**PYTHON-BASED AUTOMATIC DIGITAL DOCUMENT SCANNER WITH OPENCV**

**A Project Report**

Submitted in partial fulfillment of the

Requirements for the award of the Degree of

### BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)

**By**

Joehan Misquitta

19021

**Under the esteemed guidance of**

**Mrs. Jyoti Mirani**

**Mrs. Manisha**

**Mr. Akshay Nimkar**

Logo

Description automatically generated

**DEPARTMENT OF INFORMATION TECHNOLOGY**

### RISHI DAYARAM AND SETH HASSARAM NATIONAL COLLEGE

### AND SETH WASSIAMULL ASSOMUL SCIENCE COLLEGE

***(Affiliated to University of Mumbai)***

**MUMBAI, 400050**

**MAHARASHTRA 2021**

### RISHI DAYARAM AND SETH HASSARAM NATIONAL COLLEGE

### AND SETH WASSIAMULL ASSOMUL SCIENCE COLLEGE,

***(Affiliated to University of Mumbai)***

**MUMBAI-MAHARASHTRA-400050**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

Logo

Description automatically generated

### CERTIFICATE

This is to certify that the project entitled, **"PYTHON-BASED AUTOMATIC DIGITAL DOCUMENT SCANNER WITH OPENCV"**, is bonafied work of **JOEHAN MISQUITTA** bearing Seat. No: **19021** submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

#### **Internal Guide** Coordinator

**External Examiner**

**Date: College Seal**

## Abstract

**Document digitization is the process of transforming paper documents into a digital format using various tools and techniques that computer systems may use to automate processes or workflows. It is regarded as the most important phase on the route to becoming a digital enterprise.**

Becoming a digital business is no longer an option. If the COVID19 pandemic has taught us anything, it is that we must be exceedingly agile to adapt to disruptive occurrences.

Document digitization enables people to locate information while sitting behind their workstations. For example, if a library in Mumbai has digitized its books and documents and made them publicly available, anyone, regardless of where they live, can read the available digital content.

Document digitization improves the efficiency of your organization’s operations and ensures that processes run smoothly. Digitized documents are easy to maintain, store, retrieve, and process. They reduce the costs, time, and effort required to manage physical records.

Digitizing paper documents means creating a digital version of them. This can be done by scanning the document and converting it into a image, or by using a document management system to create an [electronic version of the document](https://theecmconsultant.com/document-version-control/).

### ACKNOWLEDGEMENT

I take this opportunity to express my deepest gratitude to the Department of B.Sc.I.T. for nurturing and teaching us the nuances required for developing his project. Our Head of Department Mr. Dinesh Himatsinghani has been very helpful for making all facilities available for us. They always have watchful eye on the progress throughout our project. I am immensely thankful to them for their valuable and timely inputs.

Once again, I am grateful to my guide Mr. Akshay Nimkar without whom this project would be impossible. I owe her my greatest appreciation. I would like to thank her for the guidance, support, patience, and faith in our capabilities and giving me the flexibility in terms of working and reporting schedules.

This Project was a great learning and humbling experience and reflects my academic work at National College. I am sure that this experience will hold me a good stead when I begin my professional careers.

### DECLARATION

I hereby declare that the project entitled, “**Python-based automatic digital document scanner with OpenCV**” done at **Mumbai**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of

#### BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY) to be submitted as final semester project as part of our curriculum.

**Name and Signature of the Student**

### 

### TABLE OF CONTENTS

## Chapter 1: INTRODUCTION

## Chapter 2: SURVEY OF TECHNOLOGIES

## Chapter 3: REQUIREMENTS AND ANALYSIS

## Chapter 4: SYSTEM DESIGN

## Chapter 5: IMPLEMENTATION AND TESTING

## Chapter 6: RESULTS AND DISCUSSION

## Chapter 7: CONCLUSIONS

## Chapter 8: REFERENCES

## Tables of Figures

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No** |  | **Contents** | **Pg.no** |
| 1. |  | INTRODUCTION |  |
|  | 1.1 | Background |  |
|  | 1.2 | Objective |  |
|  | 1.3 | Purpose, Scope, and Applicability |  |
|  | 1.3.1 | Purpose |  |
|  | 1.3.2 | Scope |  |
|  | 1.3.3 | Applicability |  |
|  | 1.4 | Achievements |  |
|  | 1.5 | Organization of Report |  |
|  |  |  |  |
| 2. |  | SURVEY OF TECHNOLOGIES |  |
|  |  |  |  |
| 3. |  | REQUIREMENTS AND ANALYSIS |  |
|  | 3.1 | Problem Definition |  |
|  | 3.2 | Requirements Specifications |  |
|  | 3.3 | Planning and Scheduling |  |
|  | 3.4 | Software and Hardware Requirements |  |
|  | 3.5 | Preliminary Product Description |  |
|  | 3.6 | Conceptual Models |  |
|  |  |  |  |
| 4. |  | SYSTEM DESIGN |  |
|  | 4.1 | Basic Modules |  |
|  | 4.2 | Procedural Design |  |
|  | 4.2.1 | Logic Diagrams |  |
|  | 4.3 | User Interface Design |  |
|  | 4.4 | Security Issues |  |
|  | 4.5 | Test Cases Design |  |
|  |  |  |  |
| 5. |  | IMPLEMENTATION AND TESTING |  |
|  | 5.1 | Implementation Approaches |  |
|  | 5.2 | Coding Details and Code Efficiency |  |
|  | 5.2.1 | Code Efficiency |  |
|  | 5.3 | Testing Approach |  |
|  | 5.3.1 | Unit Testing |  |
|  | 5.3.2 | Integrated Testing |  |
|  | 5.3.3 | Beta Testing |  |
|  | 5.4 | Test Cases |  |
|  |  |  |  |
| 6. |  | RESULTS AND DISCUSSIONS |  |
|  | 6.1 | Test Reports |  |
|  | 6.2 | User Documentation |  |
|  |  |  |  |
| 7. |  | CONCLUSIONS |  |
|  | 7.1 | Conclusion |  |
|  | 7.1.1 | Significance of the System |  |
|  | 7.2 | Limitations of the System |  |
|  | 7.3 | Future Scope of the Project |  |
|  |  |  |  |
| 8. |  | REFERENCES |  |

**Chapter 1**

## Introduction

* 1. **Background:**

In reality, files and documents are the fuel of every organization’s operational infrastructure. Businesses invest a lot of money to ensure the safety and security of their files and documents. Nonetheless, it is one of the most crucial, as well as one of the most prone to overuse and destruction, parts of the workplace.

Digitized documents are easier to manage, store, secure, and retrieve and don’t need too much physical space. They represent the foundation of any digital transformation initiative that an organization takes.

* 1. **Objective:**

The objective of this project is to digitize documents. Documents that have been digitized are easier to maintain, store, secure, share and dispose of when necessary.

Manual document digitization is a time-consuming process. It needs the use of humans. It is a method of protecting your most crucial papers, images, and documents. However, as technology advances, it is now entirely done by automation and the use of computer systems.

* 1. **Purpose, Scope and Applicability:**
     1. **Purpose:**

Document digitization improves the efficiency of your organization’s operations and ensures that processes run smoothly. Digitized documents are easy to maintain, store, retrieve, and process. They reduce the costs, time, and effort required to manage physical records.

* + 1. **Scope:**

The Scope of this project is to digitize documents. Documents that have been digitized are easier to maintain, store, secure, share and dispose of when necessary.

* + 1. **Applicability:**

Here applicability refers to how applicable to the real world the application is,

Digitized documents are easier to manage, store, secure, and retrieve and don’t need too much physical space. They represent the foundation of any digital transformation initiative that an organization takes.

* 1. **Achievements:**

During the implementation of this project I learned how to use the OpenCV library in Python and host webapps using Streamlit

* 1. **Organization of the Report:**

In this report we discuss about system analysis, system design, implementation and testing, results and discussion, conclusion and future work and references

1. **System analysis**:

In this Section, We Discuss about technologies used in this project.

1. **Requirements and Analysis:**

This section Discusses about Problem definition, Requirements Specification, Planning and Scheduling, Software and Hardware Requirements and Conceptual Models.

1. **System Design:**

This Section includes Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode, and other documentation.

1. **Implementation and Testing:**

This Section includes core segments of code, Testing Methodologies and Testing Approach (i.e., Unit Testing and integration testing).

1. **Results and Discussion:**

This Section Discusses the results of implementation and testing of the system in the project.

1. **Conclusion and Future Work:**

This Section discusses the conclusion of the report and possible future work including further development of the system.

1. **References:**

This Section includes references to sources of information used in this report.

**Chapter 2**

### System Analysis

1. **Python:**

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

1. **OpenCV:**

OpenCV (Open-Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

1. **Streamlit:**

Streamlit is an open-source python framework for building web apps for Machine Learning and Data Science. We can instantly develop web apps and deploy them easily using Streamlit. Streamlit allows you to write an app the same way you write a python code. Streamlit makes it seamless to work on the interactive loop of coding and viewing results in the web app.

1. **NumPy:**

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

* NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
* The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
* NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
* A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, to efficiently use much (perhaps even most) of today’s scientific/mathematical Python-based software, just knowing how to use Python’s built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

**Chapter 3**

### Requirements and Analysis

**3.1 Problem Definition:**

Files and documents are the fuel of every organization’s operational infrastructure. Businesses invest a lot of money to ensure the safety and security of their files and documents. Nonetheless, it is one of the most crucial, as well as one of the most prone to overuse and destruction, parts of the workplace.

Digitized documents are easier to manage, store, secure, and retrieve and don’t need too much physical space. They represent the foundation of any digital transformation initiative that an organization takes.

**3.2 Requirements Specification:**

The system has the following system requirements:

1. Users should be able to upload images of their documents.

2. System Process the image and Scan the Document.

3. System should Provide a download link for the image of the document.

**3.3 Planning and Scheduling**

**Gantt chart:**

A Gantt chart is a type of bar chart that illustrates a project schedule.

This chart lists the tasks to be performed on the vertical axis,

and time intervals on the horizontal axis. The width of the horizontal bars in

the graph shows the duration of each activity. Gantt charts illustrate the start

and finish dates of the terminal elements and summary elements of a project.

Terminal elements and summary elements constitute the work breakdown

structure of the project. Modern Gantt charts also show the dependency (i.e.,

precedence network) relationships between activities. Gantt charts can be

used to show current schedule status using percent-complete shadings and a

vertical "TODAY" line as shown here. Gantt charts are sometimes equated with

bar charts. Gantt charts are usually created initially using an early start time

approach, where each task is scheduled to start immediately when its

prerequisites are complete. This method maximizes the float time available for

all tasks.

On a Gantt chart you can easily see:

* The start date of the project
* What the project tasks are
* Who is working on each task
* When tasks start and finish
* How long each task will take
* How the tasks group together, overlap and link with each other
* The finish date of the project.

A screenshot of a computer

Description automatically generated

**3.4 Software and Hardware Requirements:**

* **Software requirements:**
  + **Languages: Python**
  + **Packages: OpenCV, NumPy, Streamlit**
* **Hardware Requirements:**
* Windows PC
* 32GB ROM
* 4GB RAM

**3.5 Preliminary Product Description:**

Preliminary Product Description is to identify he requirements and objectives of the new system and to define the functions and operation of the application.

This software uses OpenCV to scan documents in images

**3.6 Conceptual Models**

1. **ER Diagram:**

An entity–relationship model (or ER model) describes interrelated things of

interest in a specific domain of knowledge. A basic ER model is composed of

entity types (which classify the things of interest) and specifies relationships

that can exist between entities (instances of those entity types). A relationship

is how the data is shared between entities. There are three types of

relationships between entities.

By defining the entities, their attributes, and showing the relationships

between them, an ER diagram illustrates the logical structure of databases.

ER diagrams are used to sketch out the design of a database.

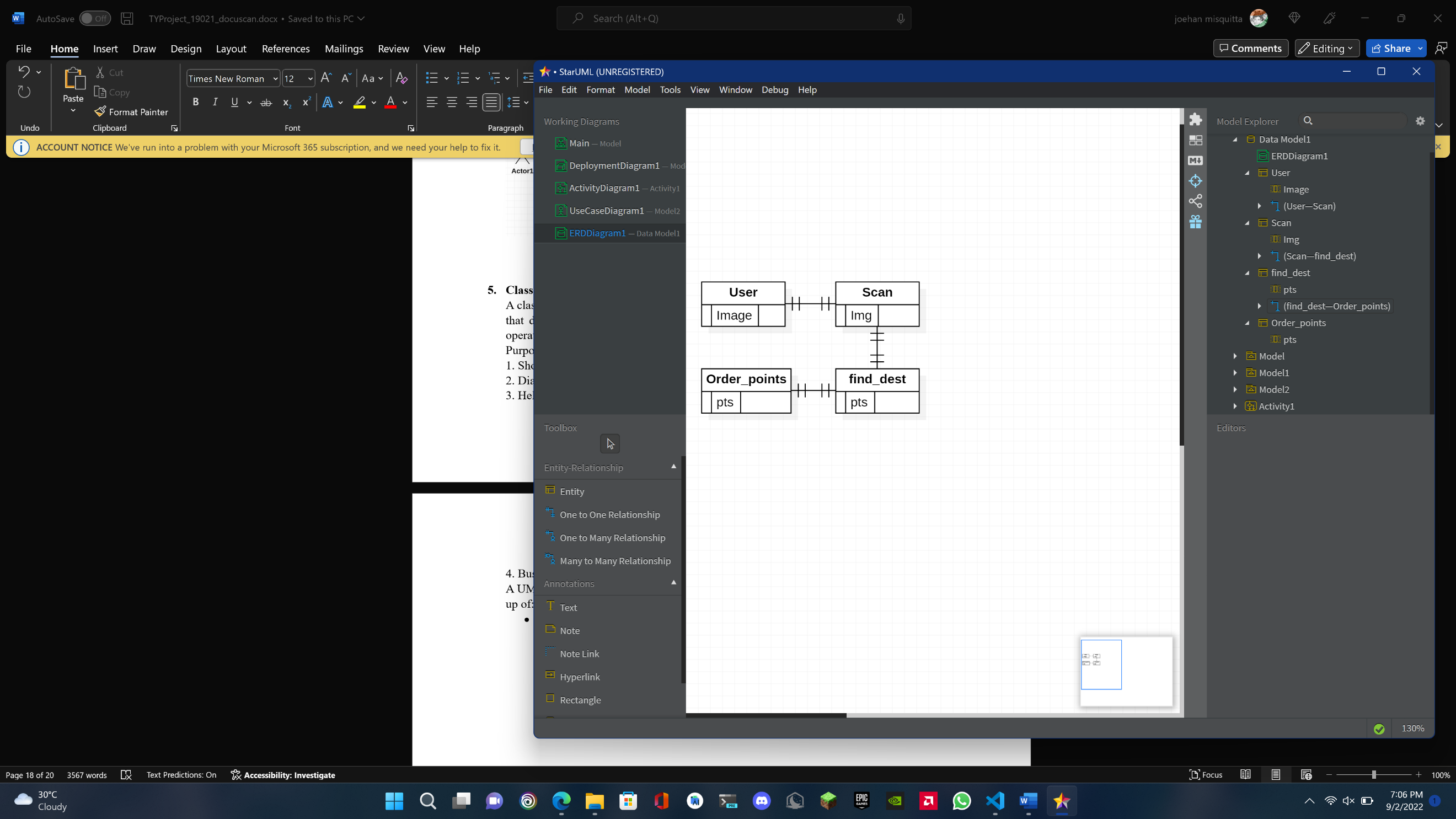
Limitations: -

* Limited relationship representation: ER model represents limited

relationship as compared to another data models like relational model etc.

* No representation of data manipulation: It is difficult to show data

manipulation in ER mode



1. **Deployment Diagram:**

Deployment diagram is a structure diagram which shows architecture of the

system as deployment (distribution) of software artifacts to deployment

targets.

Artifacts represent concrete elements in the physical world that are the result

of a development process. Examples of artifacts are executable files, libraries,

archives, database schemas, configuration files, etc.

Deployment target is usually represented by a node which is either hardware

device or some software execution environment. Nodes could be connected

through communication paths to create networked systems of arbitrary

complexity.

Note, that components were directly deployed to nodes in UML 1.x

deployment diagrams. In UML 2.x artifacts are deployed to nodes, and artifacts

could manifest (implement) components. Components are deployed to nodes

indirectly through artifacts.

Deployment diagrams could describe architecture at specification level (also

called type level) or at instance level (like class diagrams and object

diagrams).

Specification level deployment diagram shows some overview

of deployment of artifacts to deployment targets, without referencing specific

instances of artifacts or nodes.

Instance level deployment diagram shows deployment of instances

of artifacts to specific instances of deployment targets. It could be used for

example to show differences in deployments to development, staging or

production environments with the names/ids of specific build or deployment

servers or devices.

Purpose:

The term Deployment itself describes the purpose of the diagram. Deployment

diagrams are used for describing the hardware components, where software

components are deployed. Component diagrams and deployment diagrams are

closely related. Component diagrams are used to describe the components and deployment

diagrams show how they are deployed in hardware.

UML is mainly designed to focus on the software artifacts of a system. However,

these two diagrams are special diagrams used to focus on software and hardware

components. Most of the UML diagrams are used to handle logical components but deployment

diagrams are made to focus on the hardware topology of a system. Deployment

diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as −

• Visualize the hardware topology of a system.

• Describe the hardware components used to deploy software components.

• Describe the runtime processing nodes

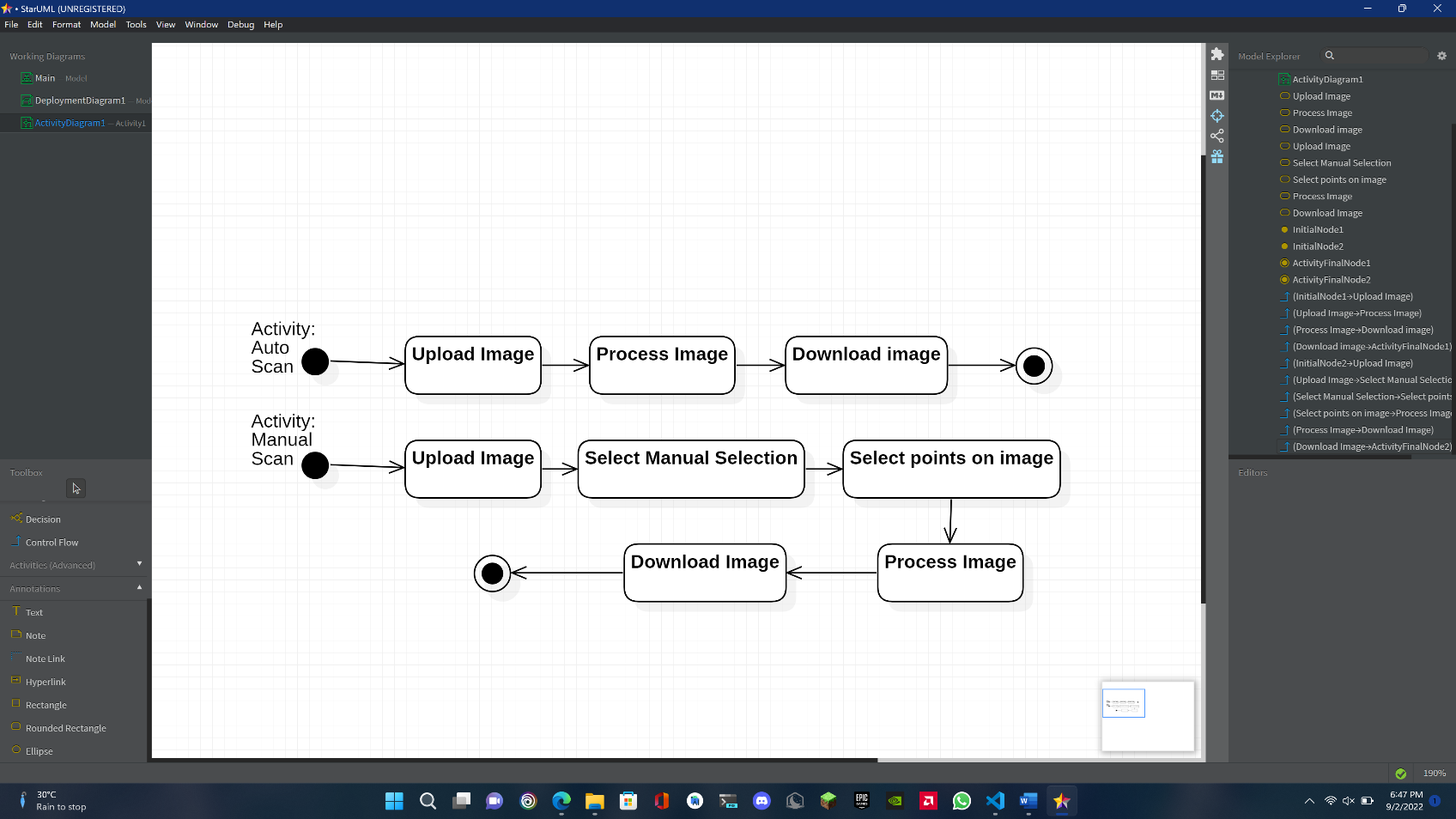
A screenshot of a computer

Description automatically generated

1. **Activity Diagram**

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

Purpose of Activity Diagrams: The basic purposes of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another



1. **Use Case Diagram**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

A use case diagram is usually simple. It does not show the detail of the use cases:

* It only summarizes some of the relationships between use cases, actors, and systems.
* It does not show the order in which steps are performed to achieve the goals of each use case.

A screenshot of a computer

Description automatically generated

1. **Class Diagram:**

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Purpose of Class Diagrams:

1. Shows static structure of classifiers in a system.

2. Diagram provides a basic notation for other structure diagrams prescribed by UML.

3. Helpful for developers and other team members too.

4. Business Analysts can use class diagrams to model systems from a business perspective

A screenshot of a computer

Description automatically generatedA UML class diagram is made up of:

* A set of classes and A set of relationships between classes

**Chapter 4**

### System Design

* 1. **Basic Modules**

Modules:

The project is divided into more manageable parts to develop each part or module separately. When all modules are ready, we integrate all the modules into one system.

These are the Modules of the Project:

* **Upload Image:**

1. **Upload Image of document**
2. **Browse files**

* **Adjust Manually**
* **Download file**

* 1. **Procedural Design**

Software Procedural Design (SPD) converts and translates structural elements into procedural explanations. SPD starts straight after data design and architectural design. This has now been mostly abandoned mostly due to the rise in preference of Object-Oriented Programming and design patterns The procedural design is often understood as a software design process that uses control commands such as: sequence, condition, repetition, which are applied to the predefined data. Sequences serve to achieve the processing steps in order that is essential in the specification of any algorithm. Conditions provide facilities for achieving selected processing according to some logical statement. Repetitions serve to achieve looping during the computation process. These three commands are implemented as ready programming language constructs. The programming languages that provide such command constructs are called imperative programming languages. The software design technique that relies on these constructs is called procedural design, or also structured design

* + 1. **Logic Diagrams**

Logic diagrams are diagrams in the field of logic, used for representation and to carry out certain types of reasoning.

**Control Flow Diagram**

**Graphical user interface, text

Description automatically generated**

* 1. **User Interface Design**

This is one of the main task of the developer to design a graphical user interface that the user attracts to and can use easily i.e. in one word it should be user-friendly. There should be a better understanding of the customer’s likes and dislikes and the features that are in trend and mesmerize the public easily. It is concerned with everything from starting the system or logging into the system to the eventual presentation of desired inputs and outputs.

Functions performed by the users

1. Upload image
2. Manual selection of endpoints in image (Optional).
3. Download Image
   1. **Security Issue**

There are no real Security issues with This project.

* 1. **Test Case Design**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test**  **Scenario** | **Test Data** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| **Optimal Image** |  | **Scanned Image** | As Expected, | **Pass** |
| **Un-Optimal Image** | **Text  Description automatically generated** | **Unable to properly process image** | As Expected, | **Pass** |

**Chapter 5**

**IMPLEMENTATION AND TESTING**

* 1. **Implementation Approach:**

Following are the steps to implement this project:

1. Install Python on your Computer
2. Install the Following dependencies/requirements:
3. OpenCV-python
4. NumPy
5. Streamlit
6. streamlit\_drawable\_canvas
7. Pillow
8. Matplotlib
9. Create a text document and paste the following command

python -m streamlit run c:\Users\joeha\OneDrive\Desktop\FinalYearProject\Project\app.py

pause

and save the file as a batch file ‘run.bat’

4. Double click the ‘run.bat’ file to run the application.

* 1. **Coding Details and Code Efficiency:**

1. **Imports**

import streamlit as st

import pathlib

from streamlit\_drawable\_canvas import st\_canvas

import cv2

import numpy as np

import io

import base64

from PIL import Image

1. **Function To Rearrange Corner Points:**

The order of the detected corners is not conventional. Therefore, we need a function that rearranges the four points in standard order.

def order\_points(*pts*):

    '''Rearrange coordinates to order:

       top-left, top-right, bottom-right, bottom-left'''

    rect = np.zeros((4, 2), *dtype*='float32')

*pts* = np.array(*pts*)

    s = *pts*.sum(*axis*=1)

    # Top-left point will have the smallest sum.

    rect[0] = *pts*[np.argmin(s)]

    # Bottom-right point will have the largest sum.

    rect[2] = *pts*[np.argmax(s)]

    diff = np.diff(*pts*, *axis*=1)

    # Top-right point will have the smallest difference.

    rect[1] = *pts*[np.argmin(diff)]

    # Bottom-left will have the largest difference.

    rect[3] = *pts*[np.argmax(diff)]

    # return the ordered coordinates

    return rect.astype('int').tolist()

1. **Function to find Destination Coordinates:**

Once you get the corner points for the documents, you only need the destination coordinates to perform perspective transform and align the documents.

Use the width and height of the document to align it with the original image and get the final destination of the document.

def find\_dest(*pts*):

    (tl, tr, br, bl) = *pts*

    # Finding the maximum width.

    widthA = np.sqrt(((br[0] - bl[0]) \*\* 2) + ((br[1] - bl[1]) \*\* 2))

    widthB = np.sqrt(((tr[0] - tl[0]) \*\* 2) + ((tr[1] - tl[1]) \*\* 2))

    maxWidth = max(int(widthA), int(widthB))

    # Finding the maximum height.

    heightA = np.sqrt(((tr[0] - br[0]) \*\* 2) + ((tr[1] - br[1]) \*\* 2))

    heightB = np.sqrt(((tl[0] - bl[0]) \*\* 2) + ((tl[1] - bl[1]) \*\* 2))

    maxHeight = max(int(heightA), int(heightB))

    # Final destination co-ordinates.

    destination\_corners = [[0, 0], [maxWidth, 0], [maxWidth, maxHeight], [0, maxHeight]]

    return order\_points(destination\_corners)

1. **The main function to scan the documents:**

def scan(*img*):

    # Resize image to workable size

    dim\_limit = 1080

    max\_dim = max(*img*.shape)

    if max\_dim > dim\_limit:

        resize\_scale = dim\_limit / max\_dim

*img* = cv2.resize(*img*, None, *fx*=resize\_scale, *fy*=resize\_scale)

    # Create a copy of resized original image for later use

    orig\_img = *img*.copy()

* 1. **Morphological operation:**

With cv2.morphologyEx(), using basic operations like erosion and dilation, you can perform advanced morphological transformations. In morphology, a **close operation** is nothing but dilation, followed by erosion. Repeating these close operations will get you a blank page. Here, We have repeated this operation 3 times.

 # Repeated Closing operation to remove text from the document.

    kernel = np.ones((5, 5), np.uint8)

*img* = cv2.morphologyEx(*img*, cv2.MORPH\_CLOSE, kernel, *iterations*=3)

* 1. **Removing the background with GrabCut:**

Once we have a blank document, the next step is to get rid of the background. We will use GrabCut to extract the foreground.

* It simply requires a bounding box around the object that is in the foreground, everything outside the bounding box is considered the background.
* GrabCut automatically rids you of all the background, even inside the bounding box. All you’re left with now is the foreground object.

Having users manually draw these bounding boxes won’t really be very ‘automatic’. So, we take the corner 20 pixels as the background, and GrabCut automatically determines the foreground and background, leaving us only with the document.

Here, the Variable **‘rect’** is the bounding box

 # GrabCut

    mask = np.zeros(*img*.shape[:2], np.uint8)

    bgdModel = np.zeros((1, 65), np.float64)

    fgdModel = np.zeros((1, 65), np.float64)

    rect = (20, 20, *img*.shape[1] - 20, *img*.shape[0] - 20)

    cv2.grabCut(*img*, mask, rect, bgdModel, fgdModel, 5, cv2.GC\_INIT\_WITH\_RECT)

    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

*img* = *img* \* mask2[:, :, np.newaxis]

* 1. **Edge Detection:**

Now that we have a blank page with no background, we perform edge detection. Use canny-edge detection to detect the edges of the document. The canny-edge detector can give the precise outline of the document.

* First convert the image of the blank page to grayscale, as canny works only on grayscale images.
* Then perform Gaussian Blur to remove noise from the image.
* Finally, do canny-edge detection on the image.
* Also, dilate the image to get a thin outline of the document.

    gray = cv2.cvtColor(*img*, cv2.COLOR\_BGR2GRAY)

    gray = cv2.GaussianBlur(gray, (11, 11), 0)

    # Edge Detection.

    canny = cv2.Canny(gray, 0, 200)

    canny = cv2.dilate(canny, cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5)))

* 1. **Contour Detection:**

After getting the edges of the document, perform contour detection to get the contours of these edges.

Once you’re done with contour detection:

* Sort the detected contours according to their size
* Keep only the largest-detected contour

# Finding contours for the detected edges.

    contours, hierarchy = cv2.findContours(canny, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_NONE)

    # Keeping only the largest detected contour.

    page = sorted(contours, *key*=cv2.contourArea, *reverse*=True)[:5]

* 1. **Detecting the Corner Points:**

Now that we have the contour for the edge, we use **‘cv2.approxPolyDP’** to get the corner points. This function approximates a curve or a polygon with another curve/polygon, with fewer vertices.

  # Detecting Edges through Contour approximation.

    # Loop over the contours.

    if len(page) == 0:

        return orig\_img

    for c in page:

        # Approximate the contour.

        epsilon = 0.02 \* cv2.arcLength(c, True)

        corners = cv2.approxPolyDP(c, epsilon, True)

        # If our approximated contour has four points.

        if len(corners) == 4:

            break

    # Sorting the corners and converting them to desired shape.

    corners = sorted(np.concatenate(corners).tolist())

Now we pass the corners variable through the **order\_points()** function and pass the result of that to the **find\_dest()** function.

# For 4 corner points being detected.

    corners = order\_points(corners)

    destination\_corners = find\_dest(corners)

* 1. **Transform Perspective to Align the Document:**

Now, use the detected corner points and the calculated destination points to do perspective transform and align the document.

# Getting the homography.

    M = cv2.getPerspectiveTransform(np.float32(corners), np.float32(destination\_corners))

    # Perspective transform using homography.

    final = cv2.warpPerspective(orig\_img, M, (destination\_corners[2][0], destination\_corners[2][1]),

*flags*=cv2.INTER\_LINEAR)

    return final

1. **Generate Download Link:**

# Generating a link to download a particular image file.

def get\_image\_download\_link(*img*, *filename*, *text*):

    buffered = io.BytesIO()

*img*.save(buffered, *format*='JPEG')

    img\_str = base64.b64encode(buffered.getvalue()).decode()

    href = f'<a href="data:file/txt;base64,{img\_str}" download="{*filename*}">{*text*}</a>'

    return href

1. **Streamlit WebApp**

# Set title.

st.sidebar.title('Document Scanner')

# Specify canvas parameters in application

uploaded\_file = st.sidebar.file\_uploader("Upload Image of Document:", *type*=["png", "jpg"])

image = None

final = None

col1, col2 = st.columns(2)

if uploaded\_file is not None:

    # Convert the file to an opencv image.

    file\_bytes = np.asarray(bytearray(uploaded\_file.read()), *dtype*=np.uint8)

    image = cv2.imdecode(file\_bytes, 1)

    manual = st.sidebar.checkbox('Adjust Manually', False)

    h, w = image.shape[:2]

    h\_, w\_ = int(h \* 400 / w), 400

    if manual:

        st.subheader('Select the 4 corners')

        st.markdown('### Double-Click to reset last point, Right-Click to select')

        # Create a canvas component

        canvas\_result = st\_canvas(

*fill\_color*="rgba(255, 165, 0, 0.3)",  # Fixed fill color with some opacity

*stroke\_width*=3,

*background\_image*=Image.open(uploaded\_file).resize((h\_, w\_)),

*update\_streamlit*=True,

*height*=h\_,

*width*=w\_,

*drawing\_mode*='polygon',

*key*="canvas",

        )

        st.sidebar.caption('Happy with the manual selection?')

        if st.sidebar.button('Get Scanned'):

            # Do something interesting with the image data and paths

            points = order\_points([i[1:3] for i in canvas\_result.json\_data['objects'][0]['path'][:4]])

            points = np.multiply(points, w / 400)

            dest = find\_dest(points)

            # Getting the homography.

            M = cv2.getPerspectiveTransform(np.float32(points), np.float32(dest))

            # Perspective transform using homography.

            final = cv2.warpPerspective(image, M, (dest[2][0], dest[2][1]), *flags*=cv2.INTER\_LINEAR)

            st.image(final, *channels*='BGR', *use\_column\_width*=True)

    else:

        with col1:

            st.title('Input')

            st.image(image, *channels*='BGR', *use\_column\_width*=True)

        with col2:

            st.title('Scanned')

            final = scan(image)

            st.image(final, *channels*='BGR', *use\_column\_width*=True)

    if final is not None:

        # Display link.

        result = Image.fromarray(final[:, :, ::-1])

        st.sidebar.markdown(get\_image\_download\_link(result, 'output.png', 'Download ' + 'Output'),

*unsafe\_allow\_html*=True)

**5.3 Testing Approaches**

**5.3.1 Unit Testing**

During the implementation for the system each module of the system was tested separately to uncover errors which its boundaries. User interface was used as a guide in this process. The validations have been done for all the inputs.

**5.3.2 Integrated Testing**

The objective of the Integrated Test is to take the until tested modules and build a program structure that has been defined in the design. We have done top-down integration, which is constructing and testing small segments where errors are easier to isolate and corrected. The integration process was performed in three steps:

* The main control was used as driver.
* Test was conducted as each module was integrated.
* Regret icon testing (conducting all or some of the previous test) to ensure that new errors have not been introduced.

**5.3.3 Beta Testing**

Beta testing also known as user testing takes place at the end users site by the end users to validate the usability, functionality, compatibility, and reliability testing. Beta testing adds value to the software development life cycle as it allows the "real" customer an opportunity to provide inputs into the design, functionality, and usability of a product. These inputs are not only critical to the success of the product but also an investment into future products when the gathered data is managed effectively.

* 1. **Test Cases Page**

A test case has components that describe input, action, and an expected response, to determine If a feature of an application is working correctly. A test case is a set of instructions on “HOW” to validate a particular test objective/target, which when followed will tell us if the expected behavior of the system is satisfied or not.

**Level 1:** In this level, you will try the basic cases from the available specification and user documentation.

**Level 2:** This is the practical stage in which writing cases will depend on actual functional and system flow of the application.

**Level 3:** This is the stage in which you will group some cases and write a test procedure. The test procedure is nothing but a group of small cases, maybe a maximum of 10.

**Level 4:** Automation of the project. This will minimize human interaction with the system and thus the QA can focus on the currently updated functionalities to test rather than remaining busy with Regression testing. Unit Test Plan/Cases should be made a separate deliverable. It should not be merged with other artifacts. Try to document all the probable test scenarios which encompasses uncommon and alternative flows. Once a project moves into construction phase, the developers tend to catch only the success situations or the situations or the situations which has been coded.

**Chapter 6**

**RESULT AND DISCUSSION**

* 1. **Test Reports**

1. **Result of Morphologic operation**

Code used to achieve the Output:

import cv2

import numpy as np

import matplotlib.pyplot as plt

def scan(*img*):

    # Resize image to workable size

    dim\_limit = 1080

    max\_dim = max(*img*.shape)

    if max\_dim > dim\_limit:

        resize\_scale = dim\_limit / max\_dim

*img* = cv2.resize(*img*, None, *fx*=resize\_scale, *fy*=resize\_scale)

    # Create a copy of resized original image for later use

    orig\_img = *img*.copy()

    # Repeated Closing operation to remove text from the document.

    kernel = np.ones((5, 5), np.uint8)

*img* = cv2.morphologyEx(*img*, cv2.MORPH\_CLOSE, kernel, *iterations*=5)

    return *img*

img = cv2.imread('inputs/img5.jpg')

scanned\_img = scan(img)

plt.imshow(scanned\_img)

print("scanned")

plt.show()

cv2.destroyAllWindows()

**Output for Morphologic operation:**

|  |  |
| --- | --- |
| **Before** | **After** |
|  | **Chart  Description automatically generated** |

1. **Result of GrabCut operation to remove Background After Morphologic operation:**

Code used to achieve the Output:

import cv2

import numpy as np

import matplotlib.pyplot as plt

def scan(*img*):

    # Resize image to workable size

    dim\_limit = 1080

    max\_dim = max(*img*.shape)

    if max\_dim > dim\_limit:

        resize\_scale = dim\_limit / max\_dim

*img* = cv2.resize(*img*, None, *fx*=resize\_scale, *fy*=resize\_scale)

    # Create a copy of resized original image for later use

    orig\_img = *img*.copy()

    # Repeated Closing operation to remove text from the document.

    kernel = np.ones((5, 5), np.uint8)

*img* = cv2.morphologyEx(*img*, cv2.MORPH\_CLOSE, kernel, *iterations*=5)

#GrabCut

mask = np.zeros(*img*.shape[:2], np.uint8)

    bgdModel = np.zeros((1, 65), np.float64)

    fgdModel = np.zeros((1, 65), np.float64)

    rect = (20, 20, *img*.shape[1] - 20, *img*.shape[0] - 20)

    cv2.grabCut(*img*, mask, rect, bgdModel, fgdModel, 5, cv2.GC\_INIT\_WITH\_RECT)

    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

*img* = *img* \* mask2[:, :, np.newaxis]

    return *img*

img = cv2.imread('inputs/img5.jpg')

scanned\_img = scan(img)

plt.imshow(scanned\_img)

print("scanned")

plt.show()

cv2.destroyAllWindows()

**Output for GrabCut operation:**

|  |  |
| --- | --- |
| **Before** | **After** |
| **Chart  Description automatically generated** | **Chart  Description automatically generated** |

1. **Result of Edge and Contour Detection after GrabCut Operation:**

Code used to achieve the Output:

import cv2

import numpy as np

import matplotlib.pyplot as plt

def scan(*img*):

    # Resize image to workable size

    dim\_limit = 1080

    max\_dim = max(*img*.shape)

    if max\_dim > dim\_limit:

        resize\_scale = dim\_limit / max\_dim

*img* = cv2.resize(*img*, None, *fx*=resize\_scale, *fy*=resize\_scale)

    # Create a copy of resized original image for later use

    orig\_img = *img*.copy()

    # Repeated Closing operation to remove text from the document.

    kernel = np.ones((5, 5), np.uint8)

*img* = cv2.morphologyEx(*img*, cv2.MORPH\_CLOSE, kernel, *iterations*=5)

#GrabCut

mask = np.zeros(*img*.shape[:2], np.uint8)

    bgdModel = np.zeros((1, 65), np.float64)

    fgdModel = np.zeros((1, 65), np.float64)

    rect = (20, 20, *img*.shape[1] - 20, *img*.shape[0] - 20)

    cv2.grabCut(*img*, mask, rect, bgdModel, fgdModel, 5, cv2.GC\_INIT\_WITH\_RECT)

    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

*img* = *img* \* mask2[:, :, np.newaxis]

        gray = cv2.cvtColor(*img*, cv2.COLOR\_BGR2GRAY)

    gray = cv2.GaussianBlur(gray, (11, 11), 0)

    # Edge Detection.

    canny = cv2.Canny(gray, 0, 200)

    canny = cv2.dilate(canny, cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (21, 21)))

    con = np.zeros\_like(*img*)

    # Finding contours for the detected edges.

    contours, hierarchy = cv2.findContours(canny, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)

    # Keeping only the largest detected contour.

    page = sorted(contours, *key*=cv2.contourArea, *reverse*=True)[:5]

    con = cv2.drawContours(con, page, -1, (0, 255, 255), 3)

    return con

img = cv2.imread('inputs/img5.jpg')

scanned\_img = scan(img)

plt.imshow(scanned\_img)

print("scanned")

plt.show()

cv2.destroyAllWindows()

**Output for Edge and Contour Detection:**

|  |  |
| --- | --- |
| **Before** | **After** |
| **Chart  Description automatically generated** | **Chart  Description automatically generated** |

1. **Result of all operations together:**

Code used to achieve the Output:

import cv2

import numpy as np

import matplotlib.pyplot as plt

def scan(*img*):

    # Resize image to workable size

    dim\_limit = 1080

    max\_dim = max(*img*.shape)

    if max\_dim > dim\_limit:

        resize\_scale = dim\_limit / max\_dim

*img* = cv2.resize(*img*, None, *fx*=resize\_scale, *fy*=resize\_scale)

    # Create a copy of resized original image for later use

    orig\_img = *img*.copy()

    # Repeated Closing operation to remove text from the document.

    kernel = np.ones((5, 5), np.uint8)

*img* = cv2.morphologyEx(*img*, cv2.MORPH\_CLOSE, kernel, *iterations*=5)

#GrabCut

mask = np.zeros(*img*.shape[:2], np.uint8)

    bgdModel = np.zeros((1, 65), np.float64)

    fgdModel = np.zeros((1, 65), np.float64)

    rect = (20, 20, *img*.shape[1] - 20, *img*.shape[0] - 20)

    cv2.grabCut(*img*, mask, rect, bgdModel, fgdModel, 5, cv2.GC\_INIT\_WITH\_RECT)

    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

*img* = *img* \* mask2[:, :, np.newaxis]

        gray = cv2.cvtColor(*img*, cv2.COLOR\_BGR2GRAY)

    gray = cv2.GaussianBlur(gray, (11, 11), 0)

    # Edge Detection.

    canny = cv2.Canny(gray, 0, 200)

    canny = cv2.dilate(canny, cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (21, 21)))

    con = np.zeros\_like(*img*)

    # Finding contours for the detected edges.

    contours, hierarchy = cv2.findContours(canny, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)

    # Keeping only the largest detected contour.

    page = sorted(contours, *key*=cv2.contourArea, *reverse*=True)[:5]

    con = cv2.drawContours(con, page, -1, (0, 255, 255), 3)

        if len(page) == 0:

        return orig\_img

    # loop over the contours

    for c in page:

        # approximate the contour

        epsilon = 0.02 \* cv2.arcLength(c, True)

        corners = cv2.approxPolyDP(c, epsilon, True)

        # if our approximated contour has four points

        if len(corners) == 4:

            break

    # Sorting the corners and converting them to desired shape.

    corners = sorted(np.concatenate(corners).tolist())

    # For 4 corner points being detected.

    # Rearranging the order of the corner points.

    corners = order\_points(corners)

    # Finding Destination Co-ordinates

    w1 = np.sqrt((corners[0][0] - corners[1][0]) \*\* 2 + (corners[0][1] - corners[1][1]) \*\* 2)

    w2 = np.sqrt((corners[2][0] - corners[3][0]) \*\* 2 + (corners[2][1] - corners[3][1]) \*\* 2)

    # Finding the maximum width.

    w = max(int(w1), int(w2))

    h1 = np.sqrt((corners[0][0] - corners[2][0]) \*\* 2 + (corners[0][1] - corners[2][1]) \*\* 2)

    h2 = np.sqrt((corners[1][0] - corners[3][0]) \*\* 2 + (corners[1][1] - corners[3][1]) \*\* 2)

    # Finding the maximum height.

    h = max(int(h1), int(h2))

    # Final destination co-ordinates.

    destination\_corners = order\_points(np.array([[0, 0], [w - 1, 0], [0, h - 1], [w - 1, h - 1]]))

    h, w = orig\_img.shape[:2]

    # Getting the homography.

    homography, mask = cv2.findHomography(np.float32(corners), np.float32(destination\_corners), *method*=cv2.RANSAC,

*ransacReprojThreshold*=3.0)

    # Perspective transform using homography.

    un\_warped = cv2.warpPerspective(orig\_img, np.float32(homography), (w, h), *flags*=cv2.INTER\_LINEAR)

    # Crop

    final = un\_warped[:destination\_corners[2][1], :destination\_corners[2][0]]

    return final

img = cv2.imread('inputs/img5.jpg')

scanned\_img = scan(img)

plt.imshow(scanned\_img)

print("scanned")

plt.show()

cv2.destroyAllWindows()

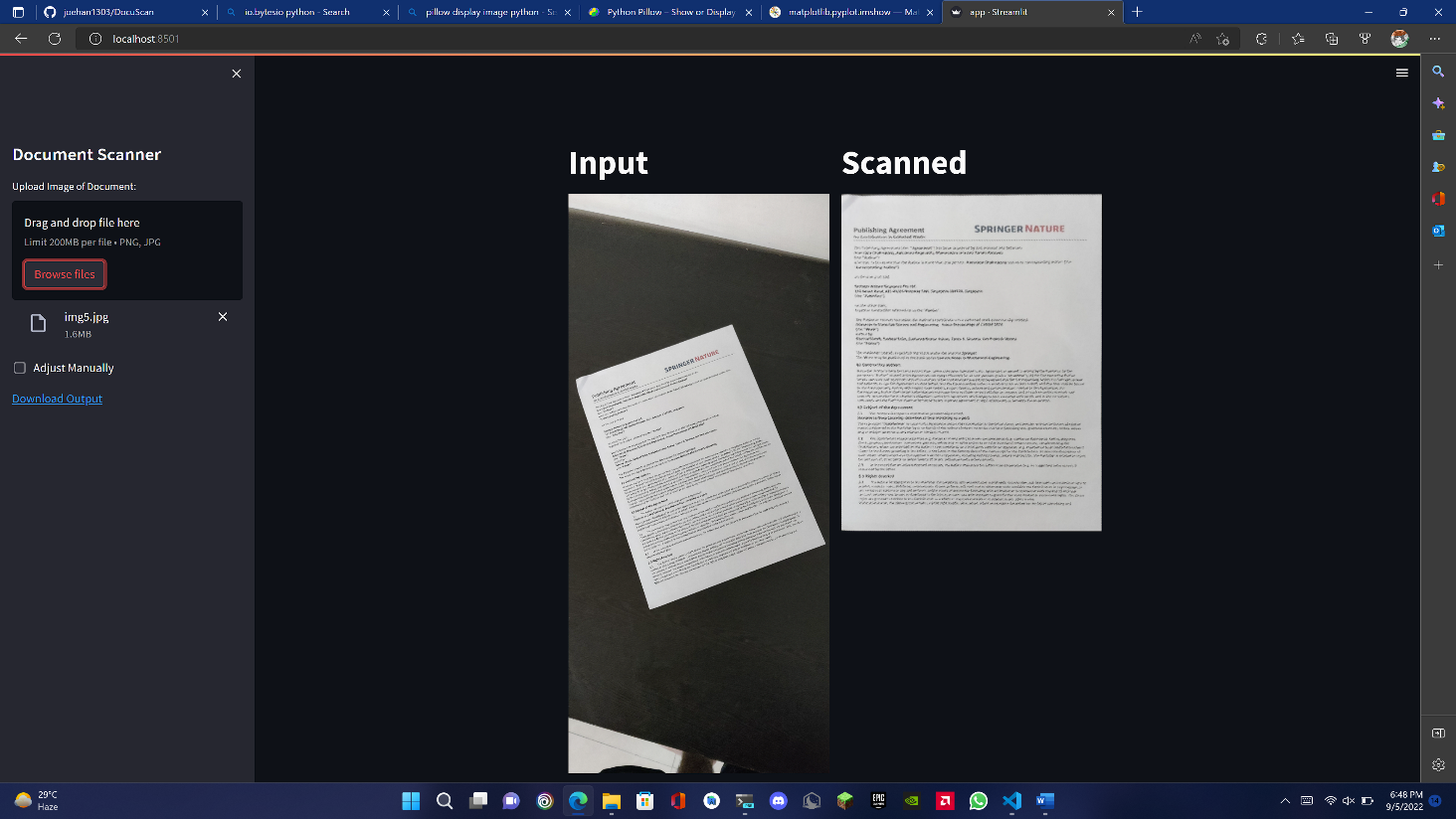
**Output for Edge and Contour Detection:**

|  |  |
| --- | --- |
| **Before** | **After** |
|  |  |

1. Final Output in the WebApp

Code used to achieve the Output: Refer to 5.2 Coding Details and Code Efficiency:

Output:



**Chapter 7**

**CONCLUSIONS**

* 1. **Conclusions**

1. **Significance of the system:**

Document digitization enables people to locate information while sitting behind their workstations. For example, if a library in Mumbai has digitized its books and documents and made them publicly available, anyone, regardless of where they live, can read the available digital content.

Document digitization improves the efficiency of your organization’s operations and ensures that processes run smoothly. Digitized documents are easy to maintain, store, retrieve, and process. They reduce the costs, time, and effort required to manage physical records.

* 1. **Limitations of the system.**

These are the current Limitations:

1. The current System is limited to processing one image at a time.
2. We can only download images in .jpg file format.
3. The System can only process .jpg and .png file formats.
   1. **Future scope of the system**

The Future Scope of this project:

1. Batch Processing using a file or a PDF file.
2. Add Option/Capability to download processed images in PDF format or a ZIP file
3. Add OCR Capability to convert images of documents to Word documents.
   1. If Appropriate automatically generate Excel Sheets from table form Data.

**Chapter 8**

**REFERENCES**

1. https://en.wikipedia.org/wiki/Python\_(programming\_language)
2. https://opencv.org/about/
3. https://numpy.org/doc/stable/user/whatisnumpy.html
4. https://medium.com/featurepreneur/what-is-streamlit-e106c6d9719d