MODERN ACADEMY

FOR ENGINEERING & TECHNOLOGY

Computer Department

Academic Year 2021/2022

December 2022

Toolbox Documentation

Course Title: Digital image processing

Course Code: CMPN332

Name	**Mohamed Magdy Mohamed**
Section	**1**
ID	**4190827**
Dr	**Sabry Mohamed Abdelmoaty**

Content:

First Installing the program:

1. These is the link of python to install first

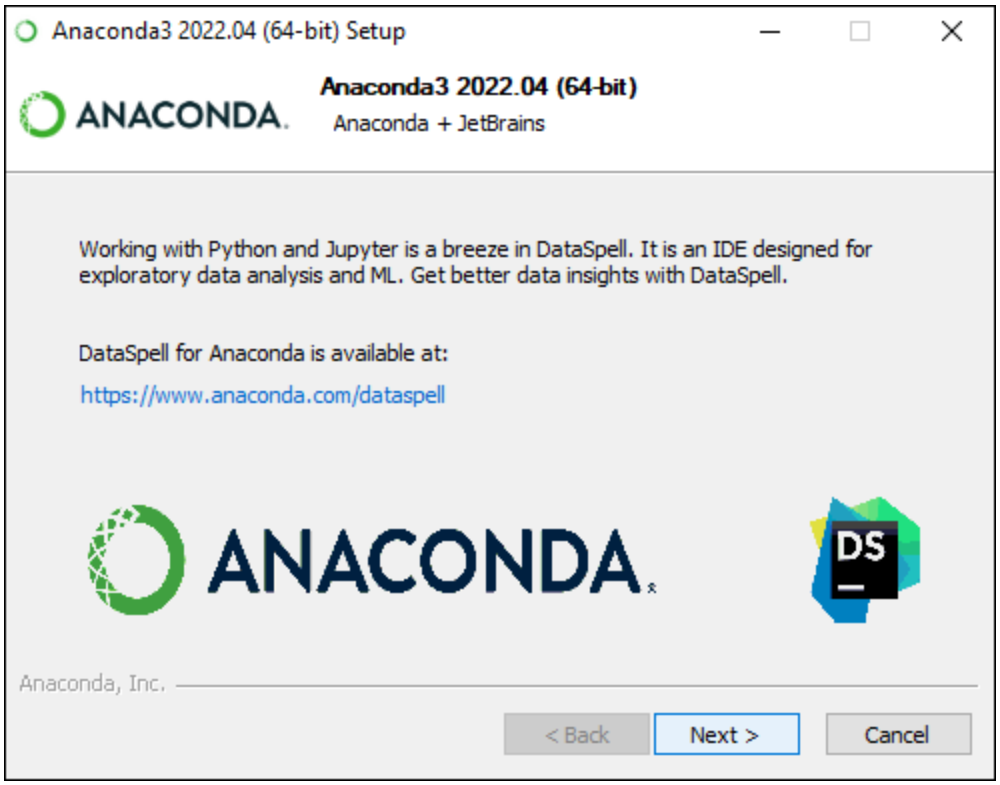
   https://www.python.org/ftp/python/3.11.1/python-3.11.1-amd64.exe

2. Then setup program by clicking next to begin setup process.

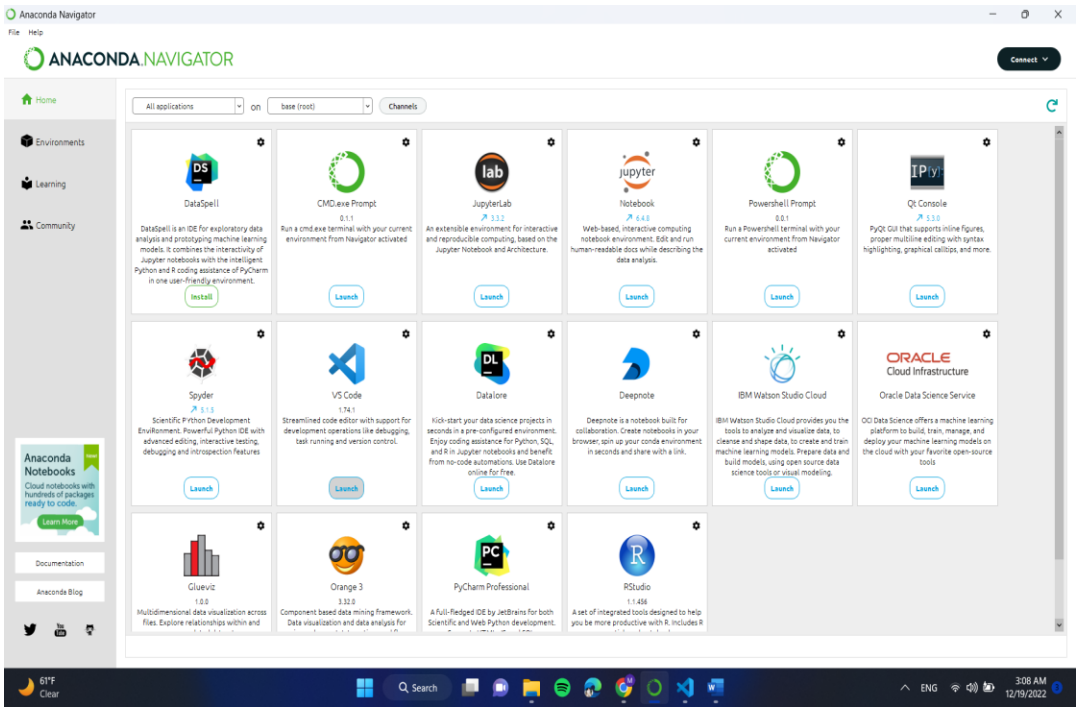
3. Install anaconda navigator and there is the link

   https://repo.anaconda.com/archive/Anaconda3-2022.10-Windows-x86_64.exe

4. Then setup anaconda by clicking next until start program



5. Install visual studio code
   https://az764295.vo.msecnd.net/stable/1ad8d514439d5077d2b0b7ee64d2ce82a9308e5a/VSCodeUserSetup-x64-1.74.1.exe
6. Then setup
7. Open visual studio code from anaconda by clicking on launch visual studio code



8. Click connect in Anaconda to download all libraries and updates

Second initializing the program:

```
import statistics
from tkinter import DISABLED, HORIZONTAL, NORMAL, NW, Entry, Button, Canvas,
Label, Scale, StringVar, Text, messagebox , Tk, filedialog, PhotoImage, RAISED
import tkinter as tk
from tkinter.font import BOLD
from tkinter.tix import *
import cv2
import PIL.Image, PIL.ImageTk
from tkinter.messagebox import showinfo
from tkinter.ttk import Combobox
import numpy as np
from numpy import asarray
import os
from matplotlib import pyplot as plt
```

Third Source Code:

```
def flipingHeFunction():
    global image
    global photo
    arr = np.zeros((image.shape[0],image.shape[1]))
    for row in range(image.shape[0]) :
        for col in range(image.shape[1]):
            arr[row,col] = image[row,(image.shape[1]-col-1)]

    for row in range(image.shape[0]) :
        for col in range(image.shape[1]):
            image[row,col] = arr[row,col]
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def flipingVeFunction():
    global image
    global photo
    arr = np.zeros((image.shape[0],image.shape[1]))
    for row in range(image.shape[0]) :
        for col in range(image.shape[1]):
            arr[row,col] = image[(image.shape[0]-row-1),col]

    for row in range(image.shape[0]) :
        for col in range(image.shape[1]):
            image[row,col] = arr[row,col]
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

# Rotate Function

def rotBtn():
    global image
    global photo
    image = cv2.rotate(image,cv2.ROTATE_90_CLOCKWISE)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

# Translate UP Function

def trUpBtn():
    global image
    global photo
```

```
    M = np.float32([[1,0,0],[0,1,10]])
    image =cv2.warpAffine(image,M,(image.shape[1],image.shape[0]))
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0 , 0 ,image=photo ,anchor=NW)

# Translation Dwon Function
def trDwonBtn():
    global image
    global photo
    M = np.float32([[1,0,0],[0,1,-10]])
    image =cv2.warpAffine(image,M,(image.shape[1],image.shape[0]))
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0 , 0 ,image=photo ,anchor=NW)

# TranslateLeft
def details():
    global image
    global photo
    print("img", image)
    smallest = np.amin(image)
    biggest = np.amax(image)
    print("smallest", smallest)
    print("biggest", biggest)
    avarage = np.average(image)
    print("avarage", avarage)

    image = asarray(image)
    orignalImg = image
    photo = PIL.ImageTk.PhotoImage(image=PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def Tleft():
    global image
    global photo
    M = np.float32([[1, 0, 10],[0, 1, 0]])
    image = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

#TranslateRight
def Treight():
    global image
    global photo
    M = np.float32([[1, 0, -10],[0, 1, 0]])
    image = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
```

```
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

#Affine transformation

def affineTransform(val):
    global image
    global photo
    point_1 = np.float32([[0, 0], [0, image.shape[1]], [image.shape[0],
image.shape[1]]])
    point_2 = np.float32([[val, 0], [0, image.shape[1]], [image.shape[0],
image.shape[1]]])
    M = cv2.getAffineTransform(point_1, point_2)
    image = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

# searing
def Sharping1():
    global image
    global photo
    kernal_shearing = np.array([[-1, -1, -1],[-1, 9, -1],[-1, -1, -1]])
    image = cv2.filter2D(image, -1, kernal_shearing)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def Sharping2():
    global image
    global photo
    kernal_shearing = np.array([[1, 1, 1],[1, -7, 1],[1, 1, 1]])
    image = cv2.filter2D(image, -1, kernal_shearing)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def Sharping3():
    global image
    global photo
    kernal_shearing = np.array([[-1,-1,-1,-1,-1],
                                [-1,2,2,2,-1],
                                [-1,2,8,2,-1],
                                [-1,2,2,2,-1],
                                [-1,-1,-1,-1,-1]]) / 8.0
    image = cv2.filter2D(image, -1, kernal_shearing)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)
```

```
def scaleOfMotion(val):
    global image
    global photo
    size = motion_blur_scale.get()
    kernal_motion_blur = np.zeros((size,size))
    kernal_motion_blur[int((size-1)/2), :] = np.ones(size)
    kernal_motion_blur = kernal_motion_blur/size
    image = cv2.filter2D(image, -1, kernal_motion_blur)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def TransformCombobox(event):
    global image
    global photo
    transform = event.widget.get()
    if transform == 'Dilation':
        binr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
        kernel = np.ones((3, 3), np.uint8)
        invert = cv2.bitwise_not(binr)
        image = cv2.dilate(invert, kernel, iterations=1)
    elif transform == 'Erosion':
        binr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
        kernel = np.ones((5, 5), np.uint8)
        invert = cv2.bitwise_not(binr)
        image = cv2.erode(invert,kernel,iterations=1)
    elif transform == 'Opening':
        binr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
        kernel = np.ones((3, 3), np.uint8)
        image = cv2.morphologyEx(binr, cv2.MORPH_OPEN, kernel, iterations=1)
    elif transform == 'Closing':
        binr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
        kernel = np.ones((3, 3), np.uint8)
        image = cv2.morphologyEx(binr, cv2.MORPH_CLOSE, kernel, iterations=1)
    elif transform == 'Gradient':
        binr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
        kernel = np.ones((3, 3), np.uint8)
        invert = cv2.bitwise_not(binr)
        image = cv2.morphologyEx(invert, cv2.MORPH_GRADIENT ,kernel)
    elif transform == 'Top Hat':
        binr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
        kernel = np.ones((13, 13), np.uint8)
        image = cv2.morphologyEx(binr, cv2.MORPH_TOPHAT ,kernel)
    elif transform == 'Black Hat':
        binr = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
```

```
        kernel = np.ones((5, 5), np.uint8)
        invert = cv2.bitwise_not(binr)
        image = cv2.morphologyEx(invert, cv2.MORPH_BLACKHAT ,kernel)

    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def FilterCombobox(event):
    global image
    global photo
    filters = event.widget.get()
    if filters == 'Gray-scale':
        motion_blur_scale['state'] = DISABLED
        disableShearingBtn()
        image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    elif filters == 'Blur 3X3':
        motion_blur_scale['state'] = DISABLED
        disableShearingBtn()
        kernal_3X3 = np.ones((3,3), np.float32) / 9
        image = cv2.filter2D(image, -1, kernal_3X3)
    elif filters == 'Blur 5X5':
        motion_blur_scale['state'] = DISABLED
        disableShearingBtn()
        kernal_5X5 = np.ones((5,5), np.float32) / 25
        image = cv2.filter2D(image, -1, kernal_5X5)
    elif filters == 'None':
        motion_blur_scale['state'] = DISABLED
        disableShearingBtn()
        kernal_identifiy = np.array([[0, 0, 0],[0, 1, 0],[0, 0, 0]])
        image = cv2.filter2D(image, -1, kernal_identifiy)
    elif filters == 'Motion-blur':
        motion_blur_scale['state'] = NORMAL
        disableShearingBtn()
    elif filters == 'sharping':
        motion_blur_scale['state'] = DISABLED
        searingBt1['state'] = NORMAL
        searingBt2['state'] = NORMAL
        searingBt3['state'] = NORMAL
    elif filters == 'Gaussian blur':
        motion_blur_scale['state'] = DISABLED
        disableShearingBtn()
        image = cv2.GaussianBlur(image,(9,9),0)
    elif filters == 'Median blur':
        motion_blur_scale['state'] = DISABLED
        disableShearingBtn()
```

```
        image = cv2.medianBlur(image,9)
    elif filters == 'Bilateral filter':
        motion_blur_scale['state'] = DISABLED
        disableShearingBtn()
        image = cv2.bilateralFilter(image,10,25,25)

    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def disableShearingBtn():
    searingBt1['state'] = DISABLED
    searingBt2['state'] = DISABLED
    searingBt3['state'] = DISABLED

def edgeDetectHor():
    global image
    global photo
    kernal = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
    image = cv2.filter2D(image, -1, kernal)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def edgeDetectVer():
    global image
    global photo
    kernal = np.array([[1, 2, 1],[0, 0, 0],[-1, -2, -1]])
    image = cv2.filter2D(image, -1, kernal)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def edgeDetectLaplacian():
    global image
    global photo
    image = cv2.Laplacian(image,cv2.CV_64F)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def edgeDetectCanny():
    global image
    global photo
    if(minInput.get() == '' or maxInput.get() == ''):
        messagebox.showwarning(title="Warning", message="Enter Min Value or Max
Value of threshold")
    elif(float(maxInput.get()) <= float(minInput.get())):
```

```
        messagebox.showwarning(title="Warning", message="The max Threshold must
bigger then min threshold")
    else:
        image = cv2.Canny(image, float(minInput.get()), float(maxInput.get()))
        photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
        canvas.create_image(0, 0, image=photo, anchor=NW)

def NegativeFunction():
    global image
    global photo
    for row in range(image.shape[0]):
        for col in range(image.shape[1]):
            image[row,col] =  255 - image[row,col]
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def mergeingImages():
    global photo2
    global image
    global photo
    arr = np.zeros((image.shape[0],image.shape[1]))
    fln2 = filedialog.askopenfilename(initialdir=os.getcwd(),title="Select
image", filetypes=(("JPG File","*.jpg"),("PNG File","*.png"),("All
Files","*.*")))
    image2 = PIL.Image.open(fln2)
    image2 = asarray(image2)
    if image2.shape[0] > 50 or image2.shape[1] > 50:
        dim2 = (50,50)
        image2 = cv2.resize(image2,dim2, interpolation = cv2.INTER_AREA)
    photo2 = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image2))
    if len(image.shape) == 3:
        image = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
    image2 = cv2.cvtColor(image2,cv2.COLOR_RGB2GRAY)
    dim = (image.shape[1],image.shape[0])
    image2 = cv2.resize(image2,dim, interpolation = cv2.INTER_AREA)
    for row in range(image.shape[0]):
        for col in range(image.shape[1]):
            arr[row,col] = image[row,col] * 0.8 + image2[row,col] * 0.2
    image = arr
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)


def resets():
    global image
```

```
    global orignalImg
    global finalEdit
    global photo
    image = orignalImg
    finalEdit = orignalImg
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def uploadImage():
    global image
    global photo
    global orignalImg
    global finalEdit
    fln = filedialog.askopenfilename(initialdir=os.getcwd(),title="Select image",
filetypes=(("JPG File","*.jpg"),("PNG File","*.png"),("All Files","*.*")))
    image = PIL.Image.open(fln)
    image = asarray(image)
    if image.shape[0] > 700 or image.shape[1] > 600:
        dim = (700,600)
        image = cv2.resize(image,dim, interpolation = cv2.INTER_AREA)
    orignalImg = image
    finalEdit = image
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)


def equalizeHist():
    global image
    global photo
    if(len(image.shape)<3):
        image = cv2.equalizeHist(image)
    elif len(image.shape)==3:
      image = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
      image[:,:,0] = cv2.equalizeHist(image[:,:,0])
      image = cv2.cvtColor(image, cv2.COLOR_YUV2BGR)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def hisPlot():
    global image
    plt.hist(image.ravel(),256,[0,256],color ='tab:green')
    plt.show()

def Saving():
    global finalEdit
```

```
    global image
    global photo
    finalEdit = image
    cv2.imwrite('Edit.jpg',finalEdit)
    messagebox.showinfo(title="Saving process", message="Saving Done Correctly")
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def restoreBtn():
    global image
    global photo
    finalEdit = cv2.imread("Edit.jpg",0)
    image = finalEdit
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)


def thresholdBtn():
    global finalEdit
    global image
    global photo
    if(minInput.get() == '' or maxInput.get() == ''):
        messagebox.showwarning(title="Warning", message="Enter Min Value or Max
Value of threshold")
    elif(float(maxInput.get()) <= float(minInput.get())):
        messagebox.showwarning(title="Warning", message="The max Threshold must
bigger then min threshold")
    else:
        ret1, thresh1 = cv2.threshold(image, float(minInput.get()),
float(maxInput.get()), cv2.THRESH_BINARY_INV)
        image = thresh1
        photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
        canvas.create_image(0, 0, image=photo, anchor=NW)


def bitPlaneCombobox(even):
    global image
    global photo
    val = even.widget.get()
    lst = []
    for row in range(image.shape[0]):
        for col in range(image.shape[1]):
            lst.append(np.binary_repr(image[row][col] ,width=8))
    if val == '128':
```

```
        image = (np.array([int(i[0]) for i in lst],dtype = np.uint8) *
128).reshape(image.shape[0],image.shape[1])
    elif val == '64':
        image = (np.array([int(i[1]) for i in lst],dtype = np.uint8) *
64).reshape(image.shape[0],image.shape[1])
    elif val == '32':
        image = (np.array([int(i[2]) for i in lst],dtype = np.uint8) *
32).reshape(image.shape[0],image.shape[1])
    elif val == '16':
        image = (np.array([int(i[3]) for i in lst],dtype = np.uint8) *
16).reshape(image.shape[0],image.shape[1])
    elif val == '8':
        image = (np.array([int(i[4]) for i in lst],dtype = np.uint8) *
8).reshape(image.shape[0],image.shape[1])
    elif val == '4':
        image = (np.array([int(i[5]) for i in lst],dtype = np.uint8) *
4).reshape(image.shape[0],image.shape[1])
    elif val == '2':
        image = (np.array([int(i[6]) for i in lst],dtype = np.uint8) *
2).reshape(image.shape[0],image.shape[1])
    elif val == '1':
        image = (np.array([int(i[7]) for i in lst],dtype = np.uint8) *
1).reshape(image.shape[0],image.shape[1])
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def logBtn():
    global image
    global photo
    c = 255 / np.log( 1 + np.max(image))
    logImage = c * (np.log(image + 1))
    image = np.array(logImage, dtype=np.uint8)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def contourBtn():
    global image
    global photo
    if(minInput.get() == '' or maxInput.get() == ''):
        m = statistics.mean(image.ravel())
        sd = statistics.stdev(image.ravel())
        ret1, thresh1 = cv2.threshold(image, m-sd, m+sd, cv2.THRESH_BINARY)
        contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
        cv2.drawContours(image, contours, -1,color=(0,255,0),thickness=3)
        print("the number of object is:",str(len(contours)))
        photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
        canvas.create_image(0, 0, image=photo, anchor=NW)

    elif(float(maxInput.get()) <= float(minInput.get())):
        messagebox.showwarning(title="Warning", message="The max Threshold must
bigger then min threshold")
    else:
        ret1, thresh1 = cv2.threshold(image, float(minInput.get()),
float(maxInput.get()), cv2.THRESH_BINARY)
        contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        cv2.drawContours(image, contours, -1,color=(0,255,0),thickness=3)

        print("the number of object is:",str(len(contours)))
        photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
        canvas.create_image(0, 0, image=photo, anchor=NW)


def matchBtnfun():
    global image
    global photo
    fln2 = filedialog.askopenfilename(initialdir=os.getcwd(),title="Select
image", filetypes=(("JPG File","*.jpg"),("PNG File","*.png"),("All
Files","*.*")))
    image2 = PIL.Image.open(fln2)
    image2 = asarray(image2)
    orb = cv2.ORB_create()
    keypoints1, descriptors1 = orb.detectAndCompute(image,None)
    keypoints2, descriptors2 = orb.detectAndCompute(image2,None)

    bf_matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

    bf_matches = bf_matcher.match(descriptors1, descriptors2)

    bf_matches = sorted(bf_matches, key=lambda x: x.distance)

    result = cv2.drawMatches(image, keypoints1, image2, keypoints2,
bf_matches[:50], None)

    cv2.imshow("Match Points",result)
    cv2.waitKey(0)
```

```
def powerCombobox(even):
    global image
    global photo
    val = even.widget.get()
    print(val)
    if val == '0.1':
        arr = np.array(255*(image/255)**0.1,dtype='uint8')
    elif val == '0.5':
        arr = np.array(255*(image/255)**0.5,dtype='uint8')
    elif val == '1.2':
        arr = np.array(255*(image/255)**1.2,dtype='uint8')
    elif val == '2.2':
        arr = np.array(255*(image/255)**2.2,dtype='uint8')
    cv2.normalize(arr,arr,0,255,cv2.NORM_MINMAX)
    cv2.convertScaleAbs(arr,arr)
    image = arr
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)
def digonalSobel():
    global image
    global photo
    kernal = np.array([[2,1,0],[1,0,-1],[0,-1,-2]])
    image = cv2.filter2D(image, -1 , kernal)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def FullsobelFun():
    global image
    global photo
    kernal_1 = np.array([[1,0,-1],[2,0,-2],[1,0,-1]])
    kernal_2 = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
    kernal_3 = np.array([[2,1,0],[1,0,-1],[0,-1,-2]])
    vertical = cv2.filter2D(image, -1 , kernal_1)
    herizontal = cv2.filter2D(image, -1 , kernal_2)
    digonal = cv2.filter2D(image, -1 , kernal_3)
    dst_1 = cv2.addWeighted(vertical,1,herizontal,1,0.0)
    image = cv2.addWeighted(dst_1,1,digonal,1,0.0)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)


def grayMethod():
    global image
    global photo
    if minInputGray.get() == '' or maxInputGray.get() == '':
```

```
        m = statistics.mean(image.ravel())
        med = statistics.median(image.ravel())
        h,w = image.shape
        for row in range(h):
            for col in range(w):
                if m > image[row][col] and med < image[row][col]:
                    image[row][col] = 255
        photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
        canvas.create_image(0, 0, image=photo, anchor=NW)

    elif int(minInputGray.get()) >= int(maxInputGray.get()):
        messagebox.showwarning(title="Warning", message="The max gray level must
bigger then min gray level")
    else:
        h,w = image.shape
        for row in range(h):
            for col in range(w):
                if int(maxInputGray.get()) > image[row][col] and
int(minInputGray.get()) < image[row][col]:
                    image[row][col] = 255
        photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
        canvas.create_image(0, 0, image=photo, anchor=NW)


def lowFreq():
    global image
    global photo
    if frequancyFilterInput.get() == '':
        r = 60
    else:
        r = int(frequancyFilterInput.get())
    rows,cols = image.shape
    H = np.zeros((rows,cols),dtype=np.float32)
    for row in range(rows):
        for col in range(cols):
            D = np.sqrt((row-rows/2)**2 + (col-cols/2)**2)
            if D <= r:
                H[row,col] = 1
            else:
                H[row,col] = 0

    f = np.fft.fft2(image)
    f_shifted = np.fft.fftshift(f)
    plt.figure(1)
    plt.imshow(np.log1p(np.abs(f_shifted)),cmap='gray')
```

```
    plt.show()
    LPF = f_shifted * H
    plt.figure(2)
    plt.imshow(np.log1p(np.abs(LPF)),cmap='gray')
    plt.show()
    reverse = np.fft.ifftshift(LPF)
    generalLPF = np.abs(np.fft.ifft2(reverse))
    image = generalLPF
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def highFreq():
    global image
    global photo
    if frequancyFilterInput.get() == '':
        r = 60
    else:
        r = int(frequancyFilterInput.get())
    rows,cols = image.shape
    H = np.zeros((rows,cols),dtype=np.float32)
    for row in range(rows):
        for col in range(cols):
            D = np.sqrt((row-rows/2)**2 + (col-cols/2)**2)
            if D <= r:
                H[row,col] = 0
            else:
                H[row,col] = 1

    f = np.fft.fft2(image)
    f_shifted = np.fft.fftshift(f)
    plt.figure(1)
    plt.imshow(np.log1p(np.abs(f_shifted)),cmap='gray')
    plt.show()
    HPF = f_shifted * H
    plt.figure(2)
    plt.imshow(np.log1p(np.abs(HPF)),cmap='gray')
    plt.show()
    reverse = np.fft.ifftshift(HPF)
    generalLPF = np.abs(np.fft.ifft2(reverse))
    image = generalLPF
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)
```

```
def small():
    global image
    global photo
    image=cv2.resize(image,None,fx=0.75,fy=0.75)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def large():
    global image
    global photo
    image=cv2.resize(image,None,fx=2,fy=2,interpolation=cv2.INTER_CUBIC)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def add():
    global image
    global photo
    M = np.ones(image.shape, dtype="uint8") * 100
    image=cv2.add(image,M)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def sub():
    global image
    global photo
    M = np.ones(image.shape, dtype="uint8") * 100
    image=cv2.subtract(image,M)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)

def saveC(path, image, jpg_quality=None, png_compression=None):
    if jpg_quality:
        cv2.imwrite(path, image, [int(cv2.IMWRITE_JPEG_QUALITY), jpg_quality])
    elif png_compression:
        cv2.imwrite(path, image, [int(cv2.IMWRITE_PNG_COMPRESSION),
png_compression])
    else:
        cv2.imwrite(path, image)
def compressionSaveAs():
    global image
    global photo
    # save the image in JPEG format with 85% quality
    fS = filedialog.asksaveasfilename(initialdir=os.getcwd(),filetypes=(("JPG
File", ".jpg"),("PNG File", ".png")),
                                      defaultextension=".jpg", title="Save As")
```

```
    saveC(fS,image,jpg_quality=85)


# test

def testBtn():
    global image
    global photo
    data = np.float32(image).reshape((-1,2))
    critical = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 20, 1.0)
    ret, label, center = cv2.kmeans(data, 2, None, critical, 10,
cv2.KMEANS_RANDOM_CENTERS)
    center = np.uint8(center)
    result = center[label.flatten()]
    image = result.reshape(image.shape)
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)


def detectShapes():
    global image
    global photo
    mean = statistics.mean(image.ravel())
    _, thrash = cv2.threshold(image, mean , mean, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thrash, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    for contour in contours:
        epsilon = 0.01 * cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, epsilon, True)
        cv2.drawContours(image, [approx], 0, (0), 3)
        x, y = approx[0][0]
        if len(approx) == 3:
            cv2.putText(image, "Triangle", (x - 25 , y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 4:
            x1 ,y1, w, h = cv2.boundingRect(approx)
            aspectRatio = float(w)/h
            print(aspectRatio)
            if aspectRatio >= 0.95 and aspectRatio <= 1.05:
                cv2.putText(image, "Square", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
            else:
                cv2.putText(image, "Rectangle", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 5:
```

```
            cv2.putText(image, "Pentagon", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 6:
            cv2.putText(image, "Hexagon", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 7:
            cv2.putText(image, "Heptagon", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 8:
            cv2.putText(image, "Octagons", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 9:
            cv2.putText(image, "Nonagon", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 10:
            cv2.putText(image, "Star", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 11:
            cv2.putText(image, "Undecagon", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        elif len(approx) == 12:
            cv2.putText(image, "Dodecagons", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
        else:
            cv2.putText(image, "Circle", (x - 25, y + 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0))
    photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(image))
    canvas.create_image(0, 0, image=photo, anchor=NW)
# Part of displaying the screen

test = Tk()
width= test.winfo_screenwidth()
height= test.winfo_screenheight()
test.geometry("%dx%d" % (width, height))
test.resizable(False , False)
test.title("Mohamed Magdy ToolBox")
test.config(background='white')
test.iconbitmap(r"E:/Programing/openCv-py/working/Project/Icons/icons8-m-67
(1).ico")
test.state('zoomed')
tip= Balloon(test)
tip.config(bg='green')

tip.label.config(bg='red',fg='white',bd=2)
def changeOnHover(button):
```

```
    button.bind("<Enter>", func=lambda e: button.config(
        border=1))
    button.bind("<Leave>", func=lambda e: button.config(
        border=0))

# image =cv2.imread("./apple.jpg")
# height, width, no_channels = image.shape
canvas = Canvas(test , width = 700, height = 600,bg='#9AECDB')
canvas.place(x=width/4,y=height/7)
# //////colors//////////////

topIconsColor="#F8EFBA"
bottomIconsColor="#F8EFBA"
leftIconColor="#f7f1e3"
reightIconColor="#f7f1e3"
textColor="#3867d6"
headingColor="white"
# ////////////////////Top/////////////////////
canvasTop = Canvas(test , width = width, height = height/7-4,bg='#38ada9')
canvasTop.place(x=0,y=0)

l = Label(test, text = "Welcome To Mohamed Magdy Tool-Box")
l.config(font =("Helvetica", 22, BOLD),bg='#38ada9',fg=headingColor,pady=10)
l.pack()

browseIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/photo.png")
browseBtn = Button(test,image=browseIcon,border=0 ,fg='black',
bg=topIconsColor,width=50,height=40, command=uploadImage)
browseBtn.place(x=450,y=75)
tip.bind_widget(browseBtn,balloonmsg="Browse Image: That button use to browse
images")
changeOnHover(browseBtn)
# restoreBtn
# zoom_at
saveIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/diskette.png")
saveBtn = Button(test, image=saveIcon,border=0, fg='black', bg=topIconsColor
,width=50,height=40, command=Saving)
saveBtn.place(x=600,y=75)
tip.bind_widget(saveBtn,balloonmsg="Save Image: That button use to Save image as
edit image in same file of project"+"\n"+" and you can restore it")
changeOnHover(saveBtn)

# ////////////
```

```
resetIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/reset.png")
resetBtn=Button(test,
image=resetIcon,border=0,fg='black',bg=topIconsColor,width=50,height=40,
command=resets)
resetBtn.place(x=750,y=75)
tip.bind_widget(resetBtn,balloonmsg="Reset Image: That button use to reset image
that choosed")
changeOnHover(resetBtn)

restoreIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/recovery.png")
restoreBtn=Button(test,
image=restoreIcon,border=0,fg='black',bg=topIconsColor,width=50,height=40,
command=restoreBtn)
restoreBtn.place(x=900,y=75)
tip.bind_widget(restoreBtn,balloonmsg="Restore Image: That button use to restore
image that name edit image"+"\n"+"to complete work on it from last step")
changeOnHover(restoreBtn)

# ////////////////////////Bottom/////////////////////////
canvasBottom = Canvas(test , width = width, height = height/7+6,bg='#38ada9')
canvasBottom.place(x=0,y=700)


testIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/test-results.png")
testBtn = Button(test, image=testIcon,border=0, fg='black',
bg=bottomIconsColor,width=50,height=40, command=testBtn)
testBtn.place(x=350,y=740)
tip.bind_widget(testBtn,balloonmsg="test your image")
changeOnHover(testBtn)

histGraphBtnIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/wave-graph.png")
histGraphBtn = Button(test, image=histGraphBtnIcon, fg='black',
bg=bottomIconsColor,width=50,height=40, border=0,command=hisPlot)
histGraphBtn.place(x=500,y=740)
tip.bind_widget(histGraphBtn,balloonmsg="Histogram: is a graph, which shows
intensity distribution of an image "+"\n"+"mean that graph display the number of
iterations for each pixel's intensity")
changeOnHover(histGraphBtn)
```

```
detectObjIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/object.png")
DetectObjBtn = Button(test,  image=detectObjIcon, fg='black', border=0
,width=50,height=40, bg=bottomIconsColor, command=detectShapes)
DetectObjBtn.place(x=650,y=740)
tip.bind_widget(DetectObjBtn,balloonmsg="object detection: used to discover and
identify geometric shapes through"+"\n"+
                "In the beginning, it converts images to two colors only through
the threshold"+"\n"+
                "then detecte the contour of shapes"+"\n"+
                "then detecte the points of each contour"+"\n"+
                "Finally count the number of point and predect the name of
shape")
changeOnHover(DetectObjBtn)

matchIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/puzzle.png")
matchBtn = Button(test, image=matchIcon, fg='black', border=0 ,
bg=bottomIconsColor,width=50,height=40, command=matchBtnfun)
matchBtn.place(x=800,y=740)
tip.bind_widget(matchBtn,balloonmsg="Match image: use to match two images
together by show intensity pixel that match in two images")
changeOnHover(matchBtn)

mergeingImagesIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/merge.png")
mergeingImages = Button(test, image=mergeingImagesIcon, fg='black',
bg=bottomIconsColor,border=0,width=50,height=40, command=mergeingImages)
mergeingImages.place(x=950,y=740)
tip.bind_widget(mergeingImages,balloonmsg="Marge image: use to merge two image
together")
changeOnHover(mergeingImages)
# compressionSaveAs
compressionSaveAsIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/data-compression.png")
compressionSaveAs = Button(test, image=compressionSaveAsIcon, fg='black',
bg=bottomIconsColor,border=0,width=50,height=40, command=compressionSaveAs)
compressionSaveAs.place(x=1100,y=740)
tip.bind_widget(compressionSaveAs,balloonmsg="Compress the image")
changeOnHover(compressionSaveAs)

# ////////////////////////Left//////////////////////
canvasLeft = Canvas(test , width = 380, height = 575,bg='#7ed6df')
canvasLeft.place(x=0,y=height/7)
```

```
label = Label(test, text="Edit", relief=
RAISED,width=20,height=2,bg='#40739e',fg='white',
borderwidth=0,font=("Helvetica", 10))
label.place(x=85, y=150)

# tip.bind_widget(transformBtn,balloonmsg="Translate Bottom: use to move image to
Bottom")

Move_up = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/angle-small-down (1).png")
translateBt1
=Button(test,image=Move_up,bg=leftIconColor,width=40,height=40,border=0,
command=trUpBtn)
translateBt1.place(x=100,y=250)
tip.bind_widget(translateBt1,balloonmsg="Translate Bottom: use to move image to
Bottom")
changeOnHover(translateBt1)

Move_down = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/angle-small-up (2).png")
translateBt2
=Button(test,image=Move_down,fg='black',bg=leftIconColor,width=40,height=40,borde
r=0, command=trDwonBtn)
translateBt2.place(x=100,y=200)
tip.bind_widget(translateBt2,balloonmsg="Translate Top: use to move image to
top")
changeOnHover(translateBt2)

Move_left = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/angle-small-right.png")
translateBt3
=Button(test,image=Move_left,fg='black',bg=leftIconColor,width=40,height=40,borde
r=0, command=Tleft)
translateBt3.place(x=150,y=250)
tip.bind_widget(translateBt3,balloonmsg="Translate Right: use to move image to
right")
changeOnHover(translateBt3)

Move_right = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/angle-small-left.png")
translateBt4
=Button(test,image=Move_right,fg='black',bg=leftIconColor,width=40,height=40,bord
er=0, command=Treight)
translateBt4.place(x=50,y=250)
```

```
tip.bind_widget(translateBt4,balloonmsg="Translate Left: use to move image to
left")
changeOnHover(translateBt4)

translateBt5 =Scale(test, orient=HORIZONTAL, from_=1, to=20,
label='skew',border=0, command=affineTransform,bg=leftIconColor, resolution=3,
tickinterval=2, length=170, sliderlength=20, showvalue=0)
translateBt5.place(x=50,y=330)
changeOnHover(translateBt5)

flipIconH = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/flip-horizontal.png")
flibBt1
=Button(test,image=flipIconH,fg='black',width=50,height=40,border=0,bg=leftIconCo
lor, command=flipingHeFunction)
flibBt1.place(x=300,y=200)
tip.bind_widget(flibBt1,balloonmsg="Flip Vertical: use to reverses the image on
the X-axis")
changeOnHover(flibBt1)

flipIconV = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/reflect.png")
flibBt2
=Button(test,image=flipIconV,fg='black',width=50,height=40,border=0,bg=leftIconCo
lor, command=flipingVeFunction)
flibBt2.place(x=300,y=250)
tip.bind_widget(flibBt2,balloonmsg="Flip Horizontal: use to reverses the image on
the Y-axis")
changeOnHover(flibBt2)

rotateIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/rotate-right.png")
rotateBt1
=Button(test,image=rotateIcon,fg='black',width=50,height=40,border=0,bg=leftIconC
olor, command=rotBtn)
rotateBt1.place(x=300,y=300)
tip.bind_widget(rotateBt1,balloonmsg="Rotate Image: That button use to rotate
image with 90 deg"+"\n"+"with clockwise")
changeOnHover(rotateBt1)
```

```
negativeBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/icons8-negative-32.png")
negativeBtn = Button(test, image=negativeBtnIcon, fg='black',
bg=leftIconColor,border=0,width=50,height=40, command=NegativeFunction)
negativeBtn.place(x=300,y=380)
tip.bind_widget(negativeBtn,balloonmsg="negative image: use to enhancing white
or"+"\n"+" grey detail embedded in dark regions of an image")
changeOnHover(negativeBtn)

histBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/icons8-adjust-32.png")
histBtn = Button(test, image=histBtnIcon, fg='black',
bg=leftIconColor,border=0,width=50,height=40, command=equalizeHist)
histBtn.place(x=300,y=430)
tip.bind_widget(histBtn,balloonmsg="Histogram Equlization: use to applied to a
dark or"+"\n"+"washed out images in order to improve image contrast")
changeOnHover(histBtn)

addBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/lightbulb.png")
addBtn = Button(test, image=addBtnIcon, fg='black',
bg=leftIconColor,border=0,width=50,height=40, command=add)
addBtn.place(x=20,y=430)
tip.bind_widget(addBtn,balloonmsg="Increase the brightness of image")
changeOnHover(addBtn)

subBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/light-bulb.png")
subBtn = Button(test, image=subBtnIcon, fg='black',
bg=leftIconColor,border=0,width=50,height=40, command=sub)
subBtn.place(x=80,y=430)
tip.bind_widget(subBtn,balloonmsg="Decrease the brightness of image")
changeOnHover(subBtn)

smallBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/reduce.png")
smallBtn = Button(test, image=smallBtnIcon, fg='black',
bg=leftIconColor,border=0,width=50,height=40, command=small)
smallBtn.place(x=140,y=430)
tip.bind_widget(subBtn,balloonmsg="Decrease the size of the image image")
changeOnHover(smallBtn)

largeBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/expand (1).png")
```

```
largeBtn = Button(test, image=largeBtnIcon, fg='black',
bg=leftIconColor,border=0,width=50,height=40, command=large)
largeBtn.place(x=200,y=430)
tip.bind_widget(subBtn,balloonmsg="Doubling the size of the image image")
changeOnHover(largeBtn)
# ////////////////////Edge Detection//////////////////////



label = Label(test, text="Edge Detection", relief=
RAISED,width=20,height=2,bg='#40739e',fg='white',
borderwidth=0,font=("Helvetica", 10))
label.place(x=85, y=500)

edgeBtn_HorIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/menu.png")
edgeBtn_Hor = Button(test, image=edgeBtn_HorIcon,
fg='black',width=50,height=40,border=0, bg=reightIconColor,
command=edgeDetectHor)
edgeBtn_Hor.place(x=20,y=580)
tip.bind_widget(edgeBtn_Hor,balloonmsg="Sobal edge detection: use to find the
edges horizental in image")
changeOnHover(edgeBtn_Hor)

edgeBtn_VerIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/vertical-lines.png")
edgeBtn_Ver = Button(test,  image=edgeBtn_VerIcon
,fg='black',width=50,height=40,border=0, bg=reightIconColor,
command=edgeDetectVer)
edgeBtn_Ver.place(x=90,y=580)
tip.bind_widget(edgeBtn_Ver,balloonmsg="Sobal edge detection: use to find the
edges vertical in image")
changeOnHover(edgeBtn_Ver)

edgeBtn_laplacianIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/edgeLaplacian.png")
edgeBtn_laplacian = Button(test,
image=edgeBtn_laplacianIcon,fg='black',width=50,height=40,border=0,
bg=reightIconColor, command=edgeDetectLaplacian)
edgeBtn_laplacian.place(x=160,y=580)
tip.bind_widget(edgeBtn_laplacian,balloonmsg="laplacian edge detection: use to
find the all edges in image")
changeOnHover(edgeBtn_laplacian)
```

```
sobelDigIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/lines (1).png")
sobelDigBtn = Button(test, image=sobelDigIcon, fg='black',
width=50,height=40,border=0 ,bg=reightIconColor, command=digonalSobel)
sobelDigBtn.place(x=230,y=580)
tip.bind_widget(sobelDigBtn,balloonmsg="Sobal edge detection: use to find the
edges diagonal in image")
changeOnHover(sobelDigBtn)

sobelIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/hexagon.png")
sobelBtn = Button(test, image=sobelIcon, fg='black'
,width=50,height=40,border=0,bg=reightIconColor, command=FullsobelFun)
sobelBtn.place(x=300,y=580)
tip.bind_widget(sobelBtn,balloonmsg="Sobal edge detection: use to find the edges
in all direction in image")
changeOnHover(sobelBtn)

# ////////////////////Reight/////////////////////////
canvasReight = Canvas(test , width = 500, height = 575,bg='#7ed6df')
canvasReight.place(x=1080,y=height/7)

label = Label(test, text="Filters", relief=
RAISED,width=20,fg="white",height=2,bg='#40739e',
borderwidth=0,font=("Helvetica", 10))
label.place(x=1220, y=150)


label4 = Label(test, text="Choose Filter",bg='#7ed6df',fg=textColor, relief=
RAISED, borderwidth=0,border=0,font=("Helvetica", 10))
label4.place(x=1100, y=200)
filterSelect = StringVar()
filters = Combobox(test,values=('None', 'Gray-scale', 'Blur 3X3', 'Blur 5X5',
                                'Motion-blur', 'Gaussian blur', 'Median blur',
'Bilateral filter',
                                'sharping')
                   ,state='readonly',textvariable=filterSelect,width=30)
filters.place(x=1100,y=220)
filters.current(0)
filters.bind("<<ComboboxSelected>>", FilterCombobox)
motion_blur_scale = Scale(test,bg=reightIconColor, orient=HORIZONTAL, from_=1,
to=20, command=scaleOfMotion, state=DISABLED, label= 'Motion Blur', resolution=3,
tickinterval=2, length=170, sliderlength=20, showvalue=0)
motion_blur_scale.place(x=1330,y=220)
```

```
label4 = Label(test, text="Power Transformation",bg='#7ed6df',fg=textColor,
relief= RAISED, borderwidth=0,border=0,font=("Helvetica", 8))
label4.place(x=1090, y=300)

powerSelect = StringVar()
powerBtn = Combobox(test,values=('None','0.1','0.5','1.2','2.2')
                    ,state='readonly',textvariable=powerSelect,width=15)
powerBtn.place(x=1090,y=320)
powerBtn.current(0)
powerBtn.bind("<<ComboboxSelected>>", powerCombobox)
tip.bind_widget(powerBtn,balloonmsg="Power transformation: use to map a narrow
range of dark input values into a wider range of output values or vice versa
depending on γ value"+"\n"
                                    +"and can chooes the value of γ from
combobox")


label4 = Label(test, text="Bit Plane Transformation",bg='#7ed6df',fg=textColor,
relief= RAISED, borderwidth=0,border=0,font=("Helvetica", 8))
label4.place(x=1240, y=300)
bitPlaneSelect = StringVar()
bitPlane = Combobox(test,values=('None','128','64','32','16','8','4','2','1')
                    ,state='readonly',textvariable=bitPlaneSelect,width=15)
bitPlane.place(x=1240,y=320)
bitPlane.current(0)
bitPlane.bind("<<ComboboxSelected>>", bitPlaneCombobox)
tip.bind_widget(bitPlane,balloonmsg="Bit Plane Slicing: is a method of
representing an image with one or more bits of the byte used for each pixel"+"\n"
                                    +"and can chooes the value of bits of the
byte from combobox")



label4 = Label(test, text="Threeshold filter", relief=
RAISED,bg='#7ed6df',fg=textColor, borderwidth=0,border=0,font=("Helvetica", 8))
label4.place(x=1380, y=300)
transformSelect = StringVar()
transformCombobox = Combobox(test,values=('None', 'Dilation', 'Erosion',
'Opening', 'Closing', 'Gradient', 'Top Hat', 'Black Hat')
                    ,state='readonly',textvariable=transformSelect,width=15)
transformCombobox.place(x=1380,y=320)
```

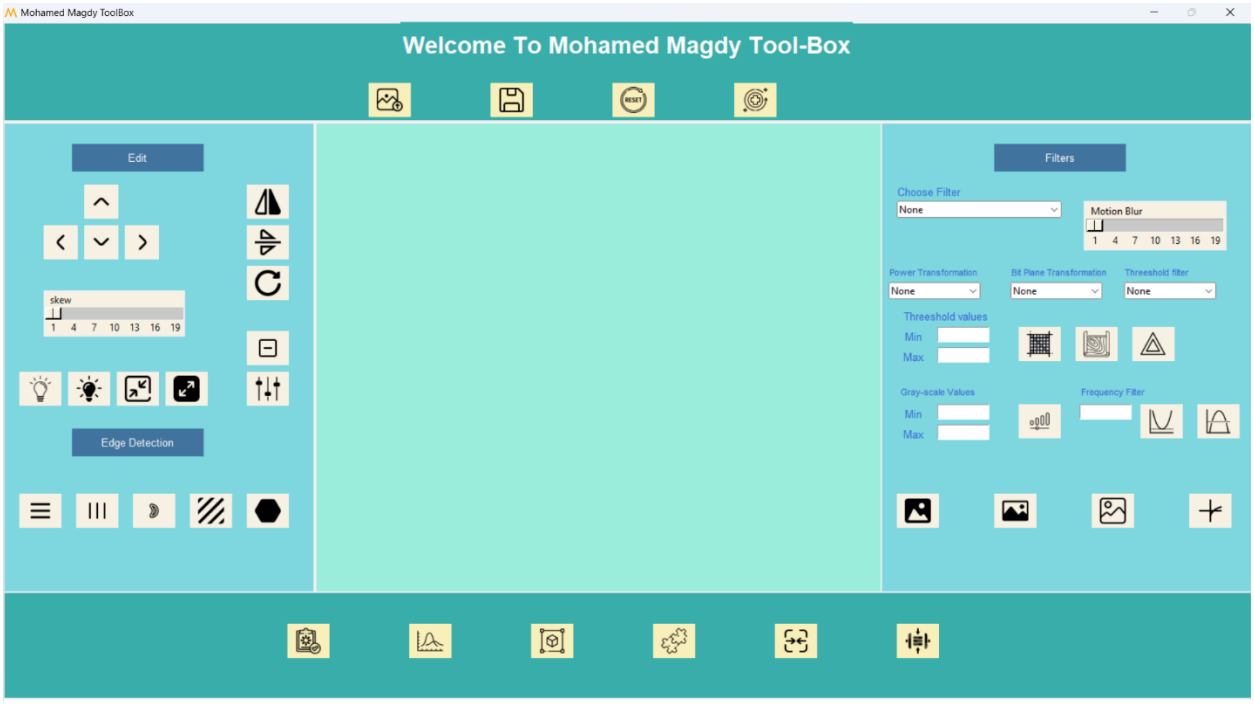
```
transformCombobox.current(0)
transformCombobox.bind("<<ComboboxSelected>>", TransformCombobox)

label = Label(test, text="Threeshold values", relief=
RAISED,width=15,height=2,bg='#7ed6df',fg=textColor,
borderwidth=0,font=("Helvetica", 10))
label.place(x=1100, y=345)
label = Label(test, text="Min", relief=
RAISED,width=5,height=2,bg='#7ed6df',fg=textColor,
borderwidth=0,font=("Helvetica", 10))
label.place(x=1100, y=370)
minInput = Entry(test, width=10)
minInput.place(x=1150,y=375)

label = Label(test, text="Max", relief=
RAISED,width=5,height=2,bg='#7ed6df',fg=textColor,
borderwidth=0,font=("Helvetica", 10))
label.place(x=1100, y=395)
maxInput = Entry(test, width=10)
maxInput.place(x=1150,y=400)

thresholdBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/pixels.png")
thresholdBtn = Button(test, image=thresholdBtnIcon,
fg='black',width=50,height=40, bg=reightIconColor,border=0, command=thresholdBtn)
thresholdBtn.place(x=1250,y=375)
tip.bind_widget(thresholdBtn,balloonmsg="Threshold: used to convert a grayscale
image to a binary image, where the pixels are either 0 or 255.")
changeOnHover(thresholdBtn)

contourBtnIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/contour.png")
contourBtn = Button(test, image=contourBtnIcon, fg='black',width=50,height=40,
bg=reightIconColor,border=0, command=contourBtn)
contourBtn.place(x=1320,y=375)
tip.bind_widget(contourBtn,balloonmsg="contour detection: used to highlight
borders of objects")
changeOnHover(contourBtn)

edgeBtn_CannyIcon=PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/hazard.png")
edgeBtn_Canny = Button(test,
image=edgeBtn_CannyIcon,fg='black',width=50,height=40,
bg=reightIconColor,border=0, command=edgeDetectCanny)
edgeBtn_Canny.place(x=1390,y=375)
```

```
tip.bind_widget(edgeBtn_Canny,balloonmsg="canny edge detection: use to find the
all edges in image"+"\n"+"by threshold values minimam and maximam")
changeOnHover(edgeBtn_Canny)

labelMaxGray = Label(test, text="Gray-scale Values", border=0
,width=15,height=2,bg='#7ed6df',fg=textColor,font=("Helvetica", 8), relief=
RAISED, borderwidth=0)
labelMaxGray.place(x=1105, y=440)

labelMinGray = Label(test, text="Min", border=0
,width=5,height=2,bg='#7ed6df',fg=textColor,font=("Helvetica", 10), relief=
RAISED, borderwidth=0)
labelMinGray.place(x=1100, y=465)

labelMaxGray = Label(test, text="Max", border=0
,width=5,height=2,bg='#7ed6df',fg=textColor,font=("Helvetica", 10), relief=
RAISED, borderwidth=0)
labelMaxGray.place(x=1100, y=490)

minInputGray = Entry(test, width=10)
minInputGray.place(x=1150,y=470)
maxInputGray = Entry(test, width=10)
maxInputGray.place(x=1150,y=495)

greyLevelIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/volume-control.png")
greyLevelBtn = Button(test, image=greyLevelIcon, bg=reightIconColor,
border=0,width=50,height=40, command=grayMethod)
greyLevelBtn.place(x=1250,y=470)
tip.bind_widget(greyLevelBtn,balloonmsg="Gray level slicing: is a way to
highlight gray range of interest to a viewer by one of two ways"+"\n"
                                    +"and can select two value from inputs of
Gray level"+"\n"
                                    +"if you dont select two values it will
calculate mean and mediam of image and do gray level")
changeOnHover(greyLevelBtn)

logTransformIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/logarithm.png")
logTransform = Button(test, image=logTransformIcon,bg=reightIconColor, border=0
,width=50,height=40, command=logBtn)
logTransform.place(x=1460,y=580)
tip.bind_widget(logTransform,balloonmsg="Log Filter: use to map a narrow range of
dark input values into a wider range of output values")
changeOnHover(logTransform)
```

```
# //////////////
labelMaxGray = Label(test, text="Frequency Filter", border=0
,width=15,height=2,bg='#7ed6df',fg=textColor,font=("Helvetica", 8), relief=
RAISED, borderwidth=0)
labelMaxGray.place(x=1320, y=440)

frequancyFilterInput = Entry(test, width=10)
frequancyFilterInput.place(x=1325,y=470)


lowFrequencylIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/icons8-minimum-value-32.png")
lowFrequencyBtn = Button(test, image=lowFrequencylIcon, bg=reightIconColor,
border=0,width=50,height=40, command=lowFreq)
lowFrequencyBtn.place(x=1400,y=470)
tip.bind_widget(lowFrequencyBtn,balloonmsg="The Low Pass Filter: the low pass
filter only allows low frequency signals from 0Hz to its cut-off frequency")
changeOnHover(lowFrequencyBtn)

heighFrequencylIcon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/icons8-bell-curve-32.png")
heighFrequencyBtn = Button(test, image=heighFrequencylIcon, bg=reightIconColor,
border=0,width=50,height=40, command=highFreq)
heighFrequencyBtn.place(x=1470,y=470)
tip.bind_widget(heighFrequencyBtn,balloonmsg="A high-pass filter: task is just
the opposite of a low-pass filter: to offer easy passage of a high-frequency
signal and difficult passage to a low-frequency signal")
changeOnHover(heighFrequencyBtn)



searingBt1Icon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/insert-picture-icon.png")
searingBt1
=Button(test,image=searingBt1Icon,fg='black',width=50,border=0,height=40,bg=leftI
conColor, command=Sharping1)
searingBt1.place(x=1100,y=580)
tip.bind_widget(searingBt1,balloonmsg="Low sharping: use to do low sharping on
image"+"\n"+"Must chooes sharping from filters to can use it")
changeOnHover(searingBt1)

searingBt2Icon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/photo (2).png")
```

```
searingBt2
=Button(test,image=searingBt2Icon,fg='black',bg=leftIconColor,border=0,width=50,h
eight=40, command=Sharping2)
searingBt2.place(x=1220,y=580)
tip.bind_widget(searingBt2,balloonmsg="Mediam sharping: use to do mediam sharping
on image"+"\n"+"Must chooes sharping from filters to can use it")
changeOnHover(searingBt2)

searingBt3Icon = PhotoImage(file= r"E:/Programing/openCv-
py/working/Project/NewIcons/image.png")
searingBt3
=Button(test,image=searingBt3Icon,fg='black',bg=leftIconColor,border=0,width=50,h
eight=40, command=Sharping3)
searingBt3.place(x=1340,y=580)
tip.bind_widget(searingBt3,balloonmsg="High sharping: use to do high sharping on
image"+"\n"+"Must chooes sharping from filters to can use it")
changeOnHover(searingBt3)
test.mainloop()
```

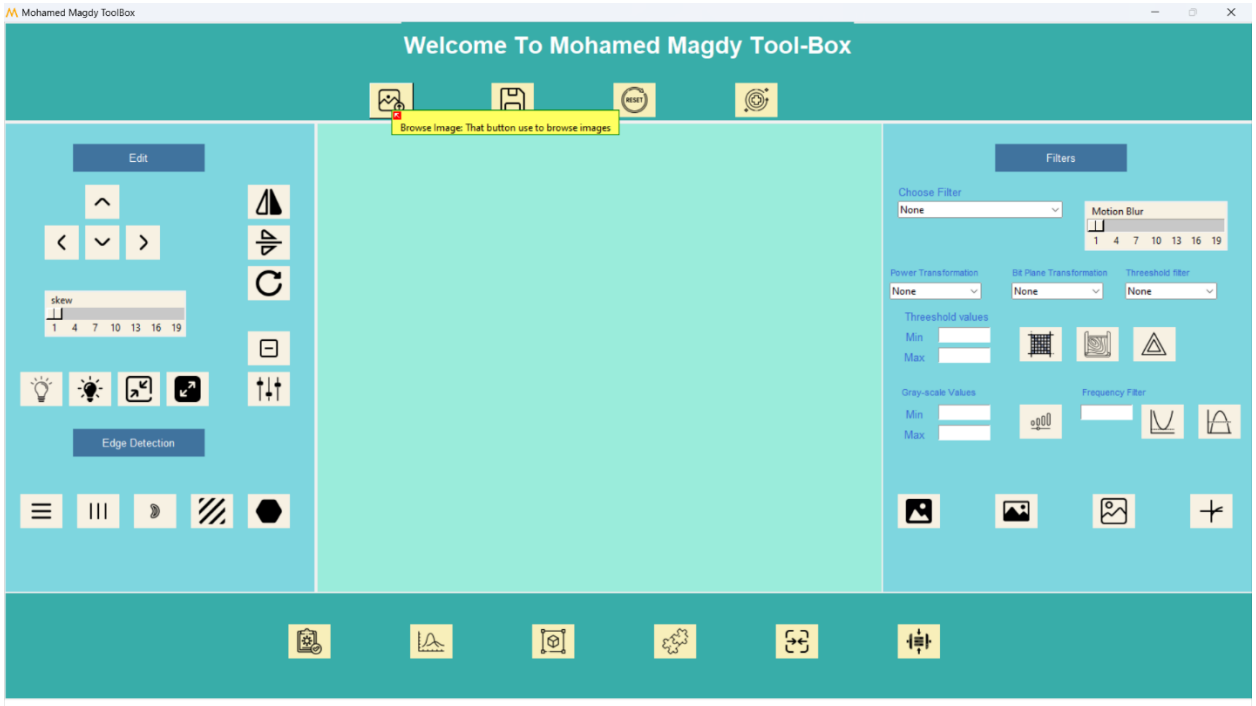
Fourth program final view:



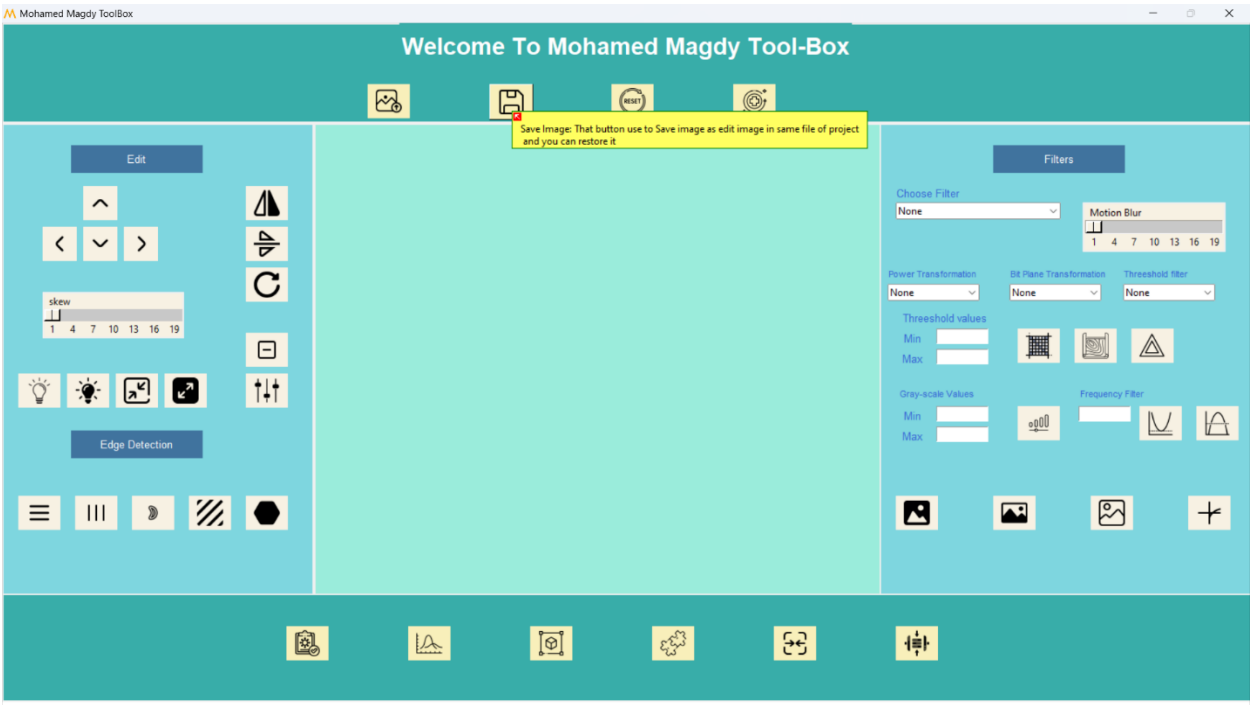
Fifth Every button and component in the program:

Main part

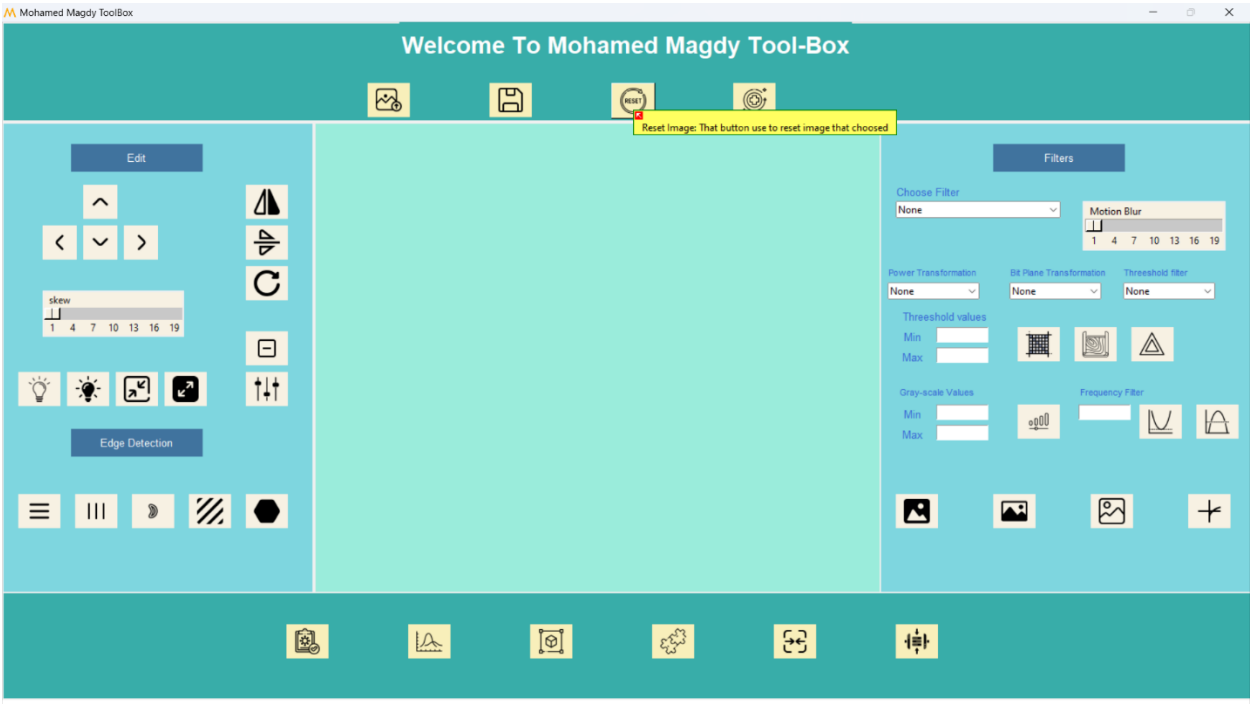
1. Upload Image



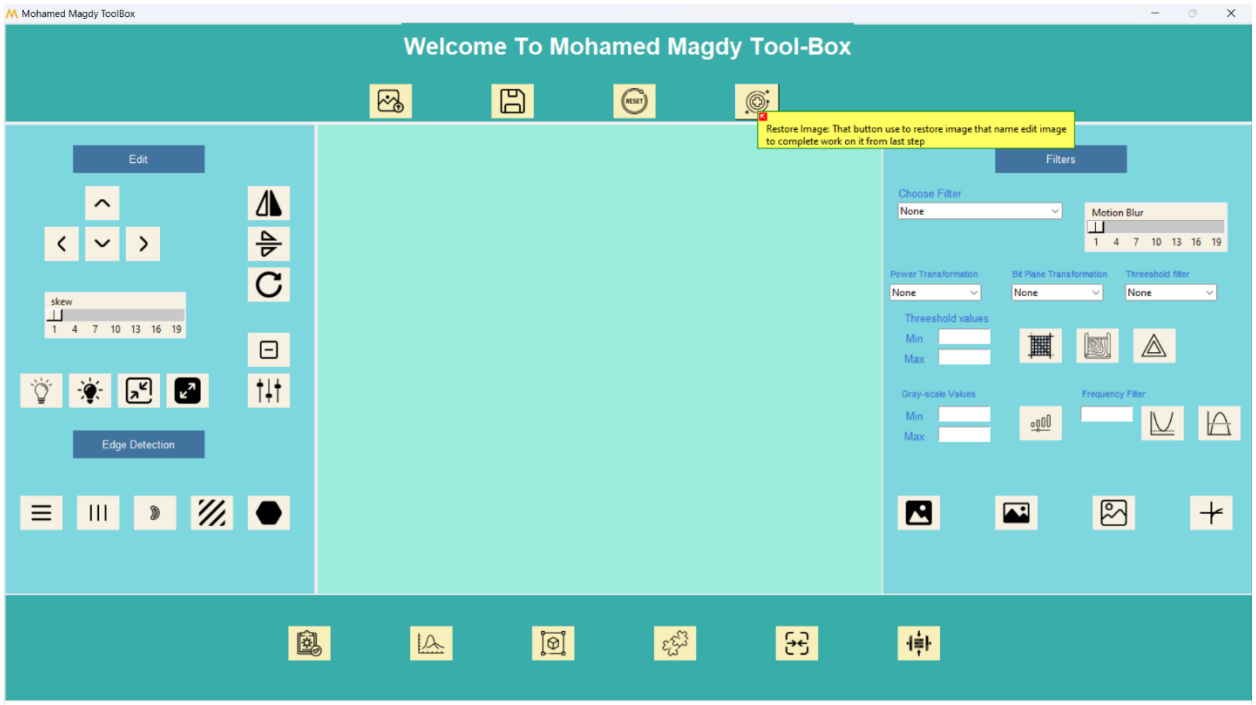
2. Save image



3. Reset Image to the first upload view

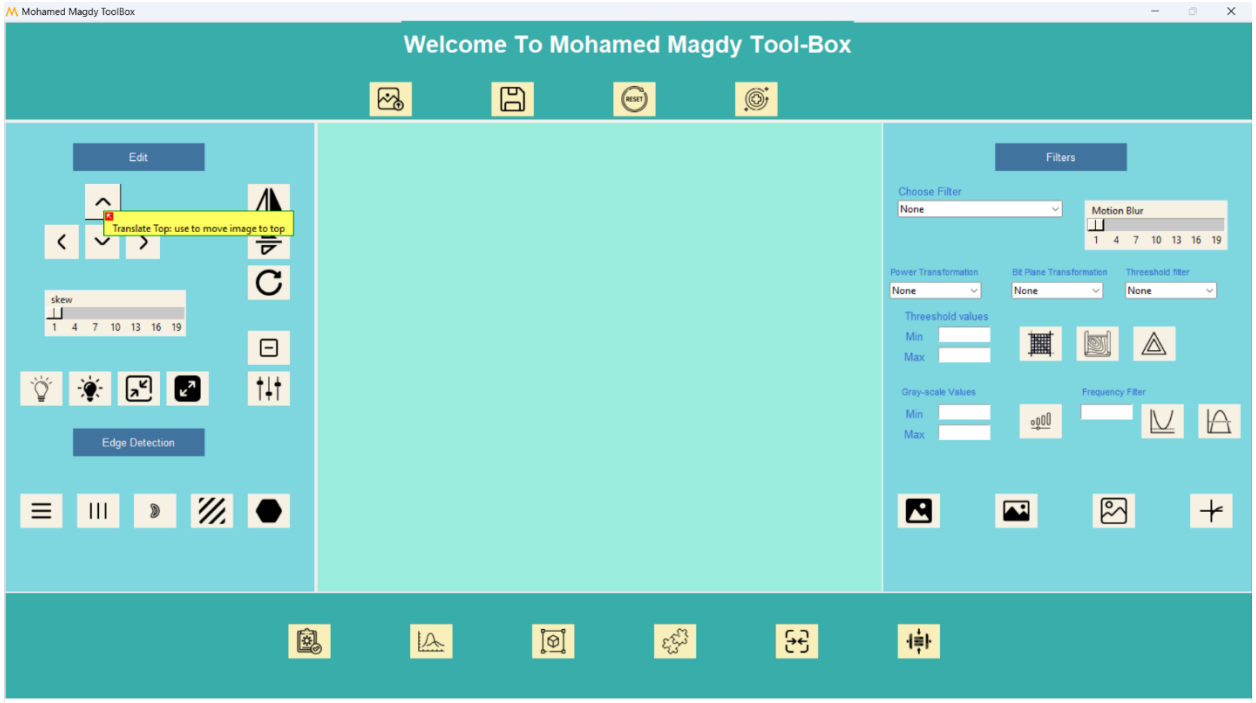


4. Restore Image after Saving it

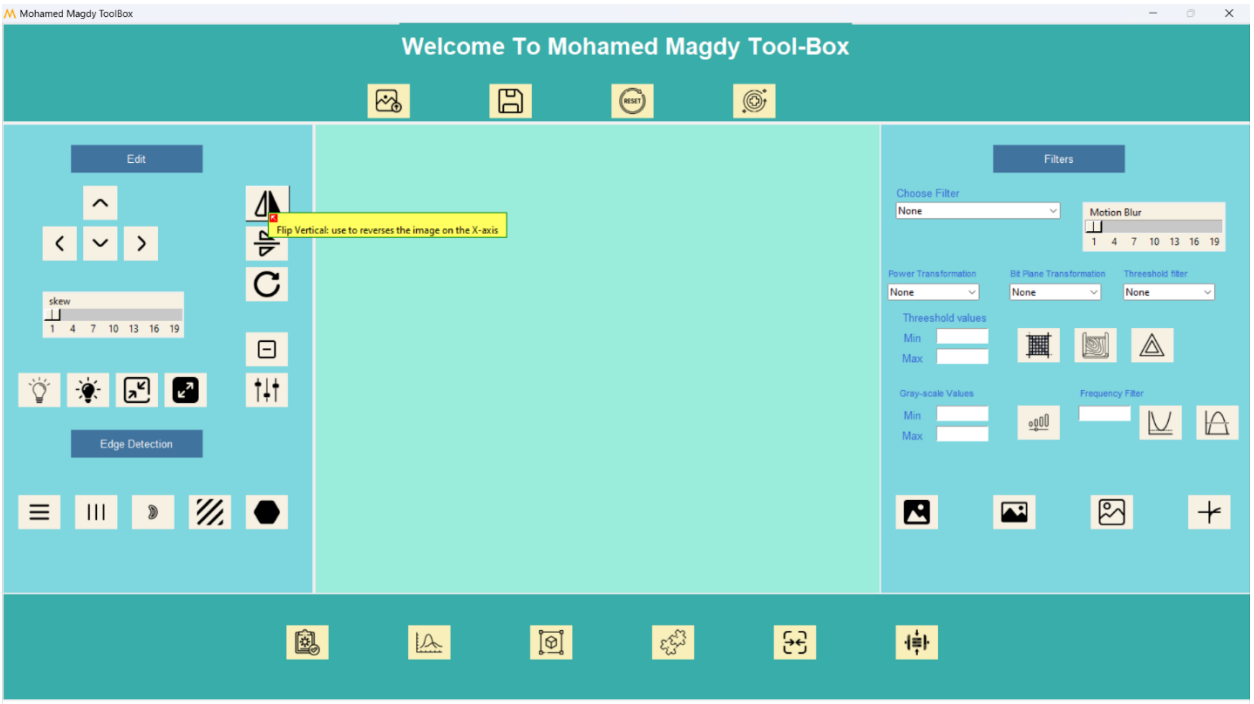


Edit part

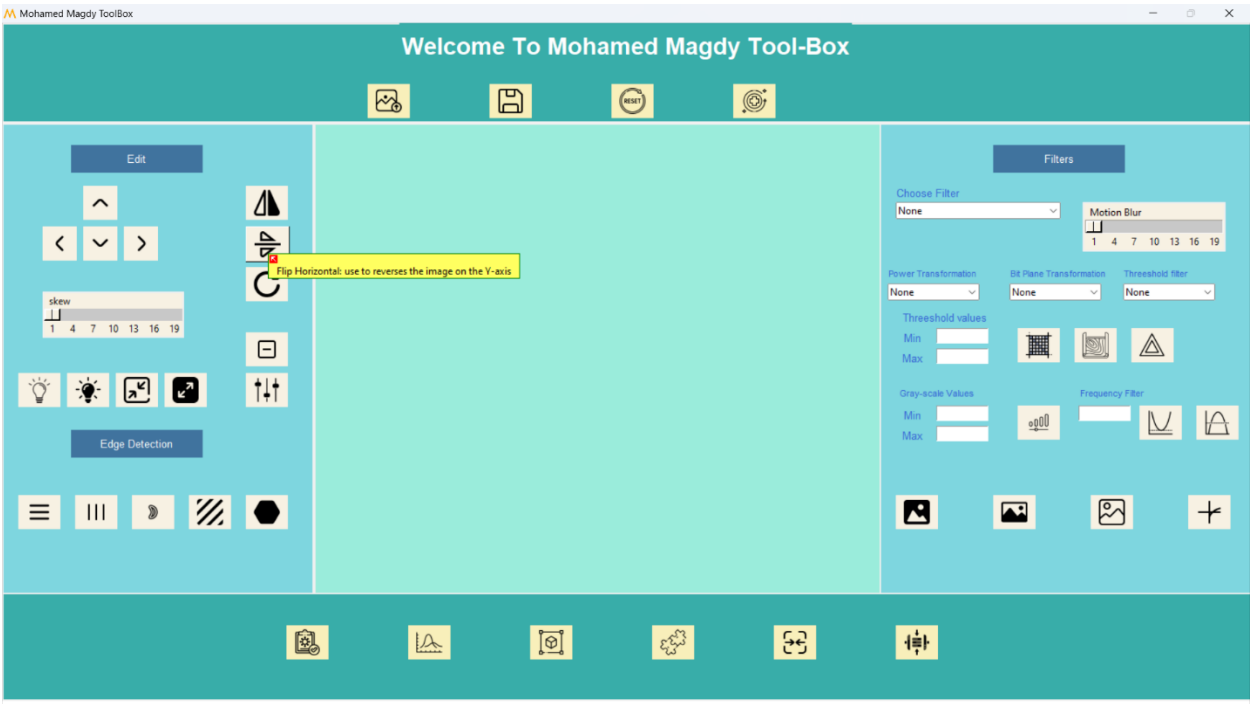
5. Translate up & down and left & right



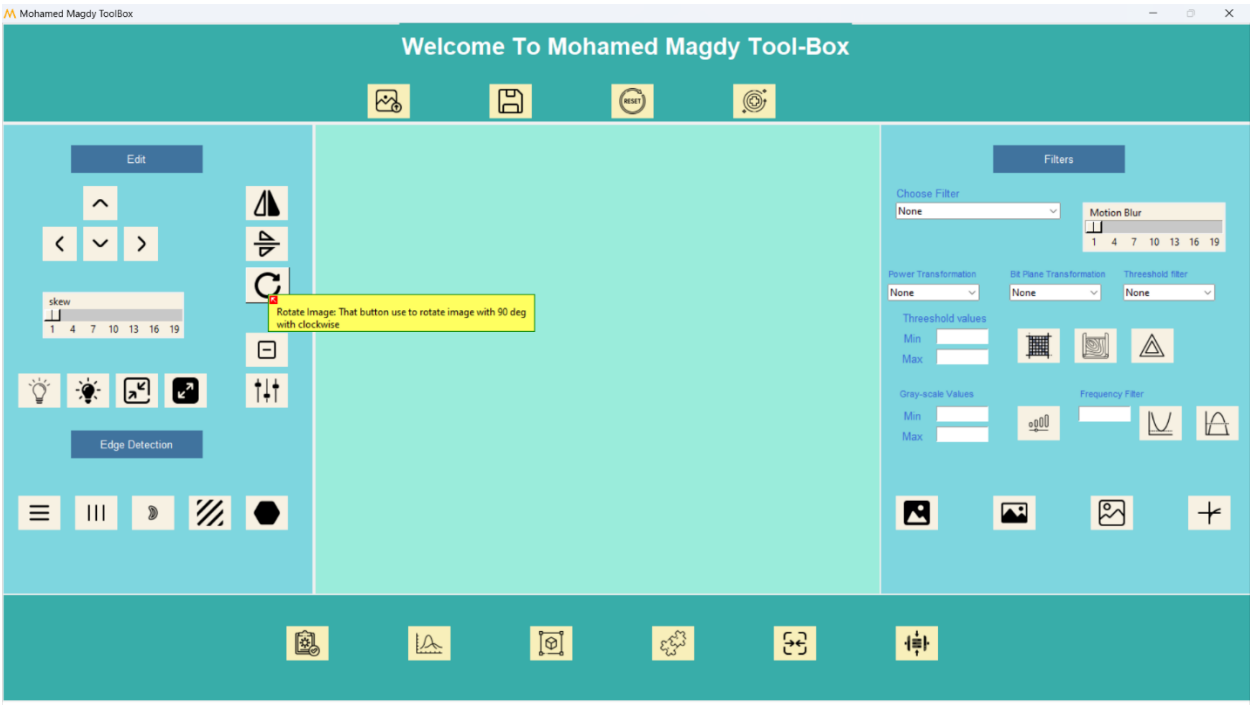
6. Flip Vertical



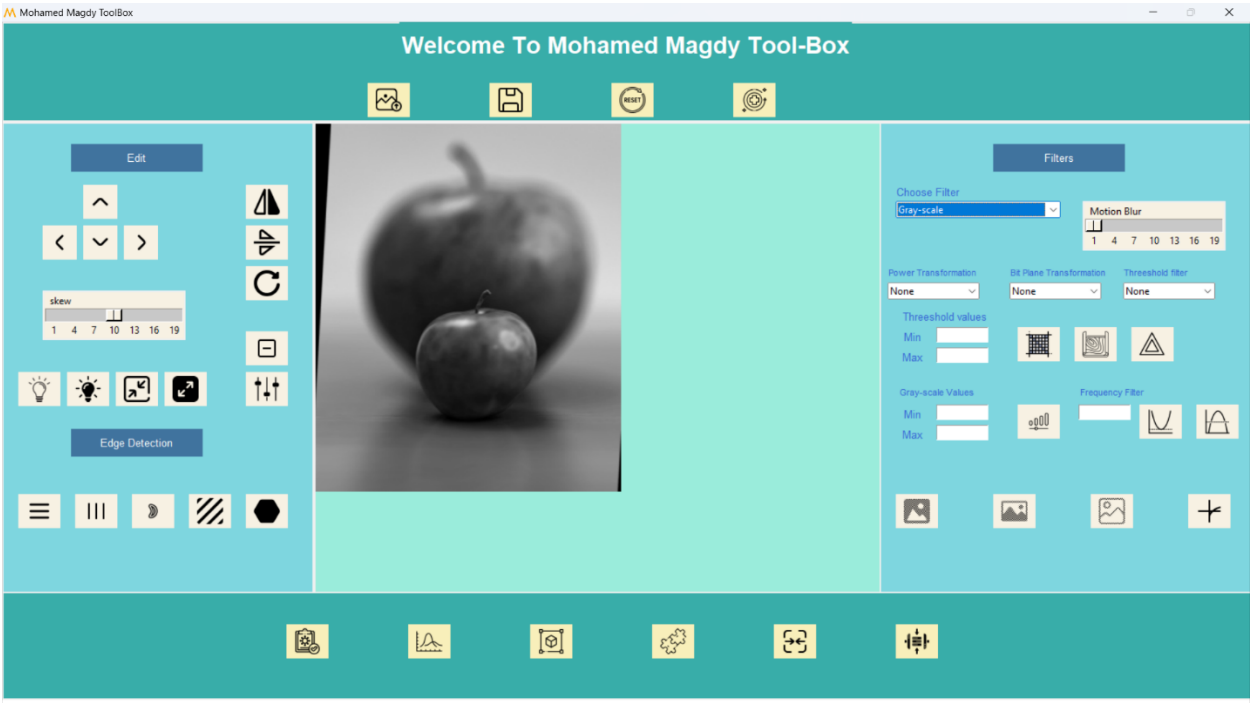
7. Flip horizontal



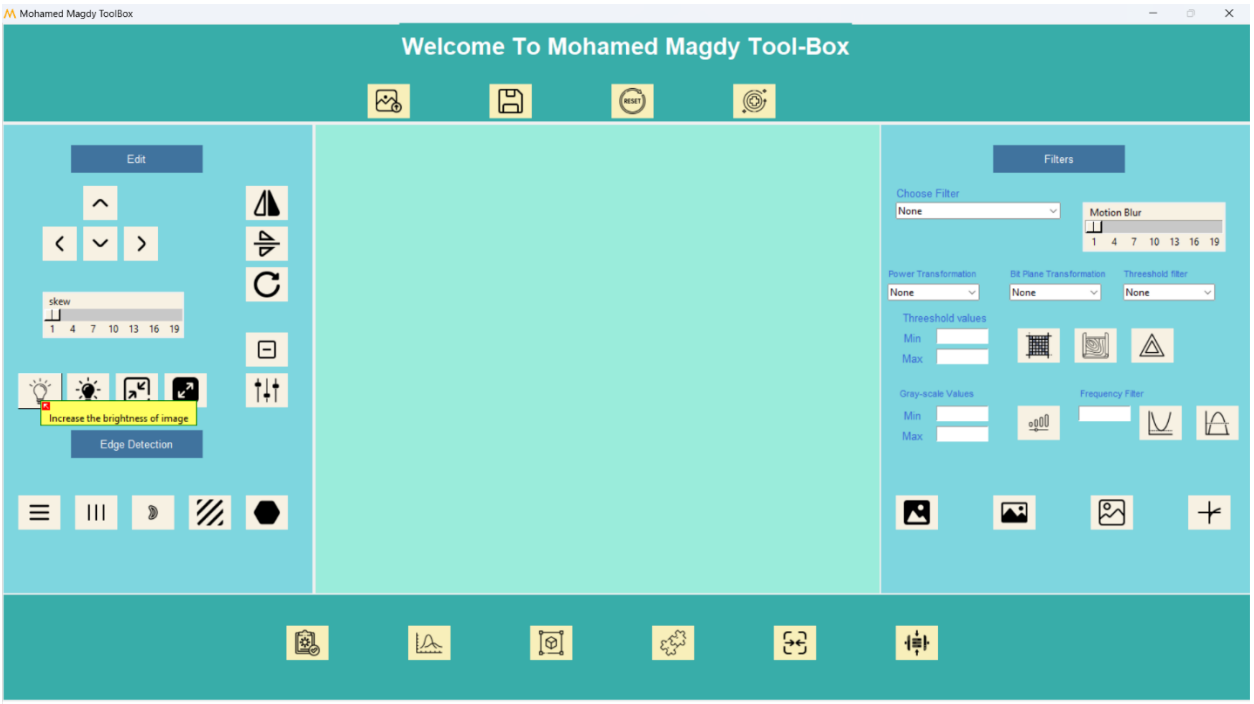
8. Rotate the Image



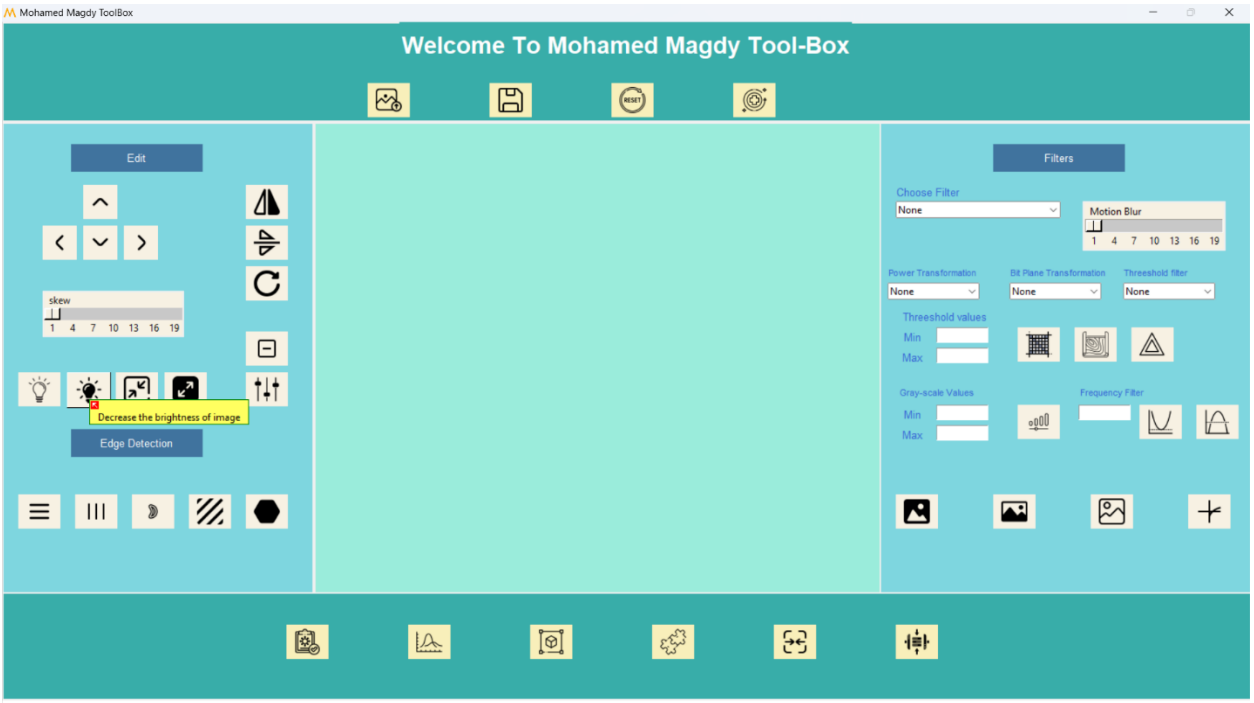
9. Skewing the image



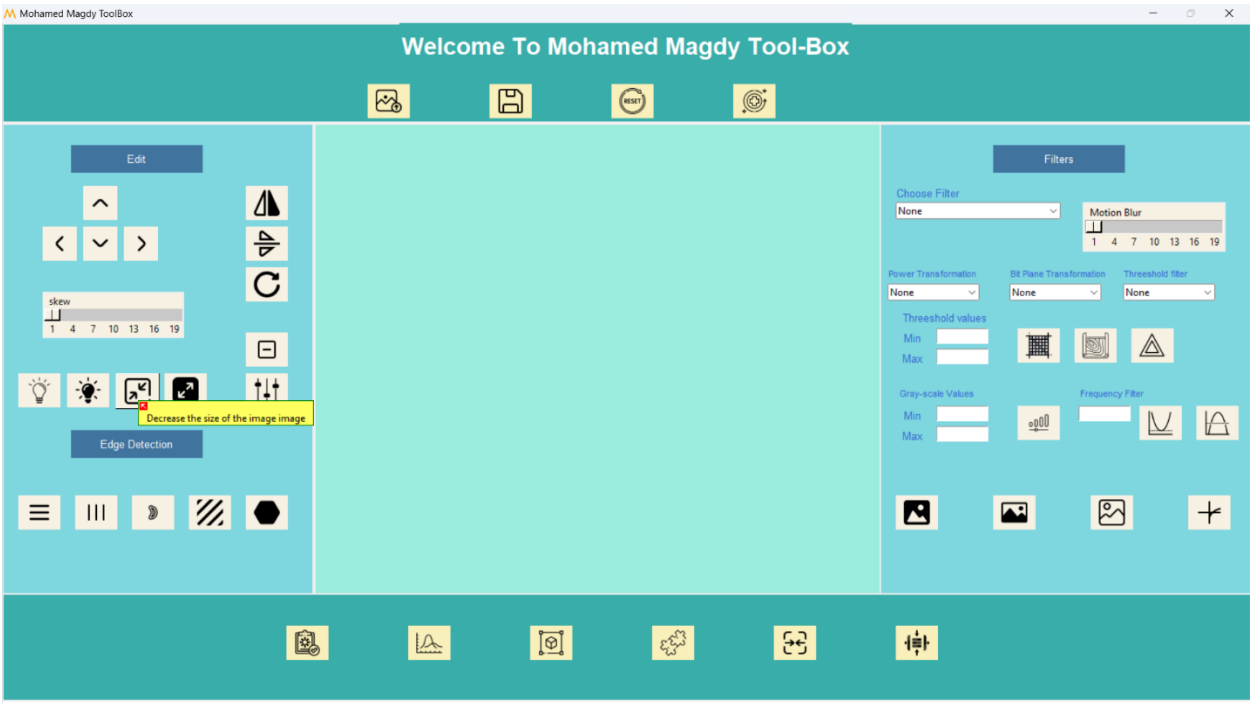
10. Increase the brightness



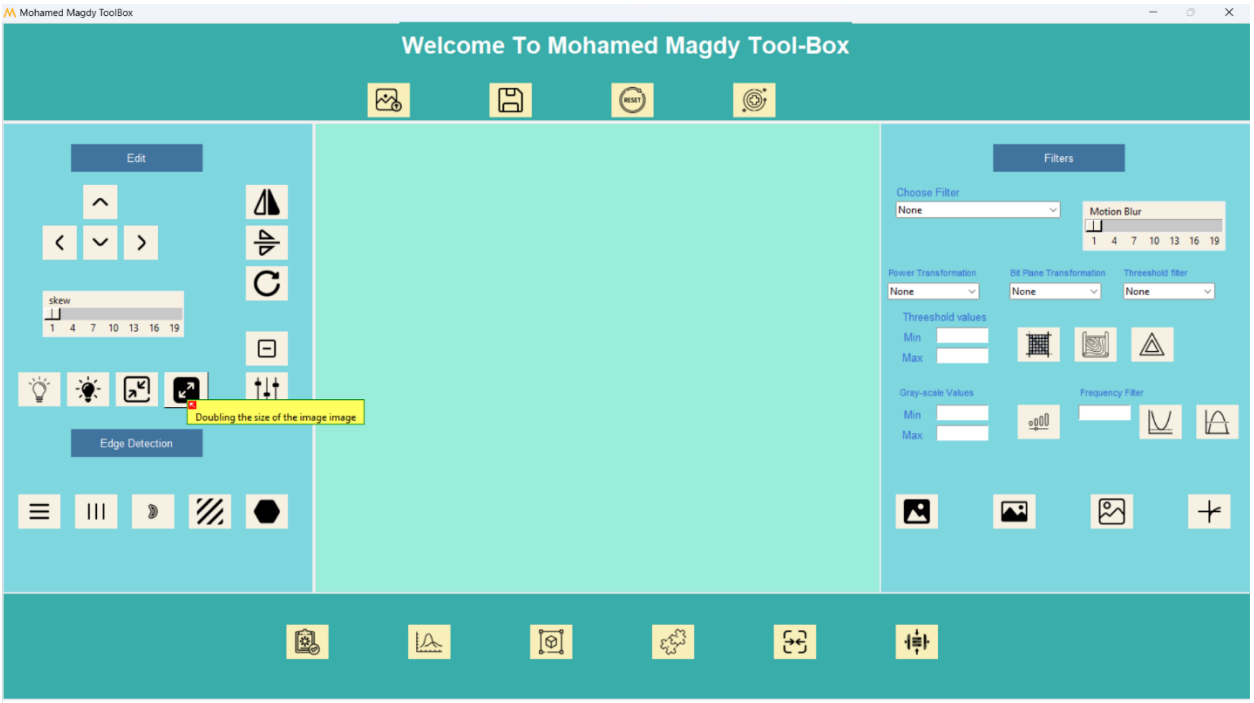
11. Decrease brightness



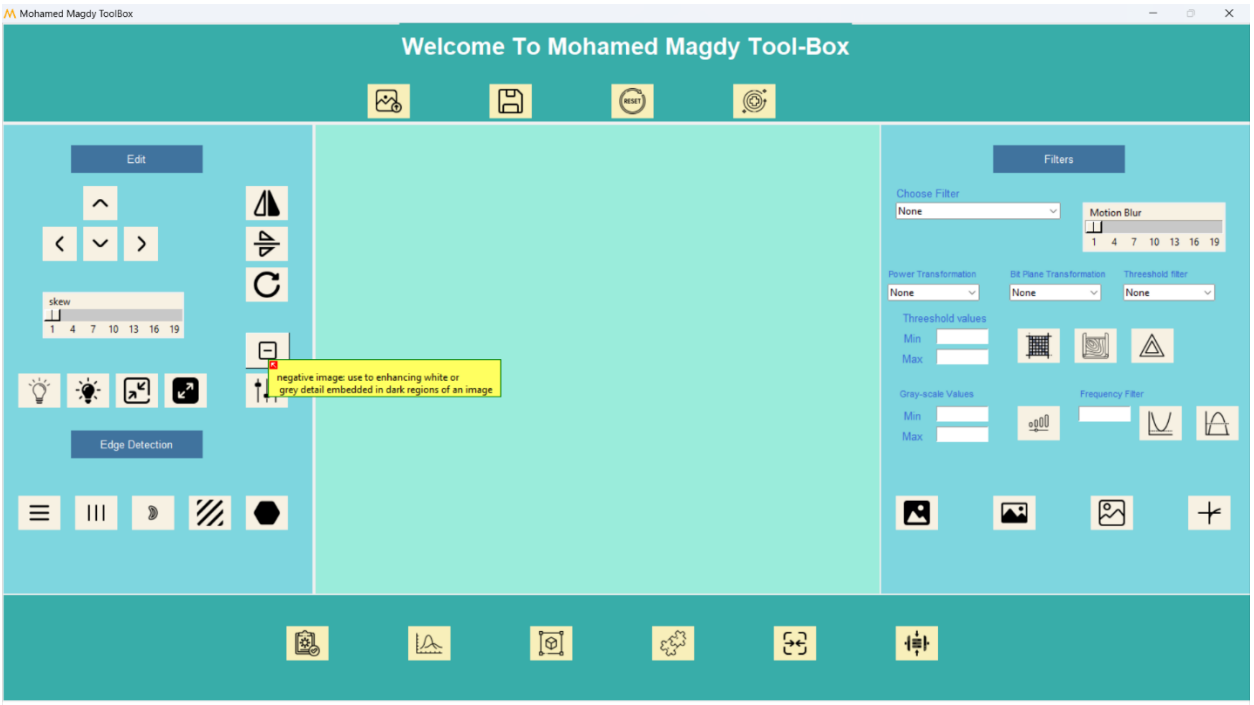
12. Minimize the image



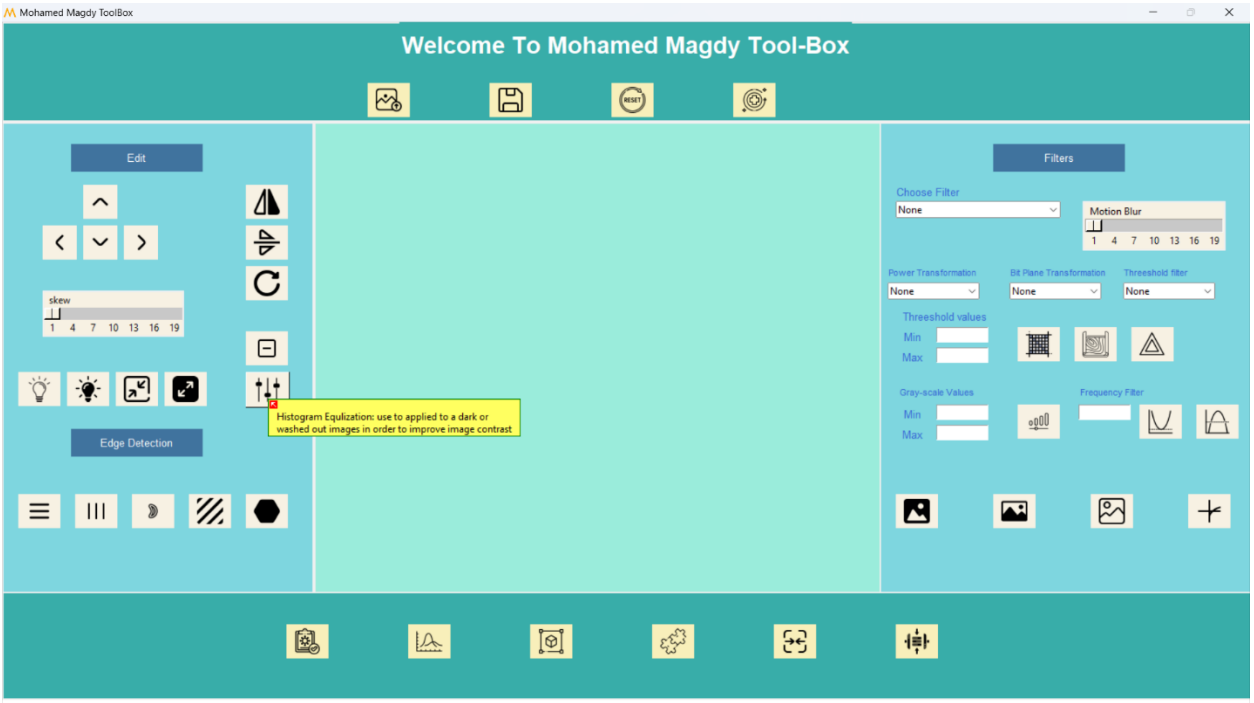
13. Doubling the size of image



14. Negative the image

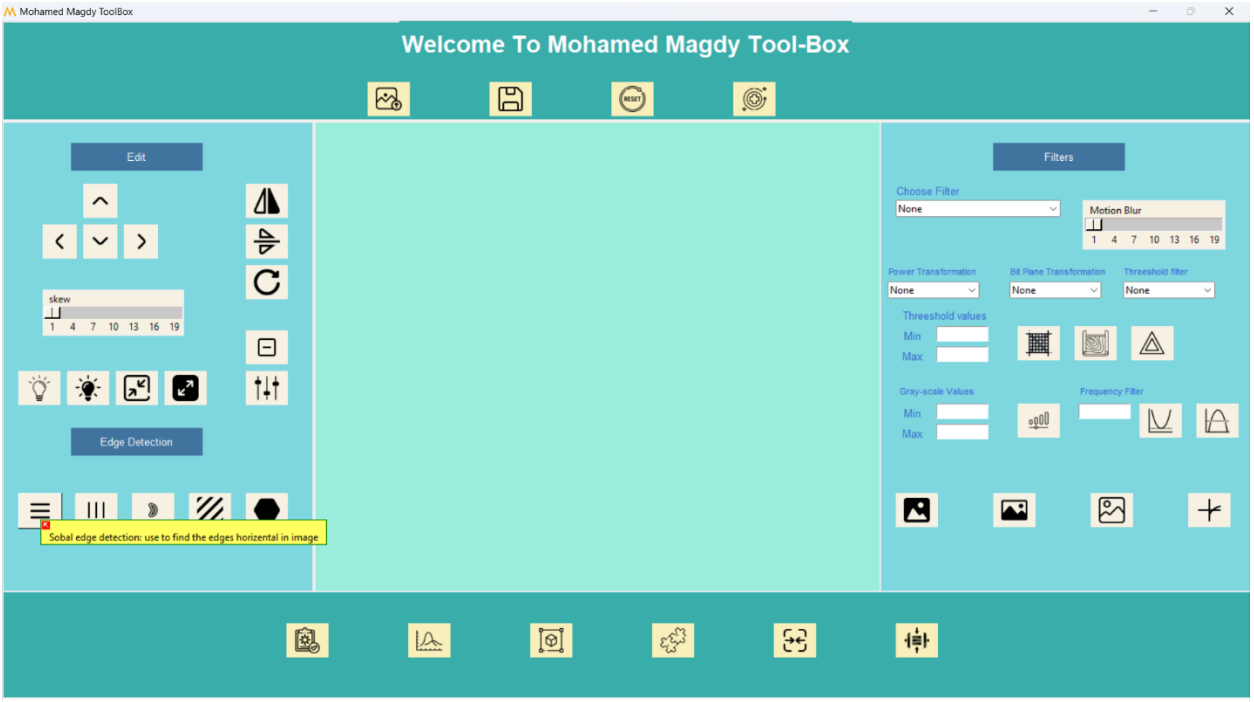


15. Histogram equalization

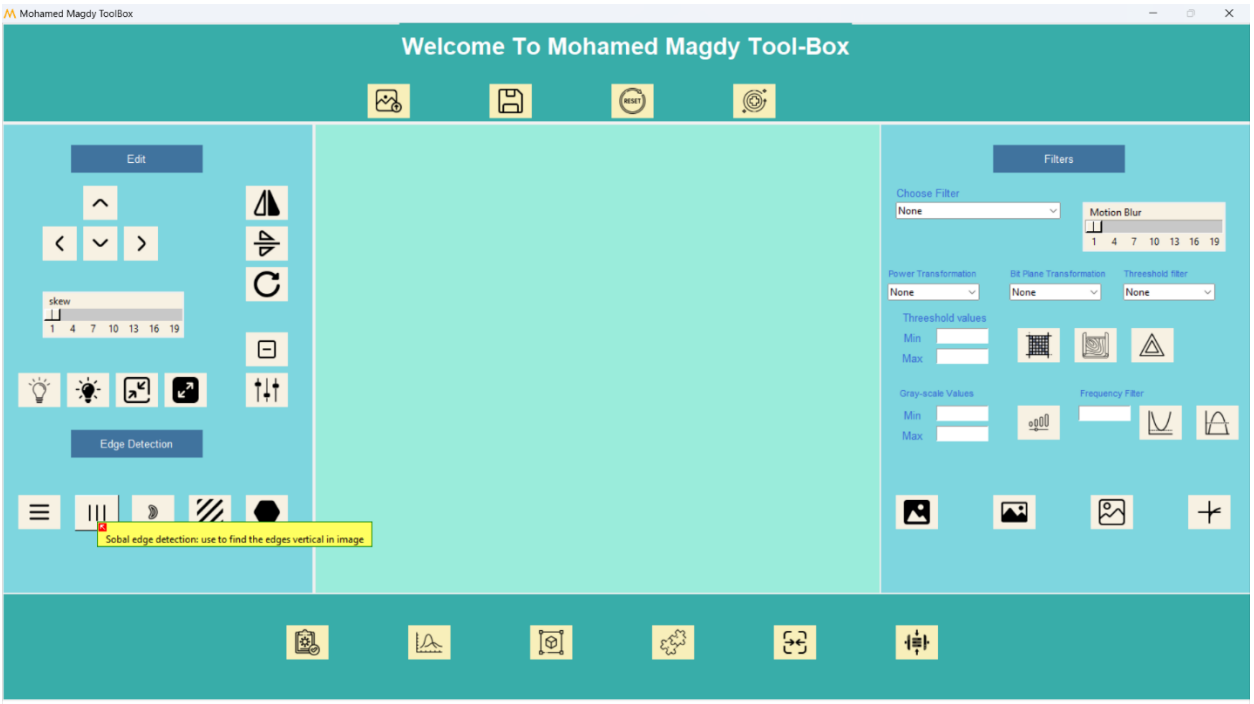


Edge detection part

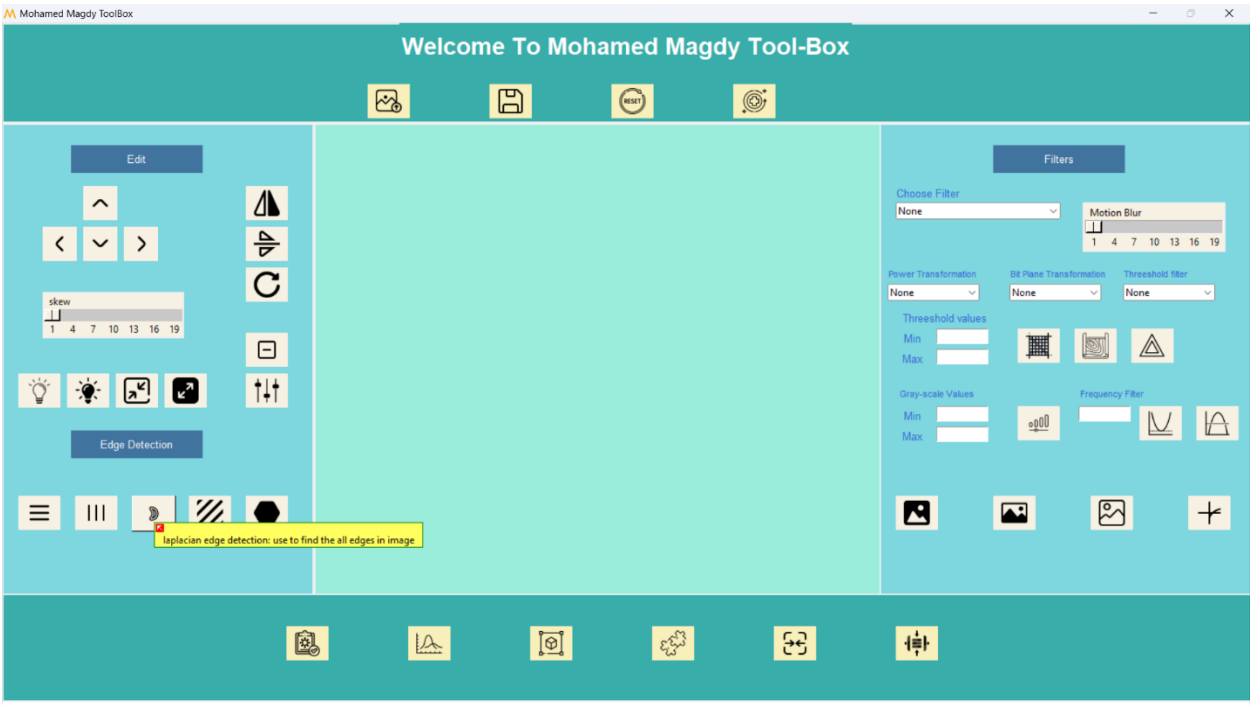
16. Sobel edge horizontal



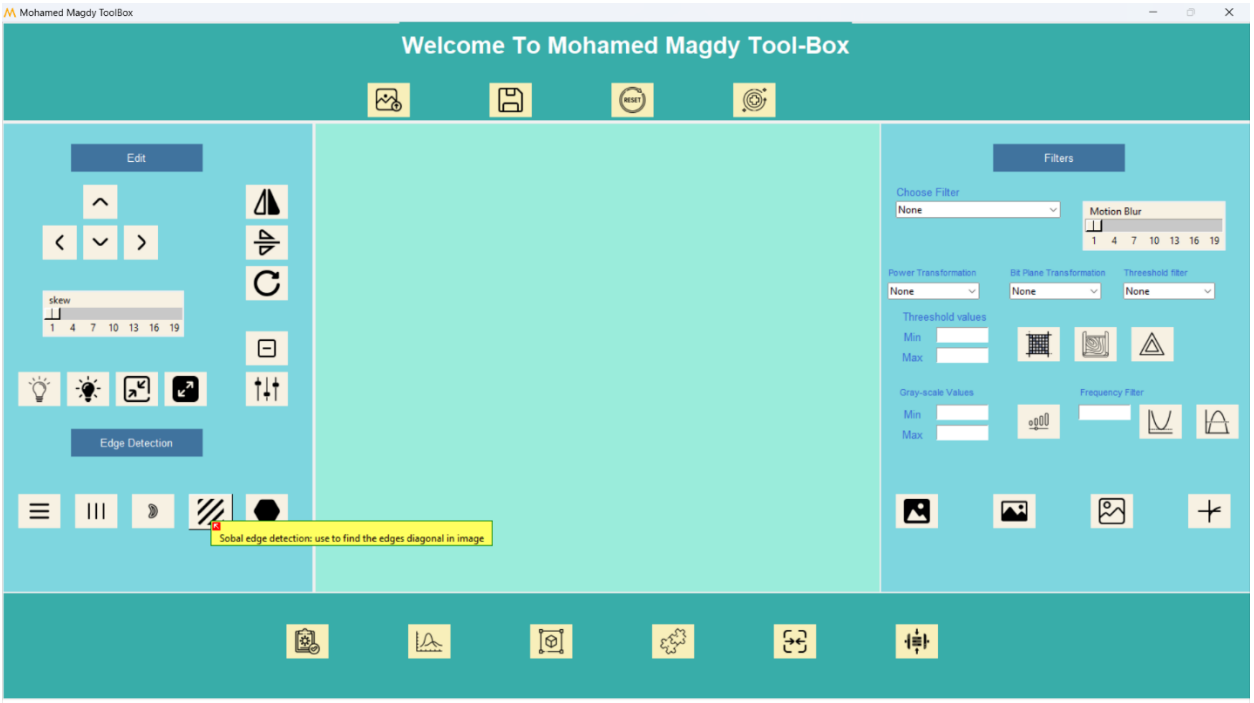
17. Sobel edge vertical



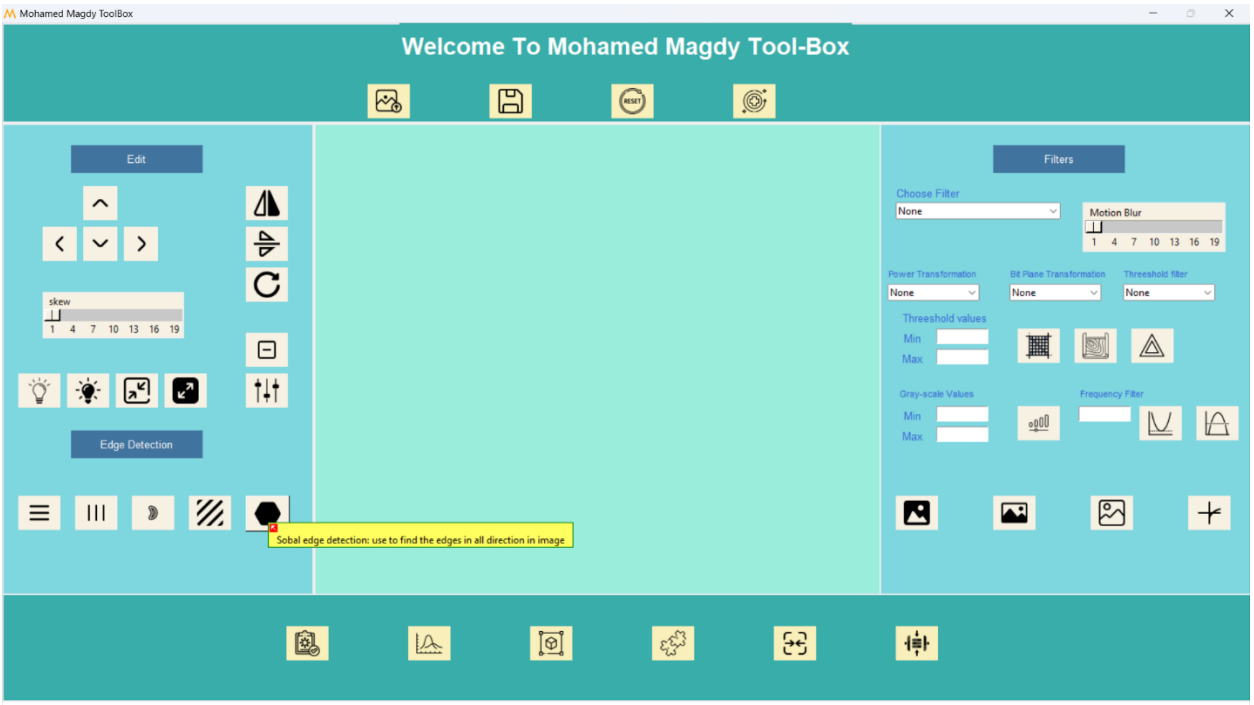
18. Laplacian edge



19. Diagonal Sobel

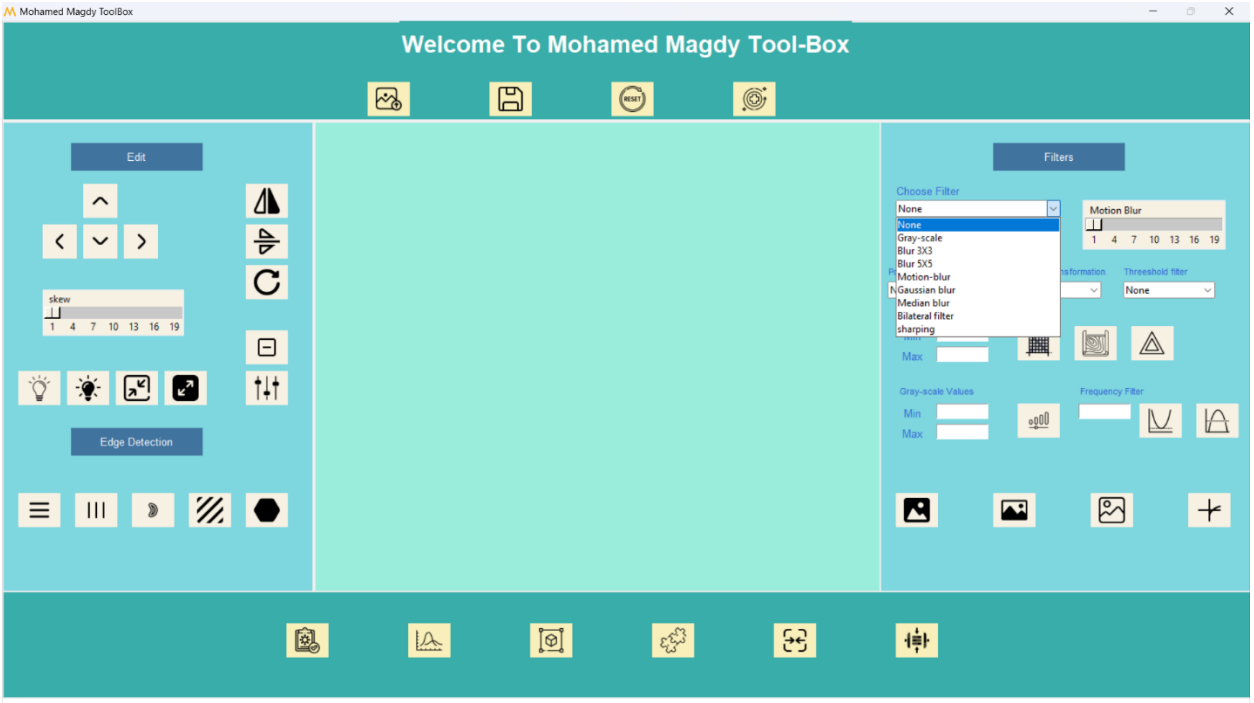


20. Full Sobel

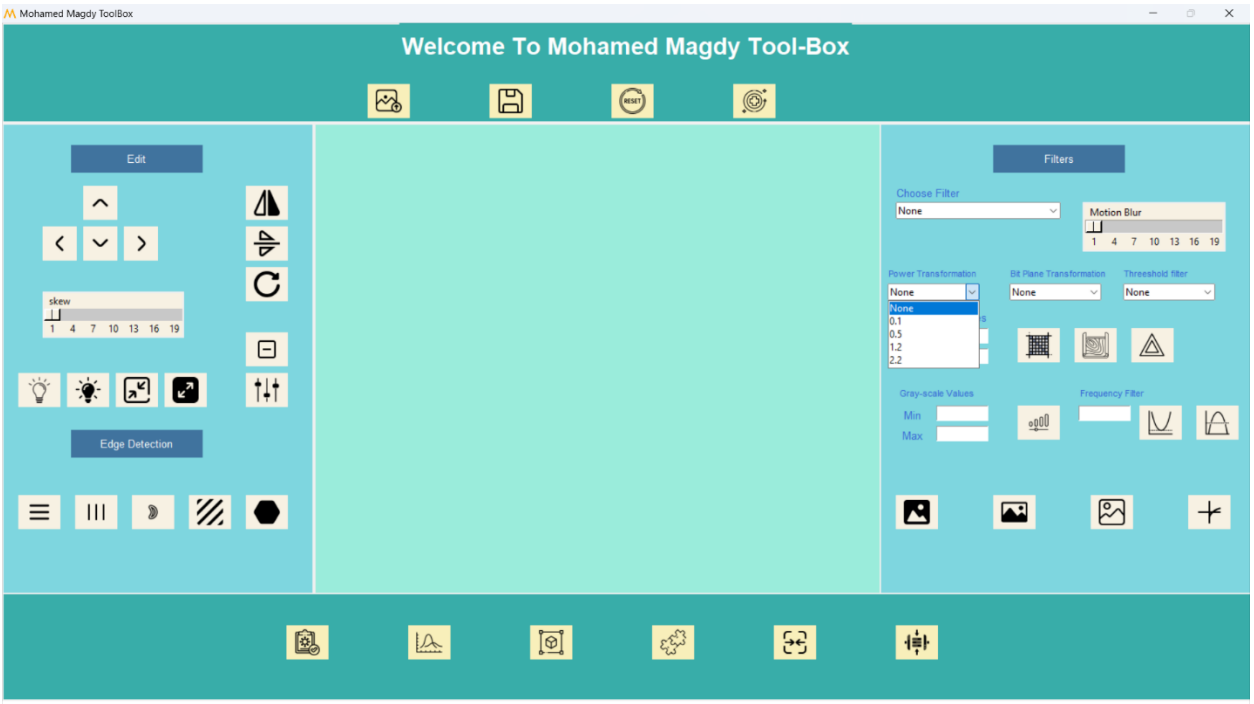


Filter part

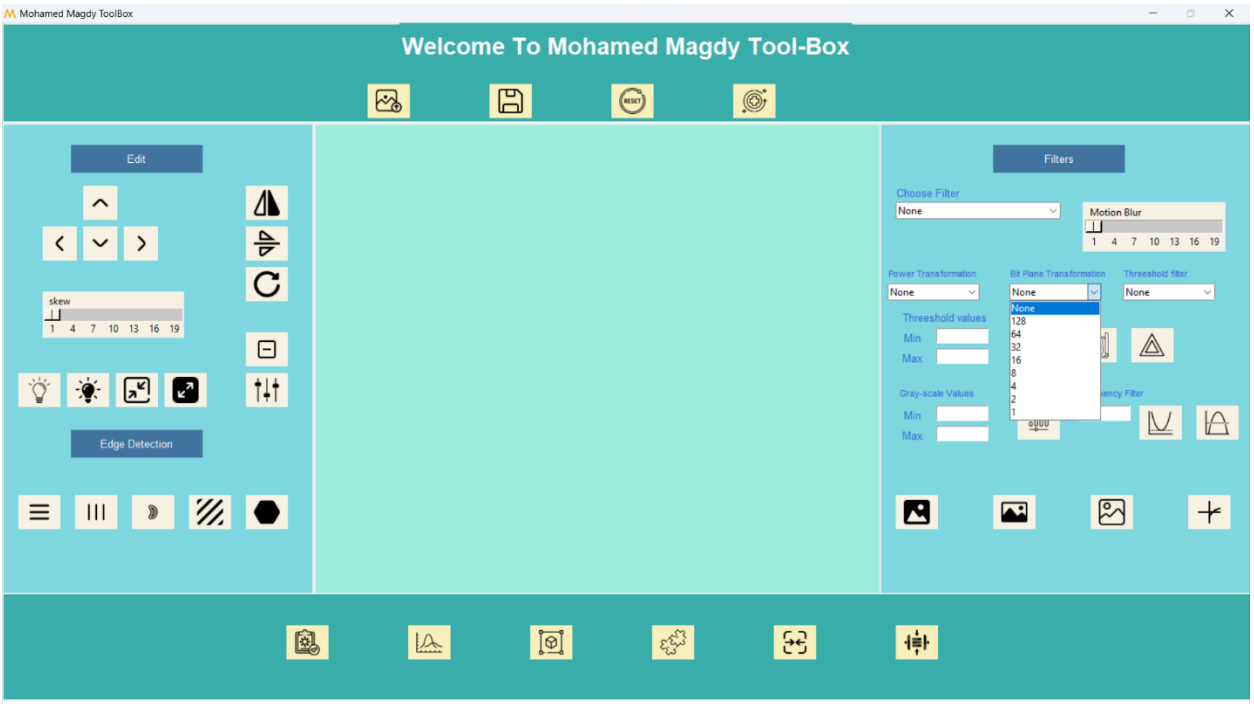
21. All available filters (gray scale, blur 3X3, blur 5X5, Motion blur, Gaussian filter, Median blur, bilateral filter, sharping) and scale to Motion blur filter.



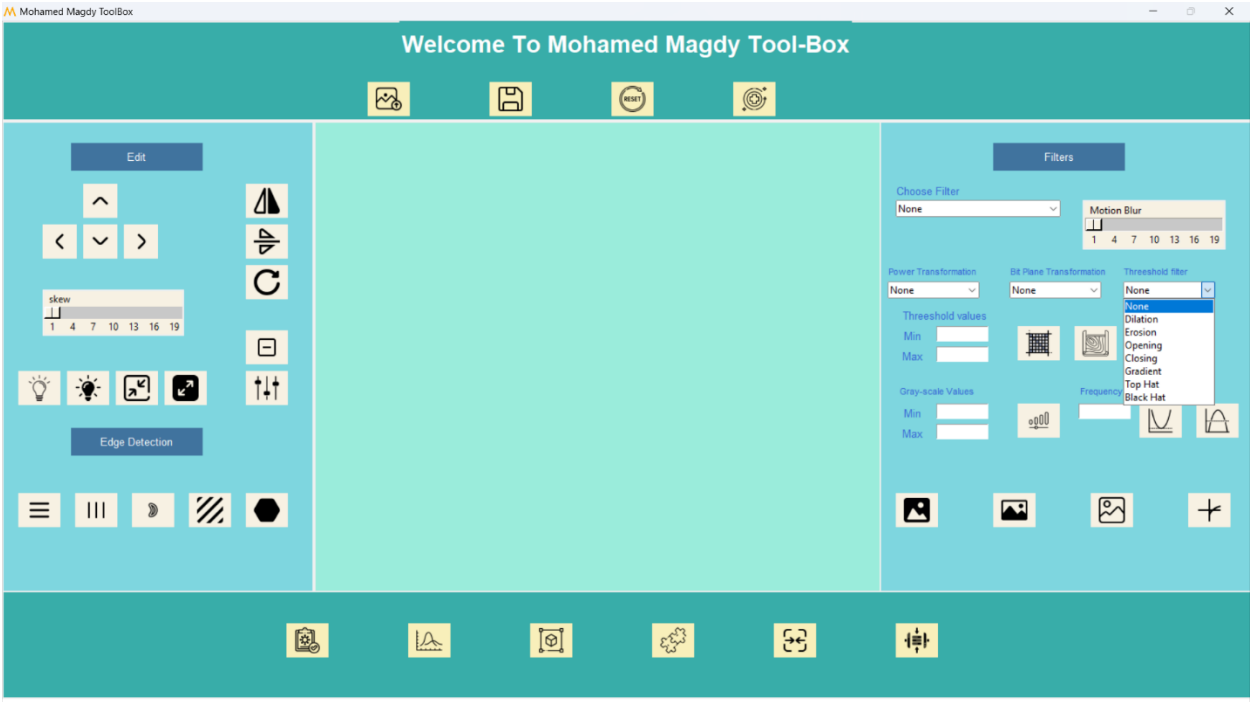
22. Power transformation filter by values (0.1,0.5,1.2,2.2)



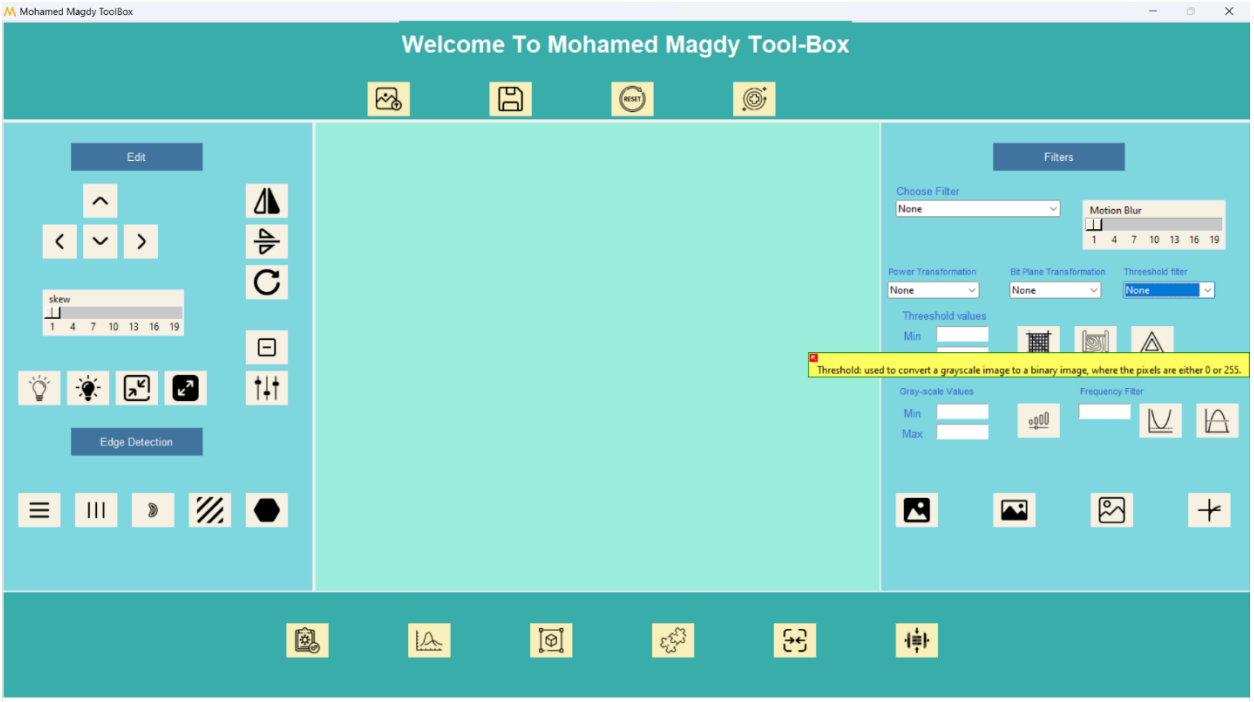
23. Bit plane filter with values (128,64,32,16,8,4,2,1)



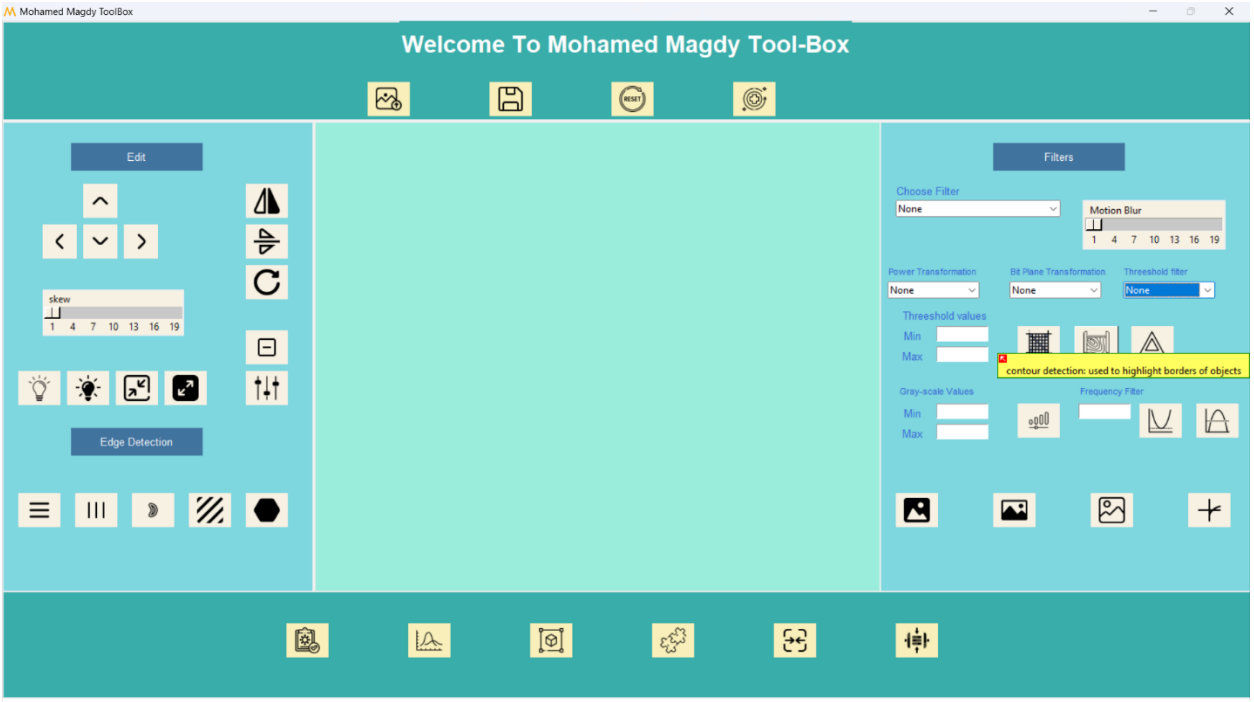
24. Threshold filter with values(dilation , erosion, opening, closing, gradient, top hat, black hat)



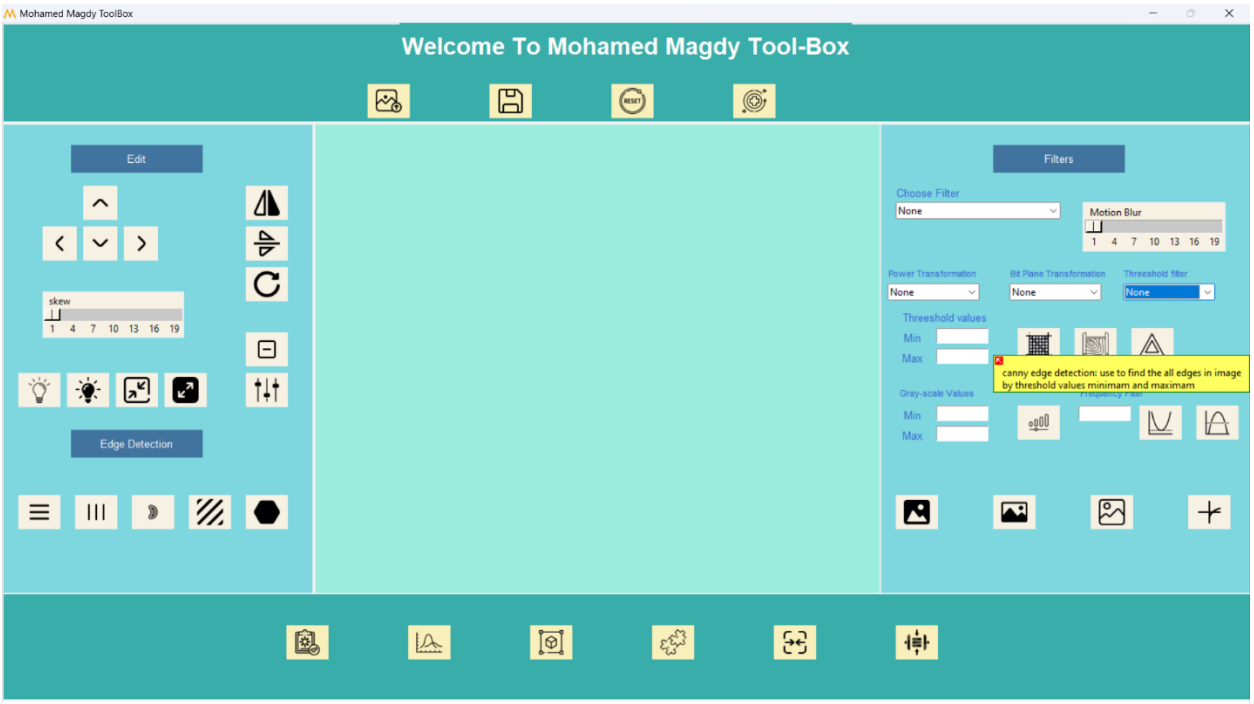
25. Threshold with min and max value



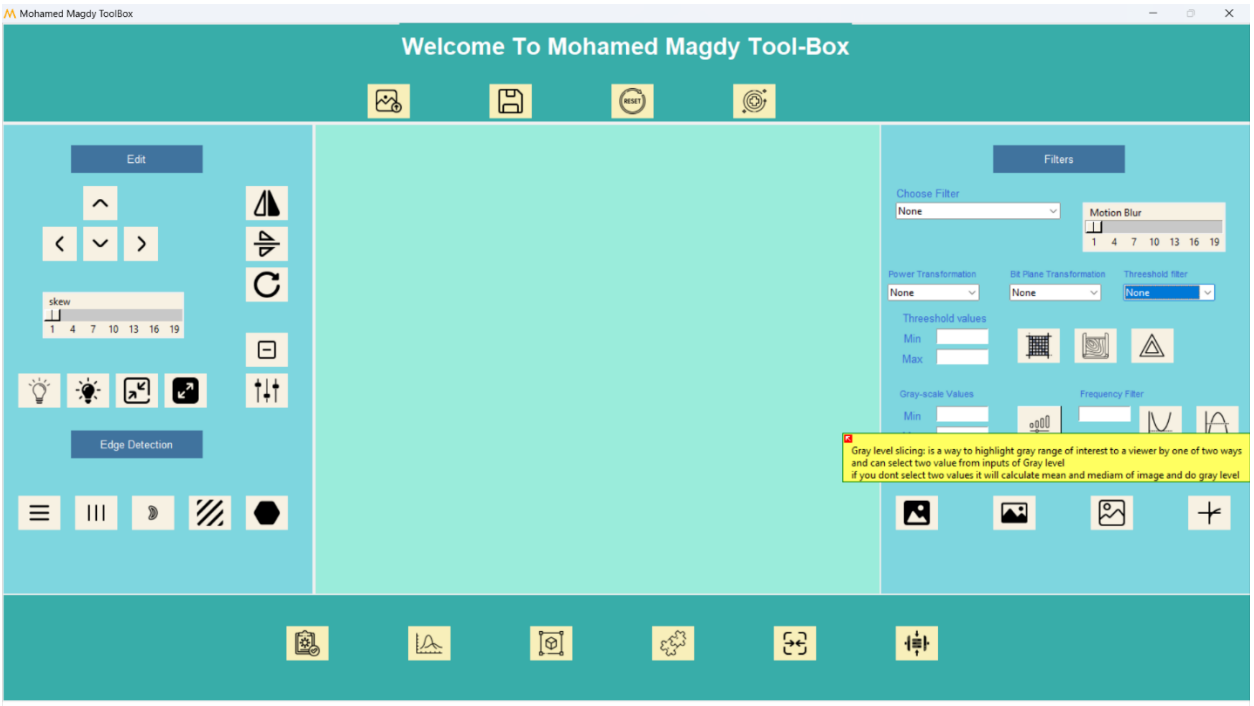
26. Contour



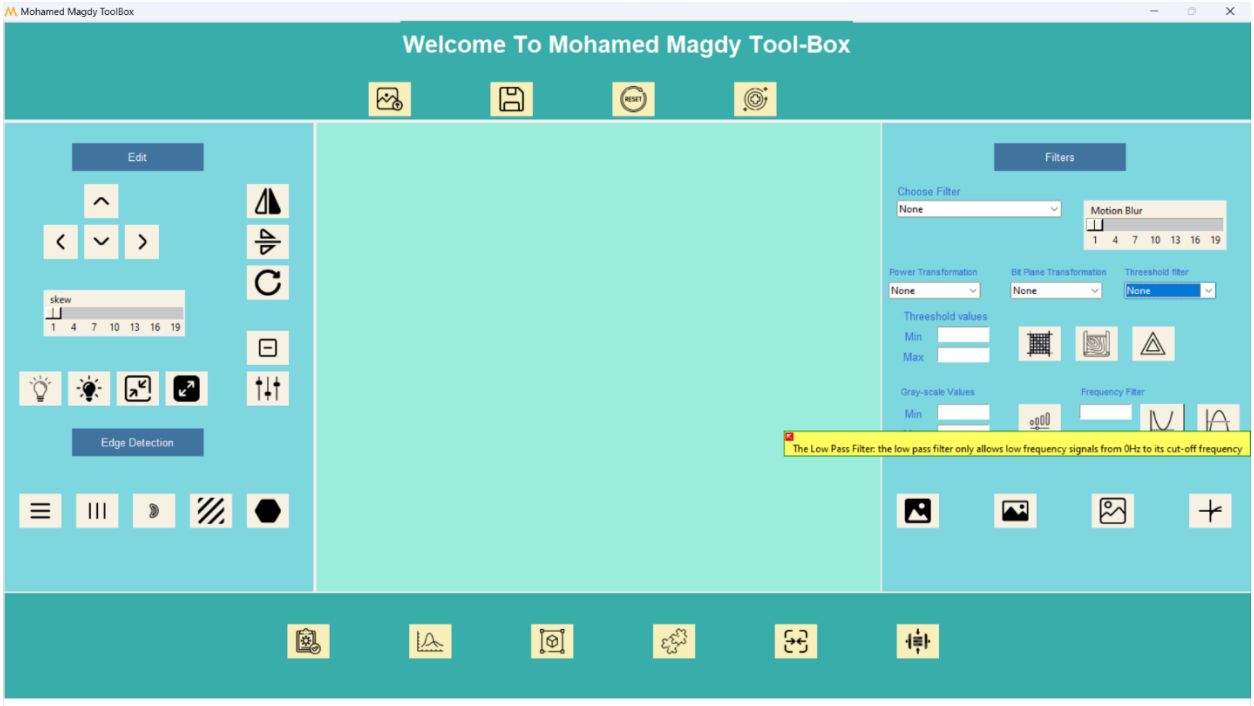
27. Canny edge



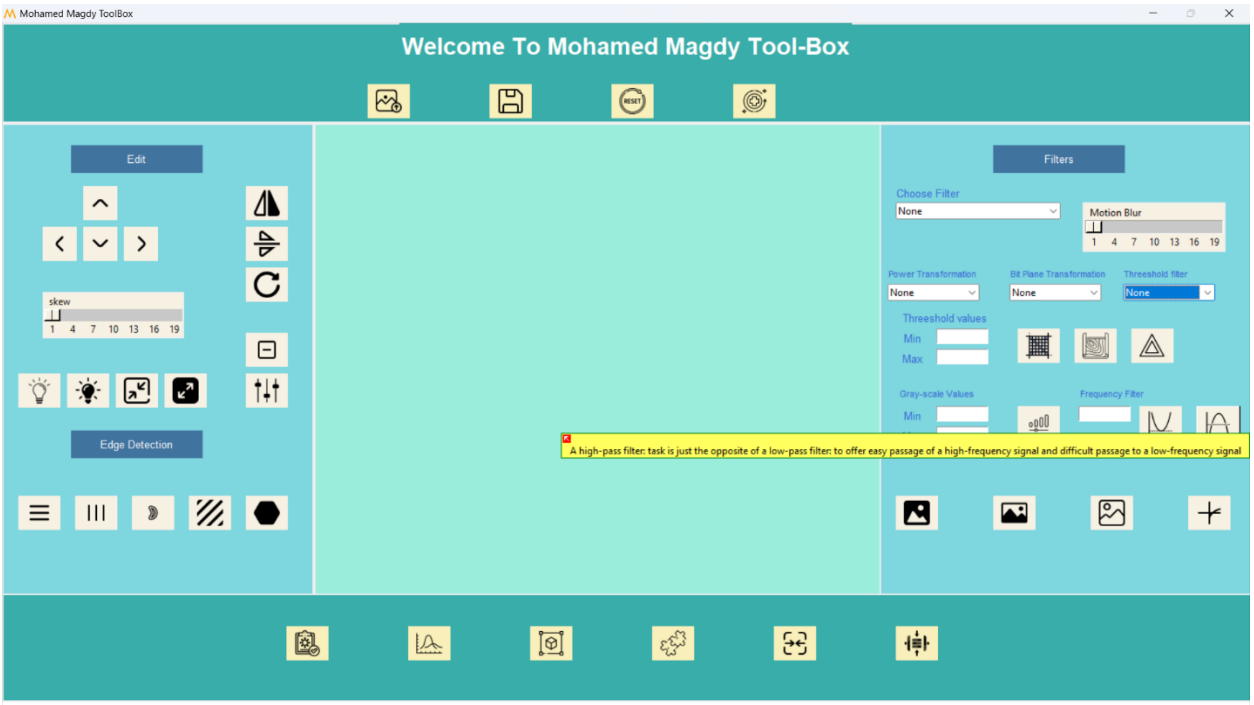
28. Gray level



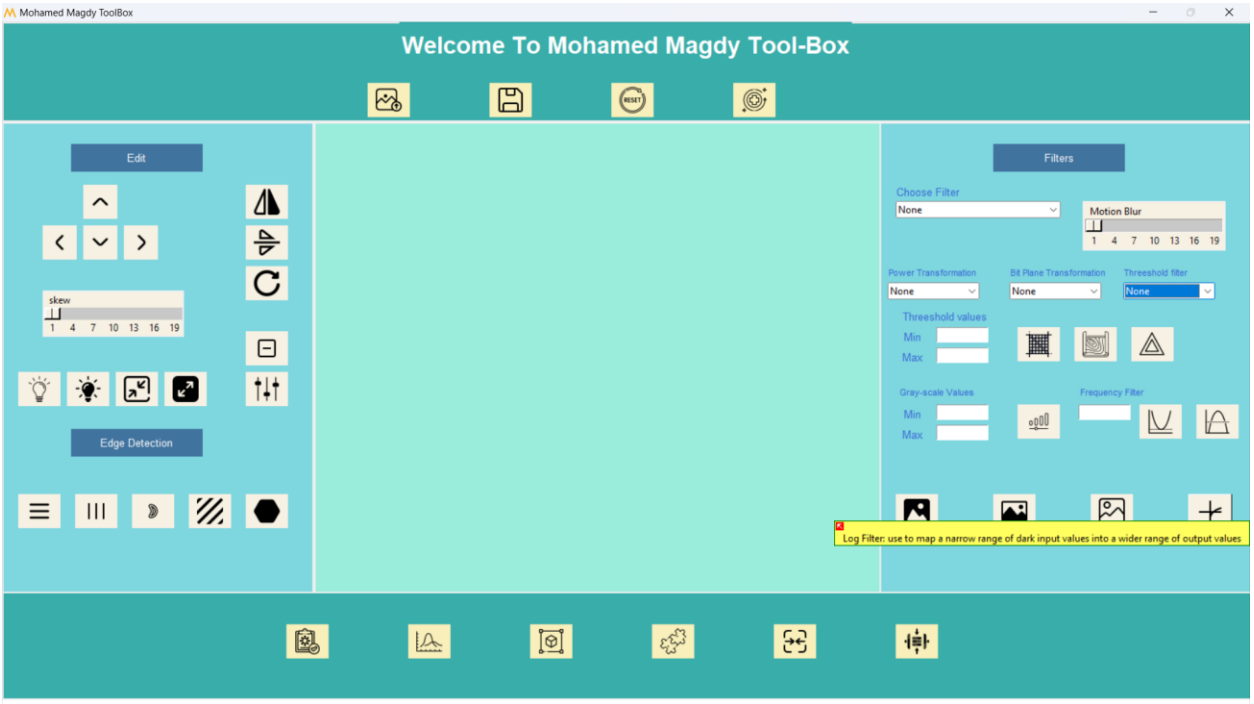
29. Low pass filter



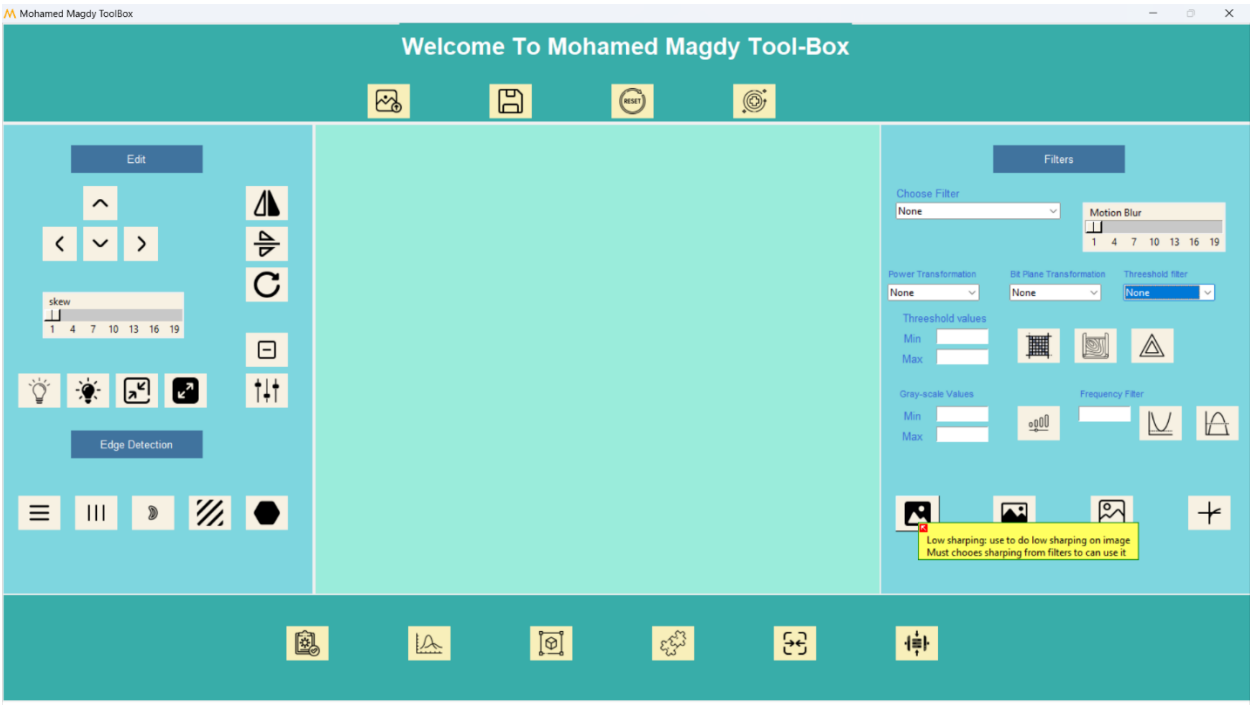
30. High pass filter



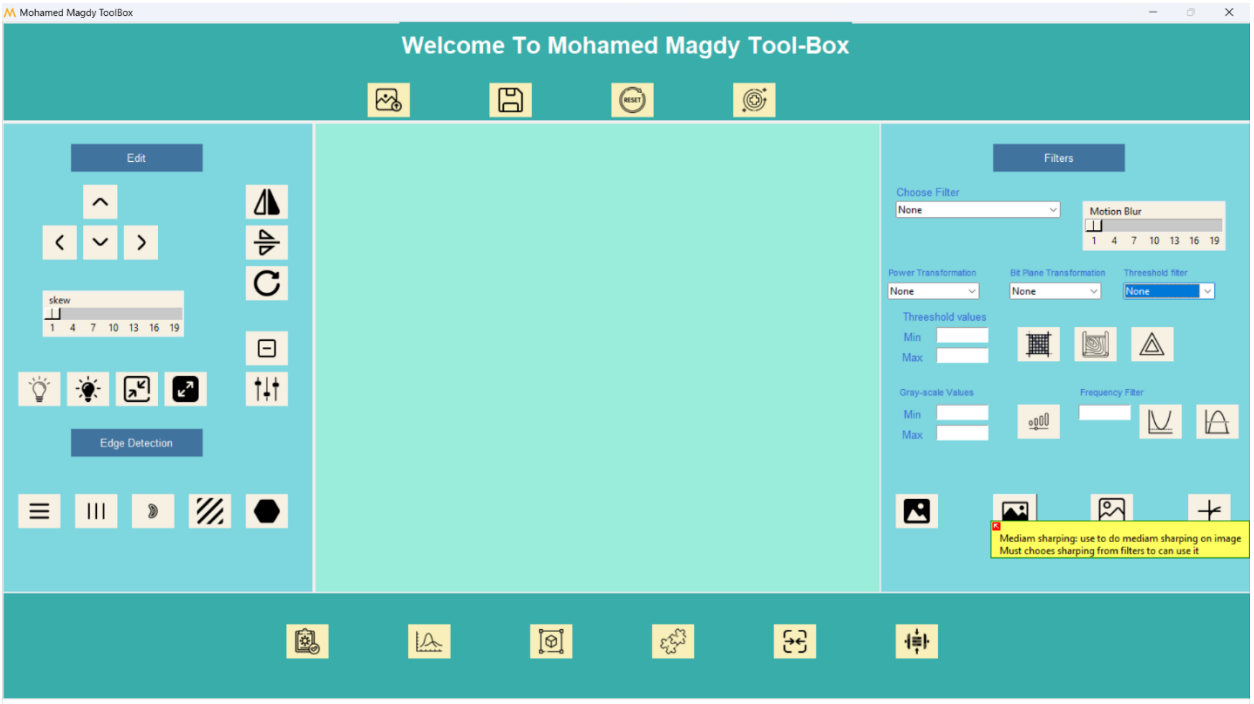
31. Log filter



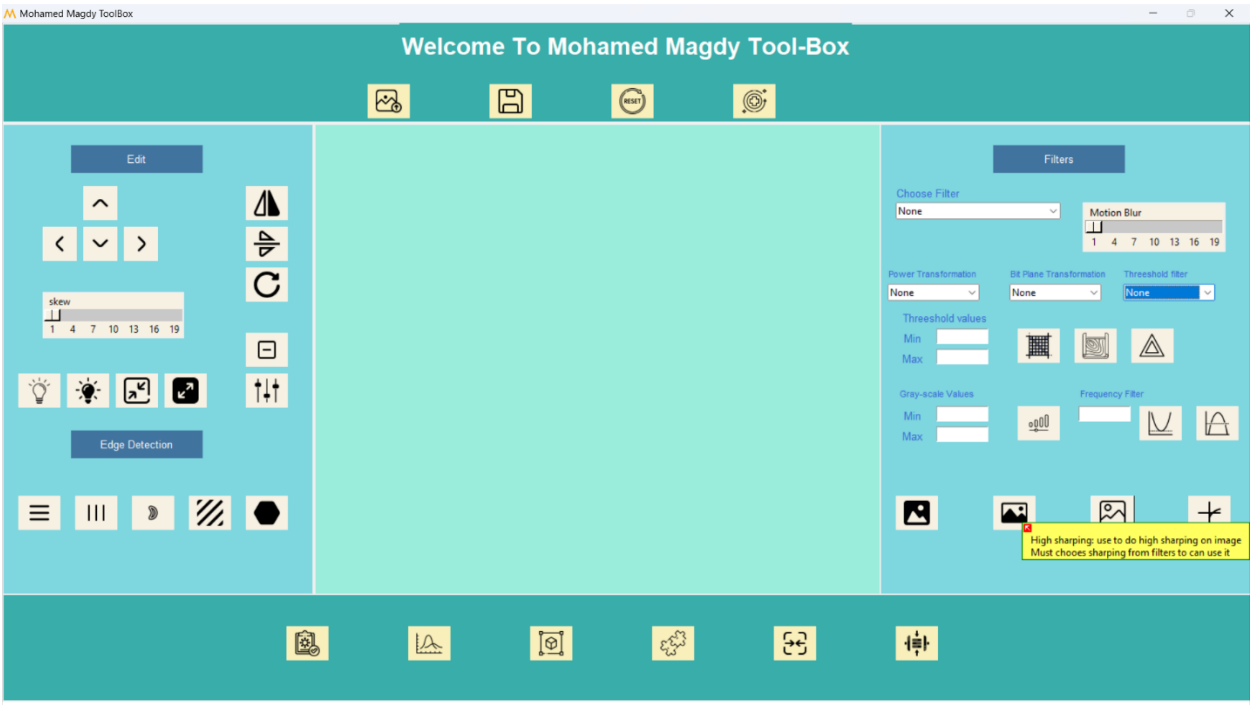
32. Low sharping



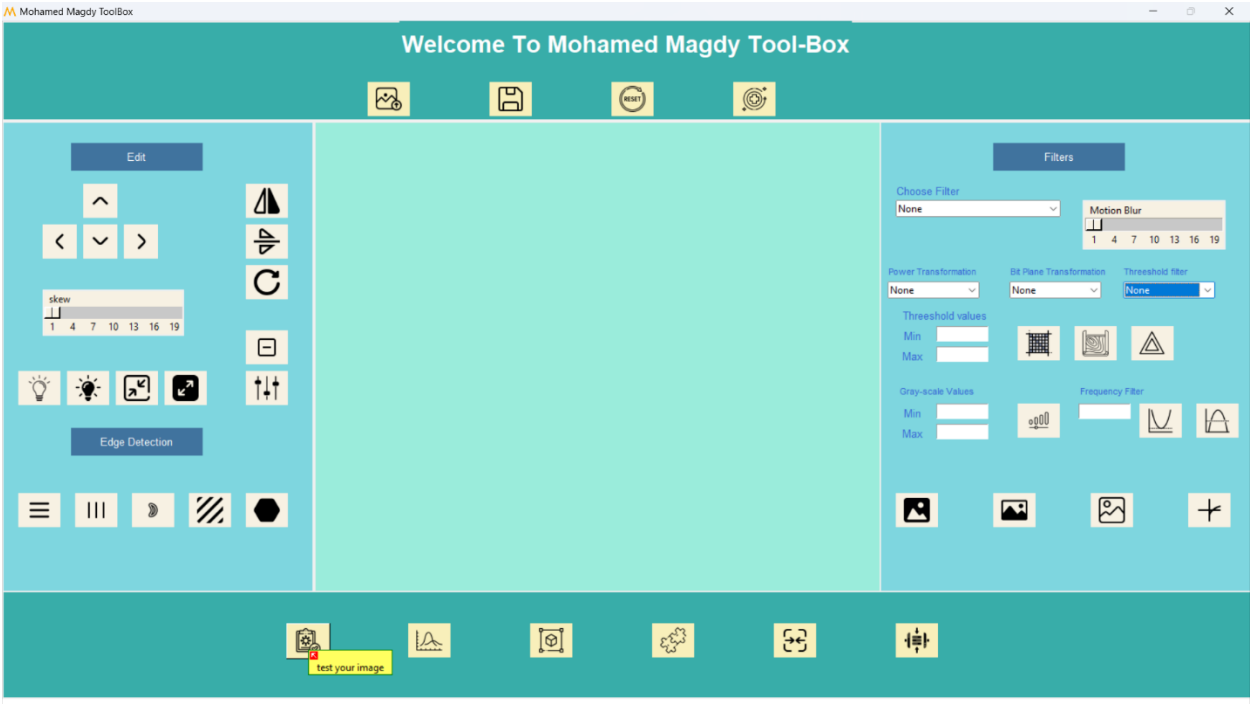
33. Medium sharping



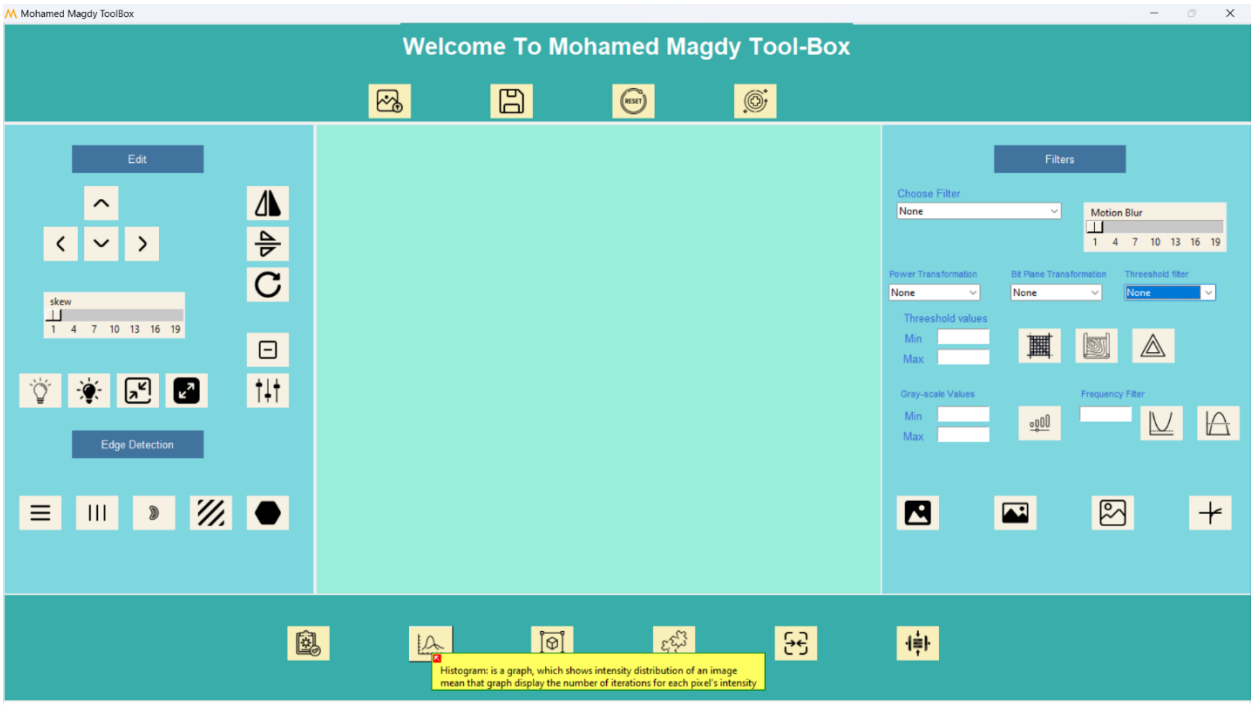
34. High sharping



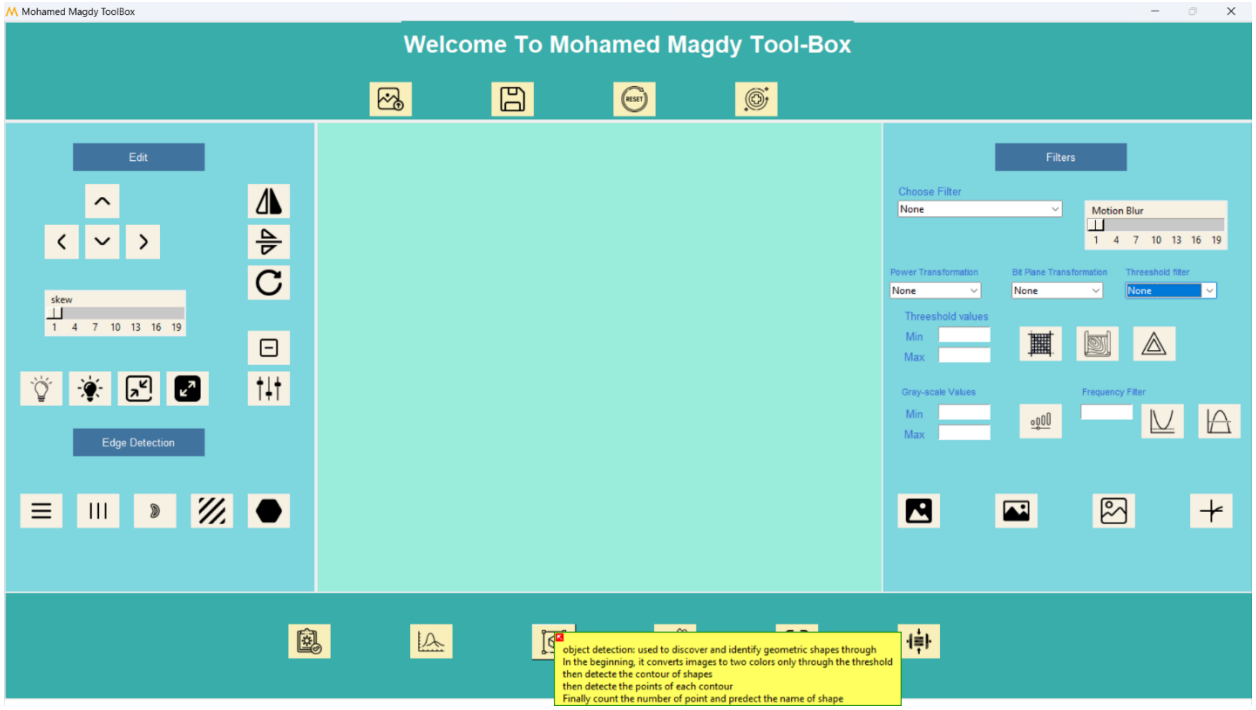
35. Test the image



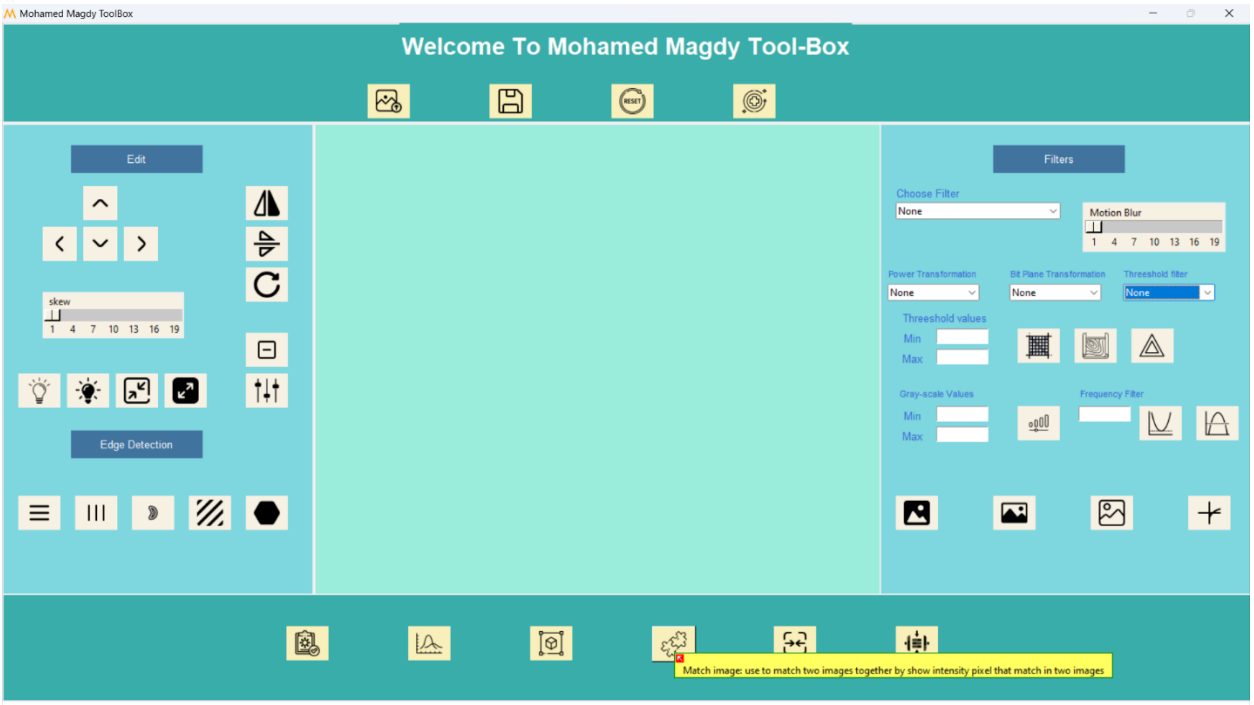
36. Histogram plot



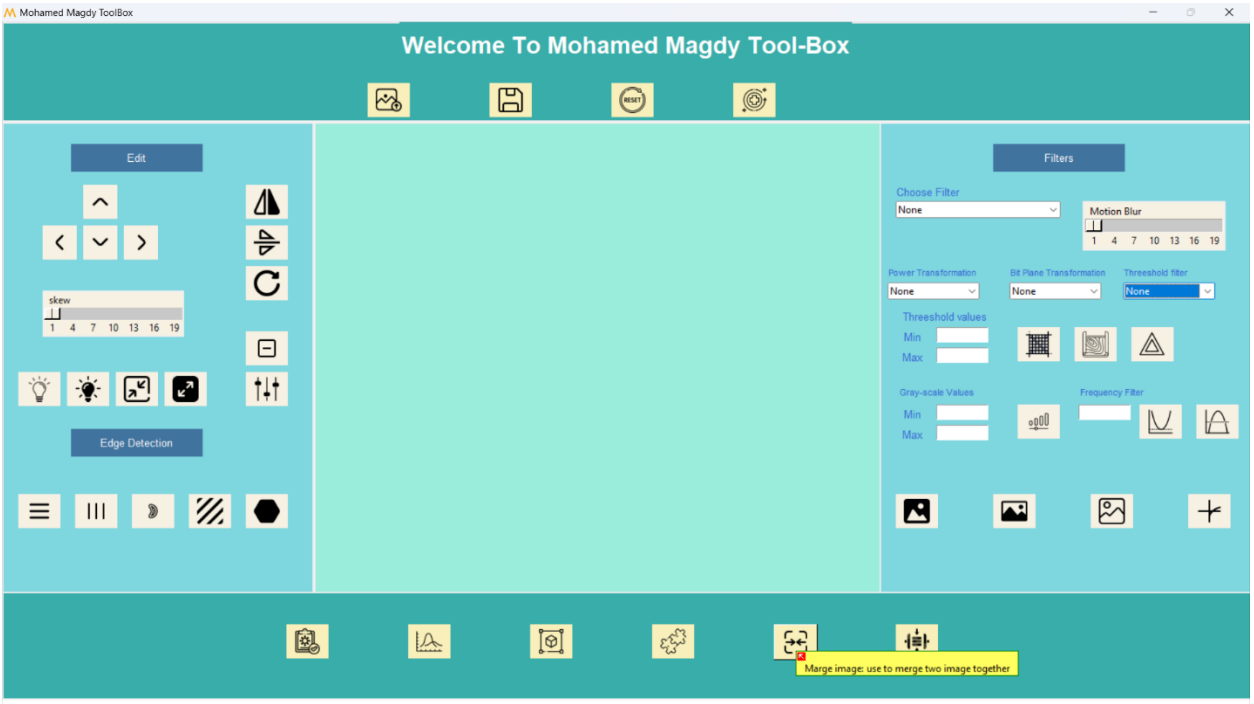
37. Shape detection



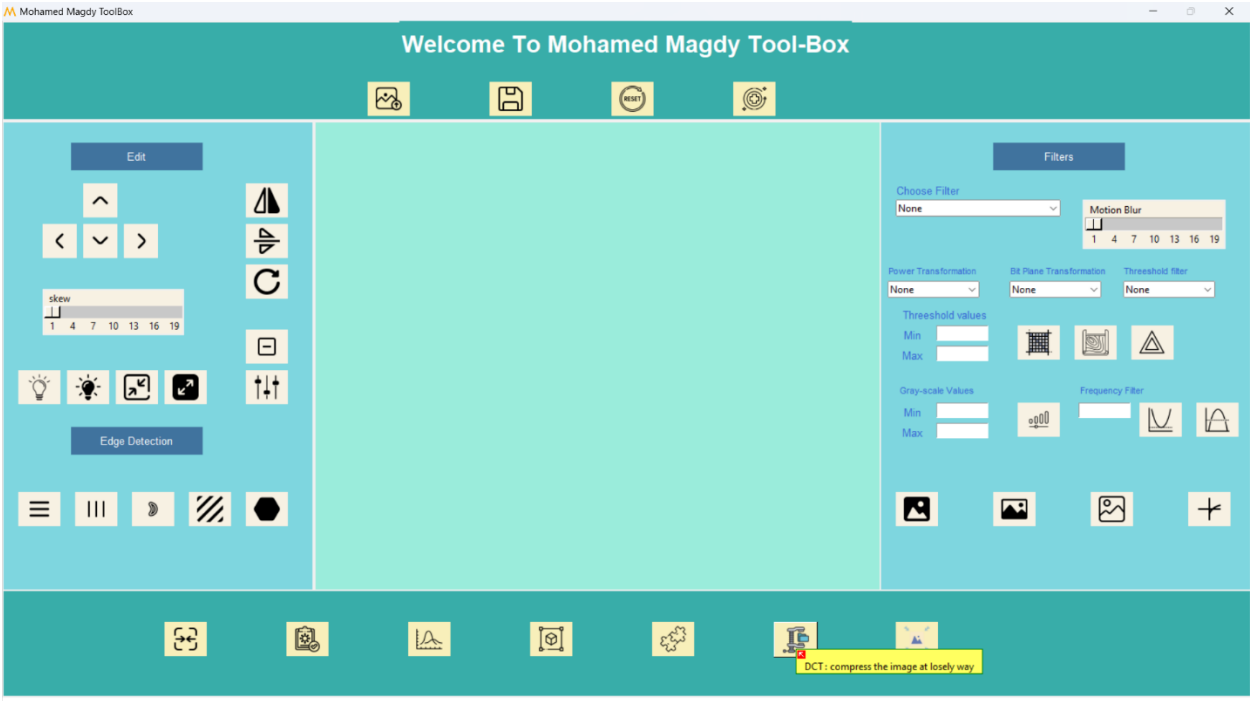
38. Matches two images



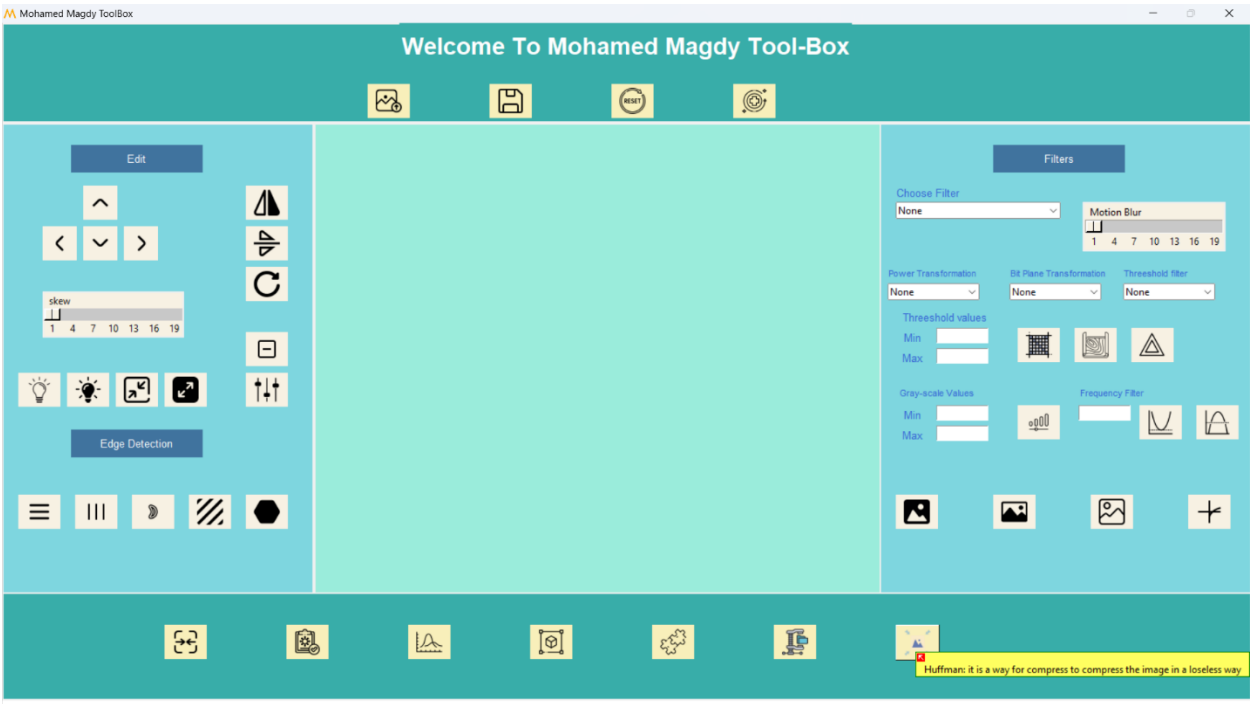
39. Merge two images



40. DCT Compression loosely way



41. Huffman compression is lossless way



End of Documentation.