CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Problem Set 04

Lucas Santos Zatta

oriented by

Flávio Luis Cardeal Pádua

Belo Horizonte

2021

Introduction

The goal of this work is to try to apply concepts seen in class as well as formulas in order to perform image analyzes. To solve this problem set, the codes implemented integrate functions created by the author with library functions(mostly, due to performance and time reasons).

For this problem set(problem set 4) we will be using Python3 with the libraries **Numpy**, **OpenCV** and **Matplotlib**. It's important to make sure that these libs are properly installed and fully working so that we can achieve the desired end results.

Questions

1. Variations of the Horn-Schunck Algorithm

```
import numpy as np
import cv2
cap = cv2.VideoCapture('traffic.mp4')
video_mode = False
print("Choose the app mode:")
print("1: Image Mode tracing")
print("2: Real Time Video Mode tracing”)
x = int(raw_input())
if x == 2:
   video_mode = True

feature_params = dict(maxCorners=100,
                      qualityLevel=0.1,
                      minDistance=7,
                      blockSize=7)
# Parameters for lucas kanade optical flow
lk_params = dict(winSize=(15, 15),
                 maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

```python
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)
# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)
generate_image = 0
while(1):
   ret, frame = cap.read()
   if generate_image == 0 and video_mode == False:
       cv2.imwrite('initial_frame.png', frame)
   frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   # calculate optical flow
   p1, st, err = cv2.calcOpticalFlowPyrLK(
       old_gray, frame_gray, p0, None, **lk_params)
   # Select good points
   good_new = p1[st == 1]
   good_old = p0[st == 1]
   # draw the tracks
   for i, (new, old) in enumerate(zip(good_new, good_old)):
       a, b = new.ravel()
       c, d = old.ravel()
       mask = cv2.line(mask, (a, b), (c, d), (255, 255, 255), 2)
       frame = cv2.circle(frame, (a, b), 1, (255, 255, 255), -1)
   img = cv2.add(frame, mask)
   if video_mode == True:
       cv2.imshow('frame', img)
   generate_image += 1
   if generate_image == 24 and video_mode == False:
       cv2.imwrite('final_frame.png', img)
       cv2.imshow('Motion Capture', img)
       cv2.waitKey()
       break
   k = cv2.waitKey(30) & 0xff
   if k == 27:
       break
   # Now update the previous frame and previous points
   old_gray = frame_gray.copy()
   p0 = good_new.reshape(-1, 1, 2)
cv2.destroyAllWindows()
cap.release()
```

Figure 1: Horn-Schunck Algorithm result

2. Mean-Shift Segmentation in OpenCV [1]

The intuition behind the meanshift is simple. Consider you have a set of points. (It can be a pixel distribution like histogram backprojection). You are given a small window (may be a circle) and you have to move that window to the area of maximum pixel density (or maximum number of points). It is illustrated in the simple image given below:

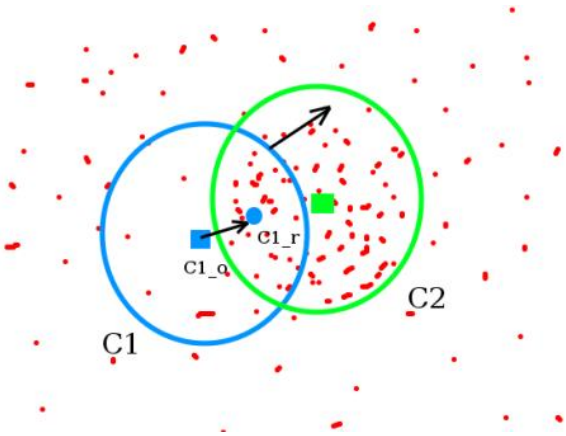

Figure 2: Mean-Shift Illustrated

The initial window is shown in blue circle with the name "C1". Its original center is marked in blue rectangle, named "C1_o". But if you find the centroid of the points inside that window, you will get the point "C1_r" (marked in small blue circle) which is the real centroid of the window. Surely they don't match. So move your window such that the circle of the new window matches with the previous centroid. Again find the new centroid. Most probably, it won't match. So move it again, and continue the iterations such that the center of window and its centroid falls on the same location (or within a small desired error). So finally what you obtain is a window with maximum pixel distribution.

As example, we will be using the following image of the band *Sticky Fingers:*

Figure 3: Example image for Mean_Shift problem

Figure 4: Results side by side

3. Detection of Calibration Marks

The Algorithm used for detection of calibration marks was the following:

```
import numpy as np
import argparse
import cv2

image = cv2.imread("boardPhoto.jpeg")
height, width = image.shape[:2]

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
tret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)

lines = cv2.HoughLinesP(image=thresh, rho=1, theta=np.pi /
                        500, threshold=10, minLineLength=25,
maxLineGap=10)

# Draw the lines
linesDrawing = np.ones((height, width, 3))

# For each line in the resulting set
print(lines)
for line in lines:
   x1, y1, x2, y2 = line[0]
   print(str(x1) + " " + str(y1) + " " + str(x2) + " " + str(y2))
   cv2.line(linesDrawing, (x1, y1), (x2, y2), (0, 255, 0), 2)


# find Harris corners
dst = cv2.cornerHarris(gray, 2, 3, 0.04)
dst = cv2.dilate(dst, None)
ret, dst = cv2.threshold(dst, 0.01*dst.max(), 255, 0)
dst = np.uint8(dst)

# find centroids
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)

# define the criteria to stop and refine the corners
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.001)
corners = cv2.cornerSubPix(gray, np.float32(
   centroids), (5, 5), (-1, -1), criteria)
```

```
# Now draw them
res = np.hstack((centroids, corners))
res = np.int0(res)
image[res[:, 1], res[:, 0]] = [0, 0, 255]
image[res[:, 3], res[:, 2]] = [0, 255, 0]


# show the images
cv2.imshow("Image", image)
cv2.imshow("Lines", linesDrawing)

cv2.waitKey(0)

# destroy all windows
cv2.destroyAllWindows()
```

Image used in this problem was taken as shown in Figure 5.

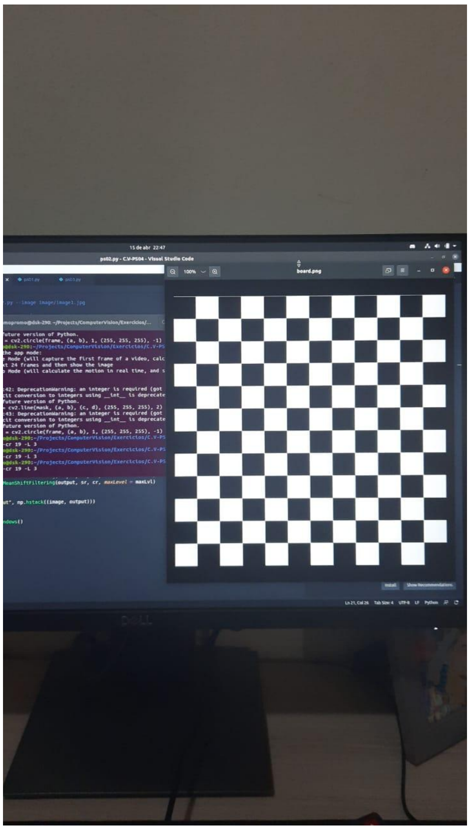

Figure 5: Example image for calibration mark detection

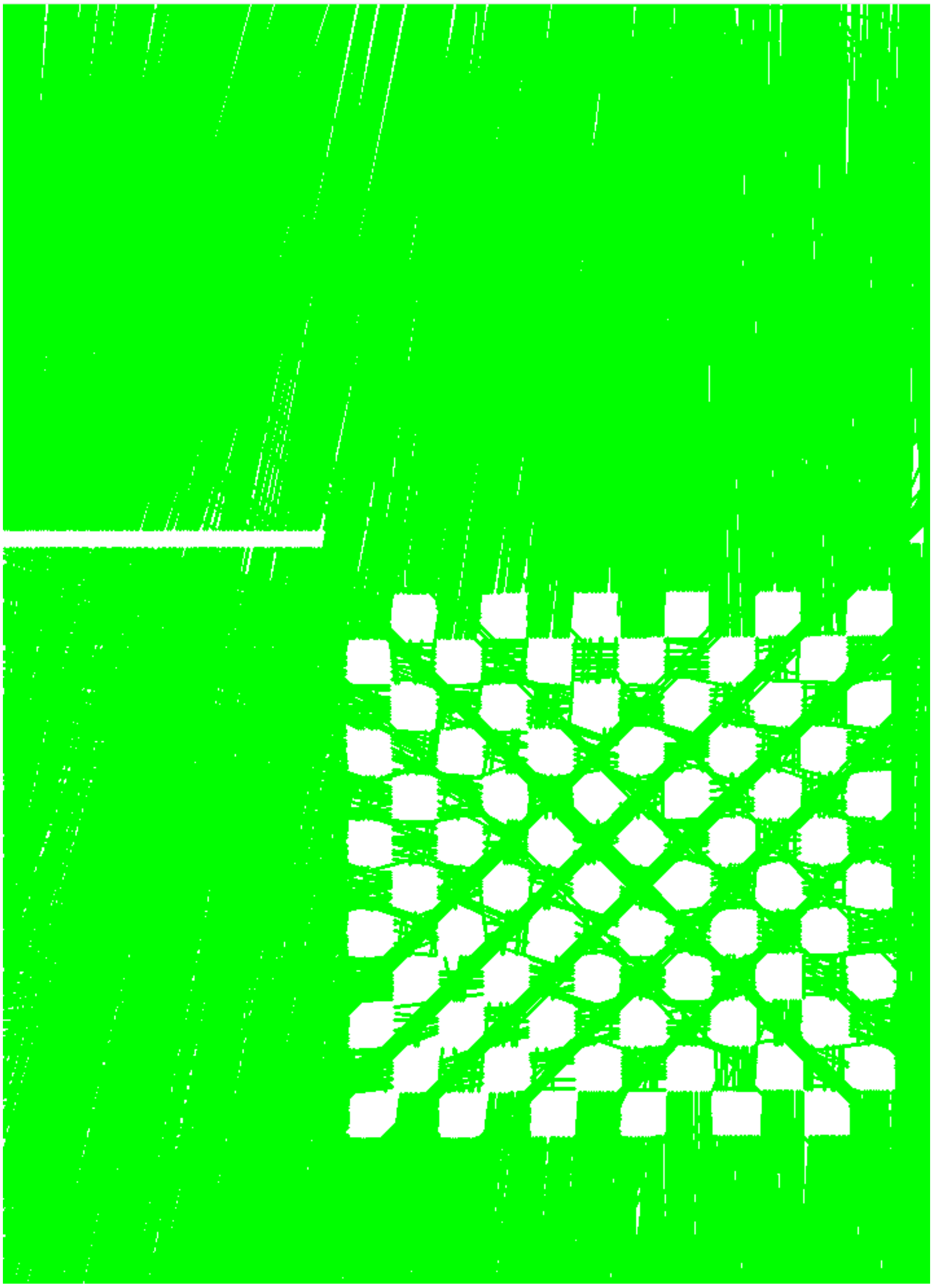

Figure 6: Result from calibration mark detection

References

[1] OpenCV. Tutorial: Meanshift and camshift
https://docs.opencv.org/3.4/d7/d00/tutorial_meanshift.html