
Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

C. A. МОЛОДЯКОВ

«КОМПЬЮТЕРНОЕ ЗРЕНИЕ»

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Санкт-Петербург
2018

УДК 004.42

Молодяков С.А.

Компьютерное зрение: лабораторный практикум. СПб.: СПбПУ, 2018.- 174 с.

Лабораторный практикум предназначен для изучения и практического освоения основных методов и алгоритмов компьютерного зрения. В нем рассмотрены вопросы разработки программ по следующим разделам: улучшение изображений, сегментация, выделение объектов и распознавание.

Основное внимание удалено использованию библиотеки компьютерного зрения OpenCV. Работы выполняются на языках Си и Python. Две лабораторные работы посвящены вопросам моделирования оптических устройств в среде GLAD.

Учебное пособие предназначено для студентов, изучающих дисциплины «Методы и средства обработки видеинформации», «Компьютерное зрение», «Анализ изображений и видео» и др.

Печатается по решению редакционно-издательского совета Санкт-Петербургского политехнического университета.

© Молодяков С. А., 2018

© Санкт-Петербургский политехнический
университет Петра Великого

Оглавление

Оглавление	3
Введение.....	4
Лабораторная работа №1 Моделирование в среде GLAD. Элементарный оптический процессор	5
Лабораторная работа №2 Моделирование в среде GLAD. Оптический процессор свертки.....	14
Лабораторная работа №3 " Гистограммы ".....	22
Лабораторная работа №4 "Фильтрация в пространственной области"	30
Лабораторная работа №5 " Фильтрация в частотной области, поиск объекта путем расчета корреляционной функции "	37
Лабораторная работа №6 "Морфологический водораздел (watershed) "	48
Лабораторная работа №7 "Подмена пикселей"	56
Лабораторная работа №8 "Улучшение изображения и сегментация"	63
Лабораторная работа №9 "Сегментация на основе цвете "	74
Лабораторная работа по теме №10 " Применение преобразования Хаара для подавления шумов на изображении "	80
Лабораторная работа №11 "Морфологическая обработка изоображений"	90
Лабораторная работа №12 "Трекер".....	101
Лабораторная работа №13 "Паралельная обработка видео. Параллельные потоки"	107
Лабораторная работа №14 "Паралельная обработка видео. CUDA "	113
Лабораторная работа №15 "Стабилизация видеозаписи с помощью ffmpeg и OpenCV"	119
Лабораторная работа №16 "Использование каскада Хаара для распознавания".....	125
Лабораторная работа №17 "Чтение QR-кодов"	129
Лабораторная работа №18 "Определение преобладающих цветов на изображении" ..	141
Лабораторная работа №19 "Стерео изображение. Стерео геометрия"	145
Темы курсовых работ (итоговых индивидуальных заданий)	171
Библиографический список.....	172

Введение

Известен большой набор методов и алгоритмов компьютерного зрения [1, 2]. Полностью освоить их в рамках одной дисциплины не представляется возможным. В учебном пособии описываются лишь некоторые простые алгоритмы, которые не требуют больших вычислительных затрат для их реализации.

Рассмотрены вопросы разработки программ по следующим разделам: улучшение изображений, сегментация, выделение объектов и распознавание. Уделяется внимание рассмотрению методов параллельной обработки видеоизображений. Методы распознавания, связанные с машинным обучением, рассматриваются лишь частично. Применение нейросетей возможно только при выполнении самостоятельной итоговой работы.

Значительное внимание удалено изучению функций библиотеки компьютерного зрения OpenCV. Две лабораторные работы посвящены вопросам моделирования оптических устройств в среде GLAD.

Представленная информация и примеры программ дают возможность студентам самостоятельно подготовиться к лабораториям, изучить материал в большем объеме, чем предусматривают лабораторные работы.

Практические навыки программирования будут полезны при разработке сложных программных проектов, связанных не только с обработкой изображений, но с обработкой другой информации.

Лабораторная работа №1

Моделирование в среде GLAD. Элементарный оптический процессор

GLAD - General Laser Analysis and Design - <http://www.aor.com>
AOR - Applied Optics Research

Задание на работу

1. Прочитайте данное описание. Установите среду моделирования на вашем компьютере.
2. Загрузите пример программы (sample), которая позволяет моделировать Элементарный оптический процессор. Запустите программу. Научитесь выводить оптическое распределение в разных форматах, на разных расстояниях продвижения оптического пучка и при разных масштабах.
3. Измените схему процессора путем включения новых элементов: линза, плоскопараллельная пластина... Покажите результат в окне вывода.
4. В модели оптического процессора поверните ось распространения света путем включения зеркала или призмы. Покажите результат в окне вывода.

Программные среды для моделирования и разработок оптических систем

1. Optical Research Associates, www.opticalres.com

Code V – программа расчета и проектирования оптических систем.

LightTools – программа 3D моделирования и анализа осветительных систем.

2. Lambda Research Corporation www.lambdares.com

Oslo – программа расчета и проектирования оптических систем. Программа OSLO (Optics Software for Layout and Optimization) предназначена для определения оптимальных размеров и форм элементов оптических систем фото- и видеотехники, систем связи, научных приборов и т.д. Кроме того, она используется для оценки качества оптических систем и для разработки специализированного программного обеспечения для оптического конструирования, тестирования и производства.

Trace Pro – программа оптико - механического моделирования и анализа оптических систем. Позволяет создавать трехмерные модели оптико-механических систем и проводить анализ распределения освещенности в системе, учитывая рассеяние, отражение, поглощение, дифракцию света

Трассировка лучей методом Монте-Карло. Поддерживает экспорт/импорт во все известные 3D форматы и импорт из Zemax, CodeV, Oslo.

3. ZEMAX Development Corporation

ZEMAX – программа расчета и проектирования оптических систем. ZEMAX содержит огромную базу данных по характеристикам различных видов стекол, способен рассчитать сложнейшую конструкцию из систем линз, зеркал, дифракционных решеток, интерференционных и абсорбционных светофильтров и прочих элементов. Он отображает ход лучей в таких системах, наглядно изображая поведение лучей в различных участках спектра, рассчитывает aberrации системы, ее разрешающую способность, потери света и множество других параметров.

4. Optical System Design Inc. www.osdoptics.com

SYNOPSIS – Программа расчета оптических систем.

5. Breault Research Organization www.breault.com

ASAP – программа для моделирования и анализа оптических и осветительных систем

ReflectorCAD – программа для конструирования отражателей.

6. Focus Software Inc., USA www.focus-software.com

LensVIEW - Database of optical designs.

Во всех перечисленных программных средах моделирования и разработок оптических систем используются два метода расчета, основанные на применении теории волновой и геометрической оптики.

Методы распространяющихся пучков (beam propagation methods, BPM) представляют комплексную амплитуду оптического поля в поточечном базисе. **Они используют уравнения волновой оптики (функции дифракции Френеля) чаще всего в параксиальном приближении**, с учетом конечных апертур. Этот метод является основным в компьютерных программах при анализе распространяющихся лазерных пучков. Методы распространяющихся пучков может использоваться при расчете распространения с разложением по плоским волнам или с использованием метода конечных разностей. Такой подход позволяет наиболее полно исследовать оптический процессор. Метод распространяющихся пучков используется в программе GLAD.

Трассировка оптических лучей или лучевой метод, – излучение представлено набором лучей. Используются приближения геометрической оптики. Описание процессора сводится к построению матриц преобразований. Лучевые методы используются для традиционных оптических расчетов, где проектировщик меняет радиус, толщину, марку стекла, асферические коэффициенты и т.п. для минимизации и баланса оптических искажений. Сюда же можно отнести гибридный лучевой метод. Он дает дополнительную возможность учета интенсивности вдоль оптических лучей, так что можно анализировать рассеивание

поверхностью, многогранные оптические сумматоры и другие оптические приложения.

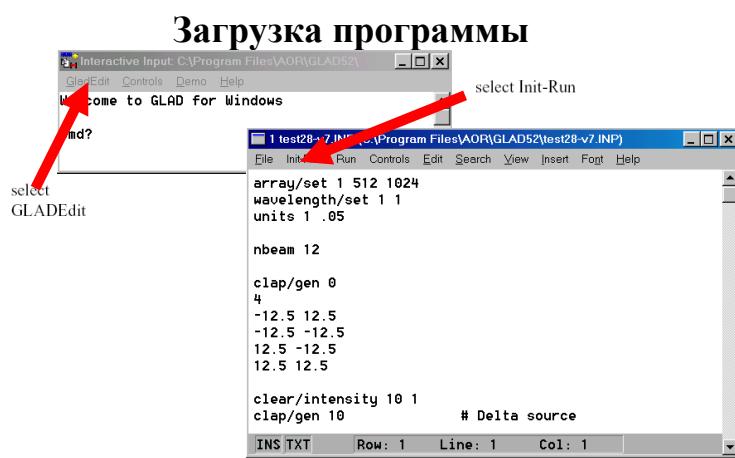
Среда моделирования GLAD

GLAD является аббревиатурой General Laser Analysis and Design, разработан в компании Applied Optics Research (AOR). Целью при создании GLAD была разработка простой в применении программы, которая может моделировать любой тип системы, которая использует когерентный свет. GLAD может быть использован для анализа большого разнообразия оптических и лазерных систем. GLAD включает в себя большинство типов оптических элементов, в том числе линзы, зеркала, отверстия, многие виды решеток, светоделители и т.д.

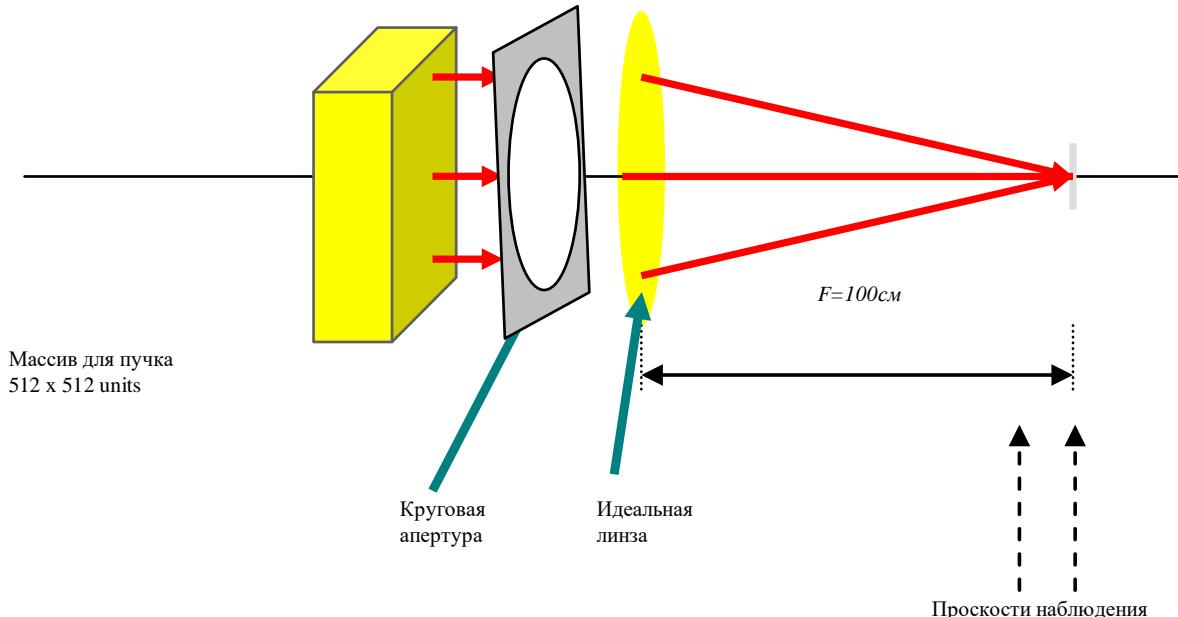
Распределение оптического излучения в оптических пучках представлено комплексным амплитудным распределением, что дает большую возможность анализа, чем геометрическая оптика - программы с представлением пучков набором (и трассировкой) лучей.

В пакете ПО представлено полное описание команд, элементов, примеров:

commands	pdf
examples	pdf
guide	pdf
license	txt
theory	pdf



Простой пример. Элементарный оптический процессор



```

array/set 1 512      # size of computer array to 512 x 512
wavelength 1 1.06    # set wavelength to 1.06 mkm
units/field 1 16      # set half-width of array to 16 cm
clap/cir/con 1 5.     # clear aperture of radius 5 cm
focallength = 100      # define and initialize a variable

lens 1 focallength      # set lens focal length using variable
set/window/rel 30 70 30 70  # set plot range for +-20% about center
title initial distribution
plot(bitmap/intensity/paintiso 1; pause

prop 99                  # propagate 99 cm
title at 99 cm (1 cm from focus)
plot(bitmap/intensity/paintiso 1; pause
prop 1                  # propagate 1cm
set/win/rel 48 52 48 52    # set plot range for +-2% about center
title at paraxial focus
plot(bitmap/intensity/paintiso 1

```

Пояснение команд

array/set 1 512 # Эта команда первая в программе. Она определяет размер массива для пучка 1 как 512 x 512. Массив любого размера должен быть определен.

wavelength 1 1.06 # Эта строка определяет длину волны для пучка 1 как 1.06 микрон.

units/field 1 16 # Устанавливает размер пучка - радиус 16 см. Команда units 1.005 устанавливает размер одной точки.

clap/cir/con 1 5. # Задает и очищает круговую апертуру радиусом 5 см (clap/sqr - квадратная апертура).

lens 1 focallength # Устанавливает линзу для пучка 1 с фокусным расстоянием focallength.

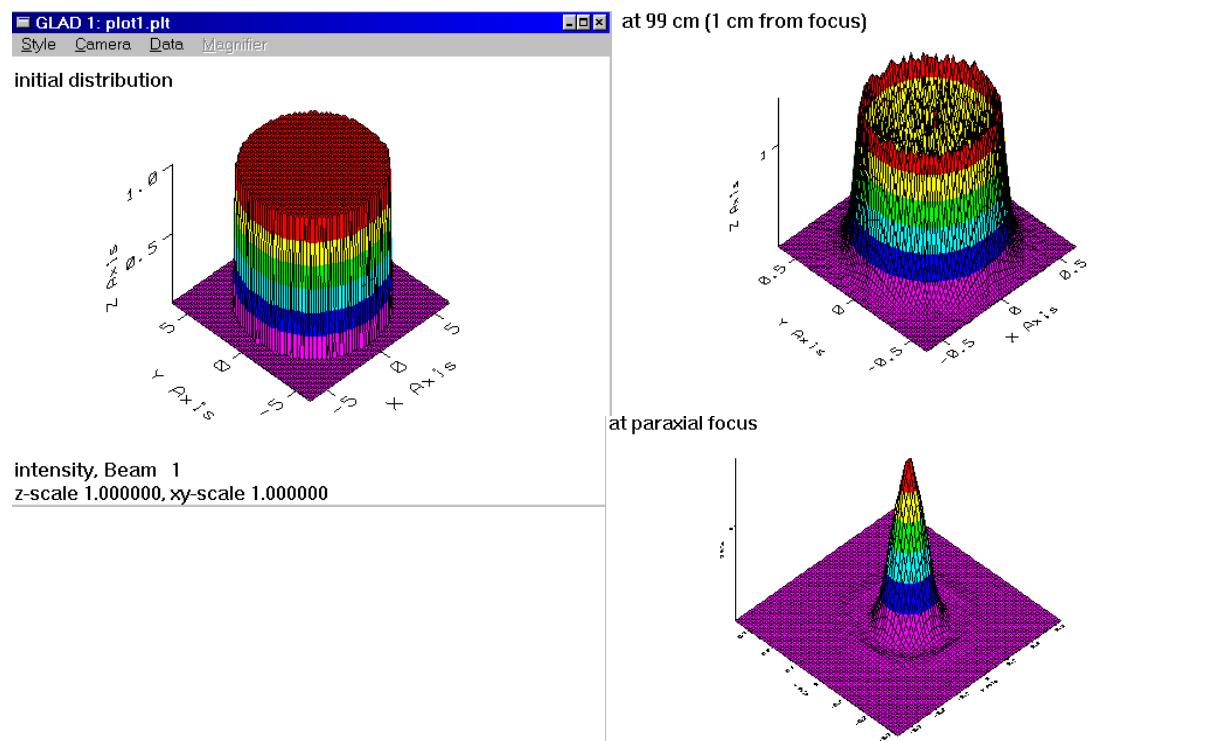
set/window/rel 30 70 30 70 # Устанавливает окно вывода в процентах от общей области.

title initial distribution

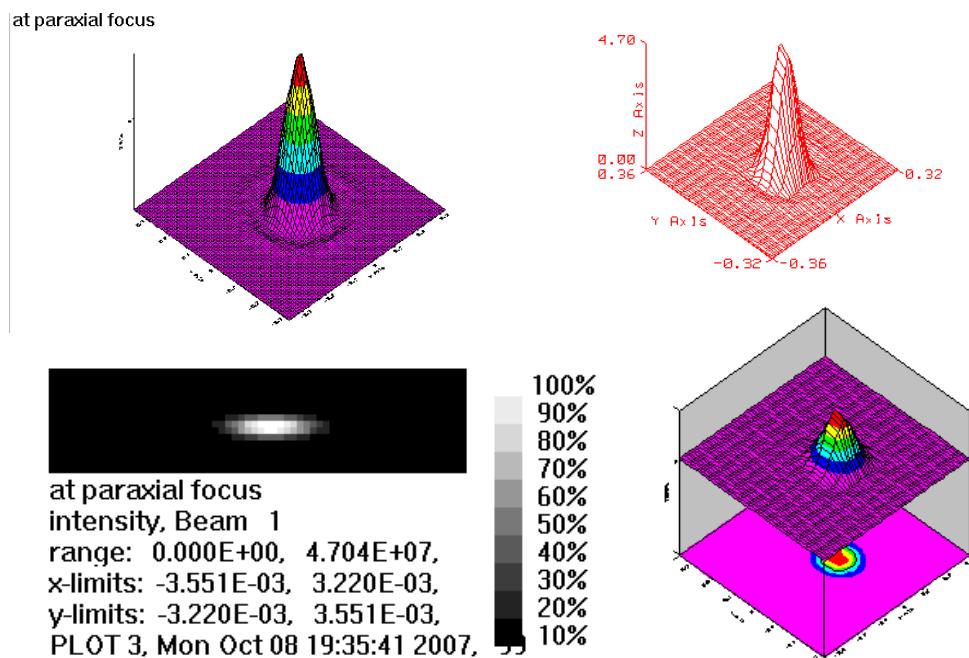
plot(bitmap/intensity/paintiso 1 # Выводит распределение интенсивности пучка из заданного окна.

prop 99 # Пучок распространяется на 99 см с учетом параксиального приближения.

Результаты моделирования



Форматы вывода



Некоторые оптические компоненты

- ADAPT Adaptive mirror model.
- BINARY Binary optics.
- CLAP Implements a clear aperture
- CORNER Corner cube reflector
- CRYSTAL Anisotropic medium for propagation
- DOUBLE Frequency doubling.
- GLASS Properties of optical glass for Lensgroup
- GRATING Grating
- HIGHNA High numerical aperture lens
- HTML Control HTML viewer.
- JSURF Fresnel transmission and reflection coefficients
- LENS Implements ideal lens.**
- LENSARRAY Array of lenses
- MIRROR Ideal mirror
- NOISE Creates complex amplitude noise
- ROD Calculates wavefront rms.
- ROOF Roof prism

Команда установки линзы

<code>lens/sph/element/(Modifier 3)</code>	<code>ibeams f1</code>
<code>lens/xcyl/element/(Modifier 3)</code>	<code>ibeams xfl</code>
<code>lens/ycyl/element/(Modifier 3)</code>	<code>ibeams yfl</code>
<code>lens/toric/element/(Modifier 3)</code>	<code>ibeams xfl yfl</code>
<code>lens/sph/surface/(Modifier 3)</code>	<code>ibeams radius index [(glass)]</code>
<code>lens/xcyl/surface/(Modifier 3)</code>	<code>ibeams xradius index [(glass)]</code>
<code>lens/ycyl/surface/(Modifier 3)</code>	<code>ibeams yradius index [(glass)]</code>
<code>lens/toric/surface/(Modifier 3)</code>	<code>ibeams xradius yradius index [(glass)]</code>
<code>lens/flat/surface</code>	<code>ibeams index</code>
<code>lens/grin</code>	<code>ibeams gamma zstep</code>

Modifier 1	Definition		
<code>/sph</code>	Rotationally symmetric thin lens (2nd order accuracy).		
<code>/xcyl</code>	Cylindrical thin lens.		
<code>/ycyl</code>	Cylindrical thin lens.	<code>ibeams</code>	Beam number.
<code>/toric</code>	Toric thin lens (2nd order accuracy)	<code>f1</code>	Focal length of rotationally symmetric thin lens.
<code>/flat</code>	Surface element having a flat surfa	<code>xfl</code>	Focal length of x-cylindrical thin lens.
<code>/grin</code>	Gradient index lens.	<code>yfl</code>	Focal length of y-cylindrical thin lens.
		<code>radius</code>	Surface radius
		<code>xradius</code>	X-radius of cylinder or toric surface.
		<code>yradius</code>	Y-radius of cylinder or toric surface.

Команды распространения излучения (Propagation)

<code>BEAMS</code>	Turns on/off beams for global commands.
<code>DIST</code>	Diffraction propagation step.
<code>PACK</code>	Packs data for some nonlinear optic commands
<code>PROP</code>	Diffraction propagation in global coordinates.
<code>RDIST</code>	Obsolete command.
<code>ZONE</code>	Exend region of constant units.
<code>ZREFF</code>	Current location of Beam along chief ray.

Формат представления амплитудно-фазового распределения в световом пучке

В пакете GLAD амплитудно-фазовое распределение (АФР) в световом пучке в заданной плоскости представляется в виде массива отсчетов размерностью $2^n \times 2^n$, где n – целое число. Числа, описывающие световое поле в дискретных точках выбранной плоскости, являются комплексными и представлены в пакете в виде суммы вещественной и мнимой частей, то есть $E_{i,j} = \operatorname{Re}(E_{i,j}) + i \cdot \operatorname{Im}(E_{i,j})$

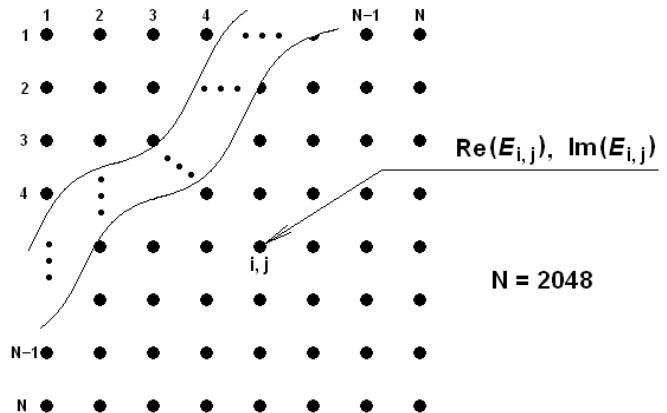
Замечание.

Представление комплексного числа:

$$Y = \text{Re} + j\text{Im} = A \cdot e^{j\varphi}$$

$$A^2 = \text{Re}^2 + \text{Im}^2 = A \cdot A^*$$

$$\varphi = \arctg(\text{Im}/\text{Re})$$



Команды ввода/вывода данных амплитудно-фазового распределения

В GLAD предусмотрены команды ввода-вывода массивов данных в формате числа с плавающей точкой (float32). Формат числа в GLAD:

- бит 1. $= 0, 1;$
- биты 2-9 $= 0 .. 255;$
- биты 10-32. $= 0 .. 8 388 608;$

Команды ввода вывода оптических распределений:

CONNECT Reads beam data from an external file.

DISCONNECT Copies beam data to external file.

INFILE Reads beam data from an external file.

OUTFILE Writes beam data to file.

READ Select device from which to read commands.

connect/keep e:\sine1.bea 1 - связать 1-й пучок с внешним файлом без стирания входного файла. Необходимая последовательность команд:

array/set 1 64 64

connect xxxx 1

Внешний файл должен содержать матрицу 64x64 элемента.

infile C:\Lab\letterA.dat 1 - чтение распределения из внешнего файла для 1-го пучка. Ключи, используемые в команде, позволяют читать разные форматы (в том числе BMP – 8бит) и для распределений: амплитуды, фазы, волнового фронта.

Пример формирования матрицы входного распределения и записи в файл

Формирование входных оптических распределений и вывод данных, полученных в GLAD, могут быть осуществлены в форматах пакетов MATLAB и Mathcad.

Пример записи матрицы распределения в MATLAB :

fid = fopen('e:\temp.mtl','wb'); ; Открытие файла на запись

for i = 1:SIZE

F = m*sin(i*200*pi/2048+Fi)*exp(-(i-C)*dx*log(k)); Задание закона изменения фазы.

fwrite(fid,cos(F),'float32') ; Запись вещественной и мнимой

составляющих поля

fwrite(fid,sin(F),'float32') ; в файл для всех значений по

координате x

end

Пример записи в файл матрицы выходного распределения

В пакете GLAD предусмотрена команда, записывающая массив интенсивности или фазы оптического пучка в выбранной плоскости в файл, формат записи которого совпадает с форматом PRN пакета Mathcad. Данная команда имеет следующий вид для записи значений интенсивности или фазы, соответственно:

outfile/intensity [имя файла]/noheader/excel [номер пучка]

outfile/phase [имя файла]/noheader/excel [номер пучка]

Лабораторная работа №2

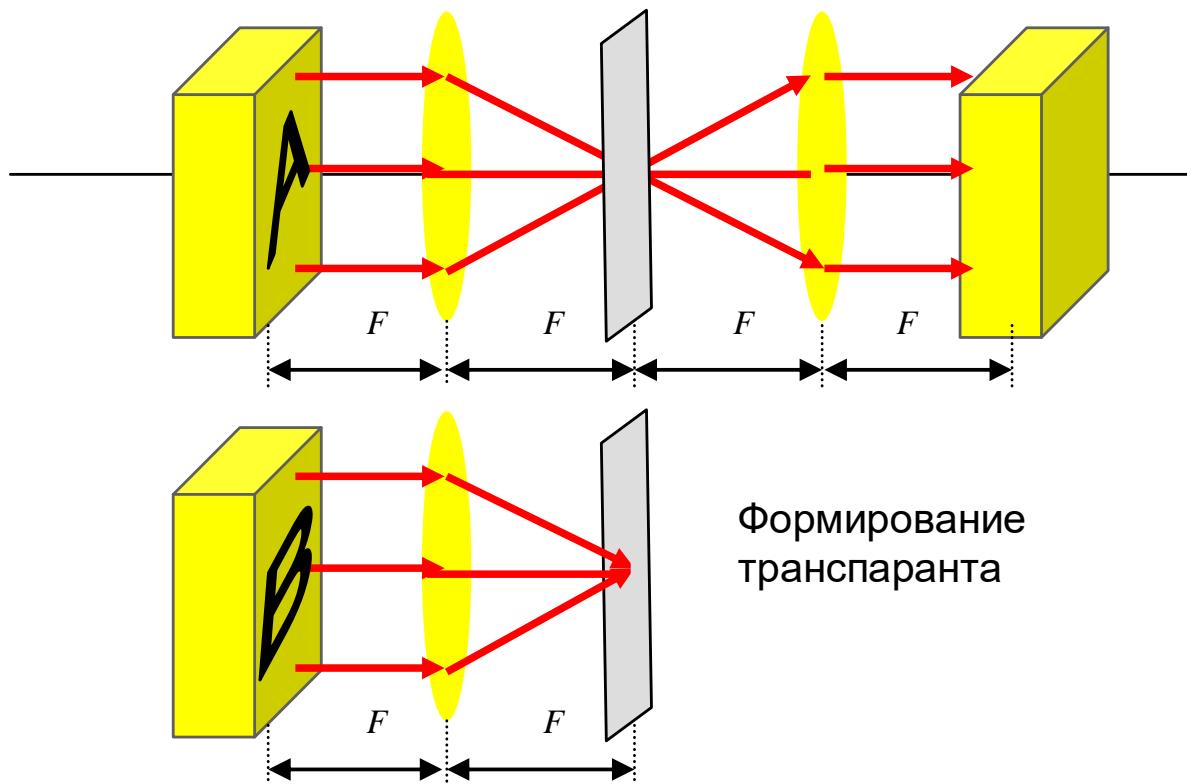
Моделирование в среде GLAD. Оптический процессор свертки

Задание на работу

1. Изучите функционирование Оптического процессора свертки.
2. Загрузите пример программы. Разберитесь с командами.
3. Исследуйте работу процессора:
 - свертка с оптимальным и фазовым фильтрами,
 - влияние поворота изображения...
4. Представьте результаты. Постройте графики.
5. Реализуйте модель процессора распознавания одной из букв при вводе изображения четырех первых букв вашей фамилии.

Оптический процессор свертки (пространственной фильтрации)

Транспарант -
перемножение

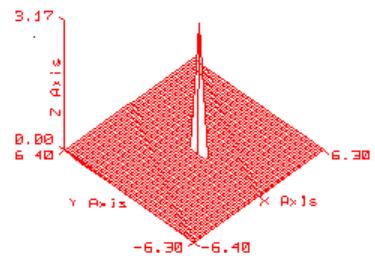
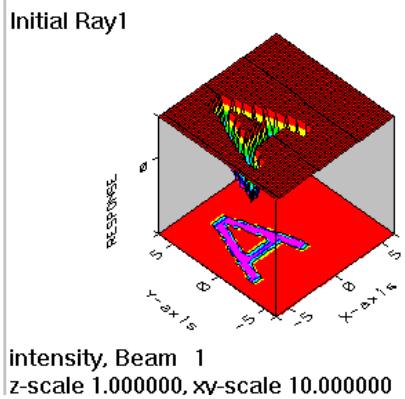


Программа вычисления свертки двух изображений, полученных из внешних файлов

```
array 1 128 128 1 # Создем первый луч с картинкой
infile C:\Lab\letterA.dat/no/binary 1
array 2 128 128 1 # Создем второй луч с картинкой
infile C:\Lab\letterA.dat/no/binary 2
plot/watch plot1.plt # Рисуем интенсивность первого луча
title Initial Ray1
plot(bitmap/intensity/paintiso 1
plot/watch plot2.plt # Рисуем интенсивность второго луча
title Initial Ray2
plot(bitmap/intensity/paintiso 2
convolve/front 1 # Преобразование Фурье для первого луча
convolve/front 2 # Преобразование Фурье для второго луча
plot/watch plot3.plt # Рисуем интенсивность преобразования Фурье
первого луча
title Initial Ray1_intensity
plot(bitmap/intensity/paintiso 1
plot/watch plot4.plt # Рисуем интенсивность преобразования Фурье
второго луча
title Initial Ray2_intensity
plot(bitmap/intensity/paintiso 2
pause # Пауза
mult/beam 1 2 # Умножаем амплитуды лучей
#convolve/beam/fft 1 2
#add/coh 1 2
plot/watch plot3.plt
title Initial Ray1_intensity_Ray2_intensity_operation
plot(bitmap/intensity/paintiso 1
convolve/back 1 # Обратное преобразование Фурье
plot/watch plot4.plt
title Initial Result
plot(bitmap/intensity/paintiso 1
```

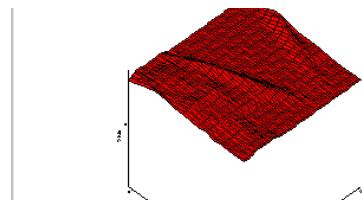
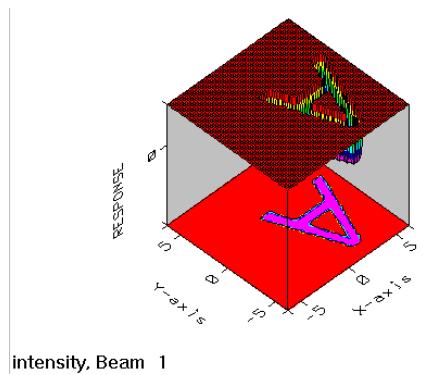
Результаты моделирования

Преобразование Фурье



Обратное преобразование

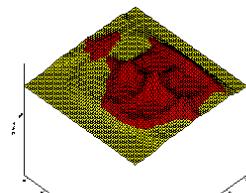
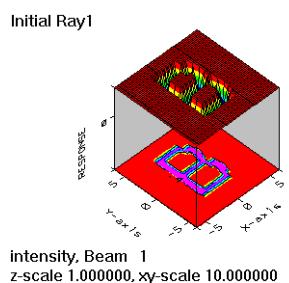
Свертка



intensity, Beam 1
z-scale 1000000000000000.000000, xy-scale 10.000000

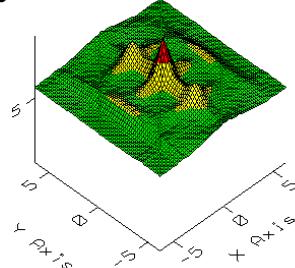
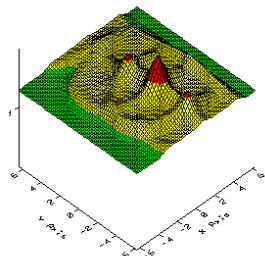
Первый и второй пучок

Свертка



Интенсивность

Корреляционная
функция



В программе использованы команды:

irradiance kbeam - вычисление интенсивности света

$$a(x,y) \rightarrow a(x,y)a(x,y)*$$

rotate kbeam theta - поворот на 180 градусов первого пучка

(корреляционная функция)

Для получения большей амплитуды корреляционного пика необходимо использовать согласованный или фазовый фильтры.

Согласованный фильтр — линейный оптимальный фильтр, построенный исходя из известных спектральных характеристик полезного сигнала и шума. Передаточная функция согласованного фильтра:

A* exp(-jφ)

Фазовый фильтр — фильтр, пропускающий все частоты сигнала с равным усилением, однако изменяющий фазу сигнала. Происходит это при изменении задержки пропускания по частотам. Передаточная функция фазового фильтра: **1* exp(-jφ)**. Вычисление фазового фильтра:

A* exp(-jφ)/A

Вычисление в GLAD:

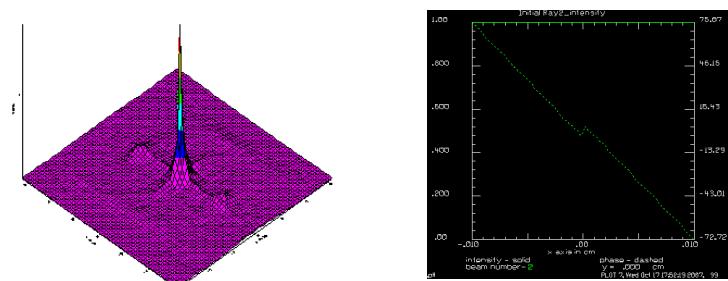
amplitude/abs - взятие амплитуды A

inverse - деление 1/A

conjugate/amplitude - преобразование в компл сопряж пучок

mult/beam - умножение

Результаты свертки с фазовым фильтром

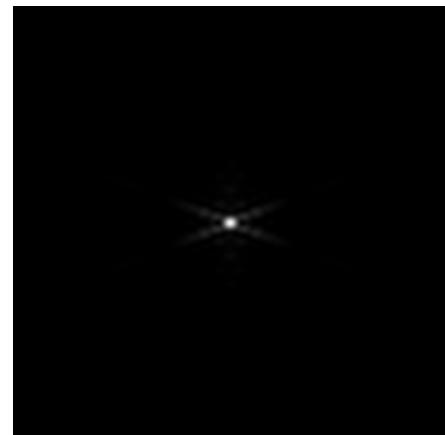
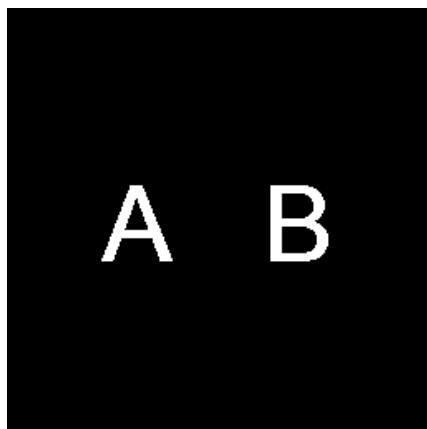


Список некоторых команд GLAD

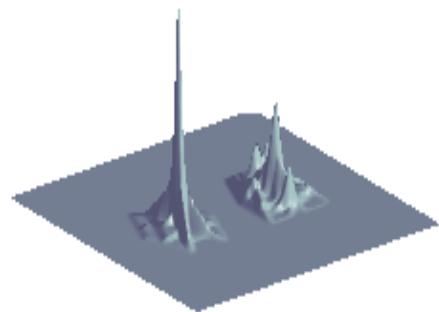
1.	Command:	ADD	Adds beams coherently or incoherently.
2.	Command:	AMPLITUDE	Takes square root or real part of distribution.
3.	Command:	ARRAY	Defines beam array size and polarization state.
4.	Command:	CLAP	Implements a clear aperture.
5.	Command:	CLAP/GEN	Implements a clear aperture of general shape.
6.	Command:	CONJUGATE	Conjugates the beam.
7.	Command:	CONNECT	Reads beam data from an external file.
8.	Command:	CONVOLVE	Convolves beam with a smoothing function.
9.	Command:	COPY	Copies one beam to another.
10.	Command:	DISCONNECT	Copies beam data to external file.
11.	Command:	ENERGY	Calculate the total energy (or power).
12.	Command:	FIELD	Displays table of complex amplitude values.
13.	Command:	GAUSSIAN	Set beam to gaussian function.
14.	Command:	GRATING	Grating.
15.	Command:	INFILE	Reads beam data from an external file.
16.	Command:	INT2PHASE, INT2WAVES	Converts irradiance function to phase screen.
17.	Command:	INTEGRATE	Integrates intensity or amplitude.
18.	Command:	INVERSE	Calculates inverse of beam distribution.
19.	Command:	MULT	Multiply the beam by a constant or another beam.
20.	Command:	NOISE	Creates complex amplitude noise.
21.	Command:	NORMALIZE	Normalizes non-zero values in beam.
22.	Command:	OBS	Implement obstruction.
23.	Command:	OPTIMIZE	Damples least squares optimization.
24.	Command:	OUTFILE	Writes beam data to file.
25.	Command:	PEAK	Sets and displays peak irradiance.
26.	Command:	PHASE	Piston and random phase aberration.
27.	Command:	PHASE/PISTON	Adds piston error.
28.	Command:	PHASE/RANDOM	Adds smoothed random phase.
29.	Command:	PHASE/SCREEN	Adds smoothed random phase (quick).
30.	Command:	PHASE2INT	Converts phase distribution to intensity distribution.
31.	Command:	PLOT	Various graphical plots. Diagnostics
32.	Command:	PLOT/XSLICE	Plot a slice through x-direction.
33.	Command:	RESCALE	Rescale the beam distribution in the array.
34.	Command:	ROTATE	Rotate distribution in the array.
35.	Command:	SINC	Define sinc function.
36.	Command:	TITLE	Defines the plot title.
37.	Command:	UDATA	Create and display summary plots
38.	Command:	UNIFORMITY	Calculate irradiance nonuniformity
39.	Command:	UNITS	Set and display sample spacing
40.	Command:	VARIABLES	Declare and set variable values.
41.	Command:	WAVELENGTH	Set and display beam wavelength
42.	Command:	WAVES2INT	Wavefront transformed to intensity

Ожидаемые результаты работы процессора

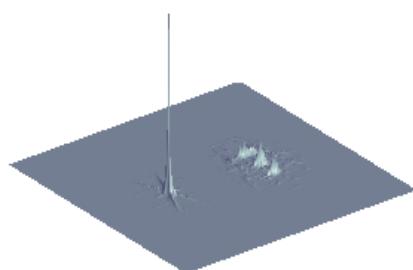
Входной сигнал - изображения букв 'A' and 'B' Fourier transform 'A'



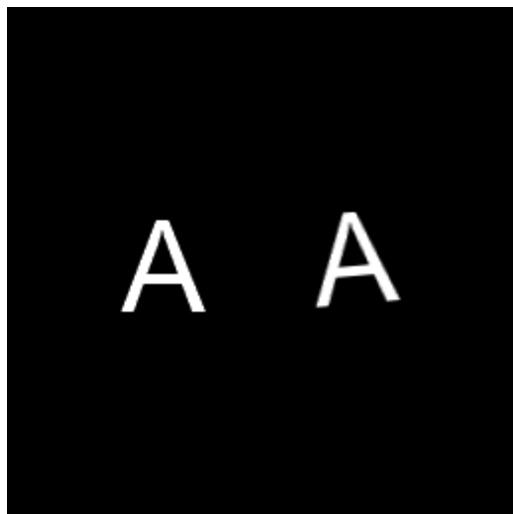
The intensity of the cross correlation



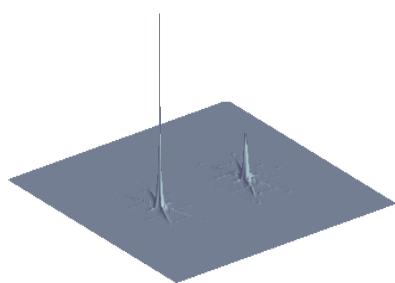
При использовании фазового фильтра



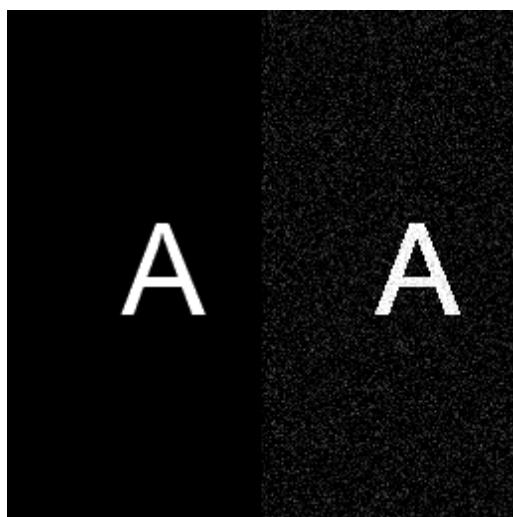
Поворот 'A' на 5 градусов



Корреляционная функция после поворота

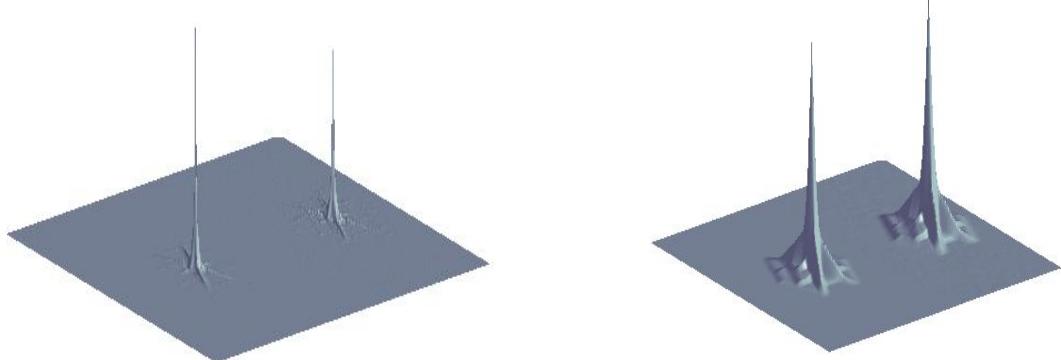


Изображение буквы 'A' в шумах

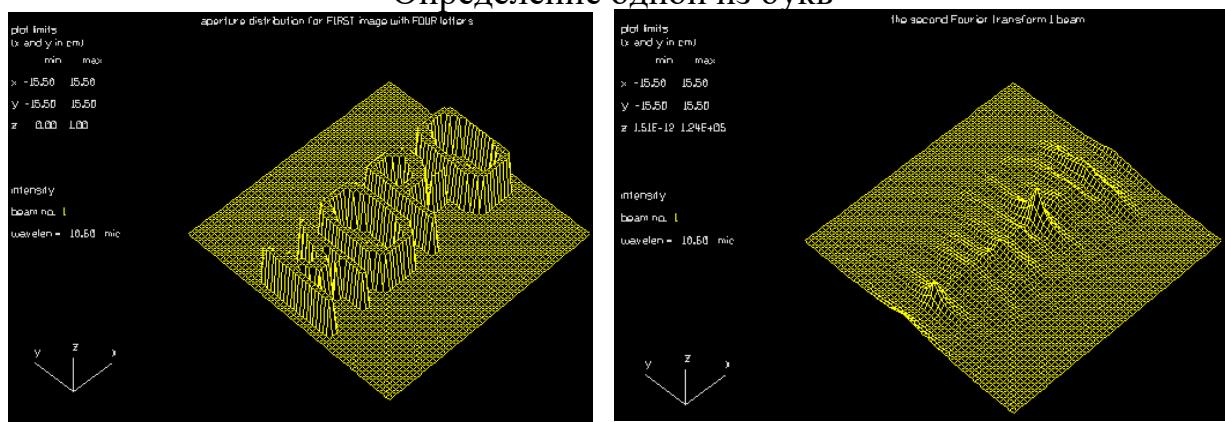


Используем фазовый фильтр

Используем согласованный фильтр



Определение одной из букв

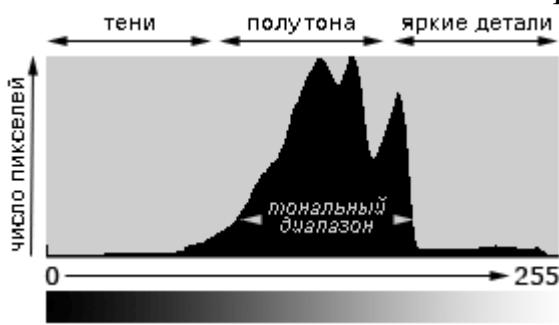


Лабораторная работа №3 " Гистограммы "

В данной работе рассматриваются вычисление гистограмм для чёрно-белых и цветных изображений и видео, выравнивание гистограмм, многомерные гистограммы. Вам будут продемонстрированы несколько примеров работы с гистограммами в OpenCV, а затем будут предложены к выполнению несколько заданий, основанных на этих примерах.

Вычисление гистограмм

Один из наиболее распространенных дефектов фотографических, сканерных и телевизионных изображений – слабый контраст. Дефект во многом обусловлен ограниченностью диапазона воспроизводимых яркостей. Под *контрастом* понимается разность максимального и минимального значений яркости. Контрастность изображения можно повысить за счет изменения яркости каждого элемента изображения и увеличения диапазона яркостей. Существует несколько методов, основанных на вычислении гистограммы.



Допустим, что имеется изображение в оттенках серого, интенсивность пикселей которого изменяется в пределах значений от a до b , где $a \geq 0$ и $b \leq 255$. Для изображения можно построить гистограмму со столбцами, отвечающими количеству пикселей определенной

интенсивности. Такого рода гистограмма позволяет представить распределение оттенков на изображении. В общем случае под **гистограммой** понимается коллекция целочисленных значений, каждое из которых определяет количество точек, обладающих некоторым свойством или принадлежащих определенному бину. На практике гистограммы применяются, чтобы получить статистическую картину о распределении каких-либо данных (пикселей, векторов признаков, направлений градиента во всех точках изображения и т.п.).

Рассмотрим прототипы стандартных функций OpenCV для создания гистограмм.

```
void calcHist(const Mat* arrays, int narrays,
    const int* channels, const Mat& mask,
    MatND& hist, int dims, const int* histSize,
    const float** ranges, bool uniform=true,
    bool accumulate=false)
```

Параметры:

arrays – исходные массивы данных или изображения. Должны иметь одинаковую глубину (CV_8U или CV_32F) и размер.

`narrays` – количество исходных массивов данных.

`channels` – массив индексов каналов в каждом входном массиве, по которым будет вычисляться гистограмма.

`mask` – маска, на которой считается гистограмма. Опциональный параметр. Если маска не пуста, то она представляется 8-битной матрицей того же размера, что и каждый исходный массив. При построении гистограммы учитываются только элементы массивов, которые соответствуют ненулевым элементам маски. Если маска пуста, то построение гистограммы выполняется на полном наборе данных.

`hist` – результирующая гистограмма, плотная в случае использования первого прототипа функции, разреженная – в случае второго. Для хранения плотной гистограммы используется структура данных `MatND`, для разреженной – `SparseMat`. `MatND` представляется в виде n-мерного массива, `SparseMat` – хэш-таблицей ненулевых значений [10].

`dims` – размерность гистограммы. Параметр принимает положительные целочисленные значения, не превышающие `CV_MAX_DIMS = 32`.

`histSize` – количество бинов по каждой размерности гистограммы.

`ranges` – интервалы изменения значений по каждой размерности гистограммы. Если гистограмма равномерная (`uniform = true`), то для любой размерности i достаточно указать только нижнюю границу изменения (по существу значение, соответствующее первому бину), верхняя граница будет совпадать с `histSize[i]-1`.

`uniform` – флаг, который определяет тип диаграммы (равномерная или нет).

`accumulate` – флаг, указывающий на необходимость очищения гистограммы перед непосредственными вычислениями. Использование данного флага позволяет использовать одну и ту же гистограмму для нескольких множеств массивов или обновлять гистограмму во времени.

Выравнивание гистограмм

Существует три основных метода повышения контраста изображения:

- линейная растяжка гистограммы (линейное контрастирование),
- нормализация гистограммы,
- выравнивание (equalization) гистограммы.

Линейная растяжка сводится к присваиванию новых значений интенсивности каждому пикселю изображения. Если интенсивности исходного изображения изменились в диапазоне от i_1 до i_2 , тогда необходимо линейно "растянуть" указанный диапазон так, чтобы значения изменились от 0 до 255. Для этого достаточно пересчитать старые значения интенсивности для всех пикселей относительно нового интервала.

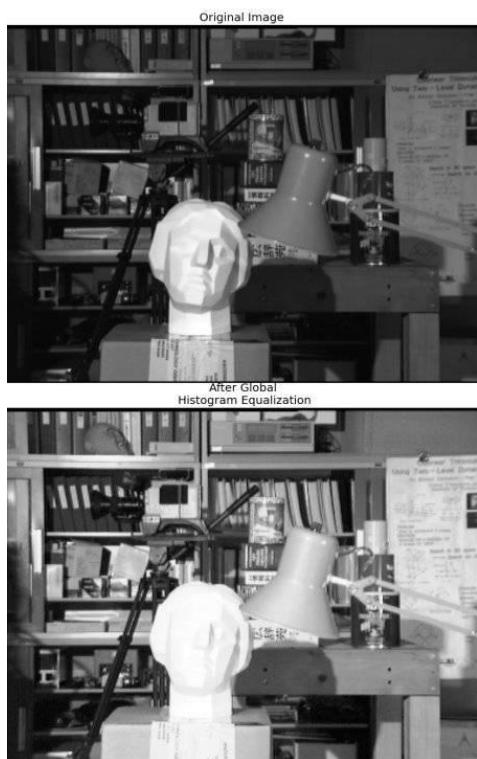
Нормализация гистограммы, в отличие от предыдущего метода, обеспечивает растяжку не всего диапазона изменения интенсивностей, а

толь ко его наиболее информативной части. Под информативной частью понимается набор пиков гистограммы, т.е. интенсивности, которые чаще остальных встречаются на изображении. Бины, соответствующие редко встречающимся интенсивностям, в процессе нормализации отбрасываются, далее выполняется обычная линейная растяжка получившейся гистограммы.

Выравнивание гистограмм – это один из наиболее распространенных способов. Цель выравнивания состоит в том, чтобы все уровни яркости имели бы одинаковую частоту, а гистограмма соответствовала равномерному закону распределения. В библиотеке OpenCV реализована функция **equalizeHist**, которая обеспечивает повышение контрастности изображения посредством выравнивания гистограммы. Прототип функции показан ниже.

```
void equalizeHist(const Mat& src, Mat& dst)
```

Адаптивное выравнивание



Рассмотренное ранее выравнивание гистограммы изображения не всегда способно давать качественный результат. Это происходит из-за того, что обычный алгоритм выравнивания действует в зависимости от контраста всего изображения целиком. Рассмотрим это на примере изображения скульптуры с нормальным контрастом на затемненном фоне. Применим к данному изображению глобальное выравнивание. Да, контраст фона улучшился, но теперь скульптура пересвеченна.

Чтобы решить данную проблему, используется адаптивное выравнивание гистограммы (). Изображение делится на маленькие блоки, называемые «плитками» (tiles), по умолчанию размером 8x8. Затем

для каждого из этих блоков строится гистограмма и выравнивается как обычно. Таким образом, в небольшой области гистограмма ограничивалась бы небольшим регионом. Однако подобное выравнивание чревато усилением шума. Чтобы этого избежать, применяется адаптивное выравнивание гистограмм с ограничением на контрастность (contrast limiting adaptive histogram equalization (**CLAHE**)). Если какой-либо гистограммный лоток находится выше указанного предела контрастности (по умолчанию 40 в OpenCV), эти пиксели обрезаются и равномерно распределяются по другим ячейкам перед применением выравнивания

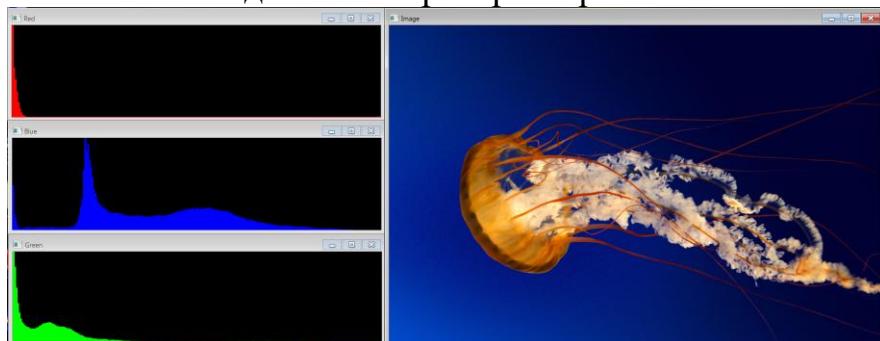
гистограммы. После выравнивания для удаления артефактов в границах плитки применяется билинейная интерполяция.

Добавление CLAHE в OpenCV происходит следующим образом: функция `createCLAHE()` создает CLAHE объект, а метод `apply(channel, dst)` применяет его к каналу.

Демонстрационные программы

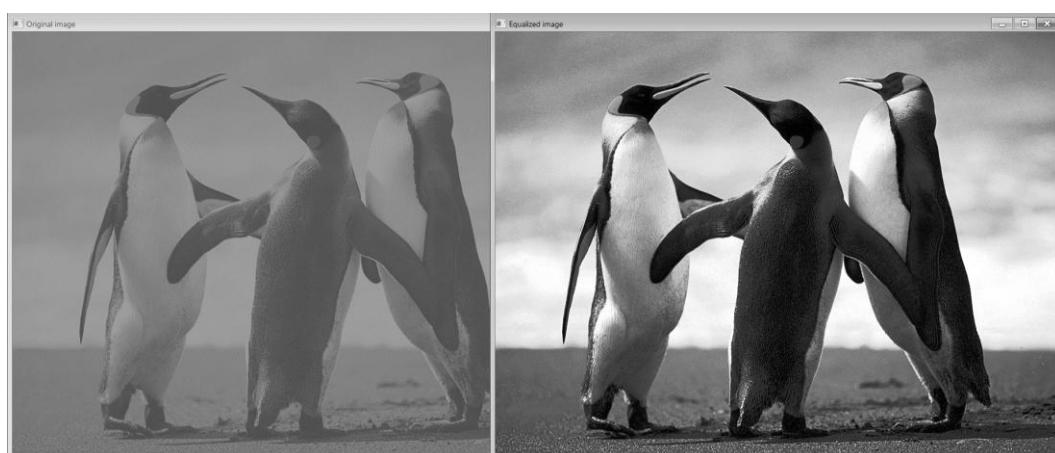
Первая подпрограмма

Данная подпрограмма осуществляет построение и отображение гистограмм по каждому каналу цветного изображения. Программа получает путь к изображению, расщепляет полученную матрицу по каналам (`split`) и вычисляет гистограмму для каждого канала изображения (`calcHist`). Необходимо помнить, что в OpenCV каналы изображения хранятся в порядке BGR, а не в RGB. Далее гистограммы выводятся на экран в качестве изображений с помощью функции `getImageFromHistogram`, применение которой рекомендуется вам при выполнении заданий лабораторной работы.



Вторая подпрограмма

Данная подпрограмма получает на вход черно-белое изображение со слабым контрастом и выполняет выравнивание гистограммы с целью улучшения изображения.



Задания

В рамках данной лабораторной работы Вам предложено выполнить 4 задания. Каждое задание выполняется в соответствующей ему функции **task** из предоставленного шаблона. Чтобы запустить задание необходимо запустить программу и ввести номер задания. Вам также предоставляются демонстрационные примеры, на основе которых вы можете написать свои программы.

1. Добавить реализацию вычисления *гистограммы яркости* для цветного изображения (а не гистограмм каналов, как в примере 1)
2. Добавить реализацию вычисления гистограмм для потокового видео с веб-камеры.
3. Добавить реализацию выравнивания гистограммы цветного изображения.
Подсказка: подумайте, подходит ли цветовая модель RGB для данной задачи. Возможно, стоит конвертировать изображение цветовую модель YCrCb, где канал Y отвечает за яркость, а два других за цвет, и выравнивать гистограмму по каналу яркости.
4. Добавить реализацию адаптивного выравнивания цветного изображения по методу CLAHE

Использованные функции

Функция	Описание
Mat::zeros(int rows, int cols, int type);	Создание пустой матрицы заданного размера
void fillConvexPoly(Mat& img, const Point* pts, int npts, const Scalar& color, int lineType = LINE_8, int shift = 0);	Рисование закрашенного полигона
Mat imread(const String& filename, int flags = IMREAD_COLOR);	Чтение изображения
void cv::imshow(const String & winname, InputArray mat)	Отображает изображение в указанном окне.
int cv::waitKey(int delay = 0)	Ожидает нажатия клавиши.
calcHist(...)	См. пункт вычисление гистограмм.
equalizeHist(...)	См. пункт выравнивание гистограмм.

Шаблон

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

using namespace cv;

void demo1();
void demo2();

void task1();
void task2();
void task3();
void task4();

Mat getImageFromHistogram(Mat hist, float scaleX = 1, float scaleY = 1, Scalar
fillColor = Scalar(255, 255, 255)) {
    double maxVal = 0;
    minMaxLoc(hist, 0, &maxVal, 0, 0);
    int rows = 64; //default height size
    int cols = hist.rows; //get the width size from the histogram
    Mat histImg = Mat::zeros(rows*scaleX, cols*scaleY, CV_8UC3);
    //for each bin
    for (int i = 0; i<cols - 1; i++) {
        float histValue = hist.at<float>(i, 0);
        float nextValue = hist.at<float>(i + 1, 0);
        Point pt1 = Point(i*scaleX, rows*scaleY);
        Point pt2 = Point(i*scaleX + scaleX, rows*scaleY);
        Point pt3 = Point(i*scaleX + scaleX, (rows - nextValue * rows /
maxVal)*scaleY);
        Point pt4 = Point(i*scaleX, (rows - nextValue * rows / maxVal)*scaleY);

        int numPts = 5;
        Point pts[] = { pt1, pt2, pt3, pt4, pt1 };

        fillConvexPoly(histImg, pts, numPts, fillColor);
    }
    return histImg;
}

int main(int argc, char** argv) {
    std::cout << "OPENCV HISTOGRAM\n\n"
              << "Demos:\n"
              << "1. Histogram of a RGB image.\n"
              << "2. Histogram equalization of a grayscale image.\n\n"
              << "Tasks:\n"
              << "3. Histogram of intensity.\n"
              << "4. Histogram of a video.\n"
              << "5. Histogram equalization of an RGB image.\n"
              << "6. Adaptive histogram equalization.\n"
              << "7. Two dimentional histograms.\n\n";
    std::cout << "Which program would you like to run?\n";
    int task = 0;
    std::cin >> task;
    std::cout << "\n";
}
```

```

switch (task) {
case 1:
    demo1();
    break;
case 2:
    demo2();
    break;
case 3:
    task1();
    break;
case 4:
    task2();
    break;
case 5:
    task3();
    break;
case 6:
    task4();
    break;
}

std::cin.get();
return 0;
}

void demo1() {
    Mat img = Mat();

    while (img.empty()) {
        std::cout << "Enter path to an image: ";
        std::string path;
        std::cin >> path;

        img = imread(path, CV_LOAD_IMAGE_COLOR);

        if (img.empty()) {
            std::cout << "Error reading the image. Try another one.\n";
        }
    }

    MatND hist;
    Mat histImg;
    int nbins = 256; // lets hold 256 levels
    int hsize[] = { nbins }; // just one dimension
    float range[] = { 0, 255 };
    const float *ranges[] = { range };
    int chnls[] = { 0 };

    std::vector<Mat> channels;
    split(img, channels);

    calcHist(&channels[0], 1, chnls, Mat(), hist, 1, hsize, ranges);
    histImg = getImageFromHistogram(hist, 3, 3, Scalar(255, 0, 0));
    imshow("Blue", histImg);

    calcHist(&channels[1], 1, chnls, Mat(), hist, 1, hsize, ranges);
    histImg = getImageFromHistogram(hist, 3, 3, Scalar(0, 255, 0));
    imshow("Green", histImg);
}

```

```

calcHist(&channels[2], 1, chnls, Mat(), hist, 1, hsize, ranges);
histImg = getImageFromHistogram(hist, 3, 3, Scalar(0, 0, 255));
imshow("Red", histImg);

imshow("Image", img);

waitKey(0);
}

void demo2() {
    Mat img = Mat();

    while (img.empty()) {
        std::cout << "Enter path to an image: ";
        std::string path;
        std::cin >> path;

        img = imread(path, CV_LOAD_IMAGE_COLOR);

        if (img.empty()) {
            std::cout << "Error reading the image. Try another one.\n";
        }
    }

    cvtColor(img, img, COLOR_BGR2GRAY);

    Mat image_equalized;
    equalizeHist(img, image_equalized);

    imshow("Original image", img);
    imshow("Equalized image", image_equalized);

    waitKey(0);
}

void task1() {
    std::cout << "You have to complete this assignment by yourself.\n";
}

void task2() {
    std::cout << "You have to complete this assignment by yourself.\n";
}

void task3() {
    std::cout << "You have to complete this assignment by yourself.\n";
}

void task4() {
    std::cout << "You have to complete this assignment by yourself.\n";
}

```

«Пахоруков Денис, гр. 23534/4. 2018 год»

Лабораторная работа №4 "Фильтрация в пространственной области"

В мире компьютерного зрения фильтрация используется для изменения изображений. Эти изменения могут быть связаны с выделением краев изображения, размытием, добавлением объектов или удалением нежелательных. Существует множество причин, по которым вы можете использовать фильтрацию изображений. Например, съемка при солнечном свете или темноте будет влиять на четкость изображений - вы можете использовать фильтры для изменения изображения, чтобы получить то, что от него требуется. Аналогично, у вас может быть размытое или «шумное» изображение, требующее уточнения и фокусировки.

Задание на выполнение работы

1. Изучите алгоритмы свертки и функцию OpenCV cvCopyMakeBorder() для обработки изображений на границах.
2. Изучите алгоритмы свертки и функцию OpenCV cvFilter2D () для обработки изображений.
3. Напишите программу, используя фильтр «увеличения яркости».
4. Напишите программу, используя фильтр «BORDER_REPLICATE» для границ.

Описание Image Filtering. Что мы понимаем под свёрткой?

Свёртка (convolution) — это математическая операция

$$y(n) = \sum_{m=0}^{\infty} h(m)x(n - m)$$

В случае работы с изображениями свёртка — это операция вычисления нового значения заданного пикселя, при которой учитываются значения окружающих его соседних пикселей. Главным элементом свёртки является **ядро свёртки** — это матрица (произвольного размера и отношения сторон; чаще всего используется квадратная матрица (по умолчанию, размеры 3x3)).

[][][]
[][я][][]
[][][][]

У ядра свёртки есть важный параметр — **якорь** — это элемент матрицы (чаще всего — центр), который прикладывается к заданному пикселью изображения.

Работает свёртка очень просто:
При вычислении нового значения выбранного пикселя изображения, ядро свёртки прикладывается своим центром к этому пикслю. Соседние пиксели так же накрываются ядром.
Далее, вычисляется сумма произведений значений пикселей изображения на значения, накрывшего данный пиксль элемента ядра.
Полученная сумма и является новым значением выбранного пикселя.
Теперь, если применить свёртку к каждому пикслю изображения, то получится некий эффект, зависящий от выбранного ядра свертки.

Пример:

пусть у нас есть клеточное поле, которое соответствует пикселям исходного изображения:

```
[47][48][49][ ][ ][ ][ ][ ][ ][ ]  
[47][50][42][ ][ ][ ][ ][ ][ ][ ]  
[47][48][42][ ][ ][ ][ ][ ][ ][ ]  
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]  
[ ][ ][ ][ ][ ][ ][ ][ ][ ]
```

и есть ядро свёртки, представляющее собой матрицу 3x3 с якорем в центре и элементами:

```
[0][1][0]  
[0][0][0]  
[0][0][0]
```

Результат операции наложения нашего ядра на пиксель со значением **50**:

```
[ ][ ][ ][ ][ ][ ][ ][ ][ ]  
[ ][48][ ][ ][ ][ ][ ][ ][ ]  
[ ][ ][ ][ ][ ][ ][ ][ ][ ]  
[ ][ ][ ][ ][ ][ ][ ][ ][ ]  
[ ][ ][ ][ ][ ][ ][ ][ ][ ]
```

Всё очень просто:

результат = $47*0 + 48*1 + 49*0 + 47*0 + 50*0 + 42*0 + 47*0 + 48*0 + 42*0$
= 48

В OpenCV операция свёртки реализуется функцией **cvFilter2D()**

```
CVAPI(void) cvFilter2D( const CvArr* src, CvArr* dst, const CvMat* kernel,  
                      CvPoint anchor CV_DEFAULT(cvPoint(-1,-1)));
```

Т.о. накладывая ядро с разными коэффициентами можно получить различные эффекты.

Алгоритм свёртки на краях изображений

Как должен вести себя алгоритм свёртки на краях изображения? Если взять рассмотренный пример, то как нужно рассчитывать свёртку, если приложить якорь ядра к пикселью со значением **47**? Хорошим решением этой проблемы может быть создание изображение большего размера, чем исходное, у которого на краях будут заданное значение пикселей. Начинать обработку картинки нужно тогда не с 0-го, а с 1-го пикселя (и до n-1-го пикселя):

```
[0][0][0][0][0][0][0][0][0][0][0][0]  
[0][47][48][49][ ][ ][ ][ ][ ][ ][0]  
[0][47][50][42][ ][ ][ ][ ][ ][ ][0]  
[0][47][48][42][ ][ ][ ][ ][ ][ ][0]  
[0][ ][ ][ ][ ][ ][ ][ ][ ][ ][0]  
[0][ ][ ][ ][ ][ ][ ][ ][ ][ ][0]  
[0][0][0][0][0][0][0][0][0][0][0][0]
```

В OpenCV уже есть функция реализующая копирование изображения с последующим окружением границей с заданным значением — **cvCopyMakeBorder()**

```
CVAPI(void) cvCopyMakeBorder( const CvArr* src, CvArr* dst, CvPoint  
offset,  
                           int bordertype, CvScalar value  
                           CV_DEFAULT(cvScalarAll(0)));
```

— копирует исходное изображение, окружая его границей

Таблица функций

Используемые функции	Назначение
<code>cvLoadImage()</code>	Получение изображения
<code>cvCloneImage(image)</code>	Клонирование изображения
<pre>CVAPI(void) cvFilter2D(const CvArr* src, CvArr* dst, const CvMat* kernel, CvPoint anchor CV_DEFAULT(cvPoint(-1,-1)));</pre>	Накладываем фильтры
<code>cvReleaseImage(&image);</code>	Освобождение ресурсов
<code>cvCreateImage (cvSize(image->width + 20, image->height + 20), image->depth, image->nChannels);</code>	Создаем картинки
<code>cvCopyMakeBorder(image, dst, cvPoint(10, 10), IPL_BORDER_CONSTANT, cvScalar(250));</code>	Обрамляем границы различными эффектами

Виды фильтров для изображений

- 1) Сглаживание (*smoothing*);
 - 2) Увеличение чёткости (*increasing clarity*);
 - 3) Увеличение яркости (*brightness*);
 - 4) Затемнение (*blackout*) и т. п.

Виды фильтров на границы изображений

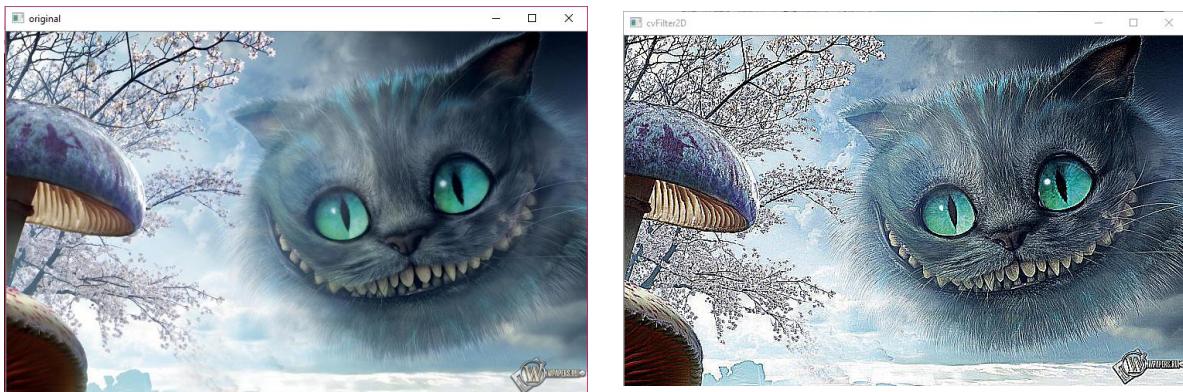
- 1) $BORDER_REPLICATE = IPL_BORDER_REPLICATE;$
 - 2) $BORDER_CONSTANT = IPL_BORDER_CONSTANT;$

```

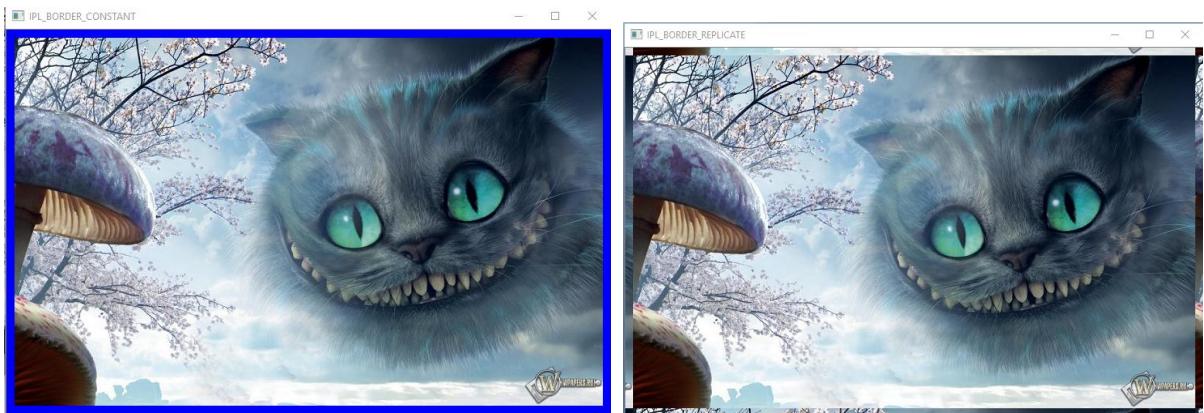
3)BORDER_REFLECT = IPL_BORDER_REFLECT;
4)BORDER_WRAP = IPL_BORDER_WRAP;
5)BORDER_REFLECT_101 = IPL_BORDER_REFLECT_101;
6)BORDER_REFLECT101 = BORDER_REFLECT_101;
7)BORDER_TRANSPARENT = IPL_BORDER_TRANSPARENT;
8)BORDER_DEFAULT = BORDER_REFLECT_101;
9)BORDER_ISOLATED = 16

```

Примеры работы программы Фильтры на изображения



Фильтры на границах



Шаблон программы

Применение фильтра «повышение резкости»

```

#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>

IplImage* image = 0;
IplImage* dst = 0;

int main(int argc, char* argv[])
{
    // имя картинки задаётся первым параметром

```

```

char* filename = argc == 2 ? argv[1] : "Image3.jpg";
// получаем картинку
image = cvLoadImage(filename, 1);

printf("[i] image: %s\n", filename);
assert(image != 0);

// клонируем картинку
dst = cvCloneImage(image);

// окно для отображения картинки
cvNamedWindow("original", CV_WINDOW_AUTOSIZE);
cvNamedWindow("cvFilter2D", CV_WINDOW_AUTOSIZE);

float kernel[9];
kernel[0] = -1;
kernel[1] = -1;
kernel[2] = -1;

kernel[3] = -1;
kernel[4] = 9;
kernel[5] = -1;

kernel[6] = -1;
kernel[7] = -1;
kernel[8] = -1;

// матрица
CvMat kernel_matrix = cvMat(3, 3, CV_32FC1, kernel);

// накладываем фильтр
cvFilter2D(image, dst, &kernel_matrix, cvPoint(-1, -1));

// показываем картинку
cvShowImage("original", image);
cvShowImage("cvFilter2D", dst);

// ждём нажатия клавиши
cvWaitKey(0);

// освобождаем ресурсы
cvReleaseImage(&image);
cvReleaseImage(&dst);

// удаляем окна
cvDestroyAllWindows();
return 0;
}

```

Применение фильтра на границы изображения

```

#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>

IplImage* image = 0;
IplImage* dst = 0;
IplImage* dst2 = 0;

int main(int argc, char* argv[])
{
    // имя картинки задаётся первым параметром

```

```

char* filename = argc == 2 ? argv[1] : "Image3.jpg";
// получаем картинку
image = cvLoadImage(filename, 1);
// создаём картинки
dst = cvCreateImage(cvSize(image->width + 20, image->height + 20), image->depth,
image->nChannels);
dst2 = cvCreateImage(cvSize(image->width + 20, image->height + 20), image->depth,
image->nChannels);

printf("[i] image: %s\n", filename);
assert(image != 0);

// окно для отображения картинки
cvNamedWindow("original", CV_WINDOW_AUTOSIZE);
cvNamedWindow("IPL_BORDER_CONSTANT", CV_WINDOW_AUTOSIZE);
cvNamedWindow("IPL_BORDER_REPLICATE", CV_WINDOW_AUTOSIZE);

// обрамляем границей
cvCopyMakeBorder(image, dst, cvPoint(10, 10), IPL_BORDER_CONSTANT, cvScalar(250));
cvCopyMakeBorder(image, dst2, cvPoint(10, 10), IPL_BORDER_WRAP, cvScalar(250));

// показываем картинку
cvShowImage("original", image);
cvShowImage("IPL_BORDER_CONSTANT", dst);
cvShowImage("IPL_BORDER_REPLICATE", dst2);

// ждём нажатия клавиши
cvWaitKey(0);

// освобождаем ресурсы
cvReleaseImage(&image);
cvReleaseImage(&dst);
cvReleaseImage(&dst2);
// удаляем окна
cvDestroyAllWindows();
return 0;
}

```

Лабораторная работа №5

"Фильтрация в частотной области, поиск объекта путем расчета корреляционной функции"

Задание на работу:

1. Изучить методы расчета корреляционной функции
2. Изучить методику фильтрации изображения в частотной области на примерах фильтра Гаусса и фильтра Баттеворта
3. Написать программу, которая будет находить шаблонное изображение на исходном изображении путем расчета корреляционной функции
4. Реализовать фильтры Баттеворта и Гаусса для частотной области
5. Написать программу, которая будет фильтровать исходное изображение при помощи этих фильтров
6. Определить время работы программы вычисления корреляционных функций при разном разрешении фильтра

Изображения для поиска по шаблону:

cars.png – исходное изображение
red_car.png, white_car.png, yellow_car.png, square.png,
bush.png, bush2.png – шаблоны для поиска

Изображение для фильтрации:

mukik.png

Поиск объекта на изображении методом поиска по шаблону

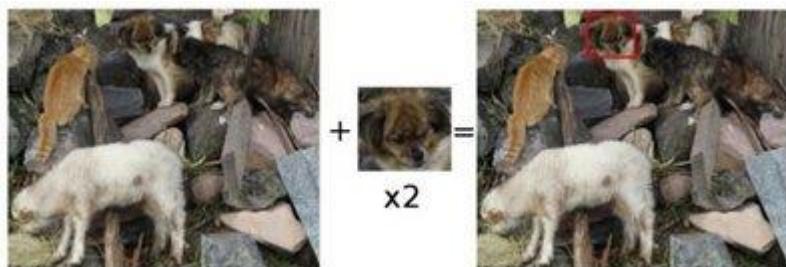
Поиск по шаблону — это метод поиска областей изображения, которые соответствуют (похожи) на изображение шаблона

Как это работает?

Нам нужно 2 простых составляющих:

1. Исходное изображение (**I**) на котором мы будем производить поиск
2. Шаблонное изображение (**T**), которое мы будем искать

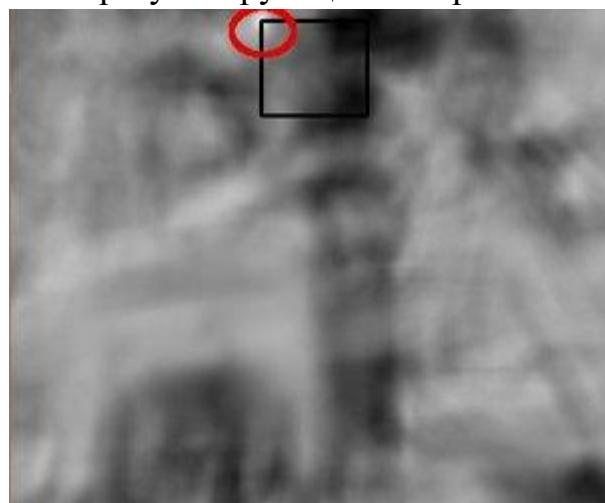
Наша цель найти наиболее совпадающую область.



Чтобы найти совпадающую область, мы должны последовательно сравнивать изображение шаблона с частью исходного изображения, перемещая изображение шаблона на 1 пиксель за раз (либо влево, либо вниз).



Для каждой точки мы будем вычислять корреляцию по определенной формуле и записывать в результирующее изображение



Затем, в зависимости от формулы расчета мы должны искать либо самую яркую точку, либо самую темную

Примеры Формул:

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

Где x' изменяется от 0 до ширины T , а y' от 0 до высоты T

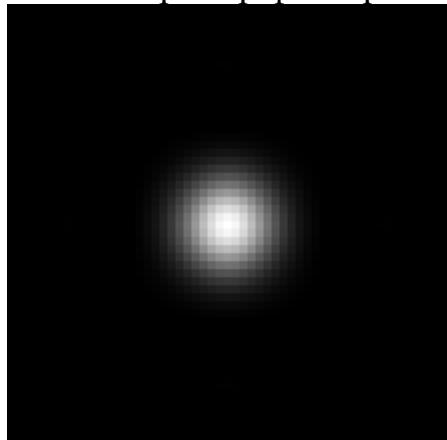
Частотная фильтрация

Множество подходов к улучшению изображений распадается на две большие категории: методы обработки в пространственной области (пространственные методы) и методы обработки в частотной области (частотные методы). Термин пространственная область относится к плоскости изображения как таковой, и данная категория объединяет подходы, основанные на прямом манипулировании пикселями изображениями. Методы обработки в частотной области основываются на модификации сигнала, формируемого путем применения к изображению преобразования Фурье. Минусом пространственной фильтрации часто является размытие контуров, снижение резкости. Частотная же фильтрация, в случае наличия помех, имеющих периодическую составляющую, позволяет оказывать более сконцентрированное влияние, по минимуму затрагивая полезную информацию.

Фильтр Гаусса — электронный фильтр, чьей импульсной переходной функцией является функция Гаусса. Фильтр Гаусса спроектирован таким образом, чтобы не иметь перерегулирования в переходной функции и максимизировать постоянную времени. Такое поведение тесно связано с

тем, что фильтр Гаусса имеет минимально возможную групповую задержку.

Пример фильтра

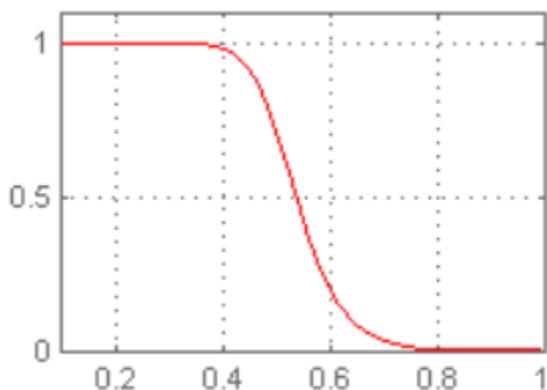


Фильтр Гаусса (Gaussian filter) обычно используется в цифровом виде для обработки двумерных сигналов (изображений) с целью снижения уровня шума. Однако при ресемплинге он дает сильное размытие изображения. Кроме того, этот фильтр используется для получения гауссовой модуляции. Этот вид модуляции применяется в системе сотовой связи GSM.

Фильтр Баттерворт — один из типов электронных фильтров. Фильтры этого класса отличаются от других методом проектирования. Фильтр Баттерворт проектируется так, чтобы его амплитудно-частотная характеристика была максимально гладкой на частотах полосы пропускания.

$$G^2(\omega) = |H(j\omega)|^2 = \frac{G_0^2}{1 + \left(\frac{\omega}{\omega_c}\right)^{2n}}$$

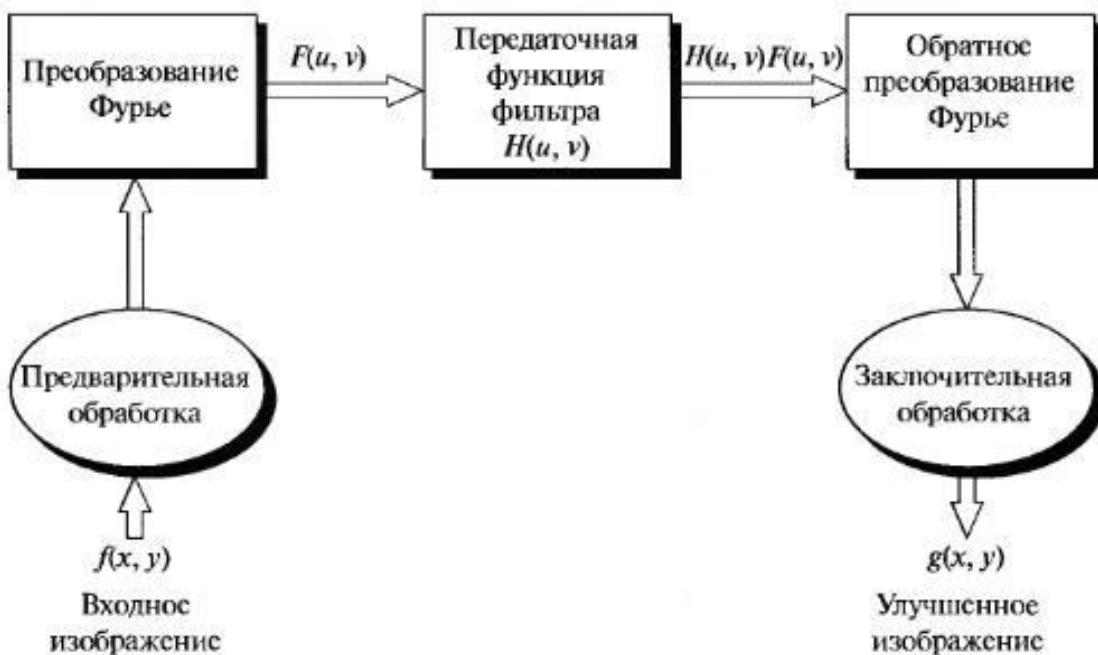
Фильтр Баттерворт



Где n – порядок фильтра; ω_c – частота среза (частота на которой амплитуда равна -3dB); G_0 - коэффициент усиления по постоянной составляющей (усиление на нулевой частоте)

Алгоритм частотной фильтрации

1. Вычисляем Фурье-образ фильтруемого изображения
 2. Умножаем Фурье-образ на частотную функцию Фильтра
 3. Делаем обратное преобразование Фурье
- 2.



Список функций OpenCV, которые могут использоваться:

<code>void integral(InputArray src, OutputArray sum, int depth=-1)</code>	Что делает: Вычисляет сумму пикселей изображения src – Входное изображение sum – Выходное изображение, каждый пиксель в позиции (X, Y) хранит в себе сумму пикселей входного изображения в позициях меньше (X, Y) depth – глубина цвета
<code>void imshow(const string& winname, InputArray mat)</code>	Показывает окно с изображением winname – Имя окна. image – изображение для показа.
<code>int waitKey(int delay=0)</code>	Ожидает нажатия клавиши, возвращает код нажатой клавиши delay – время ожидания, если = 0, то ожидает нажатия бесконечно
<code>void Mat::copyTo(OutputArray m)</code>	Копирует исходное изображение в другое m – выходное изображение
<code>void Mat::create(int</code>	Создает изображение определенного размера

rows, int cols, int type)	rows – количество строк cols – количество столбцов type – формат изображения
Mat Mat::ones(int rows, int cols int type)	создает матрицу изображения с единичными значениями пикселей (белую) rows – количество строк cols – количество столбцов type – формат изображения
Mat Mat::zeros(int rows, int cols int type)	создает матрицу изображения с нулевыми значениями пикселей (белую) rows – количество строк cols – количество столбцов type – формат изображения
Void magnitude(InputArray x, InputArray y, OutputArray magnitude)	Расчитывает магнитуду двухмерного вектора $dst(I) = \sqrt{x(I)^2 + y(I)^2}$ x – массив x координат вектора y – массив y координат вектора Magnitude – выходной массив
Void normalize(InputArray src, InputOutputArray dst, int norm_type)	Нормализует входной массив Src – входной массив Dst – выходной массив norm_type – тип нормализации

Табл. 1

Пример программы

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>

using namespace std;
using namespace cv;

const char* image_window = "Source Image";
const char* result_window = "Result window";
```

```

const char* template_window = "Template window";
const int match_method = CV_TM_SQDIFF_NORMED;

void selectTemplate(const char* tempName) {
    Mat templ;
    Mat imageForSeacrh;
    Mat result;

    cout << tempName << endl;

    templ = imread(tempName, IMREAD_COLOR);
    imageForSeacrh = imread("cars.png", IMREAD_COLOR);
    if (imageForSeacrh.empty() || templ.empty()) {
        cout << "Can't read one of the images" << endl;
        exit(1);
    }

    Mat img_display;
    imageForSeacrh.copyTo(img_display);
    int result_cols = imageForSeacrh.cols - templ.cols + 1;
    int result_rows = imageForSeacrh.rows - templ.rows + 1;
    result.create(result_rows, result_cols, CV_32F);

    *****
    * Код поиска
    *****

    imshow(result_window, result);
    imshow(image_window, img_display);
    imshow(template_window, templ);
    waitKey(0);
}

void calcDif(const char* tempName) {
    Mat templ;
    Mat imageForSeacrh;
    Mat result;

    cout << tempName << endl;

    templ = imread(tempName, IMREAD_COLOR);
    imageForSeacrh = imread("cars.png", IMREAD_COLOR);

    if (imageForSeacrh.empty() || templ.empty()) {
        cout << "Can't read one of the images" << endl;
        exit(1);
    }

    int result_cols = imageForSeacrh.cols - templ.cols + 1;
    int result_rows = imageForSeacrh.rows - templ.rows + 1;
    result.create(result_rows, result_cols, CV_32F);

    *****
    * Код поиска
    *****

    imshow(image_window, imageForSeacrh);
    imshow(result_window, result);
    imshow(template_window, templ);
}

```

```

    waitKey(0);

}

void do_correlation_searching() {
    namedWindow(image_window, WINDOW_NORMAL);
    namedWindow(result_window, WINDOW_AUTOSIZE);
    namedWindow(template_window, WINDOW_NORMAL);

    //Поиск изображения
    selectTemplate("red_car.png");
    calcDif("red_car.png");

    selectTemplate("white_car.png");
    calcDif("white_car.png");

    selectTemplate("yellow_car.png");
    calcDif("yellow_car.png");

    selectTemplate("square.png");
    calcDif("square.png");

    selectTemplate("bush.png");
    calcDif("bush.png");

    selectTemplate("bush2.png");
    calcDif("bush2.png");
}

void updateMag(Mat complex);
void updateResult(Mat complex);

Mat computeDFT(Mat image);
void shift(Mat magI);

Mat mask;

Mat create_butterworth_filter(int D, int n) {
    Mat filter = Mat::zeros(Size(256, 270), CV_32F);
    Mat tmp = Mat(filter.rows, filter.cols, CV_32F);

    ****
    * Код расчета фильтра
    ****

    filter = tmp;
    return filter;
}

void butterworth_filter() {
    cout << "butterwoth filter\n";
    String file;
    file = "mukik.png";

    Mat image = imread(file, CV_LOAD_IMAGE_GRAYSCALE);
    namedWindow("Orginal window", CV_WINDOW_AUTOSIZE);
    imshow("Orginal window", image);
    namedWindow("spectrum", CV_WINDOW_AUTOSIZE);
}

```

```

    Mat complex = computeDFT(image);
    mask = create_butterworth_filter(150, 2);
    shift(mask);
    Mat work;
    normalize(mask, work, 0, 1, NORM_MINMAX);
    imshow("mask", work);
    Mat planes[] = { Mat::zeros(complex.size(), CV_32F), Mat::zeros(complex.size(), CV_32F) };
    Mat kernel_spec;
    planes[0] = mask;
    planes[1] = mask;
    merge(planes, 2, kernel_spec);

    mulSpectrums(complex, kernel_spec, complex, DFT_ROWS);
    updateMag(complex);
    updateResult(complex);

}

void gaus_filter() {
    cout << "gausian filter\n";
    String file;
    file = "mukik.png";

    Mat image = imread(file, CV_LOAD_IMAGE_GRAYSCALE);
    namedWindow("Orginal window", CV_WINDOW_AUTOSIZE);
    imshow("Orginal window", image);
    namedWindow("spectrum", CV_WINDOW_AUTOSIZE);
    Mat gaussianKernel = Mat::ones(Size(256, 270), CV_32F); // Здесь должно быть ядро
    Гаянса
    mask = gaussianKernel;
    Mat complex = computeDFT(image);
    shift(mask);
    Mat work;
    normalize(mask, work, 0, 1, NORM_MINMAX);
    imshow("mask", work);
    Mat planes[] = { Mat::zeros(complex.size(), CV_32F), Mat::zeros(complex.size(), CV_32F) };
    Mat kernel_spec;
    planes[0] = mask; // real
    planes[1] = mask; // imaginair
    merge(planes, 2, kernel_spec);

    mulSpectrums(complex, kernel_spec, complex, DFT_ROWS); // only DFT_ROWS accepted
    updateMag(complex); // show spectrum
    updateResult(complex); // do inverse transform
}

int main(int argc, char** argv) {
    gaus_filter();
    waitKey(0);

    butterworth_filter();
    waitKey(0);

    do_correlation_searching();
    waitKey(0);

    return 0;
}

```

```

}

void updateResult(Mat complex)
{
    Mat work;
    idft(complex, work);
    Mat planes[] = { Mat::zeros(complex.size(), CV_32F), Mat::zeros(complex.size(),
CV_32F) };
    split(work, planes);

    magnitude(planes[0], planes[1], work);
    normalize(work, work, 0, 1, NORM_MINMAX);
    imshow("result", work);
}

void updateMag(Mat complex)
{

    Mat magI;
    Mat planes[] = { Mat::zeros(complex.size(), CV_32F), Mat::zeros(complex.size(),
CV_32F) };
    split(complex, planes);

    magnitude(planes[0], planes[1], magI);

    magI += Scalar::all(1);
    log(magI, magI);

    shift(magI);
    normalize(magI, magI, 1, 0, NORM_INF);
    imshow("spectrum", magI);
}

// применяет преобразование фурье к изображению
Mat computeDFT(Mat image) {
    Mat padded;
    int m = getOptimalDFTSize(image.rows);
    int n = getOptimalDFTSize(image.cols);
    copyMakeBorder(image, padded, 0, m - image.rows, 0, n - image.cols,
BORDER_CONSTANT, Scalar::all(0));
    Mat planes[] = { Mat<float>(padded), Mat::zeros(padded.size(), CV_32F) };
    Mat complex;
    merge(planes, 2, complex);
    dft(complex, complex, DFT_COMPLEX_OUTPUT);
    return complex;
}

void shift(Mat magI) {

    magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

    int cx = magI.cols / 2;
    int cy = magI.rows / 2;

    Mat q0(magI, Rect(0, 0, cx, cy));
    Mat q1(magI, Rect(cx, 0, cx, cy));
    Mat q2(magI, Rect(0, cy, cx, cy));
    Mat q3(magI, Rect(cx, cy, cx, cy));

    Mat tmp;

```

```
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
}
```

Автор: Фигловский Константин
почта: derpomorj@gmail.com

Лабораторная работа №6 "Морфологический водораздел (watershed) "

В данной работе требуется сегментировать изображения. В качестве примера используется метод водоразделов, поскольку он позволяет продемонстрировать автоматическую сегментацию изображения в реальном времени. Другие алгоритмы либо дорогостоящи для real-time обработки (grabCut), либо требуют вмешательства человека (color-based segmentation, visual background extractor). Алгоритмы вроде метода k-средних будут предложены в качестве заданий.

Алгоритм морфологического водораздела “watershed”

Метод водораздела, также называемый преобразованием водораздела, – это основанный на областях метод математической морфологии. В географии водораздел – это хребет, который делит области различных речных систем. Рассматривая изображения как геологический ландшафт, можно сказать, что линии водораздела – это границы, разделяющие участки изображений. В топографическом представлении изображения численные значения (например, уровни серого) каждого пикселя выступают в качестве высоты этой точки. Преобразование водораздела вычисляет водосборные бассейны и линии хребтов, при том, что водосборные бассейны - соответствующие области изображения, а линии хребтов – это границы этих областей.

Данный метод включает в себя следующие три базовые концепции:

1. Обнаружение и устранение разрывов;
2. Пороговая обработка;
3. Обработка областей.

Благодаря данным концепциям метод водоразделов позволяет получать более стабильные результаты сегментации (в том числе непрерывные границы областей). Этот подход так же позволяет включать в процесс сегментации добавочные ограничения.

На поверхности можно обнаружить три типа точек:

- точки локального минимума;
- точки, находящиеся на склоне, с которых вода сливаются к центру водоема;
- точки, находящиеся на гребне возвышенности.

Линии, образованные точками-гребнями, представляют собой линии водоразделов, поэтому основной задачей данного метода является именно поиск линий водоразделов. Алгоритм их поиска представляет собой следующий ход действий:

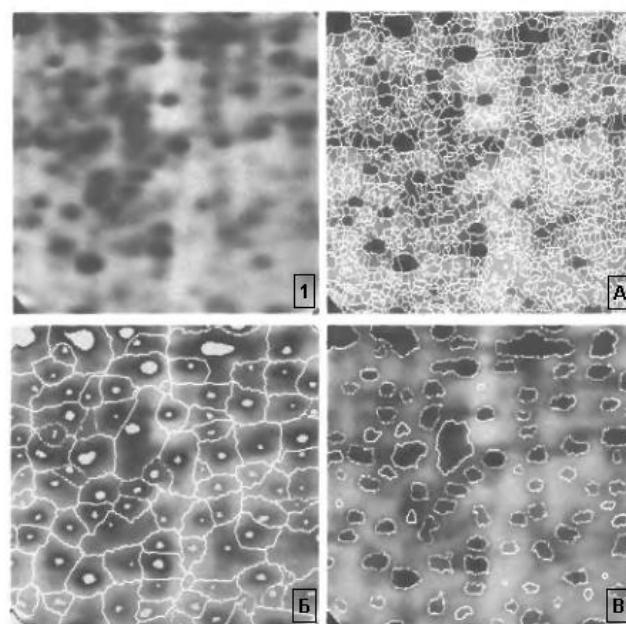
1. В местах локального минимума образуем «дырки», через которые вода начнет заполнять трехмерную поверхность.
2. Если вода с двух сторон гребня готова объединиться в один бассейн, устанавливаем перегородку.
3. Когда над водой останутся только загородки, останавливаем алгоритм.

Полученные таким образом перегородки и есть требуемые линии водоразделов. При ручном определении сегментируемых объектов используются маркеры. Маркер представляет собой связную компоненту, принадлежащую изображению. Будем различать внешние (соответствуют фону) и внутренние (относящиеся к объекту) маркеры. Процедура выбора маркера состоит из двух главных шагов:

1. предобработка;
2. выработка критериев, которым должны удовлетворять маркеры.

Пусть внутренний маркер определен как область со следующими критериями:

- окружена точками с большим значением яркости;
- ее точки образуют компоненту связности;
- все точки имеют одинаковое значение яркости.



Внутренние маркеры, найденные по этим 3 правилам, показаны на рис. (Б) светло-серым цветом. Далее применяем алгоритм сегментации по водоразделам с тем ограничением, что в качестве локальных минимумов рассматриваются только локальные маркеры. В итоге получим изображение, гораздо лучше читаемое по сравнению с исходным изображением (рис. (В)).

Приведенный случай – простейший. В общем случае маркеры могут иметь более сложное описание, включающее размеры, форму, местоположение, расстояния, текстурные и другие признаки. Самым большим достоинством маркеров является то, что можно использовать априорные знания о задаче и эффективно использовать их при решении.

Задания

1. Добавить реализацию метода watershed в авто режиме для камеры и статичного изображения;
2. Сегментировать изображение методом k-средних. Реализация метода уже имеется в openCV;
3. Улучшить сегментацию по водоразделам в автоматическом режиме, например, посредством удаления маленьких контуров.
4. Реализовать сегментацию видеопотока (камера или видеофайл) методом watershed в *ручном* режиме с использованием маркеров.
5. * Реализовать сегментацию видеопотока (камера или видеофайл) методом watershed в *автоматическом* режиме с использованием заранее подготовленных маркеров.
6. ** Показать работу color-based segmentation для статичного изображения.

Описание работы программы

Программа работает в двух 2 режимах: автоматическом и ручном. Авто режим позволяет без участия человека разбить изображение со статичной картинки и видеофайла. Реализация для видеокамеры и статичного изображения почти ничем не отличается от реализации для видеофайла, поэтому оставлена в качестве задания. Во всех вариантах используется следующий алгоритм:

1. Получить изображение;
2. Перевести в бинарный вид и получить массив контуров;
3. Создать маску с отрисованными контурами;
4. Применить watershed к этой маске, предварительно задав цвет;
5. Отрисовать итоговое изображение.

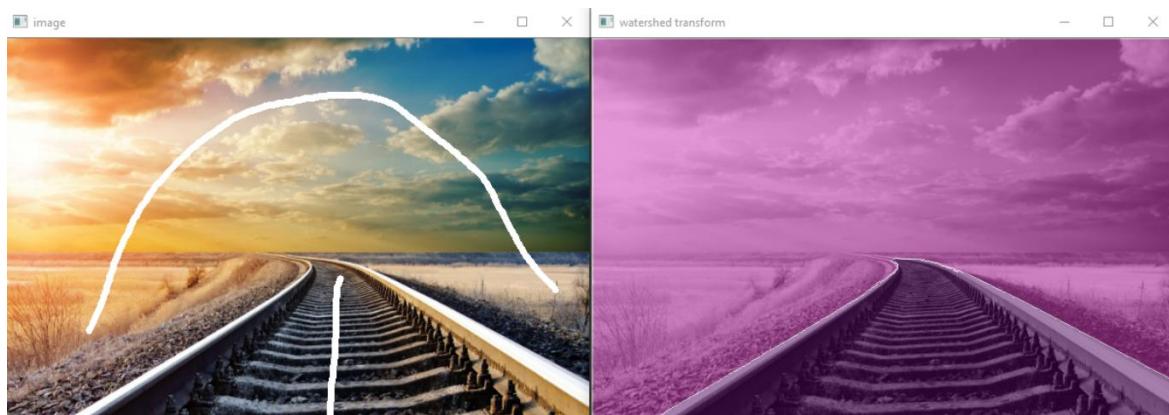
Посредством изменения параметра thresh в функции threshold можно убрать или добавить более мелкие отслеживаемые области. Для лучшего разбиения изображения желательно после определения контуров исключить те, длина которых слишком мала (<20 пикселей). Это также поможет исключить излишнюю сегментацию. Для улучшения работы алгоритма можно использовать метод нахождения локальных минимумов/максимумов.

В режиме ручного управления происходит несколько другой алгоритм подготовки изображения. Написана ручная маркировка для статичного изображения, реализация предварительной маркировки видеопотока оставлена в качестве дополнительного задания. Алгоритм работы:

1. Получить изображение;
2. Создать маску путем ручного выделения объектов;
3. Получить контуры с маски, а не с изображения;
4. Нанести на новое изображение найденные контуры и применить с ним watershed;
5. Отрисовать итоговое изображение.



1 watershed в автоматическом режиме



2 watershed в ручном режиме

Использованные функции

Функция	Описание
void cv::cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)	Преобразует изображение из одного цветового пространства в другое.
void cv::findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset = Point())	Находит контуры в двоичном изображении.
void cv::namedWindow (const String & winname, int flags = WINDOW_AUTOSIZE)	Создает окно.
void cv::imshow(const String & winname, InputArray mat)	Отображает изображение в указанном окне.
void cv::destroyWindow(const String & winname)	Уничтожает указанное окно.
int cv::waitKey(int delay =0)	Ожидает нажатия клавиши.
void drawContours(InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point())	Отрисовывает контуры из заполненного массива.
double cv::threshold (InputArray src, OutputArray dst, double thresh, double maxval, int type)	Применяет порог фиксированного уровня для каждого элемента массива.
void resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER_LINEAR)	Изменяет размеры изображения.

Применение метода водоразделов

```
#include <opencv2/core/utility.hpp>
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include <cstdio>
#include <iostream>
using namespace cv;
using namespace std;

Mat markerMask, img;
Point prevPt(-1, -1);

static void onMouse(int event, int x, int y, int flags, void*) {
    if (x < 0 || x >= img.cols || y < 0 || y >= img.rows) return;
    if (event == EVENT_LBUTTONDOWN || !(flags & EVENT_FLAG_LBUTTON)) prevPt = Point(-1, -1);
    else if (event == EVENT_LBUTTONDOWN) prevPt = Point(x, y);
    else if (event == EVENT_MOUSEMOVE && (flags & EVENT_FLAG_LBUTTON)) {
        Point pt(x, y);
        if (prevPt.x < 0) prevPt = pt;
        line(markerMask, prevPt, pt, Scalar::all(255), 5, 8, 0);
        line(img, prevPt, pt, Scalar::all(255), 5, 8, 0);
        prevPt = pt;
        cv::imshow("image", img);
    }
}
```

```

}

int main(int argc, char* argv[]) {
    cout << "METHODS OF USING WATERSHED SEGMENTATION\n";
    cout << "Enter mode: a --(auto) | h --(hand) \n";
    char s;
    cin >> s;

    if (s == 'a') {
        cout << "Enter type: p --(picture) | v --(video) | c --(camera) \n";
        cin >> s;
        if (s == 'p') {
        }
        else if (s == 'v') {
            cout << "Name of video: \n";
            string filename;
            cin >> filename;
            VideoCapture cap("vids/" + filename);
            while (cap.isOpened()) {
                Mat image;
                cap >> image;
                resize(image, image, Size(700, 500));
                cv::imshow("image", image);

                Mat imageGray, imageBin;

                cv::cvtColor(image, imageGray, CV_BGR2GRAY);
                cv::threshold(imageGray, imageGray, 130, 250,
                THRESH_BINARY);
                int compCount = 0;
                vector<vector<Point> > contours;
                vector<Vec4i> hierarchy;

                findContours(imageGray, contours, hierarchy, RETR_CCOMP,
                CHAIN_APPROX_SIMPLE);
                if (contours.empty()) continue;

                Mat markers(imageGray.size(), CV_32S);
                markers = Scalar::all(0);
                int idx = 0;
                for (; idx >= 0; idx = hierarchy[idx][0], compCount++)
                    drawContours(markers, contours, idx,
                    Scalar::all(compCount + 1), -1, 8, hierarchy, INT_MAX);
                if (compCount == 0) continue;

                std::vector<Vec3b> colorTab(compCount);
                for (int i = 0; i < compCount; i++)
                    colorTab[i] = Vec3b(rand() & 255, rand() & 255,
                    rand() & 255);

                cv::watershed(image, markers);
                Mat wshed(markers.size(), CV_8UC3);
                for (int i = 0; i < markers.rows; i++)
                    for (int j = 0; j < markers.cols; j++)
                {
                    int index = markers.at<int>(i, j);
                    if (index == -1) wshed.at<Vec3b>(i, j) =
                    Vec3b(0, 0, 0);
                    else if (index == 0) wshed.at<Vec3b>(i, j) =
                    Vec3b(255, 255, 255);
                }
            }
        }
    }
}

```

```

        else wshed.at<Vec3b>(i, j) = colorTab[index -
1];
    }
    cv::imshow("watershed transform", wshed);
    waitKey(100);
}
return 0;
}
else if (s == 'c') {
}
else cout << "Incorrect parameters";
}
else if (s == 'h') {
    cout << "Enter type: p --(picture) | v --(video) | c --(cam)\n";
    cin >> s;
    cout.clear();
    cout << "\nHELP\n"
        << "--move pressed left mouse button to mark picture\n"
        << "--press R to reset the mask\n"
        << "--press Space to apply\n\n";

    if (s == 'p') {
        while (true) {
            cout << "Name of image: \n";
            string filename;
            cin >> filename;
            Mat img0 = imread("pics/" + filename, 1), imgGray;
            resize(img0, img0, Size(600, 400));
            cv::imshow("image", img0);

            img0.copyTo(img);
            cv::cvtColor(img, markerMask, COLOR_BGR2GRAY);
            cv::cvtColor(markerMask, imgGray, COLOR_GRAY2BGR);

            markerMask = Scalar::all(0);

            setMouseCallback("image", onMouse, 0);

            while (true) {
                char c = (char)waitKey(0);

                if (c == 'e')
                    return 0;

                if (c == 27)
                    break;

                if (c == 'r') {
                    markerMask = Scalar::all(0);
                    img0.copyTo(img);
                    cv::imshow("image", img);
                }

                if (c == ' ') {
                    int i, j, compCount = 0;
                    vector<vector<Point> > contours;
                    vector<Vec4i> hierarchy;

                    findContours(markerMask, contours, hierarchy,
RETR_CCOMP, CHAIN_APPROX_SIMPLE);

```

```

        if (contours.empty()) continue;

        Mat markers(markerMask.size(), CV_32S);
        markers = Scalar::all(0);
        int idx = 0;

        for (; idx >= 0; idx = hierarchy[idx][0],
compCount++)
            drawContours(markers, contours, idx,
Scalar::all(compCount + 1), -1, 8, hierarchy, INT_MAX);
            if (compCount == 0) continue;

        vector<Vec3b> colorTab;
        for (i = 0; i < compCount; i++)
        {
            int b = theRNG().uniform(0, 255);
            int g = theRNG().uniform(0, 255);
            int r = theRNG().uniform(0, 255);
            colorTab.push_back(Vec3b((uchar)b,
(uchar)g, (uchar)r));
        }

        double t = (double)getTickCount();
        cv::watershed(img0, markers);
        t = (double)getTickCount() - t;
        cout << "execution time = " << t * 1000. /
getTickFrequency() << "\n";

        Mat wshed(markers.size(), CV_8UC3);

        for (i = 0; i < markers.rows; i++)
            for (j = 0; j < markers.cols; j++) {
                int index = markers.at<int>(i,
j);
                if (index == -1)
                    wshed.at<Vec3b>(i, j) =
                else if (index <= 0 || index >
compCount)
                    wshed.at<Vec3b>(i, j) =
                else
                    wshed.at<Vec3b>(i, j) =
colorTab[index - 1];
            }
            wshed = wshed * 0.5 + imgGray * 0.5;
            cv::imshow("watershed transform", wshed);
        }
    }
    return 0;
}
else if (s == 'v') {
}
else if (s == 'c') {
}
else cout << "Incorrect parameters";
}

«Назаренко Денис, гр. 23534/4. 2018 год»

```

Лабораторная работа №7

"Подмена пикселей"

Задание на выполнение работы

1. Напишите программу, которая захватывает видео с камеры и заменяет определённую часть изображения заданной картинкой, используя примеры 1 «Захват и отображение видео» и 2 «Замена пикселей». В качестве заданной картинки может использоваться статическая картинка или видеопоток (например, из заданного файла). Для получения пути файла с диска использовать пример 3 «Файловый диалог».
2. Объединить программу из пункта 1 с предварительным определением граничных точек изображения, подлежащих замене, с целью изменения размера вставляемого изображения. Для изменения размера изображения использовать функцию библиотеки OpenCV cv::resize().
3. По желанию. Реализуйте замену не на основе вхождения единственного пикселя в означеный регион, а на основе оценки «окна пикселей», к примеру, 3x3. Критерий замены разработать самостоятельно.

Захват и отображение видео

OpenCV, а точнее модуль HighGUI, предоставляет нам инструменты для захватывать видео покадрово, например, с видеокамеры или из файла. Способ аналогичен в обоих случаях, только при передаче в конструктор можно выбрать путь к файлу или передать идентификатор устройства. Смотрим на пример:

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
int main()
{
    cv::VideoCapture cameraVideoStream(0); // ID камеры 0
    if(!cameraVideoStream.isOpened()) // Выход если нет камеры
        return 1;
    cv::Mat cameraFrame; // Кадр
    cv::namedWindow ("camera", cv::WINDOW_NORMAL); // Окошко
    while(true)
```

```

{
    cameraVideoStream.read(cameraFrame); // Получаем кадр, так
    же как и из видео файла
    cv::imshow("camera", cameraFrame);
    char c = cvWaitKey(33);
    if (c == 27) break; // Если Esc - выходим
}
cameraFrame.release();
return 0;
}

```

Замена пикселей

Данная функция принимает вставляемое изображение source, изменяемое изображение dest, 2 ссылки на массив 3 элементов для верхней и нижней границы заменяемых цветов. Производится замена всех выбранных цветов на всём изображении соответствующими пикселями из данного изображения.

```

void replacePixels(const cv::Mat& source, cv::Mat& dest, const
uint8_t(&lowerBound)[3], const uint8_t(&upperBound)[3]) {

    uint8_t* pixelPtr = dest.data; // Получить массив пикселей из
cv::Mat
    int destCn = dest.channels();
    int sourceCn = source.channels();

    for (int i = 0; i < dest.rows; i++) {
        for (int j = 0; j < dest.cols; j++) {
            // Получение отдельной составляющей
            uint8_t b = pixelPtr[i*dest.cols*destCn + j * destCn + 0];
            uint8_t g = pixelPtr[i*dest.cols*destCn + j * destCn + 1];
            uint8_t r = pixelPtr[i*dest.cols*destCn + j * destCn + 2];

            if (b >= lowerBound[0] && b <= upperBound[0]
                && g >= lowerBound[1] && g <= upperBound[1]
                && r >= lowerBound[2] && r <= upperBound[2]) {
                // Сама замена пикселей
                for (int k = 0; k < 3; k++) {
                    pixelPtr[i*dest.cols*destCn + j * destCn + k] =
source.data[i*source.cols*sourceCn + j * sourceCn + k];
                }
            }
        }
    }
}

```

Файловый диалог

В данном примере используются возможности операционной системы Windows для графического представления работы с файловой системой, потому требуется подключить заголовочный файл Windows.h. Вся работа выполняется функцией GetOpenFileName(), которая принимает в качестве параметра ссылку на структуру OPENFILENAME.

```
std::string getSourcePath() {
    // OPENFILENAME structure initialization.
    OPENFILENAME file;

    char szFile[200];
    ZeroMemory(&file, sizeof(file));
    file.lStructSize = sizeof(file);
    file.hwndOwner = NULL;
    file.lpstrFile = szFile;
    file.lpstrFile[0] = '\0';
    file.nMaxFile = sizeof(szFile);
    // Filters for file dialog
    file.lpstrFilter =
"All\0*.*\0Image\0*.jpg;*.png;*.tiff;*.bmp\0Video\0*.avi;*.mp4\0";
    file.nFilterIndex = 1;
    file.lpstrFileTitle = NULL;
    file.nMaxFileTitle = 0;
    file.lpstrInitialDir = NULL;
    file.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;

    // Shows «Open File» window and fills file.lpstrFile field
    GetOpenFileName(&file);
    /* if file.lpstrFile requires w_char
     * Convert file.lpstrFile to std::string with this code
     *
     * std::wstring wide(file.lpstrFile);
     * std::string str(wide.begin(), wide.end());
     */
    std::string str(file.lpstrFile);
    return str;
}
```

Пример программы

В данной программе собраны примеры из предыдущих пунктов и реализован графический интерфейс для управления предоставленными возможностями. Графический интерфейс реализован с помощью библиотеки cvui, узнать подробнее о которой и посмотреть примеры

МОЖНО ПО ССЫЛКЕ <https://github.com/Dovyski/cvui>.

```
#include "Windows.h"
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

// For cvui
#define CVUI_IMPLEMENTATION
#include "cvui.h"

#define MODIFIED_NAME "Modified image"
#define SETTINGS_NAME "Settings"

void replacePixels(const cv::Mat& source, cv::Mat& dest, const
uint8_t(&lowerBound)[3], const uint8_t(&upperBound)[3];

std::string getSourcePath();

int main()
{
    cv::VideoCapture cameraVideoStream(0); // Camera ID 0
    if (!cameraVideoStream.isOpened()) // Exit if there is no
device with this id
        return 1;
    cv::Mat cameraFrame; // Frame
    cv::Mat replacer;

    /*****Image window init*****/
    cvui::init(MODIFIED_NAME);
    /*****Settings window init*****/

    /*****Pixel Replacing Settings
init*****/
    bool useReplace = false;
    uint8_t lowerBound[3]{ 0,0,0 };
    uint8_t upperBound[3]{ 255,255,255 };
    /******/

    while (true)
```

```

{
    cameraFrame.release();
    cameraVideoStream.read(cameraFrame);

    /*****Settings window draw*****/
    cvui::window(settingsFrame, 0, 0, 200, 475, "Pixel
Replacing");
    // Checkbar for pixel replacing
    cvui::checkbox(settingsFrame, 0, 25, "Use Replace",
&useReplace);
    if (!replacer.data) {
        cvui::text(settingsFrame, 100, 27, "Needs picture", 0.4,
0xFFCEFF);
        useReplace = false;
    }

    // Button for open file dialog
    if (cvui::button(settingsFrame, 30, 50, "Choose source")) {
        std::string fileName = getSourcePath();
        // This realization works only with pictures, not video
        replacer = cv::imread(fileName);
        if (!replacer.data) {
            std::cerr << "Can't open source file";
            exit(1);
        }
        resize(replacer, replacer, cv::Size(cameraFrame.cols,
cameraFrame.rows), 0, 0, cv::INTER_CUBIC);
    }

    // Trackbar start position
    int trackBarsY = 95;
    cvui::window(settingsFrame, 0, trackBarsY, 200, 190, "Lower
bound");
    cvui::text(settingsFrame, 10, trackBarsY + 50, "b");
    cvui::trackbar<uint8_t>(settingsFrame, 20, trackBarsY + 35, 165,
&lowerBound[0], 0, 255);
    cvui::text(settingsFrame, 10, trackBarsY + 100, "g");
    cvui::trackbar<uint8_t>(settingsFrame, 20, trackBarsY + 85, 165,
&lowerBound[1], 0, 255);
    cvui::text(settingsFrame, 10, trackBarsY + 150, "r");
    cvui::trackbar<uint8_t>(settingsFrame, 20, trackBarsY + 135,
165, &lowerBound[2], 0, 255);

    cvui::window(settingsFrame, 0, trackBarsY + 190, 200, 190,
"Upper bound");
    cvui::text(settingsFrame, 10, trackBarsY + 240, "b");
    cvui::trackbar<uint8_t>(settingsFrame, 20, trackBarsY + 225,
165, &upperBound[0], 0, 255);
}

```

```

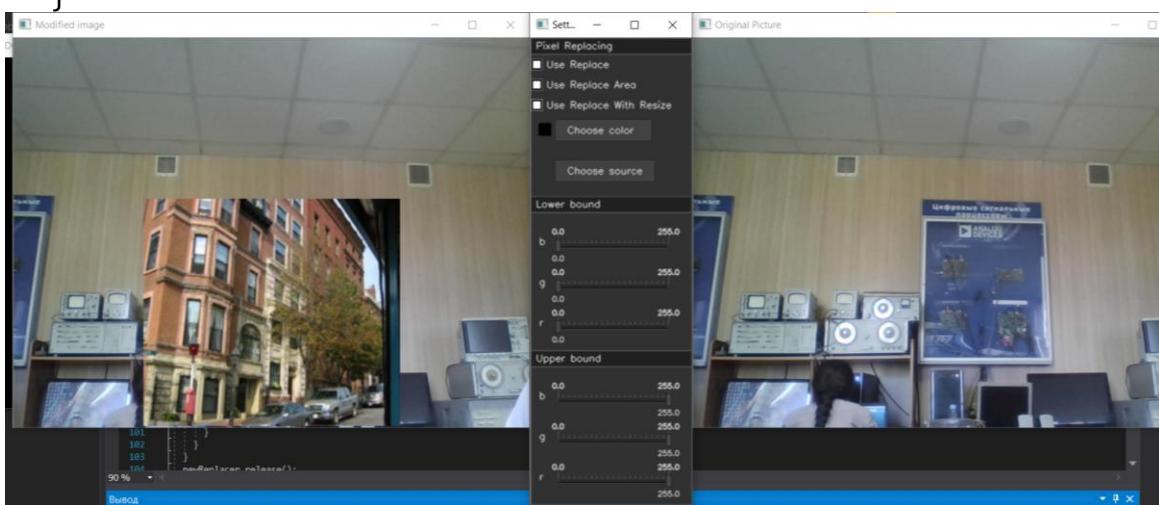
        cvui::text(settingsFrame, 10, trackBarsY + 290, "g");
        cvui::trackbar<uint8_t>(settingsFrame, 20, trackBarsY + 275,
165, &upperBound[1], 0, 255);
        cvui::text(settingsFrame, 10, trackBarsY + 340, "r");
        cvui::trackbar<uint8_t>(settingsFrame, 20, trackBarsY + 325,
165, &upperBound[2], 0, 255);

        cvui::update();
        cv::imshow(SETTINGS_NAME, settingsFrame);
        /*************************************************************************/
}

if (useReplace) {
    replacePixels(replacer, cameraFrame, lowerBound,
upperBound);
}
cv::imshow(MODIFIED_NAME, cameraFrame);

char c = cvWaitKey(33);
if (c == 27) break; // Exit if pressed Esc
}
cameraFrame.release();
replacer.release();
return 0;
}

```



Загрузка и установка

Последнюю версию OpenCV можно загрузить с Sourceforge.net. На момент написания инструкции наиболее новой версией библиотеки являлась 3.4.1. После того как вы скачаете библиотеку, вы должны её установить. Скачайте установочный файл и запустите его. Программа установки установит OpenCV, зарегистрирует фильтр DirectShow, и выполнит все необходимые процедуры. Открывайте VC++, создайте пустой проект. Далее требуется сконфигурировать проект в соответствии с

инструкцией:

- Сборка->Диспетчер конфигураций:
x64 в «Платформа» и «Активная платформа решения».

Конфигурация: Release

- Отладка->Свойства
 - Каталоги VC++: «Включаемые каталоги»: добавить
`<OPENCV_PATH>\opencv\build\include`
«Каталоги библиотек»: добавить
`<OPENCV_PATH>\opencv\build\x64\vc15\lib`
 - Компоновщик->Все параметры: «Дополнительные зависимости» добавить `opencv_world341.lib`

Из `<OPENCV_PATH>\opencv\build\x64\vc15\bin` скопировать `opencv_ffmpeg341_64.dll` и `opencv_world341.dll` в папку к запускаемому приложению, например `<VC++PROJECT_PATH>\x64\Release`

Теперь Вы готовы к началу использования OpenCV.

Для организации графического интерфейса рекомендуется использовать Header-only библиотеку CVUI, найти которую можно по адресу <https://github.com/Dovyski/cvui>. Для её использования требуется скачать файл `cvui.h` по данной ссылке и расположить его рядом с исходными файлами лабораторной работы. Для работы данной библиотеки требуется подключить только `.h` файл и библиотеку openCV. Также в одном из файлов лабораторной работы требуется объявить `CVUI_IMPLEMENTATION`:

```
#define CVUI_IMPLEMENTATION
```

Лабораторная работа №8

"Улучшение изображения и сегментация"

Задание

1. Изучить методы бинаризации изображения (Бинаризации с верхним порогом, бинаризация с нижним порогом и т.д.).
2. Изучить механизмы изменения яркости и контрастности изображения.
3. Загрузить шаблон из папки ./program/ и прописать пути к библиотеке OpenCV в Visual Studio.
4. Добавить возможность изменения типа бинаризации (различные типы адаптивной (локальной) и глобальной бинаризации). Добавить возможность их изменения при помощи OpenCV Trackbar.
5. Добавить возможность изменения яркости и контрастности изображения при помощи функций OpenCV.

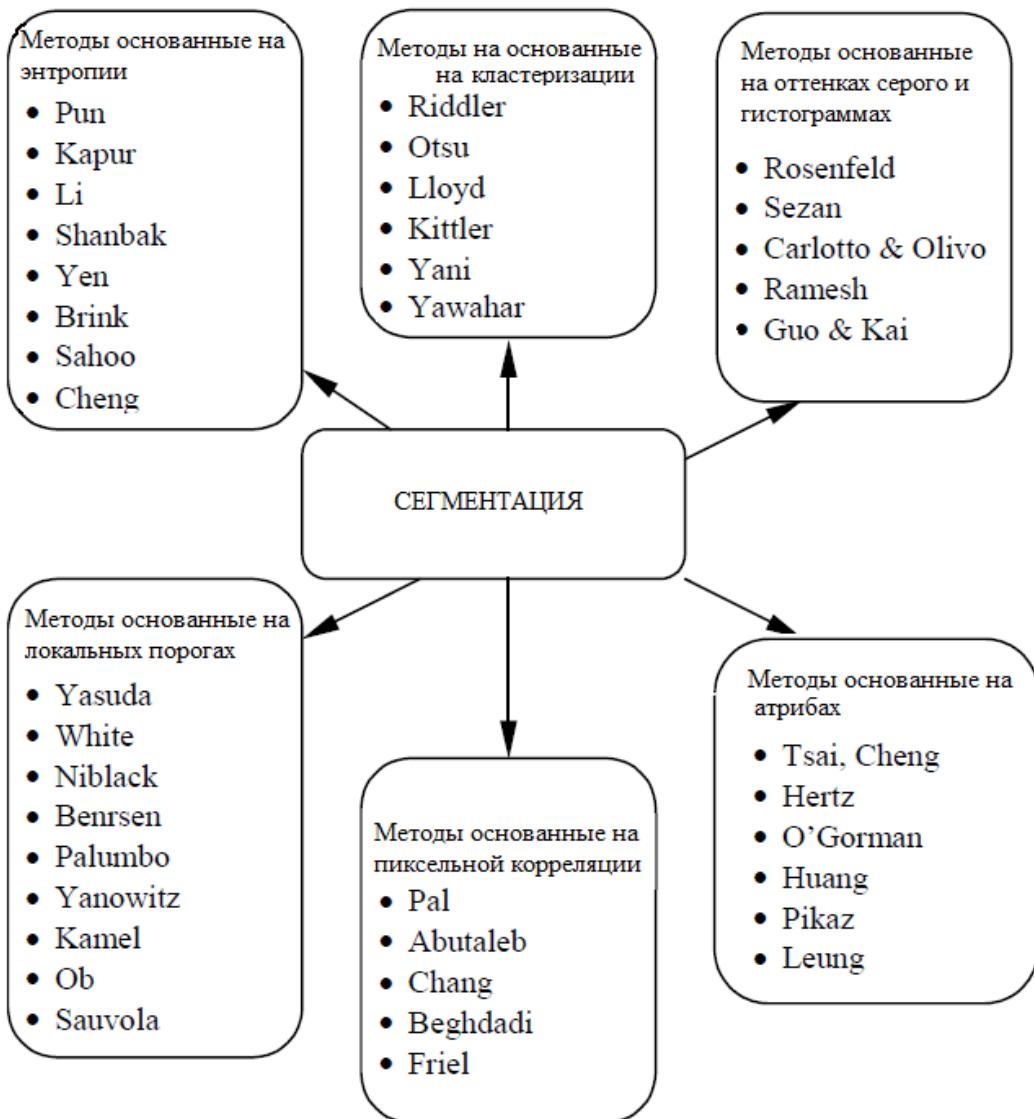
Сегментация изображения

Одной из основных задач обработки и анализа изображений является сегментация, т.е. разделение изображения на области, для которых выполняется определенный критерий однородности, например, выделение на изображении областей приблизительно одинаковой яркости. Понятие области изображения используется для определения связной группы элементов изображения, имеющих определенный общий признак (свойство).

Один из основных и простых способов — это построение сегментации с помощью порога. Порог — это признак (свойство), которое помогает разделить искомый сигнал на классы. Операция порогового разделения заключается в сопоставлении значения яркости каждого пикселя изображения с заданным значением порога.

Бинаризация

Операция порогового разделения, которая в результате дает бинарное изображение, называется бинаризацией. Целью операции бинаризации является радикальное уменьшение количества информации, содержащейся на изображении. В процессе бинаризации исходное полутоновое изображение, имеющее некое количество уровней яркости, преобразуется в черно-белое изображение, пиксели которого имеют только два значения – 0 и 1. Пороговая обработка изображения может проводиться разными способами.



Бинаризация с нижним порогом

Бинаризация с нижним порогом

Бинаризация с нижним порогом является наиболее простой операцией, в которой используется только одно значение порога:

$$f'(m, n) = \begin{cases} 0, & f(m, n) \geq t; \\ 1, & f(m, n) < t, \end{cases}$$

Все значения вместо критерия становятся 1, в данном случае 255 (белый) и все значения(амплитуды) пикселей, которые больше порога $t = 0$ (черный).

Бинаризации с верхним порогом

Иногда можно использовать вариант первого метода, который дает негатив изображения, полученного в процессе бинаризации. Операция бинаризации с верхним порогом:

$$f'(m, n) = \begin{cases} 0, & f(m, n) \leq t; \\ 1, & f(m, n) > t, \end{cases}$$

Бинаризация с двойным ограничением

Для выделения областей, в которых значения яркости пикселей может меняться в известном диапазоне, вводится бинаризация с двойным ограничением ($t_1 < t_2$):

$$f'(m, n) = \begin{cases} 0, & f(m, n) \geq t_1; \\ 1, & t_1 < f(m, n) \leq t_2; \\ 0, & f(m, n) > t_2, \end{cases}$$

Так же возможны другие вариации с порогами, где пропускается только часть данных (средне полосовой фильтр).

Неполная пороговая обработка

Данное преобразование дает изображение, которое может быть проще для дальнейшего анализа, поскольку оно становится лишенным фона со всеми деталями, присутствующими на исходном изображении.

$$f'(m, n) = \begin{cases} f(n, m), & f(m, n) > t; \\ 0, & f(m, n) \leq t, \end{cases}$$

Многоуровневое пороговое преобразование

Данная операция формирует изображение, не являющееся бинарным, но состоящее из сегментов с различной яркостью.

$$f'(m, n) = \begin{cases} 1, & f(m, n) \in D_1; \\ 2, & f(m, n) \in D_2; \\ \dots \\ n, & f(m, n) \in D_n; \\ 0, & \text{в остальных случаях,} \end{cases}$$

Что касается бинаризации, то по сути все. Хотя можно добавить, что есть глобальная, которая используется для всего изображения и так же существует локальная, которая захватывает часть картинки (изображения).

Локальная пороговая обработка

Метод Отса

Метод использует гистограмму распределения значений яркости пикселей растрового изображения. Строится гистограмма по значениям $p_i = n_i/N$, где N – это общее кол-во пикселей на изображении, n_i – это кол-во пикселей с уровнем яркости i . Диапазон яркостей делится на два класса с помощью порогового значения уровня яркости k , k — целое значение от 0 до L .

Каждому классу соответствуют относительные частоты ω_0 и ω_1 :

$$\begin{aligned}\omega_0(k) &= \sum_{i=1}^k p_i & \omega_1(k) &= \sum_{i=k+1}^L p_i = 1 - \omega_0(k) \\ \mu_0(k) &= \sum_{i=1}^k \frac{ip_i}{\omega_0} & \mu_1(k) &= \sum_{i=k+1}^L \frac{ip_i}{\omega_1}\end{aligned}$$

Средние уровни для каждого из двух классов изображения:

Далее вычисляется максимальное значение оценки качества разделения изображения на две части:

где $(\sigma_{\text{кл}})^2 = \omega_0 \omega_1 (\mu_1 - \mu_0)^2$, – межклассовая дисперсия, а $(\sigma_{\text{общ}})^2$ – это общая дисперсия для всего изображения целиком.

$$\eta(k) = \max_{k=1}^{L-1} \left(\frac{\sigma_{\text{кл}}^2(k)}{\sigma_{\text{общ}}^2} \right)$$

Глобальная пороговая обработка

Метод Бернсена

1. Обычная квадратная апертура с нечетным числом пикселей пробегает в цикле по всем пикселям исходного изображения. На каждом шаге находится Min и Max.
2. Находится среднее значение $\text{Avg} = (\text{Min} + \text{Max}) / 2$.
3. Если текущий пиксель больше $\text{Avg} < E$ — он становится белым, иначе — чёрным. E — некая константа заданная пользователем.
4. Если среднее меньше порога контраста — то текущий пиксель становится того цвета, который задавался параметром «цвет сомнительного пикселя».

Имеет ряд недостатков: после обработки монотонных областей яркости формируются сильные паразитные помехи, в некоторых случаях приводит к появлению ложных черных пятен

Методы пороговой обработки, используемые в работе

```
CVAPI(double) cvThreshold( const CvArr* src, CvArr* dst,
                           double threshold, double max_value,
                           int threshold_type );
```

— выполняет фиксированное пороговое преобразование для элементов массива.

src — исходный массив(изображение) (одноканальное, 8-битное или 32-битное)

dst — целевой массив, должен иметь тот же тип что и src или 8-битный

threshold — пороговая величина

max_value — максимальное значение (используется совместно с CV_THRESH_BINARY и CV_THRESH_BINARY_INV)

threshold_type — тип порогового преобразования:

```
#define CV_THRESH_BINARY      0 /* value = value > threshold ? max_value : 0      */
#define CV_THRESH_BINARY_INV   1 /* value = value > threshold ? 0 : max_value      */
#define CV_THRESH_TRUNC        2 /* value = value > threshold ? threshold : value */
#define CV_THRESH_TOZERO       3 /* value = value > threshold ? value : 0      */
#define CV_THRESH_TOZERO_INV   4 /* value = value > threshold ? 0 : value      */
#define CV_THRESH_MASK          7
#define CV_THRESH_OTSU         8 /* use Otsu algorithm to choose the optimal threshold value;
                                combine the flag with one of the above CV_THRESH_* values */
```

CV_THRESH_BINARY

$$dst(x, y) = \begin{cases} \text{max_value} & \text{if } \text{src}(x, y) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

CV_THRESH_BINARY_INV

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{threshold} \\ \text{max_value} & \text{otherwise} \end{cases}$$

CV_THRESH_TRUNC

$$dst(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{threshold} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$

CV_THRESH_TOZERO

$$dst(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

CV_THRESH_TOZERO_INV

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{threshold} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$

Обычное пороговое преобразование никак не учитывает, что части объектов могут иметь различную яркость из-за разности в освещённости. Это можно исправить если использовать адаптивное пороговое преобразование, которое рассматривает значение не в одном пикселе, а в окрестности пикселя. Это значение может быть просто средним значением пикселей окрестности(т.е. все пиксели равнозначны)(CV_ADAPTIVE_THRESH_MEAN_C), либо пиксели окрестности умножаются на весовой коэффициент (взвешиваются) в соответствии с функцией, например с гауссовой функцией (CV_ADAPTIVE_THRESH_GAUSSIAN_C).

```
CVAPI(void) cvAdaptiveThreshold( const CvArr* src, CvArr* dst, double max_value,
int adaptive_method CV_DEFAULT(CV_ADAPTIVE_THRESH_MEAN_C),
int threshold_type CV_DEFAULT(CV_THRESH_BINARY),
int block_size CV_DEFAULT(3),
double param1 CV_DEFAULT(5));
```

— выполняет адаптивное пороговое преобразование для элементов массива.

src — исходное изображение

dst — целевое изображение

max_value — максимальное значение (используется совместно с CV_THRESH_BINARY и CV_THRESH_BINARY_INV)

adaptive_method — используемый адаптационный алгоритм:

```
#define CV_ADAPTIVE_THRESH_MEAN_C 0
#define CV_ADAPTIVE_THRESH_GAUSSIAN_C 1
```

```
#define CV_THRESH_BINARY      0 /* value = value > threshold ? max_value : 0      */
#define CV_THRESH_BINARY_INV   1 /* value = value > threshold ? 0 : max_value      */
```

threshold_type — тип порогового преобразования:

block_size — размер окрестности (в пикселях), которая используется для расчёта порогового значения: 3, 5, 7 и т.д.

param1 — параметр, зависящий от используемого метода. Для методов CV_ADAPTIVE_THRESH_MEAN_C и CV_ADAPTIVE_THRESH_GAUSSIAN_C — это константа, вычитаемая из среднего или взвешенного значения (может быть отрицательной).

Нахождение и отрисовка контуров

Контур объекта — это его видимый край, который отделяет объект от фона. В действительности, большинство методов анализа изображений работают именно с контурами, а не с пикселями как таковыми.

Совокупность методов работы с контурами называется **контурным анализом**.

В OpenCV для поиска контуров имеется функцией **findContours**, которая имеет вид:

```
findContours(кадр, контуры, режим_группировки, метод_упаковки)
```

кадр — должным образом подготовленная для анализа картинка. Это должно быть 8-битное изображение. Поиск контуров использует для работы монохромное изображение, так что все пиксели картинки с ненулевым цветом будут интерпретироваться как 1, а все нулевые останутся нулями. На уроке про поиск цветных объектов была точно такая же ситуация.

режим_группировки — один из четырех режимов группировки найденных контуров:

- CV_RETR_LIST — выдаёт все контуры без группировки;
- CV_RETR_EXTERNAL — выдаёт только крайние внешние контуры. Например, если в кадре будет пончик, то функция вернет его внешнюю границу без дырки.
- CV_RETR_CCOMP — группирует контуры в двухуровневую иерархию. На верхнем уровне — внешние контуры объекта. На втором уровне — контуры отверстий, если таковые имеются. Все остальные контуры попадают на верхний уровень.
- CV_RETR_TREE — группирует контуры в многоуровневую иерархию.

метод_упаковки — один из трёх методов упаковки контуров:

- CV_CHAIN_APPROX_NONE — упаковка отсутствует и все контуры хранятся в виде отрезков, состоящих из двух пикселей.
- CV_CHAIN_APPROX_SIMPLE — склеивает все горизонтальные, вертикальные и диагональные контуры.
- CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS — применяет к контурам метод упаковки (аппроксимации) Teh-Chin.

контуры — список всех найденных контуров, представленных в виде векторов.

Полученные с помощью функции **findContours** контуры хорошо бы каким-то образом нарисовать в кадре. Машине это не нужно, зато нам это поможет лучше понять как выглядят найденные алгоритмом контуры. Поможет в этом ещё одна полезная функция — **drawContours**.

```
drawContours( кадр, контуры, индекс, цвет)
```

кадр — кадр, поверх которого мы будем отрисовывать контуры;
контуры — те самые контуры, найденные функцией `findContours`;
индекс — индекс контура, который следует отобразить. `-1` — если нужно отобразить все контуры;
цвет — цвет контура.

Изменение яркости и контраста

Два часто используемых процесса - это умножение и добавление константы:

$$g(x) = \alpha f(x) + \beta$$

Параметры $\alpha > 0$ и β часто называются параметрами усиления и смещения; иногда эти параметры, как говорят, контролируют контраст и яркость соответственно. Вы можете рассматривать $f(x)$ как пиксели исходного изображения и $g(x)$ как пиксели выходного изображения. Тогда, более удобно, мы можем записать выражение как:

$g(i, j) = \alpha \cdot f(i, j) + \beta$ где i и j индексы пикселей в изображении (*строка и столбец*). Для изменения яркости и контрастности изображения применим функцию:

```
void Mat::convertTo(OutputArray m, int rtype, double alpha=1,  
double beta=0 )
```

m — выходное изображение

rtype — желаемый тип выходной матрицы;

alpha — параметр усиления;

beta — параметр смещения.

Шаблон

```
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
#include <iostream>  
using namespace cv;  
using namespace std;  
int thresholdValue = 0;  
int thresholdType = 3;;
```

```

int const maxValue = 255;
int const maxType = 4;
int const maxBinaryValue = 255;

Mat src, srcGray;
Mat thresholdImage;
Mat adaptiveThresholdImage;

std::string adaptiveThresholdingWindow = "Adaptive thresholding";
std::string thresholdingWindow = "Thresholding";

void onThresholdChanged(int, void *) {

}

void showThresholding(const cv::Mat &image) {
    src = image.clone();
    cvtColor(src, srcGray, CV_BGR2GRAY);
    namedWindow(thresholdingWindow, CV_WINDOW_AUTOSIZE);
    createTrackbar("Threshold type", thresholdingWindow,
    &thresholdType, maxType, onThresholdChanged);
    createTrackbar("Threshold", thresholdingWindow, &thresholdValue,
    maxValue, onThresholdChanged);
    onThresholdChanged(0, nullptr);
}

int adaptiveMethod = 0;
const int maxAdaptiveMethod = 1;
int blockSize = 3;
int C = 1;

void onAdaptiveThresholdChanged(int, void *) {
    if (blockSize % 2 == 0) {
        blockSize++;
    }
    if (blockSize <= 3) {
        blockSize = 3;
    }
    // Добавить adaptive thresholding
}

void showAdaptiveThresholding(const cv::Mat &image) {
    src = image;
    if (src.rows > 1600 || src.cols > 2560) {
        resize(src, src, Size(src.cols/5, src.rows/5));
    }
    cvtColor(src, srcGray, CV_BGR2GRAY);
    namedWindow(adaptiveThresholdingWindow, CV_WINDOW_AUTOSIZE );

    createTrackbar("Adaptive method", adaptiveThresholdingWindow,
    &adaptiveMethod, maxAdaptiveMethod, onAdaptiveThresholdChanged);
    createTrackbar("Block size", adaptiveThresholdingWindow,
    &blockSize, maxValue, onAdaptiveThresholdChanged);
    createTrackbar("Const", adaptiveThresholdingWindow, &C, maxValue,
    onAdaptiveThresholdChanged);
}

```

```

        onAdaptiveThresholdChanged(0, nullptr);
    }

void drawCont(const Mat &image, const String windowToShow) {
    std::vector<std::vector<cv::Point>> contours;
    cv::Mat contourOutput = image.clone();
    cv::findContours(contourOutput, contours, CV_RETR_LIST,
CV_CHAIN_APPROX_NONE );
    // Отрисовка контуров
    cv::Mat contourImage(image.size(), CV_8UC3, cv::Scalar(0,0,0));
    cv::Scalar colors[3];
    colors[0] = cv::Scalar(255, 0, 0);
    colors[1] = cv::Scalar(0, 255, 0);
    colors[2] = cv::Scalar(0, 0, 255);
    for (size_t idx = 0; idx < contours.size(); idx++) {
        cv::drawContours(contourImage, contours, idx, colors[idx % 3]);
    }
    cv::imshow(windowToShow, contourImage);
    cv::moveWindow(windowToShow, 200, 0);
}

int alpha = 100;
int beta = 100;

// оригинал изображения и изображение с изменненной
гаммой/яркостью/контрастностью
Mat img_original, img_corrected;

// Измененение контраста и яркости
void basicLinearTransform(const Mat &img, const double alpha_, const
int beta_) {
    Mat res;
    // Изменить яркость и контрастность
}

// Обработка изменения контраста
void onContrastChangeListener(int, void *) {
    //basicLinearTransform(img_original, alpha_value, beta_value);
    imshow("Brightness/contrast/gamma adjustments", img_corrected);
}

// Обработка изменения яркости
void onBrightnessChangeListener(int, void *) {
    //basicLinearTransform(img_original, alpha_value, beta_value);
    imshow("Brightness/contrast/gamma adjustments", img_corrected);
}

int main( int argc, char** argv )
{
    if (argc != 2) {
        throw std::invalid_argument("Wrong number of parameters!");
    }

    img_original = imread(argv[1]);
}

```

```

if (!img_original.data) {
    throw std::invalid_argument("No image data found!");
}
img_corrected = img_original;

/* Создание окна изменения яркости/гаммы/контраста */
namedWindow("Brightness/contrast/gamma adjustments",
WINDOW_AUTOSIZE);
createTrackbar("Contrast", "Brightness/contrast/gamma
adjustments", &alpha, 500, onContrastChangeListener);
createTrackbar("Brightness", "Brightness/contrast/gamma
adjustments", &beta, 200, onBrightnessChangeListener);
onContrastChangeListener(0, 0);
onBrightnessChangeListener(0, 0);
cv::waitKey(0);
cvDestroyWindow("Brightness/contrast/gamma adjustments");

// Бинаризация изображения
showThresholding(img_corrected);
showAdaptiveThresholding(img_corrected);
cv::waitKey(0);
cvDestroyWindow(adaptiveThresholdingWindow.data());
cvDestroyWindow(thresholdingWindow.data());

// Отрисовка границ
drawCont(thresholdImage, "Contours from threshold image");
drawCont(adaptiveThresholdImage, "Contours from adaptive threshold
image");
cv::waitKey(0);

return 0;
}

```

Полянок Б.Д. (miner34006@gmail.com)

Лабораторная работа №9

"Сегментация на основе цвете "

В компьютерном зрении, сегментация — это процесс разделения цифрового изображения на несколько сегментов (множество пикселей, также называемых суперпикселями). Цель сегментации заключается в упрощении и/или изменении представления изображения, чтобы его было проще и легче анализировать. Результатом сегментации изображения является множество сегментов, которые вместе покрывают всё изображение, или множество контуров, выделенных из изображения. Все пиксели в сегменте похожи по некоторой характеристике или вычисенному свойству, например, по цвету, яркости или текстуре (в данной работе предлагается сделать сегментацию, основанную на цвете). Более точно, сегментация изображений — это процесс присвоения таких меток каждому пиксели изображения, что пиксели с одинаковыми метками имеют общие визуальные характеристики. Соседние сегменты значительно отличаются по этой характеристике.

Для сегментации изображений было разработано несколько универсальных алгоритмов и методов. Так как общего решения для задачи сегментации изображений не существует, часто эти методы приходится совмещать со знаниями из предметной области, чтобы эффективно решать эту задачу в её предметной области. Различные алгоритмы для проведения сегментации:

- Выделение краёв
- Методы разрастания областей
- Методы разреза графа
- Сегментация методом водораздела
- Сегментация с помощью модели
- Многомасштабная сегментация и другие

Выделение краев - хорошо изученная область в обработке изображений. Границы и края областей сильно связаны, так как часто существует сильный перепад яркости на границах областей. Поэтому методы выделения краёв используются как основа для другого метода сегментации. Обнаруженные края часто бывают разорванными. Но чтобы выделить объект на изображении, нужны замкнутые границы области.

Методы разреза графа могут быть эффективно применены для сегментации изображений. В этих методах изображение представляется как взвешенный неориентированный граф. Обычно пиксель или группа пикселей ассоциируется вершиной, а веса рёбер определяют (не)похожесть соседних пикселей. Затем граф (изображение) разрезается согласно критерию, созданному для получения «хороших» кластеров. Каждая часть вершин (пикселей), получаемая этими алгоритмами, считается объектом на изображении. Некоторые популярные алгоритмы этой категории — это нормализованные разрезы графов, случайное блуждание, минимальный разрез, изопериметрическое разделение

В сегментации методом водораздела рассматривается абсолютная величина градиента изображения как топографической поверхности. Пиксели, имеющие наибольшую абсолютную величину градиента яркости, соответствуют линиям водораздела, которые представляют границы областей. Вода, помещённая на любой пиксель внутри общей линии водораздела, течёт вниз к общему локальному минимуму яркости. Пиксели, от которых вода стекается к общему минимуму, образуют водосбор, который представляет сегмент.

Сегментация с помощью модели

Основное предположение этого подхода — то, что интересующие структуры или органы имеют повторяющиеся геометрические формы. Следовательно, можно найти вероятностную модель для объяснения изменений формы органа и затем, сегментируя изображение, накладывать ограничения, используя эту модель как априорную. Такое задание включает в себя (i) приведение тренировочных примеров к общей позе, (ii) вероятностное представление изменений приведённых образцов и (iii) статистический вывод для модели и изображения. Современные методы в литературе для сегментации, основанной на знании, содержат активные модели формы и внешности, активные контуры, деформируемые шаблоны и методы установления уровня.

Описание алгоритма

Сегментацию можно разделить на «полную» (сегментация в привычном понимании) и «неполную» - выделение какого-то определенного цвета на изображении. Вам предлагается готовая программа для неполной сегментации – выделение зеленого цвета на изображении (все объекты не зеленого цвета закрашиваются белым). Данная программа производит сегментацию основанную на цвете. По умолчанию вopencv цветное изображение хранится палитре BGR. Определять цвет в

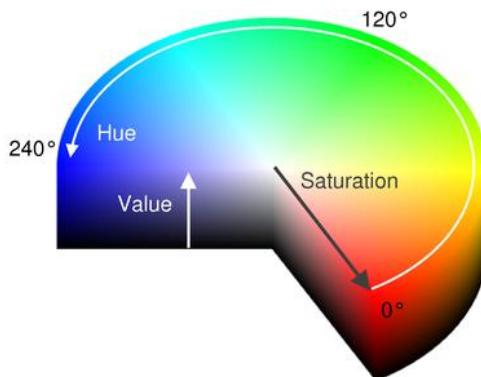
BGR не очень удобно, поэтому для начала переведем изображение в формат HSI.

HSI означает Hue, Saturation, Intensity, где:

Hue — цветовой тон, т.е. оттенок цвета.

Saturation — насыщенность. Чем выше этот параметр, тем «чище» будет цвет, а чем ниже, тем ближе он будет к серому.

Intensity (value) — значение (яркость) цвета. Чем выше значение, тем ярче будет цвет (но не белее). А чем ниже, тем темнее (0% — черный)



Так как выполняется сегментация по цвету, то больше всего нас интересует именно тон. Преобразуем изображение в палитру HSI и разобьем на три составляющие Hue Saturation Value соответственно. Зададим диапазон значений тона. В OpenCV зеленый находится в диапазоне от 34 до 72. Опытным путем был подобран диапазон от 21 до 110.

Далее пробежимся по нашему изображению. Для каждого пикселя получим все три компоненты. Если тон не укладывается в заданный диапазон или яркость слишком низкая — значит цвет не зеленый, поэтому закрашиваем все белым цветом.

Задание на работу

1. На основе данной программы вам предлагается сделать программу для полной сегментации — т.е. не только выбирать один цвет и закрашивать все остальное белым, но выделить на изображении похожие по цвету сегменты и закрасить их одним цветом (см. ниже).

2. Также можно сделать программу для сегментации видео, а не статического изображения. Подсказка: разбейте видео на кадры и произведите сегментацию кадров.

OpenCV2

Функция	Назначение
calcHist(&hsv_planes[0], 1, 0, Mat(), h_hist, 1, &histSize, &histRange, uniform, accumulate);	Вычисление гистограммы для изображения
.init(cv::Mat, cv::Rect2d); ->init(cv::Mat, cv::Rect2d);	Метод, отвечающий за инициализацию.
. segm.at<Vec3b>(now.y, now.x)=image.at<Vec3b>(pp.y, pp.x)	Обновление сегмента, получения данных о пикселе.
getStructuringElement(MORPH_ELLIPSE, Size(an * 2 + 1, an * 2 + 1), Point(an, an)); generateGradient(image);	Морфологическое замыкание для удаления остаточных шумов Находится градиент по изображению
regionMerge(Mat image ,Mat texture, Mat col, vector<int>pixelsInArea, vector<Vec3b>avgColBGR, int winSize);	Создается окошко для показа изображения
combined.at<Vec3b>(seed.y, seed.x)	Вычисляет не является ли пиксель пустым
imshow("image.jpg", im); imshow("segfinal",combined);	Выводит изображение на экран

Шаблон программы

```
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/ml/ml.hpp>
```

```

#include <opencv2/imgproc/imgproc.hpp>
#include <vector>

using namespace cv;
using namespace std;

const int GREEN_MIN = 21;

const int GREEN_MAX = 110;

void main(int argc, char** argv) {

    Mat src = imread("b.jpg"); //Исходное изображение

    imshow("input", src);
    imwrite("src.jpg", src);

    //Переводим в формат HSI
    Mat hsv = Mat(src.cols, src.rows, 8, 3); //

    vector<Mat> splitedHsv = vector<Mat>();

    cvtColor(src, hsv, CV_BGR2HSV);

    split(hsv, splitedHsv);

    //Удаляем фон
    for (int y = 0; y < hsv.cols; y++) {

        for (int x = 0; x < hsv.rows; x++) {

            // получаем HSV-компоненты пикселя
            int H = static_cast<int>(splitedHsv[0].at<uchar>(x, y));
// Тон
            int S = static_cast<int>(splitedHsv[1].at<uchar>(x, y));
// Интенсивность
            int V = static_cast<int>(splitedHsv[2].at<uchar>(x, y));
// Яркость

            //Если яркость слишком низкая либо
Тон не попадает в заданный диапазон, то закрашиваем белым
            if ((V < 20) || (H < GREEN_MIN) || (H > GREEN_MAX)) {
                src.at<Vec3b>(x, y)[0] = 255;
                src.at<Vec3b>(x, y)[1] = 255;
                src.at<Vec3b>(x, y)[2] = 255;
            }
        }
    }

    Mat tmp;

    //Морфологическое замыкание для удаления остаточных шумов.
    int an = 5;
    Mat element = getStructuringElement(MORPH_ELLIPSE, Size(an * 2 + 1, an * 2
+ 1), Point(an, an));
    dilate(src, tmp, element);
    erode(tmp, tmp, element);
}

```

```

//Переводим изображение в чернобелый формат

Mat grayscaleMat;

cvtColor(tmp, grayscaleMat, CV_BGR2GRAY);

//Делаем бинарную маску
Mat mask(grayscaleMat.size(), grayscaleMat.type());
Mat out(src.size(), src.type());
threshold(grayscaleMat, mask, 200, 255, THRESH_BINARY_INV);

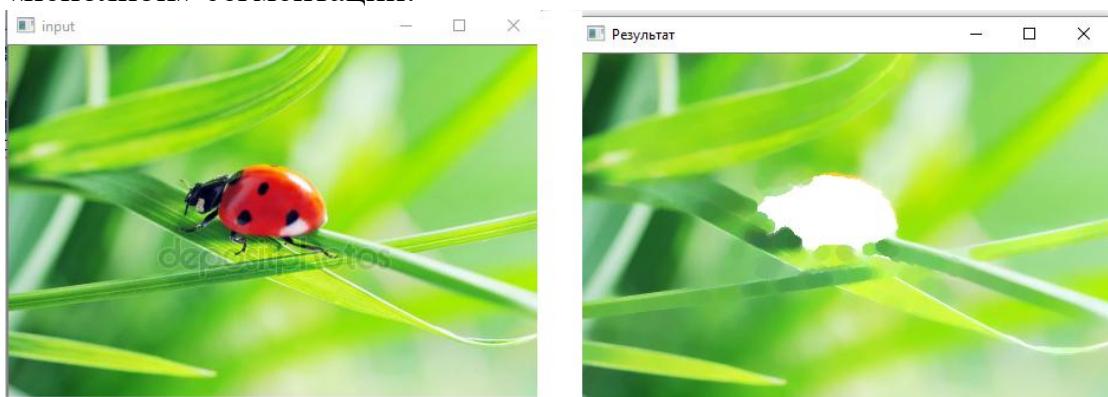
//Финальное изображение предварительно красим в белый цвет
out = Scalar::all(255);
//Копируем зашумленное изображение через маску
src.copyTo(out, mask);

imshow("До замыкания", src);
imshow("Результат замыкания", tmp);
imshow("Результат", out);

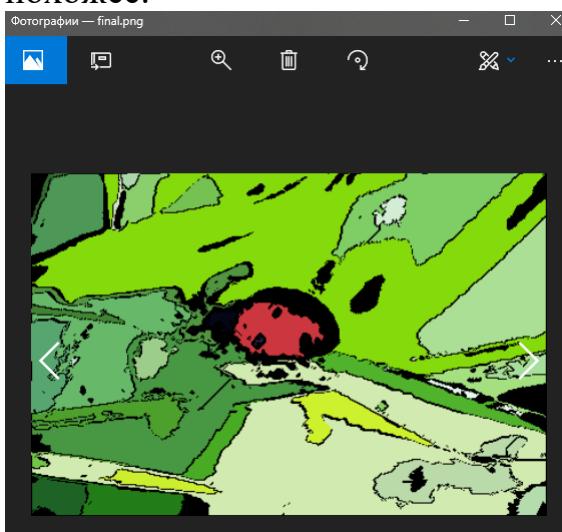
waitKey();

```

Исходное изображение и результат работы данной программы для «неполной» сегментации:



В результате полной сегментации у вас должно получиться что-то похожее:



Лабораторная работа по теме №10

" Применение преобразования Хаара для подавления шумов на изображении "

В данной работе требуется подавить шум на изображении. В качестве примера используется двумерное вейвлет преобразование (2 преобразования Хаара), т.к. вейвлет алгоритмы очень популярны в последнее время и являются пригодными для решения множества задач, а также оно позволяет эффективно и с небольшими потерями убрать шумы с изображения.

Алгоритм разложения цифровых изображений с помощью двумерного дискретного вейвлет преобразования

Преобразование Хаара

Преобразование Хаара (ПХ) является одним из простейших и базисным вейвлет преобразованием. Пусть имеется одномерный дискретный сигнал $f(f_1, f_2, \dots, f_n)$. ПХ разлагает каждый сигнал на два компонента равного размера. Первый из компонентов называется средним или аппроксимацией (approximation), а второй известен как различие (difference) или деталь (detail). Точная формула для среднего значения подсигнала (subsignal), $a^1 = (a_1, a_2, \dots, a_{n-2})$, на первом уровне для одного сигнала длины N, т. е. $f(f_1, f_2, \dots, f_n)$ имеет вид :

$$a_n = \frac{f_{2n-1} + f_{2n}}{\sqrt{2}}, \quad n = 1, 2, 3, \dots, N/2,$$

и детализирующий подсигнал, $d^1 = (d_1, d_2, \dots, d_{n-2})$, на этом же уровне

$$d_n = \frac{f_{2n-1} - f_{2n}}{\sqrt{2}}, \quad n = 1, 2, 3, \dots, N/2.$$

представляется как

Эти значения формируют два новых сигнала $a\{a_n\}$, n из Z и $d\{d_n\}$, n из Z, один из которых является огрубленной версией исходного сигнала (каждой паре элементов f соответствует их среднее арифметическое), а другой содержит информацию (будем называть ее детализирующей), необходимую для восстановления исходного сигнала. Действительно

$$f_{2n-1} = a_n + d_n,$$

$$f_{2n} = a_n - d_n.$$

К сигналу a можно применить аналогичную операцию и также получить два сигнала, один из которых является огрубленной версией a , а другой содержит детализирующую информацию, необходимую для

восстановления а. Чтобы понять, как преобразование Хаара работает, рассмотрим простой пример.

Предположим, что

$$I = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 1 & 2 \\ 3 & 4 & 5 & 6 \end{pmatrix}.$$

В случае применения 1D ПХ вдоль первой строки, коэффициенты аппроксимации следующие

$$\frac{1}{\sqrt{2}}(1+2) \text{ и } \frac{1}{\sqrt{2}}(3+4).$$

и коэффициенты различия

$$\frac{1}{\sqrt{2}}(1-2) \text{ и } \frac{1}{\sqrt{2}}(3-4).$$

То же самое преобразование применяется к другим строкам I. Помещая коэффициенты аппроксимации каждой строки в первые два столбца и соответствующие коэффициенты различия в последующие два столбца, получим следующие результаты

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 1 & 2 \\ 3 & 4 & 5 & 6 \end{pmatrix} \xrightarrow[1D \text{ ПХ на строках}]{\frac{1}{\sqrt{2}}} \begin{pmatrix} 3 & 7 & : & -1 & -1 \\ 9 & 13 & : & -1 & -1 \\ 17 & 3 & : & -1 & -1 \\ 7 & 11 & : & -1 & -1 \end{pmatrix}.$$

В приведенном соотношении коэффициенты аппроксимации и коэффициенты различия отделяются точками в каждой строке. Применяя на следующем шаге 1D ПХ к столбцу результирующей матрицы, находим, что результирующая матрица на первом уровне имеет вид

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 3 & 7 & : & -1 & -1 \\ 9 & 13 & : & -1 & -1 \\ 17 & 3 & : & -1 & -1 \\ 7 & 11 & : & -1 & -1 \end{pmatrix} \xrightarrow[1D \text{ ПХ на столбцах}]{\frac{1}{2}} \begin{pmatrix} 12 & 20 & : & -2 & -2 \\ 24 & 14 & : & -2 & -2 \\ \dots & \dots & : & \dots & \dots \\ -6 & -6 & : & 0 & 0 \\ 10 & -8 & : & 0 & 0 \end{pmatrix}.$$

Таким образом, имеем

$$A = \begin{pmatrix} 12 & 20 \\ 24 & 14 \end{pmatrix}, H = \begin{pmatrix} -2 & -2 \\ -2 & -2 \end{pmatrix},$$

$$V = \begin{pmatrix} -6 & -6 \\ 10 & -8 \end{pmatrix} \text{ и } D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

У каждой части, показанной в приведенном выше примере, есть размерность (число строк / 2) (число столбцов/2) и эти области называются A, H, V и D соответственно. A (область приближения) включает информацию о глобальных свойствах проанализированного изображения. Удаление спектральных коэффициентов от этой области приводит к самому большому искажению в исходном изображении. H (горизонтальная область), включает информацию о вертикальных строках, скрытых в изображении. Удаление спектральных коэффициентов от этой области исключает горизонтальные детали из исходного изображения. содержит информацию о горизонтальных строках, скрытых в изображении. Удаление спектральных коэффициентов от этой области устраниет вертикальные детали из исходного изображения. D (диагональная область) охватывает информацию о диагональных деталях, скрытых в изображении. Удаление спектральных коэффициентов от этой области приводит к минимальному искажению в исходном изображении. Таким образом, ПХ подходит для приложения, когда матрица изображения имеет число строк и столбцов, кратное числу

Двумерное дискретное вейвлет преобразование

Двумерное дискретное вейвлет преобразование состоит из поочередного одномерного вейвлет преобразования строк и столбцов этой матрицы. Сначала выполняются одномерные вейвлет преобразования каждой строки в отдельности, преобразованная строка записывается на прежнее место. Элементы нумеруются способом, указанным в предыдущих разделах. Далее вейвлет преобразования применяются ко всем столбцам. В результате изображение разбивается на четыре равные части. На Рис. 1 показаны стандартные обозначения квадрантов преобразованного изображения: LL, LH, HL, HH.

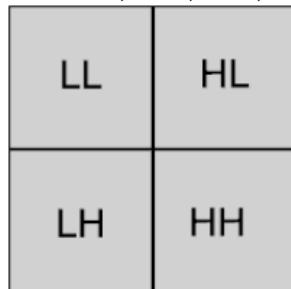


Рис. 1. Применение двумерного преобразования к квадратной матрице

Квадрант LL соответствует низкочастотным вейвлет коэффициентам, HH – высокочастотным вейвлет коэффициентам (буква L означает Low, H – High).

Если не оговорено противное, под N-кратным двумерным вейвлет преобразованием понимается применение N раз двумерного вейвлет преобразования, причём очередное двумерное вейвлет преобразование применяется к младшей четверти матрицы (квадрант LL на первом рисунке). В итоге N-кратное преобразование выглядит так, как показано на Рис. 2 (при N=3). На нем показаны принятые стандартные обозначения квадрантов изображения. Квадранты N-кратного двумерного вейвлет-преобразования имеют аналогичное обозначение.

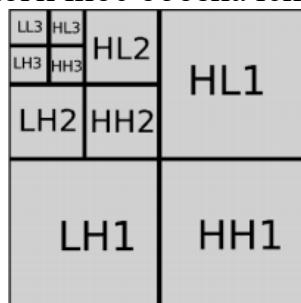


Рис. 2. Трёхкратное применение двумерного вейвлет преобразования

Обратное двумерное вейвлет преобразование рекурсивно восстанавливает младший квадрант. В случае, представленном на Рис. 2, для получения (восстановления) нового квадранта LL2 используются квадранты LL3, LH3, HL3 и HH3. Далее для восстановления квадранта LL1 используются квадранты LL2, LH2, HL2, HH2 и т. д. Аналогично выполняется N-кратное обратное вейвлет преобразование. Заметим, что указанное преобразование является иерархическим, т. е. если при применении обратного вейвлет преобразования вычисляются не все уровни, а меньшее их количество, то в квадранте LL образуется уменьшенная копия изображения, как показано на Рис.3.

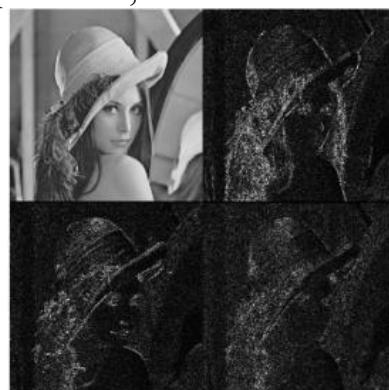


Рис. 3. Однократное применение двумерного вейвлет преобразования или применение (N–1) кратного обратного к изображению, полученному N кратным вейвлет преобразованием

В частности, если вообще не используется обратное вейвлет преобразование, то самый младший квадрант тоже является уменьшенной копией изображения. Благодаря этому свойству, обратное вейвлет преобразование позволяет вырезать фрагменты изображений при различных масштабах. Следует отметить, что, во-первых, доступные масштабы определяются количеством уровней вейвлет преобразования и, во-вторых, масштабы не произвольны, а отличаются увеличением в два раза.

Описание работы программы

Программа состоит из одной задачи. В ней показывается, каким образом с помощью двумерного вейвлет преобразования (для этой работы алгоритм был изменен с целью обработки цветных изображений без конвертации в серый цвет) и порогового значения можно убрать шумы на фоне, и тем самым сгладить изображение. Начиная с некоторого порогового значения так же можно наблюдать “лестничный эффект”.

Алгоритм для задачи:

1. Получить изображение;
2. Определив, какое количество каналов у изображения, отправить их на соответствующую количеству обработку двумерному преобразованию, затем с помощью обратного преобразования получить обратно исходное изображения;
3. Показать результат;
4. Перемещая ползунок трекбара, прикрепленного к исходному изображению, с одной позиции на другую, изменять пороговое значение, наблюдая за тем, как постепенно исчезают шумы с изображения, а также постепенно проявляется “лестничный эффект” на выходном (“Filtered”) изображении.

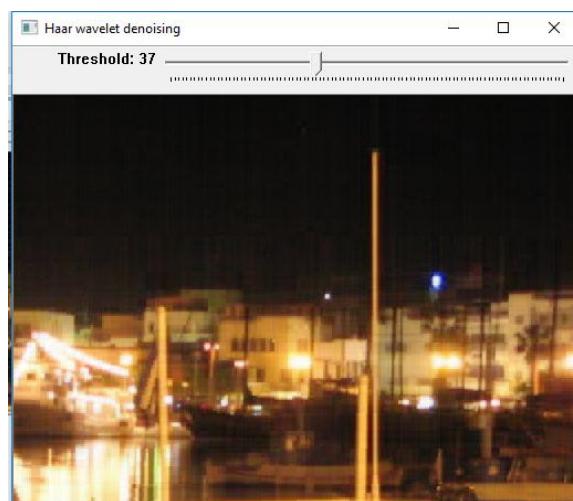
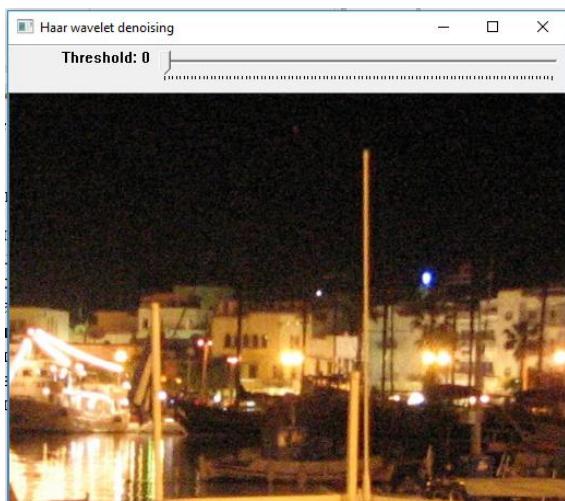


Рис. 5. Исходное изображение (слева) и обработанное с порогом равным 37

Задания

1. Дополнить задание обработкой видео на шумы с возможностью выбора порогового значения при передвижении ползунка трекбара;
2. Дополнить задание обработкой в режиме реального времени (веб-камера) на шумы с возможностью выбора порогового значения с помощью ползунка трекбара. Использовать кроме функции подавления шумов двумерным преобразованием функции из библиотеки OpenCV для подавления шумов: фото – fastNIMeansDenoisingColored, видео – fastNIMeansDenoisingMulti, fastNIMeansDenoisingColoredMulti. Сравнить их действие и показать результаты лучшего. Сравнение проводить с помощью проверки на размытость (Blur Detection).

Использованные функции

Функция	Описание
void cv::split(InputArray m, OutputArrayOfArrays mv)	Разделяет многоканальный массив на несколько одноканальных массивов.
void cv::imshow(const String & winname, InputArray mat)	Отображает изображение в указанном окне.
void cv::destroyWindow(const String & winname)	Уничтожает указанное окно.
int cv::waitKey(int delay =0)	Ожидает нажатия клавиши.
void cv::merge(InputArrayOfArrays mv, OutputArray dst)	Создает один многоканальный массив из нескольких одноканальных.
void cv::Mat::convertTo(OutputArray m, int rtype, double alpha=1, double beta=0)	Изменяет размерность каналов изображения
void resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER_LINEAR)	Изменяет размеры изображения.
void cv::copyMakeBorder(InputArray src, OutputArray dst, int top, int bottom, int left, int right, int borderType, const Scalar &value = Scalar())	Формирует барьер вокруг изображения
int cv::createTrackbar(const string& trackbarname, const string& winname, int* value, int count, TrackbarCallback onChange=0, void* userdata=0)	Создает трекбар (ползунок или управление диапазоном) с указанным именем и диапазоном, присваивает переменное значение позиции, синхронизированной с трекбаром, и указывает функцию обратного вызова onChange для вызова изменения позиции трекбара. Созданный трек-бар отображается в указанном окне winname.

Применение преобразования

Файл “task.hpp”:

```
#ifndef _TASK_HPP_
#define _TASK_HPP_

#include <opencv2/opencv.hpp>

#include <iostream>
#include <cmath>
#include <cassert>
#include <conio.h>

using namespace std;
using namespace cv;

void HaarDecomposeArray(float *A, int width);
void HaarDecomposeImage(float *A, int width, int height);
void HaarTransposeArray(float *A, int width, int height);
void HaarReconstructArray(float *A, int width);
void HaarReconstructImage(float *A, int width, int height);

// для обработки одноканального изображения
void Denoise(const Mat &input, Mat &output, float threshold);

// Трэкбар для изменения порога (threshold)
void setTrackbar(int, void*);

// Устанавливаем размер 512 * 512 для удобства
const int MAX_IMAGE_SIZE = 512;

Mat image;
Mat spec_image;
Mat denoised_image;
int slider;
float this_threshold = 0;

int task2(char *filename)
{
    image = imread(filename);
    assert(image.data);

    // Haar Wavelet requires image to be a power of two
    // We'll pad the borders and remove them later
    int max_dim = max(image.rows, image.cols);
    int padded_width = 0;
    int padded_height = 0;

    if (max_dim > MAX_IMAGE_SIZE) {
        resize(image, image, Size(image.cols * MAX_IMAGE_SIZE / max_dim, image.rows * MAX_IMAGE_SIZE / max_dim));
    }

    padded_width = (int)pow(2, ceil(log(image.cols) / log(2)));
    padded_height = (int)pow(2, ceil(log(image.rows) / log(2)));

    // Создаем барьер, чтобы потом уместить в него преобразованное изображение
    copyMakeBorder(image, spec_image, 0, padded_height - image.rows, 0, padded_width - image.cols, BORDER_CONSTANT);
```

```

//cout << "Padded image: " << g_padded_img.size() << endl;

namedWindow("Haar wavelet denoising");
//imshow("Haar wavelet denoising", g_noisy_img);
createTrackbar("Threshold", "Haar wavelet denoising", &slider, 100, setTrackbar);

setTrackbar(slider, 0);
waitKey(0);
return 0;
}

void setTrackbar(int, void*)
{
    this_threshold = 0.1 * slider / 100.0;

    if (spec_image.channels() == 3)
    {
        Mat bgr[3];
        split(spec_image, bgr);

        Denoise(bgr[0].clone(), bgr[0], this_threshold);
        Denoise(bgr[1].clone(), bgr[1], this_threshold);
        Denoise(bgr[2].clone(), bgr[2], this_threshold);

        merge(bgr, 3, denoised_image);
    }
    else
    {
        Denoise(spec_image, denoised_image, this_threshold);
    }

    denoised_image = denoised_image(Range(0, image.rows), Range(0, image.cols));
    imshow("Haar wavelet denoising", denoised_image);
}

void HaarDecomposeArray(float *A, int width)
{
    const float inv_sqrt2 = 1.f / sqrtf(2);

    float norm = 1.f / sqrtf(width);

    for (int i = 0; i < width; i++) {
        A[i] *= norm;
    }

    float *tmp = new float[width];

    while (width > 1) {
        width /= 2;

        for (int i = 0; i < width; i++) {
            tmp[i] = (A[2 * i] + A[2 * i + 1]) * inv_sqrt2;
            tmp[width + i] = (A[2 * i] - A[2 * i + 1]) * inv_sqrt2;
        }

        memcpy(A, tmp, width * 2 * sizeof(float));
    }

    delete[] tmp;
}

```

```

void HaarTransposeArray(float *A, int width, int height)
{
    float *B = new float[width*height];

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            B[x*height + y] = A[y*width + x];
        }
    }

    memcpy(A, B, sizeof(float)*width*height);

    delete[] B;
}

void HaarDecomposeImage(float *A, int width, int height)
{
    for (int i = 0; i < height; i++) {
        HaarDecomposeArray(&A[i*width], width);
    }

    HaarTransposeArray(A, width, height);

    for (int i = 0; i < width; i++) {
        HaarDecomposeArray(&A[i*height], height);
    }

    HaarTransposeArray(A, height, width);
}

void HaarReconstructArray(float *A, int width)
{
    const float inv_sqrt2 = 1.f / sqrt(2.f);
    float inv_norm = sqrtf(width);

    float *tmp = new float[width];
    int k = 1;

    while (k < width) {
        for (int i = 0; i < k; i++) {
            tmp[2 * i] = (A[i] + A[k + i]) * inv_sqrt2;
            tmp[2 * i + 1] = (A[i] - A[k + i]) * inv_sqrt2;
        }

        memcpy(A, tmp, sizeof(float)*(k * 2));

        k *= 2;
    }

    for (int i = 0; i < width; i++) {
        A[i] *= inv_norm;
    }

    delete[] tmp;
}

void HaarReconstructImage(float *A, int width, int height)
{
    for (int i = 0; i < width; i++) {

```

```

        HaarReconstructArray(&A[i*height], height);
    }

    HaarTransposeArray(A, height, width);

    for (int i = 0; i < height; i++) {
        HaarReconstructArray(&A[i*width], width);
    }

    HaarTransposeArray(A, width, height);
}

void Denoise(const Mat &input, Mat &output, float inp_threshold)
{
    assert(input.data != output.data);

    input.convertTo(output, CV_32F);
    HaarDecomposeImage((float*)output.data, output.cols, output.rows);

    for (int y = 0; y < output.rows; y++) {
        float *ptr = (float*)output.ptr(y);

        for (int x = 0; x < output.cols; x++) {
            // signbit возвращает 1 для отрицательных значений, и 1 для положительных
            // Ослабляем коэффициенты мягким сжатием
            ptr[x] = (signbit(ptr[x]) == 1 ? -1 : 1) * max(0.f, fabs(ptr[x]) -
inp_threshold);
        }
    }

    // Посмотреть результат преобразования, если требуется
    //imshow("Haar result", output);

    HaarReconstructImage((float*)output.data, output.rows, output.cols);
    output.convertTo(output, CV_8U);
}

#endif // _TASK_2_HPP_

```

Файл “main.cpp”:

```

#include <opencv2/opencv.hpp>
#include "task.hpp"

using namespace cv;

int main(int argc, char** argv)
{
    while (1)
    {
        char filename[200];
        std::cout << "Enter filename:\n";
        std::cin >> filename;
        task2(filename);

        destroyAllWindows();
        std::cout << "\n\n";
    }

    return 0;
}

```

Лабораторная работа №11

"Морфологическая обработка изображений"

Введение

Основные операции математической морфологии:

- dilation,
- эрозия.

Две базовые операции математической морфологии, на которых строится буквально абсолютно всё, — это так называемые delation, erosion. По-русски delation называют и дилатацией, и расширением, и наращиванием, кто во что горазд. По сути это, на самом деле, расширение какое-то, такое распухание объекта, которое было изначально на изображении.

Erosion — сужение, это такое похудение объекта, которое было на изображении. И сейчас мы посмотрим, как всё это происходит.

Производные морфологические операции:

- opening
- closing

Две ещё не менее важные операции, которые используются, — это размыкание и замыкание (opening/closing), которые строятся как суперпозиция из первых двух базовых. И собственно, что они делают, это тоже мы с вами проговорим.

Эти четыре операции, которые используются наиболее активно. Дальше мы ещё будем рассматривать прочие производные операции, которые в принципе, наверно, тоже используются, но в достаточно ограниченном наборе задач.

Dilation
Расширение, наращивание, дилатация

Расширение множества A по множеству B:

$$A \oplus B = \{z | (\hat{B})_z \cap A \neq \emptyset\}$$
$$A \oplus B = \bigcup_{b \in B} A_b$$


B – структурный примитив (элемент), структурообразующее множество

Давайте перейдём к рассмотрению этих самых задач. Что есть расширение. Представьте себе, что у вас есть изначальный объект, это вот этот синенький квадратик внутри, и структурный элемент у нас здесь представлен в виде вот этого диска. Для структурного элемента должен обязательно быть задан так называемый начальный элемент, и чаще всего структурный элемент у нас бывает симметричный, и начальный элемент совпадает с его центром.

Основные применения шаблонов

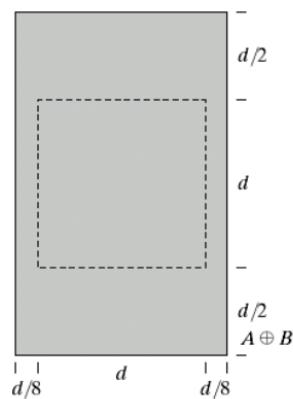
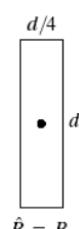
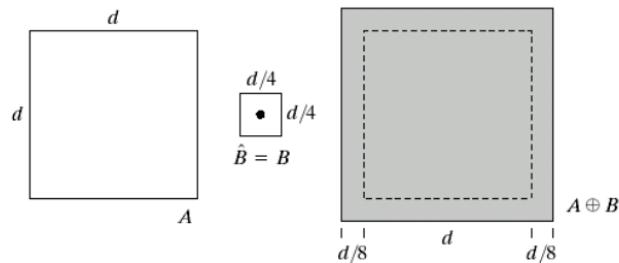
Основное применение расширения — это когда у нас есть какие-то жидкие объекты и нам нужно построить мосты между кусками и, вообще говоря, одной и той же компоненты связности. Довольно часто применяется при распознавании текста, то есть если у нас шрифт какой-нибудь такой, когда есть явные промежутки в буквах, чтобы эти буквы были распознаваемы. То есть чтобы контур, по крайней мере, был соответствующий, то применяют в частности расширение. В чём, наверно, основной минус всех морфологических методов? В том, что нужно каждый раз очень чётко подбирать маску по размеру и по форме, потому что в частности вот это вот очень сильно зависит от размера шрифта, и в том случае, если у нас здесь разрыв, скажем, в три пикселя, он свяжет это дело, если больше, то не свяжет. Либо надо применять несколько последовательных операций расширения, и смотреть, что получается, то есть такая техника.

Dilation: примеры

a b c
d e

FIGURE 9.4

- (a) Set A .
- (b) Square structuring element (dot is the center).
- (c) Dilation of A by B , shown shaded.
- (d) Elongated structuring element.
- (e) Dilation of A using this element.



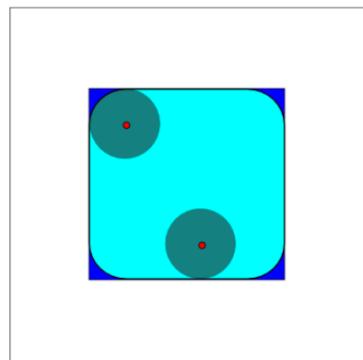
Операция «эрозия» в некотором смысле обратна расширению. Не в математическом смысле обратна. Эрозией или размыванием ещё называют следующий результат. Здесь у нас опять синенький квадрат, это наше исходное множество. Диск — это структурный элемент и результатом эрозии является такое множество, что если мы помещаем, ну то есть, так что мы передвигаем структурный элемент по нашему исходному множеству и берём только те точки, которые покрываются и изображением, и структурным элементом. То есть, если у нас вдруг структурный элемент начинает вылезать за пределы картинки, то все точки структурного элемента должны помещаться в исходный объект. Здесь центр не играет такой большой роли.

Эрозия

Erosion

Эрозия множества A по множеству B:

$$A \ominus B = \{z | B_z \subseteq A\}$$



B – структурообразующее множество (примитив)

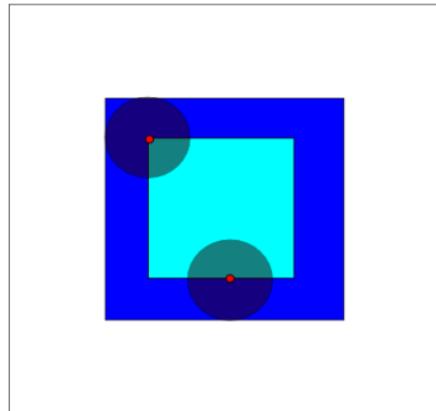
Здесь, соответственно, примеры эрозии. Тот же самый объект, который у нас был с расширением, структурный элемент точно такой же, но здесь исходный объект помечен пунктиром, а результат эрозии закрашен в серый цвет. Тут тоже исходный объект и результатом эрозии будет линия. Ещё один пример эрозии — эрозия с последующим расширением. Здесь сначала была картинка из крупных квадратиков и мелких точек. Сначала была применена эрозия со структурным элементом, размер которого превышает размер мелких квадратиков, но меньше, чем крупные. И мы видим, что у нас в итоге остались только крупные в виде очень похудевшем. А дальше можно применить расширение, чтобы их нарастить.

Эрозия

Erosion

Эрозия множества A по множеству B:

$$A \ominus B = \{z | B_z \subseteq A\}$$



B – структурообразующее множество (примитив)

На самом деле, такое последовательное применение эрозии и расширения образует две наиболее часто использующие производные операции. Одна из них — размыкание, открытие (opening). Вторая — замыкание (closing). Отличаются они только порядком операции.

Размыкание — когда мы сначала производим эрозию, потом — расширение. **Замыкание** — наоборот.

Собственно, разница в эффектах следующая: в первом случае, когда мы применяем эрозию, у нас сглаживаются контуры объекта как бы изнутри, то есть мы углы объекта округляем. А в том случае, если мы применяем замыкание, тогда у нас тоже сглаживаются углы объекта, но как бы снаружи. То есть получается, что у нас реакцию размыкания можно представить так, как будто бы вы обкатываете шариком контуры объекта, они становятся гладкие. Замыкание — это когда наоборот мы делаем соответственно расширение, а потом — делаем эрозию.

Основные свойства у них следующие. Если мы берём исходный элемент a, структурный элемент b, то результат размыкания всегда будет лежать внутри исходного объекта. А здесь наоборот. Исходный объект будет содержать в себе замыкание. Если у нас есть два множества c и d, с является подмножеством d, то при выборе одного и того же структурного элемента, что результат размыкания, что результат замыкания тоже будет, ну то есть отношение вложенности сохранится. Свойства 2 и 3 у них

одинаковые. Ну и повторное применение одной и той же операции (размыкание, размыкание) будет эквивалентно одному размыканию, то же самое с замыканием.

Задание

На основе наработок Erosion и Deletion сделать 2 морфологические операции

Opening, Closening. Напоминаю, что Opening, Closening это супер множество над Erosion и Closening соответственно. Проверить примеры на основе приведенных изображений rice.bmp, morphology_dots.jpg, morphology.png

Шаблон кода:

```
#include "stdafx.h"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

using namespace std;
using namespace cv;

#define BINARY_THRESHOLD 127

/*Function to create a diagonal 3x3 structuring element
The function checks for the flag if it is Dilate or Erode.
In either case it assigns appropriate value to the intensity
and creates an output image.*/
Mat structureType3(Mat inputImage, string flag) {
    Mat outputImage = inputImage.clone();
    int intensity;

    else if (flag == "Erode") {
        cout << "Eroding the Binary Image" << endl;
        intensity = 0;
    }

    for (int i = 0; i < inputImage.rows; i++) {
        for (int j = 0; j < inputImage.cols; j++) {
```

```

        if (inputImage.at<uchar>(i, j) == intensity) {

            outputImage.at<uchar>(i, j) = intensity;
            if (i > 0)
                outputImage.at<uchar>(i - 1, j) = intensity;
            if (j > 0)
                outputImage.at<uchar>(i, j - 1) = intensity;
            if ((i + 1) < inputImage.rows)
                outputImage.at<uchar>(i + 1, j) = intensity;
            if ((j + 1) < inputImage.cols)
                outputImage.at<uchar>(i, j + 1) = intensity;
        }
    }
    return outputImage;
}

```

```

Mat structureType2(Mat inputImage, string flag) {
    Mat outputImage = inputImage.clone();

    int intensity;
    if (flag == "Dilate") {
        cout << "Dilating the Binary Image" << endl;
        intensity = 255;
    }
    else if (flag == "Erode") {
        cout << "Eroding the Binary Image" << endl;
        intensity = 0;
    }

    for (int i = 0; i < inputImage.rows; i++) {
        for (int j = 0; j < inputImage.cols; j++) {
            if (inputImage.at<uchar>(i, j) == intensity) {

                outputImage.at<uchar>(i, j) = intensity;
                if (i > 0)
                    outputImage.at<uchar>(i - 1, j) = intensity;
                if (j > 0)
                    outputImage.at<uchar>(i, j - 1) = intensity;
                if (i > 0 && j > 0)
                    outputImage.at<uchar>(i - 1, j - 1) = intensity;
                if ((i + 1) < inputImage.rows)

```

```

        outputImage.at<uchar>(i + 1, j) = intensity;
        if ((j + 1) < inputImage.cols)
            outputImage.at<uchar>(i, j + 1) = intensity;
        if ((i + 1) < inputImage.rows && (j + 1) <
inputImage.cols)
            outputImage.at<uchar>(i + 1, j + 1) = intensity;
        if (i > 0 && (j + 1) < inputImage.cols)
            outputImage.at<uchar>(i - 1, j + 1) = intensity;
        if ((i + 1) < inputImage.rows && j > 0)
            outputImage.at<uchar>(i + 1, j - 1) = intensity;
    }
}
return outputImage;
}

```

```

Mat structureType1(Mat inputImage, string flag) {
    Mat outputImage = inputImage.clone();

    int intensity;
    if (flag == "Dilate") {
        cout << "Dilating the Binary Image" << endl;
        intensity = 255;
    }
    else if (flag == "Erode") {
        cout << "Eroding the Binary Image" << endl;
        intensity = 0;
    }

    for (int i = 0; i < inputImage.rows; i++) {
        for (int j = 0; j < inputImage.cols; j++) {
            if (inputImage.at<uchar>(i, j) == intensity) {
                outputImage.at<uchar>(i, j) = intensity;
                if ((j + 1) < inputImage.cols)
                    outputImage.at<uchar>(i, j + 1) = intensity;
            }
        }
    }
    return outputImage;
}

```

```

Mat createBinaryImage(Mat inputImage) {
    Mat binaryImage = inputImage.clone();

    for (int i = 0; i < inputImage.rows; i++) {
        for (int j = 0; j < inputImage.cols; j++) {
            if (inputImage.at<uchar>(i, j) >= BINARY_THRESHOLD) {
                binaryImage.at<uchar>(i, j) =
saturate_cast<uchar>(255);
            }
            else {
                binaryImage.at<uchar>(i, j) = saturate_cast<uchar>(0);
            }
        }
    }
    return binaryImage;
}

```

```

Mat callStructure(Mat binaryImage, string flag, int structureOption) {
    Mat outputImage;
    switch (structureOption) {
        case 1:
            cout << "Structure Type " << structureOption << endl;
            outputImage = structureType1(binaryImage, flag);
            break;
        case 2:
            cout << "Structure Type " << structureOption << endl;
            outputImage = structureType3(binaryImage, flag);
            break;
        case 3:
            cout << "Structure Type " << structureOption << endl;
            outputImage = structureType2(binaryImage, flag);
            break;
        case 4:
            cout << "Structure Type " << structureOption << endl;
            for (int k = 0; k < 4; k++) {
                outputImage = structureType2(binaryImage, flag);
                binaryImage = outputImage;
            }
            break;
        case 5:
            cout << "Structure Type " << structureOption << endl;
    }
}

```

```

        for (int k = 0; k < 7; k++) {
            outputImage = structureType2(binaryImage, flag);
            binaryImage = outputImage;
        }
        break;
    default:
        cout << "Structure Type Not Found" << endl;
    }
    return outputImage;
}

/*Function to read an image using the file name given by the user*/
Mat readImage(string &fileName, string type) {
    cout << endl << "Please Select " << type << " Image." << endl;
    cout << "Example <rice.bmp> <morphology.png> <dots_morpho.jpg>" 
<< endl;

    cin >> fileName;
    //fileName = "rice.bmp";

    cout << "File Selected: " << fileName << endl;
    Mat inputImage = imread(fileName,
CV_LOAD_IMAGE_GRAYSCALE);

    return inputImage;
}

int main() {

    int option;
    int structureOption;
    string fileName;

    cout << "Assignment 5: Morphological Operations" << endl;
    cout << "Select an Image " << endl;

    Mat inputImage = readImage(fileName, "Input");
    if (inputImage.empty()) {
        cerr << "Error: Loading image" << endl;
        _getch();
        return -1;
}

```

```

}

Mat binaryImage = createBinaryImage(inputImage);

cout << "Select the following options" << endl;
cout << " 1. Erode Binary" << endl;
cout << " 2. Dilate Binary" << endl;
cout << " 3. Open Binary" << endl;
cout << " 4. Close Binary" << endl;
cin >> option;

cout << "Please enter the type of stuctural Element" << endl;
cout << " 1. A rectangle with of 1x2" << endl;
cout << " 2. A diamond with all 1 3x3" << endl;
cout << " 3. A Square with all 1 3x3" << endl;
cout << " 4. A Square with all 1 9x9" << endl;
cout << " 5. A Square with all 1 15x15" << endl;
cin >> structureOption;

namedWindow("Input Image");
imshow("Input Image", inputImage);

namedWindow("Input Binary Image");
imshow("Input Binary Image", binaryImage);

Mat outputImage;
Mat outputImage_2;
switch (option) {
case 1:
    cout << "Image Erode Binary" << endl;
    outputImage = callStructure(binaryImage, "Erode",
structureOption);
    namedWindow("Output Image");
    imshow("Output Image", outputImage);
    break;
case 2:
    cout << "Image Close Binary" << endl;
    outputImage = callStructure(binaryImage, "Dilate",
structureOption);
    outputImage_2 = callStructure(outputImage, "Erode",
structureOption);
    namedWindow("Output Dilated Image");
}

```

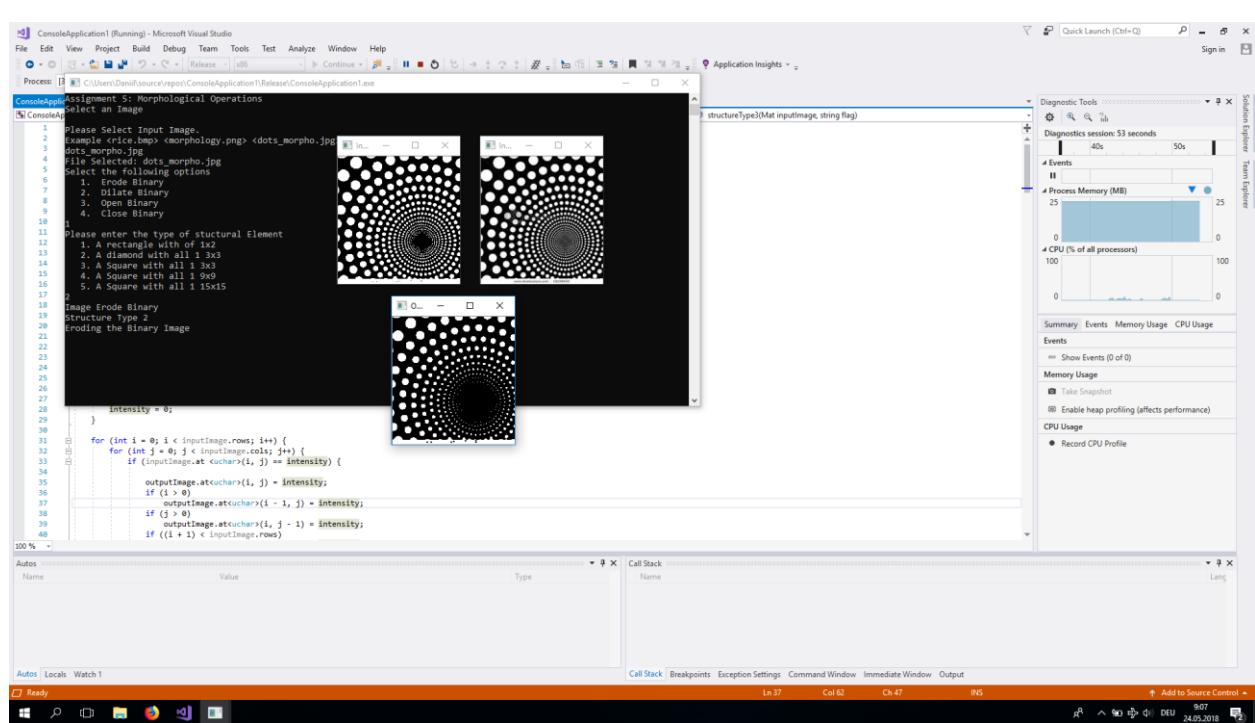
```

imshow("Output Dilated Image", outputImage);
namedWindow("Output Close Image");
imshow("Output Close Image", outputImage_2);
break;
default:
cout << "Option not found!!" << endl;
_getch();
return -1;
}

waitKey();

return 0;
}

```



Лабораторная работа №12

"Трекер"

Начиная с OpenCV 3.0 данная библиотека включает в себя tracking API. На данный момент API, предназначенное для отслеживания перемещений объекта на видео, не входит основную версию OpenCV, а находится в opencv_contrib.

tracking.hpp включает в себя 5 трекеров: BOOSTING, MIL, KCF, TLD, MEDIANFLOW. В данной работе необходимо реализовать программу, использующую каждый из этих трекеров, исследовать работу каждого из них, а также наметить основные различия.

Что мы понимаем под слежением за объектом

Упростив, отслеживание перемещения – поиск нужного объекта на каждом кадре. «Отслеживание объекта» является очень широким термином, под которым могут пониматься концептуально схожие идеи, но по-разному технически реализованные. Например:

- 1. Плотный оптический поток:** алгоритмы, оценивающие вектор движения каждого пикселя в видеокадре;
- 2. Разреженный оптический поток:** алгоритмы, отслеживающие несколько определенных точек в видеокадре;
- 3. Фильтр Калмана:** алгоритма обработки сигналов, используемый для прогнозирования местоположения движущегося объекта, на основе предшествующей информации о движении.

Часто понятия отслеживание и обнаружение считают взаимозаменяемыми, но это не так. Отслеживание позволяет просто следить за передвижением объекта, не рассматривая его сущность. Обнаружение выделяет объект из массы других. Практически любой трекинг алгоритм включает в себя и обнаружение. Особенно важно оно на первом кадре, для получения информации о том, какой объект нам нужно отслеживать. Также, алгоритм обнаружения используется каждые n кадров, на тот случай, если отслеживаемый объект был потерян. Зачем же нужно отслеживание, если можно просто на каждом кадре обнаруживать необходимый объект?

- 1. Отслеживание быстрее, чем обнаружение.** Причина для этого очевидна. Когда вы отслеживаете объект, обнаруженный в предыдущем кадре, вы многое знаете о внешнем виде объекта. Вы также знаете местоположение в предыдущем кадре, направление и скорость его движения. Поэтому в следующем кадре вы можете

использовать всю эту информацию для прогнозирования местоположения объекта.

2. **Хороший алгоритм отслеживания может работать, когда обнаружение невозможно.** Если важная часть объекта ушла из поля зрения, трекинг алгоритм может все равно знать о местоположении объекта, основываясь на предшествующей информации.
3. **Отслеживание может выделить объект из массы похожих.** Это происходит за счет информации о направлении движения, скорости и т.п. Обнаружение же просто найдёт несколько одинаковых объектов.

OpenCV Tracking API

BOOSTING Tracker. Трекер, основанный на каскаде Хаара. Первое выделение объекта – первый положительный пример. С дальнейшей работой алгоритма записывается все больше положительных примеров для отслеживаемого объекта.

MIL Tracker. Большая разница заключается в том, что вместо того, чтобы рассматривать только текущее местоположение объекта как положительный пример, он развертывается в небольшом окружении вокруг текущего местоположения, чтобы генерировать несколько потенциальных положительных примеров.

KCF Tracker. Объединяет в себе идеи двух предыдущий трекеров. Этот трекер использует тот факт, что множественные положительные образцы, используемые в трекер MIL, имеют большие перекрывающиеся области. Эти перекрывающиеся области позволяют получить некоторые математические свойства, которые используют этот трекер, чтобы сделать отслеживание быстрее и точнее в одно и то же время.

TLD Tracker. Расшифровка аббревиатуры – отслеживание, обучение, обнаружение. Трекер следует за объектом от кадра до кадра. Детектор локализует все видимые явления, которые наблюдались до сих пор, и при необходимости корректирует трекер.

MEDIANFLOW Tracker. Отслеживание за объектом происходит путём измерения расхождения между прямой и обратной траекторией (направлением) движения.

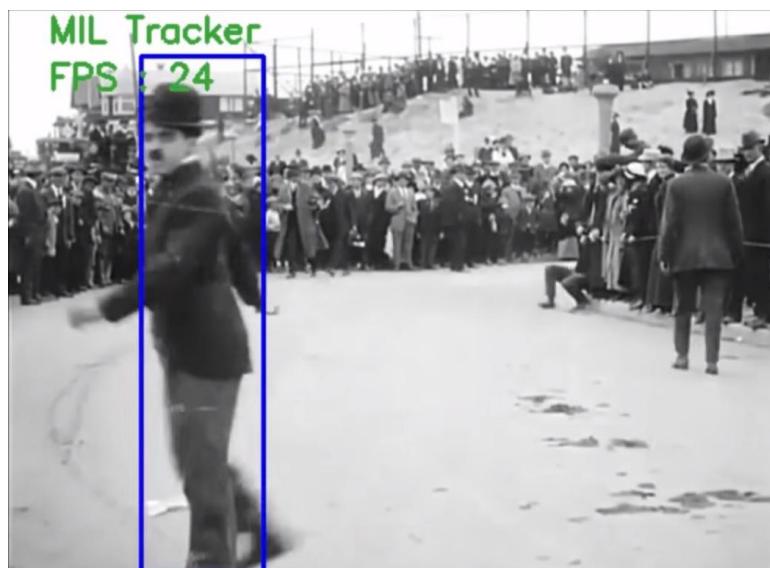
Tracking API	Назначение

cv::TrackerBoosting::create();	Создание выбранного трекера. Возвращает cv::Ptr<cv::Tracker>
.init(cv::Mat, cv::Rect2d); ->init(cv::Mat, cv::Rect2d);	Метод трекера, отвечающий за инициализацию. Принимает матрицу кадра, а также прямоугольник, содержащий первоначальное местоположение объекта.
. update(cv::Mat, cv::Rect2d); ->update(cv::Mat, cv::Rect2d);	Обновление результатов отслеживания, изменяет переданный по ссылке прямоугольник cv::Rect2d, задавая новое месторасположение объекта. Возвращает true/false в зависимости от возможности трекера определять присутствие объекта в кадре.
cv::Ptr<cv::Tracker> tracker;	Создание пока еще пустого указателя на трекер.

Задания

1. Напишите программу, демонстрирующую работу всех 5 трекеров (по отдельности). Для этого вы можете использовать шаблон.
2. Изучите работу каждого из трекеров и выделите основные особенности.
3. Сравните работу разных трекеров. Какой по вашему мнению самый оптимальный? Почему?
4. (*) Выводите траекторию движения объекта. (След движения)

Примеры работы программы





Шаблон программы

```
#include <opencv2/opencv.hpp>
#include <opencv2/tracking.hpp>
#include <opencv2/core/ocl.hpp>

#pragma comment(lib, "opencv_aruco400d.lib")
#pragma comment(lib, "opencv_bgsegm400d.lib")
#pragma comment(lib, "opencv_bioinspired400d.lib")
#pragma comment(lib, "opencv_calib3d400d.lib")
#pragma comment(lib, "opencv_ccalib400d.lib")
#pragma comment(lib, "opencv_core400d.lib")
#pragma comment(lib, "opencv_datasets400d.lib")
#pragma comment(lib, "opencv_dnn_objdetect400d.lib")
#pragma comment(lib, "opencv_dnn400d.lib")
#pragma comment(lib, "opencv_dpm400d.lib")
#pragma comment(lib, "opencv_face400d.lib")
#pragma comment(lib, "opencv_features2d400d.lib")
#pragma comment(lib, "opencv_flann400d.lib")
#pragma comment(lib, "opencv_fuzzy400d.lib")
#pragma comment(lib, "opencv_hfs400d.lib")
#pragma comment(lib, "opencv_highgui400d.lib")
#pragma comment(lib, "opencv_img_hash400d.lib")
#pragma comment(lib, "opencv_imgcodecs400d.lib")
#pragma comment(lib, "opencv_imgproc400d.lib")
#pragma comment(lib, "opencv_line_descriptor400d.lib")
#pragma comment(lib, "opencv_ml400d.lib")
#pragma comment(lib, "opencv_objdetect400d.lib")
#pragma comment(lib, "opencv_optflow400d.lib")
#pragma comment(lib, "opencv_phase_unwrapping400d.lib")
#pragma comment(lib, "opencv_photo400d.lib")
#pragma comment(lib, "opencv_plot400d.lib")
#pragma comment(lib, "opencv_reg400d.lib")
#pragma comment(lib, "opencv_rgbd400d.lib")
```

```

#pragma comment(lib, "opencv_saliency400d.lib")
#pragma comment(lib, "opencv_shape400d.lib")
#pragma comment(lib, "opencv_stereo400d.lib")
#pragma comment(lib, "opencv_stitching400d.lib")
#pragma comment(lib, "opencv_structured_light400d.lib")
#pragma comment(lib, "opencv_superres400d.lib")
#pragma comment(lib, "opencv_surface_matching400d.lib")
#pragma comment(lib, "opencv_text400d.lib")
#pragma comment(lib, "opencv_tracking400d.lib")
#pragma comment(lib, "opencv_video400d.lib")
#pragma comment(lib, "opencv_videoio400d.lib")
#pragma comment(lib, "opencv_videostab400d.lib")
#pragma comment(lib, "opencv_xfeatures2d400d.lib")
#pragma comment(lib, "opencv_ximgproc400d.lib")
#pragma comment(lib, "opencv_xobjdetect400d.lib")
#pragma comment(lib, "opencv_xphoto400d.lib")

using namespace cv;
using namespace std;

// Конвертация к string
#define SSTR( x ) static_cast< std::ostringstream & >( \
( std::ostringstream() << std::dec << x ) ).str()

int main(int argc, char **argv)
{
    // Список трекеров, представленных в OpenCV 3.0
    string trackerTypes[6] = { "BOOSTING", "MIL", "KCF", "TLD", "MEDIANFLOW" };
};

string trackerType = trackerTypes[0];

// Создайте пустой указатель на Tracker

if (trackerType == "BOOSTING") {
    // Верните указатель на созданный трекер
}
if (trackerType == "MIL") {
}
if (trackerType == "KCF") {
}
if (trackerType == "TLD") {
}
if (trackerType == "MEDIANFLOW") {

}

VideoCapture video(0);
// Выход, если не удалось распознать вебкамеру
if (video.isOpened() == false)
{
    cout << "Cannot open the video camera\n";
    cin.get(); // wait for any key press
    return -1;
}

```

```

//Здесь считывается несколько кадров, для того,
//чтобы избежать использование первого - засвеченного кадра из вебкамеры
//При работе с видеофайлом этого делать не нужно.

Mat frame;
video.read(frame);
video.read(frame);
video.read(frame);
video.read(frame);
video.read(frame);
bool ok = video.read(frame);

// Первичное выделение области, на которой находится объект
Rect2d bbox = selectROI(frame, false);

// Рисуем выделенную область на кадре
rectangle(frame, bbox, Scalar(255, 0, 0), 2, 1);
imshow("Tracking", frame);

//Инициализируйте трекер нужным кадром и выделенной областью

while (video.read(frame))
{
    // Start timer
    double timer = (double)getTickCount();

    // Обновите результат отлеживания
    bool ok;
    // Высчитываем FPS
    float fps = getTickFrequency() / ((double)getTickCount() - timer);

    if (ok)
    {
        // Объект найден, обрисовываем его прямоугольником
        rectangle(frame, bbox, Scalar(255, 0, 0), 2, 1);
    }
    else
    {
        // Объект не найден, выводим соответствующую строку
        putText(frame, "Tracking failure detected", Point(100, 80),
        FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0, 0, 255), 2);
    }
    // Выводим название используемого трекера
    putText(frame, trackerType + " Tracker", Point(100, 20),
    FONT_HERSHEY_SIMPLEX, 0.75, Scalar(50, 170, 50), 2);
    // Выводим FPS
    putText(frame, "FPS : " + SSTR(int(fps)), Point(100, 50),
    FONT_HERSHEY_SIMPLEX, 0.75, Scalar(50, 170, 50), 2);
    imshow("Tracking", frame);
    // Выход по нажатию клавиши ESC
    int k = waitKey(1);
    if (k == 27)
    {
        break;
    }
}

```

Лабораторная работа №13

"Паралельная обработка видео. Параллельные потоки"

Программные средства обработки потоков (flows) информации с фотоприемников имеют ряд особенностей, что связано, прежде всего, с двумя факторами: непрерывностью обработки и с ограниченностью производительности вычислительных ресурсов. Первый фактор определяет связь времени, которое отводится на обработку, с пространственно-временной структурой сигнала (количеством пикселей в кадре). Второй фактор связан с тем, что в реальных системах ограничены возможности по производительности процессоров, параллельности обработки и объему памяти. Всегда решается вопрос об оптимальности выбора алгоритма обработки при указанных выше ограничениях.

Основными принципами параллелизации обработки данных являются: конвейерность, векторность, суперскаляроность (параллельность выполнения команд в одном ядре процессора) и параллельность выполнения потоков команд в разных ядрах.

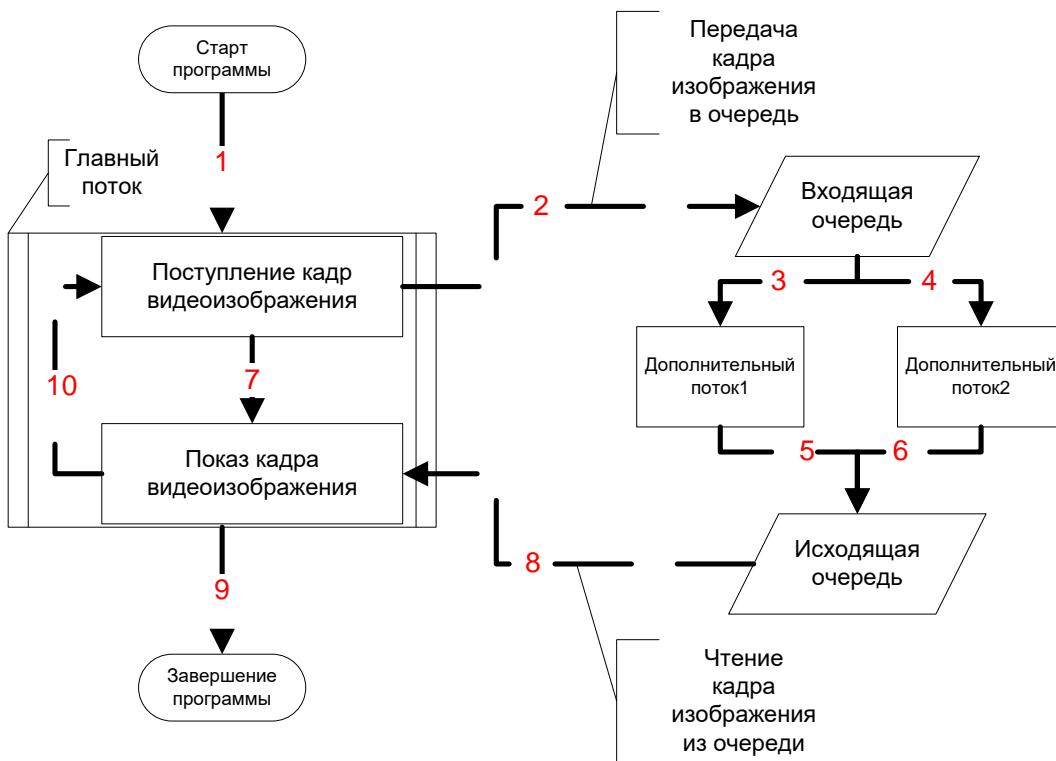
Известны три парадигмы использования вычислительных ядер. Первая парадигма определяется архитектурой применения общей памяти несколькими процессорными ядрами. Примером является архитектура процессоров i7, которые включают, например, 4 ядра, расположенных на общей шине с памятью. Программирование поддерживано с использованием библиотеки OpenMP. Она позволяет организовать множество параллельных потоков команд, выполняемых на вычислительных ядрах компьютера. Функции и директивы библиотеки предоставляют большие возможности по контролю над поведением параллельных программ. Вторая парадигма связана с архитектурой систем с распределенной памятью. Примером являются различные архитектуры суперкомпьютеров. Программирование осуществляется с использованием библиотеки MPI (Message Passing Interface). MPI представляет собой программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. Третья парадигма связана с использованием сопроцессоров. Примером является технология CUDA.

Задание

Реализуйте параллельное выполнение потоков команд при выполнении операций по обработке видео, используя первых две парадигмы построения вычислительных систем.

Сравните время выполнения при параллельном и последовательном реализации операций по обработке видео.

Схема алгоритма



Реализация

При старте главного потока (main) происходит инициализация дополнительных потоков и покадровое поступление видеоизображения с веб камеры (**стрелка 1**):

```
frame = cvQueryFrame( capture ); // Получение кадра
```

Как только кадр будет получен, он передаётся во входящую очередь (**стрелка 2**), из которой будет считан и обработан одним из дополнительных потоков (**стрелки 3 или 4**), в зависимости от того какой поток сейчас свободен.

```
myQueueIn.push(image); // Передача кадра в очередь
```

После обработки кадр поступает в исходящую очередь (**стрелки 5 или 6**).

```
myMapOut.insert(pair<long,IplImage>(currentFrameIndex,ipl)); // Возвращения кадра
```

Во время обработки кадров дополнительными потоками, основной поток проверяет, есть ли в исходящей очереди кадр (**стрелка 7**), если он есть, то он будет считан (**стрелка 8**) и показан.

```
if(!myMapOut.empty()){
```

```

    _lock1.lock(); // Блокировка
    myMapOutIt = myMapOut.begin(); // Считывание кадра
    image = myMapOutIt->second;
    myMapOut.erase(myMapOutIt);
    _lock1.unlock(); // Разблокировка
    cvShowImage("capture", &image); // Показ кадра
}

```

После этого происходит считывание нового кадр (**стрелка 10**).

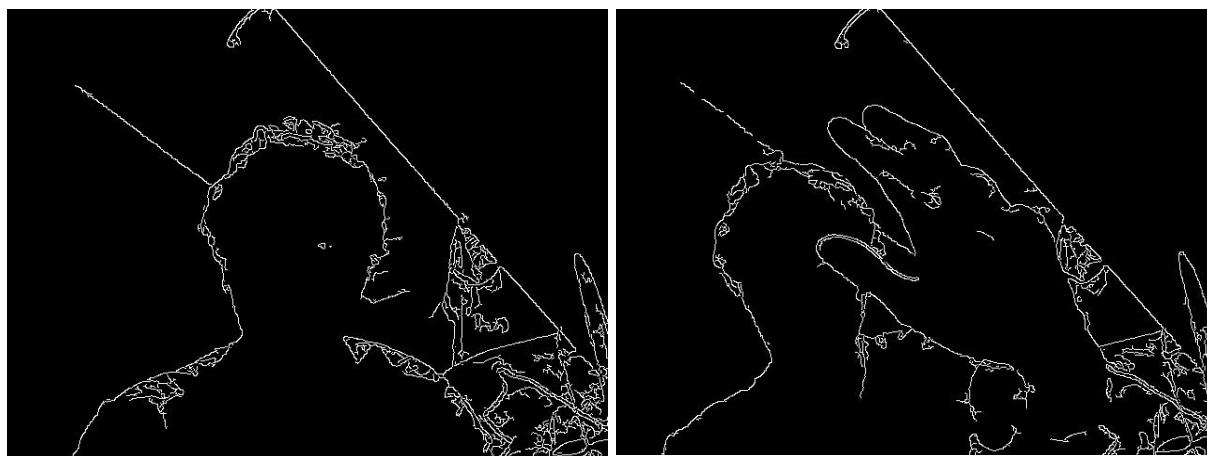
При нажатии определённой клавиши, происходит завершение приложения (**стрелка 9**).

```

char c = cvWaitKey(33); // Кадров в секунду
if(c == 27) { // нажата ESC
    break;
}

```

Результат:



На данных рисунке показан результат наложения фильтра распознавания контуров на каждый кадр параллельно.

```

X nik@nik-Satellite-C850-D3K: ~/workspace/QT/openc... - □ ×
Thread 184538880 Key = 109
Thread 184538880 Key = 110
Thread 192931584 Key = 111
Thread 192931584 Key = 112
Thread 184538880 Key = 113
Thread 184538880 Key = 114
Thread 184538880 Key = 115
Thread 192931584 Key = 116
Thread 192931584 Key = 117
Thread 184538880 Key = 118
Thread 184538880 Key = 119
Thread 192931584 Key = 120
Thread 184538880 Key = 121
Thread 192931584 Key = 122
Thread 184538880 Key = 123
Thread 184538880 Key = 124
Thread 192931584 Key = 125
Thread 184538880 Key = 126
Thread 192931584 Key = 127
Thread 192931584 Key = 128
Thread 184538880 Key = 129
Thread 184538880 Key = 130
Thread 184538880 Key = 131
Thread 184538880 Key = 132

```

Как видно на данном рисунке, при обработке задействовано несколько потоков, причём кадры обрабатываются в правильном порядке независимо от чётности кадра или номера потока.

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>
#include <queue>
#include <thread>
#include <unistd.h>
#include <mutex>
#include <map>

using namespace std;
using cv::Mat;
using cv::Size;
using std::thread;
using std::mutex;
using std::map;

queue<IplImage> myQueueIn;
map<long, IplImage>::iterator myMapOutIt;
map<long, IplImage> myMapOut;

std::recursive_mutex _lock;
std::recursive_mutex _lock1;
long currentFrame = 0;

//Настройка задержки
typedef chrono::duration<int, ratio<1,1000> > ms; //defines a ms
ms xmillisecs (10);

void filters(int parity){

    IplImage ipl;
    IplImage *edges;
    IplImage *cframe;
    long currentIndex = 0;
    bool x = 0;

    for(;;){

        _lock.lock();           // Блокировка
        if(!myQueueIn.empty()){

            ipl = myQueueIn.front();      // Запоминание текущего изображения
```

```

myQueueIn.pop();           // Удаление первого значения
++currentFrame;
currentFrameIndex = currentFrame; // Запоминание текущего индекса
printf("Thread %d Key = %d\n", this_thread::get_id(), currentFrameIndex);
x = 1;
}
_lock.unlock();           // Разблокировка

if(x){
    cframe = &ipl;
    edges = cvCreateImage( cvGetSize(cframe), IPL_DEPTH_8U, 1 );
    cvCvtColor(cframe, edges, CV_RGB2GRAY);
    cvCanny(edges, edges, 10, 100, 3);
    ipl = *edges;
    // Запись текущего значения в map
    _lock1.lock();           // Блокировка
    myMapOut.insert(pair<long,IplImage>(currentFrameIndex,ipl));
    _lock1.unlock();         // Разблокировка
    x = 0;
}
this_thread::sleep_for(xmillisecs);
}

int main(int argc, char* argv[])
{
    CvCapture* capture = cvCreateCameraCapture(CV_CAP_ANY);
    // Получаем любую подключённую камеру
    assert( capture );
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 640); // 1280;
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 480); // 960;

    // Узнаем ширину и высоту кадра
    double width = cvGetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH);
    double height = cvGetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT);
    printf("[i] %.0f x %.0f\n", width, height );

    IplImage* frame=0;

    cvNamedWindow("capture", CV_WINDOW_AUTOSIZE);

    printf("[i] press Enter for capture image and Esc for quit!\n\n");

    int counter=0;
    char filename[512];
    int x = 300;

    // Потоки для фильтрации
    std::thread an_even(filters,0);
}

```

```

an_even.detach();
std::thread odd(filters,1);
odd.detach();

IplImage image;

while(true){
    // Получаем кадр
    frame = cvQueryFrame( capture );

    // Накладываем фильтр
    image = *frame;
    _lock.lock();
    myQueueIn.push(image);
    _lock.unlock();

    if(!myMapOut.empty()){

        _lock1.lock(); // Блокировка
        myMapOutIt = myMapOut.begin();
        image = myMapOutIt->second;
        myMapOut.erase(myMapOutIt);
        _lock1.unlock(); // Разблокировка

        //      printf("SizeMyMapOut = %d\n",myMapOut.size());
        cvShowImage("capture", &image);
        //printf("Size = %d\n", myMapOut.size());
    }

    char c = cvWaitKey(33);
    if (c == 27) { // нажата ESC
        break;
    }
    else if(c == 13) { // Enter
        // сохраняем кадр в файл
        sprintf(filename, "Image%d.jpg", counter);
        printf("[i] capture... %s\n", filename);
        cvSaveImage(filename, &image);
        counter++;
    }
}
// освобождаем ресурсы
cvReleaseCapture( &capture );
cvDestroyWindow("capture");
return 0;
}

```

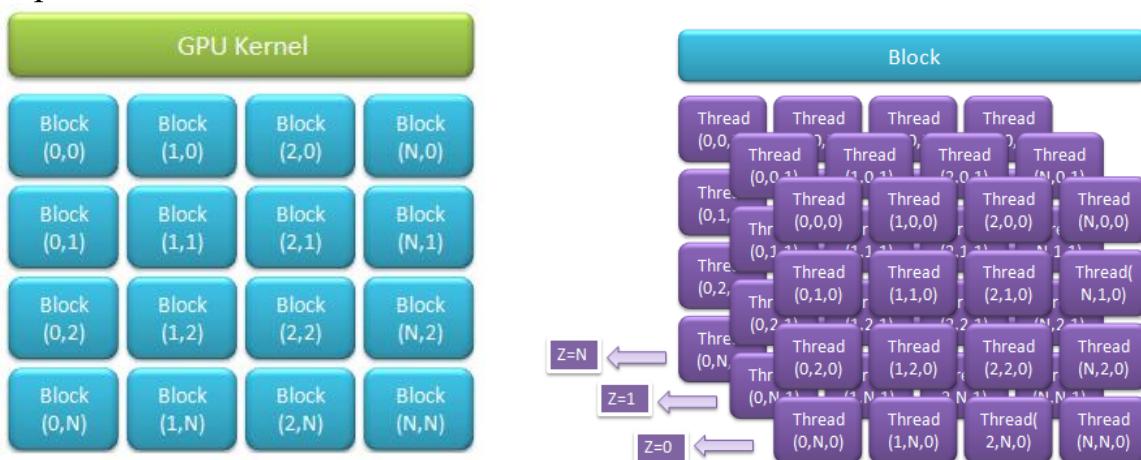
Лабораторная работа №14

"Параллельная обработка видео. CUDA "

В настоящее время в качестве сопроцессоров, обеспечивающих быструю обработку введенного в ЭВМ изображения, используются графические процессоры GPU (graphics processing unit). Графический процессор является устройством компьютера, которое выполняет операции по выводу изображения на монитор (графический рендеринг). Однако за счет имеющейся высокой резервной вычислительной мощности GPU можно с успехом использовать для задач обработки потока изображений. Высокая вычислительная мощность GPU достигается за счет параллельной обработки пикселей изображений на многих вычислительных ядрах, организованных в виде матрицы.

Наибольшими вычислительными мощностями обладают GPU компании Nvidia, которые разрабатываются и поддерживаются в рамках технологии CUDA. Технология CUDA появилась в 2006 году и представляет из себя программно-аппаратный комплекс, позволяющий эффективно писать и исполнять программы под графические адаптеры. Графический процессор организует аппаратную многопоточность, что позволяет задействовать все ресурсы графического процессора.

GPU CUDA имеет следующую вычислительную модель. Верхний уровень ядра GPU состоит из блоков, которые группируются в сетку или грид (grid) размерностью $N_1 * N_2 * N_3$. Любой блок в свою очередь состоит из нитей (threads), которые являются непосредственными исполнителями вычислений. Нити в блоке сформированы в виде трехмерного массива.



Программная часть содержит в себе всё необходимое для разработки программы: расширения языка C, компилятор, API для работы с графическими адаптерами и набор библиотек. CUDA SDK позволяет

программистам реализовывать на специальном упрощённом диалекте языка программирования Си алгоритмы, выполнимые на графических процессорах NVIDIA.

Задание

Реализуйте выполнение операций по обработке видео, используя возможности CUDA.

Сравните время выполнения при параллельном и последовательном реализации операций по обработке видео.

Возможная реализация CUDA

Для разработки можно использовать функцию удаления фона Mog2, встроенную в библиотеку OpenCV. Данная функция была выполнена в 2х режимах с применением технологии распараллеливания CUDA и последовательное выполнение на центральном процессоре.

Код, описывающий ее для последовательной версии программы, выглядит следующим образом:

```
AAtime = getTickCount();
pMOG2->operator()(resizeF, fgMaskMOG2);
BBtime = getTickCount();
```

Для параллельной версии необходимо сначала задать параметры метода Mog2

```
gpu::MOG2_GPU pMOG2_g(30);
pMOG2_g.history = 3000; //300;
pMOG2_g.varThreshold = 64; //128; //64; //32;//;
pMOG2_g.bShadowDetection = true;
Mat Mog_Mask;
gpu::GpuMat Mog_Mask_g;
```

а затем сам код функции:

```
AAtime = getTickCount();
pMOG2_g.operator()(o_frame_gpu, Mog_Mask_g, -1);
Mog_Mask_g.download(Mog_Mask);
BBtime = getTickCount();
```

Для измерения времени выполнения программ был использован таймер getTickCount().

Тестирование

Последовательная программа

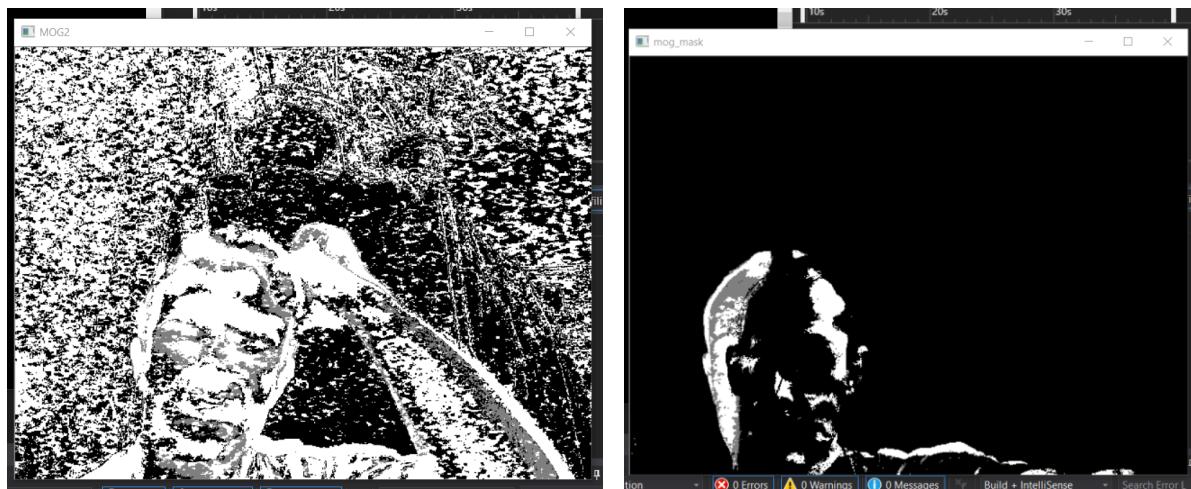
```
C:\Users\User\Desktop>
cpu 0.0364
cpu 0.0319
cpu 0.0336
cpu 0.0342
cpu 0.0308
cpu 0.0304
cpu 0.0312
cpu 0.0286
cpu 0.0307
cpu 0.0318
cpu 0.0334
cpu 0.0319
cpu 0.0334
cpu 0.0310
cpu 0.0316
cpu 0.0318
cpu 0.0305
cpu 0.0353
cpu 0.0324
cpu 0.0335
cpu 0.0404
cpu 0.0468
cpu 0.0362
cpu 0.0446
cpu 0.0359
cpu 0.0446
cpu 0.0513
cpu 0.0443
average 0.0424
Для продолжения нажм
```

Параллельная программа

```
C:\Users\User\De>
gpu 0.0035
gpu 0.0035
gpu 0.0034
gpu 0.0035
gpu 0.0035
gpu 0.0036
gpu 0.0035
gpu 0.0036
gpu 0.0036
gpu 0.0035
gpu 0.0035
gpu 0.0035
gpu 0.0035
gpu 0.0036
gpu 0.0037
gpu 0.0035
gpu 0.0035
gpu 0.0035
gpu 0.0036
gpu 0.0036
gpu 0.0038
gpu 0.0037
gpu 0.0037
gpu 0.0036
average 0.0039
Для продолжения нажм
```

Из результата выполнения программ видно, что среднее время выполнения обработки видео уменьшилось почти в 11 раз с применением технологии cuda по сравнению с выполнением на центральном процессоре.

Внешний вид программы



Программный код

Реализация на GPU

```
#include < time.h>
#include <opencv2\opencv.hpp>
#include <opencv2\gpu\gpu.hpp>
#include <string>
#include <stdio.h>

#define RWIDTH 800
#define RHEIGHT 600

using namespace std;
using namespace cv;

int main()
{
    /////////////////////////////////
    gpu::MOG2_GPU pMOG2_g(30);
    pMOG2_g.history = 3000; //300;
    pMOG2_g.varThreshold = 64; //128; //64; //32;;
    pMOG2_g.bShadowDetection = true;
    Mat Mog_Mask;
    gpu::GpuMat Mog_Mask_g;
    /////////////////////////////////
    VideoCapture cap(0); //0);
    Mat o_frame;
    gpu::GpuMat o_frame_gpu;
    /////////////////////////////////

    cap >> o_frame;
    if (o_frame.empty())
        return 0;

    /////////////////////////////////

    unsigned long AAtime = 0, BBtime = 0;
    float sum = 0;
    int i = 0;
```

```

//Mat rFrame;
Mat showMat_r;
namedWindow("origin");
namedWindow("mog_mask");

while (1)
{
    /////////////////////////////////
    cap >> o_frame;
    if (o_frame.empty())
        return 0;
    o_frame_gpu.upload(o_frame);
    AAtime = getTickCount();

    //
    pMOG2_g.operator()(o_frame_gpu, Mog_Mask_g, -1);
    //
    Mog_Mask_g.download(Mog_Mask);
    BBtime = getTickCount();
    float pt = (BBtime - AAtime) / getTickFrequency();
    //float fpt = 1 / pt;
    sum += pt;
    i++;
    printf("gpu %.4lf \n", pt);
    o_frame_gpu.download(showMat_r);
    imshow("origin", showMat_r);
    imshow("mog_mask", Mog_Mask);

    /////////////////////////////////

    if (waitKey(10) > 0)
        break;
}
printf("average %.4lf \n", sum / i);
system("Pause");
}

```

Реализация на CPU

```

// OpenCV_CPU.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include < time.h>
#include <opencv2\opencv.hpp>
#include <opencv2\gpu\gpu.hpp>
#include < string>
#include < stdio.h>

using namespace std;
using namespace cv;

int main()
{

```

```

//global variables
Mat frame; //current frame
Mat resizeF;
Mat fgMaskMOG; //fg mask generated by MOG method
Mat fgMaskMOG2; //fg mask fg mask generated by MOG2 method
Mat fgMaskGMG; //fg mask fg mask generated by MOG2 method

Ptr< BackgroundSubtractor> pMOG; //MOG Background subtractor
Ptr< BackgroundSubtractor> pMOG2; //MOG2 Background subtractor
Ptr< BackgroundSubtractorGMG> pGMG; //MOG2 Background subtractor

//pMOG = new BackgroundSubtractorMOG();
pMOG2 = new BackgroundSubtractorMOG2();
//pGMG = new BackgroundSubtractorGMG();

unsigned long AAtime = 0, BBtime = 0;
float sum = 0;
int i = 0;

//char fileName[100] = "C:\\POSCO\\video\\cctv 2.mov"; //Gate1_175_p1.avi";
//mm2.avi"; //"_p1.avi";
VideoCapture stream1(0); //0 is the id of video device.0 if you have only one
camera

Mat element = getStructuringElement(MORPH_RECT, Size(3, 3), Point(1, 1));

//unconditional loop
while (true) {
    Mat cameraFrame;
    if (!(stream1.read(frame))) //get one frame form video
        break;

    resize(frame, resizeF, Size(frame.size().width , frame.size().height ));

    AAtime = getTickCount();
    pMOG2->operator()(resizeF, fgMaskMOG2);
    BBtime = getTickCount();
    float pt = (BBtime - AAtime) / getTickFrequency();
    sum = sum + pt;
    i++;
    printf("cpu %.4lf\n", pt);
    imshow("Origin", resizeF);
    //imshow("MOG", fgMaskMOG);
    imshow("MOG2", fgMaskMOG2);
    //imshow("GMG", fgMaskGMG);
    if (waitKey(27) >= 0)
        break;
}
printf("average %.4lf \n", sum / i);
system("Pause");
}

```

Лабораторная работа №15

"Стабилизация видеозаписи с помощью ffmpeg и OpenCV"

OpenCV (Open Source Computer Vision Library) — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Она была разработана Intel, а сейчас поддерживается Willow Garage Inc. и Itseez Ltd. OpenCV распространяется под лицензией BSD, а значит, бесплатна как для учебных, так и для коммерческих целей. Реализована на C++, C, Python и Java, поддерживает Windows, Linux, Mac OS, iOS и Android. OpenCV имеет модульную структуру, что упрощает ее использование. Модуль стабилизации видео предоставляет набор функций и классов для анализа движения и обработки видео.

FFmpeg — набор свободных библиотек с открытым исходным кодом, которые позволяют записывать, конвертировать и передавать цифровые аудио- и видеозаписи в различных форматах. Он включает libavcodec — библиотеку кодирования и декодирования аудио и видео и libavformat — библиотеку мультиплексирования и демультиплексирования в медиаконтейнер.

Задание

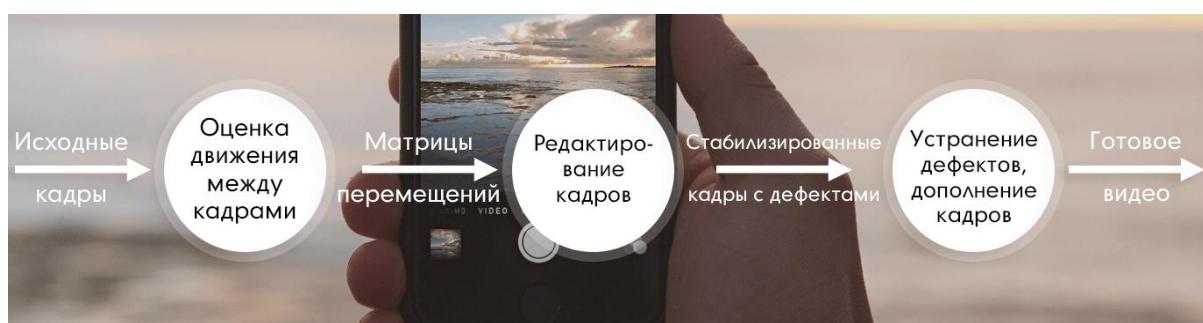
Реализуйте выполнение операций по стабилизации видеозаписи.

Оцените производительность программной реализации использованных вами методов.

Попробуйте понять производительность за счет использования методов параллелизации.

Алгоритм

Алгоритм стабилизации видео обычно включает в себя следующие этапы:



Рассмотрим процесс подробнее.

- На первом этапе стабилизации видео выполняется оценка смещения между соседними кадрами. В результате мы получаем массив матриц 3×3 , описывающих смещение между двумя смежными кадрами. Так как оценка смещения является первым этапом стабилизации, её точность имеет решающее значение.
- На следующем этапе генерируется новая последовательность кадров на основе рассчитанных смещений, после чего для повышения качества видео и устранения дефектов производится дополнительная обработка (сглаживание, устранение размытости, граничная экстраполяция, проч.). Существуют подходы устранения дефектов, учитывающие модель движения камеры, которые при правильном выборе модели позволяют получить более качественный результат. Заполнение недостающих участков, появляющихся на кадрах после трансформации, называется *video completion*.
- На видео также может присутствовать эффект размытости, основной источник ухудшения качества изображения, который вызван движением объекта в кадре во время съемки. Поэтому методы стабилизации также содержат механизм устранения размытости (*deblurring*).

Алгоритм RANSAC

Для решения задачи сопоставления изображений и оценки фундаментальной матрицы для определения параметров расположения камеры используется алгоритм RANSAC.

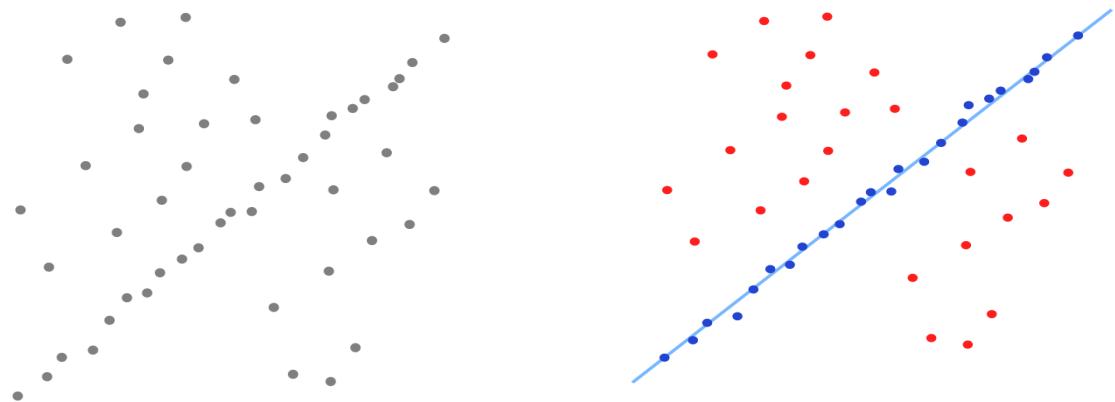
RANSAC (RANdom SAmple Consensus) — стабильный метод оценки параметров модели на основе случайных выборок. Схема RANSAC устойчива к зашумлённости исходных данных.

Пример работы алгоритма RANSAC

Рассмотрим простейший пример работы алгоритма: вписывание прямой в 2D точки. Принимая тот факт, что среди данных есть выбросы, оценка параметров стандартным способом, например, методом наименьших квадратов, приведёт к тому, что будет вычислена неверная модель, так как модель строится на основе всех точек. Метод RANSAC берёт за основу только две точки необходимые для построения прямой и с их помощью строит модель, после чего проверяет, какое количество точек соответствует модели, используя функцию оценки с заданным порогом.

На первом рисунке: Набор данных, в который надо вписать прямую - выбросы присутствуют в большом количестве.

На втором рисунке: Предложенная алгоритмом RANSAC прямая - выбросы не влияют на результат.



Оценка алгоритма RANSAC

Преимуществом алгоритма RANSAC является его способность дать надёжную оценку параметров модели, то есть возможность оценить параметры модели с высокой точностью, даже если в исходном наборе данных присутствует значительное количество выбросов.

Одним из недостатков метода RANSAC является отсутствие верхней границы времени, необходимого для вычисления параметров модели. Если использовать в качестве некоторой границы времени максимальное число итераций, полученное решение может быть не оптимальным, а также существует очень малая вероятность, что ни одна модель не будет соответствовать исходным данным. Точная модель может быть определена с некоторой вероятностью, которая становится больше, чем больше итераций, которые используются. Ещё одним недостатком метода RANSAC является то, что для выполнения алгоритма необходимо задать конкретное пороговое значение. Наконец методом RANSAC можно определить только одну модель для определённого набора данных. Как и для любого подхода, предназначенного для одной модели, существует следующая проблема: когда в исходных данных присутствуют две (или более) модели, RANSAC может не найти ни одну.

Для оценки производительности использованных методов было проведено много экспериментов на коротких видео с различными параметрами. В итоге, скорость вычислений составила всего 2 FPS для видеоролика с разрешением 640×480 , 30 FPS. Таких результатов явно недостаточно для стабилизации видео в реальном времени. К тому же скорость обработки заметно падает с увеличением разрешения видео.

Скорость стабилизации видео также зависит от характеристик сцены: количества деталей, освещённости. Узким местом здесь является первый этап – оценка движения. К примеру, можно увеличить скорость обработки до 40 FPS при тех же настройках, используя уже вычисленные матрицы перемещений. Ускорения этапа оценки движения для стабилизации видео в реальном времени можно добиться с использованием GPU и OpenCL.

Программа с использованием библиотеки ffmpeg:

```
public class Application {

    private static final String WORK_DIR = "e:\\\\redactor\\\\";

    public static void main(String[] args) {
        Redactor redactor = new RedactorImpl();
        VideoStabilizer videoStabilizer = new VideoStabilizer(redactor);

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter input filepath:");
        String input = sc.next();
        System.out.println("Enter output filepath:");
        String output = sc.next();
        sc.close();

        ShellExecuter executor = new ShellExecuter();
        try {
            executor.executeAndWait(Arrays.asList("python",
                "-i " + WORK_DIR + input,
                "-o " + WORK_DIR + output),
                new File("stab.py"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("Start stabilize file:" + input + ", to file:" + output);
        try {
            videoStabilizer.stabilize(new File(WORK_DIR + input), new
File(WORK_DIR + output));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

public class VideoStabilizer {

    private static final Logger logger =
Logger.getLogger(VideoStabilizer.class);

    private final Redactor redactor;

    public VideoStabilizer(Redactor redactor) {
        this.redactor = redactor;
    }
}
```

```

    public void stabilize(File inputFile, File outputFile) throws
IOException {

    System.out.println("Run stab: ");
    FileUtils fileUtils = new FileUtils();
    fileUtils.withTempDir( tempDir -> {
        try {
            redactor.stabilize(inputFile, outputFile, tempDir.toFile());
        } catch (IOException e) {
            logger.warn(e, e);
        }
    });
}

public class RedactorImpl implements Redactor {

    public static final String DEFAULT_CMD_EXECUTE = "c:\\\\ffmpeg\\\\bin\\\\ffmpeg";

    private static final Logger logger = Logger.getLogger(RedactorImpl.class);
    private final ShellExecuter shellExecuter;

    private final String ffmpegCmd;

    public RedactorImpl() {
        this("");
    }

    public RedactorImpl(String ffmpegPath) {
        this.shellExecuter = new ShellExecuter();
        this.ffmpegCmd = ffmpegPath + DEFAULT_CMD_EXECUTE;
    }

    public void stabilize(final File inputFile, final File outputFile, final
File workingDir) throws IOException {
        List<String> params = new ArrayList<>();

        params.add(ffmpegCmd);
        params.add("-i");
        params.add(inputFile.getAbsolutePath());
        params.add("-vf");
        params.add("vidstabdetect=shakiness=5:show=1");
        params.add("-y");
        params.add("-strict");
        params.add("-2");
        params.add("temp.mp4");

        logCommand(params);

        shellExecuter.executeAndWait(params, workingDir);

        params = new ArrayList<>();

        params.add(ffmpegCmd);
        params.add("-i");
        params.add(inputFile.getAbsolutePath());
        params.add("-vf");
        params.add("vidstabtransform");
        params.add("-y");
    }
}

```

```

params.add("-strict");
params.add("-2");
params.add(outputFile.getAbsolutePath());

logCommand(params);

shellExecuter.executeAndWait(params, workingDir);
}

```

Код скрипта openCV:

```

def getshiftmatrix( (dx,dy) ):
    return array([[ 1.,  0,      dx],
                  [ 0,  1.,     dy]])

def shiftimg(im2,shift):
    tr1=getshiftmatrix(shift)
    return cv2.warpAffine(im2,tr1,tuple(reversed(im2.shape[:2])) )
im2r= shiftimg(im2,tuple(-array(shift)))

arshifts=[]

im1= cv2.imread(sampledata[0]) # base frame
im1g = preprocess(im1)
kp1, desc1 = detector.detectAndCompute(im1g, None)

imgprev=cv2.imread(basepath4orig+sampledata[0]) #base original frame

for i,x in enumerate(sampledata):
    print x,
    im2g = preprocess(cv2.imread(x))
    kp2, desc2 = detector.detectAndCompute(im2g, None)

    raw_matches = matcher.knnMatch(desc1, trainDescriptors = desc2, k =
2) #2
    p1, p2, kp_pairs = filter_matches(kp1, kp2, raw_matches)
    dp=p2-p1

    if len(dp)<=0: shift=0,0
    else: dx,dy=np.median(dp[:,0]),np.median(dp[:,1])
    print dx,dy

    #process original frame
    imgr= shiftimg(cv2.imread(basepath4orig+x), (-dx,-dy))

    if -dy>0: imgr[:int(ceil(abs(dy))),:,:] =
imgprev[:int(ceil(abs(dy))),:,:]
        if -dy<0: imgr[-int(ceil(abs(dy))):,:,:] = imgprev[-
int(ceil(abs(dy))):,:,:]
    if -dx>0: imgr[:, :int(ceil(abs(dx))),:] =
imgprev[:, :int(ceil(abs(dx))),:]
        if -dx<0: imgr[:, -int(ceil(abs(dx))):,:] = imgprev[:, -
int(ceil(abs(dx))):,:]

    imgprev=imgr

    cv2.imwrite('shifted_'+x+'.JPG',imgr)

```

Лабораторная работа №16 "Использование каскада Хаара для распознавания"

Задание

Напишите программу с использованием каскада Хаара (дополните представленную программу) распознавания всех карт при помощи веб камеры.

Можно добавить распознавание мастей.

Использованные функции медийной библиотеки

В качестве медийной библиотеки была выбрана такая библиотека, как OpenCV. OpenCV позволяет покадрово обрабатывать видеофайлы и запись с веб-камеры. Также в этой библиотеке представлена возможность работы с каскадом Хаара.

Функция и переданные ей параметры	Назначение
<code>cv::putText(frame, "It is six!", pt1, FONT_HERSHEY_COMPLEX, 0.8, cvScalar(0, 0, 0), 1, CV_AA);</code>	Рисует текст
<code>cv::rectangle(frame, pt1, pt2, cv::Scalar(0, 0, 255), 4, 8, 0);</code>	Рисует прямоугольник
<code>cvtColor(frame, frame_gray, COLOR_BGR2GRAY);</code>	Применяет адаптивный порог к массиву.
<code>cv::equalizeHist(frame_gray, frame_gray);</code>	Вычисляет гистограмму набора массивов.
<code>cv::destroyAllWindows();</code>	Удаление всех открытых OpenCV окон.
<code>cv::imshow(window_name, frame);</code>	Показ кадра, переданного вторым параметром, в окне с называнием, указанным в первом параметре.

Обучение каскада Хаара

Для обучение каскада Хаара нам потребуются фотографии объекта в реальной среде обитания(положительные), чем более похожа выборка будет на то, что мы будем распознавать, тем лучше будет результат, к примеру если обучать распознавание лиц по фотографиям людей из студии, то уровень распознавания на улице будет значительно ниже, на это влияет многое, тени, освещение и.т.п. Далее нам потребуются выборка отрицательных фотографий, то есть тех фотографий где нет искомого объекта, также стоит отметить, что на конечный результат влияет среда где были сделаны отрицательные фотографии, поэтому отрицательные фотографии должны быть сделаны там же, где были сделаны положительные. Стоит заметить что желательно, чтобы отрицательные фотографии были черно-белыми, а их формат должен быть jpg, положительные фотографии должны быть цветными и желательно в формате bmp. От кол-во положительных и отрицательных фотографий, чем больше фотографий, тем точнее будет происходить распознавание.

Инструкция

- 1) В папку negative помещаем негативные фотографии
- 2) Заходим в папку positive и в папку rawdata помещаем положительные фотографии
- 3) Далее в папке negative запускаем create_list.bat
- 4) В папке positive запускаем objectmaker.exe, откроется программа в которой на фотографии требуется выделить искомый объект, после выделение нажимаем Space для добавления его, Enter для сохранение и перехода к следующему, если на фотографии есть несколько объектов, выделяем их по очереди и добавляем при помощи Space.
- 5) Далее запускаем samples_creation.bat
- 6) После выполнения samples_creation.bat требуется открыть haarTraning.bat в текстовом документе и поменять -pos(кол-во положительных фотографий) и -nneg(Кол-во отрицательных фотографий) на свои значения.
- 7) После чего запускаем convert.bat, который по завершению работы создаст нам наш xml файл.

Программа

```
1. #include "opencv2/objdetect.hpp"
2. #include "opencv2/highgui.hpp"
3. #include "opencv2/imgproc.hpp"
4.
5. #include <windows.h>
6. #include <iostream>
7. #include <stdio.h>
```

```

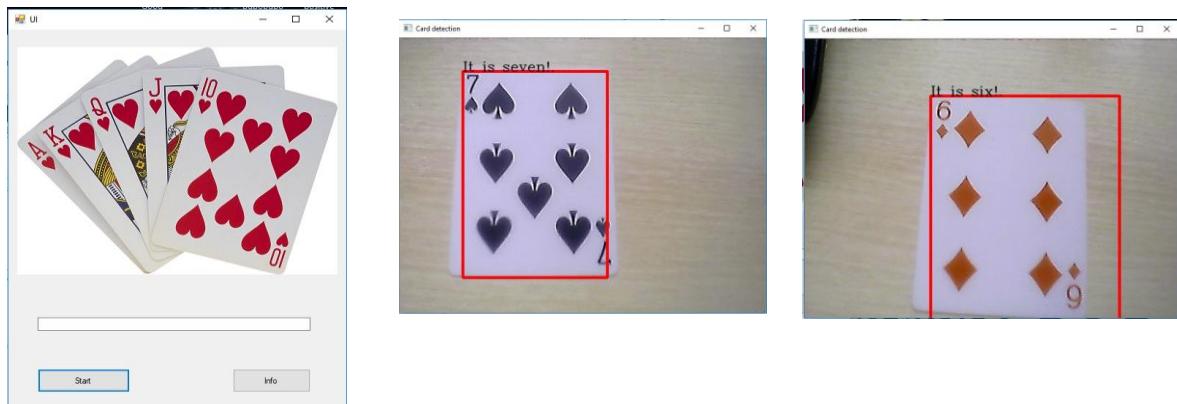
8. #include <ctime>
9. #include <string>
10.
11. using namespace std;
12. using namespace cv;
13.
14. #pragma comment (lib,"opencv_world341d.lib")
15.
16. void detectseven(Mat frame);
17. void detectsix(Mat frame);
18. String seven_cascade_name = "Seven.xml";
19. String six_cascade_name = "Six.xml";
20. CascadeClassifier seven_cascade;
21. CascadeClassifier six_cascade;
22. String window_name = "Card detection";
23. string var;
24.
25. int main(void)
26. {
27.
28.
29.     VideoCapture capture;
30.     Mat frame;
31.
32.     if (!seven_cascade.load(seven_cascade_name)) { printf("--\n");
33.         (!)Error loading card cascade\n"; return -1; };
34.     if (!six_cascade.load(six_cascade_name)) { printf("--\n");
35.         (!)Error loading card cascade\n"; return -1; };
36.     cin >> var;
37.
38.     capture.open(0);
39.     if (!capture.isOpened()) { printf("--\n");
40.         (!)Error opening video capture\n"); return -1; }
41.
42.     while (capture.read(frame))
43.     {
44.         if (frame.empty())
45.         {
46.             printf(" --(!) No captured frame -- Break!");
47.             break;
48.         }
49.
50.         if (var == "Seven") {
51.             detectseven(frame);
52.         }
53.
54.         int c = waitKey(10);
55.         if ((char)c == 27) { break; }
56.     }
57.     return 0;
58. }
59.
60.
61. void detectseven(Mat frame)
62. {
63.     std::vector<Rect> seven;
64.
65.     Mat frame_gray;
66.
67.
68.

```

```

69.     cvtColor(frame, frame_gray, COLOR_BGR2GRAY);
70.     equalizeHist(frame_gray, frame_gray);
71.
72.     seven_cascade.detectMultiScale(frame_gray, seven, 1.1, 2, 0 | CASCADE_SCALE_IMAGE,
73.                                     Size(30, 30));
74.     for (size_t i = 0; i < seven.size(); i++)
75.     {
76.         Point pt1(seven[i].x, seven[i].y);
77.         Point pt2(seven[i].x + seven[i].width * 7, seven[i].y + seven[i].height * 10
78. );
79.         cv::rectangle(frame, pt1, pt2, cv::Scalar(0, 0, 255), 4, 8, 0);
80.         putText(frame, "It is seven!", pt1, FONT_HERSHEY_COMPLEX, 0.8, cvScalar(0,
81. 0, 0, 0), 1, CV_AA);
82.     }
83. }
84.
85. void detectsix(Mat frame)
86. {
87.     std::vector<Rect> six;
88.
89.     Mat frame_gray;
90.
91.
92.     cvtColor(frame, frame_gray, COLOR_BGR2GRAY);
93.     equalizeHist(frame_gray, frame_gray);
94.
95.
96.     six_cascade.detectMultiScale(frame_gray, six, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, S
97. ize(30, 30));
98.
99.     for (size_t i = 0; i < six.size(); i++)
100.    {
101.        Point pt1(six[i].x, six[i].y);
102.        Point pt2(six[i].x + six[i].width * 7, six[i].y + six[i].height * 10
103. );
104.        cv::rectangle(frame, pt1, pt2, cv::Scalar(0, 0, 255), 4, 8, 0);
105.        putText(frame, "It is six!", pt1, FONT_HERSHEY_COMPLEX, 0.8, cvScalar(0,
106. 0, 0, 0), 1, CV_AA);
107.    }
108.
109.    imshow(window_name, frame);
110. }

```



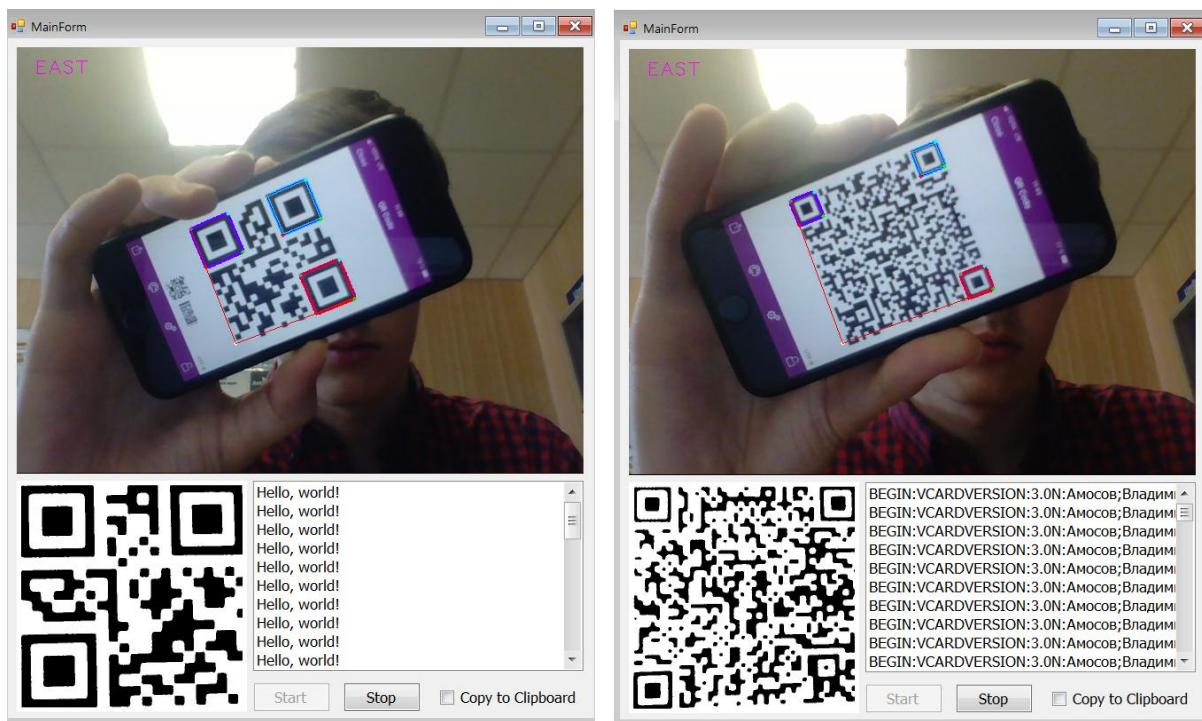
Лабораторная работа №17

"Чтение QR-кодов"

Задание

Напишите программу, которая считывает изображение с веб-камеры, анализирует его, определяет наличие QR-кода, выполняет его выравнивание по перспективе и поворот в пространстве, отображает полученное изображение и считывает его содержание. Используйте шаблон.

Скриншот шаблона



Использованные функции

Функция	Описание
Mat::zeros(int rows, int cols, int type);	Создание пустой матрицы заданного размера
cvtColor(frame, gray, CV_RGB2GRAY)	Конвертирование цветовой гаммы изображения
Mat imread(const String& filename, int flags = IMREAD_COLOR);	Чтение изображения
void cv::imshow(const String & winname, InputArray mat)	Отображает изображение в указанном окне.

void cv::destroyWindow(const String & winname)	Уничтожает указанное окно.
int cv::waitKey(int delay = 0)	Ожидает нажатия клавиши.
Canny();	Применяет обнаружение краев Кэнни.
findContours(edges, contours, hierarchy, cv::RetrievalModes::RETR_TREE, cv::ContourApproximationModes::CHAIN_APPROX_SIMPLE);	Поиск контуров
cv::moments	Создание моментов

Шаблон программы

```
#pragma once

#include <opencv2/opencv.hpp>
#include <vector>
#include <iconv.h>
#include <zbar.h>
#include <msclr\marshal_cppstd.h>

#include "QR-detection.h"

namespace QRReader {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Drawing::Imaging;

    public ref class MainForm : public System::Windows::Forms::Form
    {
    public:
        MainForm(void)
        {
            InitializeComponent();
        }

    protected:
        ~MainForm()
        {
            if (components)
            {
                delete components;
            }
        }

    private: System::Windows::Forms::PictureBox^ webcamBox;
    private: System::Windows::Forms::PictureBox^ qrcodeBox;
    protected:

    protected:
```

```

private: System::Windows::Forms::Timer^ framesTimer;
private: System::Windows::Forms::ListBox^ scannedCodesListBox;

private: System::Windows::Forms::Button^ startButton;
private: System::Windows::Forms::Button^ stopButton;
private: System::Windows::Forms::CheckBox^ copyToClipboardCheckbox;

private: System::ComponentModel::.IContainer^ components;

protected:

private:
    /// <summary>
    /// Required designer variable.
    /// </summary>

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->components = (gcnew System::ComponentModel::Container());
        this->webcamBox = (gcnew System::Windows::Forms::PictureBox());
        this->qrcodeBox = (gcnew System::Windows::Forms::PictureBox());
        this->framesTimer = (gcnew System::Windows::Forms::Timer(this-
>components));
        this->scannedCodesListBox = (gcnew
System::Windows::Forms::ListBox());
        this->startButton = (gcnew System::Windows::Forms::Button());
        this->stopButton = (gcnew System::Windows::Forms::Button());
        this->copyToClipboardCheckbox = (gcnew
System::Windows::Forms::CheckBox());
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>webcamBox))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>qrcodeBox))->BeginInit();
        this->SuspendLayout();
        //
        // webcamBox
        //
        this->webcamBox->BackColor =
System::Drawing::SystemColors::InactiveCaptionText;
        this->webcamBox->Location = System::Drawing::Point(10, 10);
        this->webcamBox->Margin = System::Windows::Forms::Padding(4);
        this->webcamBox->Name = L"webcamBox";
        this->webcamBox->Size = System::Drawing::Size(640, 480);
        this->webcamBox->TabIndex = 0;
        this->webcamBox->TabStop = false;
        this->webcamBox->Click += gcnew System::EventHandler(this,
&MainForm::webcamBox_Click);

```

```

//
// qrcodeBox
//
this->qrcodeBox->BackColor =
System::Drawing::SystemColors::ControlLightLight;
this->qrcodeBox->Location = System::Drawing::Point(10, 498);
this->qrcodeBox->Margin = System::Windows::Forms::Padding(4);
this->qrcodeBox->Name = L"qrcodeBox";
this->qrcodeBox->Size = System::Drawing::Size(260, 260);
this->qrcodeBox->TabIndex = 0;
this->qrcodeBox->TabStop = false;
//
// framesTimer
//
this->framesTimer->Interval = 200;
this->framesTimer->Tick += gcnew System::EventHandler(this,
&MainForm::framesTimer_Tick);
//
// scannedCodesListBox
//
this->scannedCodesListBox->FormattingEnabled = true;
this->scannedCodesListBox->ItemHeight = 21;
this->scannedCodesListBox->Location = System::Drawing::Point(277,
498);
this->scannedCodesListBox->Name = L"scannedCodesListBox";
this->scannedCodesListBox->Size = System::Drawing::Size(373, 214);
this->scannedCodesListBox->TabIndex = 2;
//
// startButton
//
this->startButton->Location = System::Drawing::Point(277, 725);
this->startButton->Name = L"startButton";
this->startButton->Size = System::Drawing::Size(87, 33);
this->startButton->TabIndex = 3;
this->startButton->Text = L"Start";
this->startButton->UseVisualStyleBackColor = true;
this->startButton->Click += gcnew System::EventHandler(this,
&MainForm::startButton_Click);
//
// stopButton
//
this->stopButton->Enabled = false;
this->stopButton->Location = System::Drawing::Point(379, 725);
this->stopButton->Name = L"stopButton";
this->stopButton->Size = System::Drawing::Size(87, 33);
this->stopButton->TabIndex = 3;
this->stopButton->Text = L"Stop";
this->stopButton->UseVisualStyleBackColor = true;
this->stopButton->Click += gcnew System::EventHandler(this,
&MainForm::stopButton_Click);
//
// copyToClipboardCheckbox
//
this->copyToClipboardCheckbox->AutoSize = true;
this->copyToClipboardCheckbox->Location =
System::Drawing::Point(488, 730);
this->copyToClipboardCheckbox->Name = L"copyToClipboardCheckbox";
this->copyToClipboardCheckbox->Size = System::Drawing::Size(162,
25);
this->copyToClipboardCheckbox->TabIndex = 4;

```

```

        this->copyToClipboardCheckbox->Text = L"Copy to Clipboard";
        this->copyToClipboardCheckbox->UseVisualStyleBackColor = true;
        //
        // MainForm
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(9, 21);
        this->AutoSizeMode = System::Windows::Forms::AutoSizeMode::Font;
        this->ClientSize = System::Drawing::Size(662, 765);
        this->Controls->Add(this->copyToClipboardCheckbox);
        this->Controls->Add(this->stopButton);
        this->Controls->Add(this->startButton);
        this->Controls->Add(this->scannedCodesListBox);
        this->Controls->Add(this->qrcodeBox);
        this->Controls->Add(this->webcamBox);
        this->Font = (gcnew System::Drawing::Font(L"Tahoma", 10));
        this->Margin = System::Windows::Forms::Padding(4);
        this->Name = L"MainForm";
        this->Text = L" MainForm";
        this->Load += gcnew System::EventHandler(this,
&MainForm::mainForm_Load);
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>webcamBox))->EndInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>qrcodeBox))->EndInit();
        this->ResumeLayout(false);
        this->PerformLayout();

    }
#endif

private:
    cv::VideoCapture *videoCapture;

private: System::Void framesTimer_Tick(System::Object^  sender, System::EventArgs^  e)
{
    cv::Mat frame;

    *videoCapture >> frame;

    // Creation of Intermediate 'Image' Objects required later
    cv::Mat gray(frame.size(), CV_MAKETYPE(frame.depth(), 1)); // To
hold Grayscale Image
    cv::Mat edges(frame.size(), CV_MAKETYPE(frame.depth(), 1)); // To
hold Grayscale Image
    cv::Mat qr, qr_raw, qr_gray, qr_thres;

    std::vector< std::vector<cv::Point> > contours;
    std::vector<cv::Vec4i> hierarchy;
    std::vector<cv::Point> pointsseq; //used to save the approximated sides of
each contour

    int mark, A, B, C, top, right, bottom, median1, median2, outlier;
    float AB, BC, CA, dist, slope, areat, arear, areab, large, padding;

    int align, orientation;

    int DBG = 1; // Debug Flag

    qr_raw = cv::Mat(500, 500, CV_8UC3, cv::Scalar(255, 255, 255));
}

```

```

qr      = cv::Mat(500, 500, CV_8UC3, cv::Scalar(255, 255, 255));
qr_gray = cv::Mat(500, 500, CV_8UC1, cv::Scalar(255, 255, 255));
qr_thres = cv::Mat(500, 500, CV_8UC1, cv::Scalar(255, 255, 255));

cvtColor(frame, gray, CV_RGB2GRAY);           // Convert Image captured from
Image Input to GrayScale
Canny(gray, edges, 100, 200, 3);           // Apply Canny edge detection on the
gray image

findContours(edges, contours, hierarchy,
             cv::RetrievalModes::RETR_TREE,
             cv::ContourApproximationModes::CHAIN_APPROX_SIMPLE); // Find contours
with hierarchy

mark = 0;                                     // Reset all
detected marker count for this frame

// Get
Moments for all Contours and the mass centers
std::vector<cv::Moments> mu(contours.size());
std::vector<cv::Point2f> mc(contours.size());

for (int i = 0; i < contours.size(); i++)
{
    mu[i] = cv::moments(contours[i], false);
    mc[i] = cv::Point2f(mu[i].m10 / mu[i].m00, mu[i].m01 / mu[i].m00);
}

// Start processing the contour data

// Find Three repeatedly enclosed contours A,B,C
// NOTE: 1. Contour enclosing other contours is assumed to be the three
Alignment markings of the QR code.
// 2. Alternately, the Ratio of areas of the "concentric" squares can also be
used for identifying base Alignment markers.
// The below demonstrates the first method

for (int i = 0; i < contours.size(); i++)
{
    //Find the approximated polygon of the contour we are examining
    cv::approxPolyDP(contours[i], pointsseq, cv::arcLength(contours[i],
true)*0.02, true);
    if (pointsseq.size() == 4)      // only quadrilaterals contours are
examined
    {
        int k = i;
        int c = 0;

        while (hierarchy[k][2] != -1)
        {
            k = hierarchy[k][2];
            c = c + 1;
        }
        if (hierarchy[k][2] != -1)
            c = c + 1;

        if (c >= 5)
        {

```

```

        if (mark == 0)          A = i;
        else if (mark == 1)    B = i;      // i.e., A is already
found, assign current contour to B
        else if (mark == 2)    C = i;      // i.e., A and B are
already found, assign current contour to C
        mark = mark + 1;
    }
}

if (mark >= 3)           // Ensure we have (atleast 3; namely A,B,C)
'Alignment Markers' discovered
{
    // We have found the 3 markers for the QR code; Now we need to determine
which of them are 'top', 'right' and 'bottom' markers

    // Determining the 'top' marker
    // Vertex of the triangle NOT involved in the longest side is the
'outlier'

    AB = cv_distance(mc[A], mc[B]);
    BC = cv_distance(mc[B], mc[C]);
    CA = cv_distance(mc[C], mc[A]);

    if (AB > BC && AB > CA)
    {
        outlier = C; median1 = A; median2 = B;
    }
    else if (CA > AB && CA > BC)
    {
        outlier = B; median1 = A; median2 = C;
    }
    else if (BC > AB && BC > CA)
    {
        outlier = A; median1 = B; median2 = C;
    }

    top = outlier;                                // The
obvious choice

    dist = cv_lineEquation(mc[median1], mc[median2], mc[outlier]);      //
Get the Perpendicular distance of the outlier from the longest side
    slope = cv_lineSlope(mc[median1], mc[median2], align);      // Also
calculate the slope of the longest side

    // Now that we have the orientation of the line
formed median1 & median2 and we also have the position of the outlier w.r.t. the line

    // Determine the 'right' and 'bottom' markers

    if (align == 0)
    {
        bottom = median1;
        right = median2;
    }
    else if (slope < 0 && dist < 0)          // Orientation - North
    {
        bottom = median1;

```

```

        right = median2;
        orientation = CV_QR_NORTH;
    }
    else if (slope > 0 && dist < 0)           // Orientation - East
    {
        right = median1;
        bottom = median2;
        orientation = CV_QR_EAST;
    }
    else if (slope < 0 && dist > 0)           // Orientation - South
    {
        right = median1;
        bottom = median2;
        orientation = CV_QR_SOUTH;
    }

    else if (slope > 0 && dist > 0)           // Orientation - West
    {
        bottom = median1;
        right = median2;
        orientation = CV_QR_WEST;
    }

// To ensure any unintended values do not sneak up when QR code is not
present
float area_top, area_right, area_bottom;

if (top < contours.size() && right < contours.size() && bottom <
contours.size()
> 10
&& contourArea(contours[top]) > 10 && contourArea(contours[right])
&& contourArea(contours[bottom]) > 10)
{
    std::vector<cv::Point2f> L, M, O, tempL, tempM, tempO;
    cv::Point2f N;

    std::vector<cv::Point2f> src;           // src - Source Points basically
the 4 end co-ordinates of the overlay image
    std::vector<cv::Point2f> dst;           // dst - Destination Points to
transform overlay image

    cv::Mat warp_matrix;

    cv_getVertices(contours, top, slope, tempL);
    cv_getVertices(contours, right, slope, tempM);
    cv_getVertices(contours, bottom, slope, tempO);

    cv_updateCornerOr(orientation, tempL);           //
Re-arrange marker corners w.r.t orientation of the QR code
    cv_updateCornerOr(orientation, tempM, M);           //
Re-arrange marker corners w.r.t orientation of the QR code
    cv_updateCornerOr(orientation, tempO, O);           //
Re-arrange marker corners w.r.t orientation of the QR code

    int iflag = getIntersectionPoint(M[1], M[2], O[3], O[2], N);

    src.push_back(L[0]);
}

```

```

        src.push_back(M[1]);
        src.push_back(N);
        src.push_back(O[3]);

        dst.push_back(cv::Point2f(0, 0));
        dst.push_back(cv::Point2f(qr.cols, 0));
        dst.push_back(cv::Point2f(qr.cols, qr.rows));
        dst.push_back(cv::Point2f(0, qr.rows));

        if (src.size() == 4 && dst.size() == 4) // Failsafe
for WarpMatrix Calculation to have only 4 Points with src and dst
{
    warp_matrix = getPerspectiveTransform(src, dst);
    warpPerspective(frame, qr_raw, warp_matrix,
cv::Size(qr.cols, qr.rows));
    copyMakeBorder(qr_raw, qr, 10, 10, 10, 10,
cv::BORDER_CONSTANT, cv::Scalar(255, 255, 255));

    cvtColor(qr, qr_gray, CV_RGB2GRAY);
    threshold(qr_gray, qr_thres, 127, 255, CV_THRESH_BINARY);

//threshold(qr_gray, qr_thres, 0, 255, CV_THRESH_OTSU);
//for( int d=0 ; d < 4 ; d++){ src.pop_back();
dst.pop_back(); }
}

//Draw contours on the image
drawContours(frame, contours, top, cv::Scalar(255, 200, 0), 2, 8,
hierarchy, 0);
drawContours(frame, contours, right, cv::Scalar(0, 0, 255), 2, 8,
hierarchy, 0);
drawContours(frame, contours, bottom, cv::Scalar(255, 0, 100), 2,
8, hierarchy, 0);

// Insert Debug instructions here
if (DBG)
{
    // Debug Prints
    // Visualizations for ease of understanding
    if (slope > 5)
        circle(frame, cv::Point(10, 20), 5, cv::Scalar(0, 0,
255), -1, 8, 0);
    else if (slope < -5)
        circle(frame, cv::Point(10, 20), 5, cv::Scalar(255,
255, 255), -1, 8, 0);

    // Draw contours on Trace image for analysis
    drawContours(frame, contours, top, cv::Scalar(255, 0, 100),
1, 8, hierarchy, 0);
    drawContours(frame, contours, right, cv::Scalar(255, 0,
100), 1, 8, hierarchy, 0);
    drawContours(frame, contours, bottom, cv::Scalar(255, 0,
100), 1, 8, hierarchy, 0);

    // Draw points (4 corners) on Trace image for each
Identification marker
    circle(frame, L[0], 2, cv::Scalar(255, 255, 0), -1, 8, 0);
    circle(frame, L[1], 2, cv::Scalar(0, 255, 0), -1, 8, 0);
    circle(frame, L[2], 2, cv::Scalar(0, 0, 255), -1, 8, 0);
}

```

```

        circle(frame, L[3], 2, cv::Scalar(128, 128, 128), -1, 8,
0);

        circle(frame, M[0], 2, cv::Scalar(255, 255, 0), -1, 8, 0);
        circle(frame, M[1], 2, cv::Scalar(0, 255, 0), -1, 8, 0);
        circle(frame, M[2], 2, cv::Scalar(0, 0, 255), -1, 8, 0);
        circle(frame, M[3], 2, cv::Scalar(128, 128, 128), -1, 8,
0);

        circle(frame, O[0], 2, cv::Scalar(255, 255, 0), -1, 8, 0);
        circle(frame, O[1], 2, cv::Scalar(0, 255, 0), -1, 8, 0);
        circle(frame, O[2], 2, cv::Scalar(0, 0, 255), -1, 8, 0);
        circle(frame, O[3], 2, cv::Scalar(128, 128, 128), -1, 8,
0);

    // Draw point of the estimated 4th Corner of (entire) QR
Code
    circle(frame, N, 2, cv::Scalar(255, 255, 255), -1, 8, 0);

    // Draw the lines used for estimating the 4th Corner of QR
Code
    line(frame, M[1], N, cv::Scalar(0, 0, 255), 1, 8, 0);
    line(frame, O[3], N, cv::Scalar(0, 0, 255), 1, 8, 0);

    // Show the Orientation of the QR Code wrt to 2D Image
Space
    int fontFace = cv::FONT_HERSHEY_PLAIN;

    if (orientation == CV_QR_NORTH)
    {
        putText(frame, "NORTH", cv::Point(20, 30), fontFace,
1.5, cv::Scalar(255, 0, 255), 1, 8);
    }
    else if (orientation == CV_QR_EAST)
    {
        putText(frame, "EAST", cv::Point(20, 30), fontFace,
1.5, cv::Scalar(255, 0, 255), 1, 8);
    }
    else if (orientation == CV_QR_SOUTH)
    {
        putText(frame, "SOUTH", cv::Point(20, 30), fontFace,
1.5, cv::Scalar(255, 0, 255), 1, 8);
    }
    else if (orientation == CV_QR_WEST)
    {
        putText(frame, "WEST", cv::Point(20, 30), fontFace,
1.5, cv::Scalar(255, 0, 255), 1, 8);
    }
}

showImage(frame, webcamBox);
showImage(qr_thres, qrcodeBox);

System::String^ content = decode(qr_thres);
if (content)
{
    content = CP1251toUTF8(content);
}

```

```

        scannedCodesListBox->Items->Insert(0, content);
        if (copyToClipboardCheckbox->Checked) {
            Clipboard::SetDataObject(content, true);
        }
    }

    frame.release();
}

private: System::Void showImage(cv::Mat &image, PictureBox ^pictureBox) {
    System::Drawing::Graphics^ graphics = pictureBox->CreateGraphics();
    System::IntPtr ptr(image.ptr());
    Bitmap^ bitmap;

    switch (image.type()) {
    case CV_8UC3: // non-grayscale images are correctly displayed here
        bitmap = gcnew Bitmap(image.cols, image.rows, image.step,
            PixelFormat::Format24bppRgb, ptr);
        break;
    case CV_8UC1: // grayscale images are incorrectly displayed here
        bitmap = gcnew Bitmap(image.cols, image.rows, image.step,
            PixelFormat::Format8bppIndexed, ptr);
        break;
    default:
        break;
    }

    System::Drawing::RectangleF rect(0, 0, pictureBox->Width, pictureBox->Height);
    graphics->DrawImage(bitmap, rect);
}

System::String^ decode(cv::Mat &im)
{
    zbar::ImageScanner scanner;
    zbar::Decoder decoder;

    scanner.set_config(zbar::ZBAR_NONE, zbar::ZBAR_CFG_ENABLE, 1);
    scanner.set_config(zbar::ZBAR_NONE, zbar::ZBAR_CFG_ASCII, 1);

    zbar::Image image(im.cols, im.rows, "Y800", (uchar *)im.data, im.cols *
im.rows);

    int n = scanner.scan(image);

    for (zbar::Image::SymbolIterator symbol = image.symbol_begin(); symbol != image.symbol_end(); ++symbol)
    {
        return gcnew System::String(symbol->get_data().c_str());
    }
}

private: System::String^ CP1251toUTF8(System::String^ string)
{
    std::string instr = msclr::interop::marshal_as<std::string>(string);
    std::string outstr;
    iconv_t cd;

    cd = iconv_open("CP1251", "UTF-8");
    if (cd == (iconv_t)(-1))

```

```

        throw std::runtime_error("Не удалось открыть дескриптор iconv");

    size_t in_size = instr.size();
    const char* in = instr.c_str();

    const std::size_t BUF_SIZE = 80;
    char out_buf[BUF_SIZE];
    char* out;
    size_t out_size;

    size_t k;

    while (in_size > 0)
    {
        out = out_buf;
        out_size = BUF_SIZE;

        errno = 0;
        k = iconv(cd, &in, &in_size, &out, &out_size);

        if (k == (size_t)-1 && errno) throw std::runtime_error("Ошибка");
        outstr.append(out_buf, BUF_SIZE - out_size);
    }

    if (iconv_close(cd) != 0)
        std::runtime_error("Не удается закрыть дескриптор iconv");

    return gcnew System::String(outstr.c_str());
}

private: System::Void mainForm_Load(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void startButton_Click(System::Object^ sender, System::EventArgs^ e) {
    videoCapture = new cv::VideoCapture(0);
    if (!videoCapture->isOpened()) {
        MessageBox::Show("Cannot open webcam.");
        return;
    }
    framesTimer->Start();
    startButton->Enabled = false;
    stopButton->Enabled = true;
}
private: System::Void stopButton_Click(System::Object^ sender, System::EventArgs^ e)
{
    delete videoCapture;
    framesTimer->Stop();
    stopButton->Enabled = false;
    startButton->Enabled = true;
}
};

}

«Пахоруков Денис, гр. 23534/4. 2018 год»

```

Лабораторная работа №18

"Определение преобладающих цветов на изображении"

Задание

Напишите программу, которая осуществляет подсчет пикселей всех цветов и выявляет преобладающий цвет. Применить алгоритм к видео. Используйте шаблон.

Шаблон программы

```
#include "stdafx.h"
#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>
#include <stdlib.h>
#include <stdio.h>

#include <vector>
#include <algorithm>

using namespace cv;
using namespace std;

#define RECT_COLORS_SIZE 10

// получение пикселя изображения (по типу картинки и координатам)
#define CV_PIXEL(type,img,x,y) (((type*)((img)->imageData+(y)*(img)->widthStep)+(x)*(img)->nChannels))

enum { cBLACK = 0, cWHITE, cGREY, cRED, cORANGE, cYELLOW, cGREEN, cAQUA, cBLUE,
cPURPLE, NUM_COLOR_TYPES };
char* sCTypes[NUM_COLOR_TYPES] = { "Black",
"White", "Grey", "Red", "Orange", "Yellow", "Green", "Aqua", "Blue", "Purple" };
uchar cCTHue[NUM_COLOR_TYPES] = { 0, 0, 0, 0, 20, 30, 60,
85, 120, 138 };
uchar cCTSat[NUM_COLOR_TYPES] = { 0, 0, 255, 255, 255, 255,
255, 255 };
uchar cCTVal[NUM_COLOR_TYPES] = { 0, 255, 120, 255, 255, 255,
255, 255 };

typedef unsigned int uint;

// число пикселей данного цвета на изображении
uint colorCount[NUM_COLOR_TYPES] = { 0, 0, 0, 0, 0, 0, 0,
0, 0 };

int getPixelColorType(int H, int S, int V)
{
    int color = cBLACK;

#if 1
```

```

    if (V < 75)
        color = cBLACK;
    else if (V > 190 && S < 27)
        color = cWHITE;
    else if (S < 53 && V < 185)
        color = cGREY;
    else
#endif
{
    if (H < 7)
        color = cRED;
    else if (H < 25)
        color = cORANGE;
    else if (H < 34)
        color = cYELLOW;
    else if (H < 73)
        color = cGREEN;
    else if (H < 102)
        color = cAQUA;
    else if (H < 140)
        color = cBLUE;
    else if (H < 170)
        color = cPURPLE;
    else
        color = cRED;
}
return color;
}

// сортировка цветов по количеству
bool colors_sort(std::pair< int, uint > a, std::pair< int, uint > b)
{
    return (a.second > b.second);
}

int main(int argc, char* argv[])
{
    // для хранения изображения
    IplImage* image = 0;
    IplImage *hsv = 0;
    IplImage *dst = 0;
    IplImage *color_indexes = 0;

    //
    // загрузка изображения
    //

    char img_name[] = "grass.jpg";

    // имя картинки задаётся первым параметром
    char* image_filename = argc >= 2 ? argv[1] : img_name;

    // получаем картинку
    image = cvLoadImage(image_filename, 1);

    printf("[i] image: %s\n", image_filename);
    if (!image) {
        printf("![!] Error: cant load test image: %s\n", image_filename);
        return -1;
    }
}

```

```

}

// показываем картинку
cvNamedWindow("image");
cvShowImage("image", image);

//
// преобразуем изображение в HSV
//
hsv = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 3);
cvCvtColor(image, hsv, CV_BGR2HSV);

// картинки для хранения результатов
dst = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 3);
color_indexes = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1); //для хранения
индексов цвета

// для хранения RGB-х цветов
CvScalar rgb_colors[NUM_COLOR_TYPES];

int i = 0, j = 0, x = 0, y = 0;

// обнуляем цвета
for (i = 0; i<NUM_COLOR_TYPES; i++) {
    rgb_colors[i] = cvScalarAll(0);
}

for (y = 0; y<hsv->height; y++) {
    for (x = 0; x<hsv->width; x++) {

        // получаем HSV-компоненты пикселя
        uchar H = CV_PIXEL(uchar, hsv, x, y)[0];           // Hue
        uchar S = CV_PIXEL(uchar, hsv, x, y)[1];           // Saturation
        uchar V = CV_PIXEL(uchar, hsv, x, y)[2];           // Value
(Brightness)

        // определяем к какому цвету можно отнести данные значения
        int ctype = getColorType(H, S, V);

        // устанавливаем этот цвет у отладочной картинки
        CV_PIXEL(uchar, dst, x, y)[0] = cCTHue[ctype]; // Hue
        CV_PIXEL(uchar, dst, x, y)[1] = cCTSat[ctype]; // Saturation
        CV_PIXEL(uchar, dst, x, y)[2] = cCTVal[ctype]; // Value

        // собираем RGB-составляющие
        rgb_colors[ctype].val[0] += CV_PIXEL(uchar, image, x, y)[0]; // B
        rgb_colors[ctype].val[1] += CV_PIXEL(uchar, image, x, y)[1]; // G
        rgb_colors[ctype].val[2] += CV_PIXEL(uchar, image, x, y)[2]; // R

        // сохраняем к какому типу относится цвет
        CV_PIXEL(uchar, color_indexes, x, y)[0] = ctype;

        // подсчитываем
        colorCount[ctype]++;
    }
}

```

```

// усреднение RGB-составляющих
for (i = 0; i<NUM_COLOR_TYPES; i++) {
    rgb_colors[i].val[0] /= colorCount[i];
    rgb_colors[i].val[1] /= colorCount[i];
    rgb_colors[i].val[2] /= colorCount[i];
}

// теперь загоним массив в вектор и отсортируем
std::vector< std::pair< int, uint > > colors;
colors.reserve(NUM_COLOR_TYPES);

for (i = 0; i<NUM_COLOR_TYPES; i++) {
    std::pair< int, uint > color;
    color.first = i;
    color.second = colorCount[i];
    colors.push_back(color);
}
// сортируем
std::sort(colors.begin(), colors.end(), colors_sort);

// для отладки - выводим коды, названия цветов и их количество
for (i = 0; i<colors.size(); i++) {
    printf("[i] color %d (%s) - %d\n", colors[i].first,
sCTypes[colors[i].first], colors[i].second);
}

// выдаём код первых цветов
printf("[i] color code: \n");
for (i = 0; i<NUM_COLOR_TYPES; i++)
    printf("%02d ", colors[i].first);
printf("\n");
printf("[i] color names: \n");
for (i = 0; i<NUM_COLOR_TYPES; i++)
    printf("%s ", sCTypes[colors[i].first]);
printf("\n");

// конвертируем отладочную картинку обратно в RGB
cvCvtColor(dst, dst, CV_HSV2BGR);

// показываем результат
cvNamedWindow("color");
cvShowImage("color", dst);

// ждём нажатия клавиши
cvWaitKey(0);

// освобождаем ресурсы
cvReleaseImage(&image);
cvReleaseImage(&hsv);
cvReleaseImage(&dst);
cvReleaseImage(&color_indexes);

// удаляем окна
cvDestroyAllWindows();

return 0;
}

```

Лабораторная работа №19

"Стерео изображение. Стерео геометрия"

Сейчас мы активно используем 3D изображения в повседневной жизни, например, смотрим фильмы в формате 3D, у некоторых возможно смартфон имеет стереокамеру и позволяет снимать 3D ролики. Также стереоизображение может использоваться в индустрии и в практических целях, например, для управления беспилотным транспортом, составления 3-х мерных карт и иных целей, где 3-х мерное представление объекта улучшает его понимание или крайне необходимо.

Задание. Необходимо реализовать

1. Захват видео с 2-ух веб камер и демонстрацию захваченных изображений.
2. Рассчитать дистанцию до известного (данные по размерам объекта введены пользователем) объекта
3. Рассчитать дистанцию до произвольного выбранного пользователем объекта
4. Попробовать 3D реконструкцию изображения (видео)
5. Откалибровать стереокамеру
6. Реализовать 3D реконструкцию с откалиброванной стереокамерой
7. (*) Попытаться реализовать расчёт расстояния до объекта с полностью неоткалиброванных камер
8. (*) Реализовать построение 3D модели (положение основных точек в 3-х мерном пространстве) с неоткалиброванных камер.
9. (*) Реализовать расчёт расстояния до объекта при помощи 1-ой камеры¹
- 10.(*) Построить 3D модель как в п.8 при помощи 1-ой камеры.

Теоретическая часть

Для создания 3-х мерного изображения, необходима стереосъёмка, которую можно получить 2-мя путями: снимать видео (изображения) одновременно с 2-ух камер или перемещать одну камеру.

Съемка с двух камер

Допустим у нас есть 2-е одинаковые камеры (или же мы программным образом сделаем их одинаковыми). Тогда для построения 3-х мерного изображения мы можем выполнить следующие простые действия:

¹ Здесь и далее вместо 1 камеры для удобства может быть использовано записанное видео.

1. Откалибровать стереокамеру, используя функцию CvInvoke.StereoCalibrate(...)
2. При помощи функции CvInvoke.StereoRectify строим матрицу выравнивания 2-ух изображений - Q
3. При помощи StereoBM/StereoSGBM считаем матрицу смещения.
4. При помощи CvInvoke.ReprojectImageTo3D(...), где нам пригодится матрица Q строим 3-х мерное изображение, т.е. получаем облако 3-х мерных точек

Бывает так, что строить 3-х мерное изображение нет необходимости, а нужно всего лишь рассчитать расстояние до некоторого объекта (или до группы объектов). Для этого мы можем использовать нашу откалиброванную камеру (можно использовать и не откалиброванную, но точность результата хуже) и известные формулы:

$$Z = \frac{Bf}{\text{delta}}$$

$$X = Z \frac{\left(x - \frac{w}{2}\right)}{f}$$

$$Y = Z \frac{\left(y - \frac{h}{2}\right)}{f}$$

B – линия базы

f – фокусное расстояние

x, y – координаты на изображении

h, w – высота и ширина изображения

delta = δx

Во всех случаях, если камера может изменять фокусное расстояние, постарайтесь выяснить новое фокусное расстояние. А в случае получения матрицы камер через функции CvInvoke.StereoCalibrate(...) или же через CvInvoke.CameraCalibrate(...) сделайте так, чтобы фокусное расстояние не изменялось.

Получить δx можно разными способами. Один из способов – матрица смещения.

Съемка с одной камеры

При съемке с одной камеры мы не сможем использовать CvInvoke.StereoCalibrate(...), что же нам делать? Тут нас выручит оптический поток, который можно рассчитать при помощи функции CvInvoke.CalcOpticalFlowFarneback(...), а магнитуду и угол смещения можно расчитать из оптического потока при помощи метода CvInvoke.CartToPolar(...), а дальше можно воспользоваться формулами, описанными выше.

Примечания

Смотрите информацию по классу Viz3d – для визуализации и по классу (пространству имён) CvInvoke, для обработки изображений.

<http://www.emgu.com>

<http://www.emgu.com/wiki/files/3.4.1/document/html/8dee1f02-8c8a-4e37-87f4-05e10c39f27d.htm>

Приложение

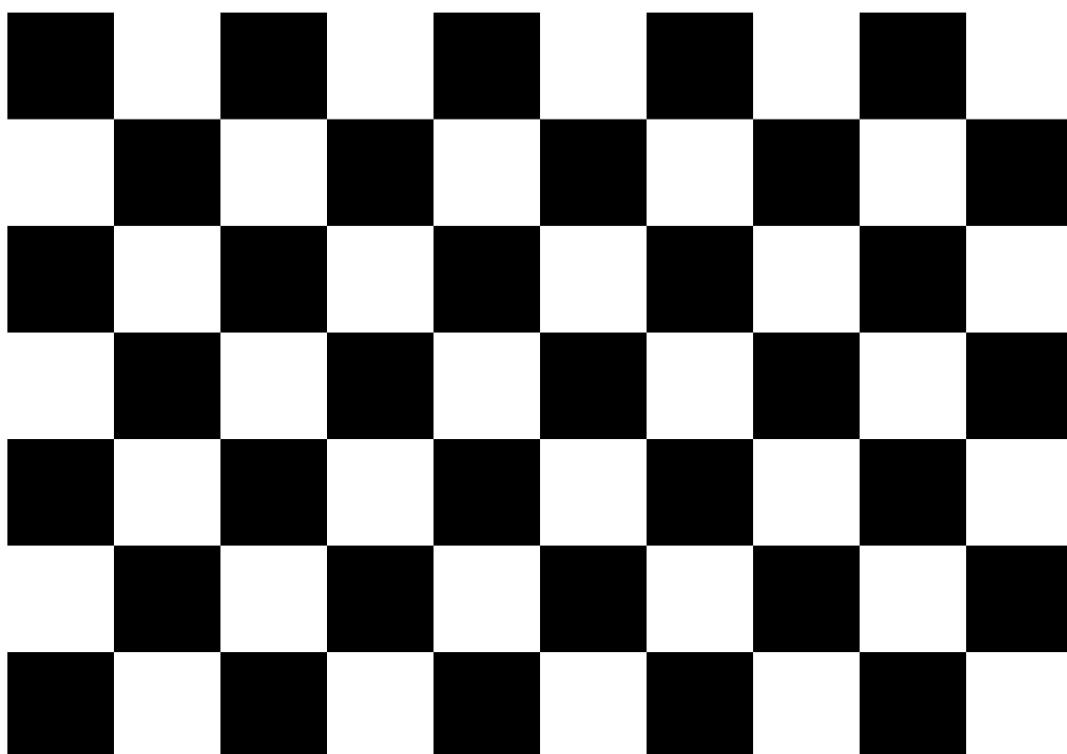


Рисунок 3. Изображение для калибровки камеры (9*6)

Самая новая версия шаблона хранится на GitHub:
<https://github.com/SPBSTRU-acm/3DReconstruction-Emgu-Cv>

Шаблон кода

```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Linq;
7. using System.Text;
8. using System.Windows.Forms;
9. using Emgu.CV;
10. using Emgu.CV.CvEnum;
11. using Emgu.CV.UI;
12. using Emgu.Util;
13. using Emgu.CV.Structure;
```

```

14.     using Emgu.CV.Tracking;
15.     using System.Diagnostics;
16.
17.     namespace ACMSemProject
18.     {
19.         public partial class Form1 : Form
20.         {
21.             private bool _isCameraMatrixCount = false;
22.             Matrix<float> cameraMatrix1 = new
23.                 Matrix<float>(3, 3);
24.             Matrix<float> cameraMatrix2 = new
25.                 Matrix<float>(3, 3);
26.             Matrix<float> distCoeff1 = new Matrix<float>(4,
27.                 1);
28.             Matrix<float> distCoeff2 = new Matrix<float>(4,
29.                 1);
30.             Matrix<double> R1 = new Matrix<double>(3, 3);
31.             //rectification transforms (rotation matrices) for Camera 1.
32.             Matrix<double> R2 = new Matrix<double>(3, 3);
33.             //rectification transforms (rotation matrices) for Camera 1.
34.             Matrix<double> P1 = new Matrix<double>(3, 4);
35.             //projection matrices in the new (rectified) coordinate systems
36.             //for Camera 1.
37.             Matrix<double> P2 = new Matrix<double>(3, 4);
38.             //projection matrices in the new (rectified) coordinate systems
39.             //for Camera 2.
40.
41.             private bool _canGrabRightImage = false;
42.             private int[] _cameraIndex = new int[2] { 0, 1 };
43.             private bool _isCalibrate = false;
44.             private VideoCapture _capture = null;
45.             private VideoCapture _capture2 = null;
46.
47.             private bool _captureInProgress;
48.             private Rectangle _regionLeft = new Rectangle(-1,
49.                 -1, -1, -1);
50.             private bool _haveRegion = false;
51.             private Point _startPoint = new Point(-1, -1);
52.
53.             private Tracker _trackerLeft = null;
54.             private Mat[] images = new Mat[2] { new Mat(),
55.                 new Mat() };
56.             private Mat _left = new Mat();
57.             private Mat _right = new Mat();
58.             private Mat _tmpRight = new Mat();
59.             private bool _makeCalibratingPhoto = false;
60.             private Viz3d viz3d;
61.
62.             private const double epsilon = 0.00000001;
63.             private double _focalLength = 3.5;

```

```

52.         private double _baseLine = 1100;
53.
54.         private float _blockSize = 25;
55.
56.         private Size _boardSize = new Size(9, 6);
57.
58.         private static int _imageBufferLength = 20;
59.
60.         private Bgr[] lineColourArray = new Bgr[9 * 6];
61.         private static int N = _imageBufferLength * 9 *
62.             6;
63.         private PointF[][] corners_points_Left = new
64.             PointF[_imageBufferLength][];//stores the calculated points
65.             from chessboard detection Camera 1
66.         private PointF[][] corners_points_Right = new
67.             PointF[_imageBufferLength][];//stores the calculated points
68.             from chessboard detection Camera 2
69.
70.         private Matrix<double> Q = new Matrix<double>(4,
71.             4); //This is what were interested in the disparity-to-depth
72.             mapping matrix
73.
74.         private int _currentImageIndex = 0;
75.         private bool _is3dStart = false;
76.
77.     public Form1()
78.     {
79.         InitializeComponent();
80.         CvInvoke.UseOpenCL = false;
81.     }
82.
83.     private void AddCaptureParams(EventHandler
84.         captureHandler)
85.     {
86.         try
87.         {
88.             _capture.ImageGrabbed += captureHandler;
89.             _capture2.ImageGrabbed +=
90.                 GrabbRightImage;
91.         }
92.         catch (NullReferenceException excpt)
93.         {
94.             MessageBox.Show(excpt.Message);
95.         }
96.         _capture.Start();
97.         _capture2.Start();
98.         _captureInProgress = true;

```

```

93.                 btnPlay.Text = "Pause";
94.             }
95.
96.         private void GrabbRightImage(object sender,
97.                                         EventArgs e)
98.             {
99.                 if (_canGrabRightImage)
100.                     {
101.                         _capture2.Read(_tmpRight);
102.                         _canGrabRightImage = false;
103.                     }
104.
105.         private void ProcessWithoutCalibration(object
106.                                         sender, EventArgs e)
107.             {
108.                 if (_capture != null && _capture.Ptr !=
109.                     IntPtr.Zero)
110.                     {
111.                         Mat tmp = new Mat();
112.                         CaptureStereoImages(_capture, _capture2,
113.                             out _left, out _right);
114.                         Mat[] edges = new Mat[2] { new Mat(), new
115.                             Mat() };
116.                         CvInvoke.Canny(_left, edges[0], 100,
117.                             200);
118.                         CvInvoke.Canny(_right, edges[1], 100,
119.                             200);
120.                         if (_haveRegion && _trackerLeft == null)
121.                         {
122.                             tmp = DrawRect(_regionLeft,
123.                               edges[0]);
124.                             _trackerLeft = new TrackerKCF();
125.                             _trackerLeft.Init(_left,
126.                               _regionLeft);
127.                         else if (_haveRegion)
128.                         {
129.                             Rectangle rectLeft = new Rectangle();
130.                             _trackerLeft.Update(_left, out
131.                               rectLeft);
132.                             _regionLeft = rectLeft;
133.                             tmp = DrawRect(_regionLeft,
134.                               edges[0]);
135.                         }
136.                     }
137.                 }
138.             }
139.         }
140.     }
141. 
```

```

130.                 double distances =
    CountDistanceThroughEdges(edges[0], edges[1], _regionLeft,
    _baseLine, _focalLength);
131.                 CvInvoke.PutText(_right, "distance =
    " + distances, new Point(0, _right.Height - 30),
132.                                         FontFace.HersheySimplex, 0.5, new
    Bgr(Color.Red).MCvScalar, 2);
133.             }
134.         else
135.             {
136.                 tmp = edges[0];
137.             }
138.
139.                 ShowStereoImages(tmp, _right);
140.             }
141.         }
142.
143.         private void CaptureStereoImages(VideoCapture
    capture1, VideoCapture capture2, out Mat mat1, out Mat mat2)
144.         {
145.             Mat mattmp1 = new Mat();
146.
147.             if (capture1 == null || capture2 == null)
148.             {
149.                 throw new NullReferenceException("Input
    exist capture parameters");
150.             }
151.             capture1.Read(mattmp1);
152.             _canGrabRightImage = true;
153.             while (_canGrabRightImage) { };
154.             mat1 = mattmp1.Clone();
155.             mat2 = _tmpRight.Clone();
156.         }
157.
158.         private void ShowStereoImages(Mat left, Mat
    right)
159.         {
160.             imageBox.Image = left;
161.             imageBox2.Image = right;
162.         }
163.
164.         private Mat DrawRect(Rectangle rect, Mat image)
165.         {
166.             Mat rectangleMat = image.Clone();
167.             CvInvoke.Rectangle(rectangleMat, new
    Rectangle(rect.X - 3, rect.Y - 3, rect.Width + 6, rect.Height +
    6), new Bgr(Color.Blue).MCvScalar, 2);
168.             return rectangleMat;
169.         }
170.

```

```

171.         private static double
172.             CountDistanceThroughEdges(Mat leftEdges, Mat rightEdges,
173.                 Rectangle roi,
174.                     double baseLine, double focalLengthPx)
175.             {
176.                 if (roi == null)
177.                 {
178.                     return -1;
179.                 }
180.                 Mat prev8bit = leftEdges;
181.                 Mat current8bit = rightEdges;
182. 
183.                 Image<Gray, byte> imageLeft =
184.                     prev8bit.ToImage<Gray, byte>();
185.                 Image<Gray, byte> imageRight =
186.                     current8bit.ToImage<Gray, byte>();
187.                 Image<Gray, float> flowX = new Image<Gray,
188.                     float>(prev8bit.Width, prev8bit.Height);
189.                 Image<Gray, float> flowY = new Image<Gray,
190.                     float>(prev8bit.Width, prev8bit.Height);
191. 
192.                 CvInvoke.CalcOpticalFlowFarneback(imageLeft,
193.                     imageRight, flowX,
194.                         flowY, 0.5, 3, 25, 10, 5, 1.1,
195.                         OpticalflowFarnebackFlag.FarnebackGaussian);
196. 
197.                 Mat magnitude = new Mat();
198.                 Mat angle = new Mat();
199. 
200.                 CvInvoke.CartToPolar(flowX.Mat, flowY.Mat,
201.                     magnitude, angle);
202. 
203.                 int matWidth = magnitude.Width;
204.                 int matHeight = magnitude.Height;
205.                 float[] magData = new float[matWidth *
206.                     matHeight];
207. 
208.                 magnitude.CopyTo(magData);
209.                 List<double> results = new List<double>();
210. 
211.                 for (int x = roi.X; x <= roi.X + roi.Width;
212.                     x++)
213.                 {
214.                     for (int y = roi.Y; y <= roi.Y +
215.                         roi.Height; y++)
216.                     {
217.                         float delta =
218.                             GetElementFromArrayAsFromMatrix(magData, y, x,
219.                                 matWidth);
220.                         if (delta < epsilon)
221.                         {
222.                             continue;
223.                         }
224.                         double distance = (baseLine *
225.                             focalLengthPx) / delta;

```

```

207.                     results.Add(distance);
208.                 }
209.             }
210.         if (results.Count == 0)
211.         {
212.             return -1;
213.         }
214.
215.         return results.Average();
216.     }
217.
218.     private static T
GetElementFromArrayAsFromMatrix<T>(T[] data, int row, int
column, int width)
219.     {
220.         return data[row * width + column];
221.     }
222.
223.     private void btnCapture_Click(object sender,
EventArgs e)
224.     {
225.         lTypeOfFocalLength.Text = "px";
226.         StartStereoCapturing(_capture,
_cameraIndex[0], _capture2, _cameraIndex[1], out _capture, out
_capture2);
227.         AddCaptureParams(ProcessWithoutCalibration);
228.
229.     }
230.
231.     private void StartStereoCapturing(VideoCapture
capture1, int cameraIndex1, VideoCapture capture2, int
cameraIndex2, out VideoCapture newCapture1, out VideoCapture
newCapture2)
232.     {
233.         newCapture1 = StartCapturing(capture1,
cameraIndex1);
234.         newCapture2 = StartCapturing(capture2,
cameraIndex2);
235.     }
236.
237.     private VideoCapture StartCapturing(VideoCapture
stopeingCaptur, int cameraIndex)
238.     {
239.         StopCapturing(stopeingCaptur);
240.         return new VideoCapture(cameraIndex);
241.     }
242.
243.     private void StopCapturing(VideoCapture capture)
244.     {
245.         if (capture != null)

```

```

246.          {
247.              capture.Stop();
248.          }
249.      }
250.
251.      private void btnPlay_Click(object sender,
252.          EventArgs e)
253.      {
254.          if (_capture != null && _capture2 != null)
255.          {
256.              if (!_captureInProgress)
257.              {
258.                  btnPlay.Text = "Pause";
259.                  _capture.Start();
260.                  _capture2.Start();
261.                  _captureInProgress = true;
262.              }
263.          else
264.          {
265.              btnPlay.Text = "Play";
266.              _capture.Pause();
267.              _capture2.Pause();
268.              _captureInProgress = false;
269.          }
270.      }
271.  }
272.
273.  private void btnStop_Click(object sender,
274.      EventArgs e)
275.  {
276.      if (_capture != null && _capture2 != null)
277.      {
278.          _capture.Stop();
279.          _capture2.Stop();
280.      }
281.
282.
283.  private void button1_Click(object sender,
284.      EventArgs e)
285.  {
286.      N = _boardSize.Width * _boardSize.Height *
287.          _imageBufferLength;
288.      _isCalibrate = false;
289.      _currentImageIndex = 0;
290.      StartStereoCapturing(_capture,
291.          _cameraIndex[0], _capture2, _cameraIndex[1], out _capture, out
292.          _capture2);

```

```

290.             CalibrateStereoCamera();
291.         }
292.
293.         private void button2_Click(object sender,
294.             EventArgs e)
295.         {
296.             qbMatrixQ.Enabled = true;
297.             StartStereoCapturing(_capture,
298.                 _cameraIndex[0], _capture2, _cameraIndex[1], out _capture, out
299.                 _capture2);
300.             Show3DVideo();
301.         }
302.         _haveRegion = false;
303.         _startPoint = new Point(-1, -1);
304.         _trackerLeft = null;
305.     }
306.
307.     private void pictureBox_DoubleClick(object sender,
308.         EventArgs e)
309.     {
310.         if (_startPoint.X < 0 && !_haveRegion)
311.         {
312.             _startPoint = e.Location;
313.         }
314.         else
315.         {
316.             _haveRegion = true;
317.             Point pos = e.Location;
318.             _regionLeft = new Rectangle(_startPoint,
319.                 new Size(_startPoint.X + pos.X,
320.                         _startPoint.Y + pos.Y));
321.         }
322.     }
323.     private void CalibrateStereoCamera()
324.     {
325.         AddCaptureParams(CalibrateStereoCameraProcess);
326.     }
327.     private void Show3DVideo()
328.     {
329.         InitializeViz3D();
330.         AddCaptureParams(Show3DVideoProcess);
331.         InitializeGroup();
332.         //Display3D(_left, _right);

```

```

333.        }
334.
335.        private void InitializeViz3D()
336.        {
337.            viz3d = new Viz3d("3D");
338.        }
339.
340.        private void Show3DVideoProcess(object sender,
341.                                         EventArgs e)
341.        {
342.            CaptureStereoImages(_capture, _capture2, out
343.            _left, out _right);
344.            Display3D(_left, _right);
345.
346.            ShowStereoImages(_left, _right);
347.        }
348.
349.        private void Display3D(Mat left, Mat right)
350.        {
351.            //TODO: try to use StereoSGBM
352.            using (StereoBM stereoSolver = new
353.                StereoBM())
353.            {
354.
355.                Mat output = new Mat();
356.                Mat left8bit = ConvertInto8bitMat(left);
357.                Mat right8bit =
358.                    ConvertInto8bitMat(right);
358.                stereoSolver.Compute(left8bit, right8bit,
359.                    output);
359.                Mat points = new Mat();
360.                float scale = Math.Max(left.Size.Width,
360.                    left.Size.Height);
361.                if (!_isCalibrate)
362.                {
363.                    Q = new Matrix<double>(
364.                        new double[,] {
365.                            {
366.                                {1.0, 0.0, 0.0, -left.Width/2}, //shift
366.                                    the x origin to image center
367.                                {0.0, -1.0, 0.0, left.Height/2},
367.                                    //shift the y origin to image center and flip it upside down
368.                                {0.0, 0.0, -1.0, 0.0}, //Multiply the z
368.                                    value by -1.0,
369.                                {0.0, 0.0, 0.0, scale}
369.                            });
370.                            _isCalibrate = true;
371.                        }
372.                    }

```

```

373.                                //Construct a simple Q matrix, if you
have a matrix from cvStereoRectify, you should use that instead
374.                                //scale the object's coordinate to within
a [-0.5, 0.5] cube
375.                                if (_isCameraMatrixCount) {
376.                                    Mat map11 = new Mat();
377.                                    Mat map12 = new Mat();
378.                                    Mat map21 = new Mat();
379.                                    Mat map22 = new Mat();
380.
CvInvoke.InitUndistortRectifyMap(cameraMatrix1, distCoeff1, R1,
P1, left8bit.Size,
381.                                DepthType.Cv16S, map11, map12);
382.
CvInvoke.InitUndistortRectifyMap(cameraMatrix2, distCoeff2, R2,
P2, left8bit.Size,
383.                                DepthType.Cv16S, map21, map22);
384.
385.                                Mat img1r = new Mat();
386.                                Mat img2r = new Mat();
387.                                CvInvoke.Remap(left8bit, img1r,
map11, map12, Inter.Linear);
388.                                CvInvoke.Remap(right8bit, img2r,
map21, map22, Inter.Linear);
389.                                left8bit = img1r;
390.                                right8bit = img2r;
391.                            }
392.
//stereoSolver.FindStereoCorrespondence(left, right,
disparityMap);
393.                                CvInvoke.ReprojectImageTo3D(output,
points, Q, false, DepthType.Cv32F);
394.                                //points =
PointCollection.ReprojectImageTo3D(output, Q);
395.                                Mat pointsArray =
points.Reshape(points.NumberOfChannels, points.Rows *
points.Cols);
396.                                Mat colorArray =
left.Reshape(left.NumberOfChannels, left.Rows * left.Cols);
397.                                Mat colorArrayFloat = new Mat();
398.                                colorArray.ConvertTo(colorArrayFloat,
DepthType.Cv32F);
399.                                WCloud cloud = new WCloud(pointsArray,
colorArray);
400.
401.                                Display3DImage(cloud);
402.
403.                                //points =
PointCollection.ReprojectImageTo3D(outputDisparityMap, q);
404.                            }

```

```

405.        }
406.
407.        private void InitializeGroup()
408.        {
409.
410.            tb00.Text = "1";
411.            tb01.Text = "0";
412.            tb02.Text = "0";
413.            tb03.Text = "-320";
414.            tb10.Text = "0";
415.            tb11.Text = "-1";
416.            tb12.Text = "0";
417.            tb13.Text = "240";
418.            tb20.Text = "0";
419.            tb21.Text = "0";
420.            tb22.Text = "-1";
421.            tb23.Text = "0";
422.            tb30.Text = "0";
423.            tb31.Text = "0";
424.            tb32.Text = "0";
425.            tb33.Text = "640";
426.        }
427.
428.        private void Display3DImage(WCloud cloud)
429.        {
430.            if (_is3dStart)
431.            {
432.                viz3d.RemoveWidget("cloud");
433.            }
434.            viz3d.ShowWidget("cloud", cloud);
435.            viz3d.SpinOnce();
436.            _is3dStart = true;
437.        }
438.        private void CalibrateStereoCameraProcess(object
439.            sender, EventArgs e)
440.        {
441.            if (_capture == null || _capture.Ptr ==
442.                IntPtr.Zero || _capture2 == null || _capture2 == IntPtr.Zero)
443.            {
444.                throw new Exception("Invalid capture
445.                    parameters");
446.            }
447.            CaptureStereoImages(_capture, _capture2, out
448.                _left, out _right);
449.            Mat displayLeft = _left;
450.            Mat displayRight = _right;

```

```

451.
452.
453.                bool find =
    AddCornersFromCurrentImages(_left, _right, out leftCorners, out
    rightCorners);
454.
455.                if (leftCorners != null)
456.                {
457.
        CvInvoke.DrawChessboardCorners(displayLeft, _boardSize,
        leftCorners, find);
458.                }
459.                if (rightCorners != null)
460.                {
461.
        CvInvoke.DrawChessboardCorners(displayRight, _boardSize,
        rightCorners, find);
462.                }
463.
464.            }
465.
466.            ShowStereoImages(displayLeft, displayRight);
467.            if (_currentIndex >= _imageBufferLength)
468.            {
469.                if (!_isCalibrate)
470.                {
471.                    StereoCameraCalibration();
472.                }
473.                else
474.                {
475.                    MessageBox.Show("Press button 3D");
476.                    return;
477.                }
478.            }
479.
480.        }
481.
482.        private bool AddCornersFromCurrentImages(Mat
    left, Mat right, out Mat leftCorners, out Mat rightCorners)
483.        {
484.            bool find1 = false;
485.            bool find2 = false;
486.            Mat leftGray = ConvertInto8bitMat(left);
487.            Mat rightGray = ConvertInto8bitMat(right);
488.
489.            Mat pointsLeft = new Mat();
490.            Mat pointsRight = new Mat();
491.            find1 =
    CvInvoke.FindChessboardCorners(leftGray, _boardSize,
    pointsLeft);

```

```

492.             find2 =
493.                 CvInvoke.FindChessboardCorners(rightGray, _boardSize,
494.                                         pointsRight);
495.             if (!(find1 && find2))
496.             {
497.                 leftCorners = null;
498.                 rightCorners = null;
499.                 return false;
500.             }
501.             leftCorners = pointsLeft;
502.             rightCorners = pointsRight;
503.             if (_currentImageIndex >= _imageBufferLength
504. || !_makeCalibratingPhoto)
505.             {
506.                 return true;
507.             }
508.             PointF[] pointsL = new
509.             PointF[_boardSize.Width * _boardSize.Height];
510.             PointF[] pointsR = new
511.             PointF[_boardSize.Width * _boardSize.Height];
512.             pointsLeft.CopyTo(pointsL);
513.             pointsRight.CopyTo(pointsR);
514.             corners_points_Left[_currentImageIndex] = new
515.             PointF[_boardSize.Width * _boardSize.Height];
516.             corners_points_Right[_currentImageIndex] =
517.             new PointF[_boardSize.Width * _boardSize.Height];
518.             corners_points_Left[_currentImageIndex] =
519.             pointsL;
520.             corners_points_Right[_currentImageIndex] =
521.             (PointF[])pointsLeft.Data;
522.             _currentImageIndex++;
523.             _makeCalibratingPhoto = false;
524.             return true;
525.         }
526.         private void StereoCameraCalibration()
527.         {
528.             MCvPoint3D32f[][] corners_object_Points = new
529.             MCvPoint3D32f[_imageBufferLength][]; //stores the calculated
size for the chessboard

```

```

530.         Mat f = new Mat();
531.         Mat e1 = new Mat();
532.
533.             //fill the MCvPoint3D32f with correct
      mesurments
534.             for (int k = 0; k < _imageBufferLength; k++)
535.             {
536.                 //Fill our objects list with the real
      world mesurments for the intrinsic calculations
537.                 List<MCvPoint3D32f> object_list = new
      List<MCvPoint3D32f>();
538.                 for (int i = 0; i < _boardSize.Height;
      i++)
539.                 {
540.                     for (int j = 0; j < _boardSize.Width;
      j++)
541.                     {
542.                         object_list.Add(new
      MCvPoint3D32f(j * _blockSize, i * _blockSize, 0.0F));
543.                     }
544.                 }
545.                 corners_object_Points[k] =
      object_list.ToArray();
546.             }
547.
548.             MessageBox.Show("End capturing images for
      calibration stereo camera");
549.
550.
      CvInvoke.StereoCalibrate(corners_object_Points,
      corners_points_Left, corners_points_Right, cameraMatrix1,
      distCoeff1,
551.             cameraMatrix2, distCoeff2, _left.Size, r,
      t, e1, f, CalibType.Default, new MCvTermCriteria(0.1e5));
552.
553.             MessageBox.Show("Succesful calibrate stereo
      camera");
554.             Rectangle rec1 = new Rectangle();
555.             Rectangle rec2 = new Rectangle();
556.             //Computes rectification transforms for each
      head of a calibrated stereo camera.
557.             MessageBox.Show("Start Rectify");
558.             CvInvoke.StereoRectify(cameraMatrix1,
      distCoeff1,
559.                                         cameraMatrix2,
      distCoeff2,
560.                                         _left.Size,
561.                                         r, t,
      R1, R2, P1, P2, Q,
562.

```

```

563.         StereoRectifyType.Default, 0,
564.                               _left.Size, ref
565.                               rec1, ref rec2);
565.                               MessageBox.Show("rect1 = " + rec1 + "\nrect2
565. = " + rec2 + "\nQ= \n" + Q[0, 0] + " " + Q[0, 1] + " " + Q[0,
565. 2] + " " + Q[0, 3] +
566.                               "\n" + Q[1, 0] + " " + Q[1, 1] + " "
566. + " " + Q[1, 2] + " " + Q[1, 3] + "\n" + Q[2, 0] + " " + Q[2, 1] + " "
567. +
567.                               Q[2, 2] + " " + Q[2, 3] + "\n" + Q[3, 0]
567. + " " + Q[3, 1] + " " + Q[3, 2] + " " + Q[3, 3]);
568.
569.                               _isCalibrate = true;
570.                               _isCameraMatrixCount = true;
571. }
572.
573.     private static Mat ConvertInto8bitMat(Mat mat)
574. {
575.     if (mat == null)
576.     {
577.         return mat;
578.     }
579.     Mat gray = new Mat();
580.     CvInvoke.CvtColor(mat, gray,
580.     ColorConversion.Bgr2Gray);
581.     return gray;
582. }
583.
584.     private void tb00_TextChanged(object sender,
584. EventArgs e)
585.     {
586.         Q[0, 0] = Double.Parse(tb00.Text);
587.     }
588.
589.     private void tb10_TextChanged(object sender,
589. EventArgs e)
590.     {
591.         Q[1, 0] = Double.Parse(tb00.Text);
592.     }
593.
594.     private void tb20_TextChanged(object sender,
594. EventArgs e)
595.     {
596.
597.         Q[2, 0] = Double.Parse(tb00.Text);
598.     }
599.
600.     private void tb30_TextChanged(object sender,
600. EventArgs e)

```

```

601.          {
602.              Q[3, 0] = Double.Parse(tb00.Text);
603.          }
604.
605.          private void tb01_TextChanged(object sender,
606.                                         EventArgs e)
607.          {
608.              Q[0, 1] = Double.Parse(tb00.Text);
609.          }
610.
611.          private void tb11_TextChanged(object sender,
612.                                         EventArgs e)
613.          {
614.              Q[1, 1] = Double.Parse(tb00.Text);
615.          }
616.
617.          private void tb21_TextChanged(object sender,
618.                                         EventArgs e)
619.          {
620.              Q[2, 1] = Double.Parse(tb00.Text);
621.          }
622.
623.          private void tb31_TextChanged(object sender,
624.                                         EventArgs e)
625.          {
626.              Q[3, 1] = Double.Parse(tb00.Text);
627.          }
628.          private void tb02_TextChanged(object sender,
629.                                         EventArgs e)
630.          {
631.              Q[0, 2] = Double.Parse(tb00.Text);
632.          }
633.
634.          private void tb12_TextChanged(object sender,
635.                                         EventArgs e)
636.          {
637.              Q[1, 2] = Double.Parse(tb00.Text);
638.          }
639.
640.          private void tb22_TextChanged(object sender,
641.                                         EventArgs e)
642.          {
643.              Q[2, 2] = Double.Parse(tb00.Text);

```

```

644.        }
645.
646.        private void tb32_TextChanged(object sender,
   EventArgs e)
647.        {
648.
649.            Q[3, 2] = Double.Parse(tb00.Text);
650.        }
651.        private void tb03_TextChanged(object sender,
   EventArgs e)
652.        {
653.
654.            Q[0, 3] = Double.Parse(tb00.Text);
655.        }
656.
657.        private void tb13_TextChanged(object sender,
   EventArgs e)
658.        {
659.
660.            Q[1, 3] = Double.Parse(tb00.Text);
661.        }
662.
663.        private void tb23_TextChanged(object sender,
   EventArgs e)
664.        {
665.
666.            Q[2, 3] = Double.Parse(tb00.Text);
667.        }
668.
669.        private void tb33_TextChanged(object sender,
   EventArgs e)
670.        {
671.
672.            Q[3, 3] = Double.Parse(tb00.Text);
673.        }
674.
675.        private void btnDistanceToChessboard_Click(object
   sender, EventArgs e)
676.        {
677.            lTypeOfFocalLength.Text = "mm";
678.            StartStereoCapturing(_capture,
   _cameraIndex[0], _capture2, _cameraIndex[1], out _capture, out
   _capture2);
679.
   AddCaptureParams(CountDistanceToChessboardProcess);
680.        }
681.
682.        private void
   CountDistanceToChessboardProcess(object sender, EventArgs e)
683.        {

```

```

684.             CaptureStereoImages(_capture, _capture2, out
685.             _left, out _right);
686.             Mat displayLeftMat = _left.Clone();
687.             Mat displayRightMat = _right.Clone();
688.             Mat[] corners = new Mat[] { new Mat(), new
689.             Mat() };
690.             bool find1 =
691.                 CvInvoke.FindChessboardCorners(displayLeftMat, _boardSize,
692.                 corners[0]);
693.             bool find2 =
694.                 CvInvoke.FindChessboardCorners(displayRightMat, _boardSize,
695.                 corners[1]);
696.             double distance = -2;
697.             if (find1 && find2)
698.             {
699.                 CvInvoke.DrawChessboardCorners(displayLeftMat, _boardSize,
700.                 corners[0], find1);
701.                 CvInvoke.DrawChessboardCorners(displayRightMat, _boardSize,
702.                 corners[1], find2);
703.                 distance =
704.                     FindDistanceToChessBoard(corners[0], corners[1], _blockSize,
705.                     _blockSize, _focalLength, _baseLine);
706.             }
707.             CvInvoke.PutText(displayLeftMat, "distance =
708.                 " + distance, new Point(10, displayLeftMat.Height - 30),
709.                 FontFace.HersheySimplex, 0.5, new
710.                 Bgr(Color.Red).MCvScalar, 2);
711.             ShowStereoImages(displayLeftMat,
712.                 displayRightMat);
713.         }
714.         private static double
715.             FindDistanceToChessBoard(Mat leftCorners, Mat rightCorners,
716.             double sizeOfBlock, Size chessboardSize,
717.             double focalLength, double baseLine)
718.         {
719.             int amountOfPoints = chessboardSize.Width *
720.                 chessboardSize.Height;
721.             if (leftCorners == null || rightCorners ==
722.                 null || chessboardSize == null)
723.             {
724.                 return -1;
725.             }
726.             PointF[][] corners = new PointF[2][];
727.             corners[0] = new PointF[amountOfPoints];

```

```

716.         corners[1] = new PointF[amountOfPoints];
717.
718.         leftCorners.CopyTo(corners[0]);
719.         rightCorners.CopyTo(corners[1]);
720.
721.         double[] disparitys = new
722.             double[amountOfPoints];
723.             for (int i = 0; i < amountOfPoints; i++)
724.             {
725.                 disparitys[i] = Math.Abs(corners[1][i].X
726. - corners[0][i].X);
727.             }
728.             double[][] mesurePointsDisparity = new
729.                 double[chessboardSize.Height][];
730.                 for (int i = 0; i < chessboardSize.Height;
731. i++)
732.                 {
733.                     mesurePointsDisparity[i] = new
734.                         double[chessboardSize.Width - 1];
735.                         for (int j = 0; j < chessboardSize.Width
736. - 1; j++)
737.                         {
738.                             double[][] coefficients = new
739.                                 double[chessboardSize.Height - 1][];
740.                                 for (int i = 0; i < coefficients.Length; i++)
741.                                 {
742.                                     coefficients[i] = new
743.                                         double[chessboardSize.Width - 1];
744.                                         for (int j = 0; j <
745. coefficients[i].Length; j++)
746.                                         {
747.                                             PointF[][] pointsL = new PointF[][] {
748.                                                 new PointF[2], new PointF[2] };
749.                                                 PointF[][] pointsR = new PointF[][] {
750.                                                 new PointF[2], new PointF[2] };
751.                                                 pointsL[0][0] =
752. GetElementFromArrayAsFromMatrix(corners[0], i, j,
chessboardSize.Width);

```

```

747.           pointsL[0][1] =
    GetElementFromArrayAsFromMatrix(corners[0], i, j + 1,
    chessboardSize.Width);
748.           pointsL[1][0] =
    GetElementFromArrayAsFromMatrix(corners[0], i + 1, j,
    chessboardSize.Width);
749.           pointsL[1][1] =
    GetElementFromArrayAsFromMatrix(corners[0], i + 1, j + 1,
    chessboardSize.Width);
750.
751.           pointsR[0][0] =
    GetElementFromArrayAsFromMatrix(corners[1], i, j,
    chessboardSize.Width);
752.           pointsR[0][1] =
    GetElementFromArrayAsFromMatrix(corners[1], i, j + 1,
    chessboardSize.Width);
753.           pointsR[1][0] =
    GetElementFromArrayAsFromMatrix(corners[1], i + 1, j,
    chessboardSize.Width);
754.           pointsR[1][1] =
    GetElementFromArrayAsFromMatrix(corners[1], i + 1, j + 1,
    chessboardSize.Width);
755.           coefficients[i][j] =
    (CountPixelsToMM(pointsL, sizeOfBlock) +
756.           CountPixelsToMM(pointsR,
    sizeOfBlock)) / 2;
757.       }
758.   }
759.
760.   double[][] distances = new
    double[chessboardSize.Height][];
761.   for (int i = 0; i < distances.Length; i++)
762.   {
763.       distances[i] = new
    double[chessboardSize.Width - 1];
764.       for (int j = 0; j < chessboardSize.Width
    - 1; j++)
765.       {
766.           if (i < chessboardSize.Height - 1)
767.           {
768.               distances[i][j] = baseLine *
    focalLength / (mesurePointsDisparity[i][j] /**
    coefficients[i][j]*);
769.           }
770.           else
771.           {
772.               distances[i][j] = baseLine *
    focalLength / (mesurePointsDisparity[i][j]/* */

```

```

773.         coefficients[chessboardSize.Height - 2][chessboardSize.Width - 2]*/);
774.     }
775.   }
776. }
777.
778.     double distance = 0;
779.     for (int i = 0; i < distances.Length; i++)
780.     {
781.       distance += distances[i].Sum();
782.     }
783.     distance /= distances.Length;
784.     return distance / 1000;
785.   }
786.
787.     private static double CountPixelsToMM(PointF[][][]
788.     points, double sizeOfBlockMM)
789.     {
790.       double[] lengths = new double[6];
791.       lengths[0] = sizeOfBlockMM /
792.         Math.Sqrt(Math.Pow((points[0][0].X - points[0][1].X), 2) +
793.           Math.Pow(points[0][0].Y -
794.             points[0][1].Y, 2));
795.       lengths[1] = sizeOfBlockMM /
796.         Math.Sqrt(Math.Pow((points[0][0].X - points[1][0].X), 2) +
797.           Math.Pow(points[0][0].Y -
798.             points[1][0].Y, 2));
799.       lengths[2] = sizeOfBlockMM * Math.Sqrt(2) /
800.         Math.Sqrt(Math.Pow((points[0][0].X - points[1][1].X), 2) +
801.           Math.Pow(points[0][0].Y -
802.             points[1][1].Y, 2));
803.     }
804.
805.     return lengths.Average();
806.   }
807.
```

```

808.         private void tbCameraIndex1_TextChanged(object
809.             sender, EventArgs e)
810.             {
811.                 if (tbCameraIndex1.Text == "")
812.                     {
813.                         return;
814.                     }
815.                 _cameraIndex[0] =
816.                     Int32.Parse(tbCameraIndex1.Text);
817.             }
818.         private void tbCameraIndex2_TextChanged(object
819.             sender, EventArgs e)
820.             {
821.                 if (tbCameraIndex2.Text == "")
822.                     {
823.                         return;
824.                     }
825.                 _cameraIndex[1] =
826.                     Int32.Parse(tbCameraIndex2.Text);
827.             }
828.         private void tbFocalLength_TextChanged(object
829.             sender, EventArgs e)
830.             {
831.                 if (tbFocalLength.Text == "")
832.                     {
833.                         return;
834.                     }
835.                 _focalLength =
836.                     Double.Parse(tbFocalLength.Text);
837.             }
838.         private void tbBaseLine_TextChanged(object
839.             sender, EventArgs e)
840.             {
841.                 if (tbBaseLine.Text == "")
842.                     {
843.                         return;
844.                     }
845.                 private void
846.                     tbWidthOfChessboard_TextChanged(object sender, EventArgs e)
847.                     {
848.                         if (tbWidthOfChessboard.Text == "")
849.                         {

```

```

850.             int h = _boardSize.Height;
851.             int w =
852.                 Int32.Parse(tbWidthOfChessboard.Text);
853.             _boardSize = new Size(w, h);
854.         }
855.         private void
856.             tbHeightOfChessboard_TextChanged(object sender, EventArgs e)
857.             {
858.                 if (tbHeightOfChessboard.Text == "")
859.                 {
860.                     return;
861.                 }
862.                 int w = _boardSize.Width;
863.                 int h =
864.                     Int32.Parse(tbWidthOfChessboard.Text);
865.                     _boardSize = new Size(w, h);
866.             }
867.         private void tbBlockSize_TextChanged(object
868.             sender, EventArgs e)
869.             {
870.                 if (tbBlockSize.Text == "")
871.                 {
872.                     return;
873.                 }
874.                 _blockSize = float.Parse(tbBlockSize.Text);
875.             }
876.         private void CalibratePhoto_Click(object sender,
877.             EventArgs e)
878.             {
879.                 _makeCalibratingPhoto = true;
880.                 CalibratePhoto.Text = "Photo for calibrate "
881.                     + (_imageBufferLength - _currentImageIndex - 1);

```

Темы курсовых работ (итоговых индивидуальных заданий)

1. Гистограммы: уравнивание и др. Использование гистограммных статистик для улучшения изображения. Линейная и нелинейная коррекции яркости.
2. Пространственная фильтрация. Сглаживающие пространственные фильтры. Пространственные фильтры повышения резкости. Использование производных первого порядка для (нелинейного) повышения резкости изображений: градиент
3. Фильтрация в частотной области. Частотные фильтры сглаживания изображения. Повышения резкости изображений частотными фильтрами. Фильтры Баттервортса. Гауссовые фильтры.
4. Восстановление изображений. Типы шума, Шум и гистограммы. Оценка шума. Подавление шума. Фильтрация Винера (Фильтрация методом минимизации среднего квадрата отклонения). Среднегеометрический фильтр
5. Выравнивание освещенности – алгоритм Retinex.
6. Реконструкция изображения по проекциям. Принципы компьютерной томографии. Проекции и преобразование Радона.
7. Обработка цветных изображений. Сглаживание и повышение резкости. Сегментация изображения, основанная на цвете. Сегментация изображения, основанная на цвете HSI, RGB. О краях и регионах
8. Пирамиды изображений. Субполосное кодирование. Преобразование Хаара
9. Морфологическая обработка изображений. Основные морфологические алгоритмы. Выделение границ. Заполнение дырок. Выделение связных компонент и др.
10. Сегментация изображений. Обнаружение точек, линий, перепадов и контуров. Связывание контуров и нахождение границ. Функция Canny. Обобщенное преобразование Хафа и его применение для поиска объектов.
11. Пороговая обработка. Обработка с глобальным порогом, с несколькими порогами, с переменным порогом.
12. Сегментация на отдельные области. Сегментация по морфологическим водоразделам. Использование движения при сегментации
13. Современные алгоритмы сегментации, метод QuickShift.
14. Использование МСП для сегментации изображений, алгоритмы TextronBoost, Semantic Textron Forests.
15. Алгоритмы сегментации, основанные на разрезах графов (Intelligent scissors, Normalized cuts, Interactive segmentation by graph cuts).
16. Распознавание объектов.
17. Нейронные сети. Модели свёрточной нейронной сети для классификации изображения. Модель Inception v-3
18. Стереокамеры. Стереогеометрия.
19. Задача сопоставление изображений. Понятие точечной особенности. Детекторы углов Харриса, LOG, DOG, Harris-Laplacian.
20. Детекторы областей (IBR, MSER).
21. Детектирование оставленных предметов.
22. Движение изображения. Анализ частичного и полного оптического потока между кадрами изображения. Функция calcOpticalFlowPyrLK и применение.
23. Отслеживание.
24. Склейка видеоизображений, алгоритм Пуассона.

Библиографический список

1. R.Szeliski, "Computer Vision: Algorithms and Applications"
<http://szeliski.org/Book/> (В свободном доступе)
2. Д. Форсайт, Ж. Понс. "Компьютерное зрение. Современный подход", Вильямс, 2004. <http://www.ozon.ru/context/detail/id/1635123/>
3. Р. Гонсалес, Р. Будс, "Цифровая обработка изображений", Техносфера, 2006. <http://www.technosphera.ru/77.html>
4. Петров М.Н. Компьютерная графика. Учебник для вузов. 3-е изд. 2011, 544 с. ISBN 978-5-459-00809-8.
5. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие.-М.: Изд-во МГУ, 2009. – 77 с. ISBN 978-5-211-05702-9
6. Mark D. Pesce Programming Microsoft Direct Show for Digital Video/Television Microsoft Press. 2003. ISBN: 0735618216
7. Горнаков С.Г. DirectX 9. Уроки программирования на C++ БХВ-Петербург, 2005.- 400 с. ISBN: 5-94157-482-7
8. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. ДМК Пресс, 2010.- 234с.
9. Молодяков С.А. Проектирование специализированных цифровых видеокамер. / СПб. : Изд-во Политехн. ун-та, 2016 .— 286 с. — ISBN 978-5-7422-5334-1

Электронные ресурсы

1. <http://opencv.org/>
2. <https://courses.graphicon.ru/main/vision/2009/lectures>
3. <http://www.cc.gatech.edu/~irfan/p/2011-Grundmann-AVSWROCP.pdf>
4. <http://en.wikipedia.org/wiki/RANSAC>
5. <https://habrahabr.ru/post/265155/>
6. <https://www.ffmpeg.org/>
7. <http://www.nersc.gov/nusers/help/tutorials/openmp/>
8. <http://scv.bu.edu/tutorials/OpenMP/>
9. Начало знакомства с CUDA <http://ru.wikipedia.org/wiki/CUDA>

Молодяков Сергей Александрович,

«ЭВМ И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА»
ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ПРОГРАММИРОВАНИЕ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

Лицензия ЛР № 020593 от 07.08.97

Налоговая льгота – Общероссийский классификатор продукции
ОК 005-93, т. 2; 95 3005 – учебная литература

Подписано в печать _____. 2018. Формат 60×84/16. Печать цифровая
Усл. печ. л. ___. Уч.-изд. л. _____. Тираж . Заказ

Отпечатано с готового оригинал-макета, предоставленного авторами
в цифровом типографском центре Издательства Политехнического
университета:

195251, Санкт-Петербург, Политехническая ул., 29.
Тел. (812) 540-40-14
Тел./факс: (812) 927-57-76