**ITMO University**

**Image Processing: Lab2**

**Prepared by**

Bassel Alshawareb

**April 11, 2023**

1. **Introduction**

Image transformations refer to the process of modifying the appearance of an image through various techniques. These techniques can be applied to a wide range of image types, including photographs, digital art, and graphics. Image transformations can be used to correct image imperfections, enhance specific features. In this lab we are reviewing some of the most common transformations used in image processing.

1. **Transformations**

In this lab we will consider applying image transformations on the following images:



Figure Images on which we will apply transformations. On the left is an image that contains coins, which have geometric shapes, and on the right is an image of woman in red dress..

Piecewise Linear Transformation:

****

Figure coins after applying Piecewise linear transformation.

Projection:

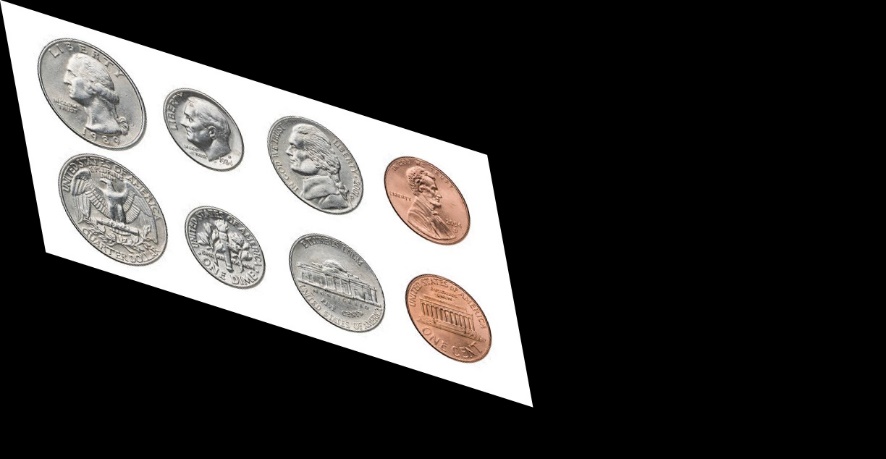
****

Figure coins after projection.

Polynomial:

****

Figure coins after applying polynomial transformation.

Sinusoidal:

****

Figure 5 Woman in the red dress after applying sinusoidal transformation

1. **Removing Barrel and binocular distortions**

We start by getting distorted images. To do that, we apply barrel and binocular transformations.



Figure A) Coins after applying Barrel. B) Coins after applying Binocular.

Now we remove the Barrel by applying binocular transformation



Figure 7 The coins image after applying barrel followed by binocular distortions.

1. **Stitching**

Image stitching or photo stitching is the process of combining multiple [photographic](https://en.wikipedia.org/wiki/Photograph) [images](https://en.wikipedia.org/wiki/Image) with overlapping fields of view to produce a segmented [panorama](https://en.wikipedia.org/wiki/Panorama) or high-resolution image.



Figure Two cropped parts from the same image.

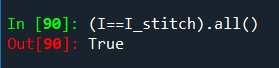


Figure The result of stitching the 2 previous images. We can realize that the output image is identical to the original image.

**Questions:**

1. One way to rotate an image without using a rotation matrix is to perform a series of translations and shears on the image to achieve the desired rotation. Another way in the case of requiring special degrees of rotations, like for example, a rotation 180 degrees. In such cases, we can perform the rotation simply by reversing the coordinates of the pixels along x and/or y axes.
2. If the transformation order 𝑛 = 4, at least four corresponding pairs of points must be specified on the original and distorted images. This is because a fourth-order transformation requires four control points to uniquely determine the transformation parameters.
3. The reason of having undefined values in the image after transformation, is because of the resulting float values from the transformation, or values outside of image boundaries. One way to deal with this problem is to take the median of the neighboring values. Another way is to average all neighboring pixels. In both cases, a parameter should be defined which is the size of the window, in which we will look for neighbors.

**Code:**

Code from file Histogram Processing

# -\*- coding: utf-8 -\*-

"""

Created on Sat Mar 25 13:21:23 2023

@author: Bassel

"""

import cv2

import numpy as np

import matplotlib.pyplot as plt

import os

class myHist:

id\_plot=0

def \_\_init\_\_(self, img=None,histSize=256, histRange=(0,256), CONFIG="BGR", EQUALIZE = True, pth = "default"):

self.img\_org = None

self.history = []

if pth == "default":

path = os.getcwd()

else:

path = pth

myHist.path\_input = os.path.join(path,"inputs")

myHist.path\_output = os.path.join(path , "outputs")

self.last\_executed = ""

self.histSize=256

self.EQUALIZE = EQUALIZE

if CONFIG=="BGR":

a=[0,1,2]

else:

a=[2,1,0]

self.order=a

myHist.histSize=histSize

myHist.histRange=histRange

if img is not None:

self.set\_img(img)

else:

self.img=None

def get\_img(self):

return self.img

def set\_img(self, img, text = None):

if img is None:

return

if text is not None:

print(text)

if self.img\_org is None:

print("added origin")

self.img\_org = img

self.img = img

self.rows, self.cols = self.img.shape[0:2]

self.history.append(self.img)

self.calc()

def calc(self,img=None):

print("Calculating Histogram")

if img is None:

if self.img is None:

print("error")

return

img=self.img

img\_s=cv2.split(img)

bHist=cv2.calcHist(img\_s,[0],None, [256], (0, 256))

gHist=cv2.calcHist(img\_s,[self.order[1]],None, [self.histSize], (0, 256))

rHist=cv2.calcHist(img\_s,[self.order[2]],None, [self.histSize], (0, 256))

self.img=img

self.bH=bHist

self.gH=gHist

self.rH=rHist

if self.EQUALIZE:

self.equalize()

def equalize(self):

if self.last\_executed == "":

self.last\_executed = "equalized"

print("Equalizing")

self.bH\_not\_normalized = self.bH

self.gH\_not\_normalized = self.gH

self.rH\_not\_normalized = self.rH

max\_b = np.max(self.bH)

max\_g = np.max(self.gH)

max\_r = np.max(self.rH)

self.bH = self.bH/max\_b

self.gH = self.gH/max\_g

self.rH = self.rH/max\_r

self.EQUALIZE = True

def show(self, image = "current", name=None):

if image == "org":

I = self.img\_org

else:

I = self.img

if name is None:

name="number"+str(myHist.id\_plot)

image\_name = name + "\_" + self.last\_executed

image\_path = myHist.path\_output + "/" + "images"

hist\_path = myHist.path\_output + "/" + "Histograms"

try:

os.mkdir(image\_path)

except IOError:

pass

try:

os.mkdir(hist\_path)

except IOError:

pass

myHist.id\_plot=myHist.id\_plot+1

fig = plt.figure(myHist.id\_plot)

t=range(256)

plt.plot(t,self.bH, color="blue")

plt.plot(t,self.gH, color="green")

plt.plot(t,self.rH, color="red")

plt.suptitle(name)

plt.savefig(hist\_path + "/" + image\_name + "\_Histogram" + ".png")

plt.show()

self.\_\_show\_img(I)

plt.imsave(image\_path + "/" + image\_name + ".jpg", I)

def \_\_show\_img(self, I):

I = cv2.cvtColor(I, cv2.COLOR\_BGR2RGB)

myHist.id\_plot=myHist.id\_plot+1

plt.figure(myHist.id\_plot)

plt.imshow(I)

plt.show()

def show\_original(self):

self.show(image = "org")

def show\_history(self):

myHist.id\_plot=myHist.id\_plot+1

for img in self.history:

self.\_\_show\_img(img)

plt.show()

def profile(img, x):

return img[x,:]

def project\_(img,xy):

return np.sum(img,xy)/(img.shape[(xy+1)%2])

if \_\_name\_\_ == "\_\_main\_\_":

pass

Code from file transformations

# -\*- coding: utf-8 -\*-

"""

Created on Thu Mar 30 16:44:51 2023

@author: Bassel

"""

import cv2

import numpy as np

import os

from scipy.optimize import fsolve

import Histogram\_processing

class transformation(Histogram\_processing.myHist):

def \_\_init\_\_(self, img=None,histSize=256, histRange=(0,256), CONFIG="BGR", EQUALIZE = True, pth = "default", on\_origin = True ):

super().\_\_init\_\_(img ,histSize, histRange, CONFIG, EQUALIZE, pth)

self.on\_origin = on\_origin

print("origin bool", self.on\_origin)

def \_\_copy\_img(self):

if self.on\_origin == True:

return self.img\_org.copy()

else:

return self.img.copy()

def shift(self, amount=50):

I = self.\_\_copy\_img()

if I is None:

print("error")

return

I1=I

I1[:,:,0]=np.clip(I1[:,:,0].astype(np.int16)+amount,0,255).astype(np.uint8)

I1[:,:,1]=np.clip(I1[:,:,1].astype(np.int16)+amount,0,255).astype(np.uint8)

I1[:,:,2]=np.clip(I1[:,:,2].astype(np.int16)+amount,0,255).astype(np.uint8)

self.set\_img(I1)

def \_\_filter\_high\_frequencies(self, H):

thresholded = np.where(H<= 10\*\*(-2),10, H) #Filtering out small frequencies

i\_min = 0

print("length of H ", thresholded.shape[0])

for i in range(thresholded.shape[0]):

if thresholded[i] !=10:

i\_min = i

break

i\_max = 255

for i in range(thresholded.shape[0]):

if thresholded[-1-i] !=10:

i\_max = 255-i

break

return float(i\_min)/255, float(i\_max)/255

def extend(self, alpha = 0.5, REMOVE\_LOW\_FREQUENCY = True):

if REMOVE\_LOW\_FREQUENCY:

self.last\_executed = "Extended"

else:

s\_alpha = str(alpha)

list\_s\_alpha = s\_alpha.split('.')

alpha\_for\_writing = '\_'.join(list\_s\_alpha)

self.last\_executed = "Extended\_with\_Alpha" + alpha\_for\_writing

self.alpha = alpha

I\_temp = self.\_\_copy\_img()

I = I\_temp.astype(np.float64)/255

Ib = I[:,:,0]

Ig = I[:,:,1]

Ir = I[:,:,2]

Iout = []

if self.EQUALIZE:

if REMOVE\_LOW\_FREQUENCY:

#removing low frequicies

Ib\_min, Ib\_max = self.\_\_filter\_high\_frequencies(self.bH)

Ig\_min, Ig\_max = self.\_\_filter\_high\_frequencies(self.gH)

Ir\_min, Ir\_max = self.\_\_filter\_high\_frequencies(self.rH)

else:

Ib\_min, Ib\_max = np.min(Ib), np.max(Ib)

Ig\_min, Ig\_max = np.min(Ig), np.max(Ig)

Ir\_min, Ir\_max = np.min(Ir), np.max(Ir)

#Extend b

Ib\_extended = ( np.clip((255\*((Ib-Ib\_min)/(Ib\_max - Ib\_min))\*\*alpha),0,255) ).astype(np.uint8)

Iout.append(Ib\_extended)

#Extend g

Ig\_extended = ( np.clip((255\*((Ig-Ig\_min)/(Ig\_max - Ig\_min))\*\*alpha),0,255) ).astype(np.uint8)

Iout.append(Ig\_extended)

#Extend r

Ir\_extended = ( np.clip((255\*((Ir-Ir\_min)/(Ir\_max - Ir\_min))\*\*alpha),0,255) ).astype(np.uint8)

Iout.append(Ir\_extended)

Iout = cv2.merge(Iout)

self.set\_img(Iout, text = "Extend is done")

def rotate(self, theta = 90):

I = self.\_\_copy\_img()

phi = theta \* np.pi / 180

T1 = np.float32(

[[1, 0, -(self.cols - 1) / 2.0],

[0, 1, -(self.rows - 1) / 2.0],

[0, 0, 1]])

T2 = np.float32(

[[np.cos(phi), -np.sin(phi), 0],

[np.sin(phi), np.cos(phi), 0],

[0, 0, 1]])

T3 = np.float32(

[[1, 0, (self.cols - 1) / 2.0],

[0, 1, (self.rows - 1) / 2.0],

[0, 0, 1]])

T = np.matmul(T3, np.matmul(T2, T1))[0:2, :]

I\_rotate = cv2.warpAffine(I, T, (np.max(I.shape), np.max(I.shape)))

self.set\_img(I\_rotate, text = "rotated")

def sinusoid(self):

I = self.\_\_copy\_img()

u, v = np. meshgrid (np. arange ( self.cols ), np. arange ( self.rows ))

u = u + 20 \* np.sin (2 \* np.pi \* v / 90)

I\_sinusoid = cv2 . remap (I, u. astype (np. float32 ), v. astype (np. float32 ), cv2. INTER\_LINEAR )

self.set\_img(I\_sinusoid)

def piecewise(self):

I = self.\_\_copy\_img()

stch=2

T = np.float32([[stch, 0, 0], [0, 1, 0]])

I\_piecewiselinear = I.copy()

I\_piecewiselinear[:, int(self.cols/2):, :] = cv2.warpAffine(I\_piecewiselinear[:, int(self.cols/2):, :], T, (self.cols - int(self.cols/2), self.rows))

self.set\_img(I\_piecewiselinear)

def projection(self):

I = self.\_\_copy\_img()

T = np. float32 ([[1.1 , 0.2 , 0.00075] ,[0.35 , 1.1 , 0.0005] ,[0, 0, 1]])

I\_projective = cv2 . warpPerspective (I, T,(2\*self.cols , 2\*self.rows ))

self.set\_img(I\_projective)

def barrel(self):

I = self.\_\_copy\_img()

xi , yi = np. meshgrid (np. arange ( self.cols ), np. arange ( self.rows ))

midx=self.cols/2

midy=self.rows/2

xi=xi-midx

yi=yi-midy

r, theta = cv2.cartToPolar(xi/midx, yi/midy)

F3 = 0.4

F5 =0

r = r + F3 \* r\*\*3 + F5 \* r\*\*5

u, v = cv2.polarToCart(r, theta)

u = u \* midx + midx

v = v \* midy + midy

I\_barrel = cv2.remap(I, u.astype(np.float32), v.astype(np.float32), cv2.INTER\_LINEAR)

self.set\_img(I\_barrel)

def debarrel(self):

I = self.\_\_copy\_img()

xi , yi = np. meshgrid (np. arange ( self.cols ), np. arange ( self.rows ))

midx=self.cols/2

midy=self.rows/2

xi=xi-midx

yi=yi-midy

r, theta = cv2.cartToPolar(xi/midx, yi/midy)

F3 = 0.17

F5 =0

r = r -F3 \* r\*\*3 -F5 \* r\*\*5

u, v = cv2.polarToCart(r, theta)

u = u \* midx + midx

v = v \* midy + midy

I\_debarrel = cv2.remap(I, u.astype(np.float32), v.astype(np.float32), cv2.INTER\_LINEAR)

self.set\_img(I\_debarrel)

def poly(self):

I = self.\_\_copy\_img()

T = np.array([[0, 0], [1, 0], [0, 1], [0.00001, 0], [0.002, 0], [0.001, 0]])

I\_polynomial = np.zeros(I.shape, I.dtype)

x, y = np.meshgrid(np.arange(self.cols), np.arange(self.rows))

xnew = np.round(T[0, 0] + x \* T[1, 0] + y \* T[2, 0] + x \* x \* T[3, 0] + x \* y \* T[4, 0] + y \* y \* T[5, 0]).astype(np.float32)

ynew = np.round(T[0, 1] + x \* T[1, 1] + y \* T[2, 1] + x \* x \* T[3, 1] + x \* y \* T[4, 1] + y \* y \* T[5, 1]).astype(np.float32)

mask = np.logical\_and(np.logical\_and(xnew >= 0, xnew < self.cols), np.logical\_and(ynew >= 0, ynew < self.rows))

if I.ndim == 2:

I\_polynomial[ynew[mask].astype(int), xnew[mask].astype(int)] = I[y[mask], x[mask]]

else:

I\_polynomial [ ynew [ mask ]. astype (int), xnew [ mask ]. astype (int ), :] =I [y[ mask ], x[ mask ], :]

self.set\_img(I\_polynomial)

def stitching(self):

I = self.\_\_copy\_img()

I\_top=I[:int(self.rows/2)+100,:,:]

self.set\_img(I\_top)

I\_bottom=I[int(self.rows/2):,:,:]

self.set\_img(I\_bottom)

templ\_size = 10

templ = I\_top[-templ\_size:, :, :]

res = cv2.matchTemplate(I\_bottom, templ, cv2.TM\_CCOEFF)

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(res)

I\_stitch = np.zeros((I\_top.shape[0] + I\_bottom.shape[0] - max\_loc[1] - templ\_size, I\_top.shape[1], I\_top.shape[2]), dtype=np.uint8)

I\_stitch[0:I\_top.shape[0], :, :] = I\_top

I\_stitch[I\_top.shape[0]:, :, :] = I\_bottom[max\_loc[1] + templ\_size:, :, :]

self.set\_img(I\_stitch)

if \_\_name\_\_ == "\_\_main\_\_":

#Demo

path = os.getcwd()

path\_input = os.path.join(path,"inputs")

path\_output = os.path.join(path , "outputs")

I=cv2.imread(path\_input + '/dark\_sky.jpg')

I=cv2.resize(I,(500,500))

img=I.copy()

img=cv2.resize(img,(500,500))

ob = transformation(on\_origin = True, img = I)

ob.rotate(45)

ob.extend()

ob.shift()

ob.sinusoid()

ob.piecewise()

ob.poly()

ob.barrel()

ob.debarrel()

ob.stitching()

ob.show\_history()