**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

отчет

**по лабораторной работе №3**

**по дисциплине «Интеллектуальные видеосистемы»**

Тема: СИНТЕЗ ПАНОРАМНЫХ ИЗОБРАЖЕНИЙ

|  |  |  |
| --- | --- | --- |
| Студенты гр. 8571 |  | Темный Д.А.  Мальцева О.Н. |
| Преподаватель |  | Поздеев А.А. |

Санкт-Петербург

2023

**Цель работы**

Целью лабораторной работы является знакомство с методами сшивания панорам. В соответствии с заданием необходимо разработать программу, реализующую синтез панорамы из двух фрагментов с помощью библиотеки OpenCV.

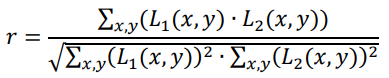
Необходимо подобрать два фрагмента (зона перекрытия изображений должна занимать не менее 25% их площади, желателен поворот или перспективное преобразование одного изображения относительно другого) осуществить синтез панорамы.

При работе программы должны быть визуализированы основные этапы:

* изображения с найденными ключевыми точками;
* результат процедуры поиска согласованных пар;
* результат трансформации изображения (один из снимков является опорным и остается без изменений, трансформируют только второй фрагмент);
* результат построения панорамы.

**Теоретические сведения**

Задача синтеза панорамы (предположим, из двух фрагментов) может быть решена путем корреляционного совмещения одного изображения с другим. Причем, совмещаемое изображение должно иметь степени свободы по масштабу и углам поворота. Последовательно трансформируя один из фрагментов (изменяя масштаб и углы поворота с некотором шагом) и совмещая его (изменяя смещение одного снимка относительно второго) с другим находят максимум отклика корреляционной функции в некоторой точке *r*, тем самым определяя оптимальные параметры для сшивки изображений.



где – опорное изображение, – совмещаемое изображение.

Данный метод, однако, мало применим на практике. Во-первых, он очень чувствителен к шумам и неодинаковой яркости фрагментов. Во-вторых, даже при фрагментах небольшого размера он требует огромного количества вычислительных ресурсов.

Общепринятый алгоритм для синтеза панорам состоит из следующих шагов:

1. Нахождение характерных точек (в англоязычной литературе используют термин «ключевая» точка «key point») на исходных изображениях в зоне перекрытия. Под характерной точкой понимают некоторый малый фрагмент изображения, в котором как значение яркостного градиента, так и производная (скорость изменения) градиента по направлению высоки. Как правило, характерных точками являются различные углы на изображениях (рис. 1). Рис. 1. Характерные точки сшиваемых изображений (зона перекрытия показана рамкой)
2. Нахождение в зоне перекрытия одних и тех же особенностей на различных снимках (рисунок 2). То есть определение соответствующих друг другу пар характерных точек.

Согласованные пары точек — это основа для синтеза панорамы. С помощью установленных пар на последующих этапах решают задачи калибровки, трансформации и объединения снимков. В случае синтеза панорамы из нескольких кадров (частей) при неизвестном заранее расположении каждого снимка, на данном этапе проводят процедуру регистрации – идентификации местоположения отдельных изображений на общей панораме и установление взаимных соответствий характерных точек.

1. В современных программных пакетах следующим шагом алгоритма сшивки является калибровка изображений. Эта процедура направлена на минимизацию искажений объектива, оптических дефектов, различий экспозиции. С помощью информации о согласованных парах характерных точек минимизируют влияние дисторсии (геометрических искажений) объектива на точность сшивки панорамы.
2. Ключевым этапом является процедура идентификации параметров уравнений трансформации изображений и последующее преобразование фрагментов с объединением в единую панораму. Данный этап требует задания вида трансформации, которое определяется типом создаваемой панорамы. Например, при отсутствии, или несущественности перспективных искажений у рис. 2. Пары согласованных точек на сшиваемых изображениях фрагментов имеет смысл использовать аффинное преобразование. Примером такого изображения может быть «сшивка» панорамы из отсканированных частей единого документа (географической карты, картины, и т.п.). В общем случае применяют перспективное преобразование, учитывающее все возможные искажения снимков.
3. Заключительным этапом является блэндинг. Это комплексная процедура, направленная на повышение визуального качества панорамы, включающая выравнивание яркости и цветовой палитры фрагментов, маскирование «швов», удаление «призраков» (движущихся объектов). Кроме того, в блэндинг включают и процедуру проецирования панорамы на заданную поверхность – сферическую, цилиндрическую, эквидистантную и пр.

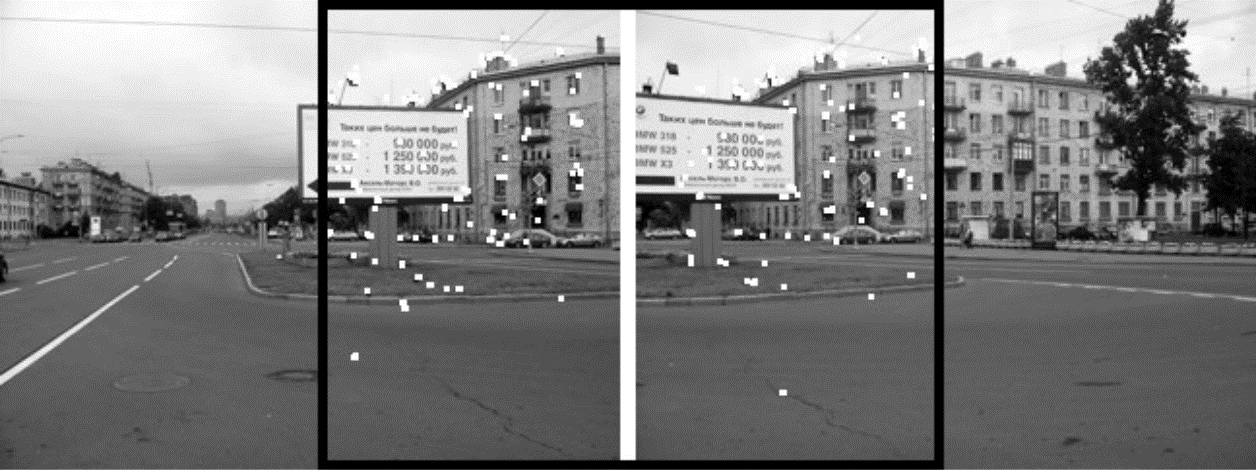


Рис. 1 – Характерные точки сшиваемых изображений (зона перекрытия показана рамкой)

Простейшим примером, демонстрирующим процесс синтеза единого изображения, является построение панорамы из двух фрагментов. Для нахождения характерных точек существует множество алгоритмов – «детекторов». Самыми распространенным являются SIFT и SURF. Базовым детектором, лежащем в основе многих других, более совершенных является уголковый детектор Харриса. Угол на изображении интересен тем, что по нему легко локализовать соответствующую особенность на различных снимках. В угловой точке градиент яркости меняет свое направление в предельном случае на 90°.

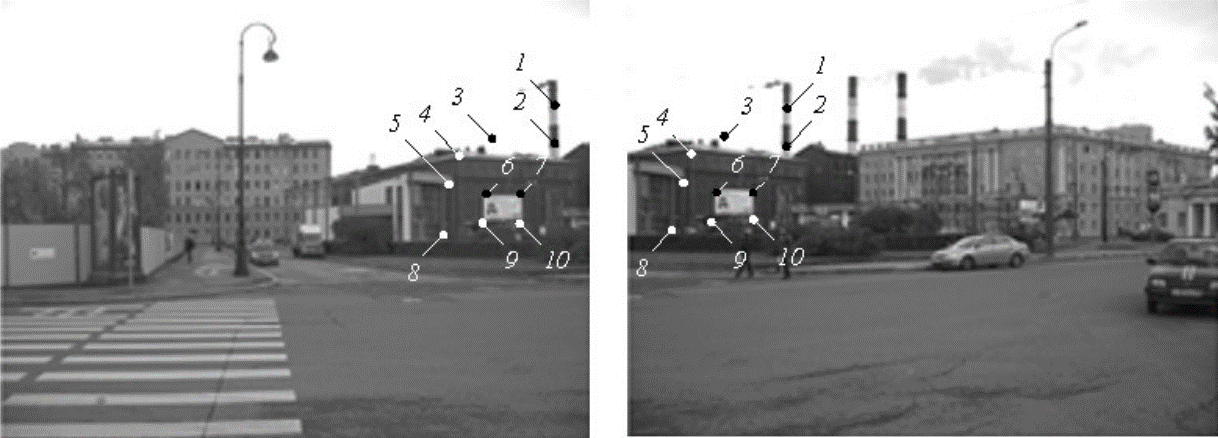
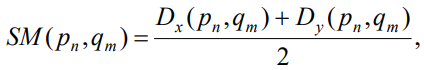


Рис. 2 – Пары согласованных точек на сшиваемых изображениях

После выделения характерных точек на двух соседних изображениях формируют пары согласованных характерных точек, представляющих на изображениях одни и те же области. Для этого производят корреляционное сопоставление блоков с центрами в найденных характерных точках. Каждый блок первого изображения сравнивают со всеми блоками второго изображения в зоне перекрытия. Мера подобия двух блоков с центрами в характерных точках и



где m, n – число пикселей в сравниваемых блоках по вертикали и горизонтали, соответственно.

Значение изменяется в диапазоне от нуля до единицы, причем для идентичных блоков.

Для оценки параметров уравнения трансформации используют алгоритм устойчивой оценки RANSAC:

1. Случайным образом из множества согласованных точек выбираются пять пар. По ним определяют параметры уравнения трансформации (создают текущую гипотезу уравнения).
2. Подсчитывают число пар характерных точек (так называемых «не выбросов», в англоязычной литературе «inliers»), для которых ошибка трансформации E при текущей гипотезе не превышает некоторого порога (обычно 2-3 пикселя), остальные пары отбрасывают.
3. Шаги 1, 2 повторяют заранее определенное число итераций. На каждой итерации получают и проверяют новую гипотезу с помощью очередных случайно выбранных пяти пар согласованных точек. В результате запоминают параметры гипотезы, для которой получилось больше всего «не выбросов».
4. По пяти парам согласованных точек, с помощью которых была получена гипотеза, запомненная на шаге 3, вновь определяют параметры уравнения трансформации.
5. Исключают выбросы (то есть пары точек, для которых ошибка превышает порог) и пересчитывают параметры уравнения трансформации, но уже с учетом всех пар «не-выбросов».

По найденным параметрам уравнения выполняют трансформации изображений и «сшивку» панорамы. Пример снимков-фрагментов и созданного единого изображения приведен на рисунке 3.



Рис. 3 – Пример созданной панорамы (нижний ряд) из двух фрагментов (верхний ряд)

**Синтез панорамы**

Исходные изображения представлены на рис. 4 и 5.



Рис. 4 – Первое исходное изображение (левое)



Рис. 5 – Второе исходное изображение (правое)

Для поиска ключевых точек использовались функции detect() и compute() класса cv::ORB детектора локальных признаков. Результаты отображения найденных точек представлены на рис. 6 и 7 для левого и правого изображения соответственно.



Рис. 6 – Характерные точки первого изображения



Рис. 7 – Характерные точки второго изображения

Визуализируем сопоставление дескрипторов, реализованного с помощью алгоритма Brute Force Matching (рис. 8).



Рис. 8 – Результат сопоставления характерных точек

Далее с помощью функции findHomography() используя алгоритм RANSAC производится поиск матрицы гомографии для трансформации (используя функцию warpPerspective()) правого изображения для последующего наложения его ключевых точек на точки левого изображения (результат представлен на рис. 9).

Так как изображения были получены вращением камеры по оси Z, в итоговом изображении присутствуют искажения перспективы.



Рис. 9 – Трансформированное второе изображение, наложенное на первое

**Вывод**

В ходе работы были опробован один из методов сшивки панорам в OpenCV. Найдены ключевые точки на изображениях с использованием ORB-детектора и отображена работа дескриптора (визуализация сопоставления найденных точек).

Найдена матрица гомографии, в соответствии с которой искажено одно из изображений и наложено на второе.

**Листинг Кода**

**main.cpp:**

#include <iostream>

#include <string>

#include <vector>

#include <utility>

#include <algorithm>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

#include <panorama.h>

int main(int argc, char\* argv[]) {

    if (argc == 2)

    {

        std::cout << "parsed too few arguments, at least 2 pictures ";

        return 1;

    }

    std::vector<cv::Mat> output\_images;

    std::vector<cv::Mat> matches\_images;

    std::vector<cv::Mat> src\_images;

    std::vector<cv::Mat> keypoint\_images;

    std::vector<std::pair <std::vector<cv::KeyPoint>, cv::Mat>> detection\_result;

    for (int i = 1; i < argc; i++)

    {

        std::string filename = argv[i];

        src\_images.push\_back(cv::imread(argv[i]));

        cv::Mat keypoint\_img;

        keypoint\_images.push\_back(keypoint\_img);

    }

    std::string output\_folder = "output";

    std::string create\_folder = "mkdir " + output\_folder;

    system(create\_folder.c\_str());

    output\_folder += "/";

    for (size\_t i = 0; i < src\_images.size(); ++i) {

        cv::cvtColor(src\_images[i], src\_images[i], cv::COLOR\_BGR2BGRA);

        detection\_result.push\_back( detect\_keypoints(src\_images[i], keypoint\_images[i]) );

    }

    for (size\_t i = 0; i < src\_images.size()-1; ++i) {

        cv::Mat out, matc;

        output\_images.push\_back(out);

        matches\_images.push\_back(matc);

        match({src\_images[i], src\_images[i+1]},

            output\_images[i], matches\_images[i],

            detection\_result[i], detection\_result[i+1]);

    }

    std::vector<cv::Mat> rec\_output;

    recursive\_panorama\_sewing(src\_images, rec\_output);

    imwrite\_vector(output\_folder + "recursive", "png", rec\_output);

    for (auto &img : output\_images)

        crop\_border(img);

    imwrite\_vector(output\_folder + "keypoints", "jpg", keypoint\_images);

    imwrite\_vector(output\_folder + "matches", "jpg", matches\_images);

    imwrite\_vector(output\_folder + "result", "jpg", output\_images);

    int confirm = 0;

    std::cout << "\nShow all? (0/1) ";

    std::cin >> confirm;

    if (confirm) {

        show\_imgs("keypoints", keypoint\_images);

        show\_imgs("matches", matches\_images);

        show\_imgs("result", output\_images, 0.5);

        show\_imgs("input", src\_images, 0.5);

        show\_imgs("recursive", rec\_output, 0.5);

        cv::waitKey(0);

    }

    std::cout << "\n";

}

**panorama.h (функции обработки):**

#include <vector>

#include <utility>

#include <utils/utils.h>

#include <opencv2/opencv.hpp>

void crop\_border(cv::Mat &input\_img)

{

    cv::Mat mask;

    cv::cvtColor(input\_img, mask, cv::COLOR\_BGRA2GRAY);

    cv::threshold(mask, mask, 0, 255, cv::THRESH\_BINARY);

    cv::Mat bounds;

    cv::findNonZero(mask, bounds);

    input\_img = input\_img(cv::boundingRect(bounds));

}

std::pair< std::vector<cv::KeyPoint>, cv::Mat > detect\_keypoints(const cv::Mat &input\_img, cv::Mat &output\_img)

{

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

    std::vector<cv::KeyPoint> keypoints;

    cv::Mat descriptor;

    detector->detect(input\_img, keypoints);

    detector->compute(input\_img, keypoints, descriptor);

    cv::drawKeypoints(input\_img, keypoints, output\_img);

    return std::make\_pair(keypoints, descriptor);

}

void sewing(const cv::Mat &input\_img, cv::Mat &output\_img)

{

    for ( int i = 0; i < input\_img.cols; i++ )

    {

        for ( int j = 0; j < input\_img.rows; j++ )

        {

            if ((output\_img.at<cv::Vec4b>(j, i)[0] == 0 && output\_img.at<cv::Vec4b>(j, i)[1] == 0 && output\_img.at<cv::Vec4b>(j, i)[2] == 0)

                || output\_img.at<cv::Vec4b>(j, i)[3] != 255)

            {

                output\_img.at<cv::Vec4b>(j, i) = input\_img.at<cv::Vec4b>(j, i);

            }

        }

    }

}

cv::Mat match(const std::vector<cv::Mat> &input\_imgs, cv::Mat &output\_img, cv::Mat &matches\_img,

    std::pair<std::vector<cv::KeyPoint>, cv::Mat> &detection\_result1,

    std::pair<std::vector<cv::KeyPoint>, cv::Mat> &detection\_result2)

{

    std::vector<std::vector<cv::KeyPoint>> keypoints = {detection\_result1.first, detection\_result2.first};

    std::vector<cv::Mat> descriptors = {detection\_result1.second, detection\_result2.second};

    cv::BFMatcher matcher(cv::NORM\_HAMMING);

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

    matcher.knnMatch(descriptors[0], descriptors[1], matches, 2);

    std::vector<cv::KeyPoint> matched1, matched2;

    std::vector<cv::Point2f> points1, points2;

    std::vector<cv::DMatch> good\_matches;

    float match\_ratio = 0.5f;

    for (size\_t i = 0; i < matches.size(); i++)

    {

        cv::DMatch first = matches[i][0];

        float dist1 = matches[i][0].distance;

        float dist2 = matches[i][1].distance;

        if (dist1 < match\_ratio \* dist2)

        {

            int new\_i = static\_cast<int>(matched1.size());

            matched1.push\_back(keypoints[0][first.queryIdx]);

            matched2.push\_back(keypoints[1][first.trainIdx]);

            good\_matches.push\_back(cv::DMatch(new\_i, new\_i, 0));

            points1.push\_back(keypoints[0][first.queryIdx].pt);

            points2.push\_back(keypoints[1][first.trainIdx].pt);

        }

    }

    cv::drawMatches(input\_imgs[0], matched1, input\_imgs[1], matched2, good\_matches, matches\_img);

    cv::Mat homography = cv::findHomography(points2, points1, cv::RANSAC);

    cv::Mat stitch\_img;

    cv::warpPerspective(input\_imgs[1], stitch\_img, homography, cv::Size(input\_imgs[1].cols\*2, input\_imgs[1].rows));

    output\_img = stitch\_img(cv::Rect(0, 0, input\_imgs[0].cols\*2, input\_imgs[0].rows));

    sewing(input\_imgs[0], output\_img);

    return output\_img;

}

void panorama(cv::Mat &left, cv::Mat &right, cv::Mat &output, int iter = 0) {

    cv::Mat keypoint\_imageL, keypoint\_imageR;

    std::pair <std::vector<cv::KeyPoint>, cv::Mat> detection\_resultL = detect\_keypoints(left, keypoint\_imageL);

    std::pair <std::vector<cv::KeyPoint>, cv::Mat> detection\_resultR = detect\_keypoints(right, keypoint\_imageR);

    cv::Mat matchimg;

    match({left, right},

            output, matchimg,

            detection\_resultL, detection\_resultR);

}

void rec\_panorama(std::vector<cv::Mat> &input, std::vector<cv::Mat> &output, int iter = 0) {

    if (input.size() != output.size()) {

        std::cout << "IO vectors mismatch";

        return;

    }

    if (iter != output.size()-1) {

        panorama(output[iter], input[iter+1], output[iter+1]);

        rec\_panorama(input, output, iter+1);

    }

}

void recursive\_panorama\_sewing(std::vector<cv::Mat> &input, std::vector<cv::Mat> &output) {

    for (auto &img : input)

        cv::copyMakeBorder(img, img, img.rows, img.rows, img.cols\*2, img.rows\*3, cv::BORDER\_CONSTANT, cv::Scalar(0));

    if (output.empty()) {

        for (size\_t i = 0; i < input.size()-1; ++i) {

            cv::Mat out;

            output.push\_back( out );

        }

    }

    output.insert(output.begin(), input[0]);

    rec\_panorama(input, output);

    output.erase(output.begin());

    for (auto &img : output)

        crop\_border(img);

    for (auto &img : input)

        crop\_border(img);

}

**utils.h (доп. функции отображения и сохранения изображений):**

#pragma once

#include <string>

#include <sstream>

#include <iostream>

#include <iomanip>

#include "wtypes.h"

#include <opencv2/opencv.hpp>

void show\_img(std::string window\_name, cv::Mat img, double resize\_koeff = 1)

{

    RECT desktop;

    const HWND hDesktop = GetDesktopWindow();

    GetWindowRect(hDesktop, &desktop);

    int maxw = desktop.right;

    int maxh = desktop.bottom;

    cv::namedWindow(window\_name, cv::WINDOW\_NORMAL);

    if ( resize\_koeff == 1 ) {

        resize\_koeff = (img.cols > maxw || img.rows > maxh) ? min((double)maxw / img.cols, (double)maxh / img.rows) : resize\_koeff;

    }

    cv::resizeWindow(window\_name, img.cols \* resize\_koeff, img.rows \* resize\_koeff);

    cv::imshow(window\_name, img);

}

void show\_imgs(std::string window\_name, std::vector<cv::Mat> imgs, double resize\_koeff = 1)

{

    for (size\_t i = 0; i < imgs.size(); ++i)

    {

        show\_img(window\_name + std::to\_string(i), imgs[i], resize\_koeff);

    }

}

void imwrite\_vector(std::string path\_to\_file, std::string extension, const std::vector<cv::Mat> &input\_img)

{

    for (size\_t i = 0; i < input\_img.size(); ++i)

    {

        cv::imwrite(path\_to\_file + std::to\_string(i) + "." + extension, input\_img[i]);

    }

}