**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(МОСКОВСКИЙ ПОЛИТЕХ)**

Транспортный факультет

Направление подготовки: 01.03.02 «Прикладная математика и информатика»

Отчет по Лабораторной работе №7

“Стабилизация изображения и поиск соответствий”

Выполнил:

Студент:

Группа: 211-172

Тимошенко Ф.Р.

Проверил:

Кожухов Д.А.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2024

## 1. Введение

Цель данного задания состояла в разработке программного средства для обработки изображений, способного отображать два изображения в одном окне и выполнять операции по вычислению изменений позиций характерных точек с использованием метода Лукаса-Канаде, а также поиску общих характерных точек изображений

## 2. Реализация

Программа разработана с использованием библиотек PyQt6 и OpenCV. Она состоит из двух классов: ImageWindow и MyWindow.

Класс ImageWindow:

**Атрибуты:**

* image\_label1, image\_label2, image\_label3: QLabel для отображения изображений до и после обработки.
* central\_widget: QWidget, центральный виджет главного окна.
* layout: QHBoxLayout для размещения виджетов.
* image\_layout: QVBoxLayout для размещения меток с изображениями.
* button\_layout: QVBoxLayout для размещения кнопок.

**Методы:**

* \_\_init\_\_: конструктор класса, инициализирующий главное окно и его компоненты.
* update\_images1, update\_images2, update\_images3: обновление изображений до и после обработки.
* img\_hide: скрытие меток с изображениями.
* update\_button: создание и настройка кнопок для выбора видео и использования камеры.

## Класс MyWindow (расширение ImageWindow)

**Дополнительные атрибуты:**

* initial\_path: Путь к выбранному изображению
* centerx, centery: Координаты центра вращения изображения.
* angle: Угол вращения изображения
* original\_image: Данные повернутого изображения
* keypoints: Характерные точки в изображении.
* keypoints\_image: Изображение с найденными характерными точками.

**Дополнительные методы:**

* on\_button1\_clicked: Загрузка изображения и начало его обработки.
* on\_button\_findh\_clicked: Начало поиска характерных точек
* on\_button\_track\_clicked: Начало отслеживания характерных точек
* on\_button\_compare\_clicked: Сравнение характерных точек в изображениях
* onChanged\_a: Обработчик изменения значения угла поворота изображения.
* download\_img: Загрузка изображения, сохранение его пути в атрибут initial\_path
* loadcv2: Загрузка изображения с помощью OpenCV.
* saved\_and\_print\_process: Сохранение и отображение изображение
* rotate: Поворот изображения на заданный угол
* find\_features: Поиск характерных точек в изображении.
* track\_features: Отслеживание характерных точек
* match\_features: Сопоставление характерных точек

## 3. Описание работы программы

1. Пользователь выбирает исходное изображение.
2. Программа отображает исходное изображение в графическом интерфейсе.
3. Пользователь может выбрать угол поворота изображения с помощью ползунка.
4. Программа поворачивает изображение на заданный угол и отображает результат.
5. Пользователь может выбрать одну из операций:
   * Найти особенности: программа находит особенности в изображении и отображает
   * Отследить особенности: программа отслеживает особенности на новом изображении и отображает траекторию.
   * Сопоставить особенности: программа сравнивает особенности между двумя изображениями (функция в разработке).
6. Результат обработки отображается в графическом интерфейсе программы.

## 4. Листинг кода

“window.py”

|  |
| --- |
| from PyQt6.QtWidgets import QComboBox, QSlider, QApplication, QMainWindow, QLabel, QVBoxLayout, QWidget, QPushButton, QHBoxLayout, QLineEdit  from PyQt6.QtGui import QPixmap  from PyQt6.QtCore import Qt  class ImageWindow(QMainWindow):      def \_\_init\_\_(self):          super().\_\_init\_\_()          self.setWindowTitle("Изображения")          self.setGeometry(50, 50, 400, 400)          self.central\_widget = QWidget()          self.setCentralWidget(self.central\_widget)          self.layout = QHBoxLayout()          self.central\_widget.setLayout(self.layout)            self.image\_layout = QVBoxLayout()          self.layout.addLayout(self.image\_layout)            self.image\_label1 = QLabel()          self.label1\_title = QLabel('До обработки')          self.image\_label2 = QLabel()          self.label2\_title = QLabel('Повернутое изображение')          self.image\_label3 = QLabel()          self.label3\_title = QLabel('Обработанное изображение')            self.image\_layout.addWidget(self.label1\_title, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.image\_label1, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.label2\_title, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.image\_label2, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.label3\_title, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.image\_label3, alignment=Qt.AlignmentFlag.AlignCenter)            self.button\_layout = QVBoxLayout()          self.layout.addLayout(self.button\_layout)          self.update\_button()          self.label1\_title.hide()          self.image\_label1.hide()          self.label2\_title.hide()          self.image\_label2.hide()          self.label3\_title.hide()          self.image\_label3.hide()          self.show()      def update\_images1(self, image\_path1):          self.label1\_title.show()          pixmap1 = QPixmap(image\_path1)            scaled\_pixmap1 = pixmap1.scaled(200, 200, Qt.AspectRatioMode.KeepAspectRatio)            self.image\_label1.setPixmap(scaled\_pixmap1)          self.image\_label1.show()            self.update()      def img\_hide(self):          self.label1\_title.hide()          self.image\_label1.hide()        def update\_images2(self, image\_path2):          self.label2\_title.show()            pixmap2 = QPixmap(image\_path2)            scaled\_pixmap2 = pixmap2.scaled(200,200, Qt.AspectRatioMode.KeepAspectRatio)            self.image\_label2.setPixmap(scaled\_pixmap2)          self.image\_label2.show()            self.update()      def update\_images3(self, image\_path3):          self.label3\_title.show()            pixmap3 = QPixmap(image\_path3)            scaled\_pixmap3 = pixmap3.scaled(200,200, Qt.AspectRatioMode.KeepAspectRatio)            self.image\_label3.setPixmap(scaled\_pixmap3)          self.image\_label3.show()            self.update()        def update\_button(self):          self.button1 = QPushButton("Выбрать изображение")          self.button1.clicked.connect(self.on\_button1\_clicked)          self.button\_layout.addWidget(self.button1)            self.button\_findh= QPushButton("Найти особенности")          self.button\_findh.clicked.connect(self.on\_button\_findh\_clicked)          self.button\_layout.addWidget(self.button\_findh)          self.button\_track= QPushButton("Отследить особенности")          self.button\_track.clicked.connect(self.on\_button\_track\_clicked)          self.button\_layout.addWidget(self.button\_track)          self.button\_compare= QPushButton("Сопоставить особенности")          self.button\_compare.clicked.connect(self.on\_button\_compare\_clicked)          self.button\_layout.addWidget(self.button\_compare)          #Ползунок на angle          self.labela\_title = QLabel('Угол')          self.button\_layout.addWidget(self.labela\_title)          self.labela\_title.hide()          self.labela = QLabel('0', self)          self.labela.setAlignment(Qt.AlignmentFlag.AlignCenter)          self.button\_layout.addWidget(self.labela)          self.labela.hide()          self.slider\_a = QSlider(Qt.Orientation.Horizontal, self)          self.slider\_a.setMinimum(-180)          self.slider\_a.setMaximum(180)          self.slider\_a.setValue(0)          self.slider\_a.setTickPosition(QSlider.TickPosition.TicksBelow)          self.slider\_a.setTickInterval(1)          self.slider\_a.valueChanged.connect(self.onChanged\_a)          self.button\_layout.addWidget(self.slider\_a)          self.slider\_a.hide() |

“main.py”

|  |
| --- |
| import cv2  import sys  from window import ImageWindow  import numpy as np  from PyQt6.QtWidgets import QApplication, QFileDialog, QVBoxLayout, QLineEdit, QLabel, QHBoxLayout, QWidget  save\_process\_path='stuff/saved/save\_proc.jpg'  class Mywindow(ImageWindow):      def \_\_init\_\_(self):          super().\_\_init\_\_()          self.initial\_path=''          self.centerx=50          self.centery=50          self.angle=-90          self.original\_image=None      #Кнопки      def on\_button1\_clicked(self):          self.img\_hide()          self.download\_img(1)          self.slider\_a.show()      def on\_button\_findh\_clicked(self):          self.slider\_a.hide()          self.find\_features()      def on\_button\_track\_clicked(self):          self.slider\_a.hide()          self.track\_features()      def on\_button\_compare\_clicked(self):          self.slider\_a.hide()      def onChanged\_a(self, value):          self.angle=value          self.labela.setText(str(value))          self.rotate()      def download\_img(self, i):          try:              self.initial\_path, \_ = QFileDialog.getOpenFileName(self, "Выберите изображение", "", "Изображения (\*.png \*.jpg \*.jpeg)")              if not self.initial\_path:                  raise FileNotFoundError("Путь к изображению не был выбран.")              if i==1:                  self.update\_images1(self.initial\_path)              else:                  raise FileNotFoundError("Куда ты хочешь картинку?")          except Exception as e:              print("Ошибка при загрузке изображения", e)              return None        def loadcv2(self, ini):          try:              if not ini:                  raise FileNotFoundError("Путь к изображению не был выбран.")              img = cv2.imread(ini)              return img          except Exception as e:              print("Ошибка при выполнении операции: ", e)              return None      def saved\_and\_print\_process(self, img):          cv2.imwrite(save\_process\_path, img)          self.update\_images2(save\_process\_path)      def rotate(self):          try:              img = self.loadcv2(self.initial\_path)              rows, cols = img.shape[:2]              #M = cv2.getRotationMatrix2D((self.centery,self.centerx), self.angle, 1)              M = cv2.getRotationMatrix2D((rows//2,cols//2), self.angle, 1)              img = cv2.warpAffine(img, M, (cols, rows), flags=cv2.INTER\_LINEAR)              self.saved\_and\_print\_process(img)          except Exception as e:              print("Ошибка при rotate: ", e)              return None      def find\_features(self):          self.original\_image = self.loadcv2(save\_process\_path)          gray = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2GRAY)          self.keypoints = cv2.goodFeaturesToTrack(gray, maxCorners=100, qualityLevel=0.01, minDistance=10)          self.keypoints = [cv2.KeyPoint(x[0][0], x[0][1], 1) for x in self.keypoints]          self.keypoints\_image = cv2.drawKeypoints(self.original\_image, self.keypoints, None, color=(0, 255, 0))          cv2.imwrite(save\_process\_path, self.keypoints\_image)          self.update\_images3(save\_process\_path)      def track\_features(self):          if self.original\_image is None:              return          if self.keypoints is None:              self.find\_features()            old\_gray = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2GRAY)          new\_image = cv2.imread(self.initial\_path)          new\_gray = cv2.cvtColor(new\_image, cv2.COLOR\_BGR2GRAY)          p0 = np.array([point.pt for point in self.keypoints], dtype=np.float32)          p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, new\_gray, p0, None)          # Добавляем отладочный вывод для проверки размерности st          print("Размерность st:", st.shape)          good\_indices = np.where(st==1)[0]  # Находим индексы хороших точек          good\_new = p1[good\_indices]  # Выбираем хорошие новые точки          good\_old = p0[good\_indices]  # Выбираем соответствующие старые точки            for i, (new, old) in enumerate(zip(good\_new, good\_old)):              a, b = new.ravel()              c, d = old.ravel()              cv2.circle(new\_image, (int(a), int(b)), 5, (0, 255, 0), -1)              cv2.line(new\_image, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)            cv2.imwrite(save\_process\_path, new\_image)          self.update\_images3(save\_process\_path)      def match\_features(self):          if self.original\_image is None:              return          new\_image = cv2.imread(self.initial\_path)          gray1 = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2GRAY)          gray2 = cv2.cvtColor(new\_image, cv2.COLOR\_BGR2GRAY)          orb = cv2.ORB\_create()          kp1, des1 = orb.detectAndCompute(gray1, None)          kp2, des2 = orb.detectAndCompute(gray2, None)          bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)          matches = bf.match(des1, des2)          matches = sorted(matches, key = lambda x:x.distance)          matching\_result = cv2.drawMatches(self.original\_image, kp1, new\_image, kp2, matches[:10], None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)          self.display\_image(matching\_result, self.processed\_label)          cv2.imwrite(save\_process\_path, matching\_result)          self.update\_images3(save\_process\_path)    if \_\_name\_\_ == "\_\_main\_\_":      app = QApplication(sys.argv)      window = Mywindow()      sys.exit(app.exec()) |