OpenCV Tutorials

v2.2

February, 2011

Contents

I C++ API tutorials **5**

1 Prerequisites **7**
1.1 Prerequisites 7

2 Features2d **9**
2.1 Detection of planar objects 9

3 Calib3d **11**
3.1 Camera calibration 11
3.2 Pose estimation 11

Index **12**

Part I

C++ API tutorials

Chapter 1

Prerequisites

1.1 Prerequisites

Download the latest release of opencv from http://sourceforge.net/projects/opencvlibrary/.
You will need cmake and your favorite compiler environment in order to build opencv from sources.
Please refer to the installation guide http://opencv.willowgarage.com/wiki/InstallGuide
for detailed instructions.

Chapter 2

Features2d

2.1 Detection of planar objects

The goal of this tutorial is to learn how to use features2d and calib3d modules for detecting known planar objects in scenes.

`Test data`: use images in your data folder, for instance, box.png and box_in_scene.png.

Create a new console project. Read two input images. Example:

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

Detect keypoints in both images. Example:

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

Compute descriptors for each of the keypoints. Example:

```
// 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);
```

Visualize the results:

```
// drawing the results
```

```
namedWindow("matches", 1);
Mat img_matches;
drawMatches(img1, keypoints1, img2, keypoints2, matches, img_matches);
imshow("matches", img_matches);
waitKey(0);
```

Find the homography transformation between two sets of points:

```
vector<Point2f> points1, points2;
// fill the arrays with the points
....
Mat H = findHomography(Mat(points1), Mat(points2), CV_RANSAC, ransacReprojThreshold);
```

Create a set of inlier matches and draw them. Use perspectiveTransform function to map points with homography:

```
Mat points1Projected;
perspectiveTransform(Mat(points1), points1Projected, H);
```

Use drawMatches for drawing inliers.

Chapter 3

Calib3d

3.1 Camera calibration

The goal of this tutorial is to learn how to calibrate a camera given a set of chessboard images.

`Test data`: use images in your data/chess folder.

Compile opencv with samples by setting BUILD_EXAMPLES to ON in cmake configuration.

Go to bin folder and use `imagelist_creator` to create an xml/yaml list of your images. Then, run `calibration` sample to get camera parameters. Use square size equal to 3cm.

3.2 Pose estimation

Now, let us write a code that detects a chessboard in a new image and finds its distance from the camera. You can apply the same method to any object with knwon 3d geometry that you can detect in an image.

`Test data`: use chess_test*.jpg images from your data folder.

Create an empty console project. Load a test image:

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

Detect a chessboard in this image using findChessboard function.

```
bool found = findChessboardCorners( img, boardSize, ptvec, CV_CALIB_CB_ADAPTIVE_THRESH );
```

Now, write a function that generates a `vector<Point3f>` array of 3d coordinates of a chessboard in any coordinate system. For simplicity, let us choose a system such that one of the chessboard corners is in the origin and the board is in the plane $z = 0$.

Read camera parameters from xml/yaml file:

```
FileStorage fs(filename, FileStorage::READ);
Mat intrinsics, distortion;
fs["camera_matrix"] >> intrinsics;
```

```
fs["distortion_coefficients"] >> distortion;
```

Now we are ready to find chessboard pose by running solvePnP:

```
vector<Point3f> boardPoints;
// fill the array
...

solvePnP(Mat(boardPoints), Mat(foundBoardCorners), cameraMatrix,
                     distCoeffs, rvec, tvec, false);
```

Calculate reprojection error like it is done in `calibration` sample (see textttopencv/samples/cpp/calibration.cpp, function `computeReprojectionErrors`).

How to calculate the distance from the camera origin to any of the corners?