Computer Vision Assignment: Image Panorama with OpenCV

Goal

Create an image panorama using OpenCV inside a Jupyter notebook. You must capture your own photographs inside the IE Tower, embed them directly in the notebook as binary strings, and stitch them into a single panoramic image.

Late submission policy: Late submissions will be penalized with a **2% reduction of the total grade for every hour** the submission is late. This penalty applies immediately after the deadline.

Image Capture Requirements

- You must use **exactly 3 images** for the panorama.
- All images must be taken by **you**, inside the **IE Tower**.
- Each student must use **different photos**. Reusing or sharing images with classmates is not allowed.
- Choose a scene with visible texture and depth (corridors, lobby areas, atrium, staircases, study spaces).
- Capture images with approximately **30–50% overlap** between consecutive shots.
- When capturing, try to **only rotate the camera**. Do not translate or displace the camera sideways or forwards/backwards.
- Try to keep exposure and focus consistent across all images.

Notebook Requirements

- Submit a single **Jupyter notebook (.ipynb)**.
- The notebook must be fully self-contained and run without external image files.
- Include the three images inside the notebook as **binary (base64) strings**.
- Display all three images **before stitching**.
- Display the **stitched panorama** after stitching.
- Use OpenCV for the panorama implementation.

Embedding Images as Binary Strings (base64)

To make your notebook self-contained, embed your images directly in the notebook as text. A standard approach is to read each image file as raw bytes and convert it to a base64-encoded string. Later, decode the string back into bytes and then into an image array.

Step A: convert your 3 photo files into base64 strings
Place your three image files (e.g., img1.jpg, img2.jpg, img3.jpg) next to the notebook, run the code below, and copy/paste the printed variables into your submission notebook.

```
import base64
from pathlib import Path
```

```
def file_to_b64(path: str) -> str:
    data = Path(path).read_bytes()
    return base64.b64encode(data).decode("utf-8")

paths = ["img1.jpg", "img2.jpg", "img3.jpg"]  # change to your filenames
b64_strings = [file_to_b64(p) for p in paths]

for i, s in enumerate(b64_strings, 1):
    print(f"IMG_{i} = '''{s}'''\n")
```

Step B: decode base64 strings into OpenCV images
After pasting IMG_1, IMG_2, and IMG_3 into your notebook, decode them and pass the resulting images to your stitching pipeline.

```
import base64
import cv2
import numpy as np

def b64_to_bgr(b64_string: str):
    raw = base64.b64decode(b64_string)
    arr = np.frombuffer(raw, dtype=np.uint8)
    img = cv2.imdecode(arr, cv2.IMREAD_COLOR)
    if img is None:
        raise ValueError("Failed to decode image.")
    return img

images = [b64_to_bgr(IMG_1), b64_to_bgr(IMG_2), b64_to_bgr(IMG_3)]
```

Performance Tip: Downsampling High-Resolution Images

Modern phones often capture very high-resolution images. If stitching is slow or unstable, you may downsample the images before stitching (for example, resizing so the width is on the order of 800–1600 pixels). This is allowed and often improves robustness.

Expected Output in the Notebook

- A clear visualization of the three input images before stitching.
- A visualization of the final stitched panorama.

Grading Rubric (3 points total)

- **1 point**: Code quality and clarity (clean structure, readable, well-organized).
- **2 points**: Correct functionality (the three images are correctly stitched into a panorama).

Common Issues and Advice

- If stitching fails, increase overlap and avoid textureless surfaces.
- Reduce motion blur by holding the camera steady.
- Avoid moving objects close to the camera.
- Remember: rotate only; do not translate the camera.