Praktikum III

(Pengolahan Citra Digital)

(A8 – A11)

A8

import sys  
import cv2  
import numpy as np  
from PyQt5 import QtCore, QtWidgets  
from PyQt5.QtCore import \*  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.uic import loadUi  
  
  
class ShowImage(QMainWindow):  
 def \_\_init\_\_(self):  
 super(ShowImage, self).\_\_init\_\_()  
 loadUi('showgui.ui', self) # Assuming 'showgui.ui' exists  
  
 self.image = None  
 self.button\_loadCitra.clicked.connect(self.load\_image)  
 self.button\_prosesCitra.clicked.connect(self.grayscale)  
  
 # Connect menu actions to respective functions  
 self.actionOperasi\_Pencerahan.triggered.connect(self.brightness)  
 self.actionSimple\_Contrast.triggered.connect(self.contrast)  
 self.actionContrast\_Stretching.triggered.connect(self.contrast\_stretching)  
 self.actionNegative\_Image.triggered.connect(self.negative\_image)  
 self.actionBiner\_Image.triggered.connect(self.binary\_image)  
  
 def load\_image(self):  
 filename, \_ = QFileDialog.getOpenFileName(self, "Open Image", "", "Image Files (\*.jpg \*.jpeg \*.png)")  
 if filename:  
 self.image = cv2.imread(filename)  
 self.display\_image(1) # Update label  
  
 def grayscale(self):  
 if self.image is not None:  
 gray = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)  
 self.image = gray  
 self.display\_image(2) # Update label\_2 (if applicable)  
  
 def brightness(self):  
 try:  
 # Convert to grayscale if necessary  
 if len(self.image.shape) == 3:  
 self.image = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)  
  
 brightness = 80 # Adjust brightness value  
 bright\_img = cv2.convertScaleAbs(self.image, alpha=1, beta=brightness)  
 self.image = bright\_img  
 self.display\_image(1) # Update label  
 except Exception as e:  
 print(f"Brightness Error: {e}") # Handle potential errors  
  
 def contrast(self):  
 try:  
 # Convert to grayscale if necessary  
 if len(self.image.shape) == 3:  
 self.image = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)  
  
 contrast = 1.7 # Adjust contrast value  
 contrast\_img = cv2.convertScaleAbs(self.image, alpha=contrast, beta=0)  
 self.image = contrast\_img  
 self.display\_image(1) # Update label  
 except Exception as e:  
 print(f"Contrast Error: {e}") # Handle potential errors  
  
 def contrast\_stretching(self):  
 if self.image is not None:  
 min\_val = np.min(self.image)  
 max\_val = np.max(self.image)  
 stretched\_img = cv2.normalize(self.image, None, 0, 255, cv2.NORM\_MINMAX)  
 self.image = stretched\_img  
 self.display\_image(1) # Update label  
  
 def negative\_image(self):  
 if self.image is not None:  
 negative\_img = 255 - self.image  
 self.image = negative\_img  
 self.display\_image(1) # Update label  
  
 def binary\_image(self):  
 if self.image is not None:  
 if len(self.image.shape) == 3: # Check for color image  
 gray = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray = self.image  
  
 # Apply binary threshold  
 threshold\_value, binary\_img = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)  
  
 # Optional: Create a comparison image (uncomment if desired)  
 # H, W = gray.shape[:2]  
 # comparison  
  
 # Compare pixel values before and after thresholding  
 H, W = gray.shape[:2]  
 comparison\_img = np.zeros((H, W, 3), np.uint8)  
 for i in range(H):  
 for j in range(W):  
 original\_value = gray[i, j]  
 binary\_value = binary\_img[i, j]  
 comparison\_img[i, j] = [original\_value, binary\_value, 0]  
 self.image = binary\_img  
 self.display\_image(1)  
  
 def display\_image(self, window=1):  
 if self.image is not None:  
 qformat = QImage.Format\_Indexed8  
 if len(self.image.shape) == 3:  
 if self.image.shape[2] == 4:  
 qformat = QImage.Format\_RGBA8888  
 else:  
 qformat = QImage.Format\_RGB888  
 img = QImage(self.image, self.image.shape[1], self.image.shape[0], self.image.strides[0], qformat)  
 img = img.rgbSwapped()  
 if window == 1:  
 self.label.setPixmap(QPixmap.fromImage(img))  
 elif window == 2:  
 self.label\_2.setPixmap(QPixmap.fromImage(img))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QtWidgets.QApplication(sys.argv)  
 window = ShowImage()  
 window.setWindowTitle('A2Praktek')  
 window.show()

sys.exit(app.exec\_())

A screenshot of a computer

Description automatically generated

A9

import sys  
import cv2  
import numpy as np  
from PyQt5 import QtCore, QtWidgets  
from PyQt5.QtCore import \*  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.uic import loadUi  
from matplotlib import pyplot as plt  
  
class ShowImage(QMainWindow):  
 def \_\_init\_\_(self):  
 super(ShowImage, self).\_\_init\_\_()  
 loadUi('showgui.ui', self)  
 self.Image = None  
 self.button\_loadCitra.clicked.connect(self.fungsi)  
 self.button\_prosesCitra.clicked.connect(self.grayscale)  
 self.actionOperasi\_Pencerahan.triggered.connect(self.brightness)  
 self.actionSimple\_Contrast.triggered.connect(self.contrast)  
 self.actionContrast\_Stretching.triggered.connect(self.contrastStretching)  
 self.actionNegative\_Image.triggered.connect(self.negativeImage)  
 self.actionBiner\_Image.triggered.connect(self.binaryImage)  
 self.actionHistogram\_Gray.triggered.connect(self.grayHistogram)  
  
 def fungsi(self):  
 self.Image = cv2.imread('BANG.jpeg')  
 self.displayImage(1)  
  
 def grayscale(self):  
 if self.Image is not None:  
 gray = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 self.Image = gray  
 self.displayImage(2)  
 self.grayHistogram() # Display histogram after converting to grayscale  
  
 def brightness(self):  
 try:  
 self.Image = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 except:  
 pass  
 brightness = 80  
 bright\_img = cv2.convertScaleAbs(self.Image, alpha=1, beta=brightness)  
 self.Image = bright\_img  
 self.displayImage(1)  
  
 def contrast(self):  
 try:  
 self.Image = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 except:  
 pass  
 contrast = 1.7  
 contrast\_img = cv2.convertScaleAbs(self.Image, alpha=contrast, beta=0)  
 self.Image = contrast\_img  
 self.displayImage(1)  
  
 def contrastStretching(self):  
 if self.Image is not None:  
 min\_val = np.min(self.Image)  
 max\_val = np.max(self.Image)  
 stretched\_img = cv2.normalize(self.Image, None, 0, 255, cv2.NORM\_MINMAX)  
 self.Image = stretched\_img  
 self.displayImage(1)  
  
 def negativeImage(self):  
 if self.Image is not None:  
 negative\_img = 255 - self.Image  
 self.Image = negative\_img  
 self.displayImage(1)  
  
 def binaryImage(self):  
 if self.Image is not None:  
 if len(self.Image.shape) == 3: # Check if the image is colored  
 gray = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray = self.Image  
 # Apply binary threshold  
 threshold\_value, binary\_img = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)  
 # Compare pixel values before and after thresholding  
 H, W = gray.shape[:2]  
 comparison\_img = np.zeros((H, W, 3), np.uint8)  
 for i in range(H):  
 for j in range(W):  
 original\_value = gray[i, j]  
 binary\_value = binary\_img[i, j]  
 comparison\_img[i, j] = [original\_value, binary\_value, 0]  
 self.Image = binary\_img  
 self.displayImage(1)  
  
 def grayHistogram(self):  
 if self.Image is not None:  
 print("Generating histogram from grayscale image")  
 plt.hist(self.Image.ravel(), 255, [0, 255])  
 plt.show()  
  
 def displayImage(self, window=1):  
 if self.Image is not None:  
 qformat = QImage.Format\_Indexed8  
 if len(self.Image.shape) == 3:  
 if self.Image.shape[2] == 4:  
 qformat = QImage.Format\_RGBA8888  
 else:  
 qformat = QImage.Format\_RGB888  
 img = QImage(self.Image, self.Image.shape[1], self.Image.shape[0], self.Image.strides[0], qformat)  
 img = img.rgbSwapped()  
 if window == 1:  
 self.label.setPixmap(QPixmap.fromImage(img))  
 elif window == 2:  
 self.label\_2.setPixmap(QPixmap.fromImage(img))  
  
app = QtWidgets.QApplication(sys.argv)  
window = ShowImage()  
window.setWindowTitle('A2Praktek')  
window.show()  
sys.exit(app.exec\_())

A screen shot of a computer

Description automatically generated

A10

import sys  
import cv2  
import numpy as np  
from PyQt5 import QtCore, QtWidgets  
from PyQt5.QtCore import \*  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.uic import loadUi  
from matplotlib import pyplot as plt  
  
class ShowImage(QMainWindow):  
 def \_\_init\_\_(self):  
 super(ShowImage, self).\_\_init\_\_()  
 loadUi('showgui.ui', self)  
 self.Image = None  
 self.button\_loadCitra.clicked.connect(self.fungsi)  
 self.button\_prosesCitra.clicked.connect(self.grayscale)  
 self.actionOperasi\_Pencerahan.triggered.connect(self.brightness)  
 self.actionSimple\_Contrast.triggered.connect(self.contrast)  
 self.actionContrast\_Stretching.triggered.connect(self.contrastStretching)  
 self.actionNegative\_Image.triggered.connect(self.negativeImage)  
 self.actionBiner\_Image.triggered.connect(self.binaryImage)  
 self.actionHistogram\_Gray.triggered.connect(self.grayHistogram)  
 self.actionHistogram\_RGB.triggered.connect(self.rgbHistogram)  
  
 def fungsi(self):  
 self.Image = cv2.imread('BANG.jpeg')  
 self.displayImage(1)  
 self.rgbHistogram()  
  
 def grayscale(self):  
 if self.Image is not None:  
 gray = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 self.Image = gray  
 self.displayImage(2)  
 self.grayHistogram()  
  
 def brightness(self):  
 try:  
 self.Image = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 except:  
 pass  
 brightness = 80  
 bright\_img = cv2.convertScaleAbs(self.Image, alpha=1, beta=brightness)  
 self.Image = bright\_img  
 self.displayImage(1)  
  
 def contrast(self):  
 try:  
 self.Image = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 except:  
 pass  
 contrast = 1.7  
 contrast\_img = cv2.convertScaleAbs(self.Image, alpha=contrast, beta=0)  
 self.Image = contrast\_img  
 self.displayImage(1)  
  
 def contrastStretching(self):  
 if self.Image is not None:  
 min\_val = np.min(self.Image)  
 max\_val = np.max(self.Image)  
 stretched\_img = cv2.normalize(self.Image, None, 0, 255, cv2.NORM\_MINMAX)  
 self.Image = stretched\_img  
 self.displayImage(1)  
  
 def negativeImage(self):  
 if self.Image is not None:  
 negative\_img = 255 - self.Image  
 self.Image = negative\_img  
 self.displayImage(1)  
  
 def binaryImage(self):  
 if self.Image is not None:  
 if len(self.Image.shape) == 3: # Check if the image is colored  
 gray = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray = self.Image  
 # Apply binary threshold  
 threshold\_value, binary\_img = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)  
 # Compare pixel values before and after thresholding  
 H, W = gray.shape[:2]  
 comparison\_img = np.zeros((H, W, 3), np.uint8)  
 for i in range(H):  
 for j in range(W):  
 original\_value = gray[i, j]  
 binary\_value = binary\_img[i, j]  
 comparison\_img[i, j] = [original\_value, binary\_value, 0]  
 self.Image = binary\_img  
 self.displayImage(1)  
  
 def grayHistogram(self):  
 if self.Image is not None:  
 plt.hist(self.Image.ravel(), 255, [0, 255])  
 plt.show()  
  
 def rgbHistogram(self):  
 if self.Image is not None:  
 color = ('b', 'g', 'r')  
 for i, col in enumerate(color):  
 histo = cv2.calcHist([self.Image], [i], None, [256], [0, 256])  
 plt.plot(histo, color=col)  
 plt.xlim([0, 256])  
 plt.show()  
  
 def displayImage(self, window=1):  
 if self.Image is not None:  
 qformat = QImage.Format\_Indexed8  
 if len(self.Image.shape) == 3:  
 if self.Image.shape[2] == 4:  
 qformat = QImage.Format\_RGBA8888  
 else:  
 qformat = QImage.Format\_RGB888  
 img = QImage(self.Image, self.Image.shape[1], self.Image.shape[0], self.Image.strides[0], qformat)  
 img = img.rgbSwapped()  
 if window == 1:  
 self.label.setPixmap(QPixmap.fromImage(img))  
 elif window == 2:  
 self.label\_2.setPixmap(QPixmap.fromImage(img))  
  
app = QtWidgets.QApplication(sys.argv)  
window = ShowImage()  
window.setWindowTitle('A2Praktek')  
window.show()  
sys.exit(app.exec\_())

A screenshot of a computer

Description automatically generated

A11

import sys  
import cv2  
import numpy as np  
from PyQt5 import QtCore, QtWidgets  
from PyQt5.QtCore import \*  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.uic import loadUi  
from matplotlib import pyplot as plt  
  
class ShowImage(QMainWindow):  
 def \_\_init\_\_(self):  
 super(ShowImage, self).\_\_init\_\_()  
 loadUi('showgui.ui', self)  
 self.Image = None  
 self.button\_loadCitra.clicked.connect(self.fungsi)  
 self.button\_prosesCitra.clicked.connect(self.grayscale)  
 self.actionOperasi\_Pencerahan.triggered.connect(self.brightness)  
 self.actionSimple\_Contrast.triggered.connect(self.contrast)  
 self.actionContrast\_Stretching.triggered.connect(self.contrastStretching)  
 self.actionNegative\_Image.triggered.connect(self.negativeImage)  
 self.actionBiner\_Image.triggered.connect(self.binaryImage)  
 self.actionHistogram\_Gray.triggered.connect(self.grayHistogram)  
 self.actionHistogram\_RGB.triggered.connect(self.rgbHistogram)  
 self.actionHistogram\_Equal.triggered.connect(self.equalHistogram)  
  
 def fungsi(self):  
 self.Image = cv2.imread('BANG.jpeg')  
 self.displayImage(1)  
  
 def grayscale(self):  
 if self.Image is not None:  
 gray = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 self.Image = gray  
 self.displayImage(2)  
  
 def brightness(self):  
 try:  
 self.Image = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 except:  
 pass  
 brightness = 80  
 bright\_img = cv2.convertScaleAbs(self.Image, alpha=1, beta=brightness)  
 self.Image = bright\_img  
 self.displayImage(1)  
  
 def contrast(self):  
 try:  
 self.Image = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 except:  
 pass  
 contrast = 1.7  
 contrast\_img = cv2.convertScaleAbs(self.Image, alpha=contrast, beta=0)  
 self.Image = contrast\_img  
 self.displayImage(1)  
  
 def contrastStretching(self):  
 if self.Image is not None:  
 min\_val = np.min(self.Image)  
 max\_val = np.max(self.Image)  
 stretched\_img = cv2.normalize(self.Image, None, 0, 255, cv2.NORM\_MINMAX)  
 self.Image = stretched\_img  
 self.displayImage(1)  
  
 def negativeImage(self):  
 if self.Image is not None:  
 negative\_img = 255 - self.Image  
 self.Image = negative\_img  
 self.displayImage(1)  
  
 def binaryImage(self):  
 if self.Image is not None:  
 if len(self.Image.shape) == 3: # Check if the image is colored  
 gray = cv2.cvtColor(self.Image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray = self.Image  
 # Apply binary threshold  
 threshold\_value, binary\_img = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)  
 # Compare pixel values before and after thresholding  
 H, W = gray.shape[:2]  
 comparison\_img = np.zeros((H, W, 3), np.uint8)  
 for i in range(H):  
 for j in range(W):  
 original\_value = gray[i, j]  
 binary\_value = binary\_img[i, j]  
 comparison\_img[i, j] = [original\_value, binary\_value, 0]  
 self.Image = binary\_img  
 self.displayImage(1)  
  
 def grayHistogram(self):  
 if self.Image is not None:  
 plt.hist(self.Image.ravel(), 255, [0, 255])  
 plt.show()  
  
 def rgbHistogram(self):  
 if self.Image is not None:  
 color = ('b', 'g', 'r')  
 for i, col in enumerate(color):  
 histo = cv2.calcHist([self.Image], [i], None, [256], [0, 256])  
 plt.plot(histo, color=col)  
 plt.xlim([0, 256])  
 plt.show()  
  
 def equalHistogram(self):  
 if self.Image is not None:  
 hist, bins = np.histogram(self.Image.flatten(), 256, [0, 256])  
 cdf = hist.cumsum()  
 cdf\_normalized = cdf \* hist.max() / cdf.max()  
 cdf\_m = np.ma.masked\_equal(cdf, 0)  
 cdf\_m = (cdf\_m - cdf\_m.min()) \* 255 / (cdf\_m.max() - cdf\_m.min())  
 cdf = np.ma.filled(cdf\_m, 0).astype("uint8")  
 self.Image = cdf[self.Image]  
 self.displayImage(2)  
 plt.plot(cdf\_normalized, color="b")  
 plt.hist(self.Image.flatten(), 256, [0, 256], color="r")  
 plt.xlim([0, 256])  
 plt.legend(("cdf", "histogram"), loc="upper left")  
 plt.show()  
  
 def displayImage(self, window=1):  
 if self.Image is not None:  
 qformat = QImage.Format\_Indexed8  
 if len(self.Image.shape) == 3:  
 if self.Image.shape[2] == 4:  
 qformat = QImage.Format\_RGBA8888  
 else:  
 qformat = QImage.Format\_RGB888  
 img = QImage(self.Image, self.Image.shape[1], self.Image.shape[0], self.Image.strides[0], qformat)  
 img = img.rgbSwapped()  
 if window == 1:  
 self.label.setPixmap(QPixmap.fromImage(img))  
 elif window == 2:  
 self.label\_2.setPixmap(QPixmap.fromImage(img))  
  
app = QtWidgets.QApplication(sys.argv)  
window = ShowImage()  
window.setWindowTitle('A2Praktek')  
window.show()  
sys.exit(app.exec\_())

A screen shot of a computer

Description automatically generated