**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

**отчЁт**

**по лабораторной работе № 6**

**по дисциплине «Интеллектуальный анализ данных»**

**Тема: Синтез панорамных изображений**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9105 |  | Шаривзянов Д. Р. |
|  |  | Чугунов Р. |
| Преподаватель |  | Поздеев А. А. |

Санкт-Петербург

2024

СИНТЕЗ ПАНОРАМНЫХ ИЗОБРАЖЕНИЙ

**Цель работы:** разработать программу, реализующую синтез панорамы из двух фрагментов с помощью библиотеки OpenCV.

Код программы:

#ifndef PANORAMA\_H

#define PANORAMA\_H

#include <vector>

#include <opencv2/opencv.hpp>

#include <io.h>

#include <direct.h>

#include "logger.hpp"

using namespace cv;

using namespace std;

string output\_path = "output/Lab 6 Panorama/";

/\*\*

 \* @brief Функция загрузки изображений

 \*

 \* Функция cutImg() загружает два изображения из входного изображения src.

 \* Она возвращает два изображения img1 и img2, которые

 \* соответствуют левой и правой частям входного изображения.

 \*

 \* @param[in] src - исходное изображение

 \* @param[out] img1 - левая часть изображения

 \* @param[out] img2 - правая часть изображения

 \*/

void cutImg(const Mat& src, Mat& img1, Mat& img2) {

    // выделяем левую часть изображения

    // Range - диапазон строк, Range - диапазон столбцов

    img1 = src(Range(0, 500), Range(0, 550));

    // выделяем правую часть изображения

    img2 = src(Range(0, 400), Range(300, 750));

}

/\*\*

 \* @brief Функция обнаружения и вычисления дескрипторов признаков

 \*

 \* Функция detectAndComputeFeatures() обнаруживает и вычисляет

 \* дескрипторы признаков для входного изображения img.

 \* Она использует алгоритм ORB.

 \*

 \* @param[in] img - входное изображение

 \* @param[out] keypoints - вектор ключевых точек

 \* @param[out] descriptors - матрица дескрипторов

 \*/

void detectAndComputeFeatures(const Mat& img, std::vector<KeyPoint>& keypoints, Mat& descriptors) {

    // создаем объект детектора ORB

    Ptr<ORB> detector = ORB::create();

    // обнаруживаем ключевые точки на изображении

    // и вычисляем дескрипторы для них

    detector->detectAndCompute(img, noArray(), keypoints, descriptors);

}

/\*\*

 \* @brief Функция поиска соответствий между дескрипторами двух изображений

 \*

 \* Функция matchDescriptors() ищет соответствия между дескрипторами

 \* двух изображений, используя алгоритм Brute-Force Matching.

 \* Она использует коэффициент match\_ratio для фильтрации

 \* не надежных соответствий.

 \*

 \* @param[in] descriptor1 - матрица дескрипторов первого изображения

 \* @param[in] descriptor2 - матрица дескрипторов второго изображения

 \* @param[in] keypoints1 - вектор ключевых точек первого изображения

 \* @param[in] keypoints2 - вектор ключевых точек второго изображения

 \* @param[out] img1\_pts - вектор координат точек на первом изображении

 \* @param[out] img2\_pts - вектор координат точек на втором изображении

 \* @param[out] good\_matches - вектор надежных соответствий

 \* @param[in] match\_ratio - коэффициент фильтрации (0.1f по умолчанию)

 \*/

void matchDescriptors(const Mat& descriptor1, const Mat& descriptor2,

                      const std::vector<KeyPoint>& keypoints1, const std::vector<KeyPoint>& keypoints2,

                      std::vector<Point2f>& img1\_pts, std::vector<Point2f>& img2\_pts,

                      std::vector<DMatch>& good\_matches, float match\_ratio = 0.1f) {

    // создаем объект матчера Brute-Force

    BFMatcher matcher(NORM\_HAMMING);

    // ищем соответствия между дескрипторами

    std::vector<std::vector<DMatch>> matches;

    matcher.knnMatch(descriptor1, descriptor2, matches, 2);

    // фильтруем не надежные соответствия

    for (const auto& match : matches) {

        if (match[0].distance < match\_ratio \* match[1].distance) {

            // добавляем координаты точек на оба изображения

            img1\_pts.push\_back(keypoints1[match[0].queryIdx].pt);

            img2\_pts.push\_back(keypoints2[match[0].trainIdx].pt);

            // добавляем надежное соответствие

            good\_matches.push\_back(DMatch(static\_cast<int>(img1\_pts.size() - 1), static\_cast<int>(img2\_pts.size() - 1), 0));

        }

    }

}

/\*\*

 \* @brief Функция отрисовки и отображения соответствий между

 \*        ключевыми точками двух изображений

 \*

 \* Функция drawAndShowMatches() отрисовывает соответствия между

 \* ключевыми точками двух изображений img1 и img2, используя

 \* алгоритм drawMatches(). Она отображает результат на экране,

 \* используя функцию imshow(), и сохраняет его в файл,

 \* используя функцию imwrite().

 \*

 \* @param[in] img1 - первое изображение

 \* @param[in] img2 - второе изображение

 \* @param[in] keypoints1 - вектор ключевых точек первого изображения

 \* @param[in] keypoints2 - вектор ключевых точек второго изображения

 \* @param[in] matches - вектор соответствий между ключевыми точками

 \*/

void drawAndShowMatches(const Mat& img1, const Mat& img2,

                        const std::vector<KeyPoint>& keypoints1, const std::vector<KeyPoint>& keypoints2,

                        const std::vector<DMatch>& matches) {

    Mat dMatches;

    drawMatches(img1, keypoints1, img2, keypoints2, matches, dMatches);

    // отображаем результат на экране

    imshow("Matches", dMatches);

    // сохраняем результат в файл

    imwrite(output\_path + "Matches.png", dMatches);

}

/\*\*

 \* @brief Создание панорамы из двух изображений

 \*

 \* Функция createPanorama() создает панораму из двух изображений img1 и img2,

 \* используя матрицу гомографии H, полученную с помощью функции findHomography().

 \* Результат - Mat-объект, содержащий панораму.

 \*

 \* @param[in] img1 - первое изображение

 \* @param[in] img2 - второе изображение

 \* @param[in] H - матрица гомографии

 \* @param[out] panorama - панорама

 \*/

void createPanorama(const Mat& img1, const Mat& img2, const Mat& H, Mat& panorama) {

    // Создаем матрицу результата, размером со сумму ширины двух изображений

    Mat result;

    warpPerspective(img2, result, H, Size(img1.cols + img2.cols, img1.rows));

    // Создаем матрицу панорамы, заполненную белым цветом

    panorama = Mat(result.rows, result.cols, result.type(), Scalar(220, 220, 220));

    // Создаем подматрицу из левой части панорамы

    Mat half(panorama, Rect(0, 0, img1.cols, img1.rows));

    // Копируем левое изображение в левую часть панорамы

    img1.copyTo(half);

    // Проходимся по всем пикселям панорамы

    for (int y = 0; y < panorama.rows; y++) {

        for (int x = 0; x < panorama.cols; x++) {

            // Берем цвет пикселя на левой и правой частях панорамы

            Vec3b color1 = panorama.at<Vec3b>(y, x);

            Vec3b color2 = result.at<Vec3b>(y, x);

            // Если правый пиксель не черный, то смешиваем цвета

            if (color2 != Vec3b(0, 0, 0)) {

                float alpha = 0.01f;

                panorama.at<Vec3b>(y, x) = color1 \* alpha + color2 \* (1.0f - alpha);

            }

        }

    }

}

/\*\*

 \* @brief Функция для выполнения лабораторной работы 6

 \*        (создание панорамы из двух изображений)

 \*

 \* @param[in] img\_bgr - входное RGB-изображение

 \*

 \* @details

 \* 1. Создаём папку для выходных изображений.

 \* 2. Разделяем исходное изображение на две части.

 \* 3. Определяем ключевые точки (ключевые точки) на каждой из частей.

 \* 4. Создаём матрицы дескрипторов для каждой из частей.

 \* 5. Определяем соответствия между ключевыми точками.

 \* 6. Создаём матрицу гомографии.

 \* 7. Создаём панораму.

 \* 8. Выводим панораму на экран.

 \* 9. Ожидаем нажатие клавиши.

 \*/

void lab6\_Panorama(const Mat& img\_bgr) {

    logger.info("Lab 6: Panorama.");

    // 1. Создаём папку для выходных изображений

    if (\_access(output\_path.c\_str(), 0) != 0) {

        if (\_mkdir(output\_path.c\_str()) == -1) {

            logger.error("Failed to create directory: {}", output\_path);

            return;

        }

        logger.info("Created directory: {}", output\_path);

    }

    Mat img1, img2;

    // 2. Разделяем исходное изображение на две части

    cutImg(img\_bgr, img1, img2);

    std::vector<KeyPoint> keypoints1, keypoints2;

    Mat descriptors1, descriptors2;

    // 3. Определяем ключевые точки на каждой из частей

    detectAndComputeFeatures(img1, keypoints1, descriptors1);

    detectAndComputeFeatures(img2, keypoints2, descriptors2);

    Mat keypoints1draw, keypoints2draw;

    // 4. Создаём матрицы дескрипторов для каждой из частей

    drawKeypoints(img1, keypoints1, keypoints1draw);

    drawKeypoints(img2, keypoints2, keypoints2draw);

    imshow("Keypoints1", keypoints1draw);

    imwrite(output\_path + "Keypoints1.png", keypoints1draw);

    imshow("Keypoints2", keypoints2draw);

    imwrite(output\_path + "Keypoints2.png", keypoints2draw);

    std::vector<Point2f> img1\_pts, img2\_pts;

    std::vector<DMatch> good\_matches;

    // 5. Определяем соответствия между ключевыми точками

    matchDescriptors(descriptors1, descriptors2, keypoints1, keypoints2, img1\_pts, img2\_pts, good\_matches);

    drawAndShowMatches(img1, img2, keypoints1, keypoints2, good\_matches);

    Mat H = findHomography(img2\_pts, img1\_pts, RANSAC);

    Mat panorama;

    // 7. Создаём панораму

    createPanorama(img1, img2, H, panorama);

    // 8. Выводим панораму на экран

    imshow("Panorama", panorama);

    imwrite(output\_path + "Panorama.png", panorama);

    // 9. Ожидаем нажатие клавиши

    waitKey();

}

#endif // PANORAMA\_H

1. Входные данные

Исходными данными является цветное изображение с множеством деталей.



Рисунок 1 – Исходное изображение с множеством деталей.

1. Разбиение на части

Разделим изображение на части для последующей сшивки, далее выделим ключевые точки.



Рисунок 2 – Разделённые части исходного изображения с найденными ключевыми точками.

1. Сопоставление ключевых точек

Сопоставим лучшие сочетания из ключевых точек.

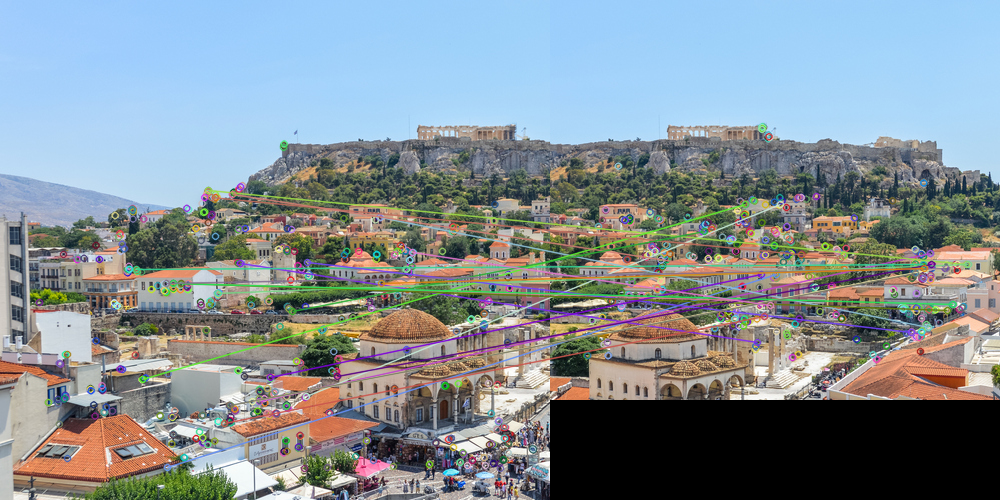


Рисунок 3 – Сопоставление ключевых точек.

1. Результат сшивки панорамы



Рисунок 4 – Результат сшивки панорамы.

**Вывод:** в ходе выполнения лабораторной работы был изучен процесс синтеза панорамных изображений, включающий нахождение ключевых точек, сопоставление их на разных снимках, калибровку изображений, определение параметров трансформации и блэндинг для устранения артефактов. Было продемонстрировано использование алгоритма RANSAC для фильтрации выбросов и повышения устойчивости результатов.