2D Image Processing - Exercise Sheet 3: SIFT and Optical FLow

The notebook guides you through the exercise. Read all instructions carefully.

- **Deadline**: 13.06.2022 @ 23:59
- **Contact**: Michael.Fuerst@dfki.de
- Submission: As PDF Printout; filename is `ex03_2DIP_group_XY.pdf`, where XY is replaced by your group number.
- Allowed Libraries: Numpy, OpenCV and Matplotlib (unless a task specifically states differently).
- Copying or sharing code is NOT permitted and results in failing the exercise. However, you could compare produced outputs if you want to. (Btw, this includes copying code from the internet.)

Submission as PDF printout. You can generate a PDF directly from jupyterlab or if that does not work, export as HTML and then use your webbrowser to convert the HTML to a PDF. For the printout make sure, that all text/code is visible and readable. And that the figures have an appropriate size. (Check your file before submitting, without outputs you will not pass!)

---

0. Infrastructure: Cloud Image Loader

This is an image loader function, that loads the images needed for the exercise from the dfki-cloud into an opencv usable format. This allows for easy usage of colab, since you only need this notebook and no other files.

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

import requests
from PIL import Image

def get_image(name, no_alpha=True):
    url = f'https://cloud.dfki.de/owncloud/index.php/s/THLirfoB6SYTetn/download?path=&
    image = np.asarray(Image.open(requests.get(url, stream=True).raw))
    if no_alpha and len(image) > 2 and image.shape[2] == 4:
```

```
        image = image[:,:,:3]
    return image[:,:,::-1].copy()
```

---

1. SIFT and Feature Matching (8 Pts)

Theory (5 Pts)

Hint: Each 0.5 points expects one argument/step/etc.

1. List what methods are used for comparing two patches in the image (1.5 Pts).
2. Explain the ideas and steps of SIFT feature detection in detail. (2 Pts)
3. What are the advantages of SIFT compared to Harris? (0.5 Pts)
4. Also explain the idea of HOG as a descriptor. (1 Pts)

Solution:

- **Patch Comparison**:

In the Simple Approach we match square windows around the point using the following steps: Here first we detect the Harris corners. Then we compute the similarity and correlation scores. Finally the point with the highest score is selected.

a. Using Proximity:

Search takes place with a point being selected and matching using a disparity window.

b. Using cross-correlate:

Search takes place by comparing the intensity of the neighborhood.

c. Using invariant descriptors:

Transforming an image helps to identify it's invariance to light, scale, view point etc.

- **SIFT**:

The idea behind SIFT is to transform an image such that it is invariant to scaling by using a keypoint descriptor. The keypoint descriptor uses parameters such as number of orientations and an array of orientation histograms (weighted by gradient magnitude and a Gaussian function with lambda equal to 1.5 times the width of the descriptor window) The steps involved in achieving this are as follows:

```
1. Scale-space extrema detection
```

Extract scale and rotation invariant interest points (i.e., keypoints) This is done by computing edge orientation around each pixel and then the histogram of the divided cells.

```
2. Keypoint localisation
```

Determine location and scale for each interest point. This is done by making a weighted orientation histogram. The peak position gives the orientation of the peak point and the accurate position is determined by fitting. The weak keypoints are eliminated by keeping the peak.

```
3. Orientation assignment
```

Assign one or more orientations to each keypoint. A degree range is selected and the angle are divided into buckets (they contain sum of weighted gradient magnitudes).

```
4. Keypoint descriptor
```

Use local image gradients at the selected scale. A partial voting method is used where histogram entries are distributed into adjacent bins by multiplying with a weight of its distance from the bin it belongs to.

- **SIFT vs Harris**:

SIFT algorithm is more vital and robust than Harris corner detection algorithm. The matching keypoint point from Harris detection can be obtained with high elapsed time and is very difficult to get highly correct and exact match keypoint. The major advantage that SIFT has over Harris Detector is that SIFT is scale invariant whereas Harris isn't.

- **HOG**:

a. Take a 16x16 window around detected intrested point. b. Compute edge orientation for each pixel. Look at orientation gradient. c. Divide into a 4x4 grid of cells. d. Compute histogram in each cell.

1. Compute the angle of the gradient of each histogram cell. Consider only 8 bins from 0 to 2pi(360) Inside the 4x4 cell, you'll get a histogram of the orientation of the gradient. Basically it's a description of edges and moves. Thereby 16 histograms* 8 orientations = 128 features. This will build our feature descriptor. Then we use vectors to compare and see if they match the same point.

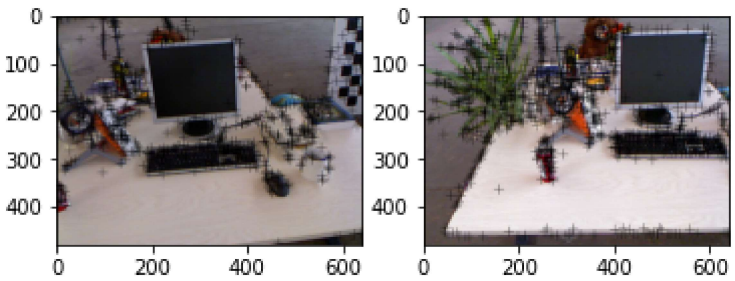
2. The reason for choosing 16x16 window was by trial and error. A bigger window is too descriptive and a smaller window is not descriptive enough. Similarly for 8 bins, another value would change the sensitivity measure. It is done in such a way that we have the smallest possible number of features which are distinctive.

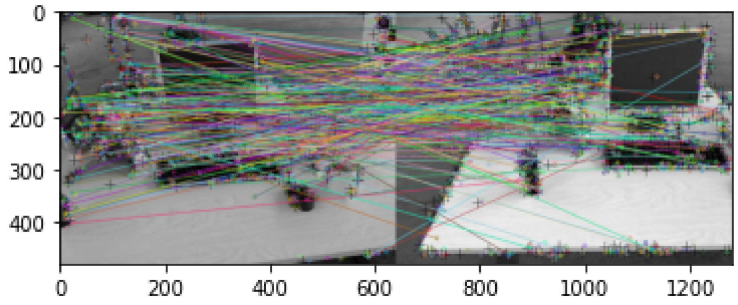
Programming Task (2 Pts)

Given two RGB images, implement SIFT feature detection on both images, create proper feature descriptors and match the features between these two images and visualize the results.

In [2]:

```
# Solution
image1 = get_image("img1.png")
image2 = get_image("img2.png")

# Implementing SIFT feaure detection on images - img1.png and img2.png
# Implementing SIFT feature detection on img1.png[image1]
SIFT_1 = cv2.SIFT_create()
kp_1 = SIFT_1.detect(image1, None)

for marker_var in kp_1:
    img1_output = cv2.drawMarker(image1, tuple(int(i) for i in marker_var.pt), None)
    cv2.imshow("SIFT_1", img1_output)

# Implementing SIFT feature detection on img2.png[image2]
SIFT_2 = cv2.SIFT_create()
kp_2 = SIFT_2.detect(image2, None)


for marker_var in kp_2:
    img2_output = cv2.drawMarker(image2, tuple(int(i) for i in marker_var.pt), None)
    cv2.imshow("SIFT_2", img2_output)


# Creating feature descriptors and matching the features between these two images
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

gray1 = cv2.cvtColor(image1, cv2.COLOR_RGB2GRAY)
gray2 = cv2.cvtColor(image2, cv2.COLOR_RGB2GRAY)

sift = cv2.SIFT_create()


kp_3, descriptors1 = sift.detectAndCompute(gray1,None)
kp_4, descriptors2 = sift.detectAndCompute(gray2,None)

match = bf.match(descriptors1, descriptors2)
match = sorted(match, key = lambda x:x.distance)

result = cv2.drawMatches(gray1, kp_3, gray2, kp_4, match[300:600], None)

plt.subplot(1,2,1)
plt.imshow(img1_output[:,:,::-1])
plt.subplot(1,2,2)
plt.imshow(img2_output[:,:,::-1])
plt.show()

plt.imshow(result)
plt.show()
```





Explanation (1 Pts):

On the above program we are implementing the SIFT feature detection on both the images. We are creating a SIFT and detecting the points and displaying the image by marking those points. Once the feature descriptors are completed we are matching the features on both the images by using the BFMatcher. Then again create the SIFT along with the descriptors. Then match those descriptors by using the drawmatches function. Then display the image after matching the descriptors.

---

2. Motion Estimation with Optical Flow (10 Pts)

Real time object tracking is a challenging task. Optical flow gives valuable information about the object movement. And moving object detection and tracking is an evolving research field due to its wide applications in traffic surveillance, 3D reconstruction, motion analysis (human and non-human), activity recognition, medical imaging, etc.

Theory (5 Pts):

Hint: Each 0.5 points expects one argument/step/etc.

1. Explain the difference between motion field and optical flow with example. (1.5 Pts)
2. Analyze if the three assumptions for optical flow tracking hold true when you would track bubbles in a glass of carbonated water. (1.5 Pts)
3. Explain the aperture problem in optical flow. How can one solve this problem by estimating the global geometric structure of the scene? (1 Pts)
4. The proposed Lucas-Kanade method has an issue with large movement. Why? How can you deal with large movement in the image? A conceptual explanation is sufficient. (1 Pts)

Solution:

1. **Motion Field vs Optical Flow**

    - Motion Field: The motion field is the projection of the 3D scene motion into the image. Motion field is equal to the real world 3D motion.

- Optical Flow: Optical flow is the apparent motion of brightness patterns in the image. Optical flow does not always corresponds to motion field. Optical flow is an approximation of the motion field.
- Example: a) A smooth sphere is rotating under constant illumination. Thus the optical flow field is zero, but the motion field is not (no visible brightness change) b) A fixed sphere is illuminated by a moving source - the shading of the image changes. Thus the motion field is zero, but the optical flow field is not (brightness change due to other factors than motion).

2. **Tracking Bubbles**

   A. First assumption->Brightness consistency. In a small region of a bubble, the brightness will remain the same and only the location will change. There will be no abrupt changes in the brightness. Although it also depends on the size of a bubble.
   B. Second assumption->Spatial coherence. The motion of whole bubble would be the same. All the neighbouring points within the bubble will have the same motioning, if we consider a pixel within the bubble.
   C. Third assumption->Temporal persistence. There are no abrupt changes in the motion of a bubble. In two consecutive frames, there will not be a huge difference in the motion of the considered patch. So it is achieved.

3. **Aperture Problem**

   - Problem : The aperture problem refers to the fact that the motion of a one-dimensional spatial structure, such as a bar or edge, cannot be determined unambiguously if it is viewed through a small aperture such that the ends of the stimulus are not visible. Different motions can be classified as similar or similar motions can be classified as different.

   - Solution: This problem can be solved by Horn-Schunck method which introduces a global constraint of smoothness. This algorithm assumes smoothness in the flow over the whole image. Thus, it tries to minimize distortions in flow and prefers solutions which show more smoothness.

4. **Lucas-Kanade large Image Movement**
   - Problem: The motion is large, ie., larger than a pixel. It is less sensitive to image noise than point-wise methods. On the other hand, since it is a purely local method, it cannot provide flow information in the interior of uniform regions of the image.
   - Solution: We can deal the larger movements by iterative refinement.

   A. Estimate velocity at each pixel by solving Lucas Kanade equations
   B. Warp I_t towards I_{t+1} using the estimated flow feild (Basically just interpolation)
   C. Repeat untill convergence

Programming Task (4 Pts)

Configure your OpenCV and read the provided mp4 video. Apply the Lucas-Kanade Optical Flow on it. The result should look like the given example.

Video: https://cloud.dfki.de/owncloud/index.php/s/THLirfoB6SYTetn/download?path=&files=tracking.mp4

In [3]:

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import os as os2

path = os2.getcwd()
final_path = os2.path.join(path)

feature_parameters = dict(maxCorners = 100, qualityLevel = 0.3,
                          minDistance = 7, blockSize = 7)

# Parameters for lucas kanade optical flow
lucasKanade_params = dict( winSize  = (15,15), maxLevel = 2,
                          criteria = (cv.TERM_CRITERIA_EPS |
                                      cv.TERM_CRITERIA_COUNT, 10, 0.03))

# Create some random colors
radom_color = np.random.randint(0, 255, (100, 3))

video_capture = cv.VideoCapture('https://cloud.dfki.de/owncloud/index.php/s/THLirfoB6S

ret, old_frame = video_capture.read()
old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
p0 = cv.goodFeaturesToTrack(old_gray, mask = None, **feature_parameters)

# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)

# Size of the video frame
fr_width = int(video_capture.get(3))
fr_height = int(video_capture.get(4))
fr_size = (fr_width, fr_height)
```

```
# Frame rate of video
fr_rate = 30

final_video = cv.VideoWriter(final_path+'/Final.avi',
                             cv.VideoWriter_fourcc(*'MJPG'),fr_rate, fr_size)

while(True):
    ret, frame = video_capture.read()
    if not ret:
        #print('No frames grabbed!')
        break
    fr_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    # calculate optical flow
    p1, st, error = cv.calcOpticalFlowPyrLK(old_gray, fr_gray, p0,
                                            None, **lucasKanade_params)

    # Select good points
    if p1 is not None:
        new_point = p1[st==1]
        old_point = p0[st==1]

    # draw the tracks
    for i, (new, old) in enumerate(zip(new_point, old_point)):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv.line(mask, (int(a), int(b)), (int(c), int(d)),
                       radom_color[i].tolist(), 2)
        frame = cv.circle(frame, (int(a), int(b)), 5,
                          radom_color[i].tolist(), -1)


    image = cv.add(frame, mask)

    cv.imshow("output",image)


    final_video.write(image)

    if cv.waitKey(fr_rate) & 0xFF == ord('q'):
        break

    old_gray = fr_gray.copy()
    p0 = new_point.reshape(-1, 1, 2)


video_capture.release()
final_video.release()
cv.destroyAllWindows()
```

In [4]:

```
import cv2
import glob

videocapture = cv2.VideoCapture('C:/Users/chait/Downloads/Final.avi');
success,img = videocapture.read()
count = 0
while success:
     # saving frame as JPEG file
    cv2.imwrite("C:/Users/chait/Downloads/frames/frame%d.jpg" % count, img)
```

```
    success,img = videocapture.read()
    print('Read a new frame: ', success)
    count += 1

image_list = []
for filename in glob.glob('C:/Users/chait/Downloads/frames/*.jpg'):
    im=Image.open(filename)
    image_list.append(im)
```

Explanation (1 Pts):

TODO Explain the visualizations you create to show your implementation works. (Also if it does not work, write what goes wrong and why you think this is the case)

Here we apply the Lucas-Kanade method to track some points on a video. To track the points, first, we find the points to be tracked. For finding these points, we will use cv2.goodFeaturesToTrack() we will capture the first frame and detect some corner points. These points will be tracked using the LucasKanade Algorithm provided by OpenCV cv2.calcOpticalFlowPyrLK().We can see the colored lines in the final frame for the change in positions.

In [ ]: