Experiment 3.3

Aim: Write a program to interpret the effectiveness of template matching techniques for video stabilization tasks.

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

Relevance of the Experiment: This experiment aims to evaluate the effectiveness of template matching techniques for video stabilization tasks. Video stabilization is crucial in various applications, such as video surveillance, autonomous navigation, and video production. By understanding and implementing template matching algorithms, we can analyze their performance in stabilizing videos, which can lead to improved video quality and enhanced user experience.

Description:

Template matching techniques can be used for video stabilization tasks to estimate and compensate for camera motion in a sequence of frames. Here's how template matching can be applied in video stabilization:

- Template Selection: A template is selected from an initial frame in the video sequence. The template represents a distinct and stable feature or region in the scene that will be used as a reference for stabilization. This can be, for example, a prominent object or a textured region with high contrast.
- Template Tracking: The selected template is then tracked across subsequent frames using template matching techniques. Template matching involves comparing the template with regions in each frame to find the best match based on a similarity measure. Common similarity measures include sum of squared differences (SSD) and normalized cross-correlation (NCC).
- Motion Estimation: The template matching process provides an estimate of the motion between frames by determining the displacement that results in the best match. This motion estimation represents the camera motion or global motion in the scene.
- Stabilization Transformation: Based on the estimated motion, a stabilization transformation is applied to the frames. This transformation compensates for the camera motion and aligns the frames to stabilize the video. The transformation can be a translation, rotation, or affine transformation, depending on the nature of the motion.
- Warping and Interpolation: The stabilized frames are warped according to the stabilization transformation to remove the effects of camera motion. This involves resampling the pixels in the frames to create a stabilized output. Interpolation

techniques, such as bilinear or bicubic interpolation, can be used to fill in the gaps and generate smooth transitions between the frames.

- Fine-tuning and Refinement: Template matching may have limitations in handling large or complex camera motions or occlusions. Therefore, additional techniques like feature-based tracking or optical flow can be integrated to refine the stabilization results and handle challenging scenarios.

Steps:

- Read the input video and select a template region.
- Extract frames from the video and convert them to grayscale.
- Apply template matching between the template region and each frame to find the best match.
- Calculate the motion vectors between consecutive frames based on the template matching results.
- Estimate the global motion from the motion vectors to stabilize the video.
- Apply the estimated motion to the frames to obtain the stabilized video.
- Save the stabilized video to an output file.

Code:

```
import cv2
import numpy as np
# Function to stabilize a video using template matching
def stabilize_video(input_video_path, output_video_path, template_path, threshold=0.8):
  cap = cv2.VideoCapture(input_video_path)
  if not cap.isOpened():
    print("Error: Could not open video.")
    return
  template = cv2.imread(template_path, cv2.IMREAD_GRAYSCALE) # Load the template as grayscale
  if template is None:
    print("Error: Could not open template image.")
    return
    # Resize the template if it's larger than the video frames
    if template.shape[0]> cap.get(4) or template.shape[1]> cap.get(3):
      template = cv2.resize(template, (int(cap.get(3)), int(cap.get(4))))
    fource = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter(output_video_path, fourcc, 30.0, (int(cap.get(3)), int(cap.get(4))))
    prev_frame = None
```

```
    while True:
      ret, frame = cap.read()
      if not ret:
        break
      if prev_frame is None:
        prev_frame = frame
        out.write(frame)
        continue
      # Convert frames to grayscale
      prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
      current_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      # Perform template matching
      result = cv2.matchTemplate(current_gray, template, cv2.TM_CCOEFF_NORMED)
      min_val, max_val, min_loc, max_loc= cv2.minMaxLoc(result)
      if max_val>= threshold:
        # Calculate the translation vector
        x_offset = max_loc[0] - template.shape[1] // 2
        y_offset = max_loc[1] - template.shape[0] //2
        # Apply the translation to stabilize the frame
        translation_matrix = np.float32([[1, 0, x_offset], [0, 1, y_offset]])
        stabilized_frame = cv2.warpAffine(frame, translation_matrix, (frame.shape[1],
frame.shape[0]))
        out.write(stabilized_frame)
      else:
        out.write(frame)
      prev_frame = frame
    cap.release()
    out.release()
    cv2.destroyAllWindows()
    print("Video stabilization complete.")
  input_video_path = "/content/WhatsApp Video 2023-11-03 at 11.55.53_530800a0.mp4"
  output_video_path = "/content/Sound effect - Directed by Robert B. Weide.mp4"
  template_path = "/content/Screenshot (244).png"
  threshold = 0.8
  stabilize_video(input_video_path, output_video_path, template_path, threshold)
```

Implementation:

```
            break

        if prev_frame is None:
            prev_frame = frame
            out.write(frame)
            continue

        # Convert frames to grayscale
        prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
        current_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Perform template matching
        result = cv2.matchTemplate(current_gray, template, cv2.TM_CCOEFF_NORMED)
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

        if max_val >= threshold:
            # Calculate the translation vector
            x_offset = max_loc[0] - template.shape[1] // 2
            y_offset = max_loc[1] - template.shape[0] // 2

            # Apply the translation to stabilize the frame
            translation_matrix = np.float32([[1, 0, x_offset], [0, 1, y_offset]])
            stabilized_frame = cv2.warpAffine(frame, translation_matrix, (frame.shape[1], frame.shape[0]))

            out.write(stabilized_frame)
        else:
            out.write(frame)

        prev_frame = frame

    cap.release()
    out.release()
    cv2.destroyAllWindows()
    print("Video stabilization complete.")

input_video_path = "videos/Doors-We-Walk-Through-5-Second-Videos.mp4"
output_video_path = "videos/output_video.mp4"
template_path = "images/Vivek.png"
threshold = 0.8

stabilize_video(input_video_path, output_video_path, template_path, threshold)
```

```
Video stabilization complete.
```