Computer Vision

Assignment 5

Keio University

STEFAN Vlad 82123434
23/05/2022

1 Video stabilization

The video stabilization code works as follows:

1. The geometrical transformation between successive frames is obtained.
2. The image trajectory is computed by successively summing the transformation parameters.
3. The trajectory is smoothed.
4. The smooth trajectory is used to modify the previous transformations to ensure that the successive images remain as close as possible to the smooth trajectory.
5. These new transformations are used to modify the video frames.
6. A slight zoom is applied to the video to avoid missing edge pixels.

2 Code

The program begins by importing the necessary libraries. It also retrieves some information about the current video, such as the number of frames and frame rates, and sets up the output video.

```
# Import numpy and OpenCV
import numpy as np
import cv2

# Read input video
cap = cv2.VideoCapture('video.mp4')

# Get frame count and frame rate
n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Get width and height of video stream
w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Define the codec for output video
fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')

# Set up output video
out = cv2.VideoWriter('video_out.mp4', fourcc, fps, (2 * w, h))
```

The algorithm iterates over groups of 2 successive frames. For each frame pair, it finds features in the first frame and then tracks them in the second one using the iterative Lucas-Kanade method. Knowing the displacement of these features, the affine transformation that explain such a displacement is estimated. Then, the translation parameters and rotation angle are retrieved and stored.

```
# Read first frame
_, prev = cap.read()

# Convert frame to grayscale
prev_gray = cv2.cvtColor(prev, cv2.COLOR_BGR2GRAY)

# Pre-define transformation-store array
transforms = np.zeros((n_frames - 1, 3), np.float32)

for i in range(n_frames - 2):
    # Detect feature points in previous frame
    prev_pts = cv2.goodFeaturesToTrack(prev_gray,
```

```
                                        maxCorners=200,
                                        qualityLevel=0.01,
                                        minDistance=30,
                                        blockSize=3)

    # Read next frame
    success, curr = cap.read()
    if not success:
        break

    # Convert to grayscale
    curr_gray = cv2.cvtColor(curr, cv2.COLOR_BGR2GRAY)

    # Calculate optical flow (i.e. track feature points)
    curr_pts, status, err = cv2.calcOpticalFlowPyrLK(prev_gray, curr_gray, prev_pts, None)

    # Sanity check
    assert prev_pts.shape == curr_pts.shape

    # Filter only valid points
    idx = np.where(status == 1)[0]
    prev_pts = prev_pts[idx]
    curr_pts = curr_pts[idx]

    # Find transformation matrix
    m, _ = cv2.estimateAffinePartial2D(prev_pts, curr_pts)

    # Extract translation
    dx = m[0, 2]
    dy = m[1, 2]

    # Extract rotation angle
    da = np.arctan2(m[1, 0], m[0, 0])

    # Store transformation
    transforms[i] = [dx, dy, da]

    # Move to next frame
    prev_gray = curr_gray
```

The image trajectory is computed by cumulatively summing transformation parameters.

```
# Compute trajectory using cumulative sum of transformations
trajectory = np.cumsum(transforms, axis=0)
```

The obtained trajectory is shown in figure 1

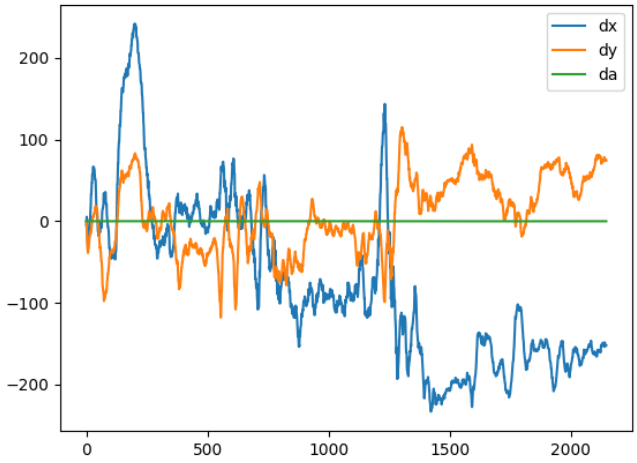

Figure 1: plot showing the cumulative sum of the translation in the x-axis, the translation in the y-axis, and the rotation angle

To smooth the previously computed curve, two functions are created. The first one applies an averaging filter on a 1D array, and the second one applies the first function to the 3 elements that constitute the image trajectory.

```
def moving_average(curve, radius):
    window_size = 2 * radius + 1
    # Define the filter
    f = np.ones(window_size) / window_size
    # Add padding to the boundaries
    curve_pad = np.lib.pad(curve, (radius, radius), 'edge')
    # Apply convolution
    curve_smoothed = np.convolve(curve_pad, f, mode='same')
    # Remove padding
    curve_smoothed = curve_smoothed[radius:-radius]
    # return smoothed curve
    return curve_smoothed


def smooth(path, radius):
    smoothed_trajectory = np.copy(path)
    # Filter the x, y and angle curves
    for j in range(3):
        smoothed_trajectory[:, j] = moving_average(path[:, j], radius=radius)

    return smoothed_trajectory
```

The smoothed trajectory is shown in figure 2

Figure 2: plot showing the smoothed cumulative sum of the translation in the x-axis, the translation in the y-axis, and the rotation angle

With the previous functions, the trajectory is smoothed. The difference between the smooth trajectory and the original one is used to adapt the transformation parameters. With these new transformation parameters, the video movement will more closely resemble the smooth trajectory.

```
# Smooth the trajectory
smooth_trajectory = smooth(trajectory, 30)

# Calculate difference in smoothed_trajectory and trajectory
difference = smooth_trajectory - trajectory

# Calculate newer transformation array
transforms_smooth = transforms + difference
```

To avoid missing edge pixels, a function is created to zoom on each frame by a specified amount.

```
def fix_border(image, scale):
    s = image.shape
    # Scale the image without moving the center
    T = cv2.getRotationMatrix2D((s[1] / 2, s[0] / 2), 0, scale)
    image = cv2.warpAffine(image, T, (s[1], s[0]))
    return image
```

The new transformation parameters are used to modify each frame. The border artifacts are corrected by applying the previous function on each of the video frames. In the end, the new frames are written to make a new video.

```
# Reset stream to first frame
cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

# Write n_frames-1 transformed frames
for i in range(n_frames - 2):
    # Read next frame
    success, frame = cap.read()
    if not success:
        break

    # Extract transformations from the new transformation array
    dx = transforms_smooth[i, 0]
    dy = transforms_smooth[i, 1]
```

```
    da = transforms_smooth[i, 2]

    # Reconstruct transformation matrix accordingly to new values
    m = np.zeros((2, 3), np.float32)
    m[0, 0] = np.cos(da)
    m[0, 1] = -np.sin(da)
    m[1, 0] = np.sin(da)
    m[1, 1] = np.cos(da)
    m[0, 2] = dx
    m[1, 2] = dy

    # Apply affine wrapping to the given frame
    frame_stabilized = cv2.warpAffine(frame, m, (w, h))

    # Write the original and stabilized frames side by side
    frame_out = cv2.hconcat([frame, frame_stabilized])

    # Fix border artifacts
    frame_stabilized = fix_border(frame_stabilized, 1.2)

    # Write the frame to the file
    out.write(frame_out)

cap.release()
out.release()
```

The previously described code has been adapted from [1][1]. The test videos can be found in this drive.

References

[1] Abhishek Singh Thakur. *Video Stabilization Using Point Feature Matching in OpenCV*. URL: https://learnopencv.com/video-stabilization-using-point-feature-matching-in-opencv/ (visited on 05/16/2022).

[1]The code is available at this link