Task: **Homography- Detection Tool(For Image detection)**

Name: Purva Pawaskar

Intern Id: 259

Objective:

To detect visual similarity between a legitimate UI (login page, app, etc.) and a suspicious/fake version using image feature matching (homography) — helping in identifying phishing or spoofed user interfaces.

Tools & Tech Used:

- Python
- OpenCV (ORB feature matching)
- Tkinter (GUI)
- NumPy, Pillow

How It Works (Steps):

1. Open the tool (a window will pop up)
2. Click buttons to upload:
    - One trusted image
    - One suspicious image
3. Click “Detect Spoof”
4. The tool:
    - Uses computer vision to match visual features
    - Shows a result like:
        1. Images look the same → probably safe
        2. Images look different → could be fake
5. It also shows a side-by-side view of how the two images match

Cybersecurity Use Case:

Detects phishing attempts or malicious clones where the attacker copies the look and feel of a real interface without changing much technically.

Output Example:

WARNING: Possible spoof detected!

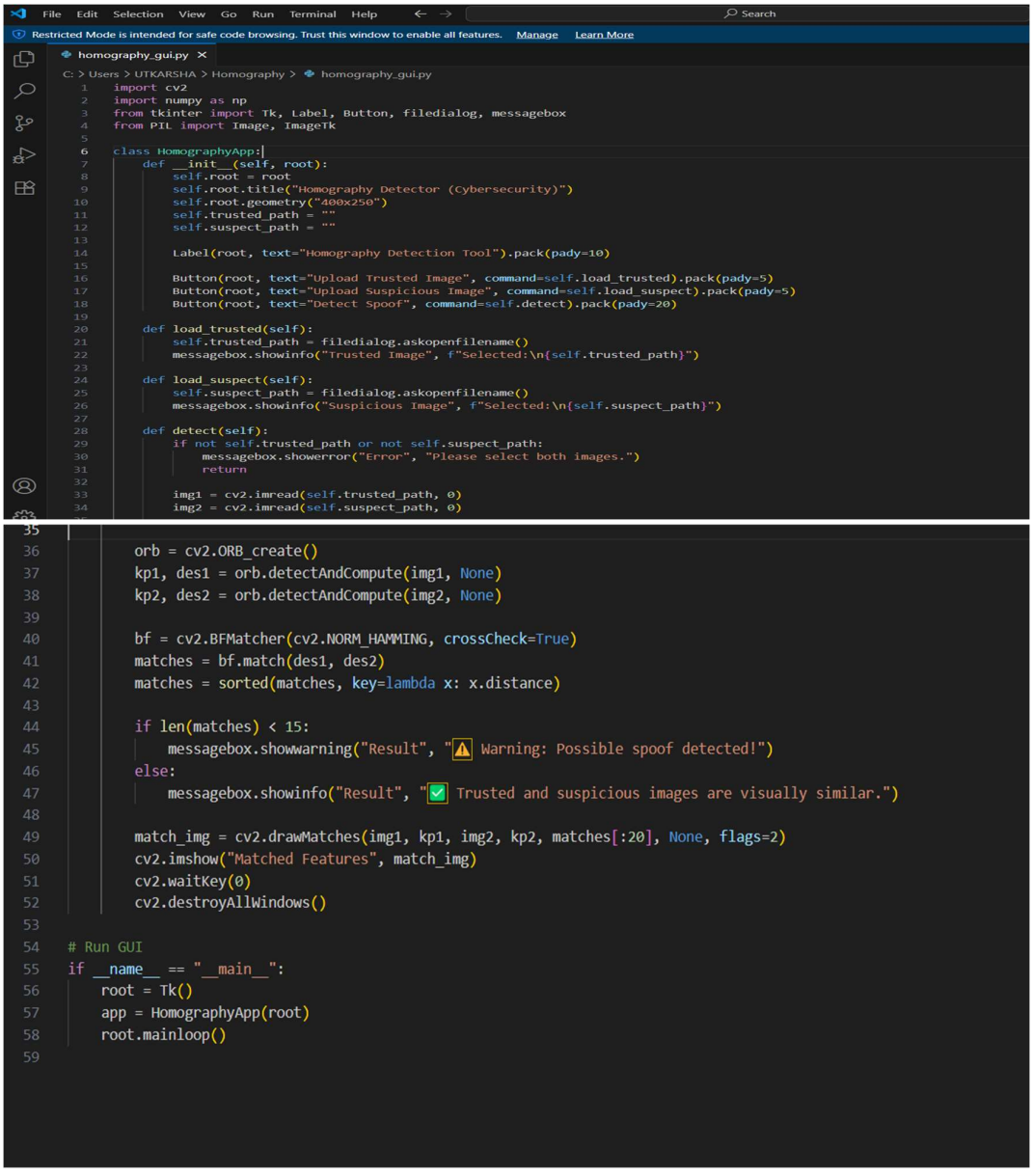
Or:

Images are visually similar.

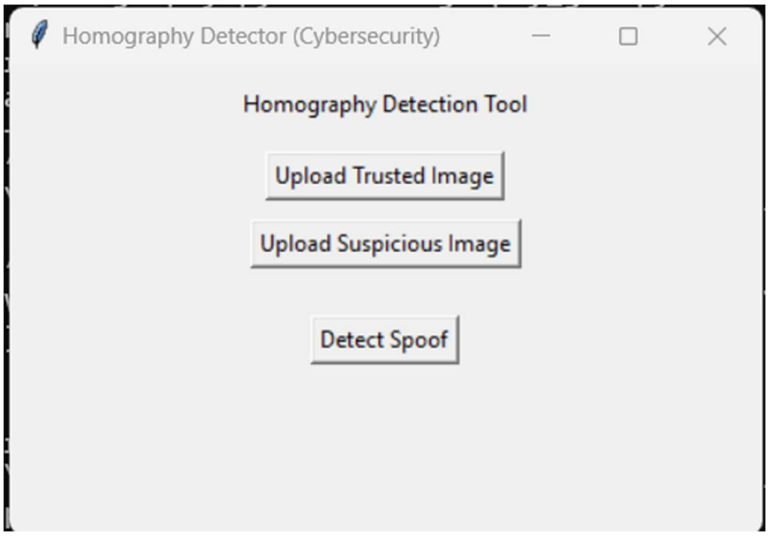
Plus, a window shows the matched key points between both images.

Files Involved:

- homography_gui.py → main Python GUI

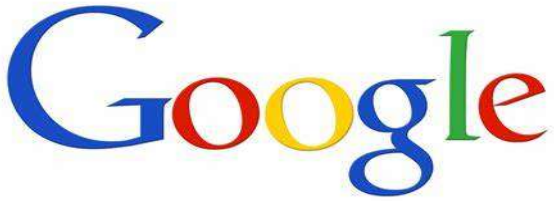


```python
import cv2
import numpy as np
from tkinter import Tk, Label, Button, filedialog, messagebox
from PIL import Image, ImageTk

class HomographyApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Homography Detector (Cybersecurity)")
        self.root.geometry("400x250")
        self.trusted_path = ""
        self.suspect_path = ""

        Label(root, text="Homography Detection Tool").pack(pady=10)

        Button(root, text="Upload Trusted Image", command=self.load_trusted).pack(pady=5)
        Button(root, text="Upload Suspicious Image", command=self.load_suspect).pack(pady=5)
        Button(root, text="Detect Spoof", command=self.detect).pack(pady=20)

    def load_trusted(self):
        self.trusted_path = filedialog.askopenfilename()
        messagebox.showinfo("Trusted Image", f"Selected:\n{self.trusted_path}")

    def load_suspect(self):
        self.suspect_path = filedialog.askopenfilename()
        messagebox.showinfo("Suspicious Image", f"Selected:\n{self.suspect_path}")

    def detect(self):
        if not self.trusted_path or not self.suspect_path:
            messagebox.showerror("Error", "Please select both images.")
            return

        img1 = cv2.imread(self.trusted_path, 0)
        img2 = cv2.imread(self.suspect_path, 0)

        orb = cv2.ORB_create()
        kp1, des1 = orb.detectAndCompute(img1, None)
        kp2, des2 = orb.detectAndCompute(img2, None)

        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
        matches = bf.match(des1, des2)
        matches = sorted(matches, key=lambda x: x.distance)

        if len(matches) < 15:
            messagebox.showwarning("Result", "⚠️ Warning: Possible spoof detected!")
        else:
            messagebox.showinfo("Result", "✅ Trusted and suspicious images are visually similar.")

        match_img = cv2.drawMatches(img1, kp1, img2, kp2, matches[:20], None, flags=2)
        cv2.imshow("Matched Features", match_img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

# Run GUI
if __name__ == "__main__":
    root = Tk()
    app = HomographyApp(root)
    root.mainloop()
```

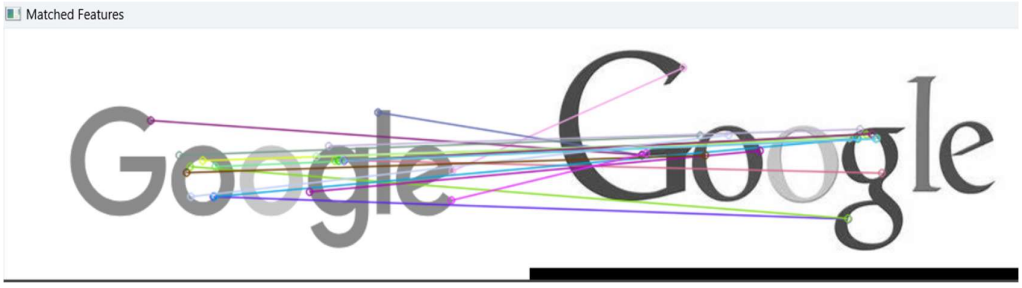


- trusted.png → real screenshot

- suspicious.png → screenshot to check



- Matches Key Point of the Image



Result:

- If image features are highly dissimilar → Spoof/fake alert
- If features are closely matching → Likely authentic

Summary:

- It uses a method called Homography, from computer vision
- It finds special key points in both images (like corners and edges)
- It compares those key points and counts how many match
- If only a few match, the images are likely different
- If many match, they're probably the same or copied