TABLE OF CONTENTS

ABSTRACT

The objective of this project is to develop a document scanner using OpenCV.

The scanner will be able to capture images of documents, apply image processing techniques to enhance the quality, and convert them into digital formats.

By automating the scanning process,we aim to improve efficiency and convenience in document management. This presentation will provide an overview of the existing document scanning systems, their limitations,the proposed system's features and benefits. Additionally, the hardware and software requirements for implementing the scanner will be discussed.

This project leverages the power of OpenCV,an open-source computer vision and machine learning software library to achieve the goal of developing seamless physical documents into digital formats.

In conclusion, the document scanner built using OpenCV represents a significant advancement in document scanning technology, offering improved efficiency, accuracy, and usability.

EXISTING SYSTEMS

Traditional Document Scanning: Flatbed scanners are commonly used for document scanning. Documents are placed on a glass surface and scanned using a moving light source and sensors. Requires manual placement of documents.

Portable Scanners: Handheld or portable scanners are available for on-the-go scanning. These scanners are compact and lightweight, allowing for easy transportation. They may use sensors or cameras to capture images of documents.

EXISTING SYSTEMS (DISADVANTAGES)

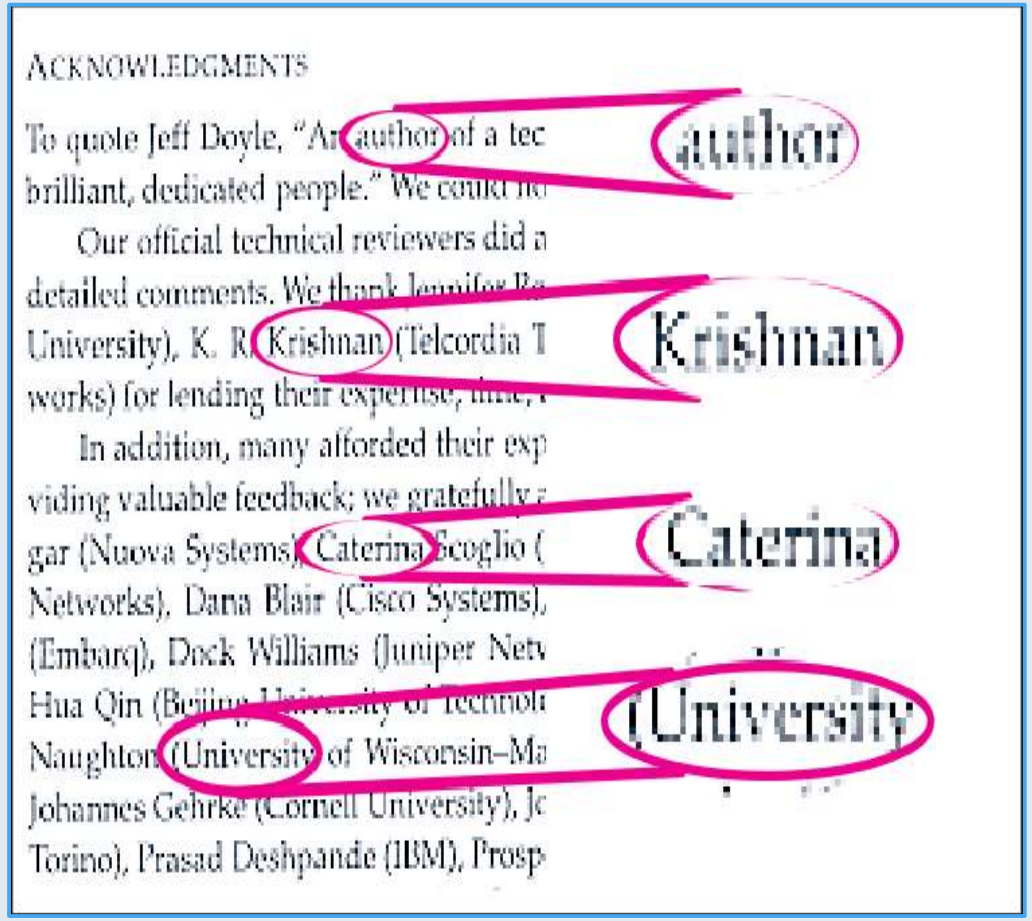
Limited Compatibility: Existing document scanning systems may not support all file formats, making it difficult to work with different types of documents.

Limited Features: Many systems may lack advanced features such as OCR (Optical Character Recognition) and document editing capabilities.

Slow Processing Speed: The scanning process can be time-consuming, especially when dealing with large volumes of documents.

Lack of Accuracy: Some systems may struggle with accurately capturing text and images .

High Cost: Implementing and maintaining existing document scanning systems can be expensive, especially for small businesses.



PROPOSED SYSTEM

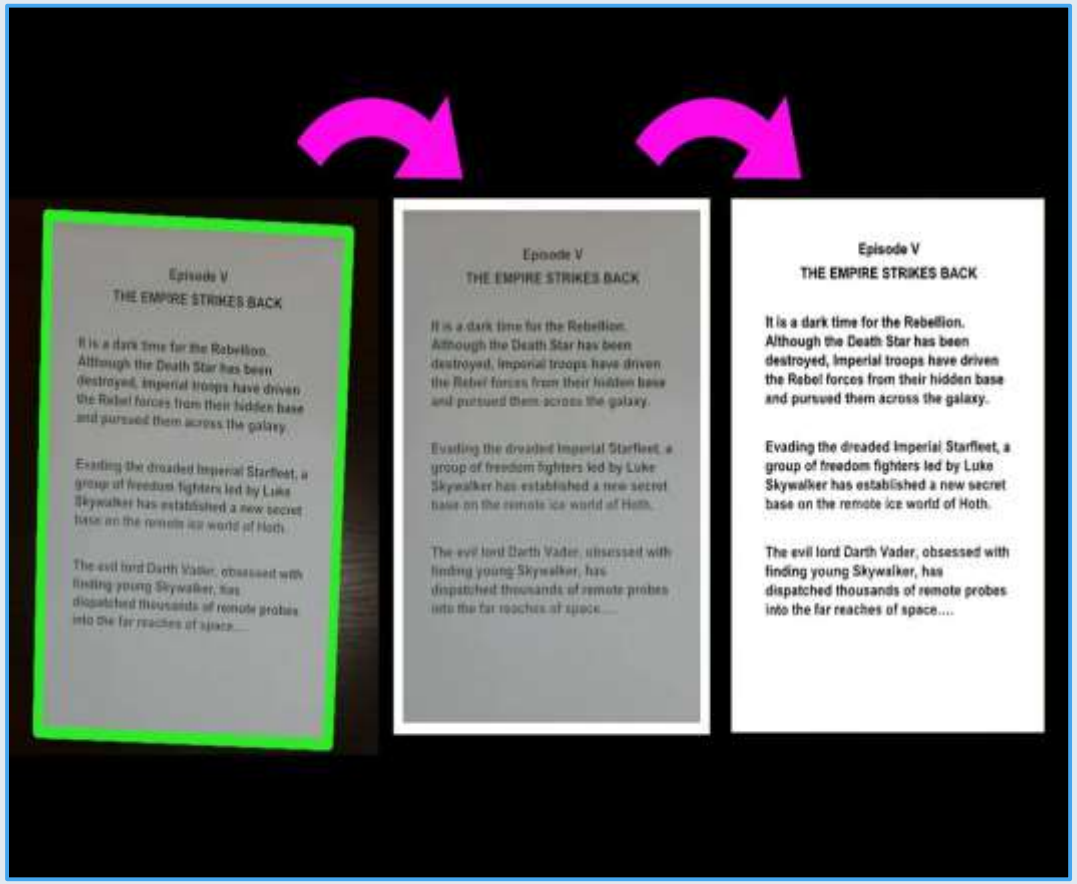
Image Acquisition: Capture an image of the document using a camera or scanner.

Pre-processing: Enhance the image quality by applying filters and adjusting brightness and contrast.

Document Detection: Identify the document boundaries and crop the image to remove any unwanted background.

Image Enhancement: Improve the readability of the document by reducing noise and enhancing the text.

Optical Character Recognition (OCR): Extract the text from the scanned document using OCR algorithms.



ADVANTAGES

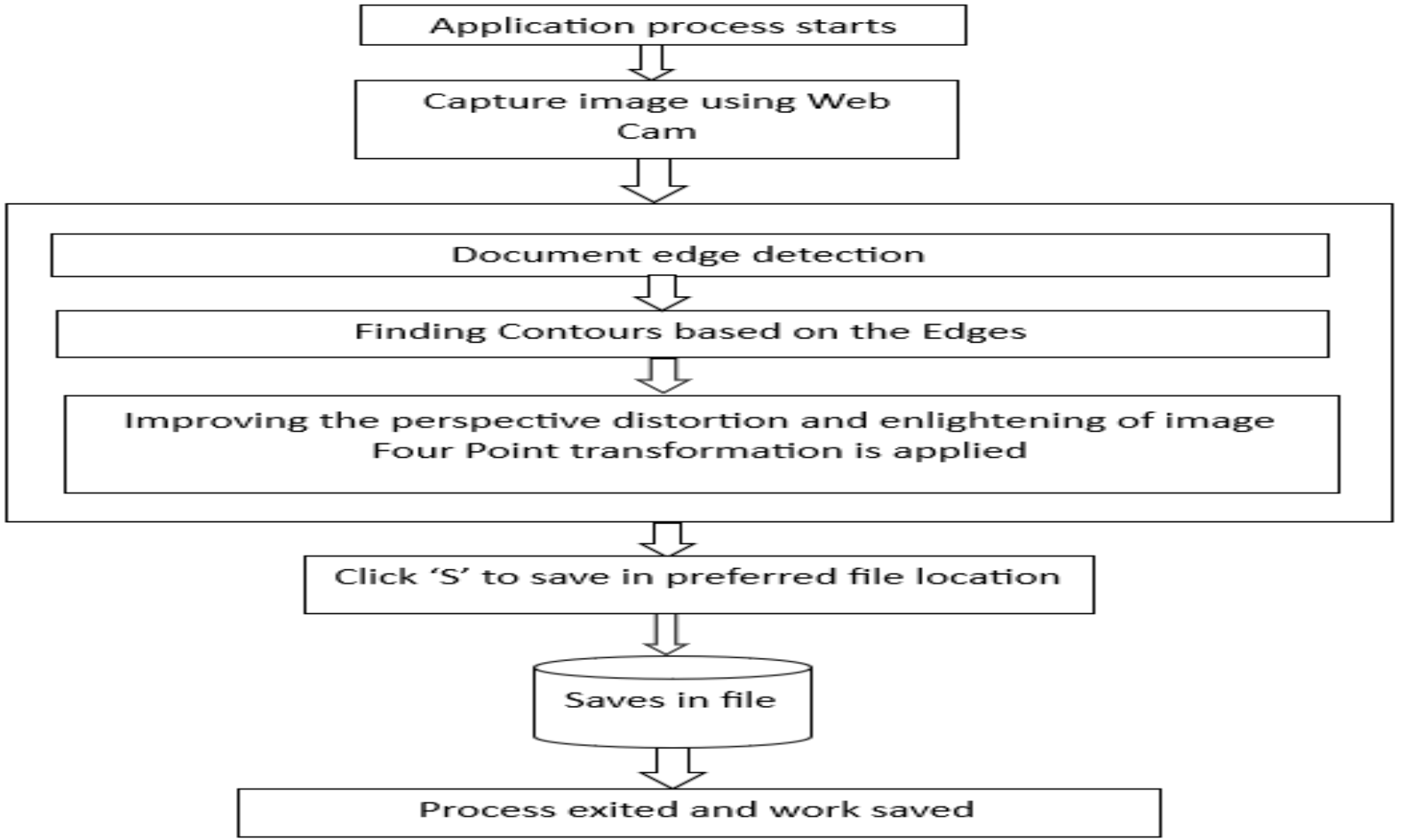
Robust Image Processing Algorithms : OpenCV provides a comprehensive suite of image processing algorithms, including edge detection, contour detection, and perspective transformation.

Flexibility and Customization : OpenCV provides a flexible framework that allows developers to customize scanning parameters to suit specific requirements.

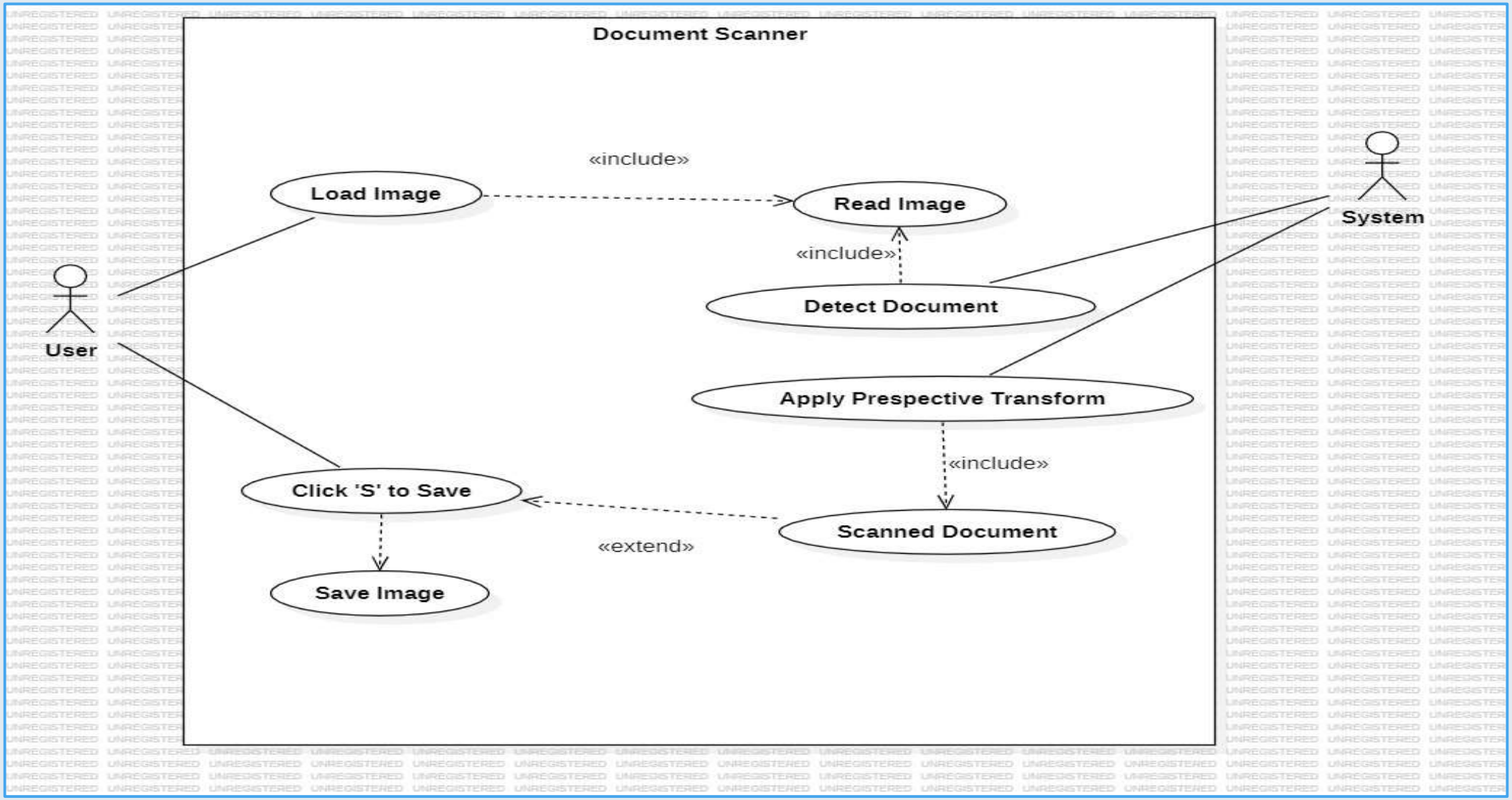
Integration with OCR and Document Analysis Libraries : OpenCV can be seamlessly integrated with Optical Character Recognition (OCR) libraries and document analysis tools to extract text, recognize handwriting, and analyze document structure.

Environmental Impact : By reducing the need for paper and physical storage space,document scanners helps to decrease the environmental impact of document handling and storage.

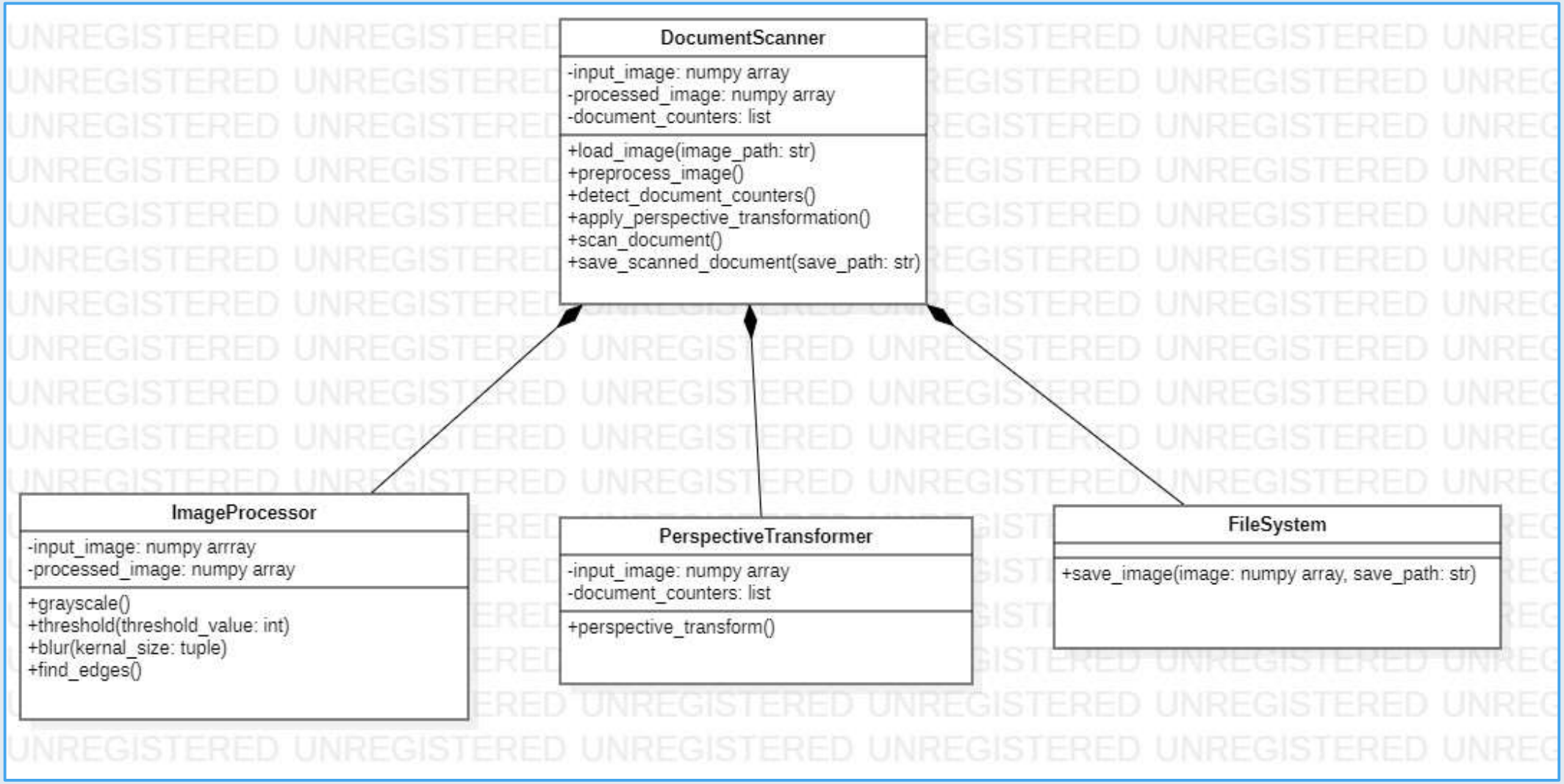
SYSTEM ARCHITECTURE



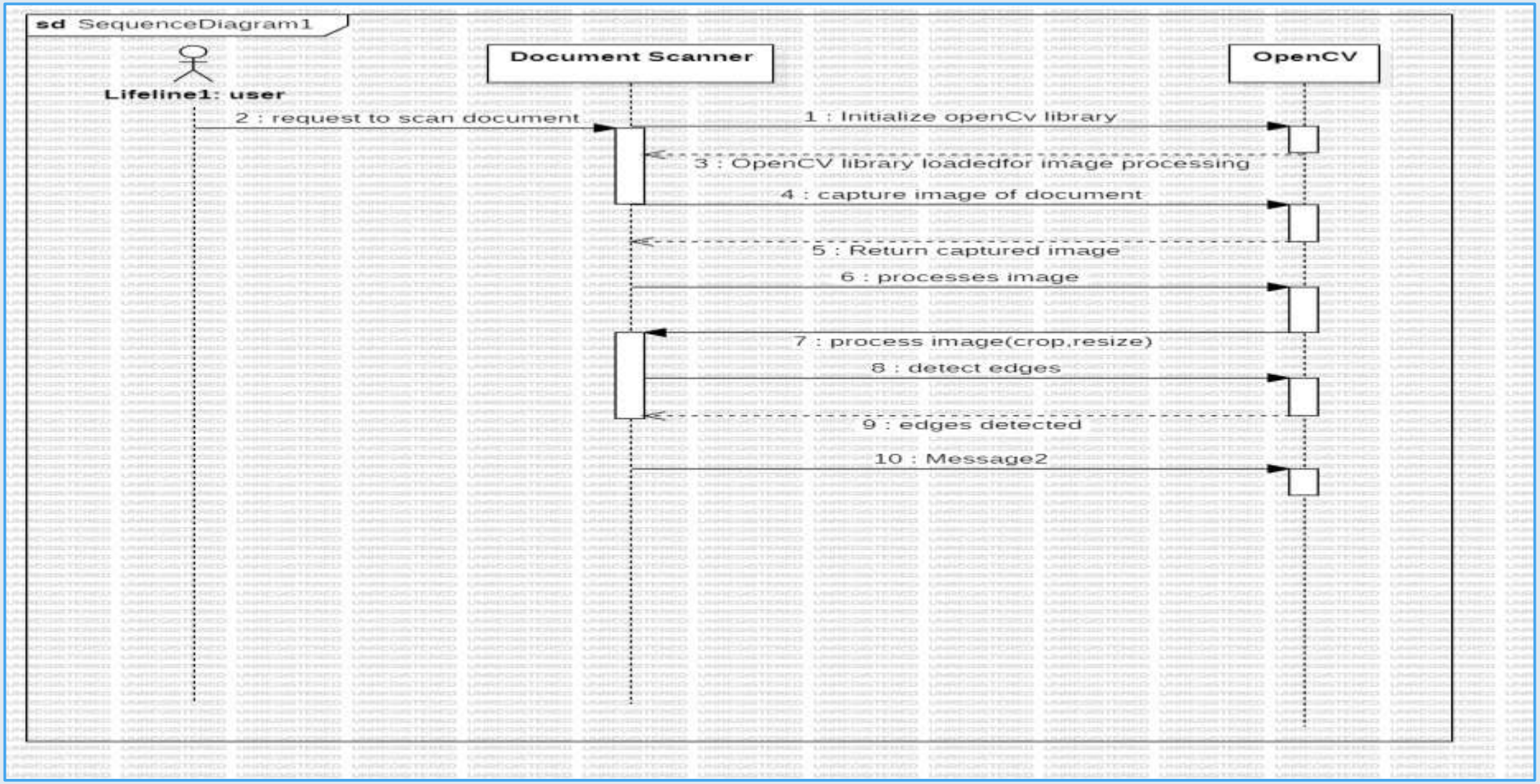
USE CASE DIAGRAM



CLASS DIAGRAM



SEQUENCE DIAGRAM



ALGORITHMS

- **OpenCV** : an open souce computer vision and machine leaning software library, can be used to detectr and recognize faces,identify objects,track moving objects,etc.
- **Image Preprocessing** : it manipulates images to make them more suitable for analysis like noise reduction , grayscale coversion.
- **Edge Detection** : Identify edges in the image using the Canny edge detector.
- **Contour Detection** : Find contours in the edge-detected image .
- **Perspective Transformation** : we can change the perspective of a given image for getting better insights into the required information.

SCREEENSHOTS AND PROJECT EXECUTION


```python
import cv2
import numpy as np
import utlis

########################################################################
webCamFeed = True
pathImage = "1.jpg"
cap = cv2.VideoCapture(1)
cap.set( propId: 10,  value: 160)
heightImg = 640
widthImg = 480
########################################################################

utlis.initializeTrackbars()
count = 0

while True:

    if webCamFeed:
        success, img = cap.read()
    else:
        img = cv2.imread(pathImage)
    img = cv2.resize(img,  dsize: (widthImg, heightImg))  # RESIZE IMAGE
    imgBlank = np.zeros( shape: (heightImg, widthImg, 3), np.uint8)  # CREATE A BLANK IMAGE FOR TESTING DEBUGING
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # CONVERT IMAGE TO GRAY SCALE
    imgBlur = cv2.GaussianBlur(imgGray,  ksize: (5, 5),  sigmaX: 1)  # ADD GAUSSIAN BLUR
    thres = utlis.valTrackbars()  # GET TRACK BAR VALUES FOR THRESHOLDS
    imgThreshold = cv2.Canny(imgBlur, thres[0], thres[1])  # APPLY CANNY BLUR
    kernel = np.ones((5, 5))
    imgDial = cv2.dilate(imgThreshold, kernel, iterations=2)  # APPLY DILATION
    imgThreshold = cv2.erode(imgDial, kernel, iterations=1)  # APPLY EROSION
```

SCREEENSHOTS AND PROJECT EXECUTION

BEFORE

AFTER


```
## FIND ALL COUNTOURS
imgContours = img.copy()  # COPY IMAGE FOR DISPLAY PURPOSES
imgBigContour = img.copy()  # COPY IMAGE FOR DISPLAY PURPOSES
contours, hierarchy = cv2.findContours(imgThreshold, cv2.RETR_EXTERNAL,
                                       cv2.CHAIN_APPROX_SIMPLE)  # FIND ALL CONTOURS
cv2.drawContours(imgContours, contours, -1, color: (0, 255, 0), thickness: 10)  # DRAW ALL DETECTED CONTOURS

# FIND THE BIGGEST COUNTOUR
biggest, maxArea = utlis.biggestContour(contours)  # FIND THE BIGGEST CONTOUR
if biggest.size != 0:
    biggest = utlis.reorder(biggest)
    cv2.drawContours(imgBigContour, biggest, -1, color: (0, 255, 0), thickness: 20)  # DRAW THE BIGGEST CONTOUR
    imgBigContour = utlis.drawRectangle(imgBigContour, biggest, 2)
    pts1 = np.float32(biggest)  # PREPARE POINTS FOR WARP
    pts2 = np.float32([[0, 0], [widthImg, 0], [0, heightImg], [widthImg, heightImg]])  # PREPARE POINTS FOR WARP
    matrix = cv2.getPerspectiveTransform(pts1, pts2)
    imgWarpColored = cv2.warpPerspective(img, matrix, dsize: (widthImg, heightImg))

    # REMOVE 20 PIXELS FORM EACH SIDE
    imgWarpColored = imgWarpColored[20:imgWarpColored.shape[0] - 20, 20:imgWarpColored.shape[1] - 20]
    imgWarpColored = cv2.resize(imgWarpColored, dsize: (widthImg, heightImg))

    # APPLY ADAPTIVE THRESHOLD
    imgWarpGray = cv2.cvtColor(imgWarpColored, cv2.COLOR_BGR2GRAY)
    imgAdaptiveThre = cv2.adaptiveThreshold(imgWarpGray, maxValue: 255, adaptiveMethod: 1, thresholdType: 1, blockSize: 7, C: 2)
    imgAdaptiveThre = cv2.bitwise_not(imgAdaptiveThre)
    imgAdaptiveThre = cv2.medianBlur(imgAdaptiveThre, ksize: 3)

    # Image Array for Display
    imageArray = ([img, imgGray, imgThreshold, imgContours],
                  [imgBigContour, imgWarpColored, imgWarpGray, imgAdaptiveThre])
```

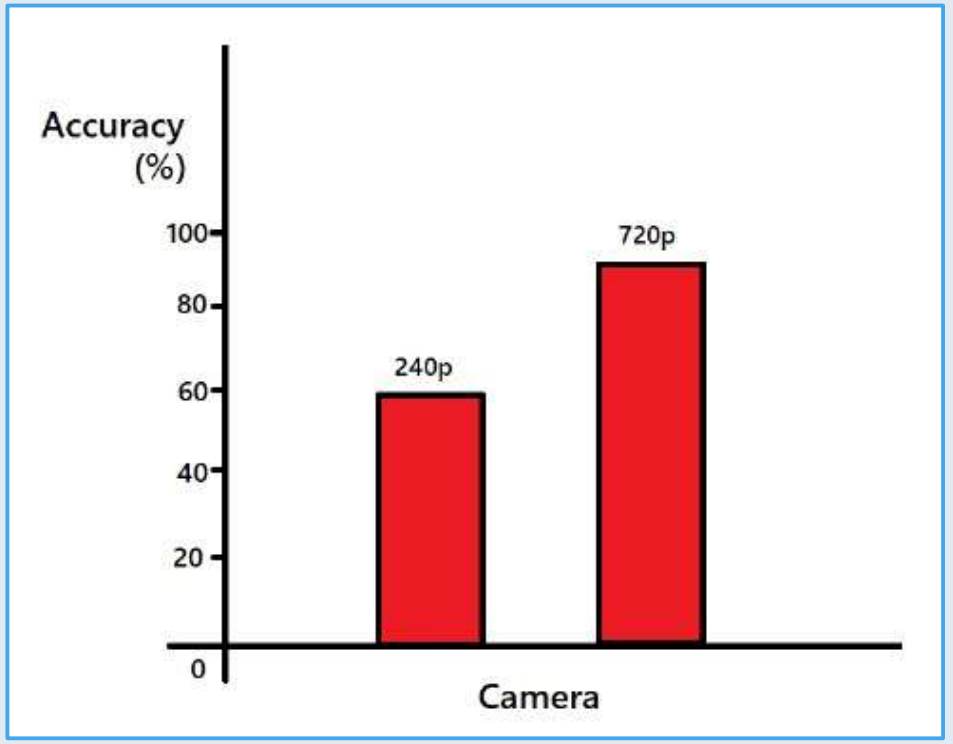
```
else:
    imageArray = ([img, imgGray, imgThreshold, imgContours],
                  [imgBlank, imgBlank, imgBlank, imgBlank])

# LABELS FOR DISPLAY
labels = [["Original", "Gray", "Threshold", "Contours"],
          ["Biggest Contour", "Warp Perspective", "Warp Gray", "Adaptive Threshold"]]

stackedImage = utlis.stackImages(imageArray, 0.75,labels)

cv2.imshow( winname: "Result", stackedImage)

# SAVE IMAGE WHEN 's' key is pressed
if cv2.waitKey(1) & 0xFF == ord('s'):
    cv2.imwrite("scanned_documents/doc" + str(count) + ".TIFF",
                imgAdaptiveThre)
    cv2.rectangle(stackedImage, ((int(stackedImage.shape[1] / 2) - 230), int(stackedImage.shape[0] / 2) + 50),
                  (1100, 350), (0, 255, 0), cv2.FILLED)
    cv2.putText(stackedImage, text: "Scan Saved", org: (int(stackedImage.shape[1] / 2) - 200, int(stackedImage.shape[0] / 2)),
                cv2.FONT_HERSHEY_DUPLEX, fontScale: 3, color: (0, 0, 255), thickness: 5, cv2.LINE_AA)
    cv2.imshow( winname: 'Result',  stackedImage)
    cv2.waitKey(300)
    count += 1
```

PERFORMANCE METRICS

- Precision : Precision measures the accuracy of the positive predictions.(83% - 87%)
- Recall : Recall measures the ability of the scanner to detect all relevant document contours.(85% - 95%)
- F1 score : The F1 score is the harmonic mean of precision and recall, providing a balanced measure of accuracy.(85% - 95%)
- Accuracy : Accuracy measures the overall correctness of the document scanner.(85% - 90%)

RESULT ANALYSIS

USE DIFFERENT CAMERAS TO CAPTURE IMAGES OF THE SAME DOCUMENT



CONCLUSION AND FUTURE SCOPE

The “Document Scanner using OpenCV" project has demonstrated the successful implementation of an efficient document scanning system. Through the utilization of OpenCV libraries and image processing techniques, the project has achieved the objective of automatically detecting, cropping, and enhancing document images.

There are several avenues for further enhancement and expansion-

Improved Document Detection : Implement advanced machine learning algorithms to enhance document detection accuracy, especially in scenarios with complex backgrounds or overlapping objects.

Multi-page Scanning : Extend the application to support the scanning of multiple pages in a single scan session, enabling users to create multi-page documents or PDF files.

Cloud Integration : Integrate cloud storage services to allow users to upload scanned documents directly to cloud platforms for easy access and sharing.

ANY QUERIES?