OpenCV 2.4 Cheat Sheet (C++)

The OpenCV C++ reference manual is here: http://docs.opencv.org. Use Quick Search to find descriptions of the particular functions and classes.

Key OpenCV Classes

- **Point**
  - Template 2D point class
- **Point3**
  - Template 3D point class
- **Size**
  - Template size (width, height) class
- **Vec**
  - Template short vector class
- **Mat**
  - 2D or multi-dimensional dense array
- **Matx**
  - Template small matrix class
- **SparseMat**
  - Multi-dimensional sparse array
- **Ptr**
  - Template smart pointer class
- **Rect**
  - 4-element vector
- **Range**
  - Integer value range
- **Rectangle**
  - 4-element vector
- **Mat_<float>**
  - 2D or multi-dimensional dense array
- **Vec3b**
  - 3-element vector
- **Size_<int>**
  - Template size (width, height) class
- **Point_<float>**
  - 2D point class

Matrix Basics

- **Create a matrix**
  - `image = Mat(240, 320, CV_8UC3);`
- **Create a matrix initialized with a constant**
  - `Mat A33(3, 3, CV_32F, Scalar(5));`
- **Create a matrix initialized with specified values**
  - `double a = CV_PI/3; Mat A22 = (Mat_<float>(2, 2) << cos(a), -sin(a), sin(a), cos(a));`
- **Create a random matrix**
  - `float* Idata=new float[480*640*3]; Mat image_alias = image;`

Matrix Manipulations: Copying, Shuffling, Part Access

- **Copy matrix to another one**
  - `src.copyTo(dat);`
- **Copy matrix from/to other structures**
  - `(without copying the data)`
  - `(copying the data)`
- **Convert matrix to/from other structures**
  - `(without copying the data)`
  - `(copying the data)`

Simple Matrix Operations

- **OpenCV implements most common arithmetical, logical and other matrix operations**, such as

  - `A33.at<float>(i,j) = A33.at<float>(j,i)+1;`
  - `Mat dyImage(image.size(), image.type());`
  - `for(int i = 0; i < image.rows; i++) { Vec3b* prevRow = image.ptr<Vec3b>(i); Vec3b* nextRow = image.ptr<Vec3b>(i+1); for(int j = 0; j < image.cols; j++) { for(int c = 0; c < 3; c++) { dyImage.at<Vec3b>(i,j)[c] = saturate_cast<uchar>(nextRow[c] - prevRow[c]); } Mat_<Vec3b>::iterator it = image.begin<Vec3b>(), itEnd = image.end<Vec3b>(); for(; it != itEnd; ++it) (*it)[1] += 255; } dyImage.copyTo(image);}`

Image Processing

- **Filtering**
  - `filter2D()` Non-separable linear filter
  - `sepFilter2D()` Separable linear filter
  - `smoothImage` Smooth the image with one of the linear or non-linear filters

- **Simple Matrix Operations**
  - `OpenCV implements most common arithmetical, logical and other matrix operations, such as`
  - `Mat dyImage(image.size(), image.type()); for(int y = 1; y < image.rows-1; y++) { Vec3b* prevRow = image.ptr<Vec3b>(y-1); Vec3b* nextRow = image.ptr<Vec3b>(y+1); for(int x = 0; x < image.cols; x++) { for(int c = 0; c < 3; c++) { dyImage.at<Vec3b>(y,x)[c] = saturate_cast<uchar>(nextRow[c] - prevRow[c][c]); } Mat_<<Vec3b>::iterator it = image.begin<Vec3b>(), itEnd = image.end<Vec3b>(); for(; it != itEnd; ++it) (*it)[1] += 255; } dyImage.copyTo(image);}`

- **Image Processing**
  - `filter2D()` Non-separable linear filter
  - `sepFilter2D()` Separable linear filter
  - `smoothImage` Smooth the image with one of the linear or non-linear filters

- **Example 1. Smooth image ROI in-place**
  - `Mat imroi = image(Rect(10, 20, 100, 100)); GaussianBlur(imroi, imroi, Size(5, 5), 1, 1);`

- **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 srcROI = dst Rect(0,10,100,100); Mat dstroi; dstroi.convertTo(dstroi, dstroi.type(), 1, 1);`

- **Example 4. Compute Laplacian of an image**
  - `Laplacian();` Morphological operations

- **Example 5. Compute the spatial image derivatives**
  - `Sobel();` Compute the spatial image derivatives
  - `Schar();` compute Schar derivative
  - `Laplacian();` compute Laplacian derivative

- **Example 6. Compute the spatial image derivatives**
  - `Laplacian();` compute Laplacian derivative
  - `Schar();` compute Schar derivative
  - `Laplacian();` compute Laplacian derivative
Example. Filter image in-place with a 3x3 high-pass kernel (preserve negative responses by shifting the result by 128):
```
filter2D(image, image, image.depth(), (Mat_<float>(3,3)<
-1, -1, -1, 9, -1, -1, -1, -1, -1>, Point(1,1), 128);
```

**Geometrical Transformations**
- `resize()` - Resize image
- `getRectSubPix()` - Extract an image patch
- `warpAffine()` - Warp image affinely
- `warpPerspective()` - Warp image perspective
- `remap()` - Generic image warping
- `convertMaps()` - Optimize maps for a faster remap() execution

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

**Various Image Transformations**
- `cvtColor()` - Convert image from one color space to another
- `threshold()` - Convert grayscale image to binary image using a fixed or a variable threshold
- `floodFill()` - Find a connected component using region growing algorithm
- `integral()` - Compute integral image
- `distanceTransform()` - Build distance map or discrete Voronoi diagram for a binary image
- `watershed()` - Marker-based image segmentation algorithms. See the samples `watershed.cpp` and `grabcut.cpp`

**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;
cvtColor(image, hsv, CV_BGR2HSV);
calcHist(hsv, 1, planes, Mat(), H, 2, hsize, 0);
```

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

Writing data to YAML (or XML)
```
// Type of the file is determined from the extension
FileStorage fs("test.yml", FileStorage::WRITE);
fs << "i" << 5 << "r" << 3.1 << "str" << "ABCDEFGH";
fs << "mx" << Mat::eye(3,3,CV_32F);
fs << "mylist" << "[i < CV_PI = 1e-1" << "":=" << "month" << 12 << "day" << 31 << "year" << 1969 << "]" << ";fs << "mystruct" << "x" << 1 << "y" << 2 << "width" << 100 << "height" << 200 << "lbp" << "[i;const uchar arr[] = {0, 1, 0, 1, 1, 0, 1};fs.writeRaw("u", arr, (int)(sizeof(arr)/sizeof(arr[0])));
fs << "]" << ");
```

**Simple GUI (highgui module)**
```
namedWindow("video", flags) - Create named highgui window
destroyWindow(window) - Destroy the specified window
imshow("video", frame) - 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.
```

**Camera Calibration, Pose Estimation and Depth Estimation**
- `calibrateCamera()` - Calibrate camera from several views of a calibration pattern.
- `solvePnP()` - Find the object pose from the known projections of its feature points.
- `calibrateStereoCamera()` - Compute the rectification transforms for a calibrated stereo camera.
- `initUndistortRectifyMap()` - Compute rectification map for each stereo camera head.
- `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.

**Object Detection**
- `matchTemplate()` - Compute proximity map for given templates.
- `CascadeClassifier` - 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`
- `HOGDescriptor` - 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`