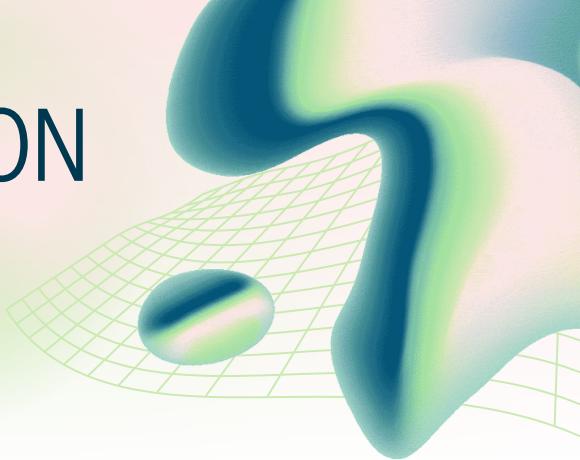


# IMAGE TRANSFORMATION - PROJECT REPORT

Submitted by: Manish, Navin, Nikhil



## Report Overview

This project reads a color image (captured with a phone camera and transferred to a computer), converts it to grayscale, resizes it to a manageable dimension while maintaining aspect ratio, stores the grayscale image as a 2D numerical matrix, and implements interactive geometric transformations: rotation, scaling, and translation.

The application is menu-driven and validates user inputs. The original and transformed images are displayed side-by-side with descriptive captions. The report also contains a template for camera technical specifications and implementation details.

## Objectives

1. Read an unedited image file from disk and convert it to grayscale.
2. Resize the image to a machine-appropriate size while preserving aspect ratio.
3. Save the grayscale image as a 2D matrix (NumPy array) to disk.
4. Implement rotation, scaling, and translation transformations in a menu-driven Python application.
5. Validate user inputs for transformation parameters.
6. Display original and transformed images side-by-side with captions.

## Hardware & Software Requirements

**Hardware:** Any modern PC (4+ GB RAM recommended). The image captured by a smartphone (any model)

**Software / Libraries:** Besides Libraries basic software requirements are also their like node.js and python

fastapi, uvicorn, opencv-python, pillow, numpy, python-multipart

## Camera Technical Specifications

**Phone model:** Realme Narzo 20A

**Camera module:** 12 MP, f/1.8, wide (*This is the primary lens in the triple camera setup*)

**Image resolution used for photo:** 4000 x 3000 (*This is the default full-resolution output for a 12 MP sensor*)

**File format:** e.g., JPEG (*no filters or edits applied*)

## About img.transform

**img.transform** is a lightweight web-based image transformation tool built using **FastAPI** and **React + TypeScript**. It allows users to upload an image and apply real-time transformations such as rotation, scaling, and translation — all directly in the browser.

The backend leverages **OpenCV** for powerful image processing, while the frontend provides an intuitive and interactive interface for visual experimentation. This project demonstrates how modern web frameworks and computer vision can combine to create practical and visually engaging tools.

The screenshot shows the homepage of the img.transform application. At the top, there is a dark header bar with the "img.transform" logo on the left and "Home About" links on the right. Below the header, there are two sections: "Original" and "Transformed". Each section has a dashed box placeholder with the text "Drop or click to upload" or "Transformed image will appear here". Underneath these sections, there is a "Transformation Parameters" section containing four input fields: "Rotation" (value 0), "Scaling" (value 1), "X-Translation" (value 0), and "Y-Translation" (value 0). To the right of these fields is a green "Transform" button.

*Homepage of img.transform*

# Code

---

**Complete Code**



It's complete code is available at GitHub Repository [GitHub - itsNavinSingh/img.transform](#)

Another Hosted link: <https://decodememe007.github.io/ImageTransformation/>

## main.py

```
from fastapi import FastAPI, File, UploadFile, Form

from fastapi.responses import Response

from fastapi.staticfiles import StaticFiles

from fastapi.middleware.cors import CORSMiddleware

import cv2

import numpy as np

import os


app = FastAPI(title="Image Transformation APP")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.post("/transform")
async def transform_image(
    file: UploadFile = File(...),
    rotation: float = Form(0.0),
    scale: float = Form(1.0),
    translate_x: float = Form(0.0),
    translate_y: float = Form(0.0),
):
    contents = await file.read()
```

```
np_img = np.frombuffer(contents, np.uint8)

img = cv2.imdecode(np_img, cv2.IMREAD_COLOR)

if img is None:

    return {"error": "Invalid image file"}

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

(h, w) = gray.shape[:2]

center = (w/2, h/2)

rotation_matrix = cv2.getRotationMatrix2D(center, rotation, scale)

rotation_matrix[0, 2] += translate_x

rotation_matrix[1, 2] += translate_y

transformed = cv2.warpAffine(gray, rotation_matrix, (w, h))

_, buffer = cv2.imencode(".jpg", transformed)

return Response(content=buffer.tobytes(), media_type="image/jpeg")

frontend_dir = os.path.join(os.path.dirname(__file__), "client", "dist")

if os.path.exists(frontend_dir):

    app.mount("/", StaticFiles(directory=frontend_dir, html=True), name="frontend")

if __name__ == "__main__":

    import uvicorn

    uvicorn.run("main:app", host="127.0.0.1", port=8080)
```

# Output

