Experiment: 2.1

**Aim:** Write a program to evaluate the efficacy of human-guided control point selection for image alignment.

**Software Required:** Any Python IDE (e.g.: PyCharm, Jupyter Notebook, GoogleCollab)

**Technique used:**

Image registration is a digital image processing technique that helps us align different images of the same scene. For instance, one may click the picture of a book from various angles.

Human-guided control point selection is a technique used in image alignment tasks where a human operator manually selects a set of control points to guide the alignment process. These control points are typically landmarks or distinctive features in the images that can be easily identified by a human observer. The overview of the process:

* Image Pair Selection
* Control Point Selection
* Control Point Matching
* Transformation Estimation
* Alignment and Image Warping
* Evaluation and Refinement

**Steps**:

1. Import necessary libraries and modules, including OpenCV, NumPy and os.
2. Load the target image (to be aligned) and the reference image
3. Convert the loaded images to grayscale, as image alignment is commonly performed on grayscale images.
4. Create an ORB (Oriented FAST and Rotated BRIEF) detector and find keypoints and descriptors for both images using the ORB detector.
5. Create a Brute Force matcher with Hamming distance as the measurement mode and match the descriptors of the two images.
6. Take the top 90% of matches based on the sorted list.
7. Define empty matrices to store the coordinates of keypoints that have been successfully matched between the two images. Populate these matrices with the matching keypoint coordinates.
8. Find the homography matrix using the RANSAC algorithm, which estimates the transformation between the two images.
9. Use the homography matrix to transform the target image to align it with the reference image. The transformed image is stored in the transformed\_img variable
10. evaluateAlignment function to calculate the Mean Square Error (MSE) between the aligned image and the reference image. MSE is used as an alignment quality metric.
11. outputResults function to display the alignment quality (MSE and RMSE) and the efficacy of the alignment.
12. Calculate the MSE and RMSE for the alignment and store them in the alignmentQuality variable.
13. Calculate the efficacy of human-guided control point selection and store it in the efficacy variable.
14. Save the aligned image as 'exp5\_output.jpg'. And display the original reference image and the transformed image using cv2\_imshow.

**Implementation**:

# Importing Libraries

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

# Open the image files.

target\_image = cv2.imread('/content/drive/MyDrive/Colab Notebooks/Images/dogs/align.jpg') # Image to be aligned.

original\_image = cv2.imread('/content/drive/MyDrive/Colab Notebooks/Images/dogs/target.jpg') # Reference image.

# Convert to grayscale.

img1 = cv2.cvtColor(target\_image, cv2.COLOR\_BGR2GRAY)

img2 = cv2.cvtColor(original\_image, cv2.COLOR\_BGR2GRAY)

height, width = img2.shape

# Create ORB detector with 5000 features.

orb\_detector = cv2.ORB\_create(5000)

# Find keypoints and descriptors.

# The first arg is the image, second arg is the mask

# (which is not required in this case).

kp1, d1 = orb\_detector.detectAndCompute(img1, None)

kp2, d2 = orb\_detector.detectAndCompute(img2, None)

# Match features between the two images.

# We create a Brute Force matcher with

# Hamming distance as measurement mode.

matcher = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck = True)

# Match the two sets of descriptors.

matches = matcher.match(d1, d2)

# Sort matches on the basis of their Hamming distance.

list(matches).sort(key = lambda x: x.distance)

# Take the top 90 % matches forward.

matches = matches[:int(len(matches)\*0.9)]

no\_of\_matches = len(matches)

# Define empty matrices of shape no\_of\_matches \* 2.

p1 = np.zeros((no\_of\_matches, 2))

p2 = np.zeros((no\_of\_matches, 2))

# coordinates of keypoints that have been successfully matched between the two images

for i in range(len(matches)):

  p1[i, :] = kp1[matches[i].queryIdx].pt

  p2[i, :] = kp2[matches[i].trainIdx].pt

# Find the homography matrix.

homography, mask = cv2.findHomography(p1, p2, cv2.RANSAC)

# Use this matrix to transform the

# colored image wrt the reference image.

transformed\_img = cv2.warpPerspective(target\_image,

          homography, (width, height))

#alignment quality using Mean Square Error (MSE)

def evaluateAlignment(aligned\_image, reference\_image):

    # Calculate the Mean Square Error (MSE) between the aligned image and the reference image

    mse = np.mean((aligned\_image - reference\_image) \*\* 2)

    return mse

# calculate the efficacy based on the alignment quality

def calculateEfficacy(alignmentQuality):

    # In this example, we assume a lower MSE (Mean Square Error) indicates better alignment.

    threshold = 100.0  # Adjust this threshold as needed.

    if alignmentQuality < threshold:

        efficacy = "High"

    else:

        efficacy = "Low"

    return efficacy

# output evaluation results and efficacy metrics

def outputResults(alignmentQuality, efficacy):

    print(f"Alignment Quality (MSE): {alignmentQuality}")

    print(f"Alignment Quality (RMSE): {np.sqrt(alignmentQuality)}")

    print(f"Efficacy: {efficacy}")

# Evaluate alignment quality

alignmentQuality = evaluateAlignment(transformed\_img, original\_image)

# Calculate efficacy of human-guided control point selection

efficacy = calculateEfficacy(alignmentQuality)

# Output evaluation results and efficacy metrics

outputResults(alignmentQuality, efficacy)

# Save the output.

cv2.imwrite('exp5\_output.jpg', transformed\_img)

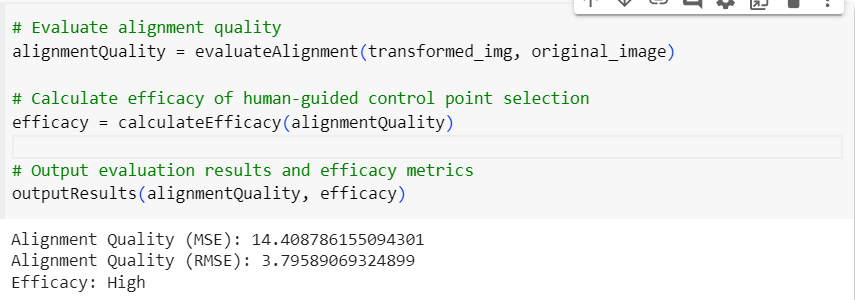
print("original image : ")

cv2\_imshow(original\_image);

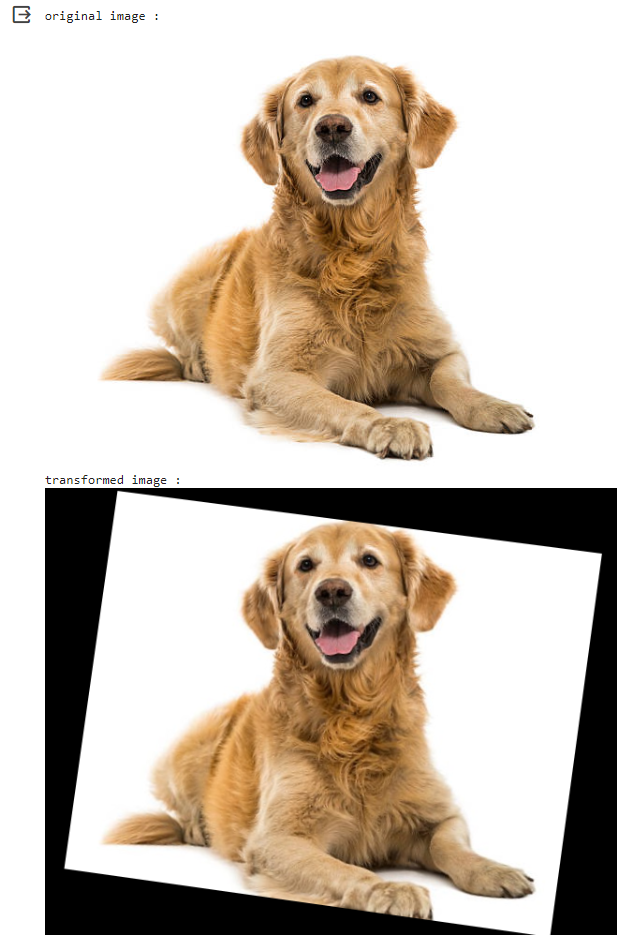
print("transformed image : ")

cv2\_imshow(transformed\_img)

**Output screenshot**:



**Output Evaluation**



**Comparing Original and Transformed Image**