OpenCV Basics

Lecture #3

January 22, 2018
... and so it begins

Intel's computer vision library: applications in calibration, stereo, segmentation, tracking, gesture, face and object recognition

Authors  Gary R Bradski, Vadim Pisarevsky
Publication date  2000/6/13
Conference  2013 IEEE Conference on Computer Vision and Pattern Recognition
Volume  2
Pages  2796-2796
Publisher  IEEE Computer Society
Description  Intel's Microcomputer Research Lab has been developing a highly optimized Computer Vision Library (CVLib) that automatically detects processor type and loads the appropriate MMX™ technology assembly tuned module for that processor. MMX optimized functions are from 2 to 8 times faster than optimized C functions. We will be demonstrating various algorithms supported by CVLib and handing out CDs containing the library. 1. Background ... Over the past year and a half, Intel has developed a computer vision library (CVLib) to support real time vision ...

Total citations  Cited by 101
Recall – “Hello World” Equivalent
... and when run locally ...
Next up … Images

Mat - The Basic Image Container

Goal

We have multiple ways to acquire digital images from the real world: digital cameras, scanners, computed tomography, and magnetic resonance imaging to name a few. In every case what we (humans) see are images. However, when transforming this to our digital devices what we record are numerical values for each of the points of the image.

For example in the above image you can see that the mirror of the car is nothing more than a matrix containing all the intensity values of the pixel points. How we get and store the pixels values may vary according to our needs, but in the end all images inside a computer world may be...
MAT as of OpenCV 2.0 (and up)

• Memory management handled for us.
• A tremendous amount of information hiding.
• All and all – a good thing.
• All common types supported
  – Gray scale
  – Color (RGB, HSV, ..)
  – In case you wondered:
    • “There are more than 150 color-space conversion methods available in OpenCV.”
Further practice reading, writing and now modifying images.
Tutorial 3 – Printing (a few) pixels

FYI – Pixels from the center of the right eye.
Tutorial 4 – Changing Pixels

```cpp
int main(int argc, char** argv) {
    Mat image;
    image = imread("IconFaceLv2.png", 1);

    // Make a small window into this larger image
    Mat eye(image, Rect(182, 218, 8, 8));
    cout << "Before color values."
        << endl;
    cout << "eye = " << endl
        << "" << eye << endl
        << endl;

    MatIterator_<Vec3b> it, end;
    for (it = eye.begin<Vec3b>(), end = eye.end<Vec3b>(); it != end; ++it) {
        (*it)[0] = 0;
        (*it)[1] = 0;
        (*it)[2] = 255;
    }

    cout << "After color values."
        << endl;
    cout << "eye = " << endl
        << "" << eye << endl
        << endl;

    namedWindow( "IconFaceLv2.png", CV_WINDOW_AUTOSIZE);
    imshow( "IconFaceLv2.png", image);
    waitKey(0);
    return 0;
}
```
Some Observations

• Pixels are shared between two Mat objects.
• Iterators simplify access.
• A color image consists of 3 channels.
  – Hence, the Vec3b usage
• Why is the 3rd channel set to 255
  – The goal is to give the Red Panda red eye.
Tutorial 5 – Affine Transforms

• We return to image warping, but now making it happen in OpenCV
Before and After Points

```c
int main( int argc, char** argv )
{
    Point2f srcTri[3];
    Point2f dstTri[3];

    Mat rot_mat( 2, 3, CV_32FC1 );
    Mat warp_mat( 2, 3, CV_32FC1 );
    Mat src, warp_dst, warp_rotate_dst;

    // Load the image
    /// Set your 3 points to calculate the Affine Transform
    srcTri[0] = Point2f( 0,0 );
    srcTri[1] = Point2f( src.cols - 1, 0 );
    srcTri[2] = Point2f( 0, src.rows - 1 );

    dstTri[0] = Point2f( src.cols*0.0, src.rows*0.33 );
    dstTri[1] = Point2f( src.cols*0.85, src.rows*0.25 );
    dstTri[2] = Point2f( src.cols*0.15, src.rows*0.7 );
```
Make it so …

Recall, this is what we discussed about 2 lectures ago when we used before/after representations of 3 2D points to define a 6DOF transform.

```cpp
    // Get the Affine Transform
    warp_mat = getAffineTransform( srcTri, dstTri );

    // Apply the Affine Transform just found to the src image
    warpAffine( src, warp_dst, warp_mat, warp_dst.size() );
```
Warp then Rotate

Pay attention to the center of rotation.

```cpp
/** Rotating the image after Warp */

/// Compute a rotation matrix with respect to the center of the image
Point center = Point( warp_dst.cols/2, warp_dst.rows/2 );
double angle = -50.0;
double scale = 0.6;

/// Get the rotation matrix with the specifications above
rot_mat = getRotationMatrix2D( center, angle, scale );

/// Rotate the warped image
warpAffine( warp_dst, warp_rotate_dst, rot_mat, warp_dst.size() );
```
Project 1 – Track the Frigatebird