Praktikum IV

(Pengolahan Citra Digital)

(B1 – B4)

Nama : Ishom Nabil Umran Alihaq

Kelas : TI-CCIT 4B

NIM : 2207412040

B1

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)  
 # operasi titik  
 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)  
 # operasi histogram  
 self.actionHistogram\_Gray.triggered.connect(self.grayHistogram)  
 self.actionHistogram\_RGB.triggered.connect(self.rgbHistogram)  
 self.actionHistogram\_Equal.triggered.connect(self.equalHistogram)  
 # operasi geometri  
 self.actionTranslasi.triggered.connect(self.translasi)  
  
 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 translasi(self):  
 h, w = self.Image.shape[:2]  
 quarter\_h, quarter\_w = h / 4, w / 4  
 T = np.float32([[1, 0, quarter\_w], [0, 1, quarter\_h]])  
 img = cv2.warpAffine(self.Image, T, (w, h))  
 self.Image = img  
 self.displayImage(2)  
  
 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 cartoon

Description automatically generated

B2

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)  
 # operasi titik  
 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)  
 # operasi histogram  
 self.actionHistogram\_Gray.triggered.connect(self.grayHistogram)  
 self.actionHistogram\_RGB.triggered.connect(self.rgbHistogram)  
 self.actionHistogram\_Equal.triggered.connect(self.equalHistogram)  
 # operasi geometri  
 self.actionTranslasi.triggered.connect(self.translasi)  
 self.action90\_Derajat.triggered.connect(self.rotasi90derajat)  
 self.action45\_Derajat.triggered.connect(self.rotasi45derajat)  
 self.action180\_Derajat.triggered.connect(self.rotasi180derajat)  
 self.action\_45\_Derajat.triggered.connect(self.rotasimin45derajat)  
 self.action\_90\_Derajat.triggered.connect(self.rotasimin90derajat)  
  
 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)  
 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 translasi(self):  
 h, w = self.Image.shape[:2]  
 quarter\_h, quarter\_w = h / 4, w / 4  
 T = np.float32([[1, 0, quarter\_w], [0, 1, quarter\_h]])  
 img = cv2.warpAffine(self.Image, T, (w, h))  
 self.Image = img  
 self.displayImage(2)  
  
 def rotasi(self, degree):  
 h, w = self.Image.shape[:2]  
 rotationMatrix = cv2.getRotationMatrix2D((w / 2, h / 2), degree, 0.7)  
 cos = np.abs(rotationMatrix[0, 0])  
 sin = np.abs(rotationMatrix[0, 1])  
 nW = int((h \* sin) + (w \* cos))  
 nH = int((h \* cos) + (w \* sin))  
 rotationMatrix[0, 2] += (nW / 2) - w / 2  
 rotationMatrix[1, 2] += (nH / 2) - h / 2  
 rot\_image = cv2.warpAffine(self.Image, rotationMatrix, (h, w))  
 self.Image = rot\_image  
 self.displayImage(2)  
  
 def rotasi45derajat(self):  
 self.rotasi(45)  
  
 def rotasimin45derajat(self):  
 self.rotasi(-45)  
  
 def rotasi90derajat(self):  
 self.rotasi(90)  
  
 def rotasimin90derajat(self):  
 self.rotasi(-90)  
  
 def rotasi180derajat(self):  
 self.rotasi(180)  
  
 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

B3

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)  
 # operasi titik  
 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)  
 # operasi histogram  
 self.actionHistogram\_Gray.triggered.connect(self.grayHistogram)  
 self.actionHistogram\_RGB.triggered.connect(self.rgbHistogram)  
 self.actionHistogram\_Equal.triggered.connect(self.equalHistogram)  
 # operasi geometri  
 self.actionTranslasi.triggered.connect(self.translasi)  
 self.action90\_Derajat.triggered.connect(self.rotasi90derajat)  
 self.action45\_Derajat.triggered.connect(self.rotasi45derajat)  
 self.action180\_Derajat.triggered.connect(self.rotasi180derajat)  
 self.action\_45\_Derajat.triggered.connect(self.rotasimin45derajat)  
 self.action\_90\_Derajat.triggered.connect(self.rotasimin90derajat)  
  
 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)  
 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 translasi(self):  
 h, w = self.Image.shape[:2]  
 quarter\_h, quarter\_w = h / 4, w / 4  
 T = np.float32([[1, 0, quarter\_w], [0, 1, quarter\_h]])  
 img = cv2.warpAffine(self.Image, T, (w, h))  
 self.Image = img  
 self.displayImage(2)  
  
 def rotasi(self, degree):  
 h, w = self.Image.shape[:2]  
 rotationMatrix = cv2.getRotationMatrix2D((w / 2, h / 2), degree, 0.7)  
 cos = np.abs(rotationMatrix[0, 0])  
 sin = np.abs(rotationMatrix[0, 1])  
 nW = int((h \* sin) + (w \* cos))  
 nH = int((h \* cos) + (w \* sin))  
 rotationMatrix[0, 2] += (nW / 2) - w / 2  
 rotationMatrix[1, 2] += (nH / 2) - h / 2  
 rot\_image = cv2.warpAffine(self.Image, rotationMatrix, (h, w))  
 self.Image = rot\_image  
 self.displayImage(2)  
  
 def rotasi45derajat(self):  
 self.rotasi(45)  
  
 def rotasimin45derajat(self):  
 self.rotasi(-45)  
  
 def rotasi90derajat(self):  
 self.rotasi(90)  
  
 def rotasimin90derajat(self):  
 self.rotasi(-90)  
  
 def rotasi180derajat(self):  
 self.rotasi(180)  
  
 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

B4

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)  
 # operasi titik  
 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)  
 # operasi histogram  
 self.actionHistogram\_Gray.triggered.connect(self.grayHistogram)  
 self.actionHistogram\_RGB.triggered.connect(self.rgbHistogram)  
 self.actionHistogram\_Equal.triggered.connect(self.equalHistogram)  
 # operasi geometri  
 self.actionTranslasi.triggered.connect(self.translasi)  
 self.action90\_Derajat.triggered.connect(self.rotasi90derajat)  
 self.action45\_Derajat.triggered.connect(self.rotasi45derajat)  
 self.action180\_Derajat.triggered.connect(self.rotasi180derajat)  
 self.action\_45\_Derajat.triggered.connect(self.rotasimin45derajat)  
 self.action\_90\_Derajat.triggered.connect(self.rotasimin90derajat)  
 self.actionZoom\_In.triggered.connect(self.zoomIn)  
 self.actionZoom\_Out.triggered.connect(self.zoomOut)  
 self.actionDimensi.triggered.connect(self.dimensi)  
 self.actionCrop.triggered.connect(self.cropImage)  
  
 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)  
 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 translasi(self):  
 h, w = self.Image.shape[:2]  
 quarter\_h, quarter\_w = h / 4, w / 4  
 T = np.float32([[1, 0, quarter\_w], [0, 1, quarter\_h]])  
 img = cv2.warpAffine(self.Image, T, (w, h))  
 self.Image = img  
 self.displayImage(2)  
  
 def rotasi(self, degree):  
 h, w = self.Image.shape[:2]  
 rotationMatrix = cv2.getRotationMatrix2D((w / 2, h / 2), degree, 0.7)  
 cos = np.abs(rotationMatrix[0, 0])  
 sin = np.abs(rotationMatrix[0, 1])  
 nW = int((h \* sin) + (w \* cos))  
 nH = int((h \* cos) + (w \* sin))  
 rotationMatrix[0, 2] += (nW / 2) - w / 2  
 rotationMatrix[1, 2] += (nH / 2) - h / 2  
 rot\_image = cv2.warpAffine(self.Image, rotationMatrix, (h, w))  
 self.Image = rot\_image  
 self.displayImage(2)  
  
 def rotasi45derajat(self):  
 self.rotasi(45)  
  
 def rotasimin45derajat(self):  
 self.rotasi(-45)  
  
 def rotasi90derajat(self):  
 self.rotasi(90)  
  
 def rotasimin90derajat(self):  
 self.rotasi(-90)  
  
 def rotasi180derajat(self):  
 self.rotasi(180)  
  
 def zoomIn(self):  
 skala = 2  
 resize\_image = cv2.resize(self.Image, None, fx=skala, fy=skala, interpolation=cv2.INTER\_CUBIC)  
 cv2.imshow('original', self.Image)  
 cv2.imshow('Zoom In', resize\_image)  
 cv2.waitKey()  
  
 def zoomOut(self):  
 skala = 0.50  
 resize\_image = cv2.resize(self.Image, None, fx=skala, fy=skala)  
 cv2.imshow('original', self.Image)  
 cv2.imshow('Zoom Out', resize\_image)  
 cv2.waitKey()  
  
 def dimensi(self):  
 resize\_image = cv2.resize(self.Image, (900, 400), interpolation=cv2.INTER\_AREA)  
 cv2.imshow('original', self.Image)  
 cv2.imshow('dimensi', resize\_image)  
 cv2.waitKey()  
  
 def cropImage(self):  
 start\_row = 50  
 end\_row = 200  
 start\_col = 100  
 end\_col = 300  
 crop\_image = self.Image[start\_row:end\_row, start\_col:end\_col]  
 cv2.imshow('original', self.Image)  
 cv2.imshow('Crop Image', crop\_image)  
 cv2.waitKey()  
  
 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