CourseName: Computer Vision Lab **Course Code:** CSP-422

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)

Description:

Template matching is a technique employed in video stabilization to address camera motion in a sequence of frames. The process involves the following steps:

Template Selection: Choose a template from an initial frame that represents a stable and distinctive feature in the scene, such as a prominent object or a textured region with high contrast.

Template Tracking: Track the selected template across subsequent frames using template matching techniques. This involves comparing the template with regions in each frame using similarity measures like sum of squared differences (SSD) or normalized cross-correlation (NCC).

Motion Estimation: The template matching process estimates the motion between frames by determining the displacement that results in the best match. This motion estimation reflects the camera or global motion in the scene.

Stabilization Transformation: Apply a stabilization transformation to the frames based on the estimated motion. This compensates for camera motion and aligns the frames for stabilization. The transformation may include translation, rotation, or affine transformation, depending on the motion's nature.

Warping and Interpolation: Warp the stabilized frames according to the stabilization transformation, removing the effects of camera motion. Resample the pixels in the frames using interpolation techniques like bilinear or bicubic interpolation to ensure smooth transitions.

Fine-tuning and Refinement: Address potential limitations of template matching, especially for handling large or complex camera motions or occlusions. Integrate additional techniques like feature-based tracking or optical flow to refine stabilization results and manage challenging scenarios.

Steps:

1. Load the source video and designate a specific template area.
2. Capture individual frames from the video and transform them into grayscale.

3. Utilize template matching to compare the template region with each frame, identifying the optimal match.
4. Compute motion vectors by analyzing the template matching outcomes between successive frames.
5. Derive the overall motion by analyzing the motion vectors to stabilize the entire video.
6. Implement the inferred motion to the frames, resulting in a stabilized video.
7. Save the stabilized video to an output file.

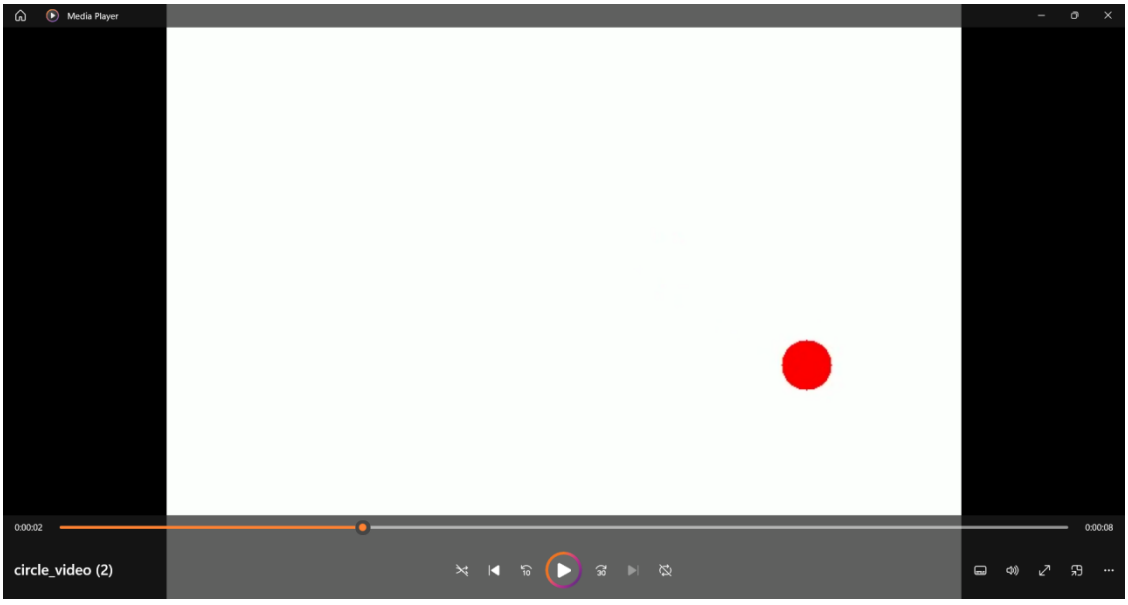
Implementation:

Code:

```
import cv2
from google.colab.patches import cv2_imshow

# Read the template image
template = cv2.imread('./template_circle.jpg', 0)  # Load template image in grayscale

# Capture the video
cap = cv2.VideoCapture('./circle_video.mp4')  # Replace 'input_video.mp4' with your
video file

# Lists to store frames with rectangles
frames_with_rectangles = []

while cap.isOpened():
    ret, frame = cap.read()

    if not ret:
        break
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Perform template matching
    res = cv2.matchTemplate(frame_gray, template, cv2.TM_CCOEFF_NORMED)
    threshold = 0.8  # Set a threshold for match quality
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    # Check if the maximum value is above the threshold
    if max_val >= threshold:
        top_left = max_loc
        h, w = template.shape[:2]
        bottom_right = (top_left[0] + w, top_left[1] + h)

        # Draw a rectangle around the matched region
```

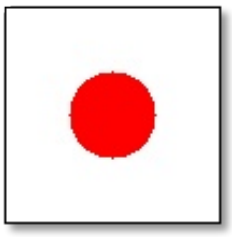
```
        frame_with_rectangle = frame.copy()  # Create a copy to draw the rectangle
        cv2.rectangle(frame_with_rectangle, top_left, bottom_right, (0, 255, 0), 2)
        frames_with_rectangles.append(frame_with_rectangle)

    if len(frames_with_rectangles) > 0:
        cv2_imshow(frames_with_rectangles[-1])  # Show the last modified frame

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()

# Define the codec and create a VideoWriter object
video_out = cv2.VideoWriter('output_video.mp4', cv2.VideoWriter_fourcc(*'mp4v'), 30,
(frame_width, frame_height))
# Write the frames with rectangles into the output video
for frame in frames_with_rectangles:
    video_out.write(frame)

video_out.release()
cv2.destroyAllWindows()
```

Normal Video:

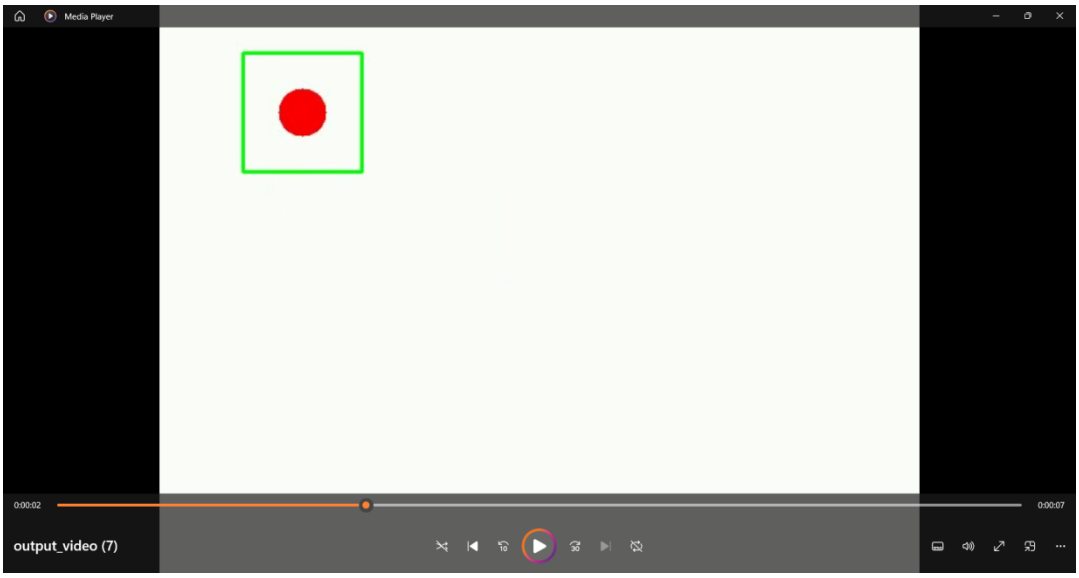


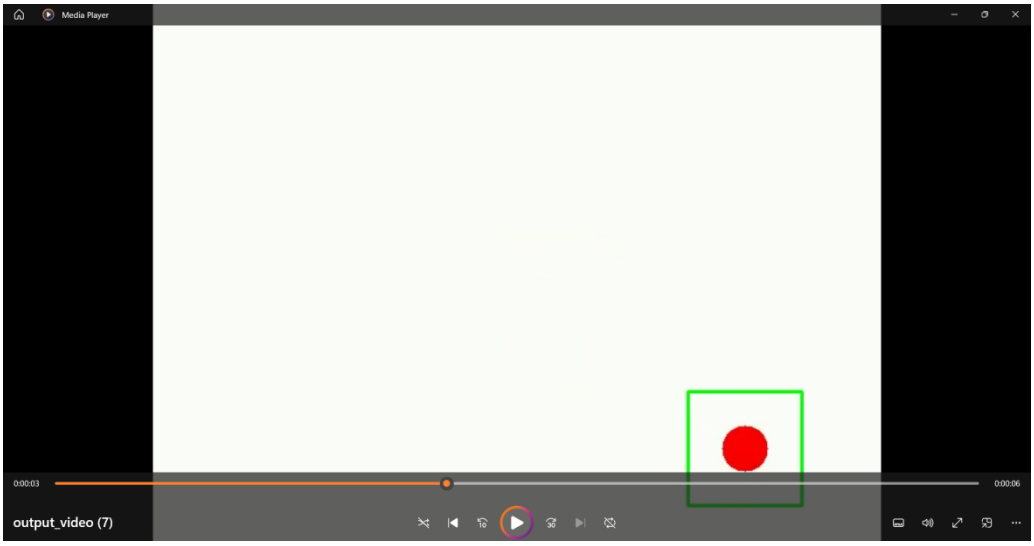
CourseName: Computer Vision Lab **Course Code:** CSP-422

Template Image

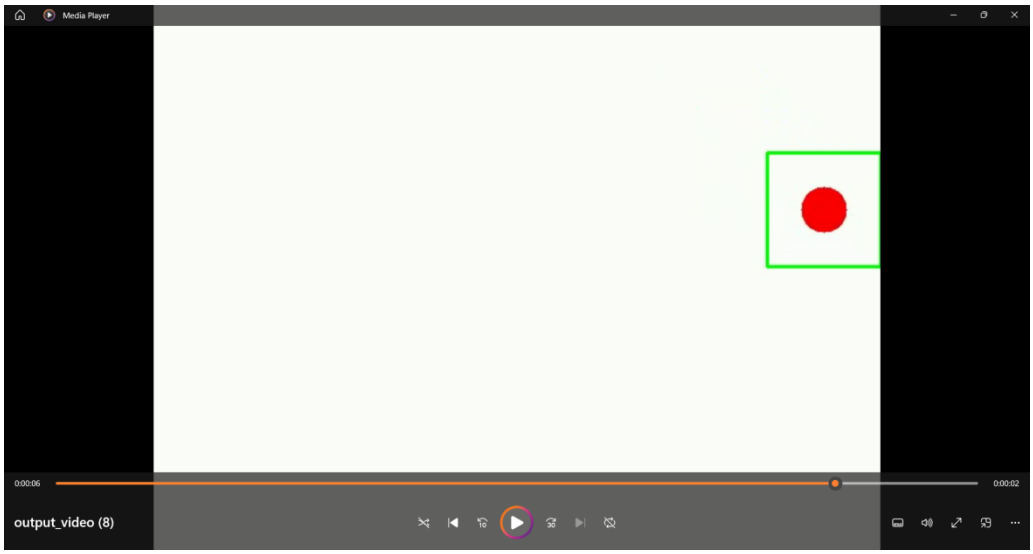


Output Video:





CourseName: Computer Vision Lab **Course Code:** CSP-422



Name: Abhinav Verma **UID:** 20BCS9258