Visión por Computador (1782)

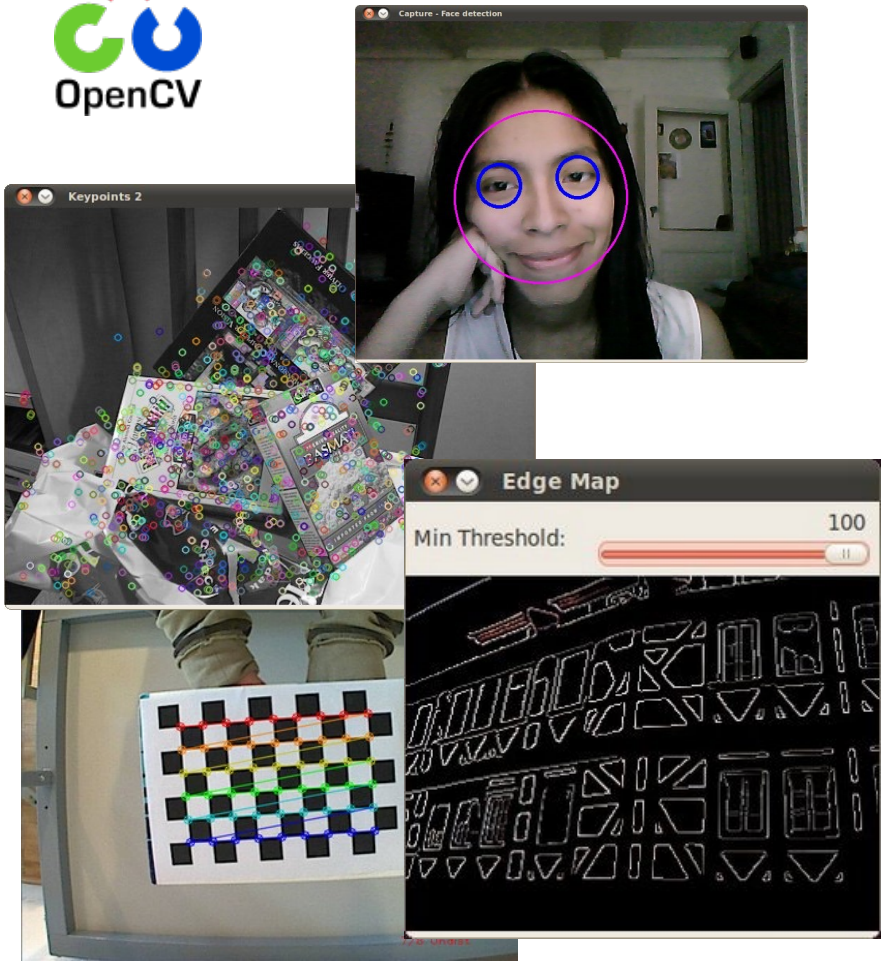
Herramientas de programación de aplicaciones
OpenCV4

Luis M. Jiménez

Lab. Automática, Robótica y Visión por Computador
Universidad Miguel Hernández
http://arvc.umh.es

Prácticas

- OpenCV: http://opencv.org



- Multiplataforma: Windows/Linux/OSX Android/iOS
- Interfaz: C/C++/Java/Python

- Adquisición imágenes/video
- Procesamiento 2D
- Extracción características
- Machine Learning
- Reconocimiento - Clasificación
- Aceleración GPU
- Calibración 3D
- Localización - Reconstrucción 3D

Instalación OpenCV

- Windows 7/8/10
  - Compiladores:
    - Microsoft Visual Studio 2015 -> VC14 (Windows 7/8/10) (OpenCV 3/4)
    - **Microsoft Visual Studio 2017** -> **VC15** (Windows 7/8/10) (OpenCV 3/4)
  - Librería OpenCV 4.5:
    - Descargaremos la versión compilada estable: 4.5.5 (VC14/VC15)
    - Basta descomprimir el fichero
- Linux/Mac
  - Bajar los fuentes
  - Utilizar ***CMake*** para compilar la librería y las aplicaciones
  - Tutoriales:
    - http://umh1782.edu.umh.es/opencv/

Programación en OpenCV (C++)

- Estructuras/Clases básicas: OpenCV prefijo -> cv::
  - cv::**Point**, cv::**Point2f**, cv::**Point3f** Puntos especificados por sus coordenadas
    - Componentes:
      - 2D: int x, y; float x, y;
      - 3D float x, y, z;
  - cv::**Size** Tamaño de una imagen
    - Componentes: int width, height;
  - cv::**Vec** (*template*): **Vec3b**, **Vec3s, Vec3f, Vec3d** ….
    - Describen los valores de un pixel (multicanal). Indexación mediante operador []
  - cv::**Scalar** equivalente a cv::**Vec4d**
  - cv::**Range** rangos de filas o columnas

    Componentes: int start, end; all()
  - cv::**Rect**: rectángulo dentro de una imagen
    - Componentes: int x, y, width, height;

  - std::**vector**: clase de la librería estándar ***std*** using namespace std;
    - Se utiliza para listas de puntos característicos

Programación en OpenCV (C++)

- Estructuras/Clases básicas:
  - cv::**Mat**
    - N-dimensional array, para almacenar imágenes
    - Algunos Componentes:
      - ***rows***, ***cols*** : (int)
      - ***channels*** : 1: grayscale, 3: BGR
      - ***depth*** : CV_<depth>C<num chan> (**CV_8UC1**) (CV_16U,CV_32F, CV_64F)
      - ***data*** : puntero al buffer memoria (***uchar*** *)
    - Los **constructores** y operador asignación (**=**) devuelven cabeceras (*Mat header*) (sin datos)
    - Algunos Métodos:
      - *mat.**at**<datatype>(row, col)[channel]* – acceso al valor de un pixel
      - *mat.**clone**()* – devuelve una copia de la imagen
      - *mat.**copyTo**(<Mat>)* – copia el contenido de un matriz en otra
      - *mat.**convertTo**(<Mat>, type)* – convierte el tipo de la matriz
      - *mat.**size**()* – Devuelve el tamaño (clase cv::***Size***)
      - *mat.**empty**()* – indica si la matriz está vacía (sin memoria)
      - *mat.**row**(y) , mat.**col**(x)* – Devuelve una cabeceara (***Mat header***) a una fila o columna de una matriz

Programación en OpenCV (C++)

- Estructuras/Clases básicas:
  - cv::**Mat**
    - Métodos Estáticos (no precisan declarar un objeto):
      - cv::Mat::***ones****(row, col, type)* – devuelve una matriz de unos
      - cv::Mat::***zeros****(row, col, type)* – devuelve una matriz de ceros
      - cv::Mat::***eye****(row, col, type)* – devuelve una matriz identidad
      - cv::Mat::***diag****(const Mat& d)* – crea una matriz diagonal a partir de un vector

    - Operaciones con matrices:
      - *mat.**t**()* – traspuesta
      - *mat.**inv**()* – inversa
      - *mat.**mul**(<Mat>)* – multiplicación
      - *mat.**empty**()* – bool (indica si tiene datos)

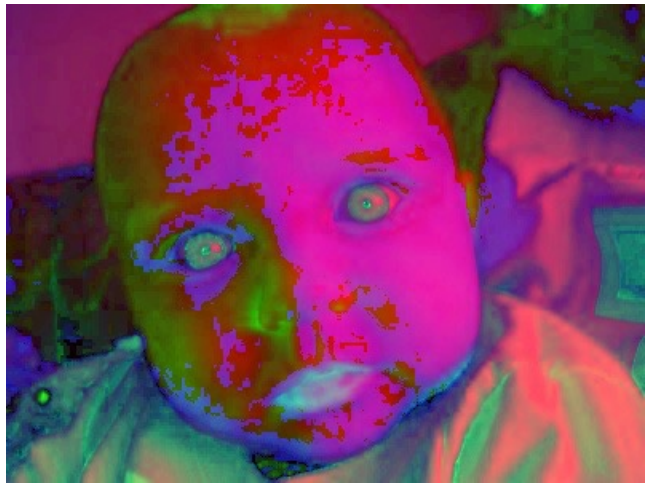
      - cv:Mat::**operator ()** – Extrae una submatriz
        - cv:Mat::**operator (**Range rowRange, Range colRange**)**
        - cv:Mat::**operator (**const Rect& roi**)** *Ejemplo: M(cv::Rect(0,0,3,3))*

Programación en OpenCV (C++)

- Alternativas a la clase cv::**Mat**
  - cv::**Matx** (template): crea pequeñas matrices con valores iniciales
    - *cv::Matx<double, 3, 3>(1.0, 0.0, 0.0,*
      0.0, 1.0, 0.0,
      0.0, 0.0, 1.0);
  - cv::**Mat_** (template): crea una matriz de tipo predefinido
    - Permite el acceso más sencillo a sus elementos mediante el operador **(f,c)**
    - Deriva de **cv::Mat** por lo que comparte todos los métodos y se puede forzar la conversión entre ambos tipos.
    - *cv::Mat_<double> M(3,3);* // crear una matriz de 3x3 de tipo double
    - *M(1,2) = 23.2;* // accede al elemento *f=1*, *c=2*
  - cv::**InputArray**, cv::**OutputArray**
    - Clases para paso de parámetros a las funciones (proxy).
    - Se puede construir a partir de:
      - cv::**Mat**, cv::**Mat_**<T>, cv::**Matx**<T, m, n>,
      - std::**vector**<T>, std::**vector**<std::**vector**<T> >
      - std::**vector**<cv::**Mat**>

Programación en OpenCV (C++)

- **Formatos de Imagen:**
  - **BGR** – formato por defecto de ***imread***(). 3 canales de color
  - **HSV** - Hue, Saturation, Value is lightness. 3 canales
  - **GRAYSCALE** – Nivel de gris. 1 canal



EJEMPLOS

http://umh1782.edu.umh.es/opencv/

EJEMPLO 1

Capturar y visualizar la imagen de una cámara

https://docs.opencv.org/4.5.5/dd/de7/group__videoio.html

https://docs.opencv.org/4.5.5/d5/dc4/tutorial_video_input_psnr_ssim.html

Captura de Imágenes de una Cámara: módulo highgui

- Primer ejemplo (**ej1.cpp**): capturar y visualizar la imagen de una cámara.
- Clase *cv::**VideoCapture***
  - Constructores:
    - cv::**VideoCapture**( ) → <VideoCapture object> : crea objeto
    - cv::**VideoCapture**(int **index,** int **apiPreference** = cv::CAP_ANY) → <VideoCapture object>
      abre el dispositivo (id de cámara)
    - cv::**VideoCapture**(const String & **filename**, int **apiPreference** = cv::CAP_ANY) →
      abre dispositivo (fichero video/sec - *str*)
  - Métodos:
    - bool cv::VideoCapture::**open** (int **device**) -> abre el dispositivo (id de cámara)
    - bool cv::VideoCapture::**open** (const string& **filename**) -> abre un fichero de video/sec
    - bool cv::VideoCapture::**isOpened** ()
    - bool cv::VideoCapture::**read** (cv::Mat& **image**) -> captura una imagen y la copia en una Matriz
    - double cv::VideoCapture::**get** (int **propId**) -> lee el valor de una propiedad de la cámara
    - bool cv::VideoCapture::**set** (int **propId**, double **value**)
      -> configura el valor de una propiedad de la cámara
    - bool cv::VideoCapture::**getExceptionMode**( )
    - void cv::VideoCapture::**setExceptionMode** (bool **enable**)
    - void cv::VideoCapture::**release**() -> Libera el dispositivo

Captura de Imágenes de una Cámara: módulo highgui

- Primer ejemplo (**ej1.cpp**): capturar y visualizar la imagen de una cámara.
- Ventana visualización:
  - void cv::**namedWindow** (const String& **winname**, int **flags**=cv::WINDOW_AUTOSIZE)

    flags: cv::WINDOW_AUTOSIZE, cv::WINDOW_NORMAL,
    cv::WINDOW_FREERATIO, cv::WINDOW_KEEPRATIO,
    cv::WINDOW_GUI_NORMAL, cv::WINDOW_GUI_EXPANDED

  - void cv::**imshow** (const String& **winname**, cv::InputArray **mat**)
  - void cv::**setWindowTitle**(const String & **winname**, const String & **title**)
    titulo de la Ventana (por defecto: winname)
  - void cv::**destroyAllWindows**()
  - void cv::**destroyWindow**(const String & **winname**)

- Eventos del teclado:
  - int cv::**waitKey**(int **delay**=0) espera en milisegundos
  - int cv::**pollKey**() comprueba si se ha pulsado un tecla desde la última llamada

Captura de Imágenes de una Cámara: módulo highgui

- Primer ejemplo (**ej1.cpp**): capturar y visualizar la imagen de una cámara

```
#include <opencv2/opencv.hpp>  // OpenCV library headers
using namespace std;
```

```
const char  * WINDOW_CAMERA1  = "(W1) Camera 1";      // windows id

int CAMERA_ID = 0;     //  default camera
int key;
cv::Size camSize;          // camera resolution
cv::VideoCapture camera;     // Cameras
cv::Mat capture;           // Images
```

```
camera.open(CAMERA_ID); //  open camera
if (!camera.isOpened())
{
        cout << "you need to connect a camera, sorry.\n";
        return -1;
}
camSize.width = (int) camera.get( cv::CAP_PROP_FRAME_WIDTH);
camSize.height = (int) camera.get( cv::CAP_PROP_FRAME_HEIGHT);

// Create the visualization windows
cv::namedWindow (WINDOW_CAMERA1, cv::WINDOW_AUTOSIZE);
```

Captura de Imágenes de una Cámara: módulo highgui

- Primer ejemplo (**ej1.cpp**): capturar y visualizar la imagen de una cámara

```
// while there are images ...
while (camera.read(capture))
{
        if(capture.empty())
                continue;    // capture has failed, continue

        // Put your image processing code here


         // Put your visualization code here
        cv::imshow(WINDOW_CAMERA1, capture);     // show image in a window

        // check keystroke to exit (image window must be on focus)
        key = cv::pollKey ();
        if (key == 'q' || key == 'Q' || key == 27 )
                break;
}
```

```
// free windows and camera resources
cv::destroyAllWindows();
if (camera.isOpened()) camera.release();
```

EJERCICIO 1b

Filtrado de Imágenes: Convolución (e1b.cpp)

- Incorporar al ejemplo previo el procesamiento de la imagen capturada y visualizar el resultado

https://docs.opencv.org/4.5.5/d4/d86/group__imgproc__filter.html

https://docs.opencv.org/4.5.5/d4/dbd/tutorial_filter_2d.html

Filtrado de Imágenes: módulo imgproc

- Ejemplo (**ej1b.cpp**): filtrado de imágenes (máscaras de convolución)
- Filtrado 2D (convolución):
  - Métodos:
    - void cv::**filter2D**(cv::InputArray **src**, cv::OutputArray **dst**, int **ddepth**, cv::InputArray **kernel**,
      cv::Point **anchor** = cv::Point(-1,-1), double **delta**=0,
      int **borderType** = cv::BORDER_DEFAULT)

      - ddepth : **-1** (misma profundidad que src)
      - anchor : **centro del kernel** cv::Point(-1,-1) -> centro de la máscara
      - delta: valor añadido al resultado
      - borderType: tipo de extrapolación en los bordes
      - cv::BORDER_TRANSPARENT, cv::BORDER_WRAP, cv::**BORDER_REFLECT_101**, cv::BORDER_REPLICATE, cv::BORDER_CONSTANT

- Conversión de Color:
  - void cv::**cvtColor** (InputArray **src,** OutputArray **dst**, int code, int **dstCn=0**)
    - Códigos: cv::COLOR_BGR2GRAY, cv::COLOR_ GRAY2BGR
    - Códigos: cv::COLOR_ BGR2HSV, cv::COLOR_ HSV2BGR ……

Filtrado de Imágenes: módulo imgproc

- Ejemplo (**ej1b.cpp**): filtrado de imágenes (máscaras de convolución)

```
cv::Mat kernel(cv::Matx<double, 3, 3>(1.0, -2.0, 1.0,
                                      2.0, -4.0, 2.0,
                                      1.0, -2.0, 1.0) );

cout << "Kernel: " << endl << kernel << endl;


……

// image processing code section

// Transform to gray level
cv::cvtColor( capture, gray_image, cv::COLOR_BGR2GRAY );    // transforms to gray level

// Apply filter to image
cv::filter2D (gray_image, filtered_image, -1 , kernel, cv::Point( -1, -1 ), 0, cv::BORDER_DEFAULT );
```

EJEMPLO 2

Leer, Procesar y Guardar imágenes en un fichero

https://docs.opencv.org/4.5.5/d4/da8/group__imgcodecs.html

https://docs.opencv.org/4.5.5/dd/d1a/group__imgproc__feature.html

https://docs.opencv.org/4.5.5/db/deb/tutorial_display_image.html

Lectura/Escritura Imágenes: módulos highgui - imgproc

- Segundo ejemplo (**ej2.cpp**): lee y procesa una imagen de un fichero

- Leer/Escribir un fichero de imagen:
  - Métodos:
    - Mat cv::**imread** (const string& **filename**, int **flags**=cv::IMREAD_COLOR)
      - **flags** →. cv::IMREAD_COLOR, cv::IMREAD_GRAYSCALE, …

    - bool cv::**imwrite** (const string& **filename**, cv::InputArray **img**,
      const vector<int>& **params**=vector<int>())

- Conversión de Color:
  - void cv::**cvtColor** (InputArray **src,** OutputArray **dst**, int code, int **dstCn=0**)
    - Códigos: cv::COLOR_BGR2GRAY, cv::COLOR_ GRAY2BGR
    - Códigos: cv::COLOR_ BGR2HSV, cv::COLOR_ HSV2BGR ……

- Procesamiento:
  - void cv::**Canny** (cv::InputArray **image**, cv::OutputArray **edges**, double **threshold1**,
    double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false)
    - Detector de Bordes

Lectura/Escritura Imágenes: módulos highgui - imgproc

- Segundo ejemplo (**ej2.cpp**): lee y procesa una imagen de un fichero

```
const char  * WINDOW_IMAGE  = "(W1) Image 1";     // window id
const char  * WINDOW_BORDERS  = "(W2) Canny Borders";      // window id

cv::Mat image;              // Images
cv::Mat gray_image;
cv::Mat borders_image;

// Create the visualization windows
cv::namedWindow (WINDOW_IMAGE,   cv::WINDOW_AUTOSIZE);
cv::namedWindow (WINDOW_BORDERS,   cv::WINDOW_AUTOSIZE);
```

```
image = cv::imread( "building.jpg" );
if (image.empty())
{
        cout << "you need to select an image, sorry.\n";
        return -1;
}

cv::cvtColor( image, gray_image, cv::COLOR_BGR2GRAY );      // transforms to gray level
cv::Canny( gray_image, borders_image, 80, 150 );     // Canny border detector
cv::imwrite( "result.jpg", borders_image ); // store result image

cv::imshow( WINDOW_IMAGE, image);  // show image in a window
cv::imshow( WINDOW_BORDERS, borders_image); // show image in a window
```

EJERCICIO 2b

- Leer imágenes de un fichero de video
  - Clase: cv::**VideoCapture**
- Detectar bordes
  - Método: cv::**Canny**
- Guardar el resultado en un fichero de video
  - Clase: cv::**VideoWriter**

https://docs.opencv.org/4.5.5/dd/de7/group__videoio.html

https://docs.opencv.org/4.5.5/d7/d9e/tutorial_video_write.html

Captura/Escritura Fichero de Video: módulo video I/O

- Ejercicio (**ej2b.cpp**): Leer, procesar y almacenar desde un video
- Clase cv::**VideoCapture**
  - Constructores:
    - cv::**VideoCapture**( ) → <VideoCapture object> : crea objeto
    - cv::**VideoCapture**(const String & **filename**) → <VideoCapture object>
      abre dispositivo (fichero video/sec - *str*)
- Clase cv::**VideoWriter**
  - Constructores:
    - cv.**VideoWriter**( ) → <VideoWriter object> : crea objeto
    - cv.**VideoWriter**(const String & **filename**, int **fourcc**, double **fps,** cv::Size **frameSize** ,
      bool **isColor**=true) → < VideoWriter object> : abre dispositivo (fichero video/sec - *str*)
      fourcc: cv::VideoWriter::**fourcc**('D', 'I', 'V', 'X')
      frameSize: cv::Size(width, height)
  - Métodos:
    - bool cv.VideoWriter::**open**(**filename, fourcc, fps, frameSize** [, **isColor**]) abre fichero video
    - bool cv.VideoWriter::**isOpened**( )
    - int cv::VideoWriter::**fourcc**(char c1, char c2, char c3, char c4)
    - cv::VideoWriter::**write**(InputArray **image**) : guarda una imagen
    - cv::VideoWriter::**release**( ) : Libera el dispositivo

Captura/Escritura Fichero de Video: módulo video I/O

- Ejercicio (**ej2b.cpp**): lee y procesa una imagen de un fichero de video

```
const char  *videoFilef = "video.mp4";       //  default video file
cv::VideoCapture inputVideo;  // input video object
cv::VideoWriter outputVideo;   // output video object
v::Size camSize;  // camera resolution

# Open input video object
inputVideo.open(videoFile);
if (!inputVideo.isOpened())
{       cout << "you need to select a video file, sorry.\n";
        return -1;
}

# Getting video resolution / FPS
camSize.width = (int) inputVideo.get(cv::CAP_PROP_FRAME_WIDTH);
camSize.height = (int) inputVideo.get(cv::CAP_PROP_FRAME_HEIGHT);
fps = (double) inputVideo.get(cv::CAP_PROP_FPS);

# Open output video object
outputVideo.open("result.avi", cv::VideoWriter::fourcc('D','I','V','X'), fps, camSize, false);
if (!outputVideo.isOpened())
{       cout << "\nI cannot open the video output file, sorry.\n";
        return -1;
}
```

Captura/Escritura Fichero de Video: módulo video I/O

- Ejercicio (**ej2b.cpp**): lee y procesa una imagen de un fichero de video

```
while (inputVideo.read(capture))
{
        if(capture.empty())
                continue;    // capture has failed, continue

        // transforms to gray level
        cv::cvtColor( capture, gray_image, cv::COLOR_BGR2GRAY );

        cv::Canny(gray_image, borders_image, 80, 150 );      // Canny border detector

        outputVideo.write(borders_image);          // writes image to video file

        cv::imshow(WINDOW_CAMERA1, borders_image);   // show image in a window

        key = cv::pollKey ();
        if (key == 'q' || key == 'Q' || key == 27)
                        break;
}
```

```
cv::destroyAllWindows()
if (inputVideo.isOpened())      inputVideo.release()
if (outputVideo.isOpened())     outputVideo.release()
```

EJEMPLO 3

Gestión de Eventos

- Manejador eventos Ratón
- Guardar imagen ventana al pulsar SHIT+LeftClick

https://docs.opencv.org/4.5.5/d7/dfc/group__highgui.html

Gestión de Eventos: módulo highgui

- Ejemplo (**ej3.cpp**):
  - Partimos del código de captura de imágenes **ej1.cpp**
- Manejador de Ratón:
  - Declararlo:
    - void **MouseHandler** (int **event**, int **x**, int **y**, int **flags**, void* **param**);
  - Asignar Manejador:
    - cv::**setMouseCallback**(WINDOW_CAMERA1, **MouseHandler**, NULL);
- Implementación Manejador:

```
// Mouse events handler  for image window
// event:        event type sent to the handler ->  cv::EVENT_MOUSEMOVE,
//          cv::EVENT_LBUTTONDOWN, cv::EVENT_LBUTTONUP, cv::EVENT_LBUTTONDBLCLK,
//          cv::EVENT_RBUTTONDOWN, cv::EVENT_RBUTTONUP, cv::EEVENT_RBUTTONDBLCLK,
//          cv::EVENT_MBUTTONDOWN, cv::EVENT_MBUTTONUP, cv::EEVENT_MBUTTONDBLCLK,
//          cv::EVENT_MOUSEWHEEL, cv::EVENT_MOUSEHWHEEL
// x:     X-coordinate position of the mouse in window
// y:     Y-coordinate position of the mouse in window
// flags: aditional flags sent to the handler ->
//          cv::EVENT_FLAG_SHIFTKEY, cv::EVENT_FLAG_CTRLKEY, cv::EVENT_FLAG_ALTKEY,
//          cv::EVENT_FLAG_LBUTTON, cv::EVENT_FLAG_RBUTTON, cv::EVENT_FLAG_MBUTTON,
// param: set in cv::SetMouseCallback
//-----------------------------------------------------------------------
```

Gestión de Eventos: módulo highgui

- Ejemplo (**ej3.cpp**):

```
const char  * WINDOW_CAMERA1  = "(W1) Camera 1";      // windows id
const int KEY_F5 = 7602176;
int CAMERA_ID = 0;      //  default camera
unsigned int id = 1;       // id for stored images
cv::Mat capture;           // Images
```

```
// Mouse Handler
cv::setMouseCallback( WINDOW_CAMERA1, MouseHandler, NULL );
```

```
void MouseHandler( int event, int x, int y, int flags, void* param)
{
    cout << "Event: " << event << ", x:" << x << ", y:"<< y << ", flags:" << flags << endl;

    // on click left mouse button and SHIFT key, saves image
    if(event==cv::EVENT_LBUTTONDOWN && (flags & cv::EVENT_FLAG_SHIFTKEY) )
    {
          ostringstream filename;
           filename << "Image" << id << ".jpg";
           cout << "Saving image window in file: " << filename.str() << endl;
            cv:: imwrite( filename.str(), capture);     // saves window image
            id++;
    }
}
```

Gestión de Eventos: módulo highgui

- Ejemplo (**ej3.cpp**):

```
// Main loop
while (camera.read(capture))
{
        if(capture.empty())
                continue;    // capture has failed, continue

        // Put your image processing code here

         // Put your visualization code here
         cv::imshow(WINDOW_CAMERA1, capture);    // show image in a window

        // Checks for a keystroke to exit (image window must be on focus)
        key = cv::pollKey (10);

        if(key == KEY_F5)
        {
          ostringstream filename;
           filename << "Image" << id << ".jpg";
           cout << "Saving image window in file: " << filename.str() << endl;
            cv:: imwrite( filename.str(), capture);    // saves window image
            id++;
        }
        else if (key == 'q' || key == 'Q' || key == 27)
                        break;
}
```

EJERCICIO 3b

Interfaz de usuario. (e3b.py)

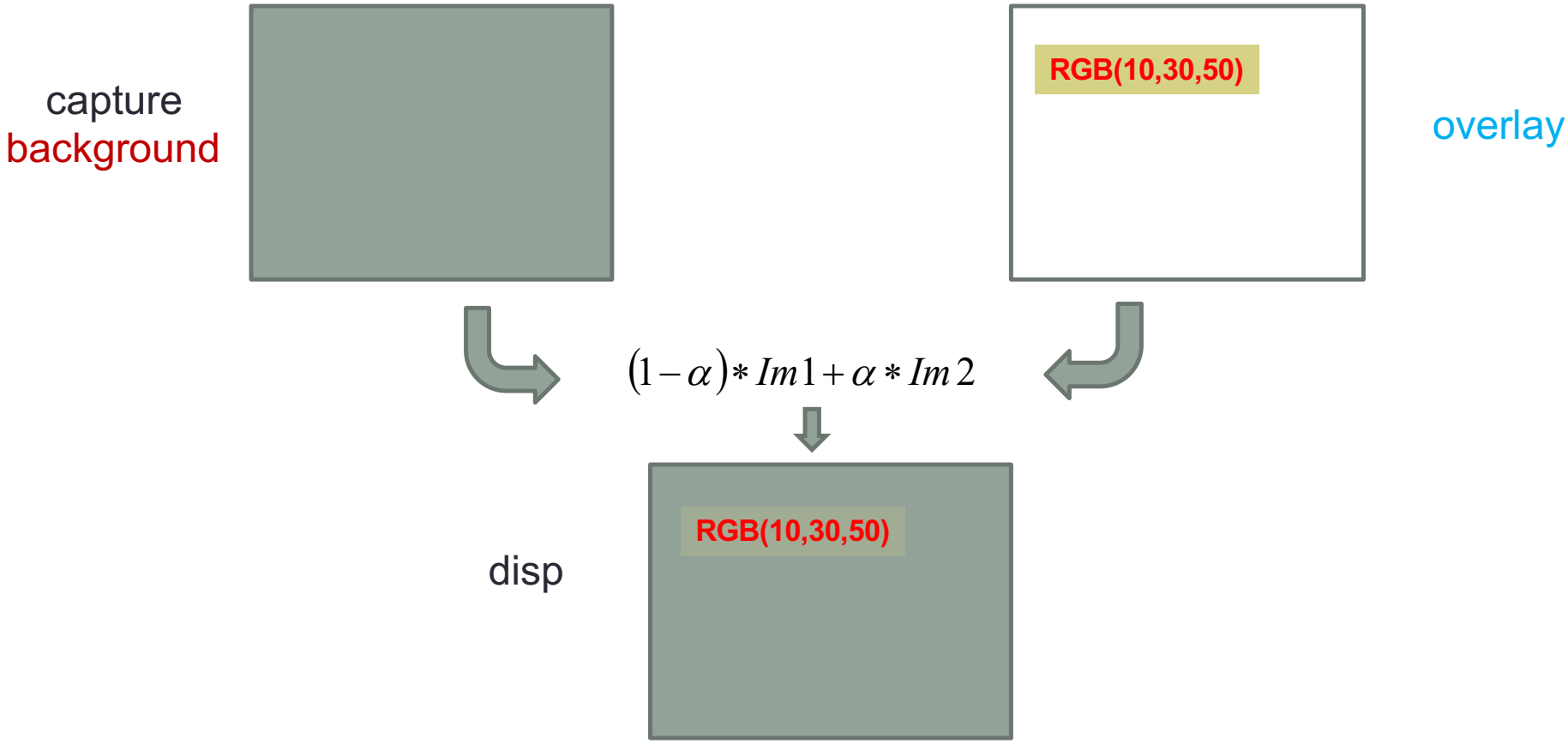
- Visualización del color en cada pixel bajo el cursor, mostrándolo en overlay semi-transparente sobre la imagen
- TrackBars: modificar parámetros de ejecución
- Botón de salida en overlay

https://docs.opencv.org/4.5.5/d7/dfc/group__highgui.html

https://docs.opencv.org/4.5.5/d6/d6e/group__imgproc__draw.html

Interfaz Usuario: módulo core

- Ejercicio:
  - Visualización del color en cada pixel bajo el cursor, mostrándolo en overlay semi-transparente sobre la imagen



cv::**addWeighted(src1**, **alpha, src2**, **beta**, **gamma** , **dst**, **dtype**=-1 **)**

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

Interfaz Usuario: módulo core

- Constructores adicionales clase **Mat**:
  - cv::**Mat** overlay = cv::**Mat**(camSize, **CV_8UC3**, cv::Scalar::**all**(0));

- Métodos adicionales clase **Mat**:
  - overlay.**setTo**(cv::Scalar::**all**(0)); // borrar una imagen

- Acceder al valor de un pixel clase **Mat**:
  - Mediante Puntero : *uchar ***ptr** = capture.**data**;*
  - Método **at**: *capture.**at**<cv::Vec3b>(fil, col)* (imagen color BGR)
    *capture.**at**<uchar>(fil, col)* (imagen escala gris)
    (Plantilla, debemos especificar el tipo para el pixel)

- Generar un string formateado en C++ (*equivalente a sprintf en C*): clase ***ostringstream***

```
ostringstream cursorColor ;
cv::Vec3b color = capture.at< cv::Vec3b>(y, x);
cursorColor << "RGB(" << (int)color[2] << ","
                                << (int)color[1] << "," << (int)color[0] << ")";
cout << cursorColor.str() << endl;    // convert to string and show it  in console
```

Interfaz Usuario: módulos highgui/core

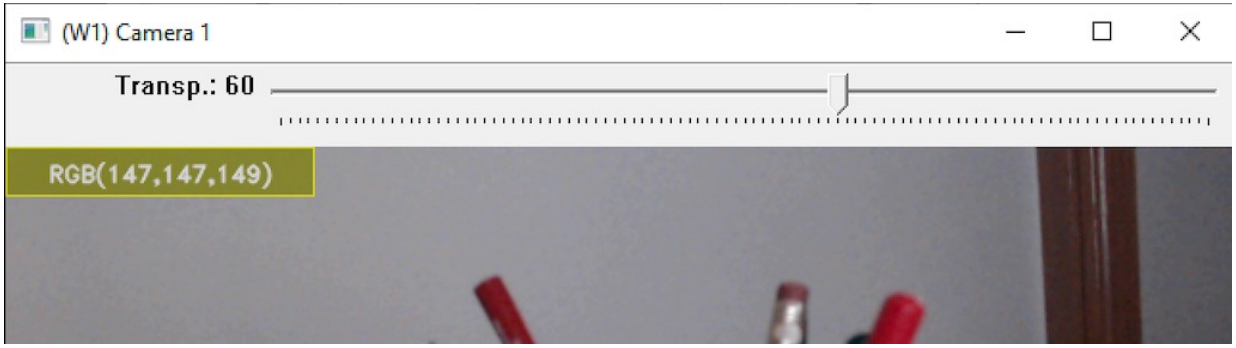
- Trackbars:
  - int cv::**createTrackbar** (const string& **trackbarname**, const string& **winname**, int* **value**, int **count**, TrackbarCallback **onChange=0**, void* **userdata=0**)

    value: puntero a la variable que queda sincronizada con la posición del Trackbar

    count: valor máximodel trackbar

```
cv::createTrackbar ("Transp.", WINDOW_CAMERA1,  &ALPHA,  100,  NULL, (void *)0 );
```

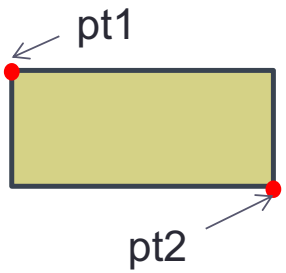
  - cv::**setTrackbarMax**(const string& **trackbarname**, const string& **winname**, int **maxval**)
  - cv::**setTrackbarMin**(const string& **trackbarname**, const string& **winname**, int **minval**)
  - cv::**setTrackbarPos**(const string& **trackbarname**, const string& **winname**, int **pos**)
  - int cv::**getTrackbarPos**(const string& **trackbarname**, const string& **winname**)



Interfaz Usuario: módulos highgui/core

- Dibujo en pantalla:
  - void cv::**line** (cv::Mat& **img**, cv::Point **pt1**, Point **pt2**, const cv::Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)
  - void cv::**rectangle** (cv::Mat& **img**, cv::Point **pt1**, Point **pt2**, const cv::Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)
  - void cv::**circle** (cv::Mat& **img**, cv::Point **center**, int **radius**, const cv::Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

  - Tipo de líneas: cv::**LINE_8**-connected line, cv::**LINE_4**-connected line, cv::**LINE_AA** - antialiased line
  - Grosor: en objetos con área podemos especificar: cv::**FILLED**




```
cv::Size BUTTON_SIZE (160,25);// overlay button Size (width,heigth)
cv::Point BUTTON_POS (0,0);     // overlay button upper left corner position (x,y)

cv::rectangle (overlay, BUTTON_POS, BUTTON_POS + cv::Point(BUTTON_SIZE),
                        cv::Scalar(0,120,120),  cv::FILLED);
```

Interfaz Usuario: módulos highgui/core

- Texto:
  - Size cv::**getTextSize** (const string& **text**, int **fontFace**, double **fontScale**, int **thickness**, int* **baseLine**)
  - void cv::**putText** (cv::Mat& **img**, const string& **text**, cv::Point **org**, int **fontFace**, double **fontScale**, cv::Scalar **color**, int **thickness**=1, int **lineType**=LINE_8, bool **bottomLeftOrigin**=false)

Fuentes: cv::FONT_HERSHEY_SIMPLEX, cv::FONT_HERSHEY_PLAIN, cv::FONT_HERSHEY_DUPLEX, cv::FONT_HERSHEY_COMPLEX, cv.FONT_HERSHEY_COMPLEX_SMALL, cv::FONT_HERSHEY_SCRIPT_SIMPLEX, cv::FONT_HERSHEY_SCRIPT_COMPLEX

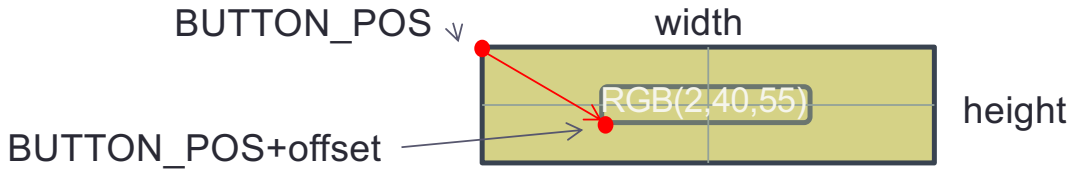
- Se pueden combinar con: | CV_FONT_ITALIC

```
cv::putText (overlay,  cursorColor.str(),  BUTTON_POS + offset ,
                        cv::FONT_HERSHEY_DUPLEX, 0.4,
                        cv::Scalar(255,255,255), 1,  CV_AA );
```

```
// offset para centrado de texto en un recuadro
cv::Size textSize;  int baseline;

textSize = cv::getTextSize(cursorColor.str(), cv::FONT_HERSHEY_DUPLEX, 0.4, 1, &baseline);

cv::Point offset = cv::Point(BUTTON_SIZE.width/2, BUTTON_SIZE.height/2) +
                   cv::Point(-textSize.width/2, textSize.height/2 + baseline/2);
```



Interfaz Usuario: módulo core

- MouseHandler:

```
void MouseHandler (int event, int x, int y, int flags, void* param)
{
        // on moving the cursor over the image
        if( event == cv::EVENT_MOUSEMOVE )
        {
                CURSOR_POS.x = x;   // save new cursor position in global variable cursorPos
                CURSOR_POS.y = y;
        }

        // on click left mouse button
        if( event==cv::EVENT_LBUTTONDOWN )
        {
                // checks if Exit button is clicked
                if( x > BUTTON_POS.x  && x < (BUTTON_POS.x + BUTTON_SIZE .width) &&
                        y > BUTTON_POS.y  && y < (BUTTON_POS.y + BUTTON_SIZE .height) )
                        EXIT = true;

        }
}
```

Interfaz Usuario: módulo core

- Inicialización:
  - Global:

```
void DrawOverlay(cv::Mat &background, cv::Mat &queryIM, double alpha);

// Variables Globales
int CAMERA_ID = 0;              //  default camera
int ALPHA = 40;            // % level of transparency
bool EXIT = false;// exit the program

cv::Point CURSOR_POS;       // current position of the cursor over the window
cv::Size BUTTON_SIZE (160,25);    // Overlay button Size (width,heigth)
cv::Point BUTTON_POS (0,0);         // overlay button uper left corner position (x,y)
```

  - Main:

```
cv::Mat capture, disp;              // Images

// Creates the visualization windows
cv::namedWindow (WINDOW_CAMERA1, cv::WINDOW_AUTOSIZE);

// enables a trackbar associated to variable ALPHA
cv::createTrackbar("Transp.", WINDOW_CAMERA1, &ALPHA, 100, NULL, (void *)0);

//       Mouse Handler
cv::setMouseCallback( WINDOW_CAMERA1, MouseHandler, NULL );
```

Interfaz Usuario: módulo core

- Función DrawOverlay:

```
void DrawOverlay( Mat &background, Mat &queryIM, double alpha)
{
    cv::Size imSize = background.size();

    //Allocates memory for overlay image of the same size as background
    cv::Mat overlay = cv::Mat( imSize, CV_8UC3, cv::Scalar::all(0));

    // if background is not BGR converts it to BGR
    if(background.channels()==1)
                cv::cvtColor(background, background, CV_GRAY2BGR);



    ……..................................


}
```

Interfaz Usuario: módulo core

- Función DrawOverlay:

```
void DrawOverlay( Mat &background, Mat &queryIM, double alpha)
{
    ……................................

   // Draws the overlay image
  cv::rectangle(overlay, BUTTON_POS, BUTTON_POS+ cv::Point(BUTTON_SIZE), cv::Scalar(0,120,120),
                                                                                    cv::FILLED);
  cv::rectangle(overlay, BUTTON_POS, BUTTON_POS+ cv::Point(BUTTON_SIZE), cv::Scalar(0,255,255), 1);

  ostringstream cursorColor;
   if(queryIM.channels() == 3) {
         cv::Vec3b color = queryIM.at< cv::Vec3b>(CURSOR_POS.y, CURSOR_POS.x);
        cursorColor << "RGB(" << (int)color[2] << "," << (int)color[1] << "," << (int)color[0] << ")";
   }
   else if(queryIM.channels() == 1) {
        uchar color = queryIM.at<uchar>(CURSOR_POS.y, CURSOR_POS.x);
        cursorColor << "Gray(" << (int)color << ")";
   }

     ……..............................................

}
```

Interfaz Usuario: módulo core

- Función DrawOverlay:

```
void DrawOverlay( Mat &background, Mat &queryIM, double alpha)
{
    ……................................

  // offset para centrado de texto en un recuadro
  cv::Size textSize;  int baseline;
  textSize = cv::getTextSize( cursorColor.str(), cv::FONT_HERSHEY_DUPLEX, 0.4, 1, &baseline);

  cv::Point offset = cv::Point(BUTTON_SIZE.width/2, BUTTON_SIZE.height/2) +
                     cv::Point(-textSize.width/2, textSize.height/2 + baseline/2);

  cv::putText (overlay,  cursorColor.str(), BUTTON_POS+offset, cv::FONT_HERSHEY_DUPLEX, 0.4,
                         cv::Scalar(255,255,255), 1,  CV_AA );


  // blends both images
  for(int i=0; i< imSize.height; i++)
      for(int j=0; j< imSize.width; j++)
          if (overlay.at<Vec3b>(i, j) != cv::Vec3b(0,0,0) )
              background.at< cv::Vec3b>(i, j) =  background.at< cv::Vec3b>(i, j) * (1-alpha)  +
                                                 overlay.at< cv::Vec3b>(i, j) * alpha;

}
```

Interfaz Usuario: módulo core

- Bucle Principal:

```
while (camera.read(capture))
{
            if(capture.empty())
                continue;// capture has failed, continue

           // copy capture to disp image and convert to BGR if necessary
           if(capture.channels()==3)
                capture.copyTo(disp);
           else if(capture.channels()==1)
                 cv::cvtColor(capture, disp, cv::COLOR_GRAY2BGR);

           // draws overlay with pixel color under cursor on capture image
           DrawOverlay(disp, capture, (double)ALPHA/100);


          cv::imshow(WINDOW_CAMERA1, disp);        // show image in a window

           // Checks for a keystroke to exit (image window must be on focus)
           key = cv::pollKey ();
           if (key == 'q' || key == 'Q' || key == 27 || EXIT)
                break;
}
```

EJERCICIO 3c

Captura dos cámaras (ej3c.cpp)

- Captura de imágenes de dos cámaras (estéreo)
- Uso de regiones de interés (Mat headers)

Mat headers

```
int CAMERA_ID[2] = {0, 1};     //  camera ids
cv::Mat draw_image;            // Mat for drawing both Cameras
cv::Mat capture[2];            // Mat headers for both cameras
cv::Size camSize(640, 480);            // capture resolution

// Creating (assigning memory)  Mat image for drawing both images
draw_image.create(camSize.height, camSize.width*2, CV_8UC3);

// Create window headers (shared memory) for both cameras over draw_image
capture[0] = draw_image.colRange(0, camSize.width);              // window camera 1 (header)
capture[1] = draw_image.colRange(camSize.width, camSize.width*2);  // window camera 2 (header)

for(int i=0; i<2; i++)
      camera[i].open(CAMERA_ID[i]); //  open  camera
```

```
while (true)
{
        for(int i=0; i<2; i++)
             camera[i].read(capture[i]);         // read camera frames

        cv::imshow("(W1) Cameras", draw_image);       // show draw image in a window

        // Checks for a keystroke to exit (image window must be on focus)
        key = cv::pollKey();

         cv::imshow(WINDOW_CAMERA1, disp);        // show image in a window
}
```

PERSISTENCIA

Ficheros XML/JSON/YAML

https://docs.opencv.org/4.x/d4/da4/group__core__xml.html

Persistencia: XML/JSON/YAML, módulo core

- Lectura/escritura de datos en ficheros XML/JSON/YAML:
  - Permite guardar y leer las estructuras básicas de OpenCV: Mat, Point, ….
  - Permite almacenar/leer vectores (clase vector) de forma transparente hasta el primer nivel de jerarquía.
  - Gestión manual para vectores de vectores, estructuras …
    - **{: } mappings (estructuras) [:] sequences (vectores)**

- Clase: cv::***FileStorage***
  - cv::FileStorage::**FileStorage**()
  - cv::FileStorage::**FileStorage**(const string& **source**, int **flags**, const string& **encoding**=string())
    - **Source**: extensión .xml .yml .yaml .json
    - **Flags**: cv::*FileStorage::READ*, cv::*FileStorage::WRITE*, cv::*FileStorage::APPEND*
- Métodos básicos:
  - cv::FileStorage::**open**(const string& **filename**, int **flags**, const string& **encoding**=string())
  - bool cv::FileStorage::**isOpened**()
  - void cv::FileStorage::**release**()
  - **Escribir: operator <<**

Nombre del elemento

Valor del elemento

```
fs << "cameraMatrix" <<  matriz;
```

  - **Leer: operator[] operator >>**

```
fs [" cameraMatrix " ] >> matriz;
```

Persistencia: XML/JSON/ YAML, módulo core

- Ejemplo:
  - Escritura:

```
cv::Mat cameraMatrix(cv::Matx<double, 3, 3>(1000, 0, 320, 0, 1000, 240, 0, 0, 1) );

cv::FileStorage fs("test.yml", cv::FileStorage::WRITE);
fs << "cameraMatrix" << cameraMatrix;
fs.release();
```

```
%YAML:1.0
cameraMatrix : !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 1000., 0., 320., 0., 1000., 240., 0., 0., 1. ]
```

  - Lectura:

```
cv::FileStorage fs2("test.yml", cv::FileStorage::READ);
cv::Mat matrix;
fs2 ["cameraMatrix"]  >> matrix;
cout << "cameraMatrix: " << matrix << endl;
fs2.release();
```

Persistencia: XML/JSON/ YAML, módulo core

- **Escritura** Mappings/Sequences
  - {: } mappings (estructuras) [:] sequences (vectores)

Ejemplo:

```
cv::FileStorage fs("test.yml", cv::FileStorage::WRITE);

fs << "features";
fs << "{:" << "x" << 167 << "y" << 49;
fs << "lbp" << "[:";

uchar lbp = rand() % 256;
for( int j = 0; j < 8; j++ )
    fs << ((lbp >> j) & 1);
fs << "] ";
fs << "} ";
fs.release();
```

```
%YAML:1.0
features: { x:167, y:49, lbp:[ 1, 0, 0, 1, 1, 0, 1, 1 ] }
```

Persistencia: XML/JSON/ YAML, módulo core

- **Lectura** Mappings/Sequences
  - {: } mappings (estructuras) [:] sequences (vectores)

Ejemplo:

```
cv::FileStorage fs2("test.yml", cv::FileStorage::READ);

cv::FileNode features = fs2["features"];

int x,y;
features["x"] >> x; features["y"]  >> y;
cout << "x=" << x<< ", y=" << y;

vector<uchar> lbpval;
features["lbp"] >> lbpval;

cout << ", lbp: (";
for( int i = 0; i < (int) lbpval.size(); i++ )
      cout << " " << (int) lbpval[i];
cout << ")" << endl;

fs2.release();
```