OpenCV Basics and Camera Calibration

Computer Vision Lab Tutorial
5 October 2012
Lorenz Meier, Kevin Koeser, Kalin Kolev
What is OpenCV?

- The most popular library in Computer Vision
- Released under the liberal BSD license
- It has C++, C, Python and Java interfaces and supports Windows, Linux, Android and Mac OS
- Offers optimized implementation of more than 2500 algorithms
- New official website: http://code.opencv.org
- OpenCV4Android: http://opencv.org/android.html
OpenCV Overview

OpenCV Overview: > 500 functions

- General Image Processing Functions
- Geometric descriptors
- Image Pyramids
- Camera calibration, Stereo, 3D
- Utilities and Data Structures
- Segmentation
- Features
- Transforms
- Machine Learning:
  - Detection
  - Recognition
- Tracking
- Fitting
- Matrix Math
- Robot support
Core Functionality

- Each image is represented as a matrix
- Basic class **Mat** for storing general n-dimensional arrays (e.g. grayscale images, color images, voxel volumes etc.)
- Fast access to its elements via pointers, e.g.

  ```
  Mat mat;
  mat.ptr<float>(i)[j] = ...
  ```

- Direct access to user-allocated data (e.g. video frames)

  ```
  Mat image(height, width, CV_8UC3, pixels, step);
  ```
Hints for Android Developers

- If you use the camera device, make sure that you have required permission in your AndroidManifest.xml

```xml
<uses-feature android:name="android.hardware.camera" />
<uses-permission android:name="android.permission.CAMERA" />
```

- If you want to write images to the storage of the mobile device via `imwrite()`, make sure that you have required permission in your AndroidManifest.xml

```xml
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```
Image Segmentation

- Interactive image segmentation

C. Rother, V. Kolmogorov, A. Blake, "GrabCut — Interactive Foreground Extraction using Iterated Graph Cuts"

- OpenCV implementation

```c
void grabCut(img, mask, rect, bgdModel, fgdModel, iterCount, mode);
```
Pinhole Camera

- Pinhole camera model

\[ P = K[R|t] \]

projection matrix

- calibration matrix

\[
K = \begin{pmatrix}
 f_x & s & p_x \\
 0 & f_y & p_y \\
 0 & 0 & 1
\end{pmatrix}
\]

extrinsic parameters

\[ R \in \mathbb{R}^{3 \times 3} \]

\[ t \in \mathbb{R}^{3} \]
Camera Calibration

- OpenCV implementation

```cpp
bool findChessboardCorners(image, patternSize, corners, flags);

double calibrateCamera(objectPoints, imgPoints, imgSize, camMatrix,
                        distCoeffs, rvecs, tvecs, flags, criteria);

Mat initCameraMatrix2D(objPoints, imgPoints, imgSize, aspectRatio);

void getOptimalNewCameraMatrix(camMatrix, distCoeffs, imgSize,
                                alpha, newImgSize, PixROI, cPP);

void undistort(src, dst, cameraMatrix, distCoeffs, newCameraMatrix);
```
Structure from Motion

- Structure-from-motion pipeline

1. Feature point extraction
2. Feature point matching/tracking
3. Camera pose estimation
Feature Point Extraction

- Abstract base class for 2D image feature detectors
  FeatureDetector

  ```cpp
  void FeatureDetector::detect(image, keypoints, mask);
  Ptr<FeatureDetector> FeatureDetector::create(detectorType);
  ```

- The following detector types are supported: FAST, STAR, SIFT (nonfree module), SURF (nonfree module), ORB, MSER, GFTT, HARRIS, Dense, SimpleBlob
Feature Point Matching

- Abstract base class for computing descriptors for image keypoints: DescriptorExtractor

  ```
  void DescriptorExtractor::compute(image, keypoints, descriptors);
  Ptr<DescriptorExtractor> DescriptorExtractor::create(descriptorType);
  ```

- The following types of descriptor extractors are supported: SIFT, SURF, ORB, BRIEF

- Alternative in case of video input: KLT tracker

  ```
  void calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts, ...);
  int buildOpticalFlowPyramid(img, pyramid, ...);
  ```
Camera Pose Estimation

- General approach: triangulation + bundle adjustment

- Special case: ground plane estimation

\[ H \in \mathbb{R}^{3 \times 3} \]

\[ x' = Hx \]
Hints to Planar Homography Estimation

- Don’t use OpenCV’s `findHomography` as it estimates a general homography.

- Note that a general homography has 8 degrees of freedom while a plane is determined by only 3 degrees of freedom (=> use additional constraints).

- Reference: R. Hartley, A. Zisserman, "Multiple View Geometry in Computer Vision"
Qualcomm`s Vuforia

- The Vuforia toolkit estimates camera intrinsics and poses by means of markers
- Camera intrinsics: QCAR::CameraCalibration
- Camera poses: QCAR::Trackable -> getPose()
- Note that in the current implementation no distortion parameters are being estimated