Tracking-Objects

January 20, 2024

[1]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import shutil
import glob
from google.colab import drive
drive.mount('/content/drive')
root_path = '/content/drive/MyDrive/cv_assig_5/'
```

```
Mounted at /content/drive
```

[2]:

```
def affine_warp(p):
    return np.hstack([np.eye(2), np.zeros((2, 1))]) + p.reshape((2, 3),␣
 ↪order='F')
```

[3]:

```
def update_affine_parameters(coordinates, template, img, x1, x2, y1, y2, p, Ix,␣
 ↪Iy):
    """Update affine parameters using Lucas-Kanade algorithm."""
    v = np.array([[x1, x1, x2, x2], [y1, y2, y2, y1], [1, 1, 1, 1]])

    # Apply affine transformation to coordinates
    affine = affine_warp(p)
    v = np.dot(affine, v)
    new_coordinates = np.dot(affine, coordinates).astype(int)

    # Fix boundaries
    new_coordinates[new_coordinates < 0] = 0
    new_coordinates[0][new_coordinates[0] >= img.shape[1]] = img.shape[1] - 1
    new_coordinates[1][new_coordinates[1] >= img.shape[0]] = img.shape[0] - 1

    # Warp image
    img_arr = img[new_coordinates[1, :], new_coordinates[0, :]].reshape(1, -1)

    b = template - img_arr

    # Calculate gradients
    sx = Ix[new_coordinates[1, :], new_coordinates[0, :]].reshape(1, -1)
    sy = Iy[new_coordinates[1, :], new_coordinates[0, :]].reshape(1, -1)

    # Build matrix A
    A = np.vstack((sx * coordinates[0, :], sy * coordinates[0, :],
                   sx * coordinates[1, :], sy * coordinates[1, :],
                   sx, sy)).T

    # Calculate Hessian and delta_p
    Hessian = A.T @ A
    Hessian_inverse = np.linalg.pinv(Hessian)
    delta_p = Hessian_inverse @ (A.T @ b.T)

    # Calculate norm of delta p
    p_norm = np.linalg.norm(delta_p)

    # Update p
    p = p.reshape(6, 1) + delta_p

    return p, p_norm, v
```

[4]:

```
def Lucas_Kanade(img_frame, template_frame, x1, y1, w, h, p):
    """Lucas-Kanade object tracking algorithm."""
    max_iterations = 100
    threshold = 1e-4
    x2 = x1 + w
    y2 = y1 + h

    norm_frame = (img_frame * (np.mean(template_frame) / np.mean(img_frame))).
  ↪astype(float)
    template_window_coordinates = np.zeros((3, (w + 1) * (h + 1)))

    # Generate coordinates for the template window
    for y in range(y1, y2 + 1):
        for x in range(x1, x2 + 1):
            template_window_coordinates[0, (y - y1) * (x2 - x1) + (x - x1)] = x
            template_window_coordinates[1, (y - y1) * (x2 - x1) + (x - x1)] = y
            template_window_coordinates[2, (y - y1) * (x2 - x1) + (x - x1)] = 1
    template_window_coordinates = template_window_coordinates.astype(int)

    # Extract intensities at template coordinates
    template_frame_intensities = template_frame[template_window_coordinates[1, :
  ↪], template_window_coordinates[0, :]].reshape(1, -1)

    Ix = cv2.Sobel(norm_frame, cv2.CV_64F, 1, 0, ksize=3)
    Iy = cv2.Sobel(norm_frame, cv2.CV_64F, 0, 1, ksize=3)

    original_p_norm = float('inf')
    curr_iteration = 0

    while original_p_norm > threshold and curr_iteration < max_iterations:
        curr_iteration += 1
        returned_p, returned_p_norm, returned_vertex =␣
  ↪update_affine_parameters(template_window_coordinates,␣
  ↪template_frame_intensities, norm_frame, x1, x2, y1, y2, p, Ix, Iy)

        if returned_p_norm < original_p_norm:
            p = returned_p

    out = cv2.polylines(img_frame, np.int32([returned_vertex.T]), 1, 0, 2)

    return p, out
```

[5]:
```
def track_object(video, rectangle, output_video_path, frame_size, frame_rate,␣
  ↪num_frames):
    """Track object in a video and save the result."""
    fourcc = cv2.VideoWriter_fourcc('M', 'J', 'P', 'G')
    outVideo = cv2.VideoWriter(output_video_path, fourcc, frame_rate,␣
  ↪frame_size, isColor=False)

    template = video[0].T
    x, y, w, h = rectangle
    p = np.array([[0, 0, 0, 0, 0, 0]]).T

    for i in range(num_frames):
        curr_frame = video[i].T.copy()
        p, out = Lucas_Kanade(curr_frame, template, x, y, w, h, p)
        outVideo.write(out.astype(np.uint8))

    outVideo.release()
    shutil.move(output_video_path, output_video_path)
```

1 car

[6]:
```
video = np.load(root_path + 'car2.npy').T
object_path = root_path + 'car.avi'
bbox = (50, 108, 155 - 50, 162 - 108)
track_object(video, bbox, object_path, video.shape[1:], 20, video.shape[0])
print('done')
```

```
done
```

2 landing

[7]:

```
video = np.load(root_path + 'landing.npy').T
object_path = root_path + 'landing.avi'
bbox = (438, 88, 560 - 438, 130 - 88)
track_object(video, bbox, object_path, video.shape[1:], 10, video.shape[0])
print('done')
```

```
done
```