Applicazioni pratiche della visione artificiale

Parte 1.4 – OpenCV

Curato da: Ing. Francesco La Rosa
Università di Messina – Facoltà di Ingegneria

Corso di Calcolatori II – A.A. 2003/2004
Ing. Giancarlo Iannizzotto

IplImage

IPL image header

```
typedef struct _IplImage
{
        int nSize; /* sizeof(IplImage) */
        int ID; /* version (=0)*/
        int nChannels; /* Most of OpenCV functions support
                          1,2,3 or 4 channels */
        int alphaChannel; /* ignored by OpenCV */
        int depth; /* pixel depth in bits: IPL_DEPTH_8U,
                      IPL_DEPTH_8S, IPL_DEPTH_16S,
                      IPL_DEPTH_32S, IPL_DEPTH_32F and
                      IPL_DEPTH_64F are supported */
        char colorModel[4]; /* ignored by OpenCV */
        char channelSeq[4]; /*  */
```

IplImage

```
int dataOrder; /* 0 - interleaved color channels, 1 - separate
                  color channels. cvCreateImage can only create
                  interleaved images */
int origin; /* 0 - top-left origin,
               1 - bottom-left origin (Windows bitmaps style) */
int align; /* Alignment of image rows (4 or 8).
              OpenCV ignores it and uses widthStep instead */
int width; /* image width in pixels */
int height; /* image height in pixels */
struct _IplROI *roi; /* image ROI. when it is not NULL, this
                        specifies image region to process */
struct _IplImage *maskROI; /* must be NULL in OpenCV */
void *imageId; /*  */
```


IplImage

```
struct _IplTileInfo *tileInfo;
int imageSize; /* image data size in bytes
(=image->height*image->widthStep
 in case of interleaved data)*/
char *imageData; /* pointer to aligned image data */
int widthStep; /* size of aligned image row in bytes */
int BorderMode[4]; /* border completion mode, ignored by
OpenCV */
int BorderConst[4];
char *imageDataOrigin; /* pointer to a very origin of image
data (not necessarily aligned) -
it is needed for correct image deallocation */
}IplImage;
```

CvMat

Multi-channel matrix

```
typedef struct CvMat
{  int type; /* CvMat signature (CV_MAT_MAGIC_VAL),
                element type and flags */
   int step; /* full row length in bytes */
   int* refcount; /* underlying data reference counter */
   union
   {     uchar* ptr;
         short* s;
         int* i;
         float* fl;
         double* db;
    } data; /* data pointers */ …
```


CvMat

```
#ifdef __cplusplus
   union
   {     int rows;
         int height; };
   union
   {     int cols;
         int width;};
#else
         int rows; /* number of rows */
         int cols; /* number of columns */
#endif
} CvMat;
```

CvMatND

Multi-dimensional dense multi-channel array

```
typedef struct CvMatND
{  int type; /* CvMatND signature
                (CV_MATND_MAGIC_VAL), element type
                and flags */
   int dims; /* number of array dimensions */
   int* refcount; /* underlying data reference counter */
   union
   {     uchar* ptr;
         short* s;
         int* i;
         float* fl;
         double* db;
    } data; /* data pointers */…
```


CvMatND

```
/* pairs (number of elements, distance between elements in
   bytes) for every dimension */
   struct
   {
   int size;
   int step;
   }
   dim[CV_MAX_DIM];

} CvMatND;
```

Helper Structures

2D point with integer coordinates: CvPoint

```
typedef struct CvPoint
{
    int x; /* x-coordinate, usually zero-based */
    int y; /* y-coordinate, usually zero-based */
}
CvPoint;
/* the constructor function */
inline CvPoint cvPoint( int x, int y );
/* conversion from CvPoint */
inline CvPoint2D32f cvPointTo32f( CvPoint point );
```


Helper Structures

CvPoint

Esempio

```
…
    int x=10, y=20;
    CvPoint punto; //dichiarazione di una variabile
                   // di tipo CvPoint
    punto=cvPoint(x, y); //inizializzo la variabile
                         // "punto" tramite la macro cvPoint
…
```

Helper Structures

CvPoint* :

- CvPoint2D32f
  - /* the constructor function */
    inline CvPoint2D32f cvPoint2D32f(double x, double y);
  - /* conversion from CvPoint */
    inline CvPoint2D32f cvPointTo32f(CvPoint point);
- CvPoint3D32f
  - /* the constructor function */
    inline CvPoint3D32f cvPoint3D32f(double x, double y, double z);

Helper Structures

Pixel size of a rectangle: CvSize

```
typedef struct CvSize
{
    int width; /* width of the rectangle */
    int height; /* height of the rectangle */
}
CvSize;
/* the constructor function */
inline CvSize cvSize( int width, int height );
```

Helper Structures

Offset and size of a rectangle: CvRect

```
typedef struct CvRect
{
    int x; /* x-coordinate of the left-most rectangle corner[s] */
    int y; /* y-coordinate of the top-most or bottom-most
          rectangle corner[s] */
    int width; /* width of the rectangle */
    int height; /* height of the rectangle */
}
CvRect;
/* the constructor function */
inline CvRect cvRect( int x, int y, int width, int height );
```


Helper Structures

***A container for 1-,2-,3- or 4-tuples of numbers*:** CvScalar

```
typedef struct CvScalar
{
    double val[4];
}
CvScalar;

/* the constructor function: initializes val[0] with val0, val[1]
   with val1 etc. */
inline CvScalar cvScalar( double val0, double val1=0,
                          double val2=0, double val3=0 );
```

Helper Structures

CvScalar

```
/* the constructor function: initializes val[0]...val[3] with val0123 */
inline CvScalar cvScalarAll( double val0123 );
/* the constructor function: initializes val[0] with val0,
   val[1]...val[3] with zeros */
inline CvScalar cvRealScalar( double val0 );
```


cvMat

Initializes matrix header

CvMat cvMat(int rows, int cols, int type, void* data = 0);

rows
: Number of rows in the matrix

cols
: Number of columns in the matrix

type
: Type of the matrix elements (see CreateMat)

data
: Optional data pointer assigned to the matrix header

cvCreateMat

Creates new matrix

CvMat* cvCreateMat(int rows, int cols, int type);

rows

Number of rows in the matrix.

cols

Number of columns in the matrix.

type

Type of the matrix elements. Usually it is specified in form *CV_<bit_depth>(S|U|F)C<number_of_channels>*, for example: *CV_8UC1* means an 8-bit unsigned single-channel matrix, *CV_32SC2* means a 32-bit signed matrix with two channels.

cvReleaseMat

Deallocates matrix

void cvReleaseMat(CvMat** mat);

mat

Double pointer to the matrix.

Note:

The function cvReleaseMat decrements the matrix data reference counter and releases matrix header.

cvSetIdentity

Initializes scaled identity matrix

void cvSetIdenity(CvArr* A, CvScalar S);

A

The matrix to initialize (not necesserily square).

S

The value to assign to the diagonal elements.

The function cvSetIdentity initializes scaled identity matrix:

A(i,j)=S if i=j,

0 otherwise

N.B.: typedef void CvArr;

cvTrace

Returns trace of matrix

CvScalar cvTrace(const CvArr* A);

A

The source matrix.

The function cvTrace returns sum of diagonal elements

of the matrix *A*.

Note:

$tr(A)=sum_i A(i,i)$

cvScaleAdd

Calculates sum of scaled array and another array

void cvScaleAdd(const CvArr* A, CvScalar S, const CvArr* B, CvArr* C);

#define cvMulAddS cvScaleAdd

A

The first source array.

S

Scale factor for the first array.

B

The second source array.

C

The destination array

Note:

The function cvScaleAdd calculates sum of scaled array and another array:

C(I)=A(I)*S + B(I)

cvMatMulAdd

Calculates shifted matrix product

void cvMatMulAdd(const CvArr* A, const CvArr* B, const CvArr* C, CvArr* D);

A

The first source array.

B

The second source array.

C

The third source array (shift). Can be NULL, if there is no shift.

D

The destination array

Note:

The function cvMatMulAdd calculates matrix product of two matrices and adds the third matrix to the product:

D = A*B + C or $D(i,j)=\sum_k (A(i,k)*B(k,j)) + C(i,j)$

cvDet

Returns determinant of matrix

CvScalar cvDet(const CvArr* A);

A

The source matrix.

The function cvDet returns determinant of the square matrix *A*.

The direct method is used for small matrices and Gaussian elimination is used for larger matrices.

cvTranspose

Transposes matrix

void cvTranspose(const CvArr* A, CvArr* B);

#define cvT cvTranspose

A

The source matrix.

B

The destination matrix.

Note:

The function cvTranspose transposes matrix *A*:

B(i, j)=A(j, i)

Note that no complex conjugation is done in case of complex matrix.

cvSolve

Solves linear system or least-squares problem

int cvSolve(const CvArr* A, const CvArr* B, CvArr* X, int method);

#define cvInv cvSolve

A

The source matrix (Ax=B).

B

The right-hand part of the linear system (Ax=B).

method

The solution (matrix inversion) method:

CV_LU - Gaussian elimination with optimal pivot element chose

CV_SVD - Singular decomposition method.

cvSVD

The function cvSVD decomposes matrix *A* into a product of a diagonal matrix and two orthogonal matrices:

$A=U*W*V^T$

Where ***W*** is diagonal matrix of *singular values* that can be coded as a 1D vector of singular values. All the singular values are non-negative and sorted (together with *U* and *V* columns) in descenting order.

cvSVD

SVD algorithm is numerically robust and its typical applications include:

- accurate eigenvalue problem solution when matrix *A* is square, *symmetric* and *positively defined* matrix, for example, when it is a covariation matrix. *W* in this case will be a vector of eigen values, and *U=V* is matrix of eigen vectors (thus, only one of *U* or *V* needs to be calculated if the eigen vectors are required)
- accurate solution of poor-conditioned linear systems

cvSVD

- least-squares solution of *overdetermined linear systems*. This and previous is done by cvSolve function with *CV_SVD* method
- **accurate calculation** of different matrix characteristics such as rank (number of non-zero singular values), condition number (ratio of the largest singular value to the smallest one), determinant (absolute value of determinant is equal to the product of singular values). All the things listed in this item do not require calculation of *U* and *V* matrices.

cvSVD

Performs singular value decomposition of real floating-point matrix

void cvSVD(CvArr* A, CvArr* W, CvArr* U=0, CvArr* V=0, int flags=0);

A

Source *M*×*N* matrix.

W

Resulting singular value matrix (*M*×*N* or *N*×*N*) or vector (*N*×*1*).

U

Optional left orthogonal matrix (*M*×*M* or *M*×*N*). If CV_SVD_U_T is specified, the number of rows and columns in the sentence above should be swapped.

cvSVD

Performs singular value decomposition of real floating-point matrix

void cvSVD(CvArr* A, CvArr* W, CvArr* U=0, CvArr* V=0, int flags=0);

V

Optional right orthogonal matrix (*N*×*N*)

flags

Operation flags; can be 0 or combination of the following:

- *CV_SVD_MODIFY_A* enables modification of matrix *A* during the operation. It speeds up the processing.
- *CV_SVD_U_T* means that the tranposed matrix *U* is returned. Specifying the flag speeds up the processing.
- *CV_SVD_V_T* means that the tranposed matrix *V* is returned. Specifying the flag speeds up the processing.

cvEigenVV

Computes eigenvalues and eigenvectors of symmetric matrix

void cvEigenVV(CvArr* A, CvArr* evects, CvArr* evals, double eps);

A

The source symmetric square matrix. It is modified during the processing.

evects

The output matrix of eigenvectors, stored as a subsequent rows.

eps

Accuracy of diagonalization (typically, DBL_EPSILON=10^{-15} is enough).

A*evects(i,:)' = evals(i)*evects(i,:)' ;

N.B.:The contents of matrix *A* is destroyed by the function.

cvDFT

Performs forward or inverse Discrete Fourier transform of 1D or 2D floating-point array

void cvDFT(const CvArr* src, CvArr* dst, int flags);

src

Source array, real or complex.

dst

Destination array of the same size and same type as the source.

flags

Transformation flags, 0 or a combination of the following flags:

CV_DXT_INVERSE - perform inverse transform

CV_DXT_SCALE - divide the result by the number of array elements

For convenience, the constant CV_DXT_FORWARD may be used instead of literal 0.

Discrete Cosine Transform

- Forward Cosine transform of 1D vector of N elements:
  $y = C \cdot x$, where C_{ik} =sqrt((i= =0?1:2)/N)·cos(Pi·(2i+1)·k/N), j=sqrt(-1);
- Inverse Cosine transform of 1D vector of N elements:
  $x = C^{-1} \cdot y = C^{T} \cdot y$
- Forward Cosine transform of 2D vector of M×N elements:
  $Y = C \cdot X \cdot C^{T}$
- Inverse Fourier transform of 2D vector of M×N elements:
  $X = C^{T} \cdot Y \cdot C$

cvDCT

Performs forward or inverse Discrete Cosine transform of 1D or 2D floating-point array

void cvDCT(const CvArr* src, CvArr* dst, int flags);

src

Source array, real 1D or 2D array.

dst

Destination array of the same size and same type as the source.

flags

Transformation flags, 0 or a combination of the following flags:

CV_DXT_INVERSE - perform inverse transform

CV_DXT_SCALE - divide the result by the number of array elements

For convenience, the constant CV_DXT_FORWARD may be used instead of literal 0.

Miscellanea

```
void CProvaDoc::OnMathSet()
{
    double a[] = { 1, 2, 3, 4,  5, 6, 7, 8,  9, 10, 11, 12 };
    double b[] = { 1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 12 };
    CvMat Ma, Mb, *Mc, *T_Ma;
    Ma=cvMat( 3, 4, CV_64FC1, a ); Mb=cvMat( 4, 3, CV_64FC1, b );
    Mc=cvCreateMat( 3, 3, CV_64FC1); //matrice 3x3 di double
    T_Ma=cvCreateMat( 4, 3, CV_64FC1);
    cvMatMulAdd( &Ma, &Mb, 0, Mc ); //Mc cvMat now contains
                                    //product of a (3x4) and b (4x3) matrices
```


Miscellanea

```
…
    CvScalar trace_a, trace_b, trace_c, valore; double det;
    valore=cvScalar(100);
    trace_a=cvTrace( &Ma );  trace_b=cvTrace( &Mb );
    trace_c=cvTrace( Mc );//18
    cvTranspose( &Ma, T_Ma);
    det=cvDet( Mc ); //sul manuale c'è scritto che cvDet
                     //restituisce un CvScalar :-(
    cvSetIdentity( Mc, valore );
    cvReleaseMat(&Mc);
}
```

Esempio

```
void CProvaDoc::OnMathSolve_SVD()
{ IplImage * image= cvCreateImage( cvSize(width,height), 8, 3 );
  int width=800, height=600, i,j;
  double MATRIX[12], UNO[6];
  CvScalar value=cvScalar(255,255,255);
  cvSet(image,value);
  CvPoint A=cvPoint(100,0); CvPoint B=cvPoint(700,600); CvPoint pt;
  cvLine( image, A, B, CV_RGB(255,0,0));//y=1*x-100; A(100,0); B(700,600)
  CvMat  Ma, Mb,*MX,*Mr;
  Mr=cvCreateMat( 6, 2, CV_64FC1);
  CvRandState rng;//dichiaro il generatore di numeri casuali
  cvRandInit( &rng, 15, 0, 10, CV_RAND_NORMAL );//inizializzo rng
  cvRand( &rng, Mr ); …
```

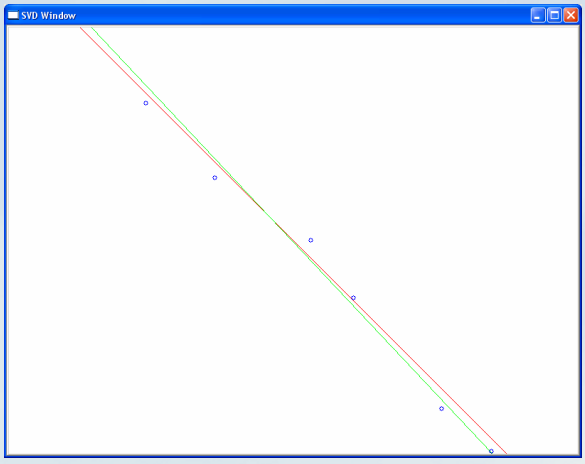

Esempio

```
… for( i=200,j=0;i<800;i+=100,j++)
    {
      MATRIX[2*j]=(double)i+Mr->data.db[2*j];  //rumore additivo
      MATRIX[2*j+1]=(double)i-100+Mr->data.db[2*j+1];
      UNO[j]=1;
      pt.x= (int)MATRIX[2*j];
      pt.y= (int)MATRIX[2*j+1];
      cvCircle(image,pt, 3, CV_RGB(0,0,255),1);
    }
      Ma=cvMat( 6, 2, CV_64FC1, MATRIX );
      Mb=cvMat( 6, 1, CV_64FC1, UNO );
      MX=cvCreateMat( 2, 1, CV_64FC1);
      cvSolve( &Ma, &Mb, MX, CV_SVD);
      cvNamedWindow("SVD Window",CV_WINDOW_AUTOSIZE);    …
```

Esempio

```
…//ax+by=1;  y=0, x=1/a; y=600,x=(1-600*b)/a;
double a=MX->data.db[0]; double b=MX->data.db[1];
A.x=(int)(1/a); A.y=0;
B.x=(int)((1-600*b)/a); B.y=600;
cvLine( image, A, B, CV_RGB(0,255,0));
char risultati[100];
sprintf(risultati,"a= %f, b=%f",a,b);
AfxMessageBox(risultati);
cvShowImage("SVD Window",image);
cvWaitKey(0);
cvDestroyAllWindows();
cvReleaseImage(&image);}
```


Esempio



Drawing Functions

cvLine - Draws simple or thick line segment

void cvLine(CvArr* img, CvPoint pt1, CvPoint pt2, double color, int thickness=1, int connectivity=8);

img -The image.

pt1, pt2 – First & second point of the line segment.

color -Line color (RGB) or brightness (grayscale image).

thickness Line thickness.

connectivity Line connectivity, 8 (by default) or 4.

N.B.: To specify the line color, the user may use the macro CV_RGB(r, g, b).

Drawing Functions

cvRectangle - Draws simple, thick or filled rectangle

void cvRectangle(CvArr* img, CvPoint pt1, CvPoint pt2, double color, int thickness=1);

img -The image.

pt1, pt2 - Two opposite rectangle vertices.

color - Line color (RGB) or brightness (grayscale image).

thickness - Thickness of lines that make up the rectangle. Negative values, e.g. CV_FILLED, make the function to draw a filled rectangle.

Drawing Functions

cvCircle - Draws simple, thick or filled circle

void cvCircle(CvArr* img, CvPoint center, int radius,
double color, int thickness=1);

img -The image.

center - Center of the circle.

radius - Radius of the circle.

color - Circle color (RGB) or brightness (grayscale image).

thickness - Thickness of the circle outline if positive, otherwise indicates that a filled circle has to be drawn.

Drawing Functions

cvEllipse - Draws simple or thick elliptic arc or fills ellipse sector

void cvEllipse(CvArr* img, CvPoint center, CvSize axes,
double angle, double startAngle, double endAngle,
double color, int thickness=1);

img - Image.

center - Center of the ellipse.

axes - Length of the ellipse axes.

angle - Rotation angle.

startAngle - Starting angle of the elliptic arc.

endAngle - Ending angle of the elliptic arc.

Drawing Functions

cvEllipse - Draws simple or thick elliptic arc or fills ellipse sector

```
void cvEllipse( CvArr* img, CvPoint center, CvSize axes,
                double angle, double startAngle, double endAngle,
                double color, int thickness=1 );
```

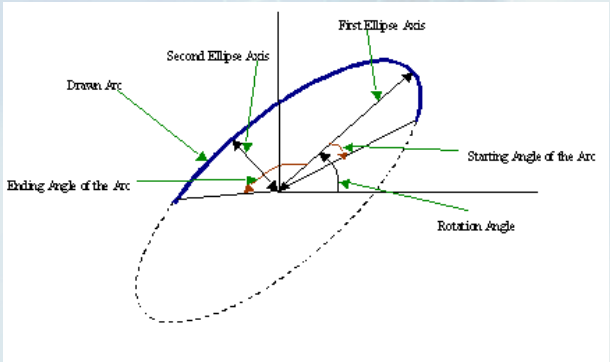
color - Ellipse color (RGB) or brightness (grayscale image).

thickness - Thickness of the ellipse arc.

Note: The function cvEllipse draws a simple or thick elliptic arc or fills an ellipse sector. The arc is clipped by ROI rectangle. All the angles are given in degrees.

Drawing Functions

cvEllipse - Draws simple or thick elliptic arc or fills ellipse sector.



Drawing Functions

cvPolyLine - Draws simple or thick polygons

void cvPolyLine (CvArr* img, CvPoint** pts, int* npts, int contours, int isClosed, double color, int thickness=1, int connectivity=8);

img - Image.

pts - Array of pointers to polylines.

npts - Array of polyline vertex counters.

contours - Number of polyline contours.

isClosed - Indicates whether the polylines must be drawn closed. If closed, the function draws the line from the last vertex of every contour to the first vertex.

color - Polygon color (RGB) or brightness (grayscale image).

thickness - Thickness of the polyline edges.

connectivity - The connectivity of polyline segments, 8 (by default) or 4.

Drawing Functions

cvFillConvexPoly - Fills convex polygon

void cvFillConvexPoly(CvArr* img, CvPoint* pts, int npts, double color);

img - Image.

pts - Array of pointers to a single polygon.

npts - Polygon vertex counter.

color -Polygon color (RGB) or brightness (grayscale image).

Note: The function cvFillConvexPoly fills convex polygon interior.

Drawing Functions

cvInitFont - Initializes font structure

void cvInitFont(CvFont* font, CvFontFace fontFace,
float hscale, float vscale, float italicScale, int thickness);

font - Pointer to the font structure initialized by the function.

fontFace - Font name identifier. Only the font CV_FONT_VECTOR0 is currently supported.

hscale - Horizontal scale. If equal to 1.0f, the characters have the original width depending on the font type. If equal to 0.5f, the characters are of half the original width.

vscale - Vertical scale. If equal to 1.0f, the characters have the original height depending on the font type. If equal to 0.5f, the characters are of half the original height.

Drawing Functions

cvInitFont - Initializes font structure

void cvInitFont(CvFont* font, CvFontFace fontFace,
float hscale, float vscale, float italicScale, int thickness);

italicScale - Approximate tangent of the character slope relative to the vertical line. Zero value means a non-italic font, 1.0f means » 45° slope, etc.

thickness - Thickness of lines composing letters outlines.

Note: The function cvInitFont initializes the font structure that can be passed further into text drawing functions.

Drawing Functions

cvPutText - Draws text string

void cvPutText(CvArr* img, const char* text, CvPoint org,
CvFont* font, int color);

img Input image.

text String to print.

org Coordinates of the bottom-left corner of the first letter.

font Pointer to the font structure.

color Text color (RGB) or brightness (grayscale image).

Drawing Functions

Esempio

```
void CProvaDoc::OnDrawMisc()
{
   IplImage * image, * imageR, * imageG, * imageB;
   CvPoint pt1,pt2,center,rect1,rect2,center_ellipse,point_text;
   CvPoint tri[3]; CvFont font;
   CvSize axes=cvSize(100,50);
   char stringa[]="Calc 2 - OpenCV";
   pt1.x=100; pt1.y=100; //estremo A della linea
   pt2.x=200; pt2.y=300; //estremo B della linea
   center.x=300; center.y=200; //centro del cerchio
   rect1.x=400; rect1.y=400; //top-left del rettangolo
   rect2.x=480; rect2.y=480; //bottom-rigth del rettangolo
   center_ellipse.x=100; center_ellipse.y=400;//centro dell'ellisse
```

Drawing Functions
Esempio

```
tri[0].x=250;  tri[0].y=10;  //vertice A del triangolo
tri[1].x=350;  tri[1].y=110; //vertice B del triangolo
tri[2].x=150;  tri[2].y=110; //vertice C del triangolo
point_text.x=30; point_text.y=460; //top-left del testo
CvPoint vettore[6];//vertici del poligono
vettore[0]=pt1; vettore[1]=pt2; vettore[2]=center;
vettore[3]=rect1; vettore[4]=rect2; vettore[5]=center_ellipse;
CvPoint * poly;
poly=&vettore[0];
```


Drawing Functions
Esempio

```
int npoint[1];  npoint[0]=6;
int width = 500, height = 500;
char wndname[] = "Drawing Demo";
image = cvCreateImage( cvSize(width,height), 8, 3 );
imageR = cvCreateImage( cvSize(width,height), 8, 1 );
imageG = cvCreateImage( cvSize(width,height), 8, 1);
imageB = cvCreateImage( cvSize(width,height), 8, 1 );
```

Drawing Functions
Esempio

```
cvCvtPixToPlane( image, imageB, imageG, imageR,0 );

cvSet(imageR, cvScalar(255));
cvSet(imageG, cvScalar(255));
cvSet(imageB, cvScalar(255));

cvCvtPlaneToPix( imageB, imageG,imageR, 0,image );

cvLine( image,pt1, pt2, CV_RGB(255,0,0) );
cvCircle(image, pt1, 5, CV_RGB(0,255,0), 1);
cvCircle(image,pt2, 5, CV_RGB(0,0,255),1);
```


Drawing Functions
Esempio

```
cvCircle(image,center, 40, CV_RGB(0,0,0),-1);
cvRectangle(image,rect1,rect2,CV_RGB(0,0,255),-1);
cvEllipse(image,center_ellipse,axes,45,-20,90,CV_RGB(255,0,255),-1);

cvPolyLine(image, &poly, npoint,1,1,CV_RGB(0,255,0));
cvFillConvexPoly(image,tri,3,CV_RGB(255,255,0));

cvInitFont(&font, CV_FONT_VECTOR0,0.5,0.5,0,2);
cvPutText(image,stringa,point_text,&font,CV_RGB(255,0,255));

cvNamedWindow(wndname, 1 );
```

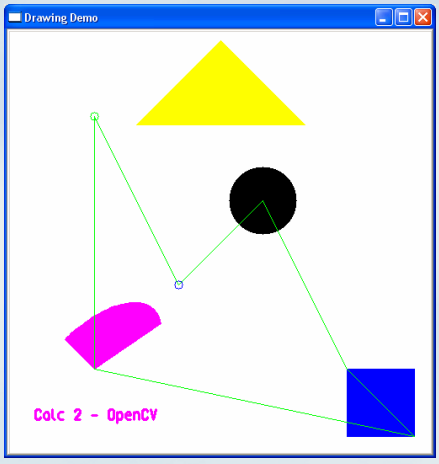
Drawing Functions
Esempio

```
    cvShowImage(wndname,image);

    cvWaitKey(0);

    cvDestroyWindow(wndname);
    cvReleaseImage(&image);
    cvReleaseImage(&imageR);
    cvReleaseImage(&imageG);
    cvReleaseImage(&imageB);
}
```


Drawing Functions
Esempio



Drawing Functions

Nel programma drawing.c contenuto nella cartella sample di OpenCV sono usate le principali funzionalità di drawing messe a disposizione da OpenCV.