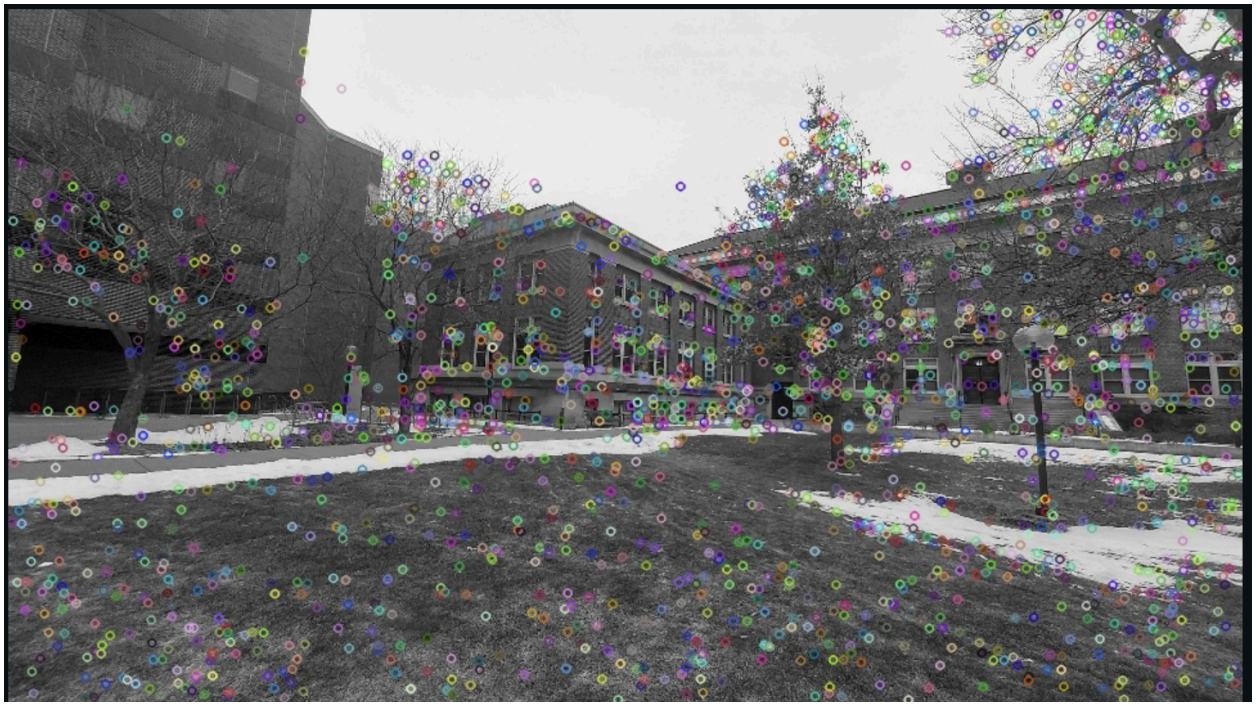
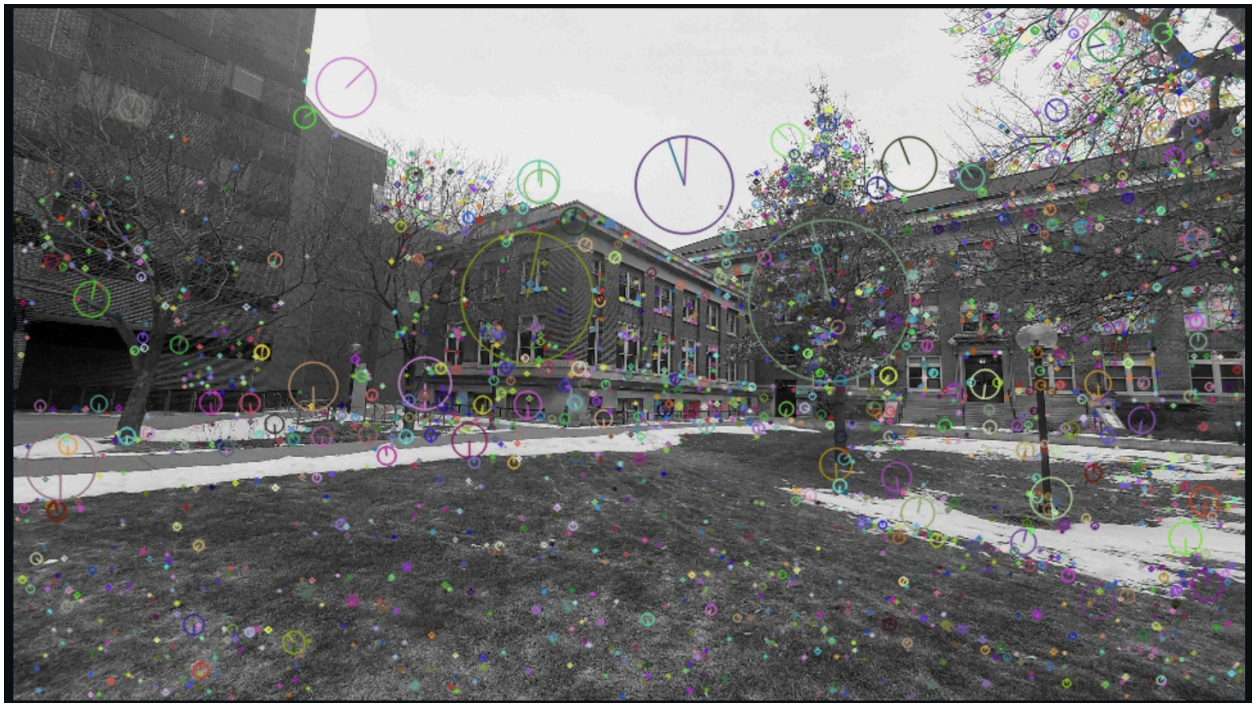
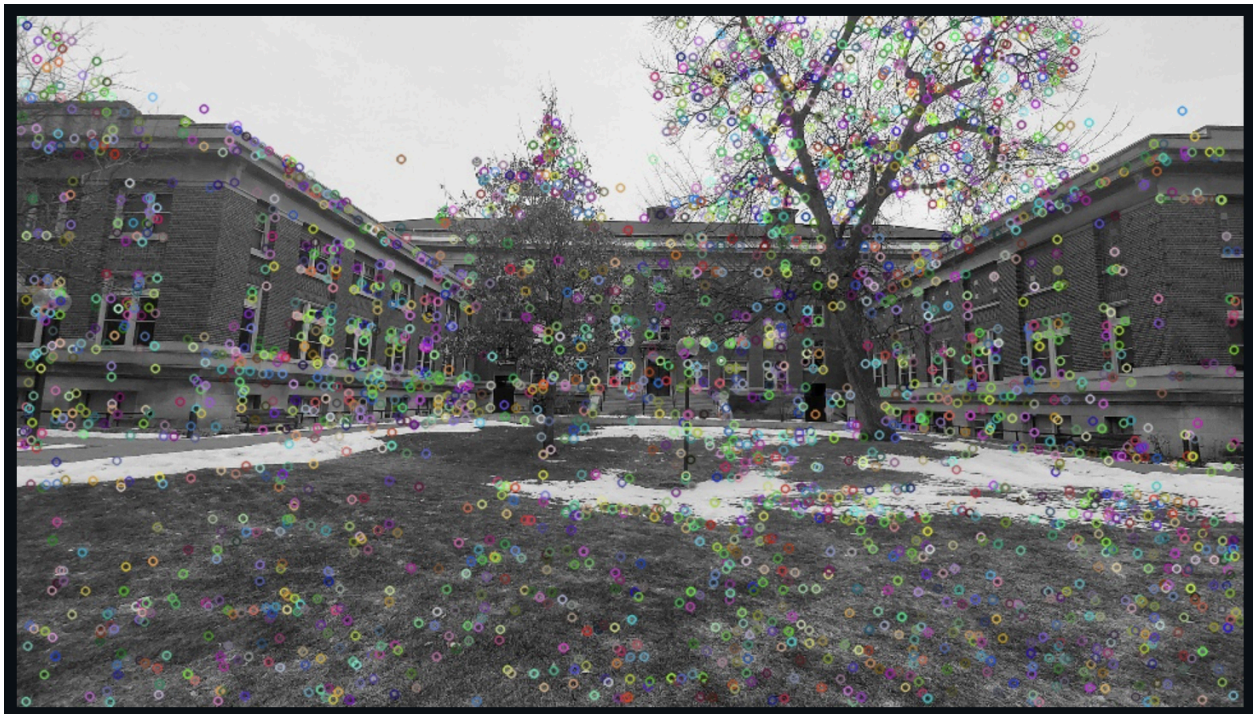
Q2)
1)

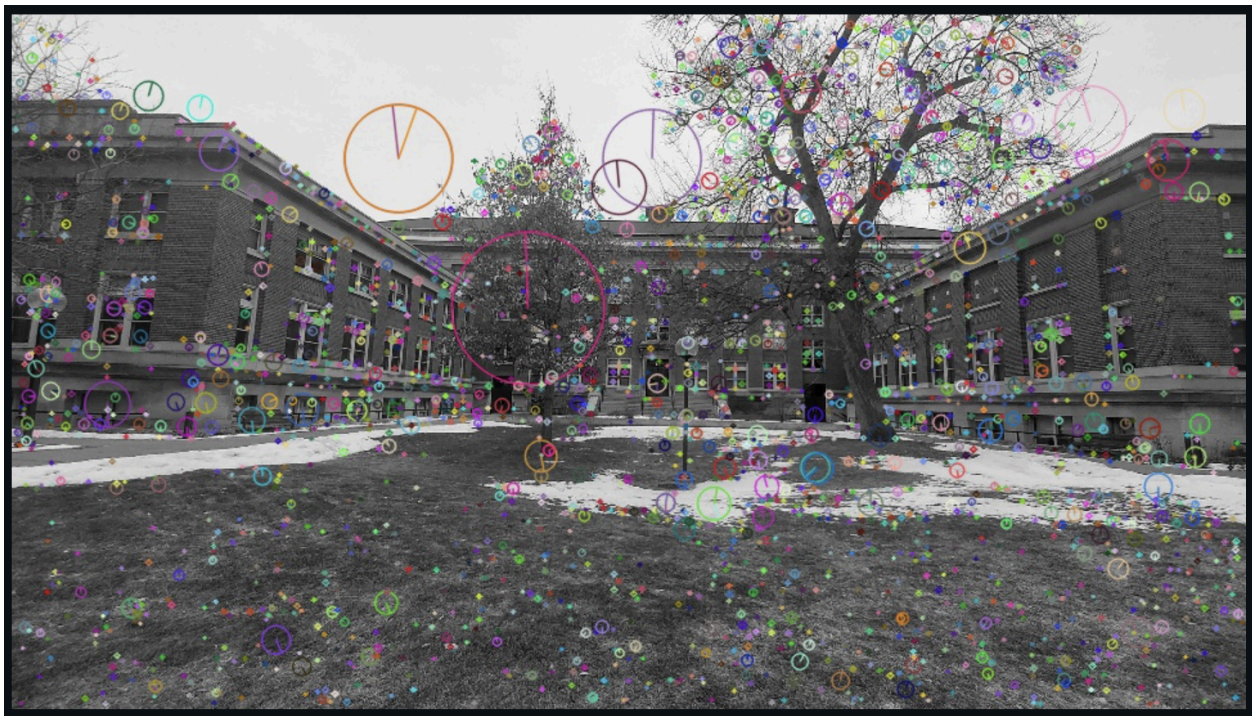
```
sift_dir = "E:\Sem 6\CV\Assignment 3\sift_keypts"
for i in img_dir :
    if "1" in i or "2" in i:
        print(i)
        img_pth = os.path.join(img_root,i)

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

        sift = cv2.SIFT_create()
        kp = sift.detect(gray,None)

        img=cv2.drawKeypoints(gray,kp,img)
        save = os.path.join(sift_dir,f"{i}")
        cv2.imwrite(save,img)


img=cv2.drawKeypoints(gray,kp,img,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_K
EYPOINTS)
        save2 = os.path.join(sift_dir,f"rich_{i}")
        print(save2)
        cv2.imwrite(save2,img)

        sift = cv2.SIFT_create()
        kp, des = sift.detectAndCompute(gray,None)

        txt_file = os.path.join(sift_dir,f"des_{i[0]}.txt")
        np.savetxt(txt_file,des)
        print(kp)
```

Img 1 :









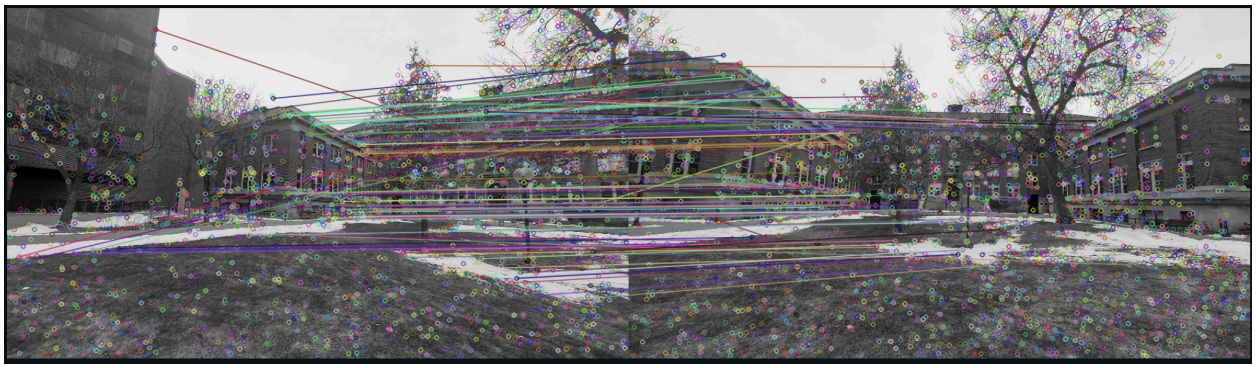
ii)

Brute force based :

```
def brute_feature_matching(img1, img2):

    image1 = cv2.imread(os.path.join(img_root,img1), cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(os.path.join(img_root,img2), cv2.IMREAD_GRAYSCALE)
```

```
    save_pth = "brute_feature_matching"
    sift = cv2.SIFT_create()

    # Detect keypoints and compute descriptors for both images
    keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
    keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

    # Initialize the feature matcher using brute-force matching
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

    # Match the descriptors using brute-force matching
    matches_bf = bf.match(descriptors1, descriptors2)

    # Sort the matches by distance (lower is better)
    matches_bf = sorted(matches_bf, key=lambda x: x.distance)

    # Draw the top N matches
    num_matches = 100
    image_matches_bf = cv2.drawMatches(image1, keypoints1, image2,
keypoints2, matches_bf[:num_matches], None)

    src_points = np.float32([keypoints1[m.queryIdx].pt for m in
matches_bf]).reshape(-1, 1, 2)
    dst_points = np.float32([keypoints2[m.trainIdx].pt for m in
matches_bf]).reshape(-1, 1, 2)

    s = os.path.join(save_pth,f"brute_matching_{img1[0]}_{img2[0]}")
    print(s)

    cv2.imwrite(f"{s}.jpg", image_matches_bf)
    # np.savetxt(f"{s}_src_points.txt",src_points)
    # np.savetxt(f"{s}_dst_points.txt",dst_points)

    print(f"image saved at : {s}.jpg " )

    return src_points, dst_points

   src_points_brute = []
```

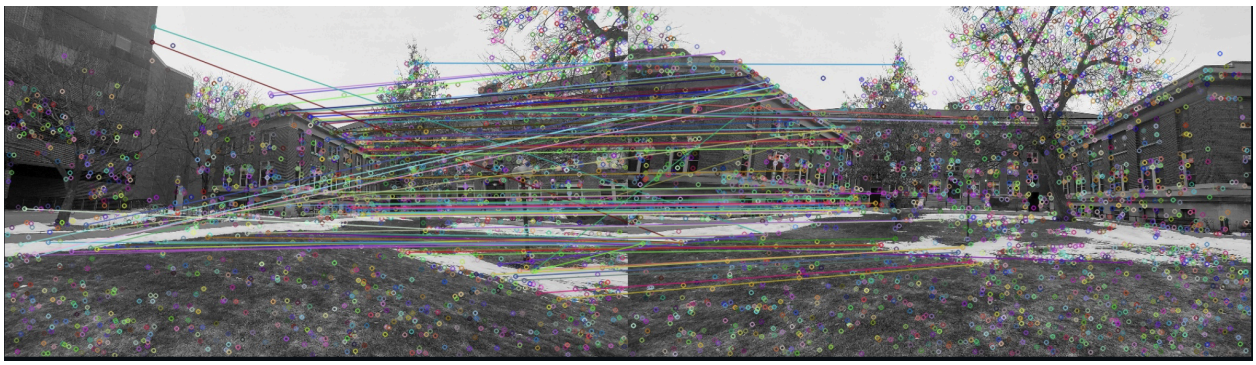
```
dst_points_brute = []

for i in range(len(img_dir)) :
    if i != len(img_dir) - 1 :
        sb , db = brute_feature_matching(img_dir[i],img_dir[i+1])
        src_points_brute.append(sb)
        dst_points_brute.append(db)
```



Flann based :

```
def flann_matching(img1, img2) :

    image1 = cv2.imread(os.path.join(img_root,img1), cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(os.path.join(img_root,img2), cv2.IMREAD_GRAYSCALE)

    num_matches = 100
    save_pth = "flann_feature_matching"
    sift = cv2.SIFT_create()

    # Detect keypoints and compute descriptors for both images
    keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
    keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

    index_params = dict(algorithm=0, trees=5)
    search_params = dict(checks=100)
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    # Match the descriptors using FLANN matching
    matches_flann = flann.match(descriptors1, descriptors2)

    # Sort the matches by distance (lower is better)
```

```
    matches_flann = sorted(matches_flann, key=lambda x: x.distance)

    # Draw the top N matches
    image_matches_flann = cv2.drawMatches(image1, keypoints1, image2,
keypoints2, matches_flann[:num_matches], None)

    src_points = np.float32([keypoints1[m.queryIdx].pt for m in
matches_flann]).reshape(-1, 1, 2)
    dst_points = np.float32([keypoints2[m.trainIdx].pt for m in
matches_flann]).reshape(-1, 1, 2)

    s = os.path.join(save_pth,f"flann_matching_{img1[0]}_{img2[0]}")
    print(s)

    cv2.imwrite(f"{s}.jpg", image_matches_flann)
    # np.savetxt(f"{s}_src_points.txt",src_points)
    # np.savetxt(f"{s}_dst_points.txt",dst_points)

    print(f"image saved at : {s}.jpg " )

    return src_points, dst_points
src_points_flann = []
dst_points_flann = []

for i in range(len(img_dir)) :
    if i != len(img_dir) - 1 :
        sf , df = flann_matching(img_dir[i],img_dir[i+1])
        src_points_flann.append(sf)
        dst_points_flann.append(df)
```



iii )

Homography matrix for img1 and img2 :

Brute force matching

```
homography, mask = cv2.findHomography(src_points_brute[0], dst_points_brute[0], cv2.RANSAC, 5.0)

# Print the estimated homography matrix
print("Estimated Homography Matrix:")
print(homography)
```

✓ 0.0s

```
Estimated Homography Matrix:
[[ 5.28050204e+02 -3.12733007e+01 -1.85290951e+05]
 [ 1.59084913e+02  3.21052436e+02 -9.72796794e+04]
 [ 5.46025613e-01 -1.35011049e-02  1.00000000e+00]]
```

```
homography, mask = cv2.findHomography(src_points_flann[0], dst_points_flann[0], cv2.RANSAC, 5.0)

# Print the estimated homography matrix
print("Estimated Homography Matrix:")
print(homography)
```

✓ 0.0s

```
Estimated Homography Matrix:
[[ 9.85070865e+01 -4.26449231e+00 -3.46094591e+04]
 [ 2.95305455e+01  6.14283815e+01 -1.81893553e+04]
 [ 1.00837617e-01  4.77673022e-04  1.00000000e+00]]
```

iv and v )

```
def wrapping(img1,img2,src_flann,dst_flann) :
    image1 = cv2.imread(os.path.join(img_root,img1))
    image2 = cv2.imread(os.path.join(img_root,img2))
    result_width = image1.shape[1] + image2.shape[1]
    result_height = image1.shape[0]

    homography, _ = cv2.findHomography(dst_flann, src_flann, cv2.RANSAC,
5.0)



    warped_image2 = cv2.warpPerspective(image2, homography, (result_width,
result_height))

    # Resize the warped image to match the width of the second image
    warped_image2_resized_rs = cv2.resize(warped_image2, (image1.shape[1],
image1.shape[0]))
```

```
    warped_image2_resized = warped_image2

    # Create the resulting canvas
    result = np.zeros((result_height, result_width, 3), dtype=np.uint8)

    # Copy images onto the resulting canvas
    result[:, :image1.shape[1]] = image1

    res_prin = copy.deepcopy(result)

    res_prin[:, image1.shape[1]:] = warped_image2_resized_rs

    warped_image2_resized[:image1.shape[0] , :image1.shape[1]] = 0

    # Blending the warped image with the second image using alpha blending
    alpha = 0.5  # blending factor
    blended_image = cv2.addWeighted(warped_image2_resized, alpha, result,
1-alpha, 0)

    blended_save = "blended_img"
    result_save = "separated_wraps"

    bs = os.path.join(blended_save,f"{img1[0]}_{img2[0]}_blend.jpg")
    rs = os.path.join(result_save,f"{img1[0]}_{img2[0]}_sep.jpg")

    cv2.imwrite(bs, blended_image)
    cv2.imwrite(rs, res_prin)

    cv2.imshow('Blended', blended_image)
    print(f"Blended imaged saved at : {bs} ")
    # cv2.imshow('Wraps', res_prin)
    print(f"Separated wraps saved at : {rs} ")

    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return


for i in range(len(img_dir)) :
```

```
    if i != len(img_dir) - 1 :
        wrapping(img_dir[i], img_dir[i+1], src_points_flann[i],
dst_points_flann[i])
```

For image 1 and 2 :
Blended img -

Separated Wraps -

vi )
Stitched image (all)