *Form1* koji predstavlja sučelje i njegova je funkcija lakše dohvaćanje elemenata sučelja. Klasa ima zadani konstruktor i konstruktor koji prima parametar *capturedImage* koji predstavlja uslikanu sliku i *form* koji predstavlja sučelje aplikacije. Osim navedenih svojstava, klasa sadrži i metode za detekciju i obradu uslikane slike. Instanca ove klase stvara se pri slikanju slike, odnosno pritiskom na gumb „Capture“ i čuva se unutar *Form1* objekta. Ostale metode klase opisane su u sljedećim potpoglavljima.

```
class DocumentScanner
{
    33 references
    public Bitmap CapturedImage { get; set; }
    19 references
    public Bitmap ResultImage { get; set; }
    4 references
    public Bitmap OriginalImage { get; set; }

    37 references
    public Form1 MyForm { get; private set; }
    0 references
    public DocumentScanner(){}

    2 references
    public DocumentScanner(Bitmap capturedImage, Form1 form)
    {
        CapturedImage = capturedImage;
        MyForm = form;
    }
```

Slika 4.8. Atributi i konstruktor klase DocumentScanner

Alternativni način je učitavanje slike s računala, a to je moguće pritiskom na gumb „Image“ (Slika 4.9.). Učitava se slika s računala i zatim se kreira instanca klase *DocumentScanner*, te mu se vrijednosti postavljaju na sličan način kao i kod slike uslikane kamerom.

```
private void imageMenu_Click(object sender, EventArgs e)
{
    filtersToolStripMenuItem.Enabled = false;
    cropImageToolStripMenuItem.Enabled = false;
    resetToOriginalImageToolStripMenuItem.Enabled = false;
    filterBox.Visible = false;

    mPoint1 = new Point(0, 0);
    mPoint2 = new Point(0, resultBoxView.Height);
    mPoint3 = new Point(resultBoxView.Width, resultBoxView.Height);
    mPoint4 = new Point(resultBoxView.Width, 0);

    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Filter = "Image Files(*.png;*.jpg)|*.png;*.jpg;|All files (*.*)|*.*";
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        if (!startCapture.Enabled)
        {
            ToggleCameraControls();
        }
        captureImage.Visible = false;
        printImage.Visible = false;
        saveImage.Visible = false;
        rotatePointsToolStripMenuItem.Enabled = true;

        resultBoxView.Image = null;

        if (capture != null)
        {
            capture.Dispose();
            capture = null;
            resultBoxView.Image = null;
        }

        Mat img = CvInvoke.Imread(ofd.FileName);
        documentScanner = new DocumentScanner(img.ToBitmap(), this);
        documentScanner.ResultImage = img.ToBitmap();

        perspectiveButton.Visible = true;
        imageBoxView.Image = new Bitmap(documentScanner.CapturedImage, resultBoxView.Size);

        documentScanner.detectPaper();
    }
}
```

Slika 4.9. Metoda *imageMenu_Click*

4.2. DETEKCIJA PAPIRA

Metoda *detectPaper* koja je pozvana na objektu *documentScanner* (Slika 4.10.) služi za detektiranje papira unutar uslikane slike. Poziva metodu *DetectEdges* koja prima parametre *sourceImage* koji predstavlja uslikanu sliku u *Mat* obliku, referencu na objekt *contours* tipa *VectorOfVectorOfPoint* u koji se spremaju detektirane konture i *hierarchy* u koji se spremaju topografski odnosi detektiranih kontura, odnosno koja kontura je dijete, a koja roditelj. Nakon što su pronađene sve konture, poziva se metoda *FindBiggestContour*. Ta metoda traži najveću pravokutnu konturu i sprema je u obliku objekta *paperVector*. Zatim se, ako je pronađena ijedna takva kontura, točke vrhova pohranjuju unutar atributa *mPoint1*, *mPoint2*, *mPoint3* i *mPoint4* koje pripadaju klasi *Form1* i poziva se metoda *drawMovablePoints* koja u okviru za pregled obrađene slike crta pokretne točke koje će služiti kao pomoć pri korigiranju detektiranih vrhova kod ispravljanje perspektive (više pojašnjeno u potpoglavlju 4.3.)

```
public void detectPaper()
{
    Mat sourceImage = CapturedImage.ToMat();
    Mat hierarchy = new Mat();
    VectorOfVectorOfPoint contours = new VectorOfVectorOfPoint();

    double maxArea = 0.0;

    DetectEdges(sourceImage, ref contours, ref hierarchy);
    VectorOfPoint paperVector = FindBiggestContour(contours, ref maxArea);

    if (maxArea != 0.0)
    {
        MyForm.mPoint1.X = (int)((paperVector[0].X / (double)CapturedImage.Width)
            * MyForm.resultBoxView.Width);
        MyForm.mPoint1.Y = (int)((paperVector[0].Y / (double)CapturedImage.Height)
            * MyForm.resultBoxView.Height);

        MyForm.mPoint2.X = (int)((paperVector[1].X / (double)CapturedImage.Width)
            * MyForm.resultBoxView.Width);
        MyForm.mPoint2.Y = (int)((paperVector[1].Y / (double)CapturedImage.Height)
            * MyForm.resultBoxView.Height);

        MyForm.mPoint3.X = (int)((paperVector[2].X / (double)CapturedImage.Width)
            * MyForm.resultBoxView.Width);
        MyForm.mPoint3.Y = (int)((paperVector[2].Y / (double)CapturedImage.Height)
            * MyForm.resultBoxView.Height);

        MyForm.mPoint4.X = (int)((paperVector[3].X / (double)CapturedImage.Width)
            * MyForm.resultBoxView.Width);
        MyForm.mPoint4.Y = (int)((paperVector[3].Y / (double)CapturedImage.Height)
            * MyForm.resultBoxView.Height);
    }
    drawMovablePoints();
}
```

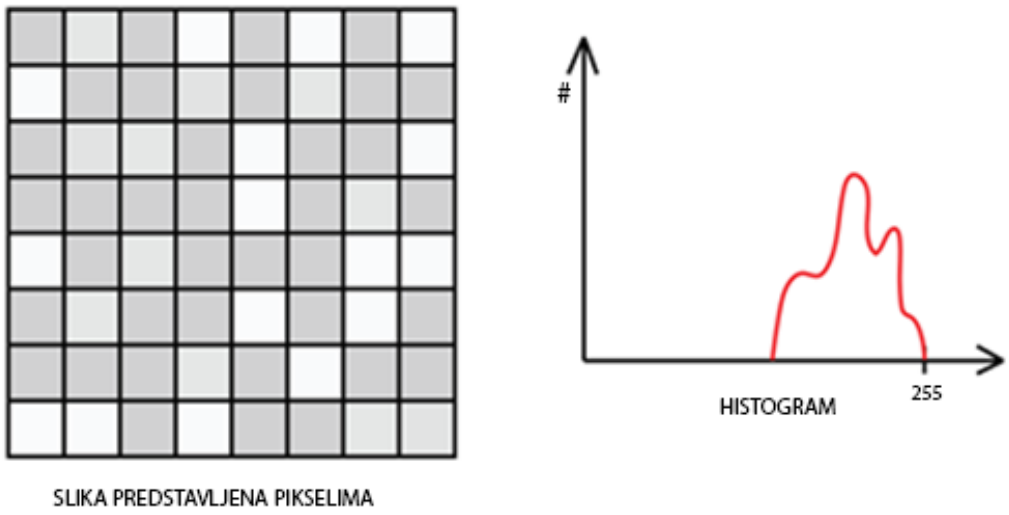
Slika 4.10. Metoda *detectPaper*

Metoda *DetectEdges* (Slika 4.11.) služi za detektiranje rubova na uslikanoj slici. Prvo se slika prebacuje u crno-bijelu boju, a zatim se poziva metoda *ThresholdBinary*. Metoda svakom pikselu provjerava intenzitet i ako je isti veći od zadanog praga (prvi parametar), postavlja njegovu vrijednost na zadani intenzitet (drugi parametar). Nakon toga se na istoj slici vrši postupak kontrastno-ograničenog adaptivnog izjednačavanja histograma (engl. *CLAHE* [12]) statičkom metodom *CvInvoke.CLAHE*. Svrha ovoga je poboljšanje kontrasta slike radi lakšeg detektiranja papira.

```
private void DetectEdges(Mat img, ref VectorOfVectorOfPoint contours, ref Mat hierarchy)
{
    Mat gaussianBlurredImage = new Mat();
    Mat cannyImage = new Mat();
    Mat claheImage = new Mat();

    Image<Gray, Byte> grayscaleImage = img.ToImage<Gray, Byte>()
        .ThresholdBinary(new Gray(100), new Gray(255));

    CvInvoke.CLAHE(grayscaleImage, 5, new Size(16, 16), claheImage);

    CvInvoke.GaussianBlur(claheImage, gaussianBlurredImage, new Size(5, 5), 1);

    CvInvoke.Canny(gaussianBlurredImage, cannyImage,
        Constants.lowerThreshold, Constants.upperThreshold);

    CvInvoke.FindContours(cannyImage, contours, hierarchy, RetrType.External,
        ChainApproxMethod.ChainApproxSimple);
    CvInvoke.BitwiseNot(cannyImage, cannyImage);

    MyForm.imageBoxView.Image = cannyImage.ToBitmap();
}
```

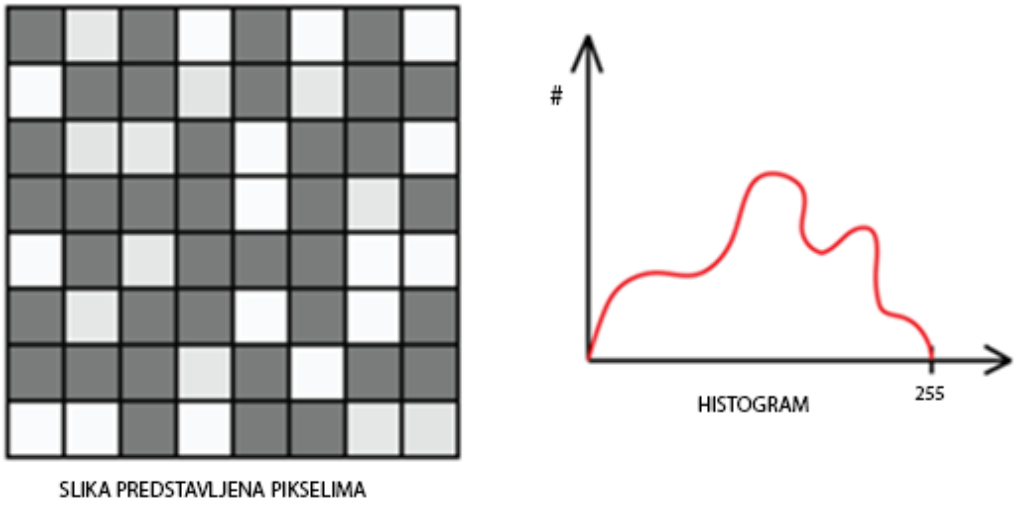
Slika 4.11. Metoda *DetectEdges*

CLAHE algoritam naprednija je verzija algoritma za adaptivno izjednačavanje histograma (*engl. adaptive histogram equalization*). Slika se sastoji od velikog broja malih kvadratića (piksela). Oni su, ovisno o bojama i vrsti slike, definirani numeričkim vrijednostima koje označavaju intenzitet. Za crno-bijelu sliku te vrijednosti su ograničene na raspon između 0 i 255 gdje 0 predstavlja crnu boju, a 255 bijelu boju. Na temelju zadane slike, stvara se histogram koji prikazuje frekvenciju pojedinih intenziteta na slici. Radi jednostavnosti, pretpostavljeno je da je zadana slika svijetla (Slika 4.12.). Njen histogram ima vrijednosti samo na desnoj strani grafa. Gruba skica vidljiva je na slici 4.12.



Slika 4.12. Slika i gruba skica histograma te slike

Kada se na ovakvu sliku primijeni izjednačavanje histograma, dobiva se slika i histogram prikazan na slici 4.13. koji je izdužen po x-osi do zadane vrijednosti.



Slika 4.13. Slika i njen histogram nakon izjednačavanja histograma

Primjenom ovakvog procesa nastaje novi problem: nastala slika može biti previše mračna ili previše svijetla. Problem se rješava tako da se zadana slika podijeli na više manjih dijelova (najčešće veličine 8x8 piksela) te se na svakome od njih napravi izjednačavanje histograma. Ovakav proces naziva se adaptivno izjednačavanje histograma. No, ponovno se pojavljuje novi

problem: ako na zadanoj slici postoje šumovi, oni će ovim procesom biti pojačani. Zbog toga se primjenjuje ograničavanje kontrasta na zadanom području, a cjelokupni proces naziva se kontrastno-ograničeno adaptivno izjednačavanje histograma. Na dobivenoj slici lakše je detektirati papir. Na primjeru prikazanom na slici 4.14. vidljiva je razlika između algoritama. Na prvoj slici je originalna slika, na drugoj je primijenjeno izjednačavanje histograma (lišće palme i dalje je tamno i teško se razlikuju detalji), te na trećoj kontrastno-ograničeno adaptivno izjednačavanje histograma na kojoj je sve vidljivo.

Slika 4.14. Slike dobivene primjenom različitih algoritama [14]

Nakon toga se statičkom metodom *CvInvoke.GaussianBlur* primjenjuje Gaussov filter koji uklanjanja moguće točke šuma i zaglađuje sliku. Sljedeći korak je pozivanje statične funkcije *CvInvoke.Canny* [13] koja služi za detekciju rubova.

Canny algoritam sastoji se od više koraka:

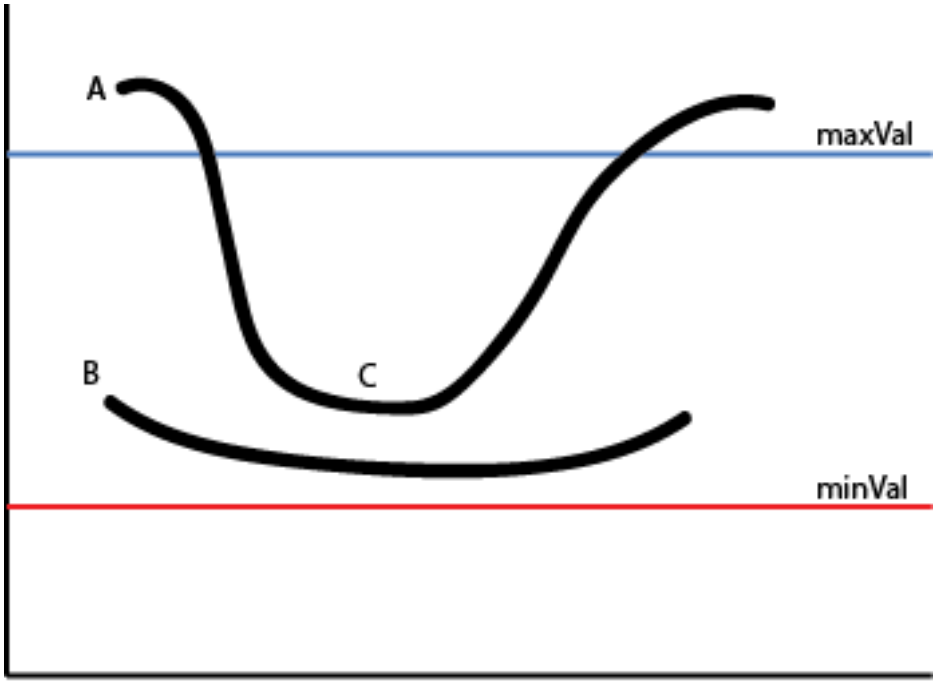
1) smanjivanje šuma
2) pronalaženje gradijenta intenziteta slike – zaglađena slika se filtrira pomoću Sobel kernela u vodoravnom i okomitom smjeru kako bi se dobila prva derivacija u vodoravnom smjeru (Gx) i okomitom smjeru (Gy). Iz ove dvije slike moguće je pronaći rubni gradijent i smjer za svaki piksel kako slijedi:

$$rubni\ gradijent\ (G) = \sqrt{G_x^2 + G_y^2} \tag{4-1}$$

$$kut\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{4-2}$$

Smjer gradijenta uvijek je okomit na rubove. Zaokružen je na jedan od četiri kuta koji predstavljaju okomiti, vodoravni i dva dijagonalna smjera.

3) ne-maksimalno potiskivanje - nakon dobivanja veličine i smjera gradijenta, vrši se potpuno provjeravanje slike kako bi se uklonili svi neželjeni pikseli koji možda ne čine rub. Za to se na svakom pikselu provjerava je li on lokalni maksimum u njegovom susjedstvu u smjeru gradijenta.
4) prag histereze – ovim korakom odlučuje se koji su rubovi stvarno rubovi, a koji nisu. Za ovo su potrebne dvije vrijednosti praga, minVal i maxVal. Svi rubovi s gradijentom intenziteta većim od maxVal sigurno su rubovi, a oni ispod minVal sigurno nisu rubovi, pa se odbacuju. Oni koji se nalaze između ova dva praga klasificirani su kao rubovi ili „ne-rubovi“ na temelju njihove povezanosti. Ako su spojeni na "sigurno-rubne" piksele, smatraju se dijelom ruba. Inače se i oni odbacuju. Primjer je prikazan na slici 4.15.



Slika 4.15. Primjer četvrtog koraka *Canny* algoritma

Rub A nalazi se iznad maxVal, pa je on „sigurno-rub“. Iako je rub C ispod maxVal, spojen je s rubom A, pa je on valjani rub. Ali rub B, iako je iznad minVal i nalazi se u istom području kao i rub C, nije spojen na nijedan „sigurno-rub“, pa se odbacuje. Ovaj korak također uklanja sitne šumove pa konačna slika ima izražene rubove. Na kraju se statičkom metodom *CvInvoke.FindContours* dohvaćaju konture rubova sa zadane slike.

Metoda *FindBiggestContour* (Slika 4.16.) za zadane konture vraća najveću pravokutnu tipa *VectorOfPoint*. Zatim prolazi kroz listu svih kontura, statičkom metodom *CvInvoke.ContourArea* izračunava površinu pojedine konture i samo ako su dovoljno velike provjerava jesu li pravokutne i jesu li veće od, do sad najveće pronađene, konture. Statička metoda *CvInvoke.ArcLength* za zadanu, zatvorenu konturu izračunava njen opseg i aproksimira ju s određenim stupnjem točnosti statičkom metodom *CvInvoke.ApproxPolyDP*. Ako je aproksimirani oblik pravokutnik, pohranjuje najveću konturu i njenu površinu.

```
private VectorOfPoint FindBiggestContour(VectorOfVectorOfPoint contours, ref double maxArea)
{
    VectorOfPoint biggest = new VectorOfPoint();

    for (int contourIndex = 0; contourIndex < contours.Size; contourIndex++)
    {
        double area = CvInvoke.ContourArea(contours[contourIndex]);
        if (area > 5000)
        {
            double perimiter = CvInvoke.ArcLength(contours[contourIndex], true);
            VectorOfPoint approximate = new VectorOfPoint();
            CvInvoke.ApproxPolyDP(contours[contourIndex], approximate, 0.02 * perimiter, true);

            if (area > maxArea && approximate.Size == 4)
            {
                if (Utility.IsRectangle(approximate))
                {
                    biggest = approximate;
                    maxArea = area;
                }
            }
        }
    }
    return biggest;
}
```

Slika 4.16. Metoda *FindBiggestContour*

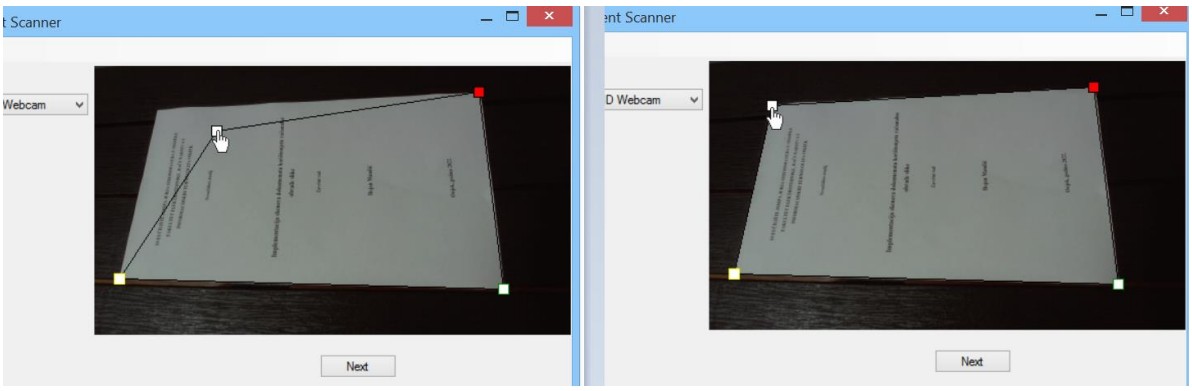
Provjera je li aproksimirana kontura pravokutnik vrši se pomoću statičke metode *Utility.IsRectangle* koja je dio statičke klase *Utility* (Slika 4.17.). Ona spaja točke u rubove statičkom metodom *PointCollection.PolyLine* i potom se provjerava jesu li kutevi lika u razumnom rasponu. Ako jesu, taj lik je pravokutnik.

```
public static class Utility
{
    1 reference
    public static bool IsRectangle(VectorOfPoint approxContour)
    {
        Point[] pts = approxContour.ToArray();
        LineSegment2D[] edges = PointCollection.PolyLine(pts, true);

        for (int j = 0; j < edges.Length; j++)
        {
            double angle = Math.Abs(
                edges[(j + 1) % edges.Length].GetExteriorAngleDegree(edges[j]));
            if (angle < 35 || angle > 145)
            {
                return false;
            }
        }
        return true;
    }
}
```

Slika 4.17. Statička metoda *IsRectangle*

4.3. ISPRAVLJANJE PERSPEKTIVE

Kao što je prethodno spomenuto, metoda *drawMovablePoints* (Slika 4.19.) na prozoru za prikaz obrađene slike crta pomične točke na vrhovima detektiranog papira koje služe za korigiranje mogućih pogrešaka u detektiranim rubovima. U slučaju da na slici nije pronađen papir, točke se crtaju na krajnjim rubovima slike. Metoda prvo crta pravokutnik u vrhovima detektiranog papira, a zatim u svakom pojedinom vrhu kvadrat koji služi za lakše uočavanje i pomicanje vrhova. Za crtanje se koristi klasa *Graphics*. Primjer korigiranja točaka prikazan je na slici 4.18.



Slika 4.18. Primjer ručnog ispravljanja pogrešno detektiranog obrisa papira

```
4 references
public void drawMovablePoints()
{
    Pen blackPen = new Pen(Color.Black, 1);
    Bitmap resizedImage = new Bitmap(ResultImage, new Size(MyForm.resultBoxView.Width, MyForm.resultBoxView.Height));

    using (var graphics = Graphics.FromImage(resizedImage))
    {
        graphics.DrawLine(blackPen, MyForm.mPoint4, MyForm.mPoint1);
        graphics.DrawLine(blackPen, MyForm.mPoint1, MyForm.mPoint2);
        graphics.DrawLine(blackPen, MyForm.mPoint2, MyForm.mPoint3);
        graphics.DrawLine(blackPen, MyForm.mPoint3, MyForm.mPoint4);

        Rectangle rectangle;
        rectangle = new Rectangle(MyForm.mPoint1.X - Constants.handleRadius, MyForm.mPoint1.Y - Constants.handleRadius,
            Constants.handleRadius * 2, Constants.handleRadius * 2);

        graphics.FillRectangle(Brushes.Red, rectangle);
        graphics.DrawRectangle(Pens.Black, rectangle);

        rectangle = new Rectangle(MyForm.mPoint2.X - Constants.handleRadius, MyForm.mPoint2.Y - Constants.handleRadius,
            Constants.handleRadius * 2, Constants.handleRadius * 2);

        graphics.FillRectangle(Brushes.White, rectangle);
        graphics.DrawRectangle(Pens.Black, rectangle);

        rectangle = new Rectangle(MyForm.mPoint3.X - Constants.handleRadius, MyForm.mPoint3.Y - Constants.handleRadius,
            Constants.handleRadius * 2, Constants.handleRadius * 2);

        graphics.FillRectangle(Brushes.White, rectangle);
        graphics.DrawRectangle(Pens.Yellow, rectangle);

        rectangle = new Rectangle(MyForm.mPoint4.X - Constants.handleRadius, MyForm.mPoint4.Y - Constants.handleRadius,
            Constants.handleRadius * 2, Constants.handleRadius * 2);


        graphics.FillRectangle(Brushes.White, rectangle)
        graphics.DrawRectangle(Pens.Green, rectangle);
    }
    MyForm.resultBoxView.Image = resizedImage;
}
```

Slika 4.19. Metoda *drawMovablePoints*

Funkcionalnost pomicanja točaka implementirana je metodama *resultBoxView_MouseDown*, *resultBoxView_MouseMove*, *resultBoxView_MouseUp* i *resultBoxView_Paint*. Osim za korigiranje rubova detektiranog papira, metoda *drawMovablePoints* koristi se i za izrezivanje slike (detaljnije opisano u poglavlju 4.4.).

Metodom *resultBoxView_MouseDown* (Slika 4.20.) određuje se na koju točku je korisnik kliknuo te se redoslijed točke pohranjuje u varijablu *mPointMoveInProgress* koja se nalazi u klasi *Form1*. Metodom *resultBoxView_MouseUp* (Slika 4.20.) poništava se vrijednost atributa *mPointMoveInProgress* kada korisnik pusti lijevi klik miša. Pomicanjem miša preko okvira za prikaz obrađene slike poziva se metoda *resultBoxView_MouseMove* (Slika 4.20.). Ako je korisnik kliknuo na neku od točaka i pomaknuo ju mišem, koordinate tog pomaka zapisuju se u atribute *mPoint1*, *mPoint2*, *mPoint3* i *mPoint4* ovisno o točki koja je pomaknuta. Pomicanje

točaka ograničeno je unutar okvira slike te iste nije moguće postaviti izvan. Prelaskom miša preko točke mijenja se oblik pokazivača radi lakšeg uočavanja točaka na slici. Na slici 4.20. izostavljen je dio koda koji se ponavlja za točku 3 jer je napisan istom logikom kao i točke 1,2 i 4.

```
private void resultBoxView_MouseDown(object sender, MouseEventArgs e)
{
    if (Math.Abs(e.X - mPoint1.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint1.Y) < Constants.handleRadius)
        mPointMoveInProgress = 1;

    else if (Math.Abs(e.X - mPoint2.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint2.Y) < Constants.handleRadius)
        mPointMoveInProgress = 2;

    else if (Math.Abs(e.X - mPoint3.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint3.Y) < Constants.handleRadius)
        mPointMoveInProgress = 3;

    else if (Math.Abs(e.X - mPoint4.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint4.Y) < Constants.handleRadius)
        mPointMoveInProgress = 4;

    else mPointMoveInProgress = 0;
}
1 reference
private void resultBoxView_MouseUp(object sender, MouseEventArgs e)
{
    mPointMoveInProgress = 0;
}
1 reference
private void resultBoxView_MouseMove(object sender, MouseEventArgs e)
{

    if (mPointMoveInProgress == 1)
    {
        if (isCropOn)
        {
            hasMoved = true;
            mPoint1.X = (e.X < 0) ? 0 : (e.X > resultBoxView.Width) ? resultBoxView.Width : e.X;
            mPoint1.Y = (e.Y < 0) ? 0 : (e.Y > resultBoxView.Height) ? resultBoxView.Height : e.Y;
            mPoint2.X = mPoint1.X;
            mPoint4.Y = mPoint1.Y;
            Refresh();
        }
        else
        {
            hasMoved = true;
            mPoint1.X = (e.X < 0) ? 0 : (e.X > resultBoxView.Width) ? resultBoxView.Width : e.X;
            mPoint1.Y = (e.Y < 0) ? 0 : (e.Y > resultBoxView.Height) ? resultBoxView.Height : e.Y;
            Refresh();
        }
    }
    else if (mPointMoveInProgress == 2)
    {
        if (isCropOn)
        {
            hasMoved = true;
            mPoint2.X = (e.X < 0) ? 0 : (e.X > resultBoxView.Width) ? resultBoxView.Width : e.X;
            mPoint2.Y = (e.Y < 0) ? 0 : (e.Y > resultBoxView.Height) ? resultBoxView.Height : e.Y;
            mPoint1.X = mPoint2.X;
            mPoint3.Y = mPoint2.Y;
            Refresh();
        }
        else
        {
            hasMoved = true;
            mPoint2.X = (e.X < 0) ? 0 : (e.X > resultBoxView.Width) ? resultBoxView.Width : e.X;
            mPoint2.Y = (e.Y < 0) ? 0 : (e.Y > resultBoxView.Height) ? resultBoxView.Height : e.Y;
            Refresh();
        }
    }
```

```
    else if (mPointMoveInProgress == 4)
    {
        if (isCropOn)
        {
            hasMoved = true;
            mPoint4.X = (e.X < 0) ? 0 : (e.X > resultBoxView.Width) ? resultBoxView.Width : e.X;
            mPoint4.Y = (e.Y < 0) ? 0 : (e.Y > resultBoxView.Height) ? resultBoxView.Height : e.Y;
            mPoint3.X = mPoint4.X;
            mPoint1.Y = mPoint4.Y;
            Refresh();
        }
        else
        {
            hasMoved = true;
            mPoint4.X = (e.X < 0) ? 0 : (e.X > resultBoxView.Width) ? resultBoxView.Width : e.X;
            mPoint4.Y = (e.Y < 0) ? 0 : (e.Y > resultBoxView.Height) ? resultBoxView.Height : e.Y;
            Refresh();
        }
    }
    else
    {
        if (Math.Abs(e.X - mPoint1.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint1.Y) < Constants.handleRadius)
            Cursor.Current = Cursors.Hand;

        else if (Math.Abs(e.X - mPoint2.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint2.Y) < Constants.handleRadius)
            Cursor.Current = Cursors.Hand;

        else if (Math.Abs(e.X - mPoint3.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint3.Y) < Constants.handleRadius)
            Cursor.Current = Cursors.Hand;

        else if (Math.Abs(e.X - mPoint4.X) < Constants.handleRadius && Math.Abs(e.Y - mPoint4.Y) < Constants.handleRadius)
            Cursor.Current = Cursors.Hand;

        else Cursor.Current = Cursors.Default;
    }
}
```

Slika 4.20. Metode *resultBoxView_MouseDown*, *resultBoxView_MouseUp* i *resultBoxView_MouseMove*

Metodom *Refresh* osvježava se sva grafika prozora i ponovno se crtaju elementi. Na taj događaj poziva se metoda *resultBoxView_Paint* (Slika 4.21.) koja ponovno crta vrhove papira, ali na novim koordinatama.

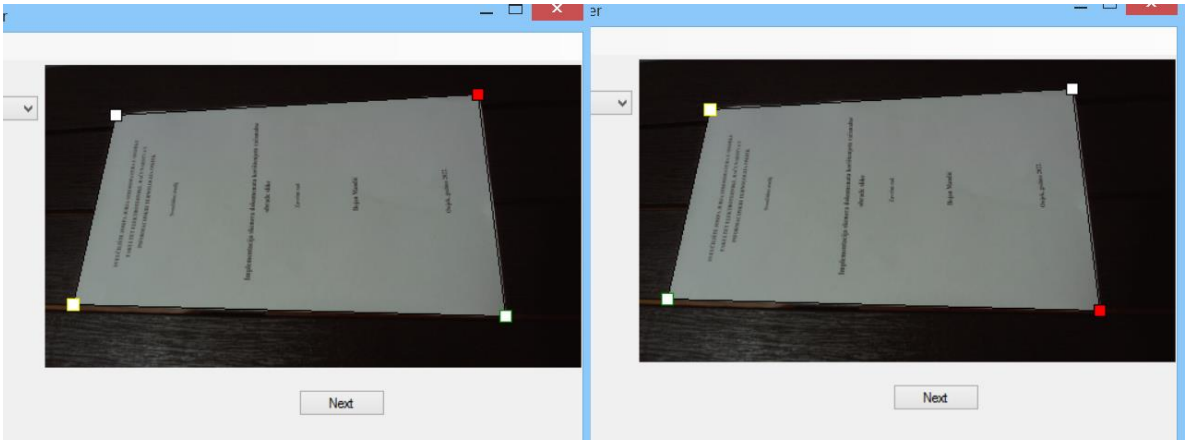
```
private void resultBoxView_Paint(object sender, PaintEventArgs e)
{
    if (documentScanner != null)
    {
        if (documentScanner.CapturedImage != null && hasMoved)
        {
            hasMoved = false;
            documentScanner.drawMovablePoints();
        }
    }
}
```

Slika 4.21. Metoda *resultBoxView_Paint*

Nakon što je korisnik zadovoljan odabranim rubovima i vrhovima papira, pritiskom na gumb „Rotate Points“ (Slika 4.22.) potrebno je rotirati točke tako da crvena točka bude u gornjem lijevom rubu ako je dokument pejsažno orijentiran, ili desno gore ako je dokument orijentiran portretno. Ako korisnik to ne učini, dokument će biti okrenut naopako. Metoda *RotatePoints* (Slika 4.22.) rotira točke u smjeru kazaljke na satu. Primjer rotiranja točaka prikazan je na slici 4.23.

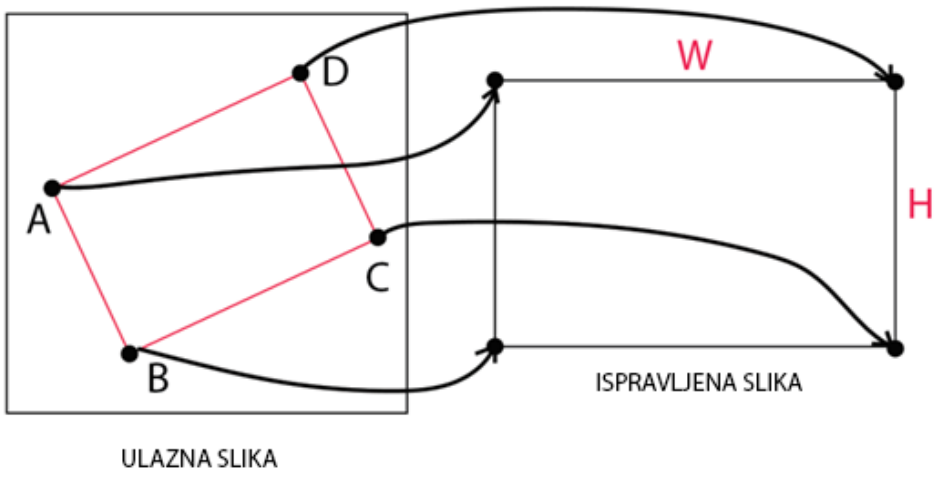
```
private void rotatePointsToolStripMenuItem_Click(object sender, EventArgs e)
{
    RotatePoints();
    documentScanner.drawMovablePoints();
}

1 reference
private void RotatePoints()
{
    Point tempPoint = new Point(mPoint1.X, mPoint1.Y);
    mPoint1 = new Point(mPoint4.X, mPoint4.Y);
    mPoint4 = new Point(mPoint3.X, mPoint3.Y);
    mPoint3 = new Point(mPoint2.X, mPoint2.Y);
    mPoint2 = new Point(tempPoint.X, tempPoint.Y);
}
```

Slika 4.22. Metode *rotatePointsToolStripMenuItem_Click* i *RotatePoints*



Slika 4.23. Primjer rotiranja točaka pritiskom na gumb „Rotate Points“

Pritiskom na gumb „Next“ (Slika 4.25.), odabrane točke na okviru za prikaz obrađene slike se pretvaraju u referentne točke na uslikanoj slici i pohranjuju u listu točaka *points* tipa *PointF*. Ovaj korak je potreban budući da se razlikuju rezolucije okvira i stvarne slike, a koordinate ucrtane na okviru za prikaz obrađene slike ne odgovaraju točkama stvarne slike. Određuju se stvarne dimenzije papira koje su potrebne za ispravljanje perspektive, pohranjuju kao vrhovi u listu točaka *maxPoints* tipa *PointF* te se određuju dimenzije papira koristeći veličine stranica papira. Poziva se statička metoda *CvInvoke.GetPerspectiveTransform* koja za zadane vrhove papira i vrhove željene slike, vraća matricu transformacije perspektive. Matrica se sastoji od parametara za transformaciju (npr. rotacija, skaliranje), translacijskog vektora i projekcijskog vektora. Konačna slika s ispravljenom perspektivom dobiva se pozivom statičke metode *CvInvoke.WarpPerspective* koja na temelju dobivene matrice transformacije i željenih dimenzija slike vrši transformaciju perspektive. Proces je ugrubo prikazan na slici 4.24. Ispravljena slika zatim se prikazuje na okviru za prikaz obrađene slike. Gumb „Next“ još se koristi i za izrezivanje slike (više o tome u potpoglavlju 4.4.)



Slika 4.24. Proces ispravljanja perspektive

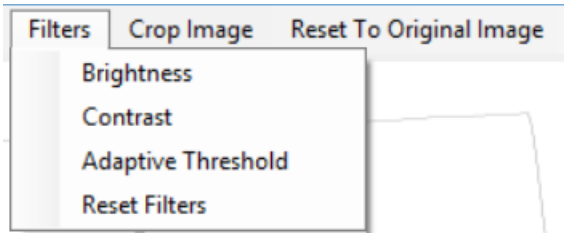
```
private void perspectiveButton_Click(object sender, EventArgs e)
{
    resetToOriginalImageToolStripMenuItem.Enabled = true;
   if (isCropOn)
    {
        documentScanner.CropPhoto();
        isCropOn = false;
        perspectiveButton.Visible = false;
    }
   else
    {
        rotatePointsToolStripMenuItem.Enabled = false;
        perspectiveButton.Visible = false;
        filtersToolStripMenuItem.Enabled = true;
        cropImageToolStripMenuItem.Enabled = true;

        PointF relativePoint1 = new PointF((int)((mPoint1.X / (double)resultBoxView.Width) * documentScanner.CapturedImage.Width),
            (int)((mPoint1.Y / (double)resultBoxView.Height) * documentScanner.CapturedImage.Height));

        PointF relativePoint2 = new PointF((int)((mPoint2.X / (double)resultBoxView.Width) * documentScanner.CapturedImage.Width),
            (int)((mPoint2.Y / (double)resultBoxView.Height) * documentScanner.CapturedImage.Height));

        PointF relativePoint3 = new PointF((int)((mPoint3.X / (double)resultBoxView.Width) * documentScanner.CapturedImage.Width),
            (int)((mPoint3.Y / (double)resultBoxView.Height) * documentScanner.CapturedImage.Height));

        PointF relativePoint4 = new PointF((int)((mPoint4.X / (double)resultBoxView.Width) * documentScanner.CapturedImage.Width),
            (int)((mPoint4.Y / (double)resultBoxView.Height) * documentScanner.CapturedImage.Height));

        PointF[] points = new PointF[] { relativePoint1, relativePoint2, relativePoint3, relativePoint4 };

        double width_AD = Math.Sqrt(Math.Pow(relativePoint1.X - relativePoint4.X, 2)
            + Math.Pow(relativePoint1.Y - relativePoint4.Y, 2));

        double width_BC = Math.Sqrt(Math.Pow(relativePoint2.X - relativePoint3.X, 2)
            + Math.Pow(relativePoint2.Y - relativePoint3.Y, 2));

        int maxWidth = (int)Math.Max(width_AD, width_BC);

        double height_AB = Math.Sqrt(Math.Pow(relativePoint1.X - relativePoint2.X, 2)
            + Math.Pow(relativePoint1.Y - relativePoint2.Y, 2));

        double height_CD = Math.Sqrt(Math.Pow(relativePoint3.X - relativePoint4.X, 2)
            + Math.Pow(relativePoint3.Y - relativePoint4.Y, 2));

        int maxHeight = (int)Math.Max(height_AB, height_CD);

        PointF[] maxPoints = new PointF[] { new PointF(0, 0), new PointF(0, maxHeight),
            new PointF(maxWidth, maxHeight), new PointF(maxWidth, 0) };

        var matrix = CvInvoke.GetPerspectiveTransform(points, maxPoints);

        Mat warpedImage = new Mat();
        CvInvoke.WarpPerspective(documentScanner.CapturedImage.ToMat(), warpedImage, matrix, new Size(maxWidth, maxHeight));
        documentScanner.ResultImage = warpedImage.ToBitmap();
        documentScanner.CapturedImage = warpedImage.ToBitmap();
        documentScanner.OriginalImage = warpedImage.ToBitmap();

        resultBoxView.Image = new Bitmap(documentScanner.ResultImage, resultBoxView.Size);

        printImage.Visible = true;
        saveImage.Visible = true;
    }
}
```

Slika 4.25. Metoda *perspectiveButton_Click* koja se poziva pri kliku na gumb „Next“

4.4. POBOLJŠANJE PRIKAZA SKENIRANE SLIKE

Kvaliteta ispravljene slike može se poboljšati na više načina. Pritiskom na gumb „Filters“ moguće je odabrati različite filtere, a pritiskom na gumb „Crop Image“ isjeći sliku.

Pritiskom na jedan od gumbova filtera (Slika 4.26.) na sliku je moguće primijeniti jedan od tri filtera: osvjetljenje, kontrast ili prilagodljivo postavljanje praga i vratiti filtere na zadane vrijednosti.



Slika 4.26. Gumbovi za primjenu filtera

Na zaslonu se zatim pojavljuje klizač koji je moguće postaviti na željenu vrijednost intenziteta odabranog filtera. Klizač za promjenu osvjetljenja prikazan je na slici 4.27.

Slika 4.27. Klizač za promjenu osvjetljenja

Metoda *brightnessToolStripMenuItem_Click* (Slika 4.28.) odgovorna je za promjenu natpisa na grafičkom sučelju i prikaz spremljene vrijednosti varijable *brightness* koja označava postavljeno osvjetljenje, a metoda *contrastToolStripMenuItem_Click* (Slika 4.28.) odgovorna je za kontrast. Metoda *adaptiveThresholdToolStripItem_Click* (Slika 4.28.) primjenjuje adaptivno postavljanje praga na temelju je li ono označeno u grafičkom sučelju ili ne.

```
private void brightnessToolStripMenuItem_Click(object sender, EventArgs e)
{
    filterBox.Text = Constants.brightnessLabel;
    filterSlider.Value = brightness;
    filterSlider.Maximum = Constants.maxBrightness;
    currentSliderValue.Text = brightness.ToString();
    maxFliterValue.Text = Constants.maxBrightness.ToString();
    filterBox.Visible = true;
}

1 reference
private void contrastToolStripMenuItem_Click(object sender, EventArgs e)
{
    filterBox.Text = Constants.contrastLabel;
    filterSlider.Maximum = Constants.maxContrast;
    filterSlider.Value = contrast;
    currentSliderValue.Text = ((float)contrast / 100).ToString();
    maxFliterValue.Text = ((float)filterSlider.Maximum / 100).ToString();
    filterBox.Visible = true;
}

1 reference
private void adaptiveThresholdToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (!adaptiveThresholdToolStripMenuItem.Checked)
    {
        adaptiveThresholdToolStripMenuItem.Checked = true;
        documentScanner.ResultImage = documentScanner.AdaptiveThreshold();
    }
    else
    {
        adaptiveThresholdToolStripMenuItem.Checked = false;
        documentScanner.ContrastBrightnessAdjust(contrast, brightness);
    }
    resultBoxView.Image = new Bitmap(documentScanner.ResultImage, resultBoxView.Size);
}
```

Slika 4.28. Metode *brightnessToolStripMenuItem_Click*, *contrastToolStripMenuItem_Click* i *adaptiveThresholdToolStripMenuItem_Click*

Metoda *AdaptiveThreshold* (Slika 4.29.) pretvara sliku u crno-bijelu i zatim na njoj vrši proces adaptivnog postavljanja praga statičkom metodom *CvInvoke.AdaptiveThreshold*. Dobivena slika je predstavljena bijelim rubovima na crnoj pozadini pa ju je potrebno invertirati metodom *CvInvoke.BitwiseNot*, a na kraju se zaglađuje korištenjem *CvInvoke.MedianBlur*.

```
public Bitmap AdaptiveThreshold()
{
    Mat img = new Mat();
    CvInvoke.CvtColor(ResultImage.ToImage<Bgr, byte>(), img, ColorConversion.Bgr2Gray);
    CvInvoke.AdaptiveThreshold(img, img, 255, AdaptiveThresholdType.GaussianC, ThresholdType.BinaryInv, 7, 2);
    CvInvoke.BitwiseNot(img, img);
    CvInvoke.MedianBlur(img, img, 5);

    return img.ToBitmap();
}
```

Slika 4.29. Metoda *AdaptiveThreshold*

Metoda *resetToolStripMenuItem_Click* (Slika 4.30.) poziva metodu *ResetFilter* (Slika 4.30.). za postavljanje zadanih vrijednosti filtera i vraća sliku koja je dobivena ispravljanjem perspektive. Zadane vrijednosti pohranjene su unutar statičke klase *Constants*, unutar koje su pohranjeni i natpisi za grafičko sučelje. Klasa *Constants* prikazana je na slici 4.31.

```
private void resetToolStripMenuItem_Click(object sender, EventArgs e)
{
    ResetFilter();
    documentScanner.ResultImage = new Bitmap(documentScanner.CapturedImage);
    resultBoxView.Image = new Bitmap(documentScanner.CapturedImage, resultBoxView.Size);
}
2 references
private void ResetFilter()
{
    brightness = Constants.defaultBrightness;
    contrast = Constants.defaultContrast;
    adaptiveThresholdToolStripMenuItem.Checked = false;

    if (filterBox.Text == Constants.contrastLabel)
    {
        filterSlider.Value = contrast;
        currentSliderValue.Text = ((float)contrast / 100).ToString();
    }
    else
    {
        filterSlider.Value = brightness;
        currentSliderValue.Text = brightness.ToString();
    }
}
```

Slika 4.30. Metoda *resetToolStripMenuItem_Click* i *ResetFilter*

```
public static class Constants
{
    public const int handleRadius = 5;
    public const int defaultBrightness = 0;
    public const int defaultContrast = 100;
    public const string brightnessLabel = "Brightness";
    public const string contrastLabel = "Contrast";
    public const int maxBrightness = 100;
    public const int maxContrast = 200;
    public const int lowerThreshold = 255 / 3;
    public const int upperThreshold = 255;

}
```

Slika 4.31. Statička klasa *Constants*

Metoda *filterSlider_Scroll* (Slika 4.32.) poziva se pri pomicanju klizača filtera, a odgovorna je za dohvaćanje postavljenih vrijednosti iz grafičkog sučelja, pozivanje metode za primjenu filtera za osvjetljenje i kontrast te metode za adaptivno postavljanje praga.

```
private void filterSlider_Scroll(object sender, EventArgs e)
{
    try
    {
        if (filterBox.Text == Constants.contrastLabel)
        {
            contrast = filterSlider.Value;
            currentSliderValue.Text = ((float)contrast / 100).ToString();
        }
        else
        {
            brightness = filterSlider.Value;
            currentSliderValue.Text = brightness.ToString();
        }

        documentScanner.ContrastBrightnessAdjust(contrast, brightness);

        if (adaptiveThresholdToolStripMenuItem.Checked)
        {
            documentScanner.ResultImage = documentScanner.AdaptiveThreshold();
        }

        resultBoxView.Image = new Bitmap(documentScanner.ResultImage, resultBoxView.Size);
    }
    catch(Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

Slika 4.32. Metoda *filterSlider_Scroll*

Metoda *ContrastBrightnessAdjust* (Slika 4.33.) primjenjuje postavljene filtere za kontrast i osvjetljenje na sliku i pohranjuje sliku.

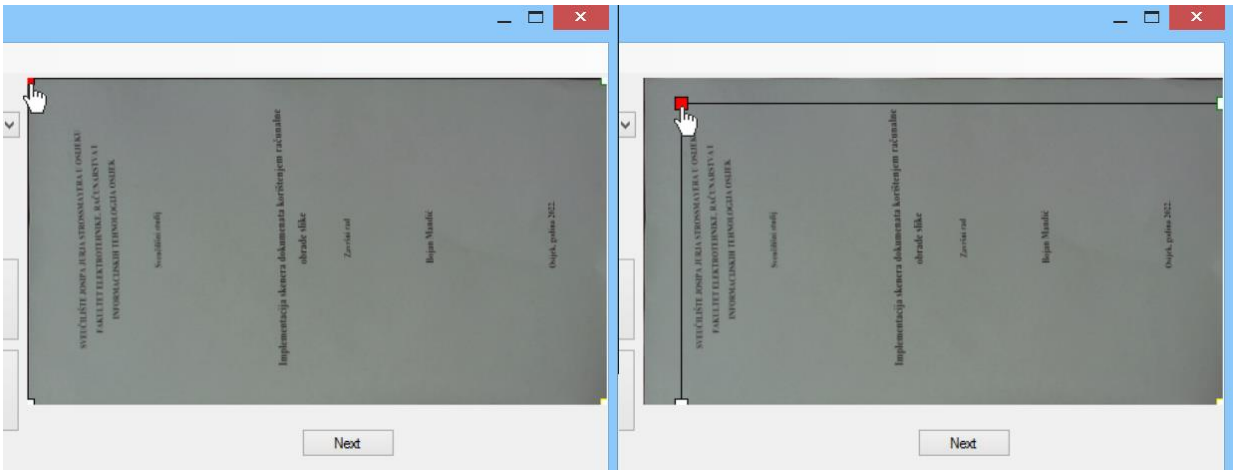
```
public void ContrastBrightnessAdjust(int contrast, int brightness)
{
    try
    {
        using (Image<Bgr, byte> adjustedImage = (CapturedImage.ToImage<Bgr,
            byte>().Mul((float)contrast / 100) + brightness))

        {
            ResultImage = new Bitmap(adjustedImage.ToBitmap());
        }

    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

Slika 4.33. Metoda *ContrastBrightnessAdjust*

Ako korisnik odluči izrezati sliku, klikom na gumb „Crop Image“ poziva se metoda *cropImageToolStripMenuItem_Click* (Slika 4.34.) koja crta pomične točke na okviru za prikaz obrađene slike, isto kao i kod korigiranja perspektive (Slika 4.19.). Jedina razlika je što se točke pomiču zajedno da bi se zadržao pravokutni oblik. Primjer je vidljiv na slici 4.35. Pomicanjem crvene točke prema dolje pomiče se i točka u desnom gornjem kutu, a pomicanjem crvene točke udesno, pomiče se i točka u lijevom donjem kutu.

```
private void cropImageToolStripMenuItem_Click(object sender, EventArgs e)
{
    mPoint1 = new Point(0, 0);
    mPoint2 = new Point(0, resultBoxView.Height);
    mPoint3 = new Point(resultBoxView.Width, resultBoxView.Height);
    mPoint4 = new Point(resultBoxView.Width, 0);
    documentScanner.drawMovablePoints();

    isCropOn = true;
    perspectiveButton.Visible = true;
}
```

Slika 4.34. Metoda *cropImageToolStripMenuItem_Click*



Slika 4.35. Primjer pomicanja točaka za izrezivanje

Pritiskom na gumb „Reset To Original Image“ brišu se svi filteri i izrezivanja, te se vraća slika koja je dobivena poslije ispravlja perspektive. Ova funkcionalnost olakšava ispravljanje mogućih pogrešaka pri izrezivanju. Metoda *resetToOriginalImageToolStripMenuItem_Click* je prikazana na slici 4.36.

```
private void resetToOriginalImageToolStripMenuItem_Click(object sender, EventArgs e)
{
    ResetFilter();
    documentScanner.CapturedImage = new Bitmap(documentScanner.OriginalImage);
    documentScanner.ResultImage = new Bitmap(documentScanner.OriginalImage);
    resultBoxView.Image = new Bitmap(documentScanner.ResultImage, resultBoxView.Size);
}
```

Slika 4.36. Metoda *resetToOriginalImageToolStripMenuItem_Click*

4.5. IZLAZNI PODACI

Uređenu sliku moguće je pohraniti na računalo u .png formatu ili ispisati pomoću printera. Metode kojima je ostvarena ova funkcionalnost prikazane su na slici 4.37. Ove metode koriste funkcije *Windows Forms-a* za prikaz jednostavnog prozora za spremanja i ispisivanje slike.

```
private void printImage_Click(object sender, EventArgs e)
{
    PrintDialog pd = new PrintDialog();
    PrintDocument doc = new PrintDocument();
    doc.PrintPage += myPrintPage;
    pd.Document = doc;

    if (pd.ShowDialog() == DialogResult.OK)
    {
        doc.Print();
    }
}

1 reference
private void saveImage_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "PNG (*.png)|*.png";
    if (sfd.ShowDialog() == DialogResult.OK)
    {
        documentScanner.ResultImage.Save(sfd.FileName, ImageFormat.Png);
    }
}
```

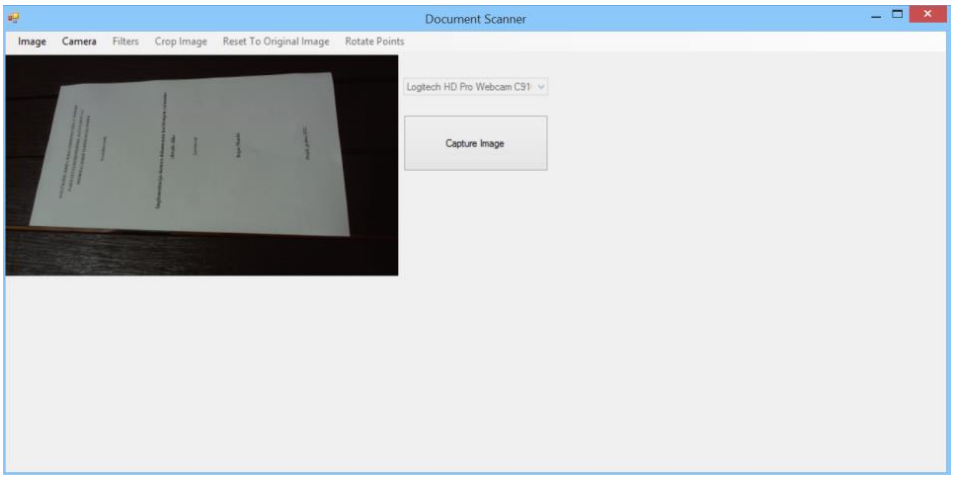
Slika 4.37. Metode *printImage_Click* i *saveImage_Click*

5. REZULTATI

5.1. USPJEŠNOST SKENIRANJA

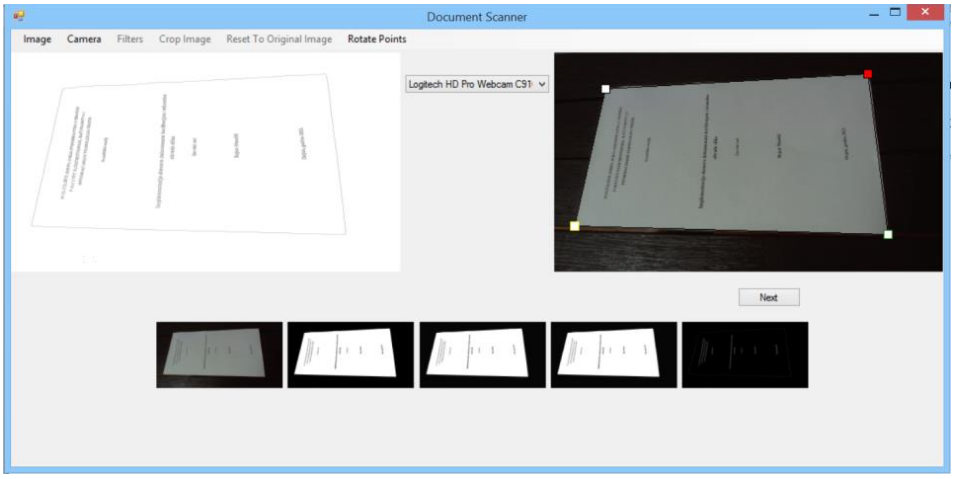
U svrhu testiranja korištena je Logitech HD Pro C910 kamera (rezolucije 1920x1080) i testiranje se odvija korak po korak korištenjem oba načina za učitavanje slike spomenuta u potpoglavlju 4.1. Prvi test je učitavanje slike direktno s kamere.

Nakon što se aplikacija pokrene, odabire se željena kamera, a to je u ovome slučaju Logitech HD Pro C910. Pritiskom na tipku Camera -> Start počinje dohvaćanje slika s kamere koje se prikazuju na lijevom okviru za slike. Ovaj proces je prikazan na slici 5.1.



Slika 5.1. Izgled aplikacije nakon odabira kamere i pritiska na gumb Camera->Start

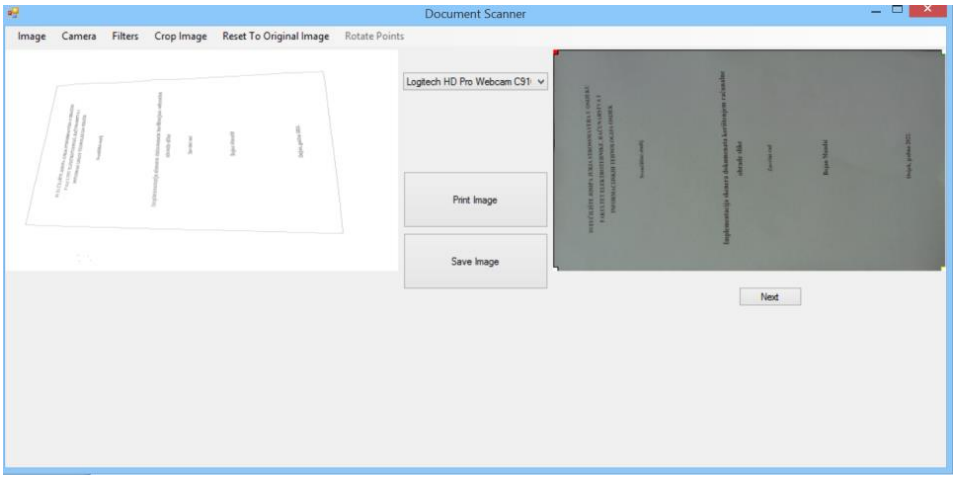
Kamera se pozicionira iznad dokumenta koji se želi uslikati i pritiskom na tipku „Capture Image“ ona se slika. Na desnom okviru za sliku pojavljuje se uslikana slika, zajedno sa ucrtanim pomičnim točkama koje su uspješno postavljene u vrhove detektiranog papira. U lijevi okvir postavlja se invertirana slika svih detektiranih rubova koja je dobivena metodom *DetectEdges*. Za potrebe ovog testiranja dodano je pet okvira za slike za prikaz međukoraka metode *DetectEdges*. S lijeva na desno okviri predstavljaju (Slika 5.2.): izvornu sliku, rezultate metode *ThresholdBinary*, *CvInvoke.CLAHE*, *CvInvoke.GaussianBlur* i *CvInvoke.Canny*.



Slika 5.2. Izgled aplikacije nakon pritiska na tipku „Capture Image“

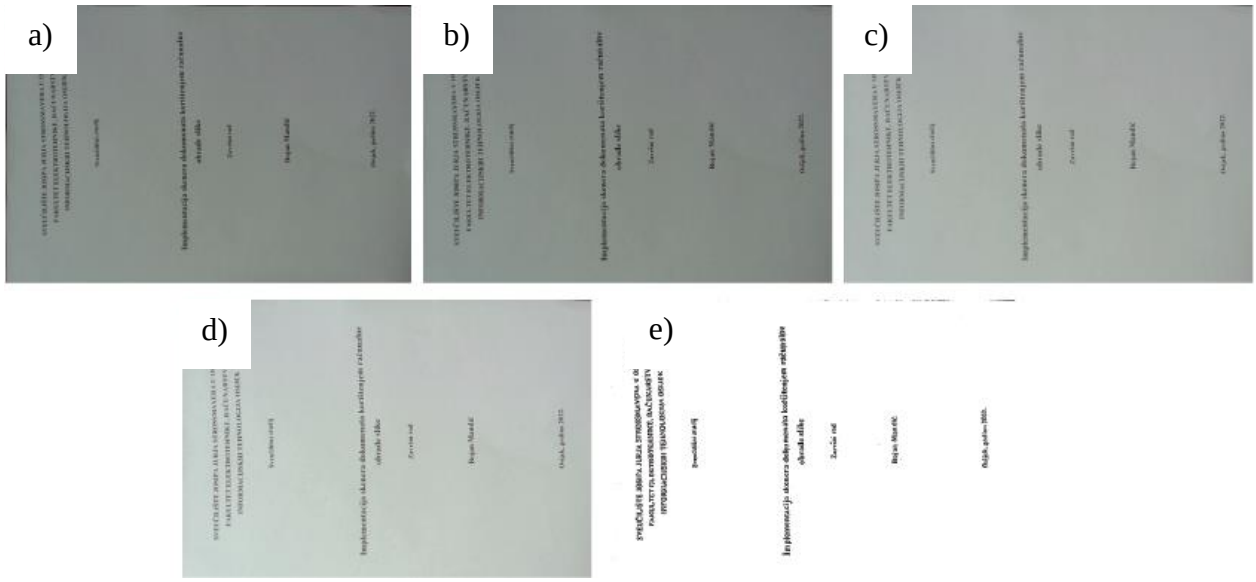
Na slici koja je dobivena metodom *CvInvoke.Canny* slabije se vide rubovi koji su predstavljeni bijelom bojom na crnoj pozadini pa je zato na lijevom okviru prikazana invertirana slika radi lakšeg uočavanja rubova. Crvena točka postavljena je u krivom vrhu i potrebno ju je rotirati u gornji desni vrh papira inače će ispravljena slika biti okrenuta naopako. Nakon tri pritiska na gumb „Rotate Points“ crvena točka dolazi u odgovarajući vrh. Pritiskom na gumb „Next“ ispravlja se perspektiva slike i prikazuje u desnom okviru.

Na ispravljenoj slici i dalje je vidljiv stol pa će se u tu svrhu koristiti alat za izrezivanje slike. Pritiskom na gumb „Crop Image“ pojavljuju se točke u vrhovima okvira kojima je moguće označiti dio slike koji se želi zadržati. Nakon što su točke namještene, pritiskom na gumb „Next“ izrezuje se željeni dio slike.



Slika 5.3. Izgled aplikacije nakon ispravljanja perspektive i pritiska na gumb „Crop Image“

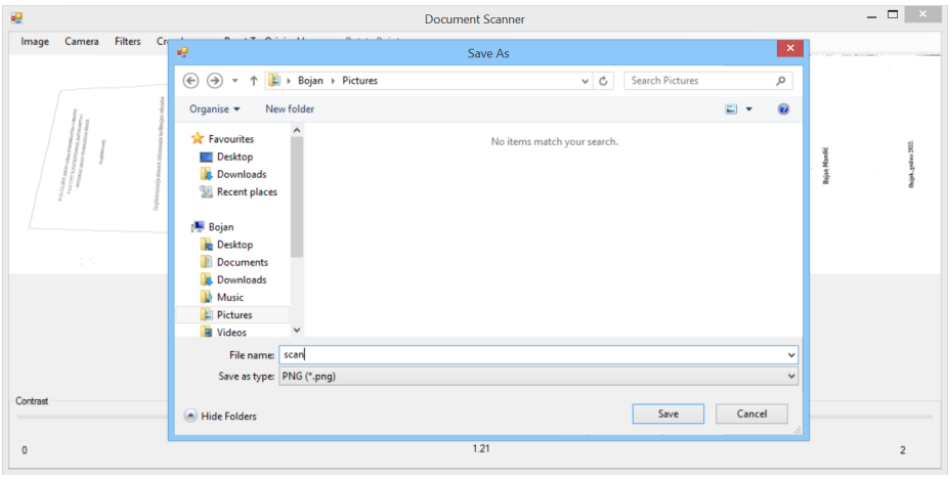
Nakon što je odrezan neželjeni dio slike, korištenjem različitih filtera poboljšava se kvaliteta slike. Prvo se koristi kontrast filter budući da je izrezana slika previše tamna. Klikom na gumb Filters -> Brightness pojavljuje se klizač koji služi za primjenu filtera. Budući da je slika i dalje loša, odabire se kontrast filter klikom na Filters -> Contrast i klizačem postavlja vrijednost. Nakon toga odabire se prilagodljivo postavljanje praga pritiskom na Filters -> Adaptive Threshold i dobiva binarna slika. Konačna slika je znatno bolje kvalitete od početne slike, što je vidljivo na slici 5.5.



Slika 5.4. Slike koje su dobivene primjenom filtera: a) originalna slika, b) izrezana slika, c) korekcija svjetline (*brightness*), d) korekcija kontrasta (*contrast*), e) prilagodljivo postavljanje praga (*Adaptive Thresholding*)

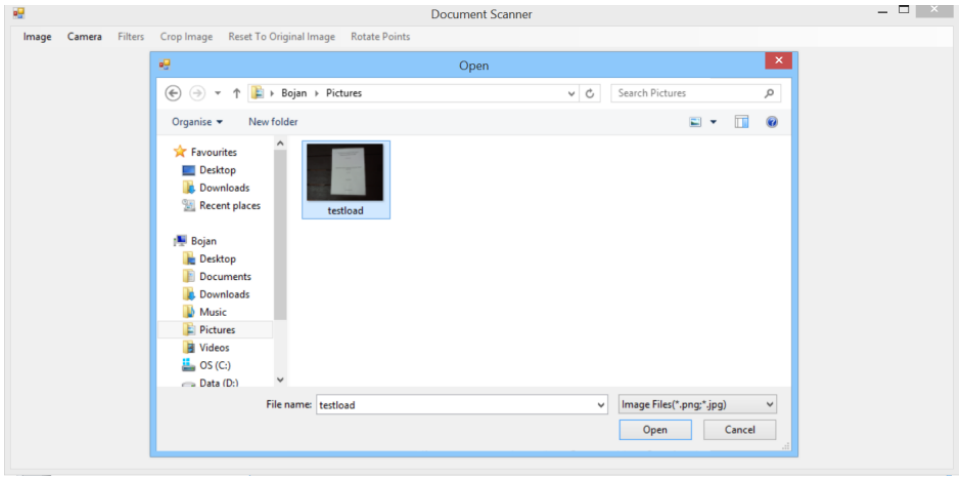
Slika 5.5. Usporedba ispravljene i konačne slike dobivene u prvom testu

Obrađenu sliku zatim je moguće pohraniti u PNG formatu pritiskom na gumb „Save Image“ u mapu „Pictures“ pod nazivom „scan“. Prozor za spremanje obrađene slike prikazan je na slici 5.6.



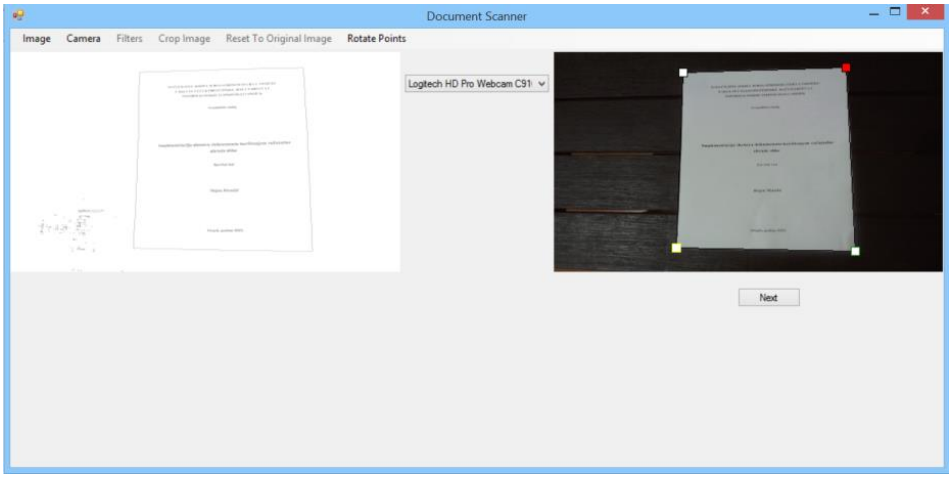
Slika 5.6. Prozor za spremanje slike

Drugi test napravljen je učitavanjem slike s računala. Sliku je moguće učitati pritiskom na gumb „Image“, a zatim u prozoru koji se otvori odabrati željenu sliku.



Slika 5.7. Prozor za odabir slike

Aplikacija uspješno detektira vrhove papira i ponavljaju se svi koraci za ispravljanje perspektive i poboljšanja kvalitete slike opisani u prethodnom testu.

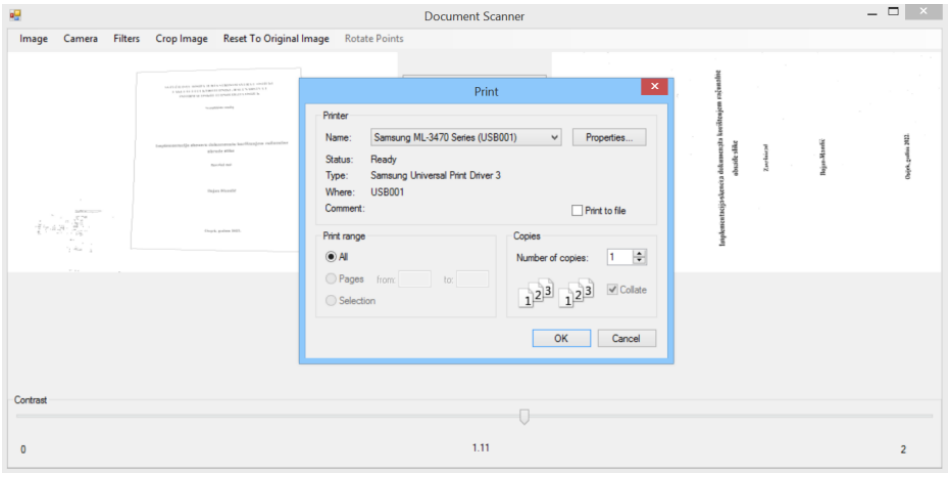


Slika 5.8. Prikaz aplikacije poslije učitavanja slike s računala

Usporedba ispravljene slike i obrađene slike dobivene u drugom testu prikazana je na slici 5.9. Kvaliteta slike znatno je poboljšana i tekst je čitljiviji nego prije.

Slika 5.9. Usporedba ispravljene i konačne slike dobivene u drugom testu

Sliku je moguće isprintati pritiskom na gumb „Print Image“. Nakon toga otvara se prozor za ispis u kojemu je moguće odabrati željeni printer i promijeniti njegove postavke (Slika 5.10).

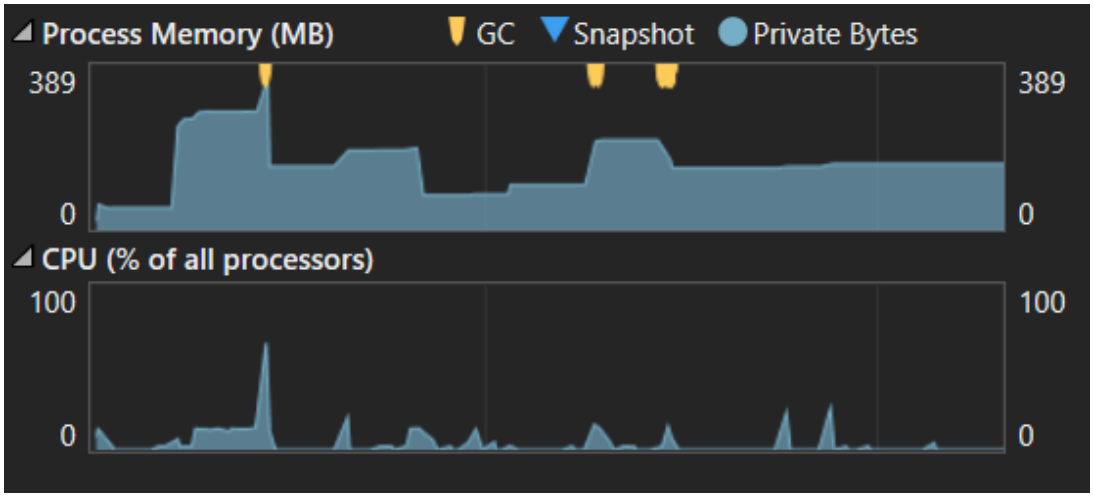


Slika 5.10. Izgled prozora za ispis

Rezultati dobiveni u prvom i drugom testu su uspješni, iako su slike bile drugačije i učitane na različite načine. Aplikacija u oba slučaja uspješno detektira rubove papira, ispravlja perspektivu i poboljšava kvalitetu obrađene slike.

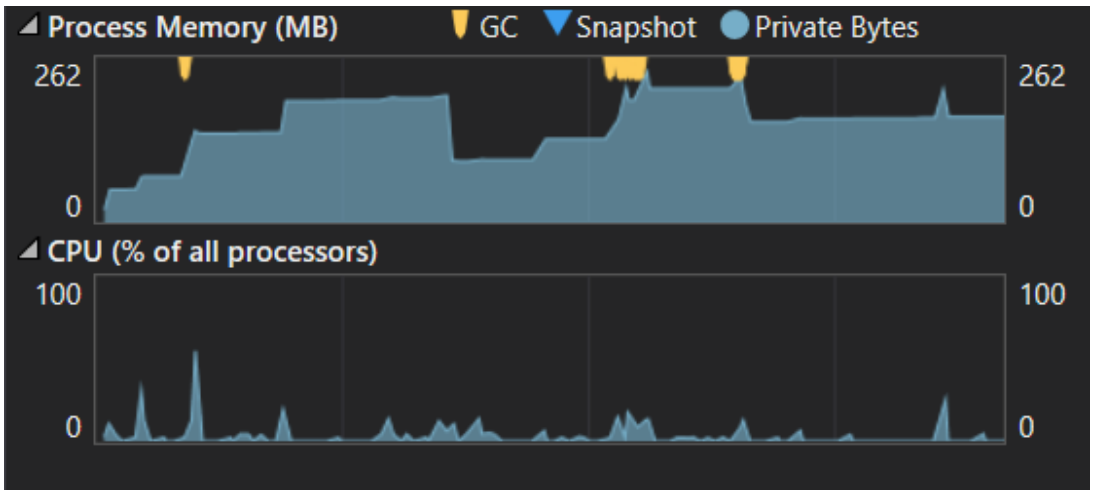
5.2. PERFORMANSE

Aplikacija je testirana na računalu koje ima Intel i7-4270HQ procesor koji radi na frekvencijama do 2.6 GHz i ima ugrađenih 12 GB DDR3 RAM memorije. Aplikacija se izvodi bez zastoja i troši relativno malo računalnih resursa. Za prvi test, aplikacija troši najmanje resursa kod učitavanja grafičkog sučelja i dohvaćanja liste kamera (52 MB RAM memorije i 13% procesorskog kapaciteta), a najviše troši pri stalnom dohvaćanju slika s kamere (219 MB i 13% procesorskog kapaciteta) i ima kratkotrajni skok kod pritiska na tipku za slikanje slike, kada se izvršava i proces detekcije papira (354 MB RAM memorije i 64% procesorskog kapaciteta). Nakon toga se aktivira sakupljač smeća koji oslobađa nepotrebnu memoriju i tada se aplikacija stabilizira (u prosjeku oko 120 MB RAM memorije i 12% procesorskog kapaciteta). Graf performansi za prvi test dobiven korištenjem alata za analizu performansi *Visual Studio-a* prikazan je na slici 5.11.



Slika 5.11. Graf performansi za prvi test

Pri drugom testu, aplikacija također troši najmanje resursa kod učitavanja grafičkog sučelja i dohvaćanja liste kamera (52 MB RAM memorije i 13% procesorskog kapaciteta), a najviše troši pri pokušaju detekcije papira na slici gdje ima skok u korištenju procesora (oko 55% procesorskog kapaciteta) i primjenjivanju različitih filtera (oko 212 MB i 15% procesorskog kapaciteta). Potrošnja resursa se poslije stabilizira i u prosjeku je oko 165 MB RAM memorije i 15% procesorskog kapaciteta. Graf performansi za drugi test prikazan je na slici 5.12.



Slika 5.12. Graf performansi za drugi test

6. ZAKLJUČAK

U sklopu završnog rada izrađena je desktop aplikacija koja korisniku omogućava skeniranje dokumenta iz učitane digitalne slike ili direktno s kamere. Nakon što je učitana, iz slike je moguće poluautomatski izrezati dokument iz pozadine i poboljšati kvalitetu slike korištenjem različitih filtera i alata. Aplikacija je namijenjena korisnicima koji nemaju računalni skener, a imaju potrebu pretvoriti dokument u digitalni oblik.

Završnim radom opisan je postupak izrade aplikacije koja se sastoji od jednostavnog sučelja, alata za detekciju papira, poluautomatskog ispravljanja perspektive i različitih alata za poboljšanje kvalitete obrađene slike. Postupak izrade objašnjen je korak po korak i sve implementirane funkcionalnosti su kasnije testirane u dva različita testa. Osim funkcionalnosti, testirane su i performanse aplikacije za oba testa. Za izradu aplikacije korištene su različite biblioteke za *C#* poput *Emgu CV-a* i *DirectShow.NET*, dok je sučelje izrađeno korištenjem *Windows Forms* platforme.

Tijekom izrade završnog rada, najveći problem bio je pronaći i popraviti uzrok curenja memorije koji se događao prilikom aktivnog dohvaćanja slika iz kamere, te pri nagloj promjeni klizača za postavljanje intenziteta filtera. Korištenjem različitih alata koje pruža razvojno okruženje *Visual Studio*, problem je vrlo brzo riješen. Aplikacija ima prostora za nadogradnju kao npr. ugrađivanje funkcionalnosti automatskog dohvaćanja teksta iz dokumenta i njegovog prijevoda na druge jezike ili dodatno poboljšanje metoda za detekciju papira.

LITERATURA

[1] C#, Microsoft, dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/ [17.08.2022].

[2] Emgu CV, dostupno na: https://www.emgu.com/wiki/index.php/Main_Page [17.08.2022].

[3] DirectShow.NET, SourceForge, dostupno na: http://directshownet.sourceforge.net/ [17.08.2022].

[4] Windows Forms, dostupno na: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-6.0 [17.08.2022].

[5] CamScanner INTSIG Information Co., Ltd, 2011., dostupno na: https://www.camscanner.com/ [17.08.2022].

[6] Adobe Scan: PDF Scanner, OCR, Adobe Inc., dostupno na: https://www.adobe.com/acrobat/mobile/scanner-app.html [03.09.2022].

[7] Evernote, Evernote Corporation, dostupno na: https://evernote.com/ [03.09.2022].

[8] OnlineCamScanner, dostupno na: https://onlinecamscanner.com/ [17.08.2022].

[9] PaperScan 3, Orpalis Imaging SAS, dostupno na: https://paperscan.orpalis.com/ [17.08.2022].

[10] .NET, Microsoft, dostupno na: https://dotnet.microsoft.com/en-us/ [17.08.2022].

[11] Open CV, Intel, dostupno na: https://opencv.org/ [17.08.2022].

[12] A. F. Villán, Mastering OpenCV 4 with Python, Packt Publishing Ltd, 2019.

[13] R. C. Gonzalez, R. E. Woods, Digital Image Processing, Prentice Hall, 2008.

[14] pyimagesearch, dostupno na: https://pyimagesearch.com/2021/02/01/opencv-histogram-equalization-and-adaptive-histogram-equalization-clahe/ [29.08.2022].

SAŽETAK

Tema završnog rada izrada je desktop aplikacije za skeniranje dokumenata korištenjem računalne obrade slike. Aplikacija je namijenjena osobama koje žele digitalizirati dokument, ali ne posjeduju računalni skener. Učitanoj slici s računala ili kamere, korisnik može ispraviti perspektivu, odrezati neželjene dijelove slike ili poboljšati kvalitetu iste korištenjem različitih filtera, a zatim konačnu sliku pohraniti na računalo ili ju ispisati. Prilikom izrade aplikacije korišten je programski jezik *C#*, kao i njegove pripadajuće biblioteke *Emgu CV* i *DirectShow.NET*. Za izradu korisničkog sučelja korišten je *Windows Forms*.

Ključne riječi: C#, Emgu CV, računalna obrada slike, skener dokumenata

ABSTRACT

Implementation of document scanner using digital image processing

Final thesis theme is about implementation of document scanner using digital image processing. Application aims to help users who don't have access to a real computer scanner but want to digitalize their documents. After loading image from computer or camera, user can correct perspective, crop unwanted parts of the image or improve its quality by using different filters. Finally, user can save the result image to the computer or print it out. For the development of the application, C# is used as a base programming language, as well as its associated libraries *Emgu CV* i *DirectShow.NET*. *Windows Forms* library is used for the development of the graphical interface.

Key words: C#, Emgu CV, digital image processing, document scanner

ŽIVOTOPIS

Bojan Mandić rođen je 28.08.1997. u Našicama. Nakon završetka osnovne škole upisuje prirodoslovno-matematičku gimnaziju u Srednjoj školi Isidora Kršnjavoga u Našicama. Nakon srednje škole 2016. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike i računarstva u Zagrebu s kojeg se 2019. godine prebacuje na preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.