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]:
```python
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 (5 Points)

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. **Pros**: Can be detected in linear time. Local features helps in object detection and classification since these features are independent of viewing conditions and clutter.

   Cons: Susceptible to noise when detecting. Loss of information in the final image.

2. Edge detection relies on detecting a sudden change in pixel value in an image. The rough process to compare each pixel with its neighbour, ie subtract two pixels values. The criterias of designing a good edge detector are:

   - Good detection : Minimise the probability of false negative and false positive, that is, detecting wrong edges due to noise in the image or missing out on the real edges.
   - Good localization : The detected edges must be somewhat close to the actual edges in position.
   - Single response : Minimize the number of local maxima, that is, the detector must return only a few or one point around the actual edge points.

   The best example of the edge detection is the Canny edge detector. To extract out the edges,

   - Image is filtered with Gaussian derivative
   - Then we take the normal of the gradient
   - Then we do thresholding and thinning, i.e. non maximum suppression, getting interpolated value between points.
   - Edge linking, where we take two threshold and construct the edge curve
3. Feature detetection relies heavily on the use of derivatives. The derivative of a continous function represents a sudden change in the function. First derivative on an image outputs local extreams when an edge is encountered. This core concept is applied further along with gausian to further enhance the feature detection methods.

4. Harris Corner Detector is a corner detector operator that refines features or corners from an image. It is built on the notion of finding the features based on the variation in intenisties as we move the refrence window patch along the image as described in above answer. The intensities of current patch are then subtracted from refrence patch and that gives us an area function. It'll be most when a corner is under the patch and that's how, features are detected. Mathematically speaking, E(u,v)=∑x,y w(x,y) [I(x+u,y+v) - I(x,y)^2] Here, Function E is derived by subracting the intensities of displaced refrence window (I(x+u,y+v)) and refrence window (I(x,y)) i.e. compute the gradient at each point. Then, using Taylor expansion, we expand the above equation and get H matrix and compute eigenvalues. Now, doing non maxima suppression, at the points where smaller eigen is greater than threshhold, chose the one with local maxima for features.

5. Harris corner detector responds good to major changes in the intensities, covering up the intensity changes and detecting features but it doesn't do well with some non geometric events like sudden illumination changes. It'll detect it as feature as it'll find it difficult to diffrentiate between it and geometric events while computing the maxima points.

6. Yes, because only the eclipse detected in intensity graph is rotated. The eigen value remains unchanged.

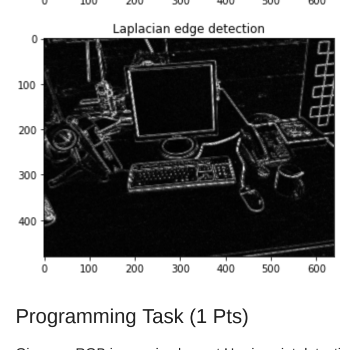
7. Invariance is a feature of objects that remains constant or doesnot change if manipulated. In computer vision, local features can be invariant to transformations, like geometric invariance which covers translation, rotation, scale and photometric invariance which covers brightness and exposure. Usually, when taking same picture from different viewpoints, it becomes very difficult to match them. So, to avoid this issue, invariance is important while describing a feature. To achieve invariance:

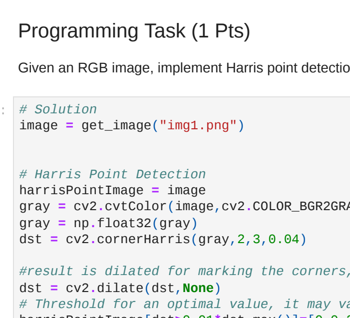
   - The detector should be invariant.e.g. Harris is invariant to rotation and translation.
   - Design invariant feature window (patch of pixels) or descriptor.

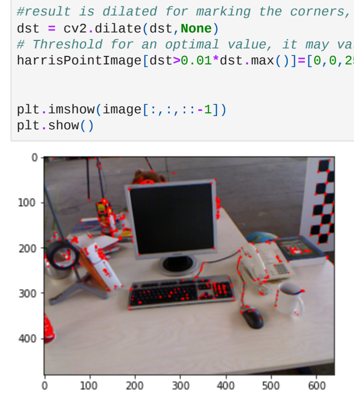
Programming Task (2 Pts)

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

In [2]:
```python
# Solution
image = get_image("img1.png")
plt.title("Orignal Image")
plt.imshow(image[:,:,::-1])
plt.show()


# Using openCV Canny Image Library
edges = cv2.Canny(image,100,200)

# Display the results of canny edge detection
plt.title("Canny edge detection")
plt.imshow(edges,cmap = 'gray')
plt.show()

# Laplacian edge detection
ddepth = cv2.CV_16S
kernel_size = 3
src_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
dst = cv2.Laplacian(src_gray, ddepth, ksize=kernel_size)
abs_dst = cv2.convertScaleAbs(dst)

# Display the results of Laplacian edge detection
plt.title("Laplacian edge detection")
plt.imshow(abs_dst,cmap = 'gray')
plt.show()

# Code ends here
```



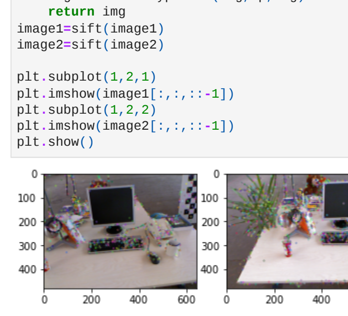




Programming Task (1 Pts)

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

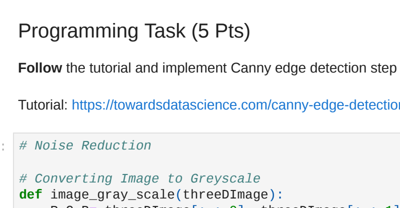
In [3]:
```python
# Solution
image = get_image("img1.png")


# Harris Point Detection
harrisPointImage = image
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)

#result is dilated for marking the corners, not important
dst = cv2.dilate(dst,None)
# Threshold for an optimal value, it may vary depending on the image.
harrisPointImage[dst>0.01*dst.max()]=[0,0,255]


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



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 [4]:
```python
# Solution
image1 = get_image("img1.png")
image2 = get_image("img2.png")

# SITF Function
def sift(img):
#     gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    kp = sift.detect(img,None)
    img=cv2.drawKeypoints(img,kp,img)
    return img
image1=sift(image1)
image2=sift(image2)

plt.subplot(1,2,1)
plt.imshow(image1[:,:,::-1])
plt.subplot(1,2,2)
plt.imshow(image2[:,:,::-1])
plt.show()
```



Explanation (1 Pts):

TODO Explain your visualization.

---

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:

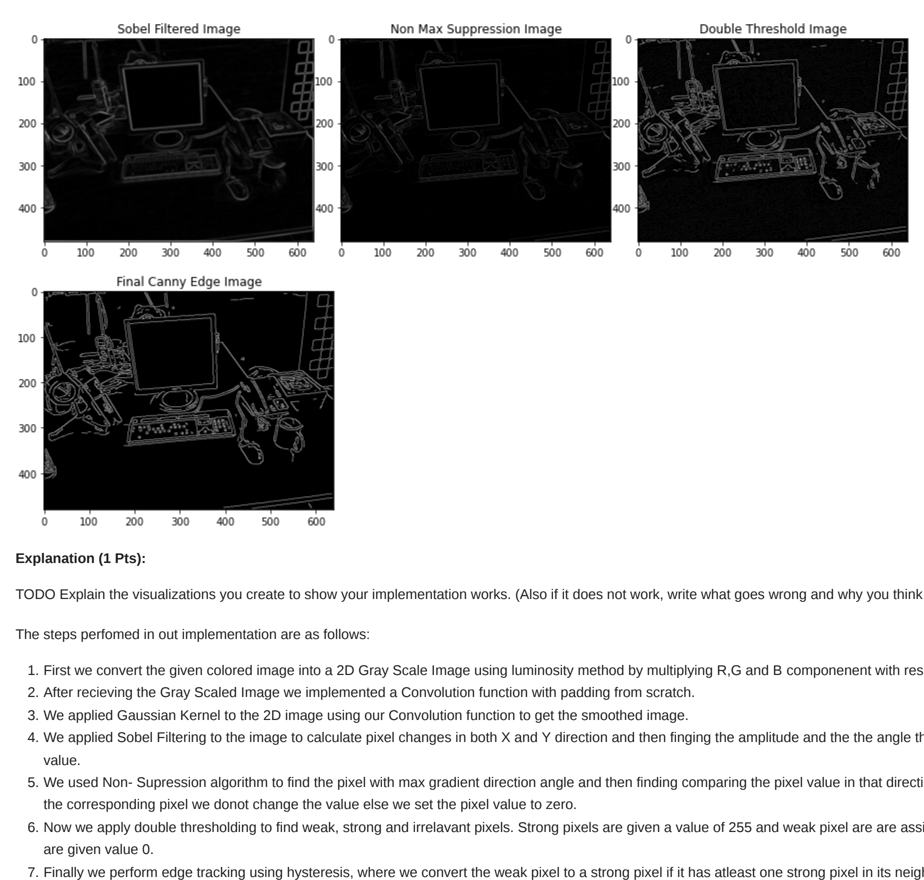
1. Noise reduction
2. Gradient calculation
3. Non-maximum suppression
4. Hysteresis thresholding
5. Edge Tracking by Hysteresis.

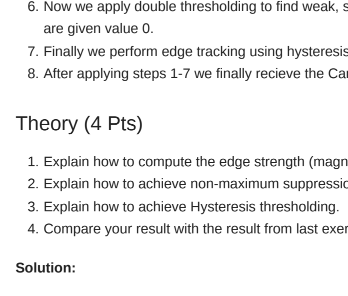
Programming Task (5 Pts)

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 [5]:
```python
# Noise Reduction

# Converting Image to Greyscale
def image_gray_scale(threeDImage):
    R,G,B= threeDImage[:,:,0], threeDImage[:,:,1],threeDImage[:,:,1]
    imgGray = 0.2989 * R + 0.5870 * G + 0.1140 * B  # using the luminosity method
    return imgGray

# Convolution
class convolve():
    def __init__(self,image,kernel,kernal_size):
        self.image=image
        self.kernal_size=kernal_size
        self.kernel=kernel
        self.padding= self.padding_required()
        self.paddedImage= self.pad_image()
        self.convolvedImage=self.convolveImage()


    def padding_required(self):
        return int(self.kernal_size//2)

    def pad_image(self):

        padded_image= np.zeros(shape=((self.image.shape[0]+2*self.padding),(self.image.shape[1]+2*self.padding)))
        padded_image[self.padding:-self.padding, self.padding:-self.padding] = self.image  # Keeping all the pixels of the image
        return padded_image

    def convolveImage(self):

        convolved_img = np.zeros(shape=(self.image.shape))
        k = self.kernel.shape[0]


        for i in range(self.image.shape[0]):
            for j in range(self.image.shape[1]):
                mat = self.paddedImage[i:i+k, j:j+k]
                convolved_img[i, j] = np.sum(np.multiply(mat, self.kernel))


        return convolved_img


# Gaussian Kernel
def gaussian_kernel(kernal_size, sigma=1):
        kernal_size = int(kernal_size) // 2  # for symmetry
        x, y = np.mgrid[-kernal_size:kernal_size+1, -kernal_size:kernal_size+1]   # Creating the x and y axis
        normal = 1 / (2.0 * np.pi * sigma**2)         # Denominator of the Gaussian
        g =  np.exp(-((x**2 + y**2) / (2.0*sigma**2))) * normal    # Gaussian distribution
        return g


def sobel_filters(img):
    filterXaxis = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)  # Filter for derivative along X Axis
    filterYaxis = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)  # Filter for derivative along Y Axis

    GradientAlongXaxis = convolve(img, filterXaxis,3).convolvedImage  # Convolving for Gradiennt along X axis with 3 as filter size
    GradientAlongYaxis = convolve(img, filterYaxis,3).convolvedImage  # Convolving for Gradiennt along Y axis with 3 as filter size


    Magnitude =(GradientAlongXaxis**2 + GradientAlongYaxis**2)**0.5   # finding the amplitude of the gradient vector.
    Magnitude = Magnitude/Magnitude.max() * 255                            # Assuring everything is in range of 0 to 255.
    direction  = np.arctan2(GradientAlongYaxis,GradientAlongXaxis)   #Finding the angle between x and y components of the gradient.

    return (Magnitude, direction)



# plt.imshow(image,cmap='gray')

def non_max_suppression(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(img, lowThresholdRatio=0.05, highThresholdRatio=0.09):

    highThreshold = img.max() * highThresholdRatio;
    lowThreshold = highThreshold * lowThresholdRatio;

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

    weak = np.int32(25)
    strong = np.int32(255)

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >= lowThreshold))

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

    return (res, weak, strong)

def hysteresis(img, weak, strong=255):
    M, N = img.shape
    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 hysteresis(img, weak, strong=255):
    M, N = img.shape
    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

# Subplots
fig, axs = plt.subplots(2,3,figsize=(15,15))
fig.tight_layout()

image = get_image("img1.png")
axs[0,0].imshow(image[:,:,::-1])
axs[0,0].set_title("Orignal Image")

# Converting Image to Gray Scale
gray_image = image_gray_scale(image)
axs[0,1].imshow(gray_image,cmap='gray')
axs[0,1].set_title("Grey Scaled Image")


# Step-1 Reducing Noise Convolving the image with gaussian Kernel
gaussianConvolvedImage=convolve(gray_image, gaussian_kernel(5, 2),5).convolvedImage
axs[0,2].imshow(gaussianConvolvedImage,cmap='gray')
axs[0,2].set_title("Gaussian Filered Image")

# Step-2 Applying the sobel filter for derivatives
sobelImage,theta= sobel_filters(gaussianConvolvedImage)
axs[1,0].imshow(sobelImage,cmap='gray')
axs[1,0].set_title("Sobel Filtered Image")

# Step-3 Applying the Non max Suppression
non_max_suppression_image = non_max_suppression(sobelImage,theta)
axs[1,1].imshow(non_max_suppression_image,cmap='gray')
axs[1,1].set_title("Non Max Suppression Image")

# Step-4 Applying Double Threshold
double_threshold_image,weak,strong = threshold(non_max_suppression_image)
axs[1,2].imshow(double_threshold_image,cmap='gray')
axs[1,2].set_title("Double Threshold Image")
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.1,
                    hspace=0.1)
plt.show()

# Step-5 Applying Hysteresis to obtain the final Canny Edge Image
cannyedgeImage=hysteresis(double_threshold_image,weak,strong)
plt.imshow(cannyedgeImage,cmap='gray')
plt.title("Final Canny Edge Image")
plt.show()
```





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)

The steps perfomed in out implementation are as follows:

1. First we convert the given colored image into a 2D Gray Scale Image using luminosity method by multiplying R,G and B componenent with respective values.
2. After recieving the Gray Scaled Image we implemented a Convolution function with padding from scratch.
3. We applied Gaussian Kernel to the 2D image using our Convolution function to get the smoothed image.
4. We applied Sobel Filtering to the image to calculate pixel changes in both X and Y direction and then finging the amplitude and the the angle theta for maximum change in pixel value.
5. We used Non- Supression algorithm to find the pixel with max gradient direction angle and then finding comparing the pixel value in that direction. If the pixel value is more than the corresponding pixel we donot change the value else we set the pixel value to zero.
6. Now we apply double thresholding to find weak, strong and irrelavant pixels. Strong pixels are given a value of 255 and weak pixel are are assigned value of 25. Irrelevant pixels are given value 0.
7. Finally we perform edge tracking using hysteresis, where we convert the weak pixel to a strong pixel if it has atleast one strong pixel in its neighbourhood.
8. After applying steps 1-7 we finally recieve the Canny Edge Image.

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. Edge magnitude is calculated by finding magnitude of the gradient using sqare root of the sum of sqare of gradients along x and y axis. Edge orientation is calculated by finding tan inverse of gradient along y axis divded by gradient along x axis
2. Non maximum suppression is achieved by finding the pixel values in the direction of maximum gradient shift and then comparing the pixel value in that direction. If the pixel value is more than the corresponding pixel we keep the pixel in the image else we set it to zero. Non-max suppression is used to check if the pixels on the same direction are more or less intense than the ones being processed.
3. Hysteris thresholding is achieved by first finding the weak, stong and irrelavant pixels by using a given threshold. Strong and weak pixels are then assiged a predefined value. After assiging the values we find the strong and week pixels in the neighbourhood of the given week pixel and then make it a strong pixel if it has any strong pixel in its neighbourhood.
4. The canny edge detector performs better in detecting the edges as it takes account of the value and direction of the gradients.