

COMPUTER VISION

Assignment 5

Keio University



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.

```
1 # Import numpy and OpenCV
2 import numpy as np
3 import cv2
4
5 # Read input video
6 cap = cv2.VideoCapture('video.mp4')
7
8 # Get frame count and frame rate
9 n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
10 fps = int(cap.get(cv2.CAP_PROP_FPS))
11
12 # Get width and height of video stream
13 w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
14 h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
15
16 # Define the codec for output video
17 fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
18
19 # Set up output video
20 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.

```
1 # Read first frame
2 _, prev = cap.read()
3
4 # Convert frame to grayscale
5 prev_gray = cv2.cvtColor(prev, cv2.COLOR_BGR2GRAY)
6
7 # Pre-define transformation-store array
8 transforms = np.zeros((n_frames - 1, 3), np.float32)
9
10 for i in range(n_frames - 2):
11     # Detect feature points in previous frame
12     prev_pts = cv2.goodFeaturesToTrack(prev_gray,
```

```

13         maxCorners=200,
14         qualityLevel=0.01,
15         minDistance=30,
16         blockSize=3)
17
18     # Read next frame
19     success, curr = cap.read()
20     if not success:
21         break
22
23     # Convert to grayscale
24     curr_gray = cv2.cvtColor(curr, cv2.COLOR_BGR2GRAY)
25
26     # Calculate optical flow (i.e. track feature points)
27     curr_pts, status, err = cv2.calcOpticalFlowPyrLK(prev_gray, curr_gray, prev_pts, None)
28
29     # Sanity check
30     assert prev_pts.shape == curr_pts.shape
31
32     # Filter only valid points
33     idx = np.where(status == 1)[0]
34     prev_pts = prev_pts[idx]
35     curr_pts = curr_pts[idx]
36
37     # Find transformation matrix
38     m, _ = cv2.estimateAffinePartial2D(prev_pts, curr_pts)
39
40     # Extract translation
41     dx = m[0, 2]
42     dy = m[1, 2]
43
44     # Extract rotation angle
45     da = np.arctan2(m[1, 0], m[0, 0])
46
47     # Store transformation
48     transforms[i] = [dx, dy, da]
49
50     # Move to next frame
51     prev_gray = curr_gray

```

The image trajectory is computed by cumulatively summing transformation parameters.

```

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

```

The obtained trajectory is shown in figure 1

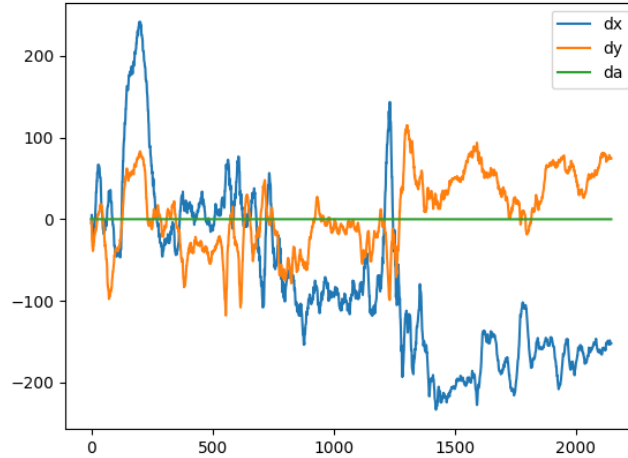


Figure 1: plot showing the cumulative sum of the translation in the x-axis, the translation in the y-axis, and the rotation angle

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.

```

1 def moving_average(curve, radius):
2     window_size = 2 * radius + 1
3     # Define the filter
4     f = np.ones(window_size) / window_size
5     # Add padding to the boundaries
6     curve_pad = np.lib.pad(curve, (radius, radius), 'edge')
7     # Apply convolution
8     curve_smoothed = np.convolve(curve_pad, f, mode='same')
9     # Remove padding
10    curve_smoothed = curve_smoothed[radius:-radius]
11    # return smoothed curve
12    return curve_smoothed
13
14
15 def smooth(path, radius):
16     smoothed_trajectory = np.copy(path)
17     # Filter the x, y and angle curves
18     for j in range(3):
19         smoothed_trajectory[:, j] = moving_average(path[:, j], radius=radius)
20
21     return smoothed_trajectory

```

The smoothed trajectory is shown in figure 2

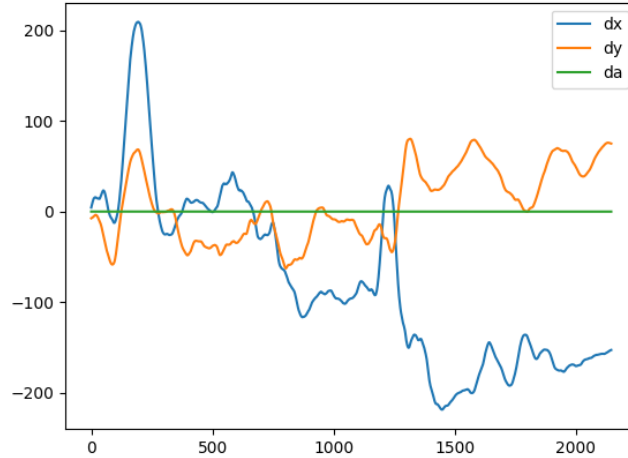


Figure 2: plot showing the smoothed cumulative sum of the **translation in the x-axis**, the **translation in the y-axis**, and the **rotation angle**

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.

```
1 # Smooth the trajectory
2 smooth_trajectory = smooth(trajectory, 30)
3
4 # Calculate difference in smoothed_trajectory and trajectory
5 difference = smooth_trajectory - trajectory
6
7 # Calculate newer transformation array
8 transforms_smooth = transforms + difference
```

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

```
1 def fix_border(image, scale):
2     s = image.shape
3     # Scale the image without moving the center
4     T = cv2.getRotationMatrix2D((s[1] / 2, s[0] / 2), 0, scale)
5     image = cv2.warpAffine(image, T, (s[1], s[0]))
6     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.

```
1 # Reset stream to first frame
2 cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
3
4 # Write n_frames-1 transformed frames
5 for i in range(n_frames - 2):
6     # Read next frame
7     success, frame = cap.read()
8     if not success:
9         break
10
11     # Extract transformations from the new transformation array
12     dx = transforms_smooth[i, 0]
13     dy = transforms_smooth[i, 1]
```

```

14     da = transforms_smooth[i, 2]
15
16     # Reconstruct transformation matrix accordingly to new values
17     m = np.zeros((2, 3), np.float32)
18     m[0, 0] = np.cos(da)
19     m[0, 1] = -np.sin(da)
20     m[1, 0] = np.sin(da)
21     m[1, 1] = np.cos(da)
22     m[0, 2] = dx
23     m[1, 2] = dy
24
25     # Apply affine wrapping to the given frame
26     frame_stabilized = cv2.warpAffine(frame, m, (w, h))
27
28     # Write the original and stabilized frames side by side
29     frame_out = cv2.hconcat([frame, frame_stabilized])
30
31     # Fix border artifacts
32     frame_stabilized = fix_border(frame_stabilized, 1.2)
33
34     # Write the frame to the file
35     out.write(frame_out)
36
37 cap.release()
38 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. *ORB (Oriented FAST and Rotated BRIEF)*. URL: [Video%20Stabilization%20Using%20Point%20Feature%20Matching%20in%20OpenCV](#) (visited on 05/16/2022).

¹The code is available at this link