# TECHNICAL REPORT

IMAGE PROCESSING FOR CROPPING SCANNED BOOK PAGES

14 OCTOMBRIE 2024

CIOPEC ANDREI SORIN
GLIGOR CĂTĂLIN ANDREI
DRAGOI ANDREI CĂTĂLIN

## Table of content:

## 1. Introduction

Scanned book pages, especially from non-professional scanners, often include unnecessary borders, margins, or background stuff around the actual content. The objective of this project is to develop a Python-based application that removes this extra space, leaving only the actual page contents. The final output will be PDF files with trimmed-up pages, improving readability and presentation.

This report provides an overview of the key image processing techniques required to achieve this task, including binarization, edge detection, contour finding, and cropping.

## 2. Theoretical Background

### *2.1. Image Representation*

Digital images are stored as matrices of pixel values. For scanned documents, grayscale images (with pixel values ranging from 0 to 255) are commonly used to simplify processing tasks such as detecting text.

### *2.2. Binarization*

Binarization converts a grayscale image into a black-and-white image (binary). This simplifies distinguishing the page content from the background. Otsu's method is a common technique that automatically selects the optimal threshold value for this task.

### *2.3. Edge Detection*

Edge detection algorithms, like the Canny edge detector, are used to locate the sharp changes in pixel intensity. These changes often correspond to the boundaries of the actual book page within a scanned image.

### *2.4. Contour Detection*

Once the edges are detected, contours - closed curves around objects in an image - can be identified. In the case of a scanned book page, the largest contour typically corresponds to the page boundaries, which can then be extracted.

### *2.5. Cropping and Perspective Correction*

After identifying the page’s contour, perspective transformation is applied to straighten any tilted or skewed pages. The image is then cropped to remove unnecessary white space, ensuring the content is neatly contained within the boundaries of the page.

## 3. Python Implementation

Python libraries like **OpenCV**, for image processing, **PyPDF2**, for handling PDF files, and **Pillow**, for image manipulation, will be used in the implementation. The basic steps for our workflow will consist of:

1. **Extracting images** from the input PDF files
2. **Preprocess** the images - convert to grayscale, binarize
3. **Detect edges and contours** to identify the page boundaries
4. **Correct perspective** if needed, and crop the image
5. **Reassemble** the processed images back into a PDF

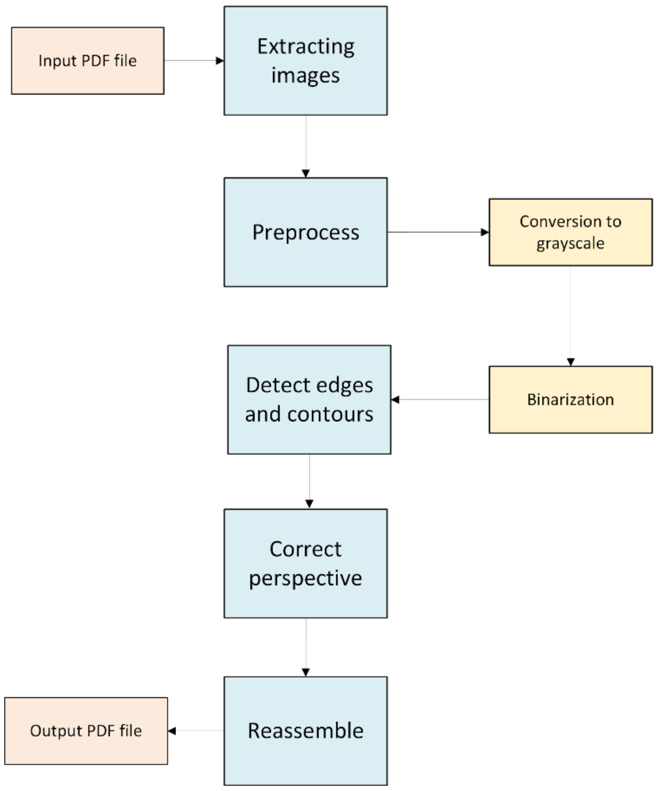


*Figure 1 Phyton implementation flowchart*

## 4. Implementation Workflow Explanation

### 1. Library Imports and GUI Setup

The code imports libraries essential for image processing (cv2, imutils, numpy), GUI handling (tkinter), and various filtering operations (skimage.filters). This includes:

- **OpenCV (cv2)** for handling image manipulation and processing.
- **Imutils** for simplifying certain OpenCV functions.
- **Numpy** for array operations, especially with image data.
- **Tkinter** for creating and managing the GUI elements.
- **PIL (Pillow)** for converting images between formats, which is useful for displaying processed images in the GUI.
- **Skimage.filters** for applying the unsharp mask filter to enhance document clarity.

### 2. Main Function for Image Processing (process_image)

The process_image function is central to the document scanning process:

- **Image Loading and Initial Checks**: The function first reads an image from the specified image_path. If no image is found, it raises an error.
- **Sharpening and Grayscale Conversion**: A sharpening kernel is applied using OpenCV's filter2D to increase edge contrast. The image is then converted to grayscale and blurred with Gaussian blur to reduce noise, making it easier to detect document edges accurately.
- **Edge Detection with Canny**: The Canny edge detector dynamically thresholds based on the image's median intensity, allowing for robust edge detection regardless of lighting conditions in the image.

### 3. Contour Detection and Document Extraction

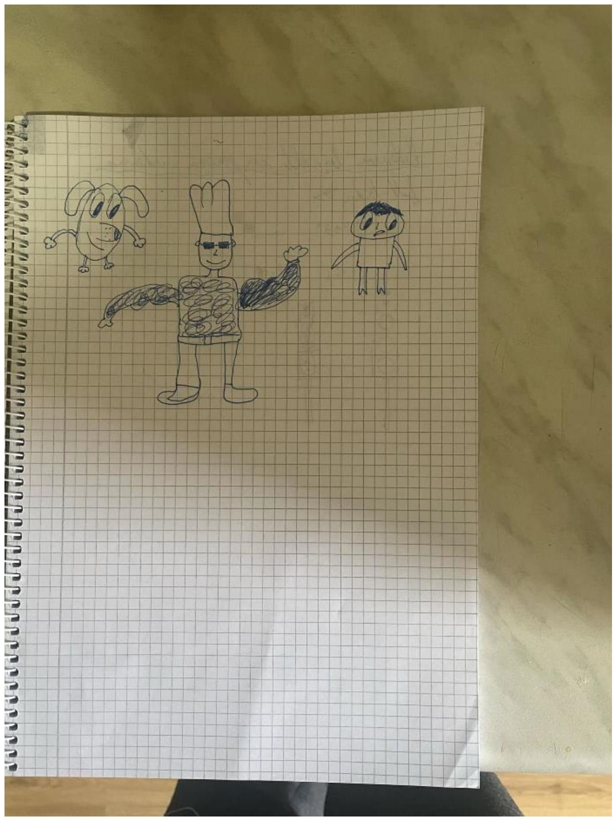
The function then finds contours in the image to identify possible document outlines:

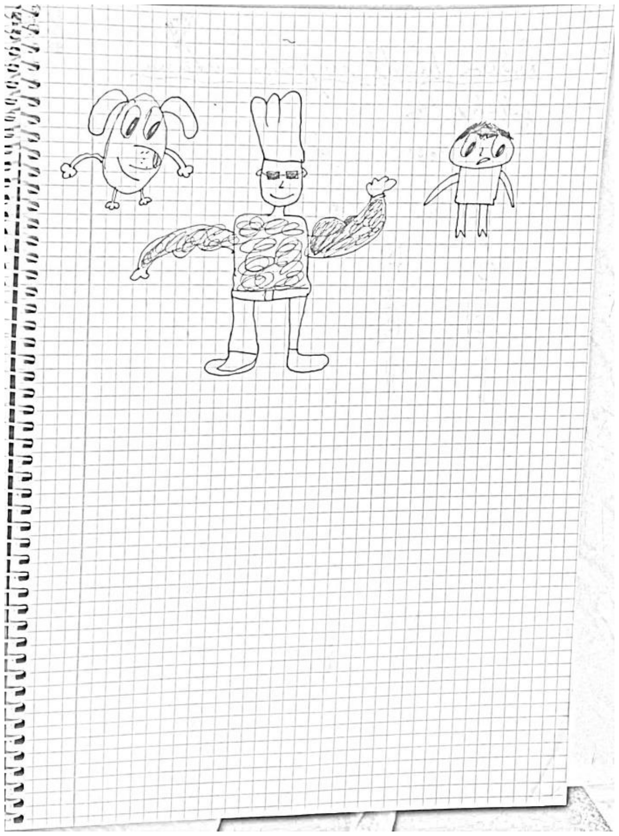
- **Dilation and Erosion**: Edges are dilated (expanded) and then eroded (contracted) to close any gaps and reduce noise in the edge detection output.
- **Selecting Suitable Contours**: Only large contours are selected, which are more likely to correspond to a document or page within the image. If no suitable contours are found, the function raises an error.

If a contour is a **quadrilateral** (i.e., has four vertices), the code assumes it is a document and applies a **perspective transform** to isolate and flatten the document within the image.

## 5. Testing Image Processing Without Graphic User Interface

For testing purposes, the input will be a raw picture, and the output will be the trimmed picture as indicated below.





## 6. Graphic User Interface

The purpose of the graphic interface is to allow users to upload a .pdf file, preview and process the file accordingly.

The GUI is presented below.

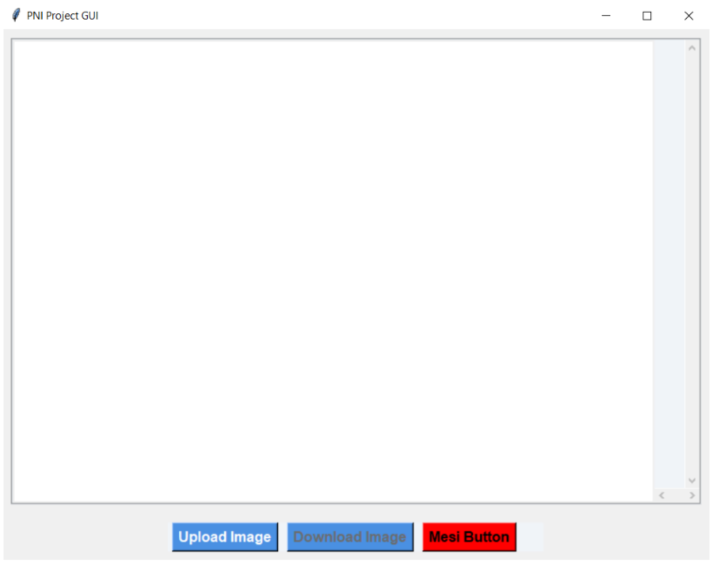


*Figure 2 Default GUI appearance*

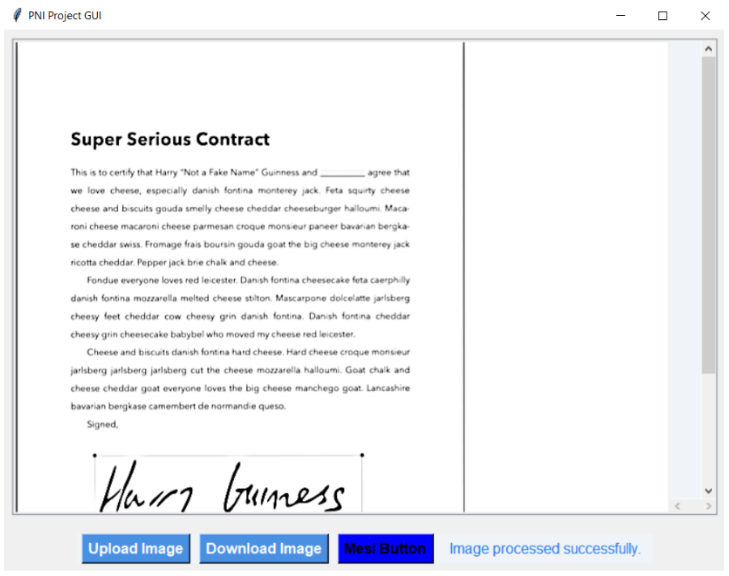


Image processed successfully.

**Super Serious Contract**

This is to certify that Harry "Not a Fake Name" Guinness and __________ agree that we love cheese, especially danish fontina monterey jack. Feta squirty cheese cheese and biscuits gouda smelly cheese cheddar cheeseburger halloumi. Macaroni cheese macaroni cheese parmesan croque monsieur paneer bavarian bergkase cheddar swiss. Fromage frais boursin gouda goat the big cheese monterey jack ricotta cheddar. Pepper jack brie chalk and cheese.

Fondue everyone loves red leicester. Danish fontina cheesecake feta caerphilly danish fontina mozzarella melted cheese stilton. Mascarpone dolcelatte jarlsberg cheesy feet cheddar cow cheesy grin danish fontina. Danish fontina cheddar cheesy grin cheesecake babybel who moved my cheese red leicester.

Cheese and biscuits danish fontina hard cheese. Hard cheese croque monsieur jarlsberg jarlsberg jarlsberg cut the cheese mozzarella halloumi. Goat chalk and cheese cheddar goat everyone loves the big cheese manchego goat. Lancashire bavarian bergkase camembert de normandie queso.

Signed,

[illegible]

*Figure 3 GUI with result*

## 7. Appendix

```
import cv2
import imutils
import numpy as np
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk, ImageOps
import skimage.filters as filters


def process_image(image_path):
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError("Image not found. Check the file name and path.")

    sharpen_kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
    sharpened = cv2.filter2D(img, -1, sharpen_kernel)

    gray = cv2.cvtColor(sharpened, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (7, 7), 0)

    v = np.median(gray)
    lower = int(max(0, (1.0 - 0.33) * v))
    upper = int(min(255, (1.0 + 0.33) * v))
    edges = cv2.Canny(gray, lower, upper)

    kernel = np.ones((5, 5), np.uint8)
    dilated = cv2.dilate(edges, kernel, iterations=3)
    closed = cv2.erode(dilated, kernel, iterations=2)

    contours = cv2.findContours(closed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)
```

```
min_area_threshold = 5000
contours = [c for c in contours if cv2.contourArea(c) > min_area_threshold]

if not contours:
    print("No contours found.")
    return None, None

contour = max(contours, key=cv2.contourArea)
epsilon = 0.02 * cv2.arcLength(contour, True)
approx = cv2.approxPolyDP(contour, epsilon, True)

if len(approx) != 4:
    print("Could not find a suitable quadrilateral.")
    return None, None

pts = approx.reshape(4, 2)
rect = np.zeros((4, 2), dtype="float32")

s = pts.sum(axis=1)
rect[0] = pts[np.argmin(s)]
rect[2] = pts[np.argmax(s)]
diff = np.diff(pts, axis=1)
rect[1] = pts[np.argmin(diff)]
rect[3] = pts[np.argmax(diff)]

width = int(max(np.linalg.norm(rect[0] - rect[1]), np.linalg.norm(rect[2] - rect[3])))
height = int(max(np.linalg.norm(rect[0] - rect[3]), np.linalg.norm(rect[1] - rect[2])))
dst = np.array([[0, 0], [width - 1, 0], [width - 1, height - 1], [0, height - 1]], dtype="float32")

M = cv2.getPerspectiveTransform(rect, dst)
```

```
    warped = cv2.warpPerspective(img, M, (width, height))


    gray_warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)

    blurred_dilation = cv2.GaussianBlur(gray_warped, (51, 51), 0)

    division = cv2.divide(gray_warped, blurred_dilation, scale=255)

    sharp = filters.unsharp_mask(division, radius=5, amount=1, preserve_range=False)

    sharp = (255 * sharp).clip(0, 255).astype(np.uint8)

    thresh = cv2.threshold(sharp, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]


    return sharp, thresh


def open_file():

    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.jpg *.jpeg *.png")])

    if file_path:

        try:

            sharp, thresh = process_image(file_path)

            if sharp is not None and thresh is not None:

                display_image(sharp)

                global processed_image

                processed_image = sharp

                download_button.config(state=tk.NORMAL)

                message_label.config(text="Image processed successfully.")

            else:

                message_label.config(text="Could not process the image.")

        except Exception as e:

            messagebox.showerror("Error", str(e))


def display_image(image):

    #Convert the image to a PIL Image

    image = Image.fromarray(image)


    img_width, img_height = image.size
    #Configure the canvas scroll region to match the image size
    canvas.config(scrollregion=(0, 0, img_width, img_height))

    #Clear the canvas before adding a new image
    canvas.delete("all")

    #Display image on the canvas
    image_tk = ImageTk.PhotoImage(image)
    canvas.create_image(0, 0, anchor="nw", image=image_tk)
    canvas.image = image_tk

def download_image():
    file_path = filedialog.asksaveasfilename(defaultextension=".pdf", filetypes=[("PDF files", "*.pdf"), ("All files", 
"*.*")])
    if file_path:
        pil_image = Image.fromarray(processed_image)
        pil_image.save(file_path, "PDF", resolution=100.0)
        message_label.config(text="Image saved successfully as PDF")

def add_mesi_background():
    if not hasattr(canvas, "bg_image"):
        bg_image = Image.open("mesi.jpg")
        smaller_width, smaller_height = 150, 100
        bg_image = bg_image.resize((smaller_width, smaller_height), Image.LANCZOS)
        bg_photo = ImageTk.PhotoImage(bg_image)
        canvas.create_image(10, 10, anchor="nw", image=bg_photo)
        canvas.bg_image = bg_photo
        message_label.config(text="Messi background added!")
    else:
        message_label.config(text="Messi background already added.")
```

```
def create_rainbow_button(parent, text, command):
    colors = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]
    index = 0

    rainbow_button = tk.Button(
        parent,
        text=text,
        command=command,
        font=("Arial", 12, "bold"),
        relief="raised"
    )

    def update_color():
        nonlocal index
        rainbow_button.config(bg=colors[index])
        index = (index + 1) % len(colors)
        rainbow_button.after(200, update_color)  # Update color every 200ms

    update_color()  # Start the color cycling
    return rainbow_button



root = tk.Tk()
root.title("PNI Project GUI")
root.geometry("800x600")

#Canvas setup with scrollbars
canvas_frame = tk.Frame(root, bg="#f0f4f8", bd=2, relief="groove")
canvas_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

canvas = tk.Canvas(canvas_frame, bg="white")
canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
```

```
x_scrollbar = tk.Scrollbar(canvas_frame, orient=tk.HORIZONTAL, command=canvas.xview)
x_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)


y_scrollbar = tk.Scrollbar(canvas_frame, orient=tk.VERTICAL, command=canvas.yview)
y_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)


canvas.config(xscrollcommand=x_scrollbar.set, yscrollcommand=y_scrollbar.set)


#Resize the canvas when the window is resized
def on_canvas_resize(event):
    canvas.config(scrollregion=canvas.bbox("all"))


canvas.bind("<Configure>", on_canvas_resize)


#Buttons and other
button_frame = tk.Frame(root, bg="#f0f4f8")
button_frame.pack(side=tk.TOP, pady=10)


upload_button = tk.Button(button_frame, text="Upload Image", command=open_file, bg="#4a90e2", fg="white",
font=("Arial", 12, "bold"))
upload_button.pack(side=tk.LEFT, padx=5)


download_button = tk.Button(button_frame, text="Download Image", command=download_image,
state=tk.DISABLED, bg="#4a90e2", fg="white", font=("Arial", 12, "bold"))
download_button.pack(side=tk.LEFT, padx=5)
mesi_button = create_rainbow_button(button_frame, "Mesi Button", add_mesi_background)
mesi_button.pack(side=tk.LEFT, padx=5)
message_label = tk.Label(button_frame, text="", font=("Arial", 12), fg="#1a73e8", bg="#f0f4f8")
message_label.pack(side=tk.LEFT, padx=10)


root.mainloop()
```

## 8. Bibliography

https://docs.opencv.org https://realpython.com/opencv-python/

https://pyimagesearch.com/2017/07/10/building-a-pdf-document-scanner-with-opencv/

https://towardsdatascience.com/image-processing-with-python-application-of-opencv-22eb80d2aff9

https://stackoverflow.com/questions/48273143/crop-image-inside-contour-opencv-python