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

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point_</td>
<td>2D point class</td>
</tr>
<tr>
<td>Point3_</td>
<td>3D point class</td>
</tr>
<tr>
<td>Size_</td>
<td>2D or multi-dimensional size class</td>
</tr>
<tr>
<td>Vec</td>
<td>Template small vector class</td>
</tr>
<tr>
<td>Mat</td>
<td>Multi-dimensional dense array class</td>
</tr>
<tr>
<td>Scalar</td>
<td>Integer value range class</td>
</tr>
<tr>
<td>Rect</td>
<td>Rectangle class</td>
</tr>
<tr>
<td>Range</td>
<td>Integer value range class</td>
</tr>
<tr>
<td>SparseMat</td>
<td>Multi-dimensional sparse array class</td>
</tr>
<tr>
<td>Ptr</td>
<td>Template smart pointer class</td>
</tr>
</tbody>
</table>

Matrix Basics

Create a matrix

Mat image(240, 320, CV_8UC3);
src.copyTo(dst);
src.convertTo(dst, type, scale, shift);
src(r).convertTo(dstroi, dstroi.type(), 1, 0);

Matrix Manipulations: Copying, Shuffling, Part Access

src.copyTo(src);
src.copyTo(dst);
src.copyTo(dst);

Matrix Operations

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

\[ \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix} \]

Example 1. Smooth image ROI in-place

Mat imgray = image(Rect(10, 20, 100, 100));
GaussianBlur(imgray, imgray, Size(5, 5), 1.2, 1.2);
Example 2. Somewhere in a linear algebra algorithm

m.row(i) += m.row(j)*alpha;

Example 3. Copy image ROI into another image with conversion

Mat src = image(Rect(10, 20, 100, 100));
Mat des = image(Rect(0, 0, 100, 100));
src.copyTo(des);

Simple Matrix Operations

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

\[ \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix} \]

Example 1. Smooth image ROI in-place

Mat imgray = image(Rect(10, 20, 100, 100));
GaussianBlur(imgray, imgray, Size(5, 5), 1.2, 1.2);
Example 2. Somewhere in a linear algebra algorithm

m.row(i) += m.row(j)*alpha;

Example 3. Copy image ROI into another image with conversion

Mat src = image(Rect(10, 20, 100, 100));
Mat des = image(Rect(0, 0, 100, 100));
src.copyTo(des);

Simple Matrix Operations

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

\[ \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix} \]

Example 1. Smooth image ROI in-place

Mat imgray = image(Rect(10, 20, 100, 100));
GaussianBlur(imgray, imgray, Size(5, 5), 1.2, 1.2);
Example 2. Somewhere in a linear algebra algorithm

m.row(i) += m.row(j)*alpha;

Example 3. Copy image ROI into another image with conversion

Mat src = image(Rect(10, 20, 100, 100));
Mat des = image(Rect(0, 0, 100, 100));
src.copyTo(des);

Simple Matrix Operations

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

\[ \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix} \]

Example 1. Smooth image ROI in-place

Mat imgray = image(Rect(10, 20, 100, 100));
GaussianBlur(imgray, imgray, Size(5, 5), 1.2, 1.2);
Example 2. Somewhere in a linear algebra algorithm

m.row(i) += m.row(j)*alpha;
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, -1, 9, -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 color image to another
- `threshold()`: Convert grayscale image to binary image
- `adaptiveThreshold()`: Convert a grayscale image to binary 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:
```
Contours
```
```
contours2.cpp
```

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" << 4.0 << "str" << "ABCDEFGH";
fs << "mxt" << Mat::eye(3,3,CV_32F);
fs << "mylist" << "[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]";
fs << "mystruct" << "{" << CV_PI << "1+1" <<
    "x": "month" << 12 << "day" << 31 << "year" << 1969 << "}" << "]";
fs << "mystruct2" << "{" << CV_PI << "1+1" <<
    "x": "month" << 2 << "y": 2 << "width": 100 << "height": 200 << "lbp": "[:
    const uchar arr[] = {0, 1, 0, 1, 1, 1, 0, 1};
fs.writeRaw("u", arr, (int)(sizeof(arr)/sizeof(arr[0])));
fs << "]" << "]";
```

Reading the data back
```
Reading and writing raster images
```
```
VideoCapture cap;
if(argc > 1) cap.open(string(argv[1])); else cap.open(0);
Mat frame; namedWindow("video", 1);
```