DATA.ML.300 Computer Vision Exercise 5

Uyen Phan

01 2024

1

$\mathbf{v}' = \begin{pmatrix} x_2' - x_1' \\ y_2' - y_1' \end{pmatrix}$ $\mathbf{v} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$

a) $s = \frac{\|\mathbf{v}'\|}{\|\mathbf{v}\|} = \frac{\sqrt{(x_2'-x_1')^2+(y_2'-y_1')^2}}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}$

b) $cos\theta = \frac{\mathbf{v}'\cdot\mathbf{v}}{\|\mathbf{v}'\|\|\mathbf{v}\|} = \frac{(x_2'-x_1')(x_2-x_1)+(y_2'-y_1')(y_2-y_1)}{\sqrt{(x_2'-x_1')^2+(y_2'-y_1')^2}\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}$

$\theta = \arccos(\frac{(x_2'-x_1')(x_2-x_1)+(y_2'-y_1')(y_2-y_1)}{\sqrt{(x_2'-x_1')^2+(y_2'-y_1')^2}\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}})$

c) $\begin{cases} x' = scos\theta x - ssin\theta y + t_x \\ y' = ssin\theta x + scos\theta y + t_y \end{cases}$

$\begin{cases} t_x = x' - scos\theta x - ssin\theta y \\ t_y = y' - ssin\theta x + scos\theta y \end{cases}$

d) $\mathbf{v}' = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$ $\mathbf{v} = \begin{pmatrix} \frac{-1}{2} \\ \frac{1}{2} \end{pmatrix}$

Using the equations retrieved in a), b) and c)
$s = 2, \theta = \frac{\pi}{2}, t_x = 0, t_y = -1$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = 2\begin{pmatrix} \cos(\frac{\pi}{2}) & -\sin(\frac{\pi}{2}) \\ \sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

2

Implementation code:

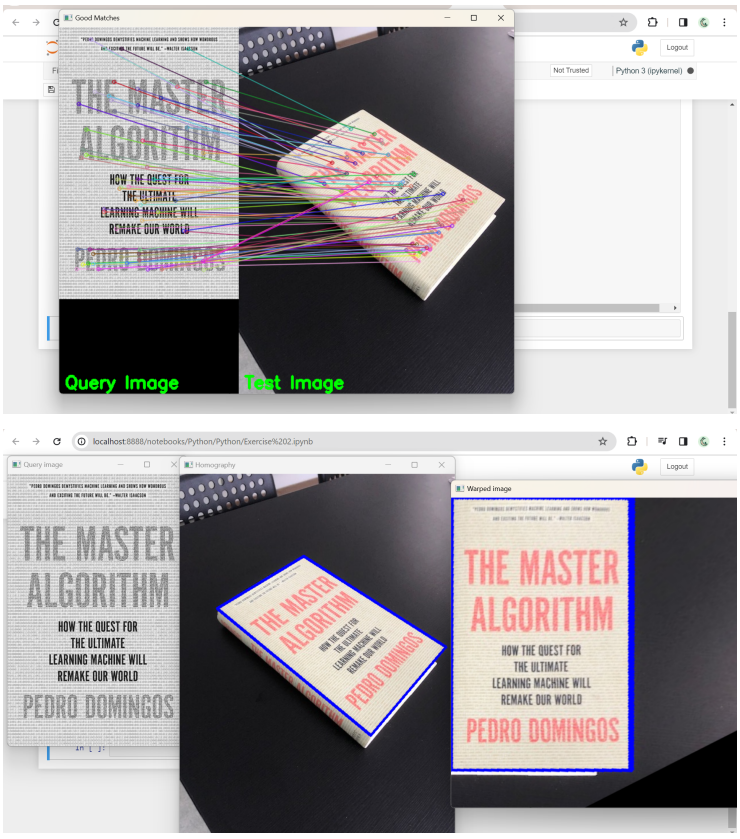
```
##-your-code-starts-here-##
grayframe = cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)
keyp_grayframe, descrip_keyframe = sift.detectAndCompute(grayframe, None)
matches = flann.knnMatch(descrip_image, descrip_keyframe, k=2)
##-your-code-ends-here-##

##-your-code-starts-here-##
```

```
matrix, mask = cv2.findHomography(query_pts, test_pts, cv2.RANSAC, 5.0)
##-your-code-ends-here-##

##-your-code-starts-here-##
im_warped = cv2.warpPerspective(test_img, np.linalg.inv(matrix), (720, 540))
##-your-code-starts-here-##
```

Plot for task 2:



3

Implementation code:

```
##-your-code-starts-here-##
x, y, w, h = faces[0]
roi_gray = gray[y:y+h, x:x+w]
```

```
    ##-your-code-ends-here-##

    ##-your-code-starts-here-##
    p0 = p0 + np.array([[x, y]])
    ##-your-code-ends-here-##

    ##-your-code-starts-here-##
    good_points = p1[isFound == 1]
    ##-your-code-starts-here-##

    ##-your-code-starts-here-##
    for point in good_points:
        x, y = point.ravel()
        cv2.drawMarker(img, (int(x), int(y)), (0, 255, 0), markerType=cv2.MARKER_CROSS, mark
                                    thickness=2)
    ##-your-code-ends-here-##

    ##-your-code-starts-here-##
    p0 = good_points.reshape(-1, 1, 2)
    gray_prev = gray.copy()
    ##-your-code-ends-here-##
```

Plot for task 3:
a) Main parts of the program:

- Initialization: previous frame and list of previous points.
- Detection: If the number of tracked points (p0) is less than or equal to 10, the program enters the detection phase. Extract ROI and identify trackable points.
- Tracking: If a sufficient number of trackable points are detected, the program enters the tracking phase. Calculate optical flow to estimate the new locations of the tracked points (p1). Select only the valid points that were successfully tracked and update the list of tracked points (p0) accordingly.
- Display: Display the video feed with the tracked points.

b) Problems:

- Tracked points may drift away from the actual feature locations due to cumulative errors in the optical flow estimation.
- The motion might be too large.

Solution: - Implement Lucas Kanade with pyramid - Parameter tuning (minimum distance, window size, etc.) for better performance - Use SIFT to detect features

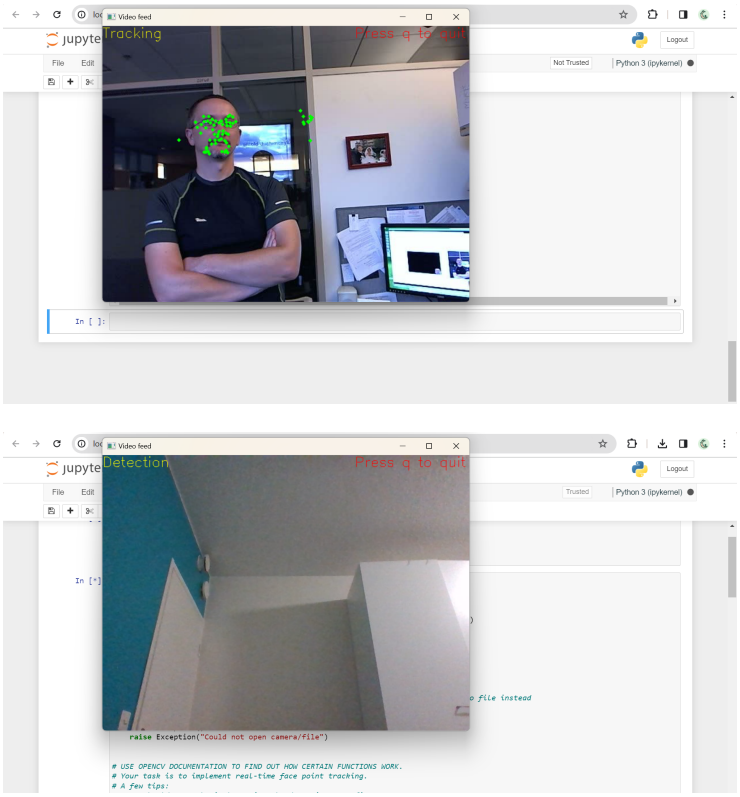


Figure 1: No face is tracked