# Introduction to Mobile Robotics

Jeff McGough

Department of Computer Science
South Dakota School of Mines and Technology
Rapid City, SD 57701, USA

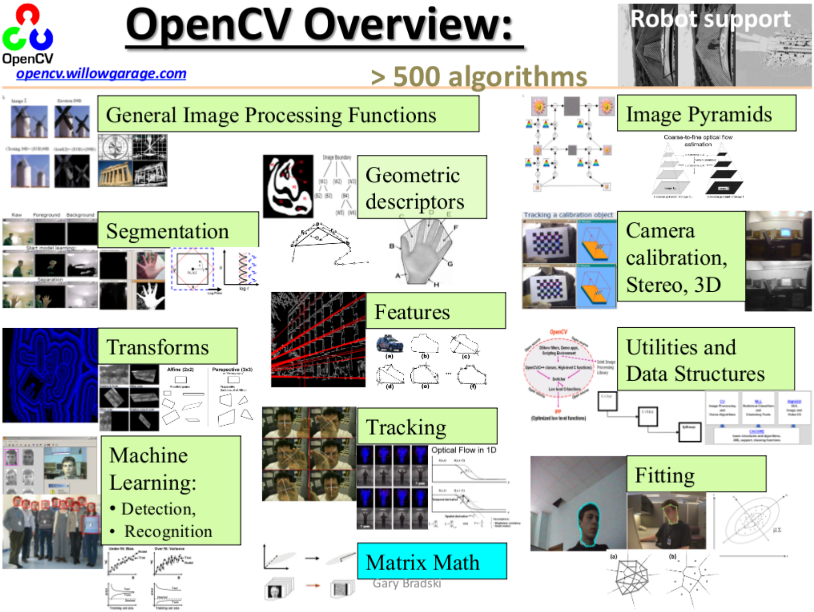
October 1, 2012

# OpenCV

Intel's[1] Open Source Computer Vision Library

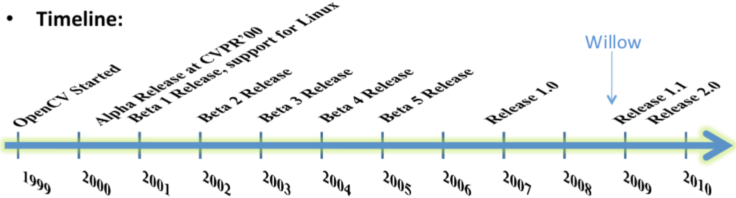
[1] now maintained by Willow Garage

# OpenCV Overview



# OpenCV History

- **Original goal:**
  - Accelerate the field by lowering the bar to computer vision
  - Find compelling uses for the increasing MIPS out in the market
- **Timeline:**




- **Staffing:**
  - Climbed in 1999 to average 7 first couple of years
  - Starting 2003 support declined between zero and one with exception of transferring the machine learning from manufacturing work (equivalent of 3 people).
  - Support to zero the couple of years before Willow.
  - 5 people over the last year

# OpenCV Requirements

Required packages

- GCC 4.x or later. This can be installed with `sudo apt-get install build-essential`
- CMake 2.6 or higher
- Subversion (SVN) client
- GTK+2.x or higher, including headers
- pkgconfig
- libpng, zlib, libjpeg, libtiff, libjasper with development files (e.g. libpjeg-dev)
- Python 2.3 or later with developer packages (e.g. python-dev)
- SWIG 1.3.30 or later (only for versions prior to OpenCV 2.3)
- libavcodec
- libdc1394 2.x

# OpenCV Source

Source: in Ubuntu it can be done using the following command, e.g.:

```
cd ~/<my_working _directory>
svn co https://code.ros.org/svn/opencv/trunk
```

# OpenCV Build

1. Create a temporary directory, which we denote as

   ```
   <cmake_binary_dir>
   ```

   where you want to put the generated Makefiles, project files as well the object files and output binaries

2. Enter the <cmake_binary_dir> directory and type

   ```
   cmake [<some optional parameters>] <path to OpenCV >
   ```

   For example

   ```
   cd ~/opencv
   mkdir release
   cd release
   cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX= /usr/local
   ```

3. Enter the created temporary directory (<cmake_binary_dir>) and proceed with:

   ```
   make
   sudo make install
   ```

# OpenCV Structure

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- **core** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **imgproc** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.

# OpenCV Structure

- **features2d** - salient feature detectors, descriptors, and descriptor matchers.
- **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- **highgui** - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.
- **gpu** - GPU-accelerated algorithms from different OpenCV modules.

  ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

# QtCreator

```
#-------------------------------------------------
#
# Project created by QtCreator 2011-11-19T14:59:28
#
#-------------------------------------------------

QT       -= gui

TARGET = QtOpenCVHello
CONFIG   += console
CONFIG   -= app_bundle

TEMPLATE = app

SOURCES += main.cpp

INCLUDEPATH += /usr/local/OpenCV-2.3.1/bin/include \

QMAKE_LIBDIR_QT -= /usr/lib

LIBS = -L/usr/local/lib -L/usr/local/OpenCV-2.3.1/bin/ \
-lopencv_core \
-lopencv_highgui \
-lopencv_imgproc \
-lopencv_features2d \
-lopencv_calib3d  \
-lopencv_objdetect \
-lopencv_contrib \
-lopencv_legacy \
-lopencv_video
```

# API

Namespace: cv

```
#include "opencv2/core/core.hpp"
...
cv::Mat H = cv::findHomography(points1, points2, CV_RANSAC, 5);
...
```

or

```
#include "opencv2/core/core.hpp"
using namespace cv;
...
Mat H = findHomography(points1, points2, CV_RANSAC, 5 );
...
```

# OpenCV Basics[2]

Images

Load an image from a file:

```
Mat img = imread(filename)
```

If you read a jpg file, a 3 channel image is created by default.

If you need a grayscale image, use:

```
Mat img = imread(filename, 0);
```

Save an image to a file:

```
imwrite(filename, img);
```

[2] http://opencv.itseez.com/doc/user_guide

# OpenCV Example

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

using namespace cv;

int main(int argc, char* argv[]){
  Mat image = imread(argv[1]);
  namedWindow("Sample Window");
  imshow("Sample Window",image);
  waitKey(0);
  return 0;
}
```

# GCC

Compiling: To compile a C file, go back to your terminal window and type:

```
gcc -Wall -g filename.c -o filename
```

To compile a C++ file, type:

```
g++ -Wall -g filename.cc -o filename
```

The -Wall switch turns on all warnings, and the -g switch adds debugging information. The -o filename specifies the name of the resulting executable (the default is a.out).

If you get compiler errors, go back to your gedit window and fix them. Save your file and recompile until you can compile and link successfully.

# GCC

If you only want to compile a function, not link it with anything else, you type:

```
gcc -c foo.c
```

This compiles the source to object code, foo.o. Use man or info for more details on compiler options or visit `http://gcc.gnu.org` .

For programs with several external functions, a longer sequence is required. Say you have a main routine prog.c, and this calls external functions, funct1.c, funct2.c, and requires the library (static linking) lib.a.

# GCC

You would have to compile each function and the main:

```
gcc -c prog.c
gcc -c funct1.c
gcc -c funct2.c
```

Then you can link these together into an executable:

```
gcc -o prog prog.o funct1.o funct2.o lib.a
```

# GCC

Debugging

To have debugging information contained in the file, you must compile with the -g switch:

```
gcc -g -c main.c
g++ -g -c reciprocal.cpp
g++ -g -o reciprocal main.o reciprocal.o
```

# GCC

In the terminal window, type gdb filename where filename is the name of the executable. At the gdb prompt, type run, followed by any required command-line arguments. Your program will start execution, but instead of producing a core dump when it crashes, you will return to the gdb prompt with a message giving you information about where the crash occurred in your code.

# GCC

Useful gdb commands include help, where (produces a stack traceback), print (allows you to print values of expressions at the time of the crash), and quit.

# GCC

Example:
gdb reciprocal
(gdb) run
<seg fault>
(gdb) where
<lists function where error occurred>
(gdb) up 2
<prints line that called function>
(gdb) print argv[1]
<find null pointer>
You can set breakpoints:
(gdb) break main
You can run code with arguments
(gdb) run 7
Step over the call
(gdb) next
Step through code

# GCC

**Libraries** If you have a standard set of functions that are called by many programs, it is useful to build a library. There are two types, an archive or a static library and a shared library (or dynamically linked library).

Statically linked libraries are built using the ar command. For example:

```
gcc -c sin.c
gcc -c cos.c
gcc -c tan.c
ar cr libtrig.a sin.o cos.o tan.o
```

# GCC

It can be used in the following example:

```
gcc -c mainprog.c
gcc -o mainprog mainprog.o libtrig.a
```

or

```
gcc -c mainprog.c
gcc -o mainprog mainprog.o -L. -ltrig
```

# GCC

Note that the latter line requires that the library name start with lib. This is standard usage and probably should be followed. The compilers take a vast array of options, you can get a feel for this by typing man gcc (on the Linux systems for example). One typical option looks like:

-lstuff Use the library named stuff when linking.

The linker searches a standard list of directories for the library, which is actually a file named 'libstuff.a'. The linker then uses this file as if it had been specified precisely by name.

The directories searched include several standard system directories plus any that you specify with '-L'.

# GCC

Dynamically linked libraries are slightly more complicated. They provide smaller executables, a more modular code and some sharing of pages. To build the above as one, try

```
gcc -c -fPIC sin.c
gcc -c -fPIC cos.c
gcc -c -fPIC tan.c
gcc -shared -fPIC -o libtrig.so sin.o cos.o tan.o
gcc -o mainprog mainprog.o -L. -ltrig
```

(PIC = position independent code)

# GCC

Information:

ps - process status

nm - list symbols from object file

objdump - display information from object files

malloc, mtrace, ccmalloc & Electric Fence - tools to find dynamic memory errors

Glibc manual: `http://www.gnu.org/software/libc/manual/`

# GCC

Make

For serious software development, the make utility is indispensable. This utility allows you to compile, link, and install large software packages without having to type in long, arcane commands. For example, to compile and link a program consisting of four source code files with gcc, you must use the following command:

g++ file1.c file2.c file3.c file4.c o exefilename

Instead, by placing a list of file dependencies and compile/link commands into a Makefile , you can simply type make to accomplish the same task.

# Make

Instead, by placing a list of file dependencies and compile/link commands into a Makefile, you can simply type make to accomplish the same task. The make utility is a large, complex program with many options.

Make will look to see if formula already exists (don't compile if you don't have to). If not, it will then look for any files of the form formula.* where * is the wildcard and matches any string. In this case it finds formula.f. Since it has a ".f" extension, make assumes this is a fortran file. It then checks the rules on how to compile fortran, and executes the following for you:

f77 -o formula formula.f

The same is true for other languages. Use ".c" as the extension for C, and use ".C" as the extension for C++4.

# Makefile

Make will work for single file programs. However, for the example above (prog ), make will need some help. You need to include a Makefile in the directory to tell Make how to assemble the program. A sample Makefile for prog follows.

# Sample Makefile

```
# This is the makefile for the prog example.
# To compile the example program type: make
# The program will be named:

CC = cc
RM=\rm -f
OBJS = prog.o funct1.o funct2.o

prog: $(OBJS)
 $(CC) -o prog $(OBJS) lib.a

prog.o: prog.c
funct1.o: funct1.c
funct2.o: funct2.c

clean:
 -$(RM) prog
 -$(RM) *.o
 -$(RM) *~
```

# Make

Make is type of scripting language.

The comment character is #. The first two lines set the string variables CC and RM. The next line sets the variable OBJS. We could have written OBJS out in the lines below, but make will do the substitution and then it is easier for us to edit a single line.

The line "prog" has the actual linking statement which is essentially:
cc -o prog prog.o funct1.o funct2.o lib.a

# Make

There is a tab before the $(CC) text and should be typed as:
<tab>$(CC) -o prog $(OBJS) lib.a

Tab is a command in Make and tells Make that a “command line” follows. The lines below the link explain to make how to build the *.o files. The last gives a handy little “clean up the directory feature”. You would use this by typing make progmake clean will run the cleanup.

# Make

The command structure is the following:
target : list of things that the target depends on
TAB command to build target
TAB if needed, additional commands to build target
to build the program. Make will only compile the pieces that have been modified.

Here is a second sample Makefile to build a program named primes from source primes.cc:

# Typical Makefile

```
# Sample Makefile
CC= gcc
INS= install
INSDIR= /usr/local/bin
LIBDIR= -L/usr/local/lib
LIBS= -lXm -lSM -lICE -lXt -lX11
SRC= main.c sub1.c sub2.c sub3.c sub4.c
INC= main.h sub2.h
OBJ= main.o sub1.o sub2.o sub3.o sub4.o
PROG= mprog
```

# Typical Makefile continued

```
mprog: $(OBJ) $(INC)
 $(CC) -o $(PROG) $(OBJ) $(LIBDIR) $(LIBS)

main.o: main.c main.h
sub1.o: sub1.c
sub2.o: sub2.c sub2.h
sub3.o: sub3.c
sub4.o: sub4.c

install: $(PROG)
 $(INS) -g root -o root $(PROG) $(INSDIR)
 echo "mprog installed"
 touch .last_install
```

# Typical Makefile continued

To build a rule, you can use a combination of internal macros and the Makefile lines:

```
.c.o:

 $(CC) -c $<
```

# OpenCV Example

```
g++ -c -pipe -g -Wall -W -D_REENTRANT -DQT_CORE_LIB
-I/usr/share/qt4/mkspecs/linux-g++ -I.
-I/usr/include/qt4/QtCore -I/usr/include/qt4
-I/usr/local/OpenCV-2.3.1/bin/include -I.
-o hello2.o hello2.cpp

g++  -o hello2 hello2.o     -L/usr/local/lib
-L/usr/local/OpenCV-2.3.1/bin/ -lopencv_core
-lopencv_highgui -lopencv_imgproc -lopencv_features2d
-lopencv_calib3d -lopencv_objdetect -lopencv_contrib
-lopencv_legacy -lopencv_video -lQtCore -lpthread
```

# Makefile

```
OPTS = -pipe -g -Wall -W -D_REENTRANT -DQT_CORE_LIB

INCS = -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/include/qt4/QtCore \
-I/usr/include/qt4 -I/usr/local/OpenCV-2.3.1/bin/include -I.

LIBPATH = -L/usr/local/lib -L/usr/local/OpenCV-2.3.1/bin/

LIBS = -lopencv_core -lopencv_highgui -lopencv_imgproc -lopencv_features2d
-lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy \
-lopencv_video -lQtCore -lpthread


hello2: hello2.o
g++  -o hello2 hello2.o   $(LIBPATH) $(LIBS)

hello2.o:  hello2.cpp
g++ -c $(OPTS)   -o hello2.o hello2.cpp
```

# Automatic allocation and deallocation

```
#include "cv.h"
#include "highgui.h"

using namespace cv;

int main(int, char**)
{
    VideoCapture cap(0);
    if(!cap.isOpened()) return -1;

    Mat frame, edges;
    namedWindow("edges",1);
    for(;;)
    {
        cap >> frame;
        cvtColor(frame, edges, CV_BGR2GRAY);
        GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
        Canny(edges, edges, 0, 30, 3);
        imshow("edges", edges);
        if(waitKey(30) >= 0) break;
    }
    return 0;
}
```

The array frame is automatically allocated by the >> operator since the video frame resolution and the bit-depth is known to the video capturing module. The array edges is automatically allocated by the cvtColor function. It has the same size and the bit-depth as the input array. The number of channels is 1 because the color conversion code `CV_BGR2GRAY` is passed, which means a color to grayscale conversion.

# Basic Image Operations

- **cv::Mat** object replaces the original C standard IplImage and CvMat classes.
- All original functions and classes of the C standard OpenCV components in the Bradski book are still available and current. However you will need to read that book for it.
- **namedWindow** is used for viewing images.
- In general, default string as input with original image size set. Else, use string as input name and 0 for adjustable size.
- `http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/index.html`
- `http://livingston3.info/?wiki=opencv-manual`

# Structures

- Point,Point2f - 2D Point
- Size - 2D size structure
- Rect - 2D rectangle object
- RotatedRect - Rect object with angle
- Mat - image object

# Mat

Functions

- Mat.at<datatype>(row, col)[channel] - returns pointer to image location
- Mat.channels() - returns the number of channels
- Mat.clone() - returns a deep copy of the image
- Mat.create( rows, cols, TYPE) - re-allocates new memory to matrix
- Mat.cross(<Mat>) - computes cross product of two matricies
- Mat.depth() - returns data type of matrix
- Mat.dot(<Mat>) - computes the dot product of two matrices

# Types

Image TYPES

- The TYPE is a very important aspect of OpenCV
- Represented as CV_<Datatype>C<# Channels>

PixelTypes shows how the image is represented in data

- BGR - The default color of imread(). Normal 3 channel color
- HSV - Hue is color, Saturation is amount, Value is lightness. 3 channels
- GRAYSCALE - Gray values, Single channel
  - OpenCV requires that images be in BGR or Grayscale in order to be shown or saved.

# Tags

| OpenCV Tag | Representation | OpenCV Value |
|---|---|---|
| CV_8U | 8 bit unsigned integer | 0 |
| CV_8S | 8 bit signed integer | 1 |
| CV_16U | 16 bit unsigned integer | 2 |
| CV_16S | 16 bit signed integer | 3 |
| CV_32S | 32 bit signed integer | 4 |
| CV_32F | 32 bit floating point number | 5 |
| CV_64F | 64 bit floating point number | 6 |

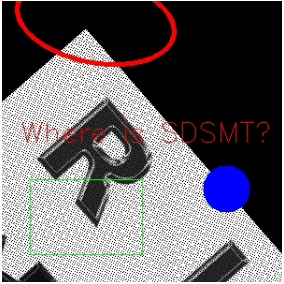
# Basic Image Operations

- void circle(image, Point(x,y),int rad, CV_BGR(b,g,r), int thickness=1)
- void ellipse(image, RotatedRect box, CV_BGR(b,g,r), int thickness=1)
- void line(image, Point(x,y), Point(x,y), CV_BGR(b,g,r), int thickness= 1)
- void rectangle(img, Point(x,y), Point(x,y), CV_BGR(b,g,r), int thickness)
  NOTE: negative thickness will fill in the rectangle

MORE...

http://opencv.willowgarage.com/documentation/cpp/core_drawing_functions.html

# Drawing ...




```
int main(int argc, char* argv[])
{
    Mat image(300,300,CV_8UC3);
    Mat sub = imread(argv[1]);
    float x,y, th=0.85398;

    for(int i=0; i<sub.rows;i++)
      for(int j=0; j<sub.cols;j++) {
         x = (j+0)*cos(th) - (i-0)*sin(th);
         y = (j+0)*sin(th) + (i-0)*cos(th);
         if(x+90 >= 0 && y+30 >= 0 && x+90 < image.cols && y+30 < image.rows)
             image.at<Vec3b>(y+30,x+90) = sub.at<Vec3b>(i,j);
         }

    //Draw an ellipse
    RotatedRect rotrect(Point(100,20),Size(90,170),101);
    ellipse(image, rotrect,Scalar(0,0,255),3);

    //draw a circle
    circle(image, Point(240,200),25,Scalar(255,0,0,0),-1);

    //Draw a box
    rectangle(image, Point(30,190),Point(150,270),Scalar(0,255,0),1);

    //Place text
    putText(image, "Where is SDSMT?", Point(20,150), FONT_HERSHEY_SIMPLEX,1,Scalar(0,0,255));

    namedWindow("Source");
    imshow("Source",image);
    waitKey(0);
    return 0;
}
```

# Drawing ...

- OpenCV allows you to use the mouse to interact with the screen. Note that this feature is from OpenCV 1.0 and is compatible with Mat objects.
- This program allows you to draw dots on the image.

```
struct OPTIONS{
  OPTIONS(): X(-1),Y(-1),drawing_dot(false){}
  int X;
  int Y;
  bool drawing_dot;
};
OPTIONS options;

void my_mouse_callback( int event, int x, int y, int flags, void* param ){
  IplImage* image = (IplImage*) param;

  switch( event ){

    case CV_EVENT_LBUTTONDOWN:
      options.X = x;
      options.Y = y;
      options.drawing_dot = true;
      break;
  }
}

int main(int argc, char* argv[]){

  IplImage* image = cvLoadImage(argv[1]);
  Mat frame = imread(argv[1]);

  namedWindow("Wyatt");
  cvSetMouseCallback("Wyatt", my_mouse_callback, (void*) image);

  //Take new points from user
  while(cvWaitKey(15) != 27){
    if( options.drawing_dot ){

      circle(frame,Point(options.X,options.Y),3,CV_RGB(255,255,0),2);
      options.drawing_dot = false;
    }

    imshow("Wyatt",frame);
    waitKey(10);
  }
  cvReleaseImage(&image);

  return 0;
```

# Converting colorspaces

cvtColor( image, image, code)

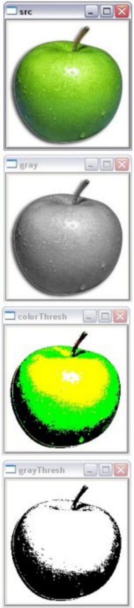
Codes: CV_<colorspace>2<colorspace>

Examples

- CV_BGR2GRAY
- CV_BGR2HSV
- CV_BGR2LUV

# Basic Image and Legacy Operations

```
int main()
{
  IplImage* src;
  IplImage* colorThresh;
  IplImage* gray;
  IplImage* grayThresh;
  int threshold = 120, maxValue = 255;
  int thresholdType = CV_THRESH_BINARY;
  src = cvLoadImage("apple.jpg", 1);
  colorThresh = cvCloneImage( src );
  gray = cvCreateImage( cvSize(src->width, src->height), IPL_DEPTH_8U, 1 );
  cvCvtColor( src, gray, CV_BGR2GRAY );
  grayThresh = cvCloneImage( gray );
  cvNamedWindow( "src", 1 );
  cvNamedWindow( "gray", 1 );
  cvShowImage( "src", src );
  cvShowImage( "gray", gray );
  cvThreshold(src, colorThresh, threshold, maxValue, thresholdType);
  cvThreshold(gray, grayThresh, threshold, maxValue, thresholdType);
  cvNamedWindow( "colorThresh", 1 );
  cvNamedWindow( "grayThresh", 1 );
  cvShowImage( "colorThresh", colorThresh );
  cvShowImage( "grayThresh", grayThresh );
  cvWaitKey(0);
  cvDestroyWindow( "src" );
  cvDestroyWindow( "colorThresh" );
  cvDestroyWindow( "gray" );
  cvDestroyWindow( "grayThresh" );
  cvReleaseImage( &src );
  cvReleaseImage( &colorThresh );
  cvReleaseImage( &gray );
  cvReleaseImage( &grayThresh );
  return 0;
}
```



# Example

```
int main()
{
    Mat image= imread("img2.jpg");
    Mat gimage, contours, contoursInv, contoursInv2;
    cv::Rect* rect=0;
    std::vector<cv::Vec2f> lines;
    vector<Vec4i> lines2;

    if (!image.data) {
        cout << "Warning Mr. Robbenson! Dr. Smith deleted the file." << endl;
        return 1;
    }

    cvtColor(image,gimage,CV_BGR2GRAY);
    blur( gimage, gimage, Size(3,3) );
    cout << "size:  " << image.size().height << " , " << image.size().width << endl;
    namedWindow("Source");
    Canny(gimage,contours,125,350);
    threshold(contours, contoursInv, 128,255, THRESH_BINARY_INV);
    contoursInv2 = contoursInv.clone();
    int count = countNonZero(contoursInv);
    cout << "Count:  "  << count << endl;
    floodFill(contoursInv, Point(490,100), Scalar(0,0,0), rect, 20, 20 );
    int count2 = countNonZero(contoursInv);
    std::cout << "Count:  "  << count2 << endl;
    floodFill(contoursInv, Point(10,100), Scalar(0,0,0), rect, 20, 20 );
    int count3 = countNonZero(contoursInv);
    cout << "Count:  "  << count3 << endl;
    cout << "Right = " << (count - count2) << endl;
    cout << "Left = " << (count2 - count3) << endl;
    imshow("Source",contoursInv);
    waitKey(0);
 return 1;
}
```

# First Example - Video

```
int main()
{
    VideoCapture cap(0);
    if(!cap.isOpened()) return -1;

    Mat frame, edges;
    namedWindow("edges",1);
    for(;;)
    {
        cap >> frame;
        cvtColor(frame, edges, CV_BGR2GRAY);
        GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
        Canny(edges, edges, 0, 30, 3);
        imshow("edges", edges);
        if(waitKey(30) >= 0) break;
    }



    return 1;
}
```

# More OpenCV

Image Normalization:

- normalize(imagein, imageout, low, high, method);
- Image normalization is the process of stretching the range of an image from [a, b] to [c, d].
- This is incredibly important for visualization because if the image is beyond [0,255] it will cause truncation or unsightly effects.

Thresholding:

- threshold(image, image, thresh, maxVal, CODE);
- CODE - this is the method of thresholding. Different actions will be taken depending on this code.

# More OpenCV

Edge Detection

- Sobel Edge Detection
  void cv::Sobel(image in, image out, CV_DEPTH, dx, dy);
- Scharr Edge Detection
  void cv::Scharr(image in, image out, CV_DEPTH, dx, dy);
- Laplacian Edge Detection
  void cv::Laplacian( image in, image out, CV_DEPTH);

# More OpenCV

Image Smoothing

- Image smoothing is used to reduce the the sharpness of edges and detail in an image. OpenCV includes most of the commonly used methods.
- void GaussianBlur(imagein, imageout, Size ksize, sig);
  Note that there are more options,
  however this should keep things simple
- void medianBlur (imagein, imageout, Size ksize);
- Other functions include generic convolution, separable convolution, dilate, and erode.