2D Image Processing - Exercise Sheet 2: Edges and Corners

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

- **Deadline**: 30.05.2022 @ 23:59
- **Contact**: Michael.Fuerst@dfki.de
- Submission: As PDF Printout; filename is `ex01_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.)
- Points: total 21, passing 12.5 or more

NEWS: 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=&files={name}'
    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. Understanding Image Features

It is a tricky task for a computer to find features. The first step is called feature detection, once you have found it, you should be able to find the same in other images. Then in the second step, you need to find a way to describe the features you have found, which is called feature description. Once you have the features and its description, you can find same features in all images and align them, stitch them together or do whatever you want.

Theory (3.5 Points)

Removed questions 8-10.

1. Explain what the pros and cons of local features (edge, corner and point).
2. Explain the criteria of deigning a good edge detector, give an example and explain the rough process.
3. Explain the core ideas of feature detection mathematically.
4. Explain the ideas of Harris corner detector based on the answer of (3).
5. Is the Harris corner detector robust with respect to intensity changes in the image? Why or why not?
6. Is the Harris corner detector robust with respect to rotation? Why or why not?
7. Explain the importance of invariance when describe a feature. How to achieve invariance?
8. ~~List what methods are used for comparing two patches in the image.~~
9. ~~Explain the ideas and steps of SIFT feature detection in detail. What are the advantages of SIFT compared to Harris?~~
10. ~~Also explain the idea of HOG as a descriptor.~~

Solution:

1)The pros of local features:

a) Locality The features are local, so robust to occlusion and clutter.

b) Distinctiveness Large database of objects can be differentiated.

c) Quantity Can handle immensely large quantities in an image.

d) Efficiency Acceptable performance is achieved.

e) Generality Different types of features can be exploited in different situatuons.

The cons of local feauturs:

a) Feature mismatch When a detector tries to match the local features in 2 images then the features may not match accurately. The is no accurate pairing.

b) Keypoint detection Hard to detect the same keypoint independently in different images.

2)Criteria for designing a good edge detector:

a) Good Detection The probability of false positives (spurious edges caused by noise) and false negatives (missing real edges) must be minimised.

b) Good localization The edge closest to true edge must be detected.

c) Single response Only a single point must be returned for one true edge point. The number of local maxima must be minimised.

Example: Lets consider the true edge to be 4 points along a vertical line. A good edge detector should make sure to reduce the effect of the noise causing the points to be scattered in the X-direction. It should also make sure to localise the points such that they form a straight line. Another factor would be to restrict a single response point along the X-axis.

3)Consider a Window W shifted by (u,v). The shift results in a summing up sqaured distances error E(u,v).

$$E(u, v) = \sum_{(x,y) \in W} [I(x + u, y + v) - I(x, y)]^2$$

Considering the Taylor series expansion when the motion (u,v) is small then first order approximation is good and the following assumption is made:

$$E(u, v) \approx \sum_{(x,y) \in W} [I(x, y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - I(x, y)]^2$$

$$E(u, v) \approx \sum_{(x,y) \in W} \left[\begin{bmatrix} I_x & I_y \end{bmatrix} \quad \begin{bmatrix} u \\ v \end{bmatrix} \right]^2$$

$$E(u, v) = \sum_{(x,y) \in W} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

4)In the above example, we need to find the direction in which we will get the largest and smallest E values. These directions can be found using the eigenvectors of H, i.e., $\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$

This can be done by investigating the shape of the error function. We visualize H as an ellipse with axis lengths and directions determined by its eigenvalues and eigenvectors.

5)The Harris corner detector is partially robust to intensity changes, i.e., partially invariant to affine intensity changes. When the shift takes place, the intensity value is added by a constant term thereby not affecting it completely.

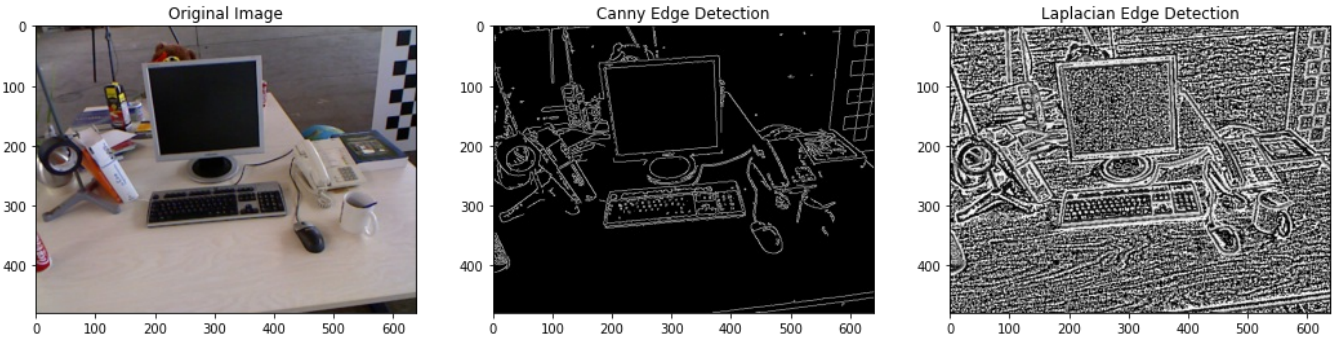
6)The Harris corner detector is robust to rotation, i.e., invariant to rotation. The reason is that the eigen values remain the same even when a shape is rotated. Hence the value is not affected.

7)A feature is said to be invariant if transformations to the image does not result in changes to the feature. Invariance is achieved through geometric(translation, rotation, scale) or photometric(brightness, exposure) transformations.

Programming Task (2 Pts)

Given an RGB image, implement Canny edge detection and Laplace edge detection and visualize the results.

In [2]:
```
# Solution
image = get_image("img1.png")
#converting image to grayscale
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)


ROWS=1; COLS=3
plt.figure(figsize=(18,6))
#applying 5*5 gaussian filter to remove the noise in the gray image since the edge detection is
#susceptible to noise
gblur = cv2.GaussianBlur(gray,(5,5),0)

#apply canny edge detection by giving lower and upper threshold
canny_edge = cv2.Canny(image, 100, 200)

#apply laplacian edge detection to the blurred image
laplacian_edge = cv2.Laplacian(gblur, cv2.CV_64F, ksize=3)
lap_image = np.uint8(laplacian_edge)
plt.subplot(ROWS, COLS, 1) # Draw in first slot
plt.title("Original Image")
plt.imshow(image[:,:,::-1])

plt.subplot(ROWS, COLS, 2) # Draw in second slot
plt.title("Canny Edge Detection")
plt.imshow(canny_edge, 'gray') #displaying the image in the gray format

plt.subplot(ROWS, COLS, 3) # Draw in third slot
plt.title("Laplacian Edge Detection")
plt.imshow(lap_image, 'gray')
plt.show()
```



Explanation (1 Pts):

Canny Edge detection:

Canny Edge detection is a famous edge detection algorithm which involves multi-steps

-Noise Reduction

-To find intensity Gradient

-Non-Maximum Suppression

-Hysteresis Thresholding

In openCV we have an inbuilt function which puts all these functions into single function. The inbuilt function takes an image as an input with high and low threshold value for hysteresis process. The output will be having only the edges which is present in the image. We have displayed the image in the gray format.

Laplacian Edge detection:

Laplacian edge detection is a technique to identify the edge in the given image. We need to take the second order derivative in both dimensions.

Laplacian operator is defined by:

$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

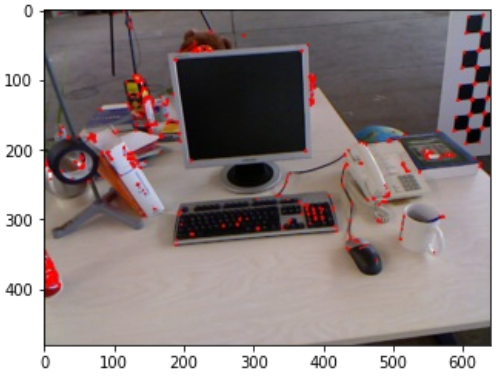
The Laplace operator is implemented in openCV by Laplacian function which takes the input image as gray scale image, the depth of the destination image along with the kernel size. The output image is displayed in the grayscale image.

By comparing both the function canny edge and laplacian edge detection, the Canny edge detection is more efficient than the laplacian edge. The noise in the Laplacian edge is more when compared to the canny edge detection which is more clear and sharp.

Programming Task (1 Pts)

Given an RGB image, implement Harris point detection and visualize the results.

In [3]:
```
# Solution
image = get_image("img1.png")

#converting the image into gray scale image
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)

#converting the grayscale image into floating point since the corner harris function takes floating point
fgray = np.float32(gray)
#Applying the corner harris function with block size 2 and the ksize is 3
charris = cv2.cornerHarris(fgray,2,3,0.04)
#result is dilated for marking the corners
dst = cv2.dilate(charris,None)
# Threshold for an optimal value, it may vary depending on the image.
image[dst>0.01*dst.max()]=[0,0,255]

plt.imshow(image[:,:,::-1])
plt.show()
```



Explanation (1 Pts):

TODO Explain your visualization. Where there is a large variation in the intensity of the image in all directions that regions are known as corners. It is basically finds the difference in the intensity for a displacement of (u, v) in all directions. So we have to maximize difference of function E(u,v) to detect the corner.

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2$$

After all the derivations the final equation which determines if the window contains a corner or not is-

$$R = det(M) - k(trace(M))^2$$

If R value is small, then the region is flat.

If R value is less than 0, then the region is edge.

If R vlaue is large, then the region is corner.

In the program we are using the corner haris funtion to detect the cornerS in the image. We are converting the given image into grayscale image and that into floating point since the corner harris function takes floating point. We are passing the block size of the window, the aperture parameter of soble derivative and harris detector free parameter in the equation. Then are dilating the output image and reverting back to the original image with optimal threshold value marking the corners with RED color.

~~Programming Task (0 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.~~

---

2. Canny Edge Detection

In the lecture we have discussed Canny edge detection and you have also tried it in the last exercise with OpenCV. Now, it is time to follow the original idea to implement Canny edge detection from scratch. (No libraries allowed!)

The Canny edge detection algorithm is composed of 5 steps:

1. Noise reduction
2. Gradient calculation
3. Non-maximum suppression
4. Hysteresis thresholding

5. Edge Tracking by Hysteresis.

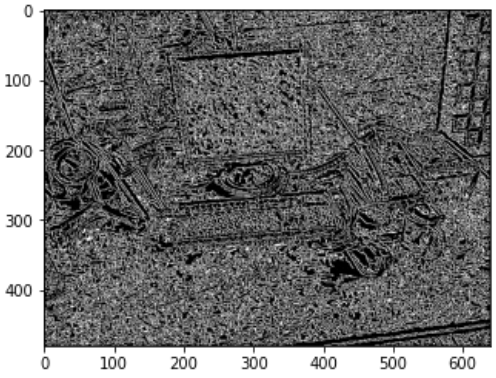
Programming Task (5 Pts)

Follow the tutorial and implement Canny edge detection step by step on a grayscale image.

Tutorial: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

In [12]:
```
# Solution

# Solution
import scipy.misc
from scipy.ndimage import convolve

image = get_image("img1.png")
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)

class cannyEdgeDetector:
    def __init__(self, imgs, sigma=1, kernel_size=5, low_pixel=75, high_pixel=255, low_threshold=0.05,
                 high_threshold=0.15):
        self.imgs = imgs
        self.imgs_final = []
        self.img_smoothed = None
        self.gradientMat = None
        self.thetaMat = None
        self.nonMaxImg = None
        self.thresholdImg = None
        self.low_pixel = low_pixel
        self.high_pixel = high_pixel
        self.sigma = sigma
        self.kernel_size = kernel_size
        self.low_threshold = low_threshold
        self.high_threshold = high_threshold
        return

    def gaussian_kernel(self, k_size, sigma_value=1):
        k_size = int(k_size) // 2
        x, y = np.mgrid[-k_size:k_size+1, -k_size:k_size+1]
        var = 1 / (2.0 * np.pi * sigma_value**2)
        gk =  np.exp(-((x**2 + y**2) / (2.0*sigma_value**2))) * var

        return gk



    def sobel_filters(self, img):
        Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
        Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)

        Ix = convolve(img, Kx)
        Iy = convolve(img, Ky)

        G = np.hypot(Ix, Iy)
        G = G / G.max() * 255
        theta = np.arctan2(Iy, Ix)
        return (G, theta)

    def non_max_suppression(self, img, D):
        M, N = img.shape
        Z = np.zeros((M,N), dtype=np.int32)
        angle = D * 180. / np.pi
        angle[angle < 0] += 180


        for i in range(1,M-1):
            for j in range(1,N-1):
                try:
                    q = 255
                    r = 255

                   #angle 0
                    if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                        q = img[i, j+1]
                        r = img[i, j-1]
                    #angle 45
                    elif (22.5 <= angle[i,j] < 67.5):
                        q = img[i+1, j-1]
                        r = img[i-1, j+1]
                    #angle 90
                    elif (67.5 <= angle[i,j] < 112.5):
                        q = img[i+1, j]
                        r = img[i-1, j]
                    #angle 135
                    elif (112.5 <= angle[i,j] < 157.5):
                        q = img[i-1, j-1]
                        r = img[i+1, j+1]
```

```
                    if (img[i,j] >= q) and (img[i,j] >= r):
                        Z[i,j] = img[i,j]
                    else:
                        Z[i,j] = 0


                except IndexError as e:
                    pass

        return Z

    def threshold(self, img):

        high_threshold = img.max() * self.high_threshold;
        low_threshold = high_threshold * self.low_threshold;

        M, N = img.shape
        res = np.zeros((M,N), dtype=np.int32)

        weak = np.int32(self.low_pixel)
        strong = np.int32(self.high_pixel)

        strong_i, strong_j = np.where(img >= high_threshold)
        zeros_i, zeros_j = np.where(img < low_threshold)

        weak_i, weak_j = np.where((img <= high_threshold) & (img >= low_threshold))

        res[strong_i, strong_j] = strong
        res[weak_i, weak_j] = weak

        return (res)

    def hysteresis(self, img):

        M, N = img.shape
        weak = self.low_pixel
        strong = self.high_pixel

        for i in range(1, M-1):
            for j in range(1, N-1):
                if (img[i,j] == weak):
                    try:
                        if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1, j+1] == strong)
                            or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                            or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or
                            (img[i-1, j+1] == strong)):
                            img[i, j] = strong
                        else:
                            img[i, j] = 0
                    except IndexError as e:
                        pass
        return img


    def detect(self):


        self.img_smoothed = convolve(gray, self.gaussian_kernel(self.kernel_size, self.sigma))
        self.gradientMat, self.thetaMat = self.sobel_filters(self.img_smoothed)
        self.nonMaxImg = self.non_max_suppression(self.gradientMat, self.thetaMat)
        self.thresholdImg = self.threshold(self.nonMaxImg)
        img_final = self.hysteresis(self.thresholdImg)
        return img_final

detector = cannyEdgeDetector(gray, sigma=1.4, kernel_size=5, low_threshold=0.09, high_threshold=0.17,
                             low_pixel=100)
image_final = detector.detect()
output_image = Image.fromarray(image_final.astype(np.uint8))
plt.imshow(output_image,'gray')
plt.show()
```



Explanation (1 Pts):

Explanation (4 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)

Canny Edge detection is a famous edge detection algorithm which involves multi-steps functions

-Noise Reduction We need to remove the noise which is present in the image since the edge detection is susceptible to noise. We are using Gaussian filter to remove the noise.

-To find intensity Gradient The blurred image is then filtered with sobel kernel in both vertical and horizontal direction and to get the first derivative in horizontal direction Gx and vertical direction Gy. These gradient is always perpendicular to the edges.

-Non-Maximum Suppression In this function we are scanning the complete image to remove the unwanted pixels which may not constitute the edge. To get this, at every pixel, pixel is checked if it is a local maximum in its neighbhorhood in the direction of gradient.

-Hysteresis Thresholding In this function we are calculating two threshold values, minimum value and maximum value. Anything above the maximum value is considered as edge and anything below the minimum value is discarded.

In this program the output is having more noise when compared to the inbuilt function. The edges are not clear and sharp. This is might be due the image is converted into the arrays where all the operations are done to the arrays. Then the arrays are converted into the image where some of the operations might be miss out. So the inbuilt function is more efficient than the manuallyu coded program.

Theory (4 Pts)

1. Explain how to compute the edge strength (magnitude) and edge orientation.
2. Explain how to achieve non-maximum suppression.
3. Explain how to achieve Hysteresis thresholding.
4. Compare your result with the result from last exercise.

Solution:

1.Explain how to compute the edge strength (magnitude) and edge orientation. -> a)Edge strength is related to the local image contrast along the normal.On the basis of the extent of horizontal and vertical intensity gradients at pixel location in an input image, edge strength can be computed. Edge strength is computed by the formula:

$$||\Delta f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

b)Edge orientation or edge direction is perpendicular to the direction of maximum intensity change(i.e., edge normal) Edge direction is computed by the formula:

$$\theta = tan^{-1}\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right)$$

2.Explain how to achieve non-maximum suppression.

->Steps to achieve non-maximum suppression are as follows:

The non-maximun suppression algorithm finds the maximum value pixel in the edge direction by traversing all the points of the gradient intensity matrix. Each pixel has an edge direction in radians and pixel intensity[0-255]

a)Consider the size of the original gradient intensity matrix and create a matrix with the same size of original gradient intensity matrix and initialize it to 0.

b)Consider the angle value from the angle matrix and identify the edge direction.

c)Check if the itensity of the pixel in the same direction has a higher intensity than the pixel that is currently processed.

d)Return the processed image with the non-max suppression algorithm.

3.Explain how to achieve Hysteresis thresholding.

->To solve the problem of which edges are really edges and which are not, Hysteresis thresholding is used. Steps to achieve Hysteresis thresholding are as follows:

First step is to apply double threshold on an image. By applying double threshold, we can identify three kinds of pixels:

a)The pixels that have high intensity are strong pixels. They contribute to an output image.

b)The pixels that have not so strong intensity value and also not small enough to be considered as irrelevant for the edge detection are weak pixels.

c)Other pixels are considered as irrelevant for the edge detection.

The next step is to apply edge tracking by Hysteresis. Observing the threshold results, we can say that if there is atleast one strong pixel around the one being processed, then the hysteresis consists of transforming weak pixels into strong pixels.

4.Compare your result with the result from last exercise.

When we used the inbuilt function of the Canny edge detection the output of the image is more clear and the edges are more sharp which can be easily visible. However even though we use the same functions which is there in the inbuilt function, the output of the manually coded program is not that clear and the output image contains more noises and the edges are not sharp as compared to the output of the inbuilt function.

Free Points (2.5 Points)

Due to the tasks removed and since we do not want to mess with olat you receive 2.5 free points.

In [ ]: