Open Source Computer Vision Library

Victor Eruhimov
ITSEEZ

Microsoft Computer Vision School
Outline

- Functionality
- Programming with OpenCV
- OpenCV on CPU & GPU
- Mobile vision
OpenCV History

- **Original goal:**
  - Accelerate the field by lowering the bar to computer vision
  - Find compelling uses for the increasing MIPS out in the market

- **Timeline:**

- **Staffing:**
  - Climbed in 1999 to average 7 first couple of years
  - Starting 2003 support declined between zero and one with exception of transferring the machine learning from manufacturing work I led (equivalent of 3 people).
  - Support to zero the couple of years before Willow.
  - 5 people over the last year
OpenCV Functionality Overview

**Image processing**
- General Image Processing
- Transforms
- Fitting
- Optical Flow
- Segmentation

**Video, Stereo, and 3D**
- Camera Calibration
- Pose estimation
- Features
- Depth Maps
- Object detection
OpenCV Architecture and Development

Languages:
- C
- C++
- Python
- CUDA
- JAVA (plans)

Technologies:
- CUDA
- SSE
- TBB

3rd party libs:
- Eigen
- IPP
- Jasper
- JPEG, PNG
- OpenNI
- QT
- TBB
- VideoInput

Development:
- Maintainers
- Contributors

Modules:
- Core
- ImgProc
- HighGUI
- GPU
- ML
- ObjDetect
- Video
- Calib3D
- Features2D
- FLANN

QA:
- Buildbot
- Google Tests

Target archs:
- X36
- X64
- ARM
- CUDA

Target OS:
- Windows
- Linux
- Mac OS
- Android
OpenCV License

• Based on BSD license
• Free for commercial and research use
• Does not force your code to be open
• You need not contribute back
  – But you are very welcome to contribute back!
OpenCV sponsors

- Willow Garage
- NVIDIA
- Intel
- Google
Where is OpenCV Used?

- Academic and Industry Research
- Security systems
- Image/video search and retrieval
- Structure from motion in movies
- Machine vision factory production inspection systems
- Automatic Driver Assistance Systems
- Safety monitoring (Dam sites, mines, swimming pools)
- Robotics

Well over 3M downloads!
Robotics Operation System

- Meta operating system for robotics
- Obtain, build, write, and run code across multiple computers, and multiple robots

Hello, I am Texai418

Hi, I'm a PR2 Beta
Usage examples: ROS

- Imagery infrastructure
- Camera calibration
- Object detection
- Pose estimation
- Human awareness
Hackathons
3D Processing: PCL

- Point Cloud Library
  - http://pointclouds.org

Misc, stats:
- 35 releases already (0.1.x → 0.9.9)
- over 100 classes
- over 80k lines of code (PCL, ROS interface, Visualization)
- young library: only 12 months of development so far, but we had code lying around for 3-5 years
- external dependencies on eigen, cminpack, FLANN
PCL: Finding Normals

```cpp
p.setInputCloud (data);
p.setInputNormals (normals);
p.SetRadiusSearch (0.01);
```
PCL: Filtering by surface curvature
PCL:
Using 3D features to classify surface types
OpenCV Czar
Outline

• OpenCV Overview
• Functionality
  • Programming with OpenCV
  • OpenCV on CPU & GPU
  • Mobile vision
How to choose which algorithms to put into the library?

- Well established
- Works on any data
  - Productized by a commercial company
  - Patented
- Easy to reproduce

VS

- Cutting edge
- Works on Lenna only
- Have to hallucinate the missing pieces
  - And then it works on Lenna only
Imgproc 2

Segmentation

Shape analysis
Features2d contents

**Detection**

- SIFT
- SURF
- FAST
- STAR
- MSER
- HARRIS
- GFTT (Good Features To Track)

**Description**

- SIFT
- SURF
- Calonder
- Ferns
- One way
- HoG

**Matching**

**Matchers available**

- BruteForce
- FlannBased
- BOW

**Matches filters**

(under construction)

- Cross check
- Ratio check
Detector testbench

• Measures of detector repeatability are taken from

• Test images are taken from
  http://www.robots.ox.ac.uk/~vgg/data/data-aff.html

• Testbench is located in
  opencv_extra/testdata/cv/detectors_descriptors_evaluation/detectors
Descriptor testbench

- Test images are taken from [http://www.robots.ox.ac.uk/~vgg/data/data-aff.html](http://www.robots.ox.ac.uk/~vgg/data/data-aff.html)
- Testbench is located in `opencv_extra/testdata/cv/detectors_descriptors_evaluation/descriptors`
Calib3d module

- Camera calibration
- 3D -> 2D projection
- Homography in 2D and 3D
- PnP problem solver
- Stereo vision
- Fundamental matrix
- Template detectors
Pinhole camera model

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
= R
\begin{pmatrix}
X_0 \\
Y_0 \\
Z_0
\end{pmatrix}
+ t
\]

\[
x' = \frac{X}{Z}
\]

\[
y' = \frac{Y}{Z}
\]

\[
u = f_x x' + c_x
\]

\[
v = f_y y' + c_y
\]
Camera Calibration

See samples/cpp/calibration.cpp

3D view of checkerboard

Un-distorted image
Stereo

• Once the left and right cameras are calibrated internally (intrinsics) and externally (extrinsics), we need to rectify the images.
Homography

\[ \tilde{u} = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + h_{33}} \]

\[ \tilde{v} = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + h_{33}} \]

\[ \begin{pmatrix} \tilde{u}w \\ \tilde{v}w \\ w \end{pmatrix} = H \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \]
Random Sample Consensus

• Do \( n \) iterations until \( \#\text{inliers} > \text{inlierThreshold} \)
  – Draw \( k \) matches randomly
  – Find the transformation
  – Calculate inliers count
  – Remember the best solution

Problem: \#matches, \#inliers, \( k \) matches. How many iterations of RANSAC do you need to get the right answer?

The number of iterations required ~ \( C \cdot \left( \frac{\#\text{matches}}{\#\text{inliers}} \right)^k \)
CLASSIFICATION / REGRESSION

(new) Fast Approximate NN (FLANN)
Naïve Bayes
CART
Random Forests

(new) Extremely Random Trees
(new) Gradient Boosting Trees
Statistical Boosting, 4 flavors
SVM
Face Detector

CLUSTERING
K-Means
EM
(Mahalanobis distance)

TUNING/VALIDATION
Cross validation
Bootstrapping
Variable importance
Sampling methods
ML Lib Example:
Boosting: Face Detection with Viola-Jones Rejection Cascade

In samples/cpp, see:
Multicascadeclassifier.cpp
Pedestrian Detection: HOG Descriptor

• Object shape is characterized by distributions of:
  – Gradient magnitude
  – Edge/Gradient orientation

• Grid of orientation histograms
Object detection

Tracking

2D
CamShift();
MeanShift();

calcOpticalFlowPyrLK()
Also see dense optical flow:
calcOpticalFlowFarneback()

3D
KalmanFilter::
Posit();

38
Gary Bradski (c) 2008
Useful OpenCV Links

OpenCV Wiki:
http://opencv.willowgarage.com/wiki

OpenCV Code Repository:
svn co https://code.ros.org/svn/opencv/trunk/opencv

New Book on OpenCV:
http://oreilly.com/catalog/9780596516130/

Or, direct from Amazon:

Code examples from the book:
http://examples.oreilly.com/9780596516130/

Documentation
http://opencv.willowgarage.com/documentation/index.html
Outline

• OpenCV Overview
• Functionality
• Programming with OpenCV
• OpenCV on CPU & GPU
• Mobile vision
## Key OpenCV Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point_</td>
<td>Template 2D point class</td>
</tr>
<tr>
<td>Point3_</td>
<td>Template 3D point class</td>
</tr>
<tr>
<td>Size_</td>
<td>Template size (width, height) class</td>
</tr>
<tr>
<td>Vec</td>
<td>Template short vector class</td>
</tr>
<tr>
<td>Scalar</td>
<td>4-element vector</td>
</tr>
<tr>
<td>Rect</td>
<td>Rectangle</td>
</tr>
<tr>
<td>Range</td>
<td>Integer value range</td>
</tr>
<tr>
<td>MatND</td>
<td>Multi-dimensional dense array</td>
</tr>
<tr>
<td>SparseMat</td>
<td>Multi-dimensional sparse array</td>
</tr>
<tr>
<td>Ptr</td>
<td>Template smart pointer class</td>
</tr>
</tbody>
</table>

Main Structures
cv::Mat and std::vector

std::vector<Point3f>
XYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZXYZ

A vector of n points

cv::Mat
• Image parameters
• Reference counter
• Pointer to data

RGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRB

Nx1 3-channel image
Mat M(480,640,CV_8UC3); // Make a 640x480 img
Rect roi(100,200, 20,40); // Rectangle
Mat subM = M(roi); // Take a sub region,
// no copy is done

Mat_<Vec3b>::iterator it= subM.begin<Vec3b>(),
    itEnd = subM.end<Vec3b>();
//Zero out pixels in subM where blue > red
for(; it != itEnd; ++it)
    if( (*it)[0] > (*it)[2]) (*it)[0] = 0;
Matrix Manipulation

```
src.copyTo(dst)      Copy matrix to another one
src.convertTo(dst,type, scale, shift) Scale and convert to another datatype
m.clone()            Make deep copy of a matrix
m.reshape(nch, nrows) Change matrix dimensions and/or number of channels without copying data
m.row(i), m.col(i)   Take a matrix row/column
m.rowRange(Range(i1,i2)) Take a matrix row/column span
m.colRange(Range(j1,j2))
m.diag(i)            Take a matrix diagonal
m.Range(i1,i2), Range(j1,j2), Take a submatrix
m(roi)
m.repeat(ny, nx)     Make a bigger matrix from a smaller one
flip(src, dst, dir)  Reverse the order of matrix rows and/or columns
split(...)           Split multi-channel matrix into separate channels
merge(...)           Make a multi-channel matrix out of the separate channels
mixChannels(...)     Generalized form of split() and merge()
randShuffle(...)     Randomly shuffle matrix elements
```

Example 1. Smooth image ROI in-place
   Mat imgroi = image(Rect(10, 20, 100, 100));
   GaussianBlur(imgroi, imgroi, Size(5, 5), 1.2, 1.2);
Example 2. Somewhere in a linear algebra algorithm
   m.row(i) += m.row(j)*alpha;
Example 3. Copy image ROI to another image with conversion
   Rect r(1, 1, 10, 20);
   Mat dstroi = dst(Rect(0,10,r.width,r.height));
   src(r).convertTo(dstroi, dstroi.type(), 1, 0);
```
Simple Matrix Operations

- `add()`, `subtract()`, `multiply()`, `divide()`, `absdiff()`, `bitwise_and()`, `bitwise_or()`, `bitwise_xor()`, `max()`, `min()`, `compare()`
- correspondingly, addition, subtraction, element-wise multiplication ... comparison of two matrices or a matrix and a scalar.

Example. Alpha compositing function:
```c
void alphaComposite(const Mat& rgba1,
                    const Mat& rgba2, Mat& rgba_dest)
{
    Mat a1(rgba1.size(), rgba1.type()), ra1;
    Mat a2(rgba2.size(), rgba2.type());
    int mixChannels[] = {3, 0, 3, 1, 3, 2, 3, 3};
    mixChannels(&rgba1, 1, &a1, 1, mixChannels, 4);
    mixChannels(&rgba2, 1, &a2, 1, mixChannels, 4);
    subtract(Scalar::all(255), a1, ra1);
    bitwise_or(ra1, Scalar(0, 0, 255), a1);
    bitwise_or(a2, Scalar(0, 0, 255), a2);
    multiply(a2, ra1, a2, 1. / 255);
    multiply(a1, rgba1, a1, 1. / 255);
    multiply(a2, rgba2, a2, 1. / 255);
    add(a1, a2, rgba_dest);
}
```

- `sum()`, `mean()`, `meanStdDev()`, `norm()`, `countNonZero()`, `minMaxLoc()`
- various statistics of matrix elements.
- `exp()`, `log()`, `pow()`, `sqrt()`, `cartToPolar()`, `polarToCart()`
- the classical math functions.
- `scaleAdd()`, `transpose()`, `gemm()`, `invert()`, `solve()`, `determinant()`, `trace()` `eigen()` `SVD`.
- the algebraic functions + SVD class.
- `dft()`, `idft()`, `dct()`, `idct()`
- discrete Fourier and cosine transformations

For some operations a more convenient algebraic notation can be used, for example:
```c
Mat delta = (J.t())*J + lambda*Mat::eye(J.cols, J.cols, J.type())
           .inv(CV_SVD)*(J.t())*err;
```

implements the core of Levenberg-Marquardt optimization algorithm.
New C++ API: Usage Example

Focus Detector

**C:**

double calcGradients(const IplImage *src, int aperture_size = 7)
{
    CvSize sz = cvGetSize(src);
    IplImage* img16_x = cvCreateImage(sz, IPL_DEPTH_16S, 1);
    IplImage* img16_y = cvCreateImage(sz, IPL_DEPTH_16S, 1);
    
    cvSobel(src, img16_x, 1, 0, aperture_size);
    cvSobel(src, img16_y, 0, 1, aperture_size);
    
    IplImage* imgF_x = cvCreateImage(sz, IPL_DEPTH_32F, 1);
    IplImage* imgF_y = cvCreateImage(sz, IPL_DEPTH_32F, 1);
    
    cvScale(img16_x, imgF_x);
    cvScale(img16_y, imgF_y);
    
    IplImage* magnitude = cvCreateImage(sz, IPL_DEPTH_32F, 1);
    cvCartToPolar(imgF_x, imgF_y, magnitude);
    double res = cvSum(magnitude).val[0];
    
    cvReleaseImage(&magnitude);
    cvReleaseImage(&imgF_x);
    cvReleaseImage(&imgF_y);
    cvReleaseImage(&img16_x);
    cvReleaseImage(&img16_y);
    return res;
}

**C++:**

double contrast_measure(const Mat& img)
{
    Mat dx, dy;
    
    Sobel(img, dx, 1, 0, 3, CV_32F);
    Sobel(img, dy, 0, 1, 3, CV_32F);
    magnitude(dx, dy, dx);
    
    return sum(dx)[0];
}
Simple Image Processing

- `filter2D()`                      Non-separable linear filter
- `sepFilter2D()`                  Separable linear filter
- `boxFilter()`, `GaussianBlur()`, `medianBlur()`, `bilateralFilter()`  
  Smooth the image with one of the linear or non-linear filters
- `Sobel()`, `Schar()`, `Laplacian()`  
  Compute the spatial image derivatives
  
  \[ \Delta I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \]

- `erode()`, `dilate()`               
  Erode or dilate the image

Example. Filter image in-place with a 3x3 high-pass kernel (preserve negative responses by shifting the result by 128):

```c
filter2D(image, image, image.depth()), (Mat_<float>(3,3)<< -1, -1, -1, -1, 9, -1, -1, -1, -1), Point(1,1), 128);
```
Image Conversions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resize()</code></td>
<td>Resize image</td>
</tr>
<tr>
<td><code>getRectSubPix()</code></td>
<td>Extract an image patch</td>
</tr>
<tr>
<td><code>warpAffine()</code></td>
<td>Warp image affinely</td>
</tr>
<tr>
<td><code>warpPerspective()</code></td>
<td>Warp image perspective</td>
</tr>
<tr>
<td><code>remap()</code></td>
<td>Generic image warping</td>
</tr>
<tr>
<td><code>convertMaps()</code></td>
<td>Optimize maps for a faster remap() execution</td>
</tr>
</tbody>
</table>

Example. Decimate image by factor of $\sqrt{2}$:
```cpp
Mat dst; resize(src, dst, Size(), 1./sqrt(2), 1./sqrt(2))
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cvtColor()</code></td>
<td>Convert image from one color space to another</td>
</tr>
<tr>
<td><code>threshold()</code></td>
<td>Convert grayscale image to binary image</td>
</tr>
<tr>
<td><code>adaptiveThreshold()</code></td>
<td>Convert grayscale image to binary image using a fixed or a variable threshold</td>
</tr>
<tr>
<td><code>floodFill()</code></td>
<td>Find a connected component using region growing algorithm</td>
</tr>
<tr>
<td><code>integral()</code></td>
<td>Compute integral image</td>
</tr>
<tr>
<td><code>distanceTransform()</code></td>
<td>build distance map or discrete Voronoi diagram for a binary image.</td>
</tr>
<tr>
<td><code>watershed()</code></td>
<td>marker-based image segmentation algorithms. See the samples <code>watershed.cpp</code> and <code>grabcut.cpp</code>.</td>
</tr>
<tr>
<td><code>grabCut()</code></td>
<td></td>
</tr>
</tbody>
</table>
Histograms

calcHist()     Compute image(s) histogram
calcBackProject() Back-project the histogram
equalizeHist() Normalize image brightness and contrast
compareHist() Compare two histograms

Example. Compute Hue-Saturation histogram of an image:
Mat hsv, H; MatND tempH;
cvtColor(image, hsv, CV_BGR2HSV);
int planes[]={0, 1}, hsize[] = {32, 32};
calcHist(&hsv, 1, planes, Mat(), tempH, 2, hsize, 0);
H = tempH;
I/O

Writing and reading raster images
imwrite("myimage.jpg", image);
Mat image_color_copy = imread("myimage.jpg", 1);
Mat image_grayscale_copy = imread("myimage.jpg", 0);

The functions can read/write images in the following formats:
BMP (.bmp), JPEG (.jpg, .jpeg), TIFF (.tif, .tiff), PNG (.png), PBM/PGM/PPM (.p?m), Sun Raster (.sr),
JPEG 2000 (.jp2). Every format supports 8-bit, 1- or 3-channel images. Some formats (PNG, JPEG 2000) support 16 bits per channel.

Reading video from a file or from a camera
VideoCapture cap;
if(argc > 1) cap.open(string(argv[1])); else cap.open(0);
Mat frame; namedWindow("video", 1);
for(;;) {
    cap >> frame; if(!frame.data) break;
    imshow("video", frame); if(waitKey(30) >= 0) break;
}
Serialization I/O

Data I/O

XML/YAML storages are collections (possibly nested) of scalar values, structures and heterogeneous lists.

Writing data to YAML (or XML)

```cpp
// Type of the file is determined from the extension
FileStorage fs("test.yml", FileStorage::WRITE);
fs << "i" << 5 << "r" << 3.1 << "str" << "ABCD\nFGH";
fs << "mtx" << Mat::eye(3, 3, CV_32F);
fs << "mylist" << "[" << CV_PI << "1+1" << "{":" << "month" << 12 << "day" << 31 << "year"
<< 1969 << "}" << "]";
fs << "mystruct" << "{" << "x" << 1 << "y" << 2 << "width" << 100 << "height" << 200 << "lbp" << ":[";
const uchar arr[] = {0, 1, 1, 0, 1, 1, 0, 1};
fs.writeRaw("u", arr, (int)(sizeof(arr)/sizeof(arr[0])));
fs << "]" << "}";
```

Scalars (integers, floating-point numbers, text strings), matrices, STL vectors of scalars and some other types can be written to the file storages using `<<` operator.
Serialization I/O

Reading the data back

// Type of the file is determined from the content
FileStorage fs("test.yml", FileStorage::READ);
int i1 = (int)fs["i"]; double r1 = (double)fs["r"];
string str1 = (string)fs["str"];
Mat M; fs["mtx"] >> M;
TreeNode tl = fs["mylist"];
CV_Assert(tl.type() == FileNode::SEQ && tl.size() == 3);
double t0 = (double)tl[0]; string t1 = (string)tl[1];
int m = (int)tl[2]["month"], d = (int)tl[2]["day"];
int year = (int)tl[2]["year"];
TreeNode tm = fs["mystruct"];
Rect r; r.x = (int)tm["x"], r.y = (int)tm["y"];
r.width = (int)tm["width"], r.height = (int)tm["height"];
int lbp_val = 0;
TreeNodeIterator it = tm["lbp"].begin();
for(int k = 0; k < 8; k++, ++it)
    lbp_val |= ((int)*it) << k;

Scalars are read using the corresponding FileNode’s cast
operators. Matrices and some other types are read using >>
operator. Lists can be read using FileNodeIterator’s.
namedWindow(winname, flags) Create named highgui window
destroyWindow(winname) Destroy the specified window
imshow(winname, mtx) Show image in the window
waitKey(delay) Wait for a key press during the specified time interval (or forever). Process events while waiting. Do not forget to call this function several times a second in your code.
createTrackbar(...) Add trackbar (slider) to the specified window
setMouseCallback(...) Set the callback on mouse clicks and movements in the specified window

See camshiftdemo.c and other OpenCV samples on how to use the GUI functions.
Camera Calibration, Pose, Stereo

calibrateCamera() Calibrate camera from several views of a calibration pattern.
findChessboardCorners() Find feature points on the checkerboard calibration pattern.
solvePnP() Find the object pose from the known projections of its feature points.
stereoCalibrate() Calibrate stereo camera.
stereoRectify() Compute the rectification transforms for a calibrated stereo camera.
initUndistortRectifyMap() Compute rectification map (for remap()) for each stereo camera head.
StereoBM, StereoSGBM The stereo correspondence engines to be run on rectified stereo pairs.
reprojectImageTo3D() Convert disparity map to 3D point cloud.
findHomography() Find best-fit perspective transformation between two 2D point sets.

To calibrate a camera, you can use calibration.cpp or stereo_calib.cpp samples. To get the disparity maps and the point clouds, use stereo_match.cpp sample.
Problem: planar object detection
Read two input images:

Mat img1 = imread(argv[1], CV_LOAD_IMAGE_GRAYSCALE);

Detect keypoints in both images:

// detecting keypoints
FastFeatureDetector detector(15);
vector<KeyPoint> keypoints1;
detector.detect(img1, keypoints1);

Compute descriptors for each of the keypoints:

// computing descriptors
SurfDescriptorExtractor extractor;
Mat descriptors1;
extractor.compute(img1, keypoints1, descriptors1);

Now, find the closest matches between descriptors from the first image to the second:

// matching descriptors
BruteForceMatcher<L2<float>> matcher;
vector<DMatch> matches;
matcher.match(descriptors1, descriptors2, matches);
Keypoints example
Matching descriptors example
Geometry validation
## Object Recognition

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matchTemplate</td>
<td>Compute proximity map for given template.</td>
</tr>
<tr>
<td>CascadeClassifier</td>
<td>Viola’s Cascade of Boosted classifiers using Haar or LBP features. Suits for detecting faces, facial features and some other objects without diverse textures. See facedetect.cpp</td>
</tr>
<tr>
<td>HOGDescriptor</td>
<td>N. Dalal’s object detector using Histogram-of-Oriented-Gradients (HOG) features. Suits for detecting people, cars and other objects with well-defined silhouettes. See peopledetect.cpp</td>
</tr>
</tbody>
</table>
#!/usr/bin/python

This program is a demonstration python ROS Node for face and object detection using haar-like features. The program finds faces in a camera image or video stream and displays a red box around them. Python implementation by Roman Stanchak, James Bowman

import roslib
roslib.load_manifest('opencv_tests')
import sys
import os
from optparse import OptionParser
import rospy
import sensor_msgs.msg
import cv_bridge
import cv
import cv2

# Parameters for haar detection
# From the API:
# The default parameters (scale_factor=2, min_neighbors=3, flags=0) are tuned
# for accurate yet slow object detection. For a faster operation on real video:
# images the settings are:
# scale_factor=1.2, min_neighbors=2, flags=CV_HAAR_DO_CANNY_PRUNING,
# min_size=<minimum possible face size

min_size = (20, 20)
image_scale = 2
haar_scale = 1.2
min_neighbors = 2
haar_flags = 0
if __name__ == '__main__':

    pkgdir = roslib.packages.get_pkg_dir("opencv")
    haarfile = os.path.join(pkgdir, "opencv/share/opencv/haarcascades/haarcascade_frontalface_alt.xml")

    parser = OptionParser(usage="usage: %prog [options] [filename|camera_index]")
    parser.add_option("-c", "--cascade", action="store", dest="cascade", type="str", help="Haar cascade file, default %default", default = haarfile")
    (options, args) = parser.parse_args()

    cascade = cv.Load(options.cascade)
    br = CvBridge()

    def detect_and_draw(imgmsg):
        img = br.imgmsg_to_cv(imgmsg, "bgr8")

        # allocate temporary images
        gray = cv.CreateImage((img.width, img.height), 8, 1)
        small_img = cv.CreateImage((cv.Round(img.width / image_scale), cv.Round(img.height / image_scale)), 8, 1)

        # convert color input image to grayscale
        cv.CvtColor(img, gray, cv.CV_BGR2GRAY)

        # scale input image for faster processing
        cv.Resize(gray, small_img, cv.CV_INTER_LINEAR)
        cv.EqualizeHist(small_img, small_img)

        if(cascade):
            faces = cv.HaarDetectObjects(small_img, cascade, cv.CreateMemStorage(0), haar_scale, min_neighbors, haar_flags, min_size)

        if faces:
            for ((x, y, w, h), n) in faces:
                # the input to cv.HaarDetectObjects was resized, so scale the
                # bounding box of each face and convert it to two CvPoints
                pt1 = (int(x * image_scale), int(y * image_scale))
                pt2 = (int((x + w) * image_scale), int((y + h) * image_scale))
                cv.Rectangle(img, pt1, pt2, cv.RGB(255, 0, 0), 3, 8, 0)

        cv.ShowImage("result", img)
        cv.WaitKey(6)

    rospy.init_node('rosfacedetect')
    image_topic = rospy.resolve_name("image")
    rospy.Subscriber(image_topic, sensor_msgs.msg.Image, detect_and_draw)
    rospy.spin()
Outline

- OpenCV Overview
- Functionality
- Programming with OpenCV
- OpenCV on CPU & GPU
- Mobile vision
Hardware optimization: Intel architectures

• Technologies
  • SSE
  • TBB
  • IPP

• Highlights of C++ in-house optimization
  • arithmetical operations on large matrices/images: add, sub, absdiff - 5-6x faster
  • image filtering: e.g. median 3x3 filter is 20x faster!
  • geometrical transformations: resize is 2.5 faster
  • template matching: 2-2.5 faster
  • large matrix processing: SVD of 50x50-1000x1000 matrices is 1.4-2.7x faster
OpenCV GPU Module

Goals:

• Provide developers with a convenient computer vision framework on the GPU

• Maintain conceptual consistency with the current CPU functionality

• Achieve the best performance with GPUs
  – Efficient kernels tuned for modern architectures
  – Optimized dataflows (asynchronous execution, copy overlaps, zero-copy)
OpenCV GPU Module Contents

• Image processing building blocks:
  - Color conversions
  - Geometrical transforms
  - Per-element operations
  - Integrals, reductions
  - Template matching
  - Filtering engine
  - Feature detectors

• High-level algorithms:
  - Stereo matching
  - Face detection
  - Feature matching
OpenCV GPU Module Usage

• Prerequisites:
  – Get sources from SourceForge or SVN
  – CMake
  – NVIDIA Display Driver
  – NVIDIA GPU Computing Toolkit (for CUDA)
• Build OpenCV with CUDA support
• `#include <opencv2/gpu/gpu.hpp>`

http://opencv.willowgarage.com/wiki/InstallGuide
OpenCV GPU Data Structures

- **Class GpuMat**
  - For storing 2D image in GPU memory, just like class cv::Mat
  - Reference counting
  - Can point to data allocated by user

- **Class CudaMem**
  - For pinned memory support
  - Can be transformed into cv::Mat or cv::gpu::GpuMat

- **Class Stream**
  - Overloads with extra Stream parameter

```cpp
// class GpuMat
GpuMat(Size size, int type);
GpuMat(const GpuMat& m);
extatic GpuMat (const Mat& m);
GpuMat& operator = (const GpuMat& m);
GpuMat& operator = (const Mat& m);
void upload(const Mat& m);
void upload(const CudaMem& m, Stream& stream);
void download(Mat& m) const;
void download(CudaMem& m, Stream& stream) const;

// class Stream
bool queryIfComplete();
void waitForCompletion();
void enqueueDownload(const GpuMat& src, Mat& dst);
void enqueueUpload(const Mat& src, GpuMat& dst);
void enqueueCopy(const GpuMat& src, GpuMat& dst);
```
OpenCV GPU Module Example

Mat frame;
VideoCapture capture(camera);
cv::HOGDescriptor hog;

hog.setSVMDetector(cv::HOGDescriptor ::getDefaultPeopleDetectorector());
capture >> frame;

vector<Rect> found;
hog.detectMultiScale(frame, found, 1.4, Size(8, 8), Size(0, 0), 1.05, 8);

Mat frame;
VideoCapture capture(camera);
cv::gpu::HOGDescriptor hog;

hog.setSVMDetector(cv::HOGDescriptor ::getDefaultPeopleDetectorector());
capture >> frame;

GpuMat gpu_frame;
gpu_frame.upload(frame);

vector<Rect> found;
hog.detectMultiScale(gpu_frame, found, 1.4, Size(8, 8), Size(0, 0), 1.05, 8);

• Designed very similar!
• NPP is **NVIDIA Performance Primitives** library of signal and image processing functions (similar to **Intel IPP**)
  – NVIDIA will continue adding new primitives and optimizing for future architectures

• GPU module uses NPP whenever possible
  – Highly optimized implementations for all supported NVIDIA architectures and OS
  – Part of CUDA Toolkit – no additional dependencies

• **OpenCV extends NPP and uses it to build higher level CV**
OpenCV GPU Module Performance

Tesla C2050 (Fermi) vs. Core i5-760 2.8GHz (4 cores, TBB, SSE)

– Average speedup for primitives: \(33 \times\)
  
  • For “good” data (large images are better)
  • Without copying to GPU

What can you get from your computer?

– opencv\samples\gpu\performance
OpenCV GPU: Viola-Jones Cascade Classifier

• Used for face detection
• Speed-up ~ 6x
• Based on NCV classes (NVIDIA implementation)
OpenCV GPU: Histogram of Oriented Gradients

- Used for pedestrian detection
- Speed-up ~ 8×
OpenCV GPU: Speeded Up Robust Features

- SURF \((12\times)\)
- Brute-force matcher
  - K-Nearest search \((20-30\times)\)
  - In radius search \((3-5\times)\)
OpenCV GPU: Stereo Vision

- Stereo Block Matching ($7\times$)
  - Can run Full HD real-time on Dual-GPU

- Hierarchical Dense Stereo
  - Belief Propagation ($20\times$)
  - Constant space BP ($50-100\times$)
Epipolar geometry

Fundamental matrix constraint

\[(x_L, y_L, 1) \cdot F \cdot \begin{pmatrix} x_R \\ y_R \\ 1 \end{pmatrix} = 0\]
Block matching

\[ SSD_{x,y} = \sum_{i=x-R_u}^{x+R_u} \sum_{j=y-R_v}^{y+R_v} (Left_{i,j} - Right_{i-d,j})^2 \]
Parallel algorithm
Оптимизация кода на CUDA

Проблема №1: occupancy

• Дефицит регистров
  – Отказ от текстурного механизма
  – Ручная оптимизация кода на CUDA

• Дефицит SMEM
  – Использование unsigned char и short
  – Подбор размера блока потоков
Оптимизация кода на CUDA

Проблема №3: сложные вычисления

• Переход к относительной адресации памяти внутри потоков
• Переход на быстрые операции
• Удаление повторных вычислений
Stereo on HD in realtime
Outline

• OpenCV Overview
• Functionality
• Programming with OpenCV
• OpenCV on GPU

• Mobile vision
Traditional cameras vs. camera phones
ARM is Pervasive and Open

Units in Billions

Source: ARM, Mercury Research, NVIDIA
Using OpenCV for Android

Wiki with the latest information:
http://opencv.willowgarage.com/wiki/Android

Support/discussion group:
https://groups.google.com/group/android-opencv
Practical session

• Look for opencv_practical_session.tar.gz on your USB stick
• Unzip and install OpenCV
• Make sure you have your favorite compiler environment ready
• CMake is optional but highly recommended
QUESTIONS?
OpenCV GPU: Viola-Jones Cascade Classifier

- Used for face detection
- Speed-up ~ 6x
- Based on NCV classes (NVIDIA implementation)
Canny Edge Detector
Distance Transform

• Distance field from edges of objects

Flood Filling
Hough Transform

Gary Bradski, Adrian Kahler 2008
Space Variant vision: Log-Polar Transform

Screen shots by Gary Bradski, 2005
Scale Space

void cvPyrDown(
    IplImage* src,
    IplImage* dst,
    IplFilter  filter = IPL_GAUSSIAN_5x5);

void cvPyrUp(
    IplImage* src,
    IplImage* dst,
    IplFilter  filter = IPL_GAUSSIAN_5x5);
Thresholds

Screen shots by Gary Bradski, 2005
Histogram Equalization

Screen shots by Gary Bradski, 2005
Contours
Morphological Operations Examples

- Morphology - applying Min-Max. Filters and its combinations

Image $I$

Erosion $I \ominus B$

Dilatation $I \oplus B$

Opening $I \Theta B = (I \Theta B) \ominus B$

Closing $I \Theta B = (I \Theta B) \ominus B$

Grad($I$) = ($I \Theta B$) - ($I \ominus B$)

TopHat($I$) = $I$ - ($I \Theta B$)

BlackHat($I$) = ($I \Theta B$) - $I$
Image textures

- Inpainting:
- Removes damage to images, in this case, it removes the text.
Segmentation

- Pyramid, mean-shift, graph-cut
- Here: Watershed

*Screen shots by Gary Bradski, 2005*
Recent Algorithms: GrabCut

- Graph Cut based segmentation

Images by Gary Bradski, © 2010
Motion Templates (work with James Davies)

- Object silhouette
- Motion history images
- Motion history gradients
- Motion segmentation algorithm

Charts by Gary Bradski, 2005
Segmentation, Motion Tracking and Gesture Recognition

Screen shots by Gary Bradski, 2005
New Optical Flow Algorithms

// opencv/samples/c/lkdemo.c
int main(...){
...
CvCapture* capture = <...> ?
    cvCaptureFromCAM(camera_id) :
    cvCaptureFromFile(path);
if( !capture ) return
    -1;
for(;;) {
    IplImage* frame=cvQueryFrame(capture);
    if(!frame) break;
    // ... copy and process image
    cvCalcOpticalFlowPyrLK( ...)
    cvShowImage( "LkDemo", result );
    c=cvWaitKey(30); // run at ~20-30fps speed
    if(c >= 0) {
        // process key
    }
} cvReleaseCapture(&capture

lkdemo.c, 190 lines
(needs camera to run)

\[
\begin{align*}
I(x + dx, y + dy, t + dt) &= I(x, y, t); \\
- \frac{\partial I}{\partial t} &= \frac{\partial I}{\partial x} \cdot (dx / dt) + \frac{\partial I}{\partial y} \cdot (dy / dt); \\
G \cdot \partial X &= b, \\
\partial X &= (\partial x, \partial y), G = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, b = \sum I_t \begin{bmatrix} I_x \\ I_y \end{bmatrix}
\end{align*}
\]
Tracking with CAMSHIFT

• Control game with head

Screen shots by Gary Bradski, 2005
Projections

Affine (2x2)
- Parallelograms

Perspective (3x3) or “Homography”
- Trapazoids (Includes all of Affine)

Screen shots by Gary Bradski, 2005
Stereo ... Depth from Triangulation

• Involved topic, here we will just skim the basic geometry.
• Imagine two perfectly aligned image planes:

Depth “Z” and disparity “d” are inversely related:
Stereo

• In aligned stereo, depth is from similar triangles:

\[
\frac{T - (x^l - x^r)}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{x^l - x^r}
\]

• Problem: Cameras are almost impossible to align
• Solution: Mathematically align them:
Stereo Rectification

• Algorithm steps are shown at right:
• Goal:
  – Each row of the image contains the same world points
  – “Epipolar constraint”
Outline

- OpenCV Overview
- Cheatsheet
- Simple Programs

- Features2D
- Applications
Features 2d contents

Detection

- SIFT
- SURF
- FAST
- STAR
- MSER
- HARRIS
- GFTT (Good Features To Track)

Description

- SIFT
- SURF
- Calonder
- Ferns
- One way

Matching

Matchers available
- BruteForce
- FlannBased
- BOW

Matches filters
(under construction)
- Cross check
- Ratio check
class FeatureDetector
{
public:
    virtual ~FeatureDetector() {}

    // Detect keypoints in an image.
    virtual void detect( const Mat& image, vector<KeyPoint>& keypoints,
                         const Mat& mask=Mat() ) const = 0;

    // Detect keypoints in an image set.
    void detect( const vector<Mat>& imageCollection,
                 vector<vector<KeyPoint> >& pointCollection,
                 const vector<Mat>& masks=vector<Mat>() ) const;

    virtual void read( const FileNode& fn ) {}
    virtual void write( FileStorage& fs ) const {}

protected:
    ...
};
Creating a detector

• Statically
  SurfFeatureDetector detector;

• Using class factory
  cv::Ptr<FeatureDetector> detector =
  createFeatureDetector(“SURF”);
Mat img = imread( "test.png" );
vector<KeyPoint> keypoints;

SurfFeatureDetector detector;
detector.detect(img, keypoints);
Descriptor interfaces

• For descriptors that can be represented as vectors in multidimensional space:
  - DescriptorExtractor and DescriptorMatcher

• More general interface (one way, decision-tree-based descriptors):
  - GenericDescriptorMatcher
class CV_EXPORTS DescriptorExtractor
{
  public:
  virtual ~DescriptorExtractor() {}
  // Compute the descriptors for a set of keypoints in an image.
  virtual void compute( const Mat& image, vector<KeyPoint>& keypoints,
                        Mat& descriptors ) const = 0;
  // Compute the descriptors for a keypoints collection detected in image collection.
  void compute( const vector<Mat>& imageCollection,
                vector<vector<KeyPoint>>& pointCollection,
                vector<Mat>& descCollection ) const;

  virtual void read( const FileNode& ) {}  
  virtual void write( FileStorage& ) const {}
  virtual int descriptorSize() const = 0;
  virtual int descriptorType() const = 0;
  protected:
  ...  
};
DescriptorExtractor creating

- Statically
  SurfDescriptorExtractor descriptorExtractor;
- Using class factory
  cv::Ptr<DescriptorExtractor> descriptorExtractor = createDescriptorExtractor("SURF");
DescriptorExtractor running

Ptr<FeatureDetector> detector =
    createFeatureDetector("FAST");
Ptr<DescriptorExtractor> descriptorExtractor =
    createDescriptorExtractor("SURF");

vector<KeyPoint> keypoints;
detector->detect(img, keypoints);
Mat descriptors;
descriptorExtractor->compute(img, keypoints, descriptors);
DescriptorMatcher interfaces

• Two groups of match methods
  – to match descriptors of image pair
  – to match descriptors of one image to image set

• Each group consists from tree type methods
  – match()
  – knnMatch()
  – radiusMatch()
Matching of image pair

// detecting keypoints
SurfFeatureDetector detector;
vector<KeyPoint> keypoints1, keypoints2;
detector.detect( img1, keypoints1 );
detector.detect( img2, keypoints2 );

// computing descriptors
SurfDescriptorExtractor extractor;
Mat descriptors1, descriptors2;
extractor.compute( img1, keypoints1, descriptors1 );
extractor.compute( img2, keypoints2, descriptors2 );

// matching descriptors
BruteForceMatcher<L2<float> > matcher;
vector<DMatch> matches;
matcher.match( descriptors1, descriptors2, matches );
Visualize keypoints

Mat img_points;
drawKeypoints(img, keypoints, img_points);
namedWindow(“keypoints”, 1);
imshow(“keypoints”, img_points);
waitKey();

Visualize matches

Mat img_matches;
drawMatches(img1, keypoints1,
             img2, keypoints2, img_matches);
namedWindow(“matches”, 1);
imshow(“matches”, img_matches);
waitKey();
Running the sample

- Download OpenCV
- Compile
- Run matcher_simple:
  bin/matcher_simple ..../opencv/samples/c/box.png
  ..../opencv/samples/c/box_in_scene.png

- Select a detector that gives the maximum number of keypoints
- Switch SIFT and SURF descriptors
Cross-check outlier match filtering

BruteForceMatcher<L2<float>> descriptorMatcher;
vector<DMatch> filteredMatches12, matches12, matches21;
descriptorMatcher.match(descriptors1, descriptors2, matches12);
descriptorMatcher.match(descriptors2, descriptors1, matches21);

for( size_t i = 0; i < matches12.size(); i++ )
{
    DMatch forward = matches12[i];
    DMatch backward = matches21[forward.trainIdx];
    if( backward.trainIdx == forward.queryIdx )
        filteredMatches12.push_back( forward );
}
Ratio test to filter matches

\[ \text{Ratio} = \frac{\text{MinDist}_1}{\text{MinDist}_2} \in (0,1] \quad (\text{less is better}) \]

if \( \text{Ratio} < \text{threshold}(0.3) \Rightarrow \text{inlier} \), else outlier
Calculating inliers (planar objects case)

- Detect keypoints
- Find matches using descriptors
- Filter matches using cross-check
- Calculate best homography
- Filter outliers
- Run
  
  bin/descriptor_extractor_matcher SURF SURF
  ../../../opencv/samples/c/box.png
  ../../../opencv/samples/c/box_in_scene.png 3

  The last parameter is the reprojection threshold for RANSAC
OpenCV and ROS

- OpenCV2 package to fetch and compile opencv
- Messages:
  - sensor_msgs::Image
  - sensor_msgs::CameraInfo
- cv_bridge to convert between messages and images
- image_geometry::PinholeCameraModel and image_geometry::StereoCameraModel to manage 2d <-> 3d conversions
Q&A

• Foils will be available at http://itseez.com