OpenCV Basics

Lecture #3

January 22, 2018

Colorado State University

… 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 then 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, Intel1 has developed a computer vision library (CVLib) to support real time vision ...
Total citations	Cited by 101

Recall – "Hello World" Equivalent

docs.opencv.org

OpenCV 2.4.9.0 documentation » OpenCV Tutorials » Introduction to OpenCV » previous | next | index

OpenCV

Search

Table Of Contents

Load and Display an Image
- Goal
- Source Code
- Explanation
- Result

Previous topic

Cross compilation for ARM based Linux systems

Next topic

Load, Modify, and Save an Image

This Page

Show Source

Load and Display an Image

Goal

In this tutorial you will learn how to:

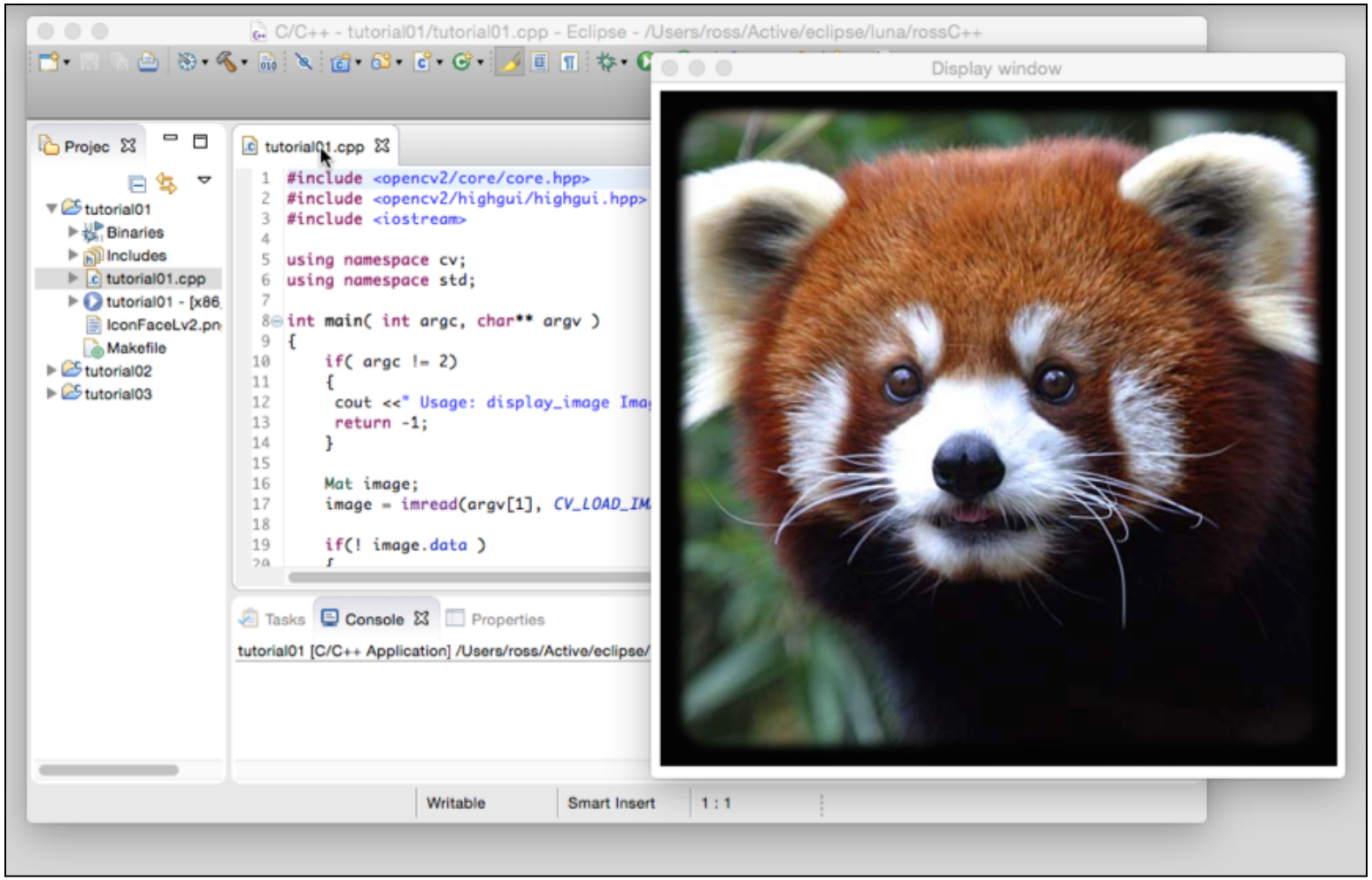
- Load an image (using imread)
- Create a named OpenCV window (using namedWindow)
- Display an image in an OpenCV window (using imshow)

Source Code

Download the source code from here.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    if( argc != 2)
    {
     cout <<" Usage: display_image ImageToLoadAndDisplay" << endl;
     return -1;
    }

    Mat image;
    image = imread(argv[1], CV_LOAD_IMAGE_COLOR);   // Read the file
```

… and when run locally …



Colorado State University

Next up … Images

OpenCV

Search

Table Of Contents

Previous topic

core module. The Core Functionality

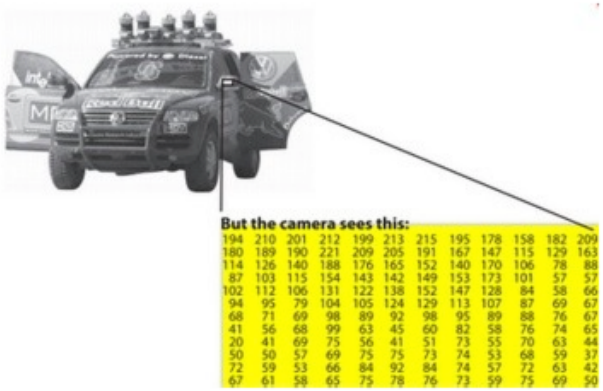
Next topic

How to scan images, lookup tables

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."

Tutorial 2

tutorial02.cpp

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

OpenCV 2.4.9.0 documentation » OpenCV Tutorials » Introduction to OpenCV » previous | next | index

OpenCV

Search

Table Of Contents

Load, Modify, and Save an Image

Note: We assume that by now you know how to load an image using imread and to display it in a window (using imshow). Read the *Load and Display an Image* tutorial otherwise.

Goals

In this tutorial you will learn how to:

- Load an image using imread
- Transform an image from BGR to Grayscale format by using cvtColor
- Save your transformed image in a file on disk (using imwrite)

```
Mat gray_image;
cvtColor( image, gray_image, CV_BGR2GRAY );

imwrite( "Gray_Image.jpg", gray_image );

namedWindow( imageName, CV_WINDOW_AUTOSIZE );
namedWindow( "Gray image", CV_WINDOW_AUTOSIZE );

imshow( imageName, image );
imshow( "Gray image", gray_image );

waitKey(0);

return 0;
}
```

Further practice reading, writing and now modifying images.

Colorado State University

Tutorial 3 – Printing (a few) pixels

Terminal — tutorial03 — 164×49

```
[Adams:luna/rossC++/tutorial03] ross% ls
total 896
drwxr-xr-x  9 ross  staff     306 Jan 29 19:22 .
drwxr-xr-x  6 ross  staff     204 Jan 28 10:42 ..
-rw-r--r--  1 ross  staff    9297 Jan 29 19:20 .cproject
-rw-r--r--  1 ross  staff     811 Jan 28 10:42 .project
-rw-r--r--  1 ross  staff   79486 Jan 29 19:22 Gray_Image.jpg
-rw-r--r--  1 ross  staff  326508 May 18  2009 IconFaceLv2.png
-rw-r--r--  1 ross  staff     149 Jan 29 13:30 Makefile
-rwxr-xr-x  1 ross  staff   22340 Jan 29 19:22 tutorial03
-rw-r--r--  1 ross  staff    1017 Jan 29 19:19 tutorial03.cpp
[Adams:luna/rossC++/tutorial03] ross% tutorial03 IconFaceLv2.png
tiny =
 [8, 12, 23, 32, 45, 52, 43, 34, 20, 9;
  20, 49, 87, 116, 142, 152, 141, 131, 83, 32;
  104, 156, 166, 176, 176, 164, 178, 211, 139, 93;
  158, 202, 189, 167, 158, 164, 194, 204, 143, 101;
  149, 165, 151, 138, 140, 146, 159, 170, 141, 118;
  114, 115, 123, 142, 160, 155, 137, 155, 165, 161;
  95, 111, 132, 147, 158, 153, 130, 136, 162, 162;
  78, 100, 120, 118, 112, 104, 82, 90, 107, 104;
  71, 78, 93, 92, 80, 67, 43, 60, 67, 63;
  65, 61, 73, 73, 64, 56, 36, 48, 51, 49]
```

Gray image

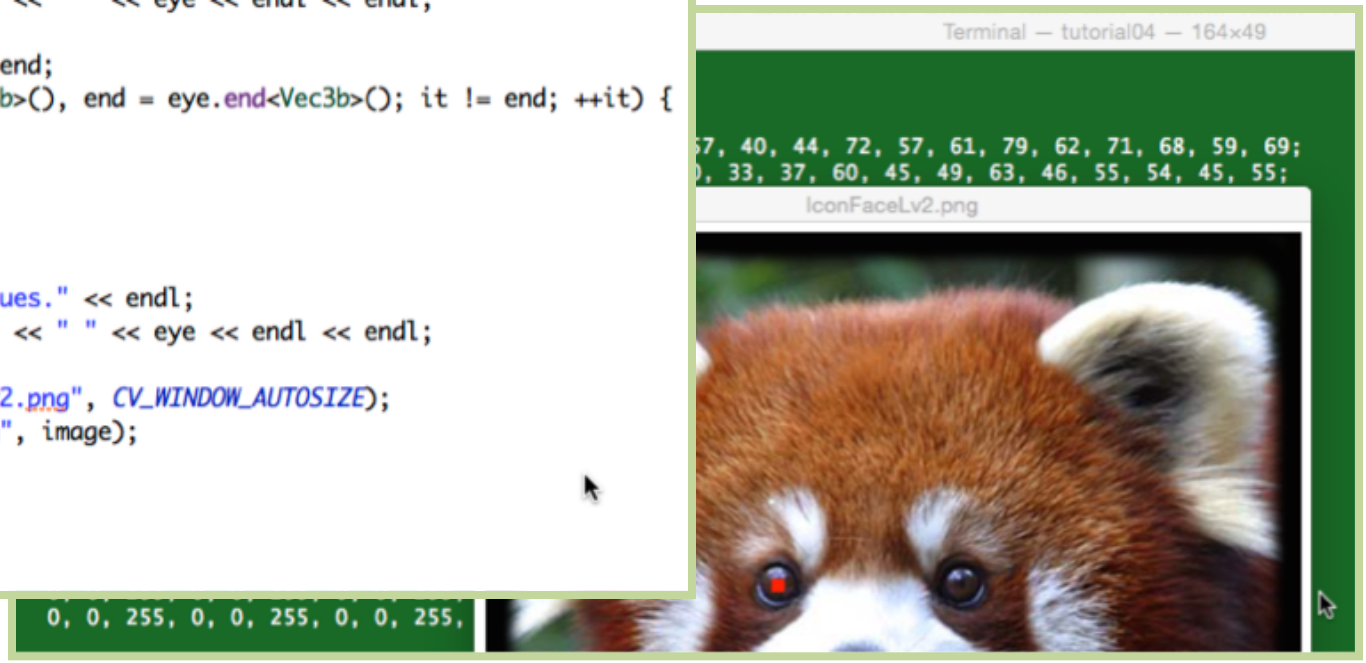
```
Mat gray_image;
cvtColor( image, gray_image, CV_BGR2GRAY );
/* Make a small window into this larger image */
Mat tiny (gray_image, Rect(180, 210, 10, 10));
cout << "tiny = " << endl << " " << tiny << endl << endl;
```

FYI – Pixels from the center of the right eye.

Colorado State University

Tutorial 4 – Changing Pixels

```
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;
}
```

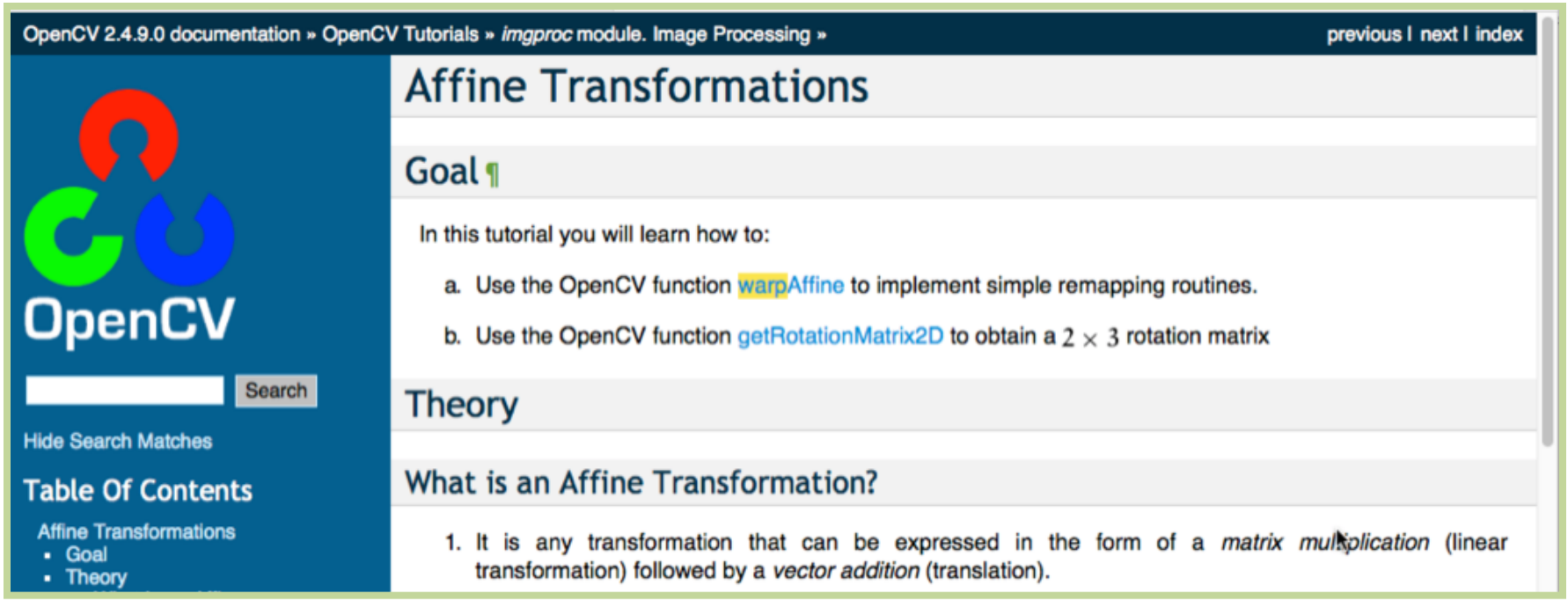


Some Observations

- 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



OpenCV 2.4.9.0 documentation » OpenCV Tutorials » *imgproc* module. Image Processing » previous | next | index

OpenCV

Search

Hide Search Matches

Table Of Contents

Affine Transformations
- Goal
- Theory

Affine Transformations

Goal ¶

In this tutorial you will learn how to:

a. Use the OpenCV function warpAffine to implement simple remapping routines.

b. Use the OpenCV function getRotationMatrix2D to obtain a 2×3 rotation matrix

Theory

What is an Affine Transformation?

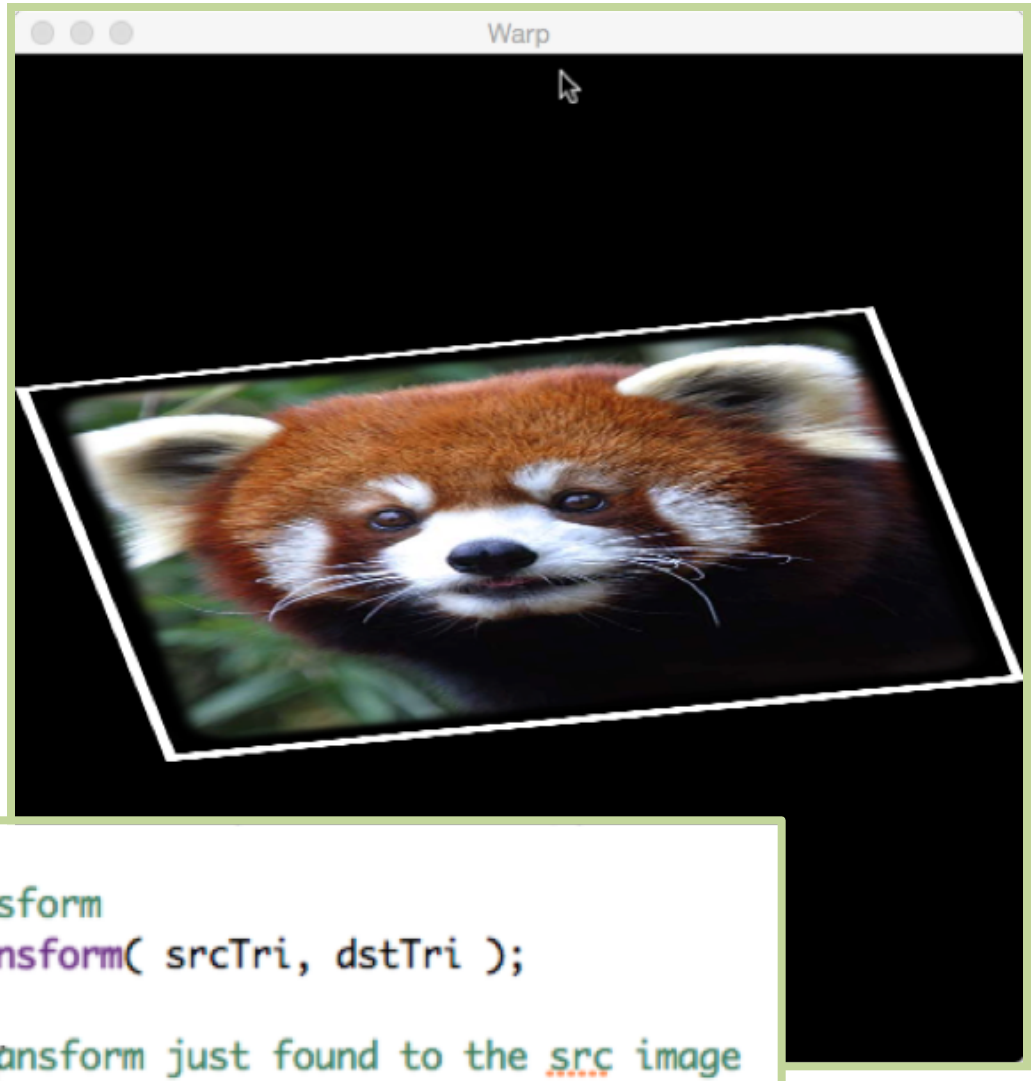
1. It is any transformation that can be expressed in the form of a *matrix multiplication* (linear transformation) followed by a *vector addition* (translation).

Before and After Points

```
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;

  /// 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.




```
43
44    /// Get the Affine Transform
45    warp_mat = getAffineTransform( srcTri, dstTri );
46
47    /// Apply the Affine Transform just found to the src image
48    warpAffine( src, warp_dst, warp_mat, warp_dst.size() );
49
```

Warp then Rotate

Pay attention to the center of rotation.


```
/** 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