Assignment 2: Computer Vision Report

- mohamed morsy 8199
- ahmed samir 8210
- youssef awad 8179

Part 1: Augmented Reality

Algorithm Explanation

The goal of this part is to create an augmented reality application by overlaying frames from a source video (ar_source.mov) onto a book detected in a target video (book.mov). [1]

1. **Feature Detection:** For the first frame of the book video, we detect SIFT (Scale-Invariant Feature Transform) keypoints and descriptors. We do the same for a static image of the book cover (cv_cover.jpg). [2]
2. **Feature Matching:** We use a Brute-Force (BF) matcher with KNN (k-Nearest Neighbors, k=2) to find potential matches. [3]
3. **Match Filtering:** Lowe's ratio test is applied to filter for good matches, keeping only those where the distance to the best match is significantly smaller (e.g., 75%) than the distance to the second-best match. [4]
4. **Homography Estimation:** Using the set of good matches, we compute the 3×3 homography matrix (H). [5] This matrix maps points from the book cover's plane to the book's plane in the video frame. We use cv2.findHomography with the RANSAC algorithm, which is robust to outliers (bad matches) that the ratio test might have missed.
5. **AR Frame Preparation:** Each frame from the ar_source.mov has a different aspect ratio than the book cover. [6]We crop the center of the AR frame to match the book's aspect ratio. [7]
6. **Video Processing:** We loop through every frame of both videos: [8]
   - **Per-Frame Homography:** We re-calculate the homography between the book cover and the *current* book video frame. [9]
   - **Warping:** We use the calculated homography H to warp the cropped AR frame so that its corners align perfectly with the detected book corners in the video frame.
   - **Overlay:** We create a mask from the warped AR frame and use it to replace the book's pixels in the video frame with the warped AR frame's pixels, creating the final AR effect. [10]
7. **Output:** All processed frames are written to a new output video file (ar_output.mov). [11]

Code Explanation (part1.py)

- get_sift_matches(img1, img2): Implements steps 1-3. It uses cv2.SIFT_create() and cv2.BFMatcher().knnMatch().
- compute_homography_ransac(pts_src, pts_dst): Implements step 4 using cv2.findHomography with cv2.RANSAC.
- crop_center_to_aspect(...): Implements step 5, cropping the AR frame to the target aspect ratio.
- warp_and_overlay(...): Implements step 6 (warping and overlaying). It uses cv2.warpPerspective to warp the AR frame and a corresponding mask to combine the images.
- **Main Script:** The main part of the script loads the videos, computes the initial homography for the first frame, and then loops through all subsequent frames to process and write the final video.

Part 2: Image Mosaics

Algorithm Explanation

This part focuses on stitching two images into a single panoramic mosaic. [12]We warp one image into the coordinate system (plane) of the second image. [13]

1. **Feature Detection & Matching:** Same as Part 1 (SIFT, KNN Matcher, Ratio Test) to find correspondences between the two input images (pano_image1.jpg, pano_image2.jpg). [14] We select the top 50 matches.
2. **Homography Estimation:** We write a function to compute the homography matrix H using the Direct Linear Transform (DLT) algorithm. [15]This involves setting up a system of linear equations ($Ax=b$) from at least 4 point correspondences. [16] The solution is found using Singular Value Decomposition (SVD) and reshaped into the 3×3 matrix H.
3. **Image Warping (Manual):** The warp_image function demonstrates inverse warping, as recommended to avoid holes. [17171717]It creates a grid of coordinates for the *output* (warped) image, maps these coordinates *back* to the *source* image using the inverse of H (H^{-1}), and samples the pixel values. [18]
4. **Interpolation:** Because the mapped coordinates ($H^{-1}p'$) are not integers, we use **bilinear interpolation** to find the correct color value from the 4 nearest pixels in the source image. [19]
5. **Mosaic Creation:** The create_mosaic function implements the final stitching:
    - It calculates the bounding box of the final mosaic by transforming the corners of img1 using H and finding the min/max coordinates relative to img2.
    - It creates a new image large enough to hold both views. [20]
    - It uses cv2.warpPerspective (a built-in, optimized function) to warp img1 into the final mosaic canvas.

- It copies img2 directly onto the canvas at the correct position, overlaying it. [21]

Code Explanation (part2.py)

- get_sift_matches(...): Same as Part 1, finds and filters SIFT matches.
- compute_homography(points_src, points_dst): Implements the manual DLT algorithm (Step 2) using np.linalg.svd.
- bilinear_interpolate(img, x, y): A helper function for Step 4.
- warp_image(img, H): A manual implementation of Step 3 (inverse warping).
- create_mosaic(img1, img2, H): Implements Step 5 using cv2.warpPerspective to create the final 03_mosaic.jpg.

Part 3: Bonus (3-Image Mosaic)

Algorithm Explanation

The bonus task is to extend the mosaic algorithm to stitch 3 images. [22]We follow the sequential process suggested in the assignment: [23]

1. **Stitch 1 & 2:** We first create a mosaic from pano_image1.jpg and pano_image2.jpg using the exact same steps as in Part 2. [24] This gives us an intermediate mosaic, mosaic_12.
2. **Stitch (1+2) & 3:** We now treat mosaic_12 as our new "source" image (img1) and pano_image3.jpg as our new "destination" image (img2).
3. **Find New Correspondences:** We run the SIFT matching algorithm again, this time finding features that match between mosaic_12 and pano_image3.jpg. [25]
4. **Compute New Homography:** We compute a *new* homography, $H_{\text{mosaic\_to\_3}}$, that maps points from the intermediate mosaic's plane to the third image's plane.
5. **Create Final Mosaic:** We call the create_mosaic function one more time, using mosaic_12, pano_image3.jpg, and $H_{\text{mosaic\_to\_3}}$ as inputs. [26] This produces the final 3-image panorama.