Начинаем работать с библиотекой OpenCV

Вадим Писаревский, ведущий инженер компании
ITSEEZ

Содержание

- Общая информация
- Обзор функциональности
- Подготовка к работе
- Первые примеры
- Работы с матрицами
- Более сложная функциональность, примеры

Краткая справка

- Страничка: http://opencv.org
- Фактически, самая популярная библиотека компьютерного зрения.
- Написана на C/C++, исходный код открыт, включает более 1000 функций/алгоритмов.
- Лицензия BSD (разрешается бесплатное использование дома, для учебы, на работе)
- Разрабатывается с 1998г, сначала в Интел, теперь в компании Itseez при активном участии сообщества.
- >13,000,000 загрузок (без учета трафика с github)
- Используется многими компаниями, организациями, ВУЗами, например в Intel, Google, MagicLeap, NVidia, Stanford …

Архитектура и разработка OpenCV

Languages:

C/C++
Python
Java

(Matlab, Ruby, Haskell, C#, Lua, Closure)

Acceleration Technologies:

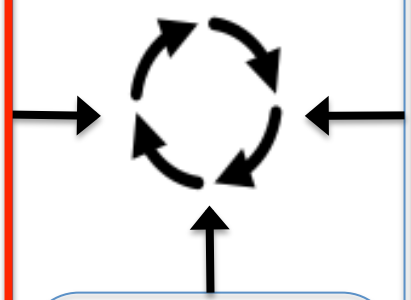
CUDA
OpenCL
parallel for (OpenMP, TBB…)
SIMD (SSE, NEON)

3rd-party libs:

zlib, png, tiff
jpeg, jasper, webp
Gtk, Qt, Cocoa, Win32
OpenNI, V4L … (20+ backends)
intel ipp, tbb
eigen
ffmpeg
Tesseract (OCR)
VTK
libmv
caffe (in progress)

Development:

Maintainers
Contributors



Build & QA:
CMake, Python
Doxygen
Github, Builbot
GTest

Modules:

“main” opencv:
core, imgproc, photo, video, calib3d features2d ml, flann objdetect, shape highgui, viz stitching, superres videostab …

opencv_contrib:
rgbd, tracking
text, reg
face, bioinspired
ximgproc, xphoto
xfeatures2d

Target Arch:

x86/x64
ARM
…

OS’es:

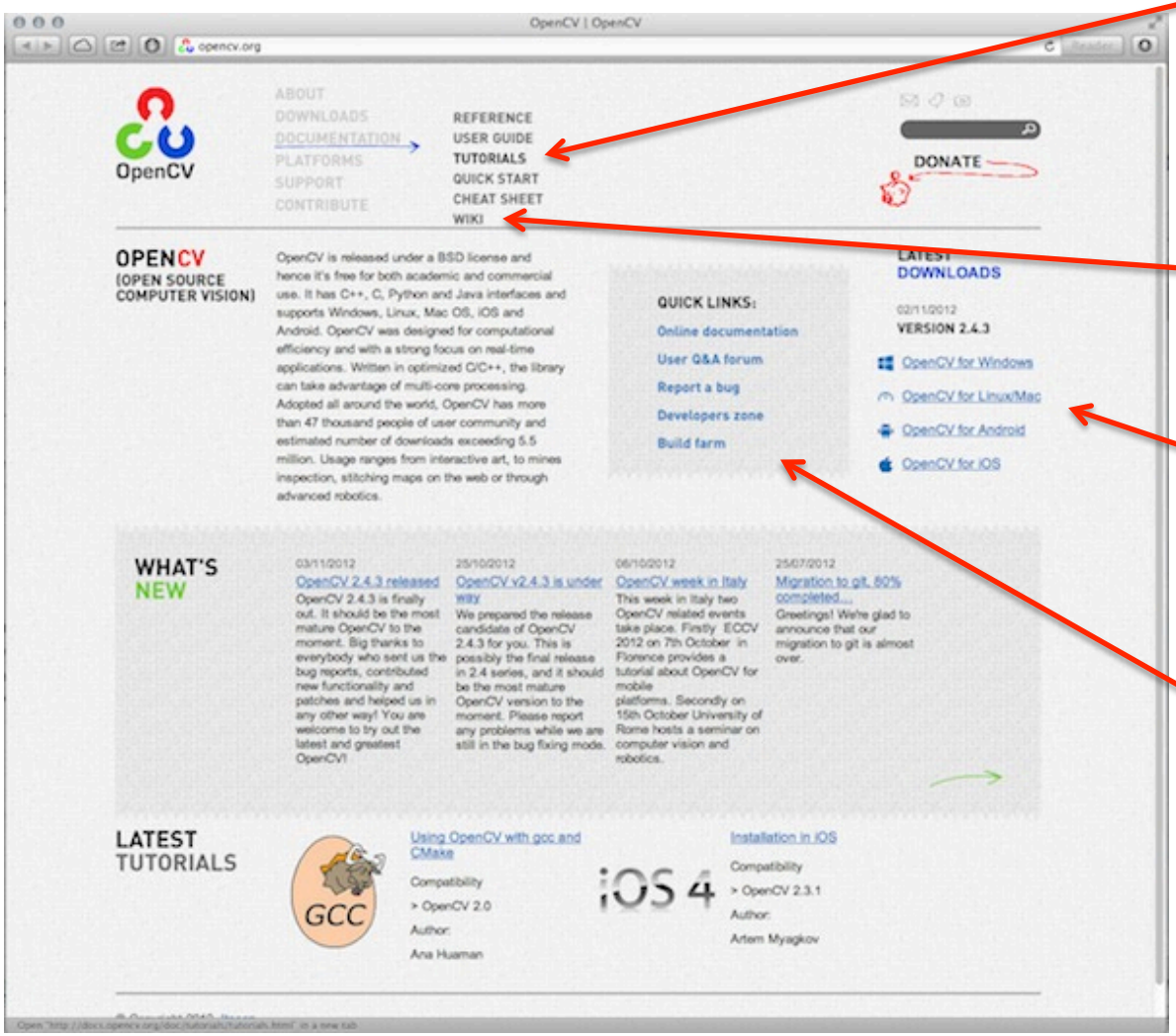
Windows
Linux
Android
OSX
iOS

QNX, BSD, …

Отдельная большая тема

С чего начать?

http://opencv.org:



Читаем уроки,
Распечатываем
cheatsheet.

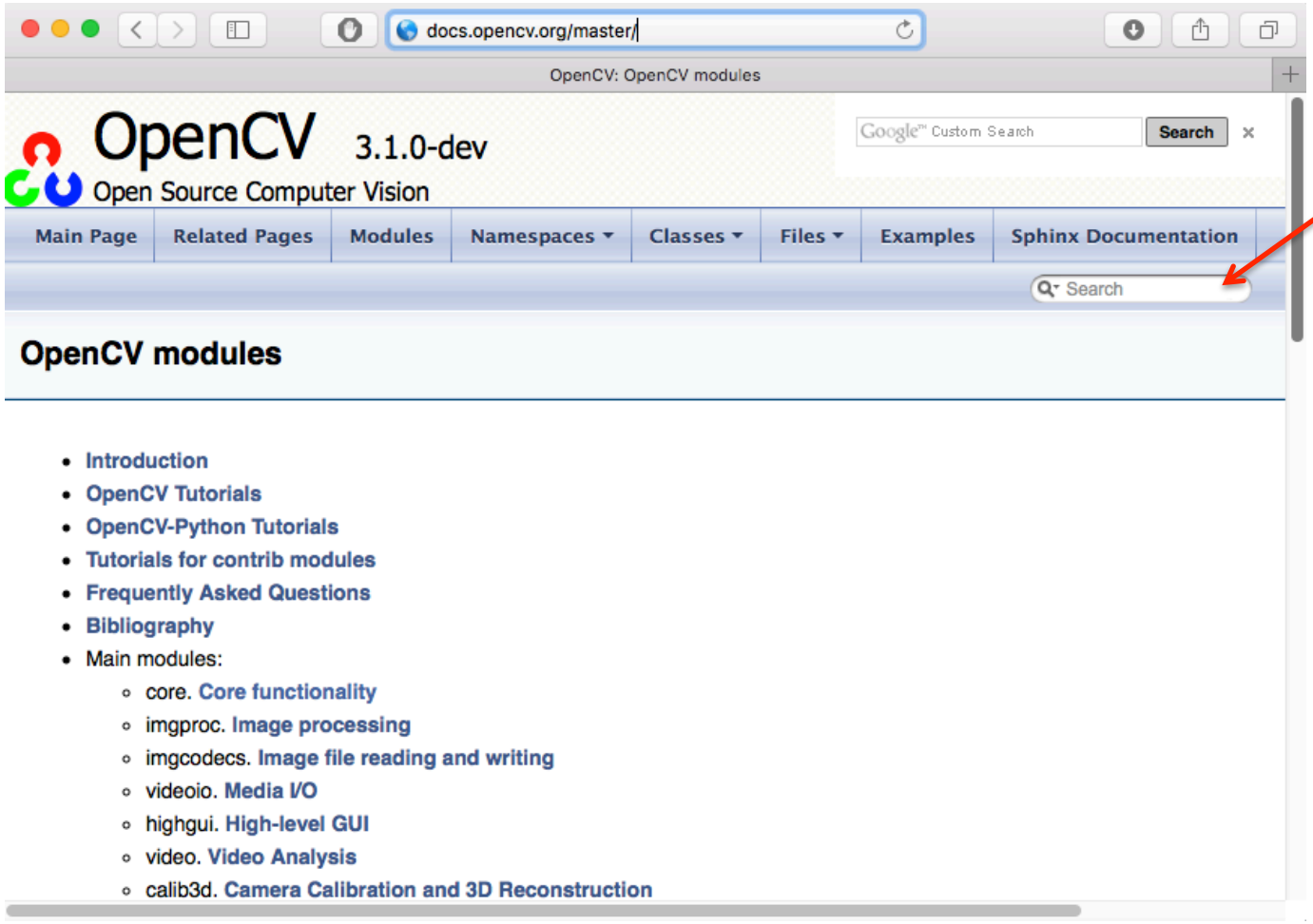
Ссылка WIKI ведет на
сайт для разработчиков
http://code.opencv.org

Качаем

Другие сайты OpenCV
(см. дальше)

Сайт с документацией

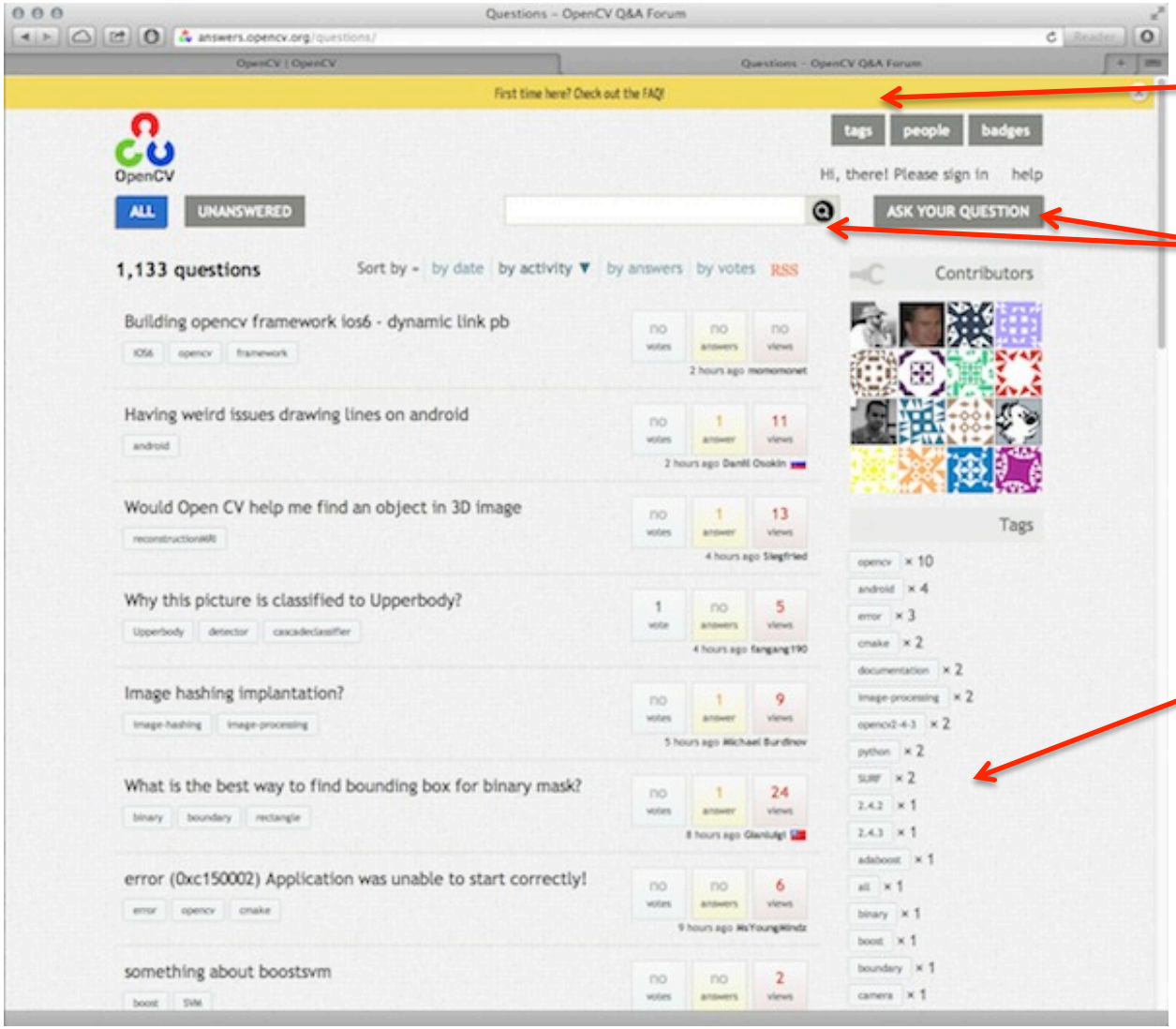
http://docs.opencv.org/master: справочник, руководство, уроки



Поиск по документации.

Задать вопрос, найти ответ

http://answers.opencv.org: сделан по аналогии со StackOverflow



Начинаем с FAQ

1. Поискать ответы
2. Задать свой вопрос

теги

Быстрый старт

1. Устанавливаем компилятор C/C++, Python 2.7.x (http://python.org), NumPy (http://numpy.scipy.org), cmake (http://cmake.org)
2. Клонируем репозиторий с github или качаем архив, например https://github.com/Itseez/opencv/tree/3.0.0, и аналогично opencv_contrib (только для OpenCV 3.x), строим OpenCV и *не инсталируем его*!

```
cmake –D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules …
```

3. Создаем каталог с проектом и кладем туда следующий CMakeList.txt:

```
cmake_minimum_required(VERSION 2.8)
project(myopencv_sample)
find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})
set(the_target "myopencv_sample")
add_executable(${the_target} main.cpp) # add other .cpp
                                       # and .h files here
target_link_libraries(${the_target} ${OpenCV_LIBS})
```

4. Создаем main.cpp (см. дальше). Можно взять один из готовых примеров из opencv/samples/cpp.
5. Указываем cmake, где найти OpenCVConfig.cmake и генерируем проект или Makefile’s.
6. Открываем сгенерированный проект, строим.

Первая программа: находим ребра

```
#include "opencv2/opencv.hpp"

using namespace cv;
int main(int argc, char** argv)
{
    Mat img, gray, edges;
    img = imread(argv[1], 1);
    imshow("original", img);
    cvtColor(img, gray, COLOR_BGR2GRAY);
    GaussianBlur(gray, gray, Size(7, 7), 1.5);
    Canny(gray, edges, 0, 50);
    imshow("edges", edges);
    imwrite(“result.png”, edges);
    waitKey();
    return 0;
}
```

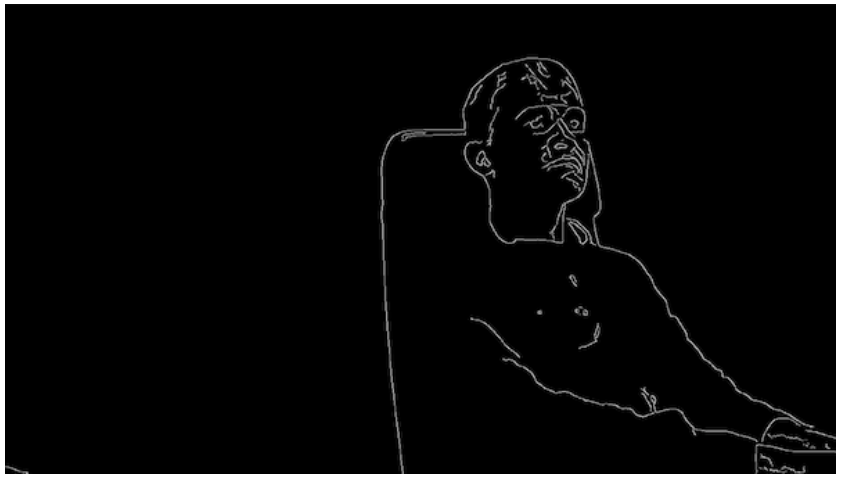
Настраиваем параметры

```
#include "opencv2/opencv.hpp"
using namespace cv;
Mat img, gray, edges;

int smoothness = 15;
static void on_smoothness(int, void*) {
    cvtColor(img, gray, COLOR_BGR2GRAY);
    double sigma = smoothness*0.1 + 1;
    int ksize = cvRound(sigma*4+1)|1;
    GaussianBlur(gray, gray, Size(ksize, ksize), sigma);
    Canny(gray, edges, 0, 50);
    imshow("edges", edges);
}
int main(int argc, char** argv) {
    img = imread(argc > 1 ? argv[1] : "lena.jpg", 1);
    imshow("original", img); imshow("edges", img);
    createTrackbar("smoothness", "edges",
            &smoothness, 30, on_smoothness, 0);
    on_smoothness(0, 0);
    waitKey(); return 0;
}
```

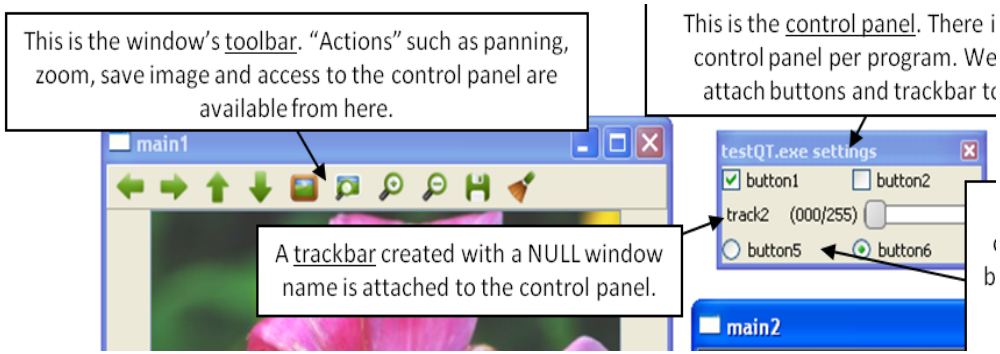
Добавляем видео

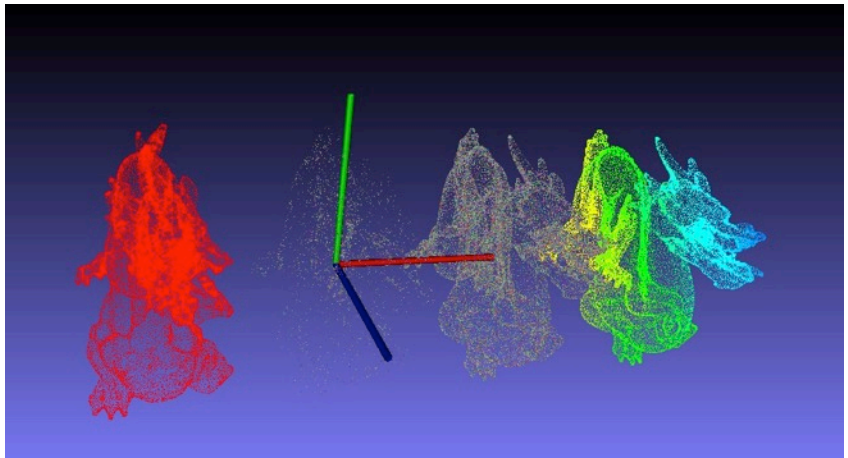
```
#include "opencv2/opencv.hpp”
using namespace cv;
int main(int argc, char** argv) {
    Mat img, gray, edges;
    int smoothness = 15;
    bool firstframe = true;
    VideoCapture cap(0); // use VideoCapture cap(argv[1]) to grab video from the file.
    for(;;) {
        cap >> img; if(img.empty()) break;
        cvtColor(img, gray, COLOR_BGR2GRAY);
        double sigma = smoothness*0.1 + 1;
        int ksize = cvRound(sigma*4+1)|1;
        GaussianBlur(gray, gray, Size(ksize, ksize), sigma);
        Canny(gray, edges, 0, 50);
        imshow("edges", edges);
        if(firstframe) {
            createTrackbar("smoothness", "edges", &smoothness, 30, 0, 0);
            firstframe = false;
        }
        if(waitKey(30) >= 0) break; }
    return 0; }
```



HighGUI (=ui+imgcodec+videoio)

- Окна с “памятью”
- Обработка нажатий клавиш.
- Обработка событий от мыши (см. https://github.com/opencv/opencv/blob/master/samples/cpp/camshiftdemo.cpp)
- Слайдеры.
- Чтение/запись изображений
- Чтение/запись видео
- В случае наличия Qt – много дополнительных средств (тулбар, кнопки, зум, значения пикселей …)
- См. также модуль VIZ для визуализации 3D данных: http://habrahabr.ru/company/itseez/blog/217021/





cv::Mat – многомерный многоканальный массив

cv::Mat A(h, w, CV_8UC3);
- Размеры, step
- Счетчик ссылок (=1)
- Указатель на данные

cv::Mat B = A;
- Размеры, step
- Счетчик ссылок (=2)
- Указатель на данные

cv::Mat C=A(roi);
- Размеры ROI, step
- Счетчик ссылок (=3)
- Указатель на данные

Расположение в памяти матрицы C

RGBRGBRGBRGB**********RGBRGBRGBRGB**********...

cv::Mat – универсальный тип данных

BGR/RGB: CV_8UC3

grayscale/bayer: CV_8UC1

mask: CV_8UC1

HSV/Lab/…: CV_8UC3

hi-quality photo: CV_16UC3, CV_32FC3

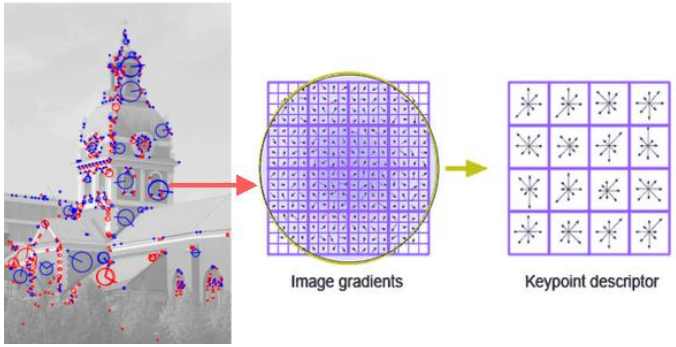
matrix: CV_32F, CV_64F

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

dense motion field: CV_32FC2

depth/RGBD: CV_32F, CV_32FC4

feature descriptors: CV_8U/CV_32F



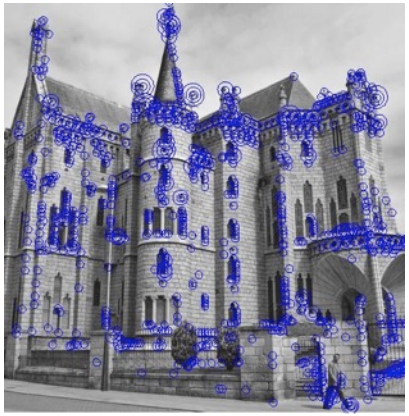
???

CV_MAKETYPE(
CV_8U … CV_64F,
1 .. 512)

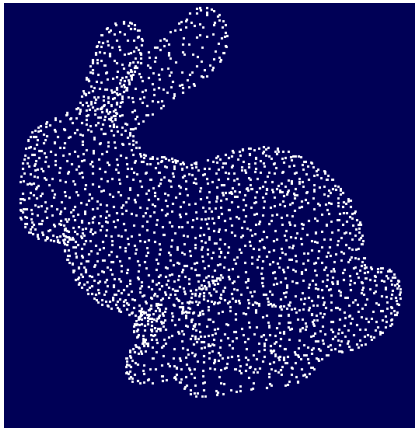
std::vector – еще один универсальный тип

sparse motion field:
vector<Point2f>

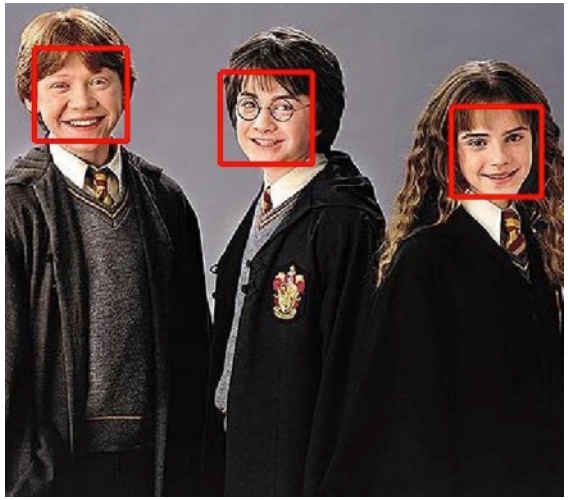
keypoints: vector<KeyPoint>



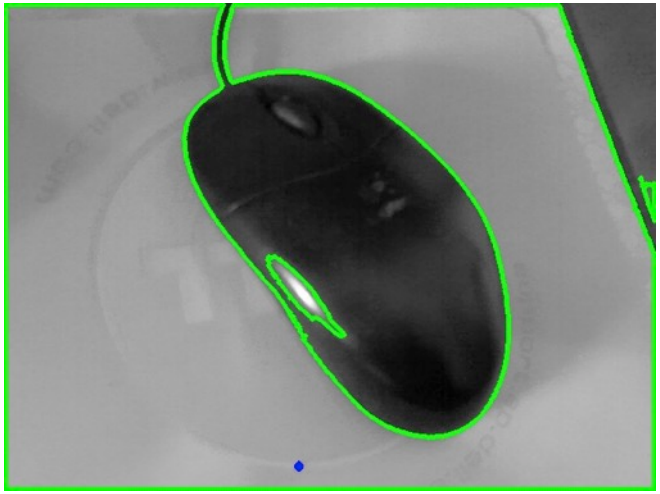
pointcloud: vector<Point3f>



object detection: vector<Rect>,
vector<Detection>



contours: vector<vector<Point> >



Представление некоторых std::vector как cv::Mat

std::vector<Point3f> v(N);

XYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZ

Массив из N точек

cv::Mat a(v);

- Размеры (Nx1), step(=12)
- Счетчик ссылок (отсутствует)
- Указатель на данные

XYZ
XYZ
…
XYZ

Nx1
3-канальное
изображение

```
namespace cv {
CV_EXPORTS_W Scalar mean(InputArray src, InputArray mask = noArray());
}
…
cv::Mat img(…); Scalar mean_brightness = mean(img);
std::vector<Point2f> ptset = …; Scalar mass_center = mean(ptset);
```

Работаем с матрицами

```
Mat M = Mat::zeros(480,640,CV_8UC1); // Создаем 8-битное одноканальное изображение 640x480,
    заполненное нулями
Rect roi(100, 200, 20, 20);   // Определяем ROI
Mat subM = M(roi);            //  “выделяем” ROI в отдельную матрицу
                              //  без копирования
subM.at<uchar>(y,x)=255; // изменяем пиксель в строке y и столбце x ROI
                              // и (x+100, y+200) в исходном изображении
```

```
float a = CV_PI/3;
Mat R = (Mat_<float>(2, 2) << cos(a), -sin(a), sin(a), cos(a)); // заполняем матрицу по элементам
float iRdata[] = {cos(a), sin(a), -sin(a), cos(a)};
Mat iR = Mat(2, 2, CV_32F, R2data); // альтернативный вариант (данные не копируются)
CV_Assert(norm(R*iR, Mat::eye(3, 3, CV_32F), NORM_L2) <= 0.01); // используем матричные выражения
```

```
Mat img = imread("lena.jpg", 1); // читаем картинку как цветную (BGR)
Mat planes[3], noise(img.size(), CV_32F);
cvtColor(img, img, COLOR_BGR2YUV); // конвертируем RGB => YUV
split(img, planes); // разделяем на отдельные цветовые плоскости
randn(noise, Scalar::all(0), Scalar::all(30)); // генерируем гауссовый шум
GaussianBlur(noise, noise, Size(3, 3), 0.5); // сглаживаем его
add(planes[0], noise, planes[0], noArray(), CV_8U); // добавляем шум к яркостной компоненте
merge(planes, 3, img); // объединяем каналы обратно
cvtColor(img, img, COLOR_YUV2BGR); // превращаем картинку снова в BGR
```

core: операции над матрицами, “мини Matlab”

Mat::zeros, Mat::eye, Mat::ones, Mat::setTo, randu, randn – заполнение/ инициализация матриц элементов, объединение и выделение отдельных каналов.
Mat::operator (), Mat::row, Mat::col, Mat::rowRange, Mat::colRange, Mat::diag, Mat::reshape – выделение частей матриц и изменение формы без копирования
Mat::copyTo, Mat::clone, Mat::repeat, vconcat, hconcat, flip, transpose, randShuffle, split, merge, mixChannels – копирование и различные перемешивания
Mat::convertTo, normalize – преобразование к другому диапазону и/или к другому типу данных
add, subtract, multiply, divide, absdiff – поэлементные арифметические операции
bitwise_and, bitwise_or, bitwise_xor, bitwise_not – логические операции
compare, min, max – поэлементное сравнение, минимум, максимум
sum, mean, meanStdDev, norm, minMaxLoc – сбор статистики по матрице, сравнение матриц
gemm, Mat::dot, Mat::cross, solve, eigen, SVD, determinant, trace, solvePoly, solveLP, MinProblemSolver – линейная алгебра, нахождение корней полиномов, решение задач оптимизации
exp, log, sqrt, pow, cartToPolar, polarToCart – стандартные поэлементные математические операции
reduce, sort, sortIdx – операции над строками и столбцами
dft, dct – дискретные ортогональные преобразования

http://docs.opencv.org/opencv_cheatsheet.pdf -

здесь перечислено гораздо больше функций

Перебор элементов

```
// оцениваем “резкость” в выбранном ROI,
// например для реализации автофокуса
float contrast = 0.f;
for(int i = 1; i < subM.rows-1; i++) {
    const uchar* ptr = subM.ptr<uchar>(i);
    for(int j = 1; j < subM.cols-1; j++, ptr++) {
        int dx = ptr[1] – ptr[-1], dy = ptr[subM.step] – ptr[-subM.step];
        contrast += sqrtf((float)(dx*dx + dy*dy));
}
// вариант с итераторами
Mat subsubM = subM(Rect(1, 1, subM.cols-2, subM.rows-2));
Mat_<uchar>::iterator it= subsubM.begin<uchar>(),
                      itEnd = subsubM.end<uchar>();
float contrast = 0.f; int step = (int)subM.step;
for(; it != itEnd; ++it) {
   uchar* ptr = &(*it);
   int dx = ptr[1] – ptr[-1], dy = ptr[step] – ptr[-step];
   contrast += sqrtf((float)(dx*dx + dy*dy));
}
```

FileStorage: запись/чтение структурированных данных

// Записываем данные

```
1. { FileStorage fs("test.yml", FileStorage::WRITE);
2. fs << “intval” << 5 << “realval” << 3.1 << “str” << “ABCDEFGH”;
3. fs << “mtx” << Mat::eye(3,3,CV_32F);
4. fs << “mylist” << “[“ << 1 << 2 << 3 << 4 << 5 << “]”;
5. fs << “date” << “{:” << “month” << 12 << “day” << 31 << “year” << 2015 << “}”;
6. const uchar arr[] = {0, 1, 1, 0, 1, 1, 0, 1};
7. fs << “bitmask” << “[:”; fs.writeRaw(“u”, arr, 8);
8. fs << “]”; }
```

// И считываем их обратно

```
1. { FileStorage fs("test.yml", FileStorage::READ);
2. int intval = (int)fs[“intval”]; double realval = (double)fs[“realval”];
3. string str = (string)fs[“str”];
4. Mat mtx; fs[“mtx”] >> mtx;
5. vector<int> mylist; FileNode mylist_node = fs[“mylist”];
6. size_t n = mylist_node.size(); FileNodeIterator mylist_it = mylist_node.begin();
7. for( i = 0; i < n; i++, ++it) { mylist.push_back((int)*it); }
8. FileNode dn = fs[“date”]; int month = (int)dn[“month”], day=(int)dn[“day”],
   year=(int)dn[“year”];
9. vector<uchar> bitmask; fs[“bitmask”] >> bitmask; }
```

Функциональность основного OpenCV

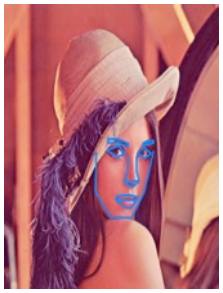
Базовая функциональность

A + B
Ax = B
FFT(A)
<?xml>…

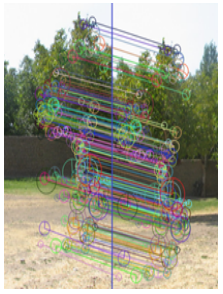
Обработка изображений

Фильтрация, цветовые преобр.

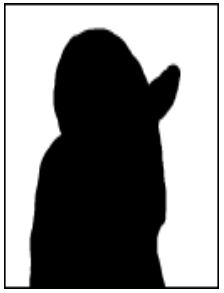
Трансформации



Ребра, контурный анализ

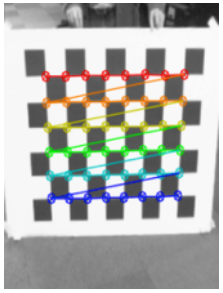


Особые точки

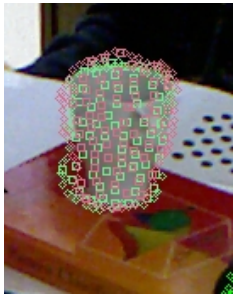


Сегментация

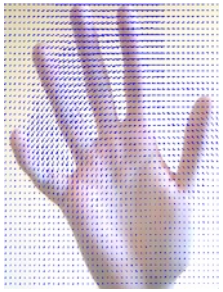
Видео, Стерео, 3D



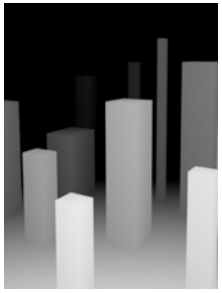
Калибрация камер



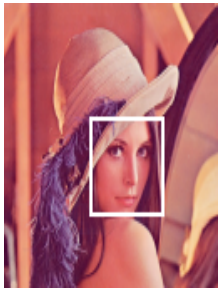
Вычисление положения в пространстве



Оптический поток

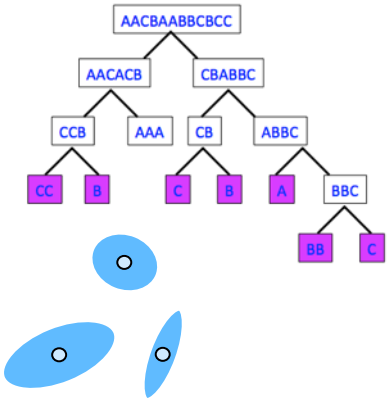


Построение карты глубины



Нахождение объектов

Машинное обучение



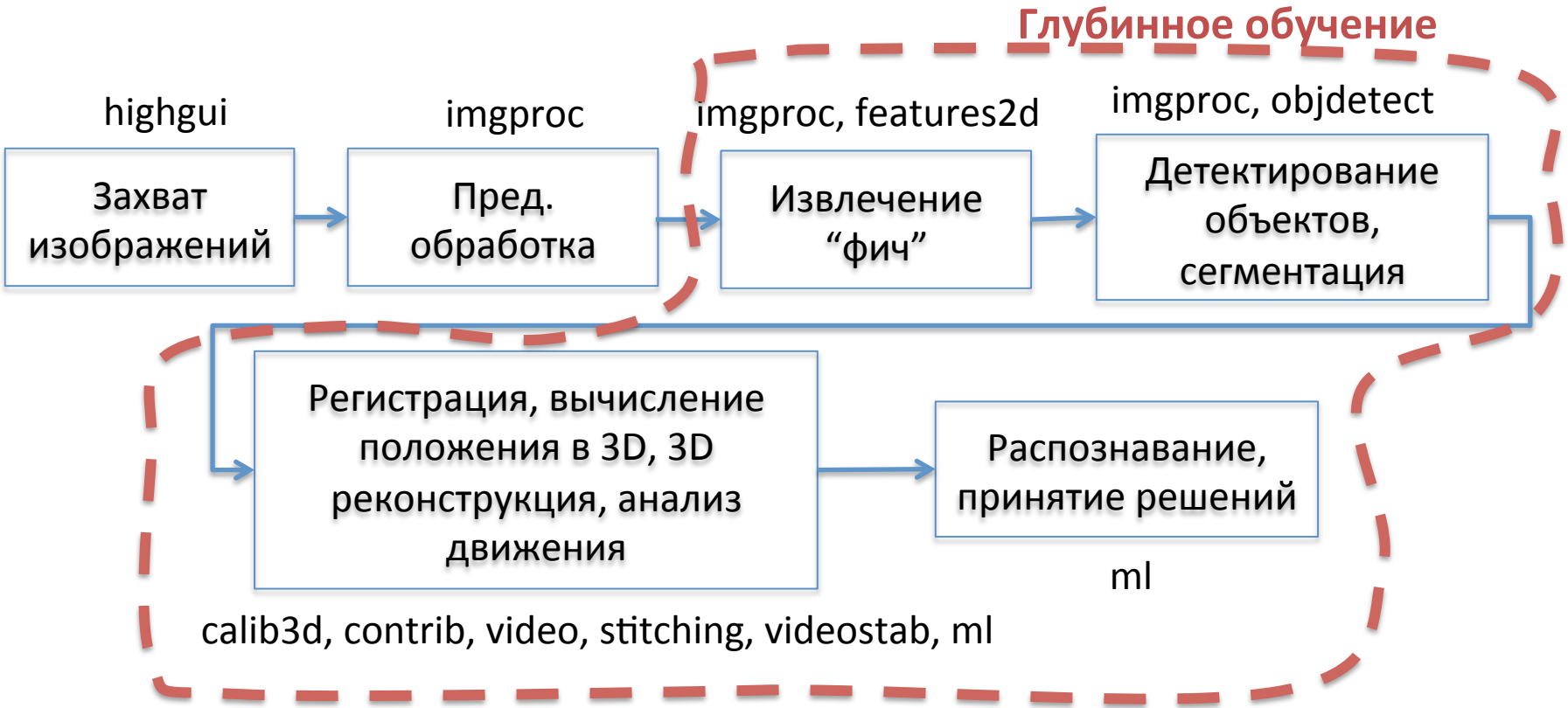
OpenCV в приложениях комп. зрения

OpenCV – базовая, в-целом низкоуровневая библиотека.

Мы создаем строительные блоки, кирпичики для приложений.

Сами приложения предстоит построить пользователям библиотеки.

Общая схема типичного приложения CV



Некоторые основные задачи компьютерного зрения

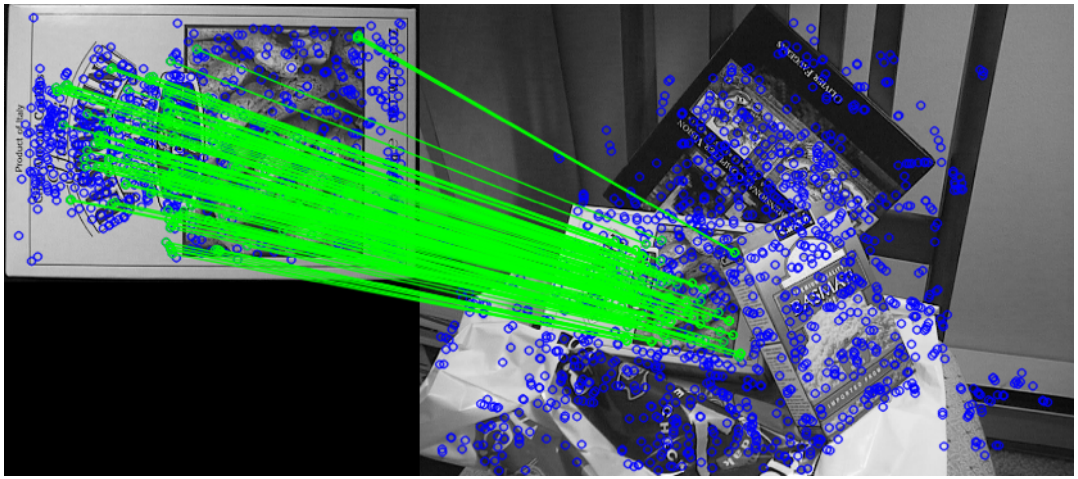
1. Сопоставить 2+ изображений или видеопоследовательностей (регистрация)
2. Найти заданный объект или объекты заданного класса на изображении
3. Найти похожее изображение в базе
4. Определить 3D позу объекта и его частей, 3D структуру всей сцены
5. “Улучшить” изображение
6. Проанализировать движение объектов/частей объектов на изображении, определить движение/положение камеры
7. Разбить изображение на отдельные элементы (сегментация)
8. Проанализировать, что изображено, что происходит.

Каждая из задач имеет десятки вариантов, в зависимости от внешних условий, ограничений, требований по точности и скорости и т.д. Задачи могут комбинироваться, вкладываться друг в друга. Задачи могут решаться специальным железом.

См. также http://www.frontiersincomputervision.com/slides/FCV_Taxo_Zisserman.pdf

http://szeliski.org/Book/

Примеры решение некоторых задач с использованием OpenCV



HDR + вычитание фона используя opencv_contrib/bioinspired

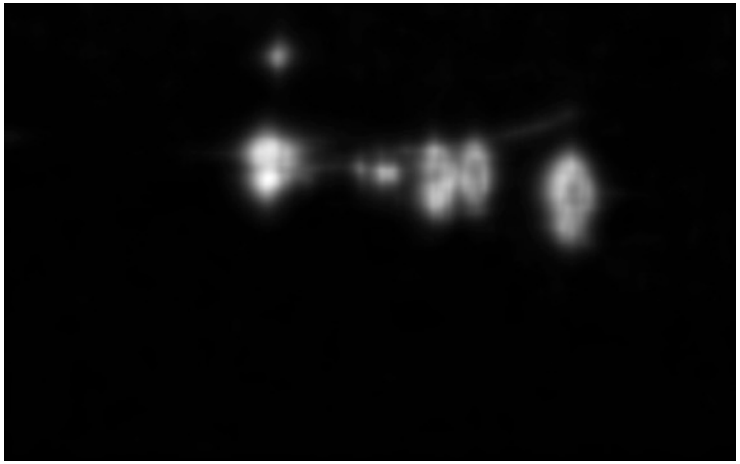
Input video

```
#include “opencv2/bioinspired.hpp”

VideoCapture cap(…);
Ptr<bioinspired::Retina> retina;
Mat frame, parvo, magno;
for(;;) {
  cap >> frame;
  if(!retina)
    retina = bioinspired::createRetina(
                              frame.size());
  retina->run(frame);
  retina->getParvo(parvo);
  retina->getMagno(magno);
}
```

Parvo

Magno



Вопросы?