Computer Vision Homework 2

Corresponding Points Selection (select_corresponding_points_images):

1. This function allows the user to manually select pairs of corresponding points between two images. These points are used to establish a spatial relationship between the images.
2. By selecting specific points in each image, it ensures that there are sufficient matches for computing an accurate homography, which is the transformation that aligns the images.

Homography Computation (computeH):

1. This function calculates the homography matrix using the selected corresponding points. The homography matrix defines a transformation that maps points from one image to the other.
2. Using Singular Value Decomposition (SVD) on a system of linear equations, it finds the best transformation to align the two sets of points, allowing for accurate warping of one image to fit onto the other.

Steps in warp function:

1. **Calculate Corner Positions After Transformation**:
2. We take the four corners of the image (top-left, top-right, bottom-left, bottom-right) and apply the homography matrix to get the transformed corner positions in the new image.
3. This helps determine the bounding box for the warped image in the panorama.
4. **Determine Output Dimensions**:
5. We find the minimum and maximum x and y coordinates from the transformed corners, which define the new width and height of the warped image. This step also helps calculate offsets (x_offset and y_offset) to correctly position the warped image in the canvas.
6. These offsets represent how much the warped image needs to be shifted to fit within the new coordinate system.
7. **Create the Destination Canvas**:
8. We initialize a blank canvas (warped_image) with the calculated dimensions to hold the warped image. This ensures that the warped image is large enough to capture the entire transformed area.
9. **Generate a Coordinate Grid**:
10. Using np.meshgrid, we create a grid of x and y coordinates for the destination canvas.
11. The grid is then offset by the calculated x_offset and y_offset so that the warped image aligns correctly.
12. **Apply Inverse Homography for Mapping**:
13. We use the inverse homography matrix (H_inv) to map coordinates from the canvas back to the original image. This is necessary because it's easier to map from the transformed image back to the original coordinates than directly map every pixel from the original image to the transformed space.
14. For each point in the destination grid, we calculate the corresponding point in the original image using the inverse homography. This produces src_x and src_y, which are the x and y coordinates in the original image.
15. **Remapping Using OpenCV**:
16. Finally, we use cv2.remap with bilinear interpolation to map the original image pixels to the warped image. This step ensures a smooth transformation by interpolating pixel values.
17. The cv2.remap function fills the destination canvas (warped_image) with the transformed pixel values from the original image.

Steps in blend_images function:

1. **Calculate Canvas Size and Offsets**:
2. Using the dimensions of the base image and warped image, we determine the canvas size. We consider the minimum and maximum x and y values, which account for the offsets.
3. The canvas is sized to handle both images without cropping, ensuring they fit side by side or overlap correctly.
4. **Initialize the Canvas**:
5. We create a blank canvas (canvas) with the calculated width and height, filled with zeros (black background). This canvas will hold the blended images.
6. **Place the Base Image on the Canvas**:
7. We calculate offsets (x_base_offset and y_base_offset) to correctly position the base image on the canvas.
8. The base image is then directly copied to the canvas at its respective offset.
9. **Overlay the Warped Image**:
10. We calculate the position of the warped image on the canvas using x_warped_offset and y_warped_offset.
11. To handle overlapping areas, we create a mask (mask_warped) for the warped image. The mask identifies non-black pixels in the warped image (areas with content).
12. **Blend Using Mask**:
13. We use the mask to overlay the warped image onto the canvas only where it has non-zero values (i.e., ignoring black background pixels in the warped image).
14.

Stitching Two Images (stitch_two_images):

1. Combines all previous functions to stitch two images together.

First out then middle

Final image

Mosaic left

Mosaic right

Left to right

Mosaic 1

Mosaic 2

I think middle out and first out then middle is better than left to right because the shape becomes more consistent.

Mosaic 3

Mosaic 4

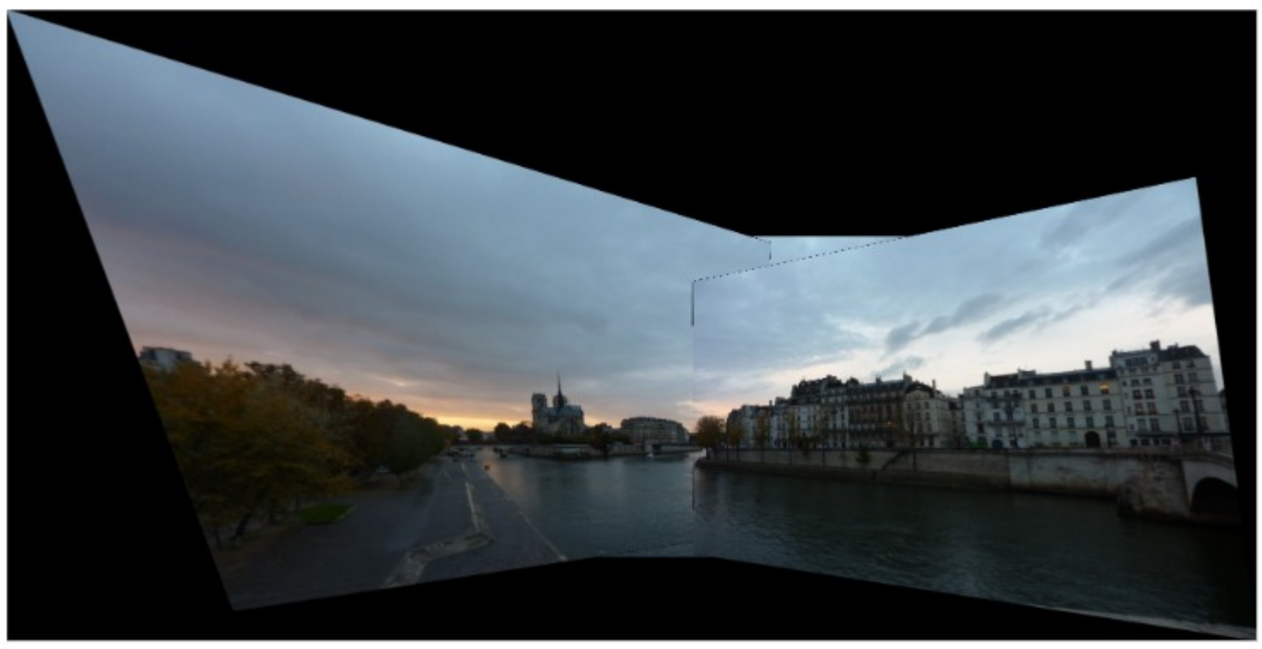
Middle out

Mosaic 1

Mosaic 2

Paris Image Set

10 points



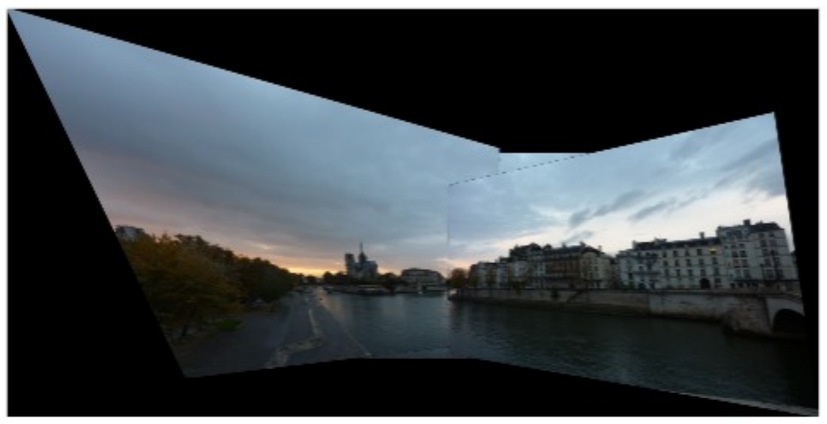
5 points

5 wrong points

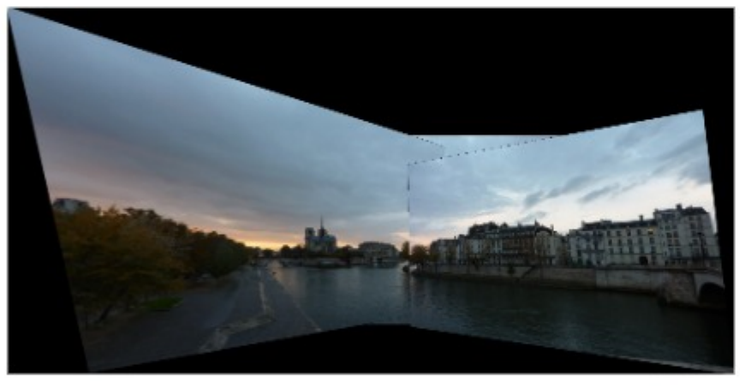
Number of points effect a lot because It determines the homography matrix. If less points exist then it becomes wrong. For better results Choose points on edges, corners, or unique patterns to ensure they’re easily distinguishable and consistent across images

Noisy Points

Variance = 2



Variance = 1



Variance = 0.5

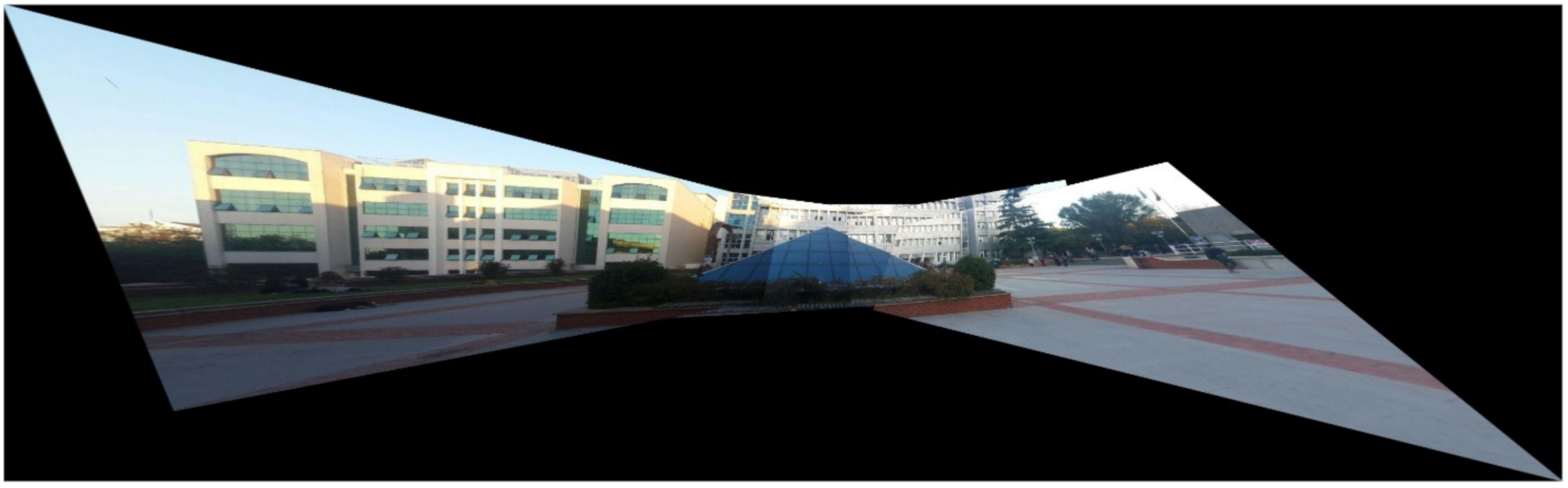
Since these are random generated it changes at every iteration. I set different varainces to selected points. and the effects is like in the images Adding noise simulates real-world inaccuracies in point detection However, high noise levels can distort the homography

Normalized

To improve numerical stability, we normalize the points so they’re centered around the origin and scaled. After computing HHH, we denormalize it to match the original coordinates.

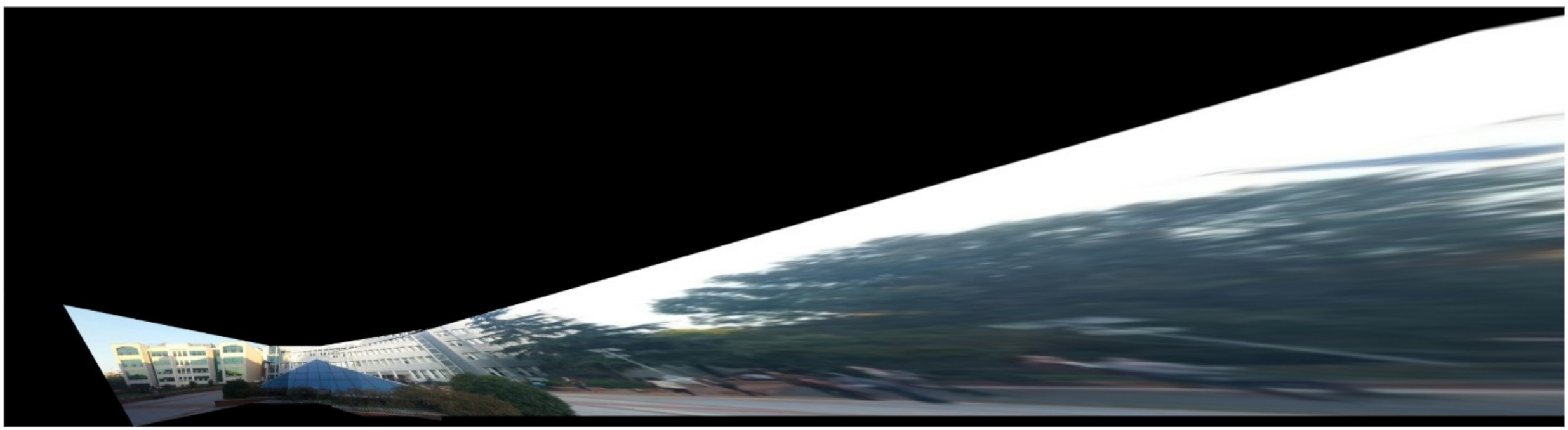
For good selection of points
RANSAC can help filter outliers by fitting a homography model on different subsets of points
We should distribute points across the entire overlapping area rather than clustering them in one region

North Campus



Middle out method

ın the left to right of north campus there was memmory issues so I couldnt tried it. Even I resized the image but It was impossible to do. The size got too high because of images. And ram got full its broked.



First out then middle method