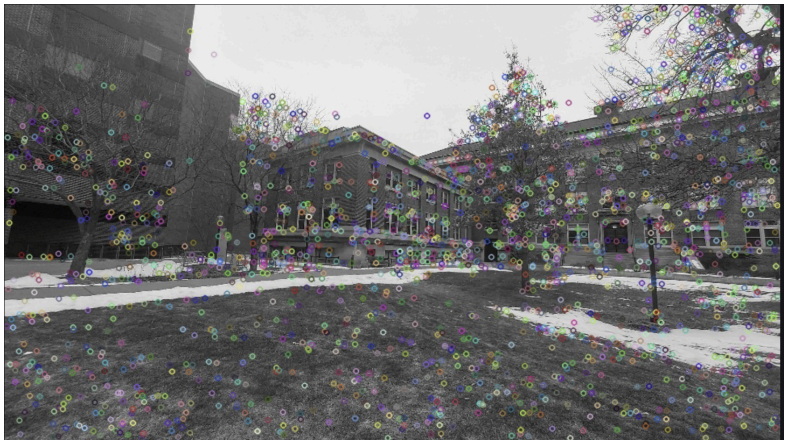
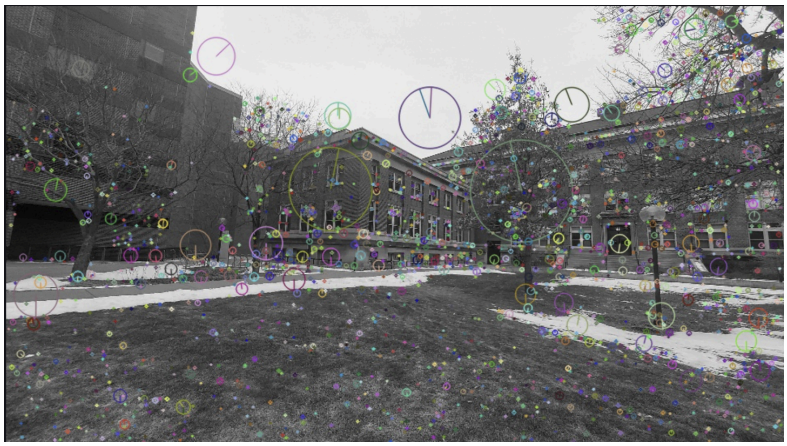
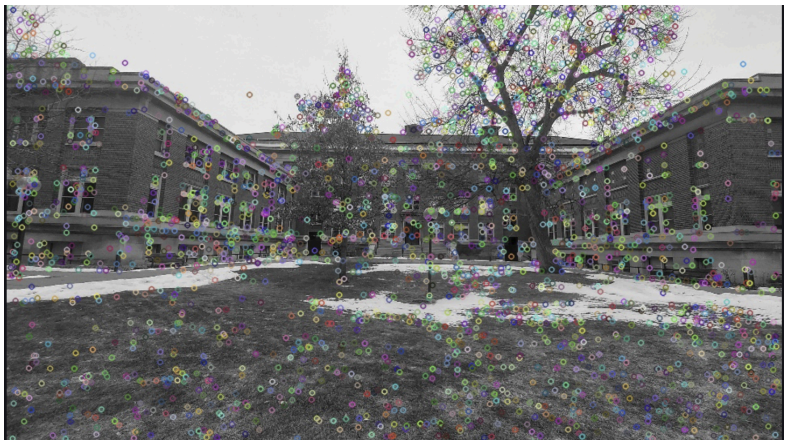
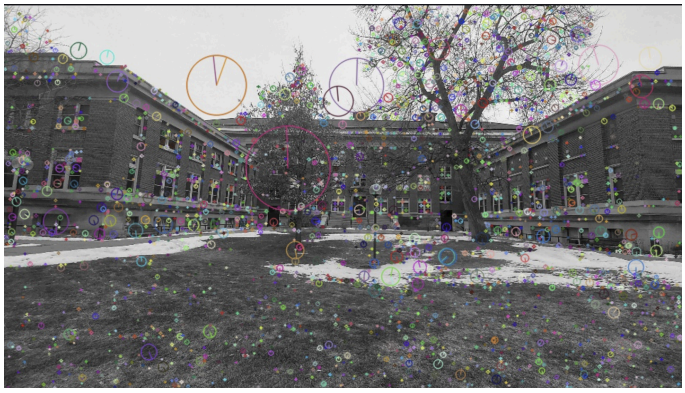
q2)
i)

```
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)
```









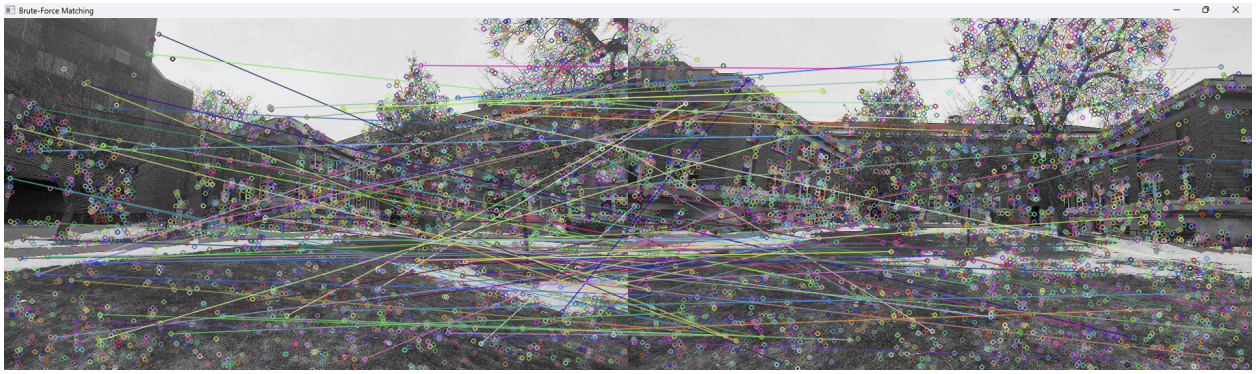
Keypoints for all images were stored

ii)

```
# Load the images
image1 = cv2.imread(os.path.join(sift_dir,'1.jpg'), cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread(os.path.join(sift_dir,'2.jpg'), cv2.IMREAD_GRAYSCALE)

# Initialize the feature detector and extractor (e.g., SIFT)
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)

# Initialize the feature matcher using FLANN matching
index_params = dict(algorithm=0, trees=5)
```

```
search_params = dict(checks=50)
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)

# Display the images with matches
cv2.imshow('Brute-Force Matching', image_matches_bf)
cv2.imshow('FLANN Matching', image_matches_flann)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Brute force matching :



Flann Based matching :

iii)

```
# Load the images
image1 = cv2.imread(os.path.join(sift_dir,'1.jpg'))
image2 = cv2.imread(os.path.join(sift_dir,'2.jpg'))

# Convert the images to grayscale
gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

# Initialize the feature detector and extractor (e.g., SIFT)
sift = cv2.SIFT_create()

# Detect keypoints and compute descriptors for both images
keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
keypoints2, descriptors2 = sift.detectAndCompute(gray2, 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.match(descriptors1, descriptors2)

# Extract the matched keypoints
src_points = np.float32([keypoints1[m.queryIdx].pt for m in
matches]).reshape(-1, 1, 2)
dst_points = np.float32([keypoints2[m.trainIdx].pt for m in
matches]).reshape(-1, 1, 2)

# Estimate the homography matrix using RANSAC
homography, mask = cv2.findHomography(src_points, dst_points, cv2.RANSAC,
5.0)

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

```
# Initialize the feature matcher using FLANN matching
index_params = dict(algorithm=0, trees=5)
search_params = dict(checks=50)
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 = sorted(matches_flann, key=lambda x: x.distance)

# Extract matching keypoints
src_points = np.float32([keypoints1[match.queryIdx].pt for match in
matches]).reshape(-1, 1, 2)
dst_points = np.float32([keypoints2[match.trainIdx].pt for match in
matches]).reshape(-1, 1, 2)

# Estimate the homography matrix
homography, _ = cv2.findHomography(src_points, dst_points, cv2.RANSAC,
5.0)
print("Estimated Homography Matrix:")
print(homography)
```

Matrix calculated for img1 and img2 (brute force matching):

```
Estimated Homography Matrix:
[[ 2.09497410e+02 -1.24818230e+01 -7.35850630e+04]
 [ 6.28724990e+01  1.27780663e+02 -3.85195947e+04]
 [ 2.15587391e-01 -4.63788961e-03  1.00000000e+00]]
```

Matrix calculated for img1 and img2 (flann based matching):

```
Estimated Homography Matrix:
[[-1.64778037e+02  1.47991059e+01  5.65460009e+04]
 [-5.00828425e+01 -9.74795359e+01  3.01338496e+04]
 [-1.71955484e-01  6.91711855e-03  1.00000000e+00]]
```

iv and v)

```
# Warp the first image using the homography
result_width = image1.shape[1] + image2.shape[1]
```

```
result_height = image1.shape[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 = cv2.resize(warped_image2, (image1.shape[1],
image1.shape[0]))

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

# Copy images onto the resulting canvas
result[:image1.shape[0], :image1.shape[1]] = image1
result[:image1.shape[0], image1.shape[1]:] = warped_image2_resized


# Blending the warped image with the second image using alpha blending
alpha = 0.5  # blending factor
blended_image = cv2.addWeighted(warped_image2_resized, alpha, image1, 1 -
alpha, 0)

# Display the blended image
cv2.imshow('Blended Image', blended_image)
cv2.imshow('Img 1', image1)
cv2.imshow('Img 2', image2)
cv2.imshow('Result', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Img 1 :

Img 2 :

Side by side :

Blended :

vi)
Final stitched image :


Stitched Image