Visión por Computador (1782)

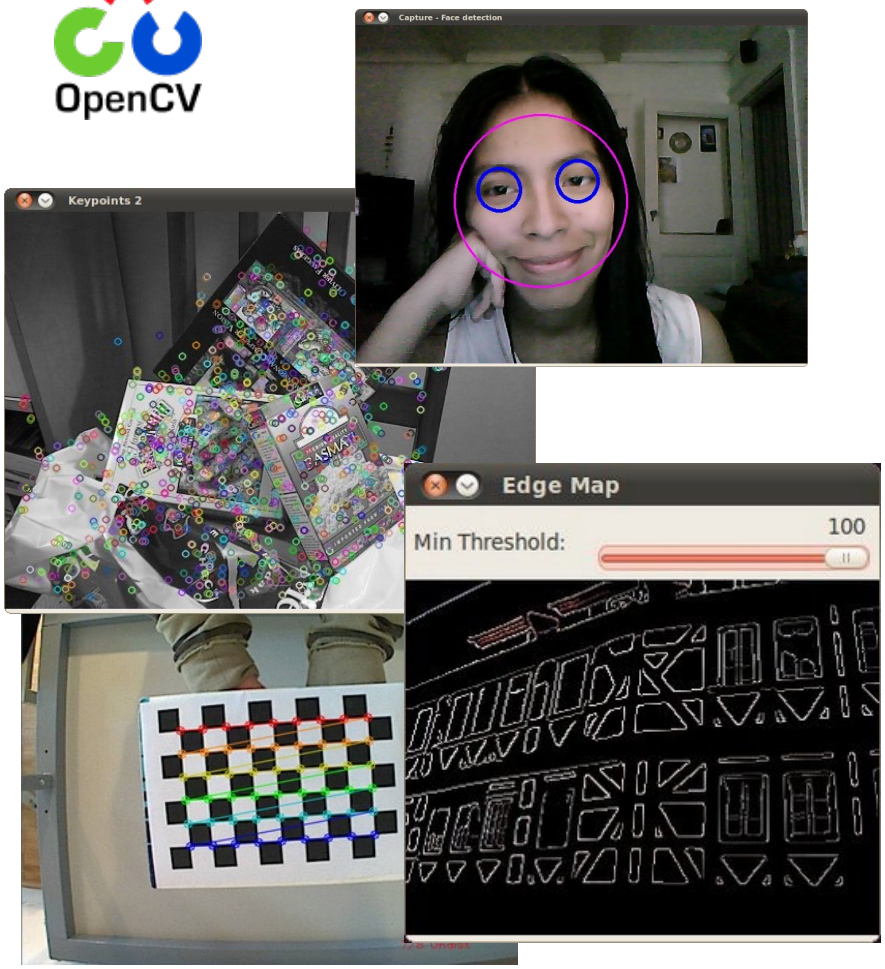
Herramientas de programación de aplicaciones
OpenCV2

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 XP/7/8/10
  - Compiladores:
    - **Microsoft Visual Studio 2010** -> **VC10** (funciona con Win XP/7/8/10)
    - Microsoft Visual Studio 2012 -> VC11 (Windows 7/8/10)
    - Microsoft Visual Studio 2013 -> VC12 (Windows 7/8/10)
    - 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 2.4:
    - Descargaremos la versión compilada estable: 2.4.11 (VC10) 2.4.13 (VC14)
    - Basta descomprimir el fichero
- Linux/Mac
  - Bajar los fuentes
  - Utilizar ***CMake*** para compilar la librería y las aplicaciones
  - Tutoriales:
    - http://umh1782.edu.umh.es/opencv2/

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***
    - Se utiliza para listas de puntos característicos

using namespace std;

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++)

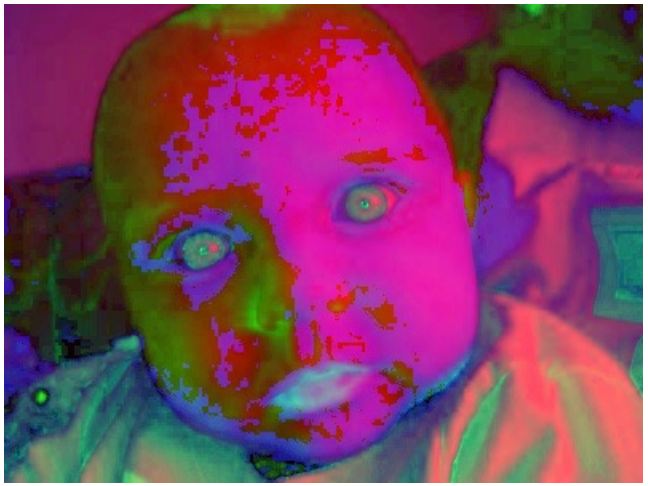
- Estructuras/Clases básicas:
  - cv::**Mat**
    - Métodos Estáticos (no precisan declarar un objeto):
      - cv::Mat::***ones****(row, col, type)* – devuelve una matriz de unos
      - cv::Mat::***zeros****(row, col, type)* – devuelve una matriz de ceros
      - cv::Mat::***eye****(row, col, type)* – 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::**vecto**r<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/opencv2/

Captura de Imágenes de una Cámara: módulo highgui

- Primer ejemplo (**ej1.cpp**): capturar y visualizar la imagen de una cámara. https://docs.opencv.org/2.4/modules/highgui/doc/highgui.html
- Clase *cv::**VideoCapture***
  - Métodos:
    - bool VideoCapture::**open** (int **device**) -> abre el dispositivo (id de cámara)
    - bool VideoCapture::**open** (const string& **filename**) -> abre un fichero de video/sec. imágenes
    - bool VideoCapture::**isOpened** ()
    - bool VideoCapture::**read** (cv::Mat& **image**) -> captura una imagen y la copia en una Matriz
    - double VideoCapture::**get** (int **propId**) -> lee el valor de una propiedad de la cámara
    - bool VideoCapture::**set** (int **propId**, double **value**)
      -> configura el valor de una propiedad de la cámara
    - void VideoCapture::**release**() -> Libera el dispositivo
- Ventana visualización:
  - void cv::**namedWindow**(const string& **winname**, int **flags**=cv::WINDOW_AUTOSIZE)
  - void cv::**imshow**(const string& **winname**, cv::InputArray **mat**)
  - void cv::**destroyAllWindows**()
- Eventos del teclado:
  - int cv::**waitKey**(int **delay**=0) espera en milisegundos

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

        // wait 10ms for a keystroke to exit (image window must be on focus)
        key = cv::waitKey (10);
        if (key == 'q' || key == 'Q' || key == 27 )
                break;
}
```

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

EJEMPLO 2

Leer, Procesar y Guardar imágenes en un fichero

http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html

https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.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**=1)

    - 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_BGR2GRAY, CV_GRAY2BGR
    - Códigos: CV_BGR2HSV, CV_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_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

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

http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html

MANEJADOR DE EVENTOS

http://http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html

Gestión de Eventos: módulo highgui

- Ejemplo (**ej1b.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,
// x: X-coordinate position of the mouse in window
// y:Y-coordinate position of the mouse in window
// flags: adtional flags sent to the handler ->
//         CV_EVENT_FLAG_SHIFTKEY, CV_EVENT_FLAG_CTRLKEY, CV_EVENT_FLAG_ALTKEY
// param: set in setMouseCallback
//---------------------------------------------------------------
```

Gestión de Eventos: módulo highgui

- Ejemplo (**ej1b.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 (**ej1b.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

        // wait 10ms for a keystroke to exit (image window must be on focus)
        key = cv::waitKey (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;
}
```

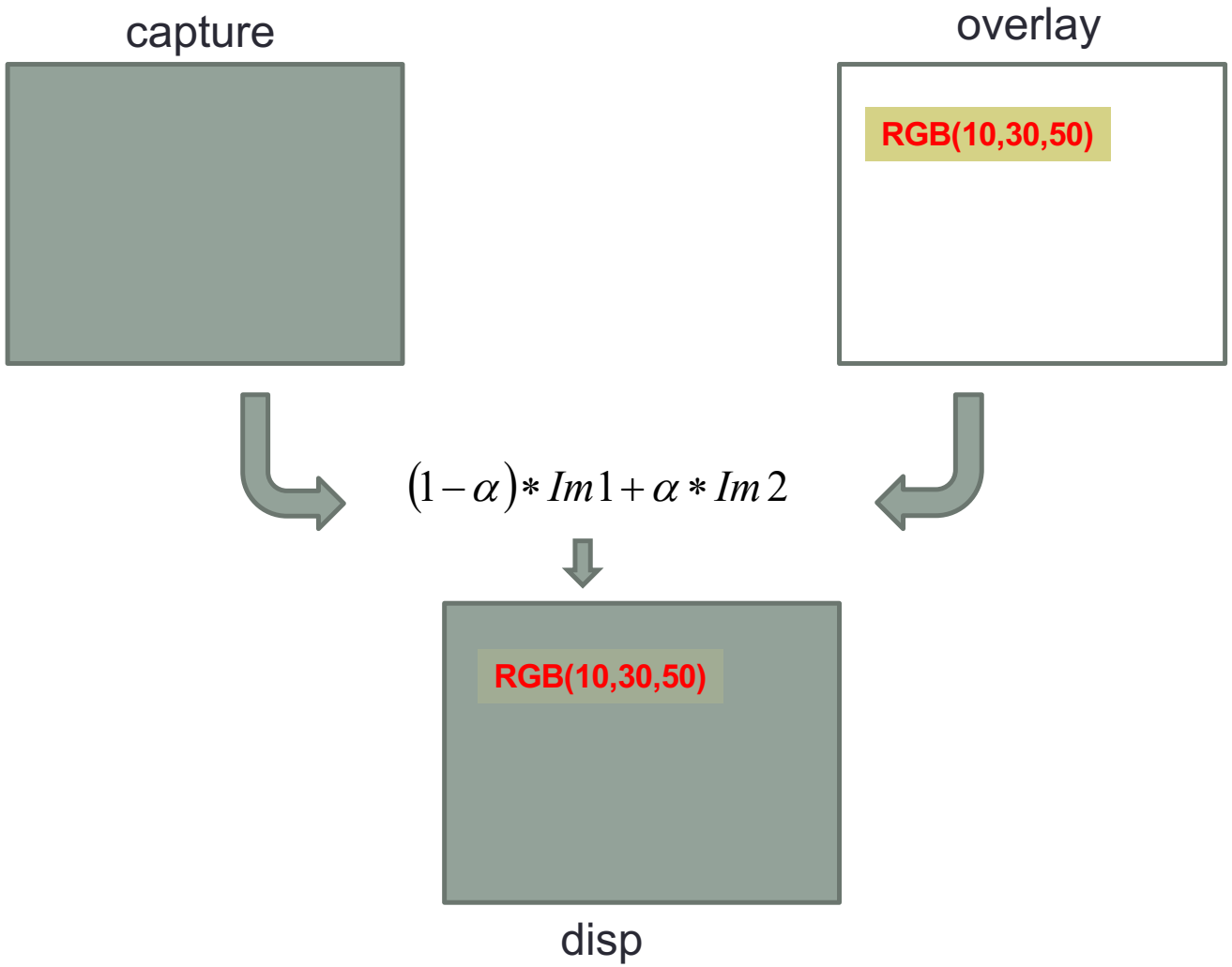
EJERCICIO

Interfaz de usuario

https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

Interfaz Usuario: módulo core

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



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***

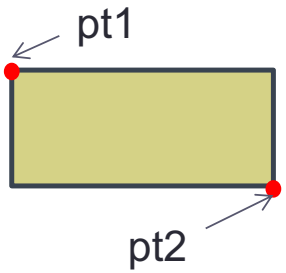
```
ostringstream cursorColor ;
cv::Vec3b color = capture.at< cv::Vec3b>(y, x);
cursorColor << "RGB(" << (int)color[2] << ","
                                    << (int)color[1] << "," << (int)color[0] << ")";
cout << cursorColor.str() << endl;    // convert to string and show it  in console
```

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**)

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

- 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: **8**-connected line, **4**-connected line, **CV_AA** - antialiased line
  - Grosor: en objetos con área podemos especificar: **CV_FILLED**



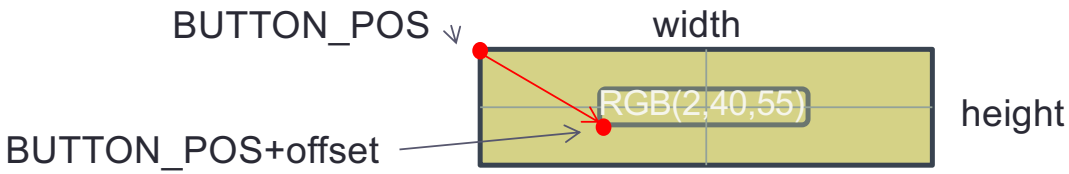

```
cv::Size BUTTON_SIZE (160,25);// overlay button Size (width,heigth)
cv::Point BUTTON_POS (0,0);     // overlay button upper left corner position (x,y)

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**=8, bool **bottomLeftOrigin**=false)
  - Fuentes: CV_FONT_HERSHEY_SIMPLEX, CV_FONT_HERSHEY_PLAIN, CV_FONT_HERSHEY_DUPLEX
  - 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_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

           // wait 10ms for a keystroke to exit (image window must be on focus)
           key = cv::waitKey (10);
           if (key == 'q' || key == 'Q' || key == 27 || EXIT)
                break;
}
```

FILTRADO DE IMÁGENES

Convolución (ej2b)

https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html

Filtrado de Imágenes: módulo imgproc

- Ejemplo (**ej2b.cpp**): filtrado de imágenes (máscaras de 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

```
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) );

cv::cvtColor( capture, gray_image, CV_BGR2GRAY );       // transforms to gray level

cv::filter2D (gray_image, filtered_image, -1 , kernel, cv::Point( -1, -1 ), 0, cv::BORDER_DEFAULT );
```

PERSISTENCIA

Ficheros XML/YAML

https://docs.opencv.org/2.4/modules/core/doc/xml_yaml_persistence.html

Persistencia: XML/YAML, módulo core

- Lectura/escritura de datos en ficheros XML/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
    - **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/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/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/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();
```

EXTRACCIÓN DE CONTORNOS

Módulo imgproc (ej2c)

https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej2c.cpp**): extracción de contornos (listas de puntos enlazados)
- Extraer contornos de una imagen binaria (**Canny**):
  - Métodos:
    - void cv::**findContours**(cv::InputOutputArray **image**, cv::OutputArrayOfArrays **contours**, cv::OutputArray **hierarchy**, int **mode**, int **method**, cv::Point **offset**=cv::Point())

Modes: **CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP, CV_RETR_TREE**
Methods: **CV_CHAIN_APPROX_NONE, CV_CHAIN_APPROX_SIMPLE ,**
CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS
hierarchy: **vector<** cv::**Vec4i>** **hierarchy[i][0]->next , [1] -> previous, [2]->child, [3]-> parent**

```
cv::cvtColor( capture,  gray_image, CV_BGR2GRAY );       // transforms to gray level
cv::Canny (gray_image,  edge_image, 50, 200, 3);      // extracts edges (binary)

vector<vector< cv::Point> > contours_image;
vector< cv::Vec4i> hierarchy;

cv::findContours ( edge_image,  contours_image,   hierarchy,
                   CV_RETR_EXTERNAL,  CV_CHAIN_APPROX_NONE );
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej2c.cpp**): extracción de contornos (listas de puntos enlazados)
- Interpolar y filtrar contornos:
  - Métodos:
    - void cv::**approxPolyDP**(cv::InputArray **curve**, cv::OutputArray **approxCurve**, double **epsilon**, bool **closed**)

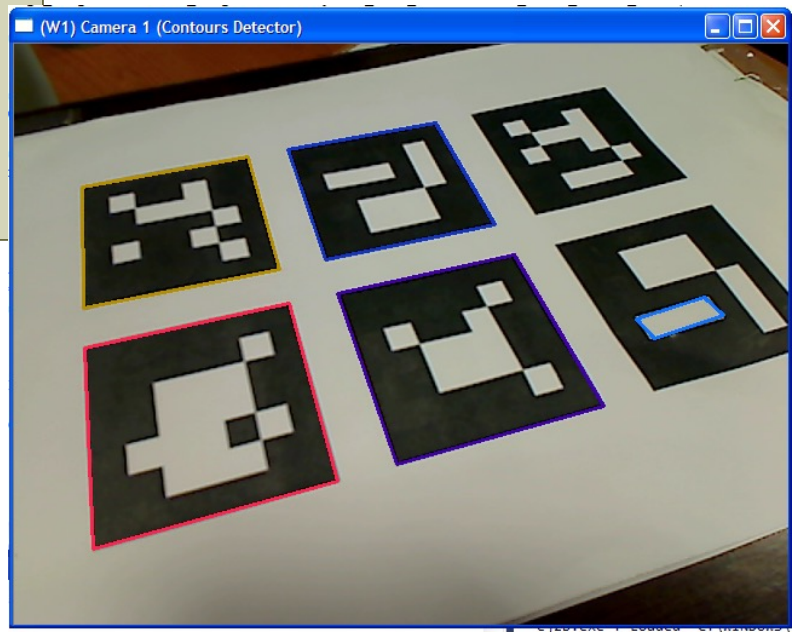
```
// Filter out non rectangular contours
vector<vector< cv::Point> > contours_draw;

for ( unsigned int i=0;i<contours_image.size();i++ )
{
        //approximate to a polygon

        vector< cv::Point > approxCurve;
        cv::approxPolyDP(contours_image[i], approxCurve, double(contours_image[i].size())*0.05, true);

        //checks that the polygon has 4 points, is convex and is big enough
        if ( approxCurve.size() == 4   && cv::isContourConvex( approxCurve)
                        && cv::contourArea( approxCurve)>200 )
                contours_draw.push_back (approxCurve);
}
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej2c.cpp**): extracción de contornos (listas de puntos enlazados)

- Mostrar contornos:
  - void cv::**drawContours**(cv::InputOutputArray **image**, cv::InputArrayOfArrays **contours**,
    int **contourIdx**, const Scalar& color, int thickness=1, int lineType=8,
    cv::InputArray hierarchy=noArray(), int maxLevel=INT_MAX, cv::Point offset=cv::Point())

```
for( unsigned int i = 0; i< contours_draw.size(); i++ )
{
  cv::Scalar color( rand()&255, rand()&255, rand()&255 );
  cv::drawContours( capture, contours_draw, i, color, 2, 8 );
}
```



Extracción de Contornos: módulo imgproc

- Ejercicio: Almacenar los contornos en un fichero YAML/XML

```
// Writing the file if there are contours
if(contours_draw.size() > 0 )
{
        cv::FileStorage fs("contours.yml", cv:: FileStorage::WRITE);
        if(fs.isOpened())
        {
                fs << "size" <<  (int)contours_draw.size();
                fs << "contours" << "[:";// first level (vector of contours)
                for( unsigned int i = 0; i< contours_draw.size(); i++ )
                {
                        fs << "[:";  //second level (vector of points)
                        for( unsigned int j = 0; j< contours_draw[i].size(); j++ )
                        {
                                fs << "{:";// third level (class Point)
                                fs << "x" << contours_draw[i][j].x << "y" << contours_draw[i][j].y;
                                fs << "}";
                        }
                        fs << "]";// second level vector
                }
                fs << "]"; // first level vector
                fs.release();
        }
}
```

Extracción de Contornos: módulo imgproc

- Ejercicio: Leer los contornos de un fichero YAML/XML

```
// Reading contours from file

cv::FileStorage fs2("contours.yml", cv::FileStorage::READ);
if(fs2.isOpened())
{
        cv::FileNode node_level1 = fs2["contours"];              // first level (vector of contours)
        for( unsigned int i = 0; i< node_level1.size(); i++ )
        {
                cv::FileNode node_level2 = node_level1[i];  //second level (vector of points)
                vector<cv::Point> contour;
                for( unsigned int j = 0; j< node_level2.size(); j++ )
                {
                        cv::Point pt;
                        node_level2[j]["x"] >> pt.x;
                        node_level2[j]["y"] >> pt.y;
                        contour.push_back(pt);
                }
                cout << contour << endl;
        }
        fs2.release();
}
```

EJERCICIO

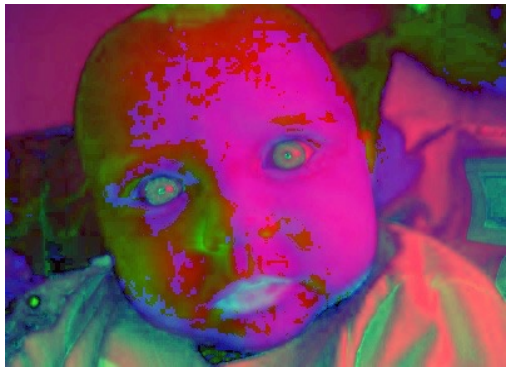
Segmentación de regiones por Color

- Conversión de color
- Extracción de canales de una imagen
- Umbralización
- Filtrado morfológico
- Extracción de contornos
- Cálculo de momentos

https://docs.opencv.org/2.4/modules/imgproc/doc/imgproc.html

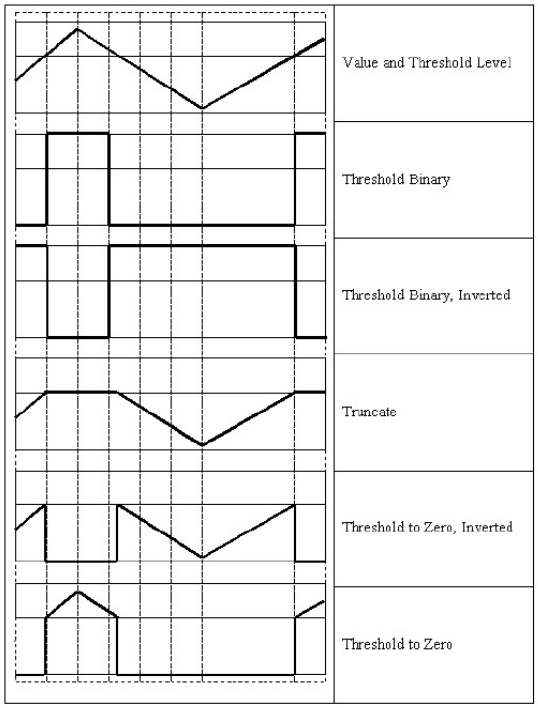
Ejercicio: segmentación de regiones por color

- Programa base: **ej3.cpp**
- Módulos imgproc /core
- Funciones: Conversión de Color:
  - void cv::**cvtColor** (cv::InputArray **src,** cv::OutputArray **dst**, int code, int **dstCn=0**)
    - Códigos:
      - **CV_BGR2HLS, CV_HLS2BGR**
      - **CV_BGR2HSV, CV_HSV2BGR**
      - **CV_BGR2XYZ, CV_XYZ2BGR**
      - **CV_BGR2Lab, CV_Lab2BGR**
      - **CV_BGR2Luv, CV_Luv2BGR …..**
  - void cv::**split**(cv::InputArray **m**, cv::OutputArrayOfArrays **channels**)
    - vector< cv::Mat> channels;
  - void cv::**merge**(cv::InputArrayOfArrays **mv**, cv::OutputArray **dst**)



Ejercicio: segmentación de regiones por color

- Módulos imgproc /core
  - Umbralización:
    - double cv::**threshold**(cv::InputArray **src**, cv::OutputArray **dst**, double **thresh**, double **maxval**, int **type**)
      - Type:
        - cv::**THRESH_BINARY,**
        - cv::**THRESH_BINARY_INV**
        - cv::**THRESH_TRUNC**
        - cv::**THRESH_TOZERO**
        - cv::**THRESH_TOZERO_INV**
      - **Cálculo umbral automático:**
        - **(+ cv::THRESH_OTSU)**



    - double cv::**inRange** (cv::InputArray **src**, cv::InputArray **lowerb**, cv::InputArray **upperb,** cv::OutputArray **dst**)
      - **lowerb, upperb: pueden ser imágenes o escalares**

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0$$

Ejercicio: segmentación de regiones por color

- Módulo imgproc
- Funciones:
  - Morfología:
    - void cv::**erode**(cv::InputArray **src**, cv::OutputArray **dst**, cv::InputArray **kernel**, cv::Point **anchor**=Point(-1,-1), int **iterations**=1)
    - void cv::**dilate**(cv::InputArray **src**, cv::OutputArray **dst**, cv::InputArray **kernel**, cv::Point **anchor**=Point(-1,-1), int **iterations**=1)
    - void cv::**morphologyEx**(cv::InputArray **src**, cv::OutputArray **dst**, int **op**, cv::InputArray **kernel**, cv::Point **anchor**=Point(-1,-1), int **iterations**=1)
      - **op**: cv::**MORPH_OPEN**, cv::**MORPH_CLOSE**, cv::MORPH_GRADIENT, cv::MORPH_TOPHAT, cv::MORPH_BLACKHAT

    Kernel: Mat cv::**getStructuringElement**(int **shape**, cv::Size **ksize**, cv::Point **anchor**=Point(-1,-1))

    shape: cv::**MORPH_CROSS,** cv::**MORPH_RECT,** cv::**MORPH_ELLIPSE**

  - Operaciones lógicas:
    - void cv::**bitwise_and**(cv::InputArray **src1**, cv::InputArray **src2**, cv::OutputArray **dst**, cv::InputArray **mask**=noArray())
    - void cv::**bitwise_or**(cv::InputArray **src1**, cv::InputArray **src2**, cv::OutputArray **dst**, cv::InputArray **mask**=noArray())
    - void cv::**bitwise_not**(cv::InputArray **src**, cv::OutputArray **dst**, cv::InputArray **mask**=noArray())

Ejercicio: segmentación de regiones por color

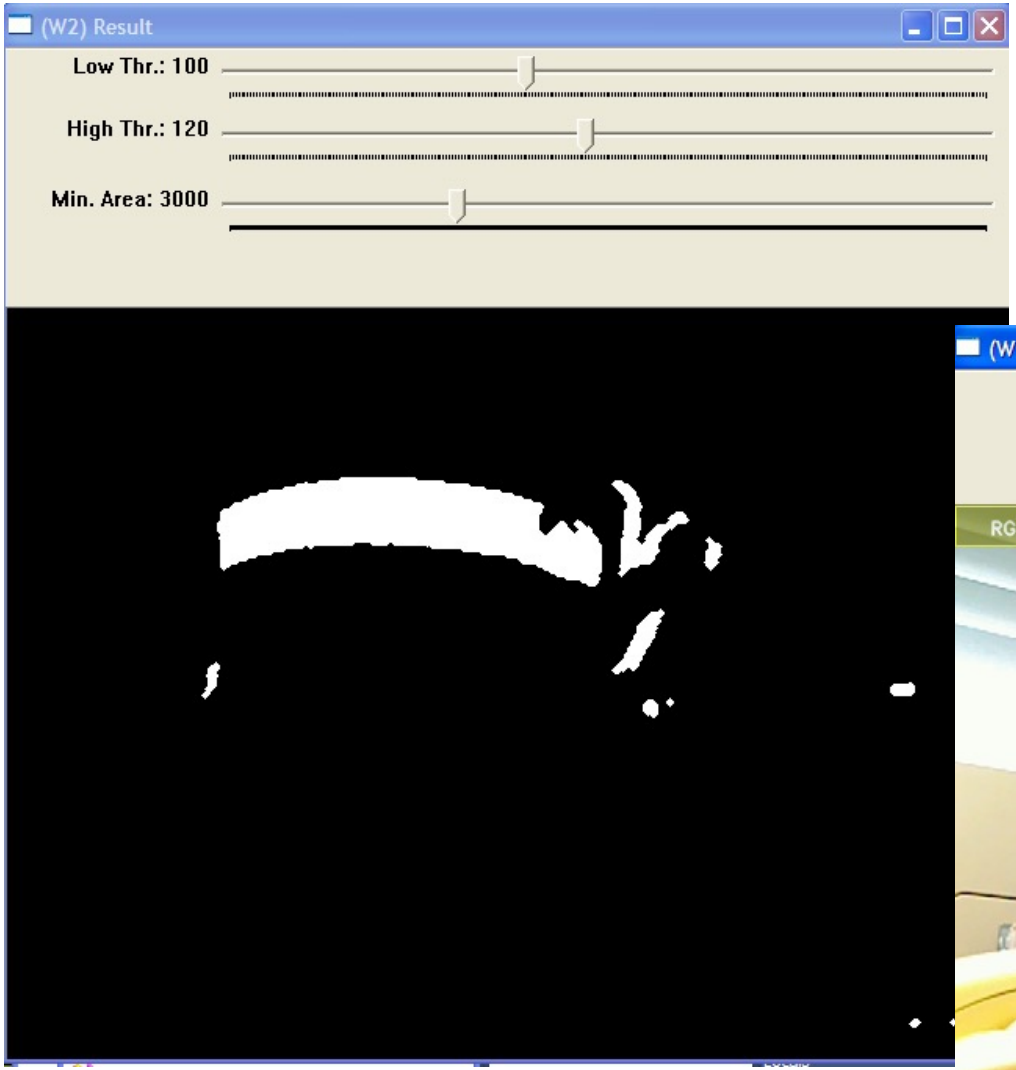
- Módulo imgproc
- Funciones:
  - Momentos:
    - cv::**Moments** cv::**moments**(cv::InputArray **array**, bool **binaryImage**=false)
    - cv::**Moments** class:
      - spatial moments -> **double** m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;
      - central moments -> **double** mu20, mu11, mu02, mu30, mu21, mu12, mu03;
      - central normalized moments -> **double** nu20, nu11, nu02, nu30, nu21, nu12, nu03;
    - void cv::**HuMoments**(const cv::Moments& **moments**, double **hu**[7])
    - void cv::**HuMoments**(const cv::Moments& **moments**, cv::Mat &**hu**)

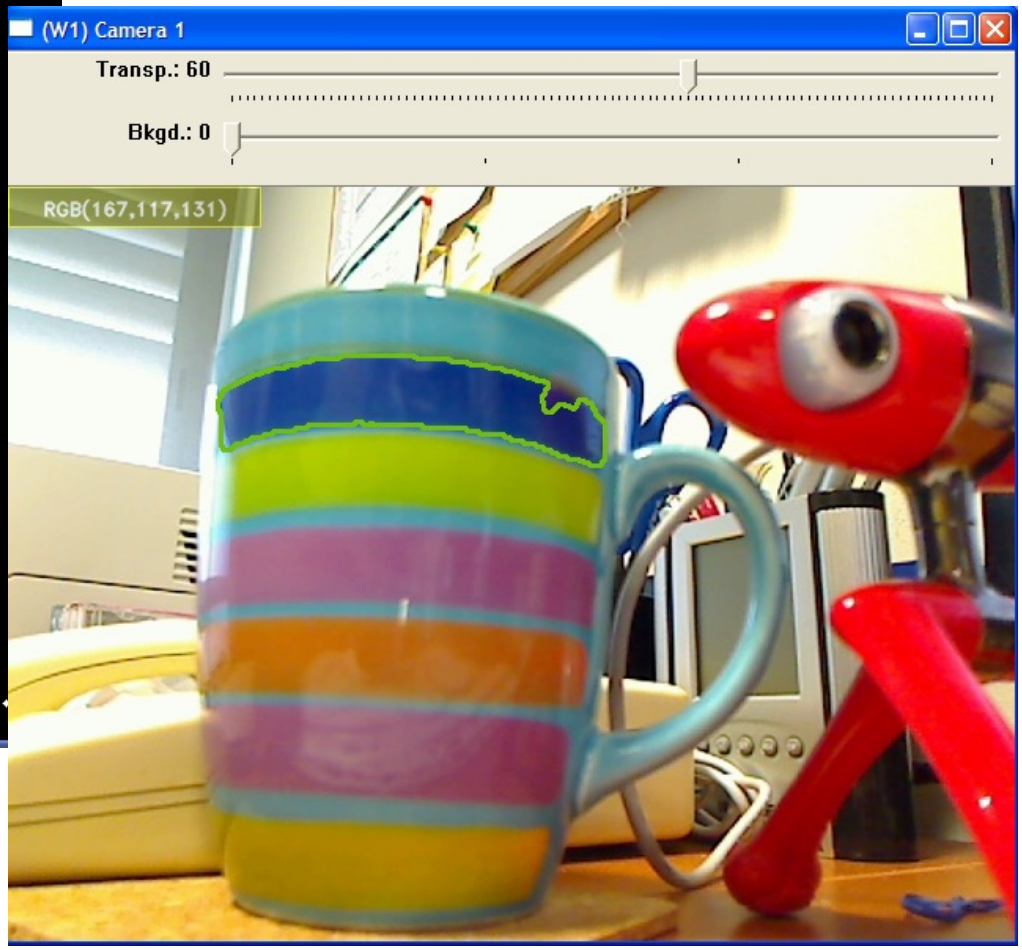
$$\mathtt{m}_{ji} = \sum_{x,y} \left(\mathtt{array}(x,y) \cdot x^j \cdot y^i\right) \qquad \bar{x} = \frac{\mathtt{m}_{10}}{\mathtt{m}_{00}},\ \bar{y} = \frac{\mathtt{m}_{01}}{\mathtt{m}_{00}}$$

$$\mathtt{mu}_{ji} = \sum_{x,y} \left(\mathtt{array}(x,y) \cdot (x-\bar{x})^j \cdot (y-\bar{y})^i\right) \qquad \mathtt{nu}_{ji} = \frac{\mathtt{mu}_{ji}}{\mathtt{m}_{00}^{(i+j)/2+1}}.$$

$$\begin{aligned}
hu[0] &= \eta_{20} + \eta_{02} \\
hu[1] &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
hu[2] &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
hu[3] &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
hu[4] &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
hu[5] &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
hu[6] &= (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}$$

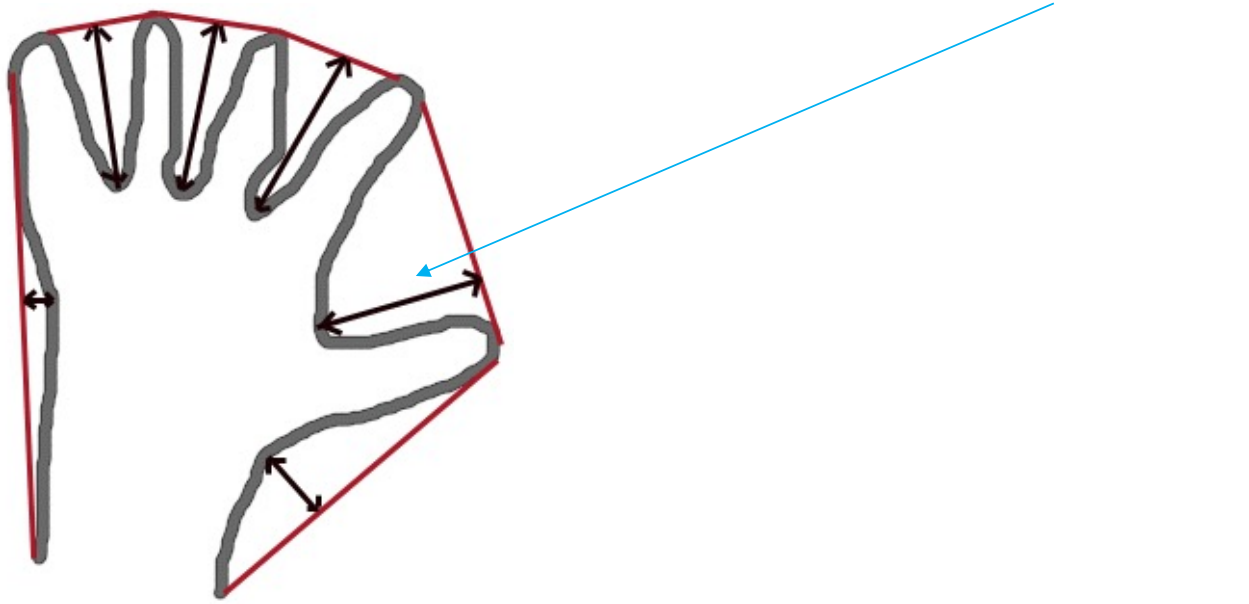
Ejercicio: segmentación de regiones por color





Ejercicio: segmentación de regiones por color

- **Convex Hull**
  - void cv::**convexHull** (cv::InputArray **points**, cv::OutputArray **hull**, bool **clockwise**=false, bool **returnPoints**=true)
    - **points:** vector< cv::**Point** > (un solo contorno)
    - **hull: (2 opciones):**
      - Contorno (puntos)**:** vector< cv::**Point** >
      - Índices de los puntos del contorno: vector<**int**>
  - void cv::**convexityDefects** (cv::InputArray **contour**, cv::InputArray **convexhull**, cv::OutputArray **convexityDefects**)
    - **convexhull :** Índices del contorno: vector<**int**>
    - **convexityDefects** : **vector< cv::Vec4i>** : (start_index, end_index, farthest_pt_index, fixpt_depth)



EXTRACCIÓN CARACTERÍSTICAS

Módulo features2D (ej4)

https://docs.opencv.org/2.4/modules/features2d/doc/features2d.html

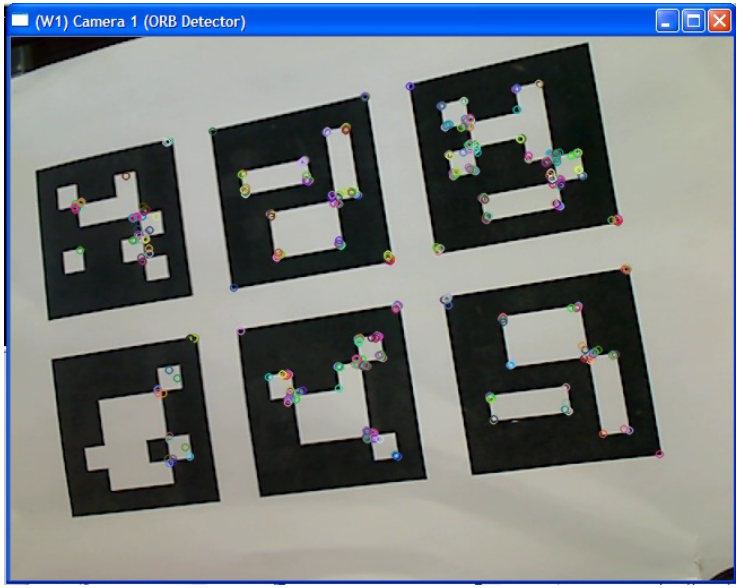
Extracción Características: módulo features2D

- Ejemplo (**ej4a.cpp**): extracción de características (***Common Interface***)

- Extractores:
  - Lista de puntos: vector<cv::**KeyPoint**> keypoints;
    - cv::KeyPoint::**KeyPoint(**cv::Point2f **_pt**, float **_size**, float **_angle**=-1, float **_response**=0, int **_octave**=0, int **_class_id**=-1)
  - **"HARRIS"**, **"FAST"**, **"ORB"**, **"MSER"**, **"BRISK"**, **"STAR"**, **"GFTT"**, "**Dense**", **"SimpleBlob**"
  - **"SIFT"**, **"SURF"**, precisan de una incialización cv::**initModule_nonfree**()
  - Modificadores (**PREFIJO**): **"Grid"** **"Pyramid"**

- Descriptores: cv::**Mat** descriptors;
  - Vectores de características (por filas):
  - **"ORB"**, **"BRISK"**, **"BRIEF"**, **"FREAK"**
  - **"SIFT"**, **"SURF"**,
  - Modificadores (**PREFIJO**): "**Opponent**"

```
Capturing images.
 Hit q/Q to exit.
Detected: 500 points
Descriptor Dimension: 32
[168, 114, 58, 109, 96, 14, 115, 120, 121, 10, 118, 88, 95, 27, 11, 48, 180, 17,
 70, 106, 224, 180, 49, 15, 231, 217, 32, 1, 66, 128, 106, 163]_
```



Extracción Características: módulo features2D

- Ejemplo (**ej4a.cpp**): extracción de características

- Extractores:
  - Lista de puntos: vector<cv::**KeyPoint**> keypoints;
  - **"HARRIS", "FAST", "ORB", "MSER", "BRISK", "STAR", "GFTT", "Dense", "SimpleBlob", "SIFT", "SURF"**

```
cv::initModule_nonfree();// necessary for SIFT/SURF detectors

vector<cv::KeyPoint> keypoints;// Vector for storing detected points

cv::Ptr<cv::FeatureDetector> detector;// Feature Detector handler

detector = cv::FeatureDetector::create("ORB");

(*detector).detect(gray_image, keypoints);// Detects featured points

cv::drawKeypoints (capture, keypoints, capture,
                   cv::Scalar::all(-1), cv::DrawMatchesFlags::DRAW_OVER_OUTIMG );

                                +cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS
```

Dibuja sobre el contenido de la imagen ***capture***

Extracción Características: módulo features2D

- Ejemplo (**ej4a.cpp**): extracción de características

- Descriptores:
  - Vectores de características (por filas): cv::**Mat** descriptors;
  - **"ORB"**, **"BRISK"**, **"BRIEF"**, **"FREAK"**, **"SIFT"**, **"SURF"**

```
cv::Mat descriptors;     // Matrix for storing descriptors, each row is a keypoint descriptor

cv::Ptr<cv::DescriptorExtractor> descriptor;  // Feature Descriptor handler

descriptor = cv::DescriptorExtractor::create("ORB");

(*descriptor).compute(gray_image, keypoints, descriptors);
                                                    // Calculates descriptor vector
```

```
Capturing images.
 Hit q/Q to exit.
Detected: 500 points
Descriptor Dimension: 32
[168, 114, 58, 109, 96, 14, 115, 120, 121, 10, 118, 88, 95, 27, 11, 48, 180, 17,
 70, 106, 224, 180, 49, 15, 231, 217, 32, 1, 66, 128, 106, 163]_
```

Extracción Características: módulo features2D

- Ejemplo (**ej4b.cpp**): extracción de características (***Custom Interface***)
- Clase específica: **SIFT/SURF**:
  - Permite especificar parámetros al algoritmo

```
#include <opencv2/nonfree/nonfree.hpp>
cv::initModule_nonfree(); // necessary for SIFT/SURF detectors
```

```
vector< cv::KeyPoint> keypoints;// Vector for storing detected points
cv::Mat descriptors;   // Matrix for storing descriptors, each row is a keypoint descriptor

cv::SiftFeatureDetector   detector( maxPoints);  // Feature Detector handler
cv::SiftDescriptorExtractor descriptor;  // Feature descriptor handler

detector.detect(gray_image, keypoints);// Detects featured points
descriptor.compute(gray_image, keypoints, descriptors);
```

```
cv::SurfFeatureDetector   detector( minHessian ); // Feature Detector handler
cv::SurfDescriptorExtractor descriptor;  // Feature descriptor handler
```

Extracción Características: módulo features2D

- Ejemplo (**ej4b.cpp**): extracción de características (***Custom Interface***)
- Clases disponibles: (parámetros específicos en cada clase)

- Detectores:
  - cv::**SiftFeatureDetector**
  - cv::**SurfFeatureDetector**
  - cv::**FastFeatureDetector**
  - cv::**OrbFeatureDetector**
  - cv::**GoodFeaturesToTrackDetector**
  - cv::**MserFeatureDetector**
  - cv::**StarFeatureDetector**
  - cv::**DenseFeatureDetector**
  - cv::**SimpleBlobDetector**
  - cv::**BRISK**

- Modificadores (se le pasa un puntero a un detector creado con el operador ***new***)
  - cv::**GridAdaptedFeatureDetector**
  - cv::**PyramidAdaptedFeatureDetector**

- Descriptores:
  - cv::**SiftDescriptorExtractor**
  - cv::**SurfDescriptorExtractor**
  - cv::**BriefDescriptorExtractor**
  - cv::**OrbDescriptorExtractor**
  - cv::**FREAK**
  - cv::**BRISK**

- Modificadores: (se le pasa un puntero a un descriptor creado con el operador ***new***)
  - cv::**OpponentColorDescriptorExtractor**

CORRESPONDENCIA CARACTERÍSTICAS

Módulo features2D

DescriptorMatcher

https://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html

Correspondencia Características: módulo features2D

- Ejemplo (**ej4c.cpp**): extracción de características /matching

- Extractores/Descriptores: **Surf**
- Matcher: **BFMatcher** (Brute Force) / **FlannBasedMatcher**
  - **BFMatcher:** parámetro medida de distancia
    - cv::**NORM_L1**, cv::**NORM_L2** -> SIFT/SURF,
    - cv::**NORM_HAMMING**, cv:: **NORM_HAMMING2** -> ORB, BRIEF, BRISK

- Componentes clase cv::**DMatch**:
  - ***queryIdx:*** **índice vector de descriptores 1 (cámara)**
  - ***trainIdx:*** **índice vector de descriptores 2 (referencia)**
  - ***distance:*** **distancia calculada entre vectores de descripción**

- Métodos clase cv::**DescriptorMatcher**:
  - void cv::DescriptorMatcher::**match**(const cv::Mat& **queryDescriptors**,
    const cv::Mat& **trainDescriptors**,
    vector<cv::DMatch>& **matches**, const cv::Mat& **mask**=cv::Mat())
  - cv::DescriptorMatcher::**knnMatch**(…) : busca k vecinos más cercanos
  - cv::DescriptorMatcher::**radiusMatch**(…): busca vecinos a distancia r

Correspondencia Características: módulo features2D

- Ejemplo (**ej4c.cpp**): extracción de características /matching

- Inicialización:

```
cv::Mat capture, reference_image;     // Images
cv::Mat matches_image;                // draw maches
cv::Mat gray_image;

vector< cv::KeyPoint> keypoints, keypoints_ref;   // Vector for storing detected points
cv::Mat descriptors, descriptors_ref;  // Matrix for storing descriptors
vector< cv::DMatch > matches;              // Vector for storing matches

cv::SurfFeatureDetector   detector(5000);  // Feature Detector handler (Hessian threshold)
cv::SurfDescriptorExtractor descriptor;     // Feature descriptor handler

cv::FlannBasedMatcher matcher;          // FLANN matcher handler

bool reference_state = false;     // we don't have a reference yet
```

Correspondencia Características: módulo features2D

- Ejemplo (**ej4c.cpp**): extracción de características /matching

- Bucle principal:

```
while (camera.read(capture))
{
        cv::cvtColor( capture, gray_image, CV_BGR2GRAY );       // transforms to gray level

       detector.detect(gray_image, keypoints);// Detects featured points
       descriptor.compute(gray_image, keypoints, descriptors);

      if(!reference_state)      // set reference image on first frame
      {
         keypoints_ref  = keypoints;
         descriptors_ref  = descriptors.clone();
         reference_image  = capture.clone();
         reference_state = true;
      }
       // Find matches between camera and reference
       matcher.match( descriptors, descriptors_ref, matches);


       .......
}
```

Correspondencia Características: módulo features2D

- Ejemplo (**ej4c.cpp**): extracción de características /matching

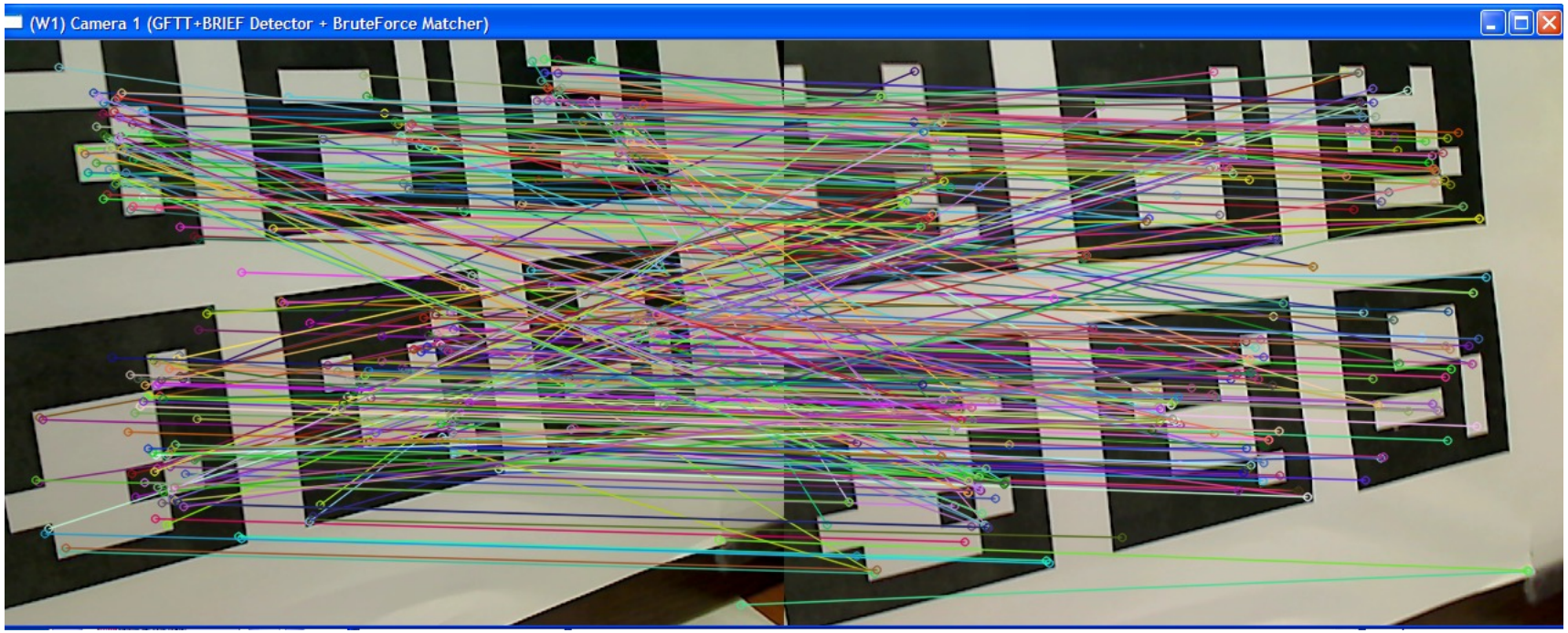
- Visualización Correspondencias:

```
// Display matches
cv::drawMatches( capture, keypoints, reference_image, keypoints_ref,
                 matches, matches_image, cv::Scalar::all(-1), cv::Scalar::all(-1),
                 vector<char>(), cv::DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

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

// shows matches on console
for( int i = 0; i < (int)matches.size(); i++ )
{
   cout  << "-- Match [" << i << "] Keypoint Cam: " <<  matches[i].queryIdx;
   cout  << " -- Keypoint Ref: " << matches[i].trainIdx;
   cout  << " -- dist: " << matches[i].distance << endl;
}
```

Correspondencia Características: módulo features2D

- Ejemplo (**ej4c.cpp**): extracción de características /matching
- Nueva referencia:

```
key = cv::waitKey (10);
if (key == 'q' || key == 'Q'|| key == 27)   break;
else if( key == ' ')          // klick SPACE for a new reference image
{
        keypoints_ref  = keypoints;
        descriptors_ref  = descriptors.clone();
        reference_image  = capture.clone();
}
```



CORRESPONDENCIA CARACTERÍSTICAS

Módulo features2D

Correspondencia entre dos cámaras

Correspondencia Características: módulo features2D

- Ejemplo (**ej4s.cpp**): correspondencia de dos cámaras

- Inicialización de objetos:

```
int CAMERA_ID[2] = {0, 1};//  camera ids

cv::VideoCapture camera[2];     // Camera handlers
cv::Mat draw_image;          // Mat for drawing matches
cv::Mat capture[2];          // Mat headers for both cameras
cv::Mat gray_image[2];

vector< cv::KeyPoint> keypoints[2];  // Vector for storing detected points
cv::Mat descriptors[2];  // Matrix for storing descriptors, each row is a keypoint descriptor
vector< cv::DMatch> matches;              // Vector for storing matches

cv::FastFeatureDetector detector(50);      // FAST Feature Detector handler
cv::BriefDescriptorExtractor extractor;     // BRIEF Feature descriptor handler (Binary)
cv::BFMatcher matcher(cv:: NORM_HAMMING, true);  // Brute Force matcher handler
```

Correspondencia Características: módulo features2D

- Ejemplo (**ej4s.cpp**): correspondencia de dos cámaras
- Almacenamiento de imágenes:
  - Mat Headers:

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

// Creates 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)
```

```
// Configure cameras
    for(int i=0; i<2; i++)
    {
        camera[i].open(CAMERA_ID[i]); //  open  camera
        if (!camera[i].isOpened())
        {
            cout << "you need to connect the camera (" << i+1 << "), sorry.\n";  getchar();  return -1;
        }
        //set capture properties
        camera[i].set(CV_CAP_PROP_FRAME_WIDTH, camSize.width);
        camera[i].set(CV_CAP_PROP_FRAME_HEIGHT, camSize.height);
    }
```

Correspondencia Características: módulo features2D

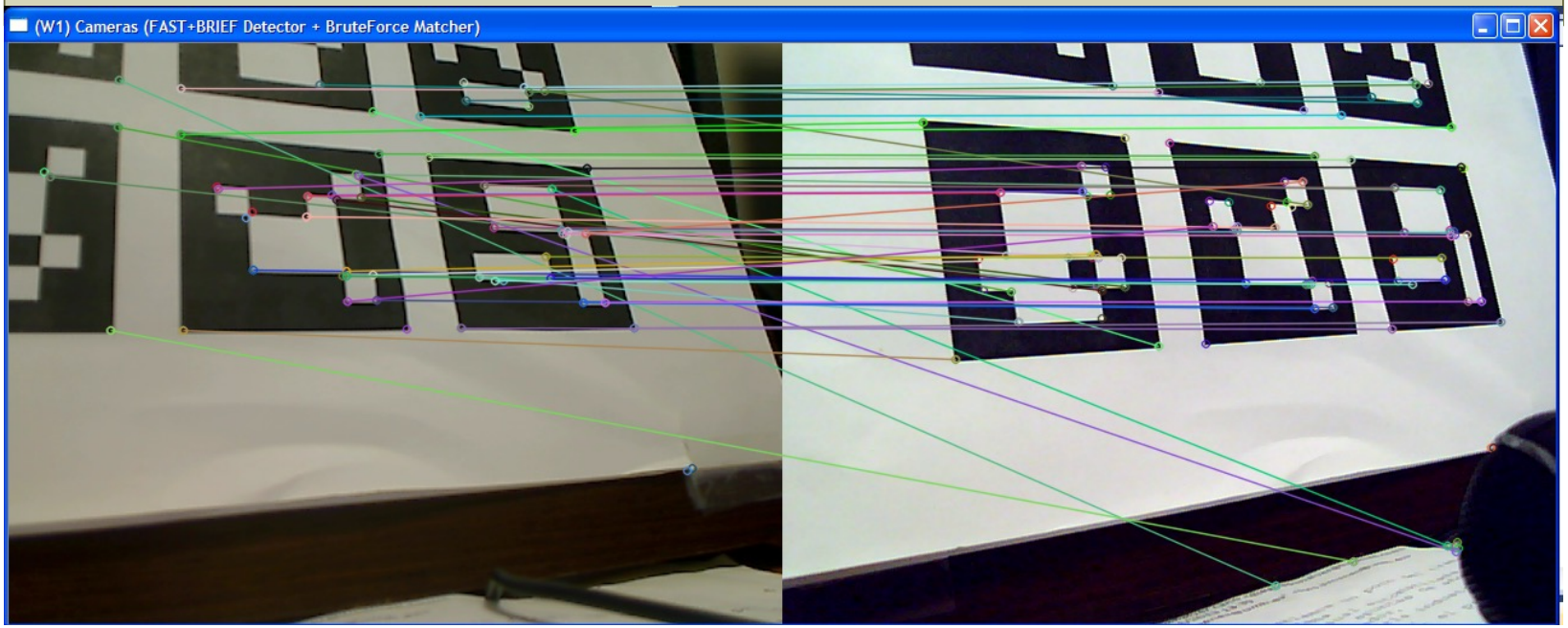
- Ejemplo (**ej4s.cpp**): correspondencia de dos cámaras
- Procesamiento para cada cámara:

```
while (1)
{
        // read image from each camera
        for(int i=0; i<2; i++)
            camera[i].read(capture[i]); // read camera frame

        if(capture[0].empty() || capture[1].empty() )
                continue;    // capture has failed, continue

        for(int i=0; i<2; i++)
        {
            cv::cvtColor( capture[i], gray_image[i], CV_BGR2GRAY );  // color to gray

            detector.detect(gray_image[i],  keypoints[i]);// Detects featured points

            extractor.compute(gray_image[i],  keypoints[i], descriptors[i]);
                                // Calculates featured  descriptors
        }
        // Find matches between camera and reference
      ……..
}
```

Correspondencia Características: módulo features2D

- Ejemplo (**ej4s.cpp**): correspondencia de dos cámaras
- Visualización correspondencias:

```
// Find matches between camera and reference
matcher.match( descriptors[0], descriptors[1], matches);

// Display matches
cv::drawMatches( capture[0], keypoints[0], capture[1], keypoints[1],
                 matches, draw_image, cv::Scalar::all(-1), cv::Scalar::all(-1),
                 vector<char>(), cv::DrawMatchesFlags::DRAW_OVER_OUTIMG );
cv::imshow(WINDOW_CAMERAS, draw_image);// show draw image in a window
```



DETECCIÓN DE MARCAS

Librería ARUCO (Grupo AVA Univ. Córdoba)

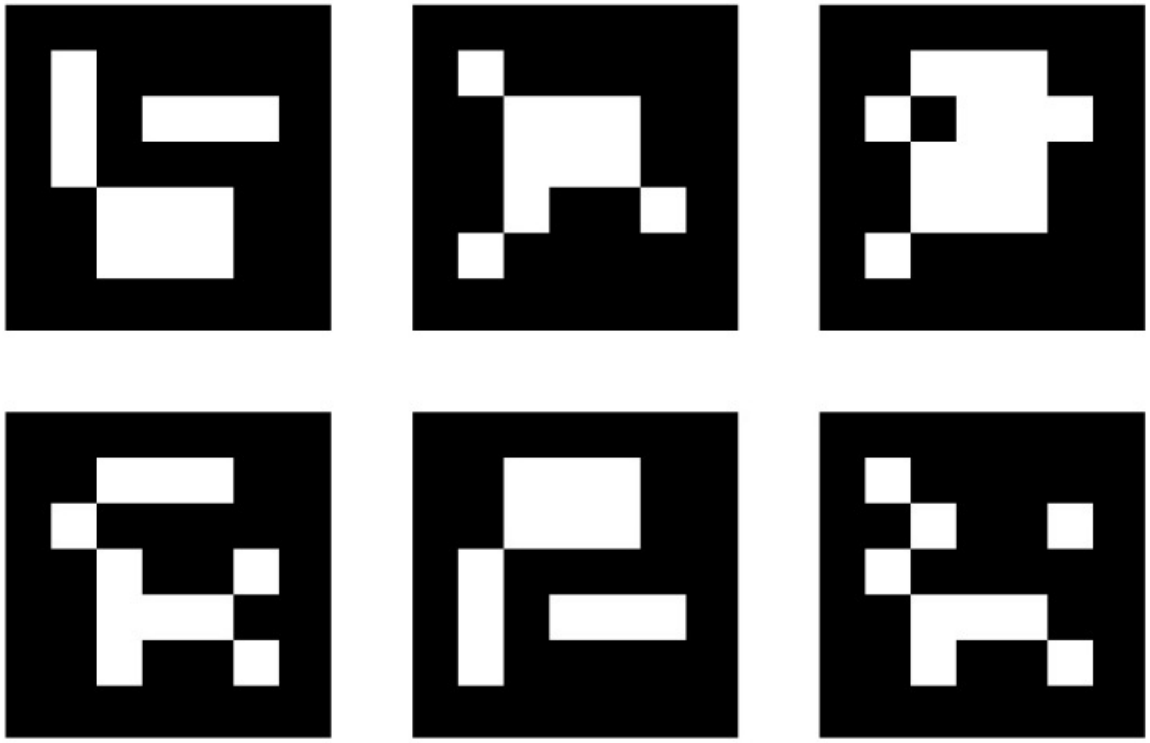
https://www.uco.es/investiga/grupos/ava/node/26

Detección de Marcas: ARUCO

- Instalación de la librería **ARUCO** (Grupo AVA Univ. Córdoba)
  - Tutoriales: http://umh1782.edu.umh.es/material/opencv/
  - Windows ***VC10/VC12-OpenCV 2.4.8/11***: Descargar la librería ya compilada
    - http://umh1782.edu.umh.es/material/opencv/#Software
    - Descomprimir en la carpeta **c:\opencv**
    - Añadir la librería a las hojas de propiedades
      - Carpeta librerías: ***c:\opencv\lib***
      - Librería adicional: ***aruco125d.lib (OpenCV 2.4.8-VC10)***
        - ***Aruco125ocv248vc10d OpenCV 2.4.8-VC10)***
        - ***Aruco125ocv2411vc10d OpenCV 2.4.11-VC10)***
        - ***Aruco125ocv248vc12d OpenCV 2.4.8-VC12)***
        - ***Aruco125ocv248vc12d OpenCV 2.4.11-VC12)***
  - Otras plataformas: descargar código fuente y compilar con ***CMake***
    - http://sourceforge.net/projects/aruco/

Detección de Marcas: ARUCO

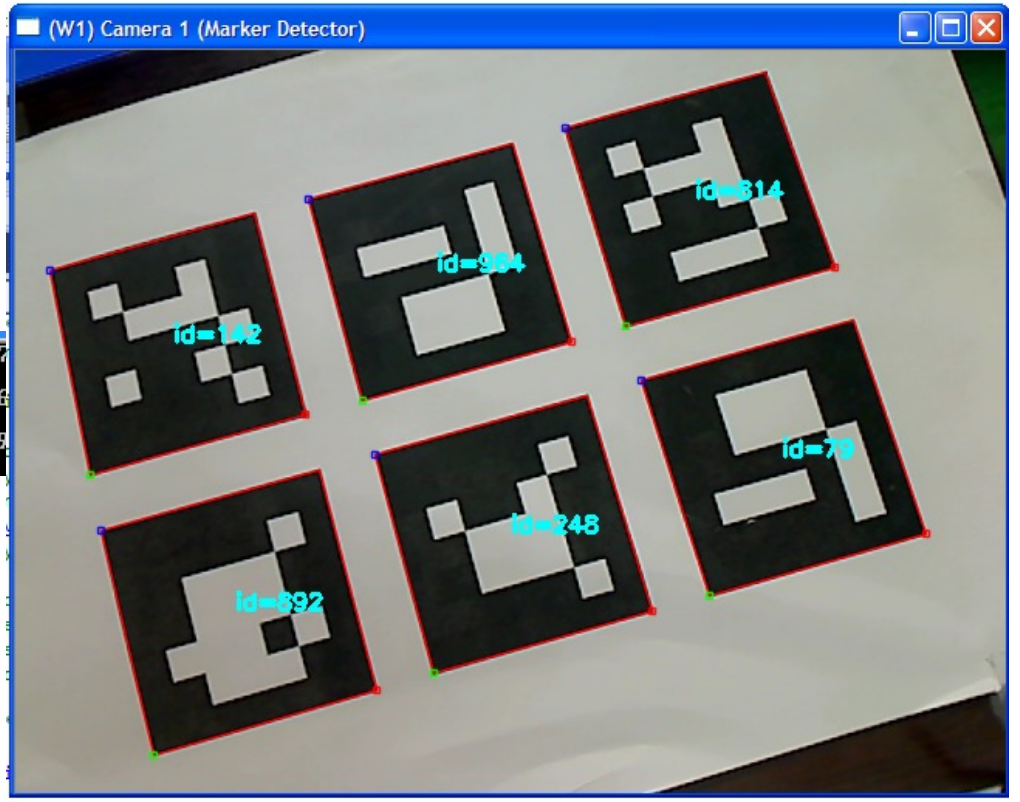
- Ejemplo (**ej5.cpp**)
  - Detectar y localizar marcadores codificados en la imagen
  - Cámara no calibrada
  - http://umh1782.edu.umh.es/opencv2/
  - Fichero de tabla con marcadores: ***markersBoard.pdf***



Detección de Marcas: ARUCO

- Ejemplo (**ej5.cpp**)
- Clases:
  - aruco::**MarkerDetector**
    - void MarkerDetector::**detect** (const cv::Mat &input, vector<aruco::Marker> &detectedMarkers)
  - aruco::**Marker**
    - void aruco::Marker::**draw**(cv::Mat &in, cv::Scalar color, int lineWidth=1, bool writeId=true)

```
Marker found id=142  Corners:[203.199, 241.962][62.3554, 279.877
54][170.196, 109.235]
Marker found id=248  Corners:[430.853, 373.421][286.081, 413.286
21][387.444, 230.51]
Marker found id=814  Corners:[550.907, 147.541][413.062, 185.569
][504.638, 19.194]
```



Detección de Marcas: ARUCO

- Ejemplo (**ej5.cpp**)
  - Detectar y localizar marcadores codificados en la imagen

- Inicialización:

```
// ARUCO Library (Markers)
#include <aruco/aruco.h>
```

```
cv::VideoCapture camera;
cv::Mat capture, gray_image;

bool markerWasFound = false;
aruco::MarkerDetector MDetector;   // handler for marker detector
vector<aruco::Marker> Markers;              // storage for detected markers
```

Detección de Marcas: ARUCO

- Ejemplo (**ej5.cpp**)
  - Detectar y localizar marcadores codificados en la imagen

```
while (camera.read(capture))
{
        cv::cvtColor( capture, gray_image, CV_BGR2GRAY );// transforms to gray level

       MDetector.detect(gray_image, Markers);  // detects markers on image
       markerWasFound = (Markers.size()>0);  // checks if some marker was found

       //for each marker, draw info and its boundaries in the image
       for (unsigned int i=0; i<Markers.size(); i++)
       {
              cout << "Marker found id=" << Markers[i].id ;
              cout << "  Corners:";
              for(unsigned int j=0; j< 4; j++)
                     cout << Markers[i][j];
              cout << endl;

              Markers[i].draw(capture, cv::Scalar(0,0,255), 1);
       }

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

       ……….
}
```

DETECCIÓN DE MARCAS

Librería ARUCO (Grupo AVA Univ. Córdoba)

Cálcular la POSE 3D

Detección de Marcas: ARUCO

- Ejemplo (**ej6.cpp**)
  - Detectar y localizar marcadores codificados en la imagen
  - Calcular la POSE y dibujar un cubo en 3D proyectado en la imágen
  - Cámara Calibrada
  - http://umh1782.edu.umh.es/opencv2/
  - Fichero de configuración de marcadores: ***ARUCO_board.yaml***
  - Fichero de calibración de la cámara: ***camera.yaml***

```
%YAML:1.0
aruco_bc_nmarkers: 6
aruco_bc_mInfoType: 0
aruco_bc_markers:
  - { id:79, corners:[ [ -350., -225., 0. ], [ -150., -225., 0. ], [
      -150., -25., 0. ], [ -350., -25., 0. ] ] }
  - { id:248, corners:[ [ -100., -225., 0. ], [ 100., -225., 0. ], [
      100., -25., 0. ], [ -100., -25., 0. ] ] }
  - { id:892, corners:[ [ 150., -225., 0. ], [ 350., -225., 0. ], [
      350., -25., 0. ], [ 150., -25., 0. ] ] }
  - { id:814, corners:[ [ -350., 25., 0. ], [ -150., 25., 0. ], [ -150.,
      225., 0. ], [ -350., 225., 0. ] ] }
  - { id:964, corners:[ [ -100., 25., 0. ], [ 100., 25., 0. ], [ 100.,
      225., 0. ], [ -100., 225., 0. ] ] }
  - { id:142, corners:[ [ 150., 25., 0. ], [ 350., 25., 0. ], [ 350.,
      225., 0. ], [ 150., 225., 0. ] ] }
```

```
%YAML:1.0
# default calibration data

calibration_date: "Friday Feb 10 14:09:29 2012\n"

camera_matrix: !!opencv-matrix
 rows: 3
 cols: 3
 dt: d
 data: [ 523.18685, 0., 308.07685, 0., 521.88861, 194.65737, 0., 0., 1. ]

distortion_coefficients: !!opencv-matrix
 rows: 4
 cols: 1
 dt: d
 data: [ 0.05141, -0.18643, 0.00977, -0.00059 ]
```

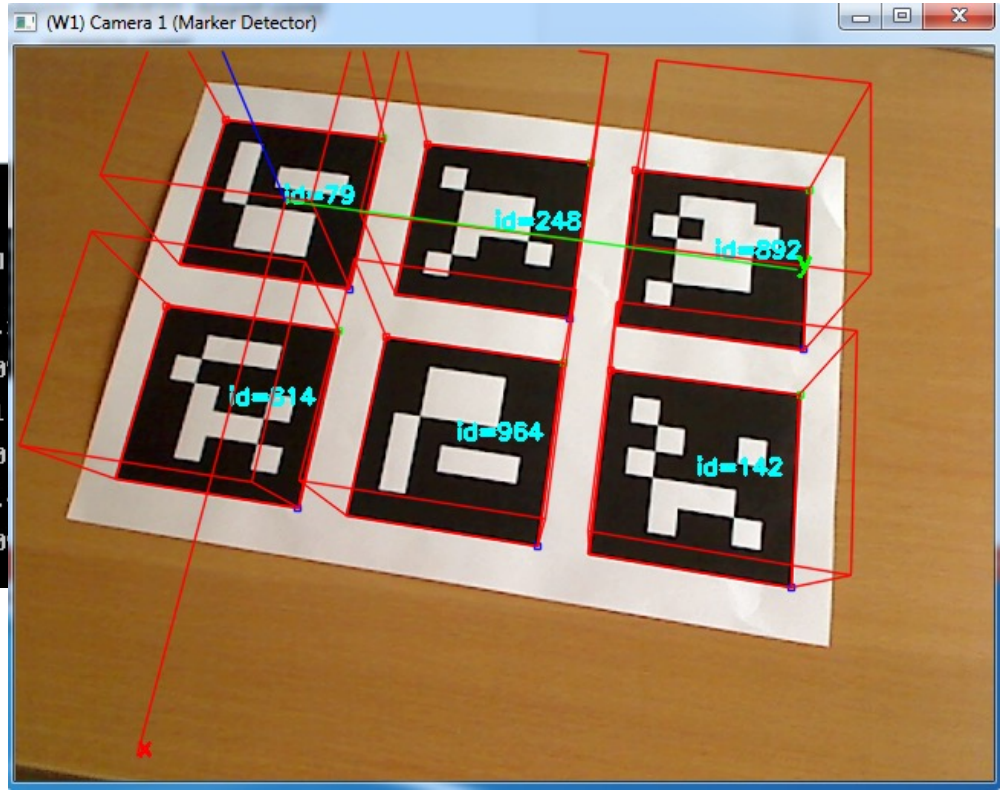
Detección de Marcas: ARUCO

- Ejemplo (**ej6.cpp**)
- Clases:
  - aruco::**CameraParameters**
    - void CameraParameters:: **setParams**(cv::Mat cameraMatrix, cv::Mat distorsionCoeff, cv::Size size)
  - aruco::**BoardConfiguration**
    - void BoardConfiguration ::**readFromFile(** cv::FileStorage &fs**)**
  - aruco::**Marker**
    - void Marker::**calculateExtrinsics**(float markerSize, const aruco::CameraParameters &CP, bool setYPerpendicular=true)
    - Componentes de **Marker**: cv::Mat **Rvec**,**Tvec**;
- Funciones dibujo:
  - aruco::CvDrawingUtils::**draw3dCube** (cv::Mat &Image, Marker &m, const aruco::CameraParameters &CP, bool setYperpendicular=false)
  - aruco::CvDrawingUtils::**draw3dAxis** (cv::Mat &Image, Marker &m, const aruco::CameraParameters &CP)

Detección de Marcas: ARUCO

- Ejemplo (**ej6.cpp**)
  - Detectar y localizar marcadores codificados en la imagen
  - Calcular la POSE y dibujar un cubo en 3D proyectado en la imágen
  - Cámara Calibrada
  - Fichero de configuración de marcadores: ***ARUCO_board.yaml***
  - Fichero de calibración de la cámara: ***camera.yaml***

```
Camera Calibration Matrix A:
[523.18685000000001, 0, 308.07685;
  0, 521.88861, 194.65737;
  0, 0, 1]
Distortion Coefs: [0.05141; -0.18643; 0.009769999999999999; -0.00059]
Capturing images.
 Hit q/Q to exit.
79=(585.029,313.091) (435.003,313.135) (429.88,179.141) (572.962,170.
.107226 0.0258152 0.288654 Rxyz=1.94742 -2.01441 -0.164697
Marker id:79  Rvec: [1.9474186; -2.0144053; -0.16469748]  Tvec: [0.10
25815174; 0.28865433]
142=(241.294,160.179) (137.063,167.925) (146.863,64.2792) (246.151,51
=-0.0782438 -0.0574171 0.350953 Rxyz=1.95555 -1.98461 -0.0788051
Marker id:142  Rvec: [1.9555527; -1.9846056; -0.078805067]  Tvec: [-0
 -0.057417136; 0.35095277]
79=(544.986,306.213) (408.565,308.547) (402.23,186.169) (531.816,177.
.096894 0.0284821 0.315065 Rxyz=1.91834 -1.9919 -0.0517132
Marker id:79  Rvec: [1.9183398; -1.9919027; -0.051713243]  Tvec: [0.0
.028482057; 0.31506509]
```



Detección de Marcas: ARUCO

- Ejemplo (**ej6.cpp**)
  - Declaración de objetos

```
// ARUCO Library (Markers)
#include <aruco/aruco.h>
```

```
bool markerWasFound = false;
aruco::MarkerDetector MDetector;   // handler for marker detector
vector<aruco::Marker> Markers;              // storage for detected markers

float markerSize = (float)0.076; // size of ARUCO marker (meters)
aruco::BoardConfiguration boardInfo;// ARUCO board info
aruco::CameraParameters cameraParameters; //ARUCO class for camera parameters

cv::Mat cameraMatrix,  distCoeffs;     // Intrinsic Camera Calibration parameters
```

Detección de Marcas: ARUCO

- Ejemplo (**ej6.cpp**)
  - Inicializar objetos:

```
// Load ARUCO board info
boardInfo.readFromFile("ARUCO_board.yaml");

// Load camera calibration data
cv::FileStorage fs ("camera.yaml", cv::FileStorage::READ);
if (fs.isOpened())
{
        fs["camera_matrix"] >> cameraMatrix;
        fs["distortion_coefficients"] >> distCoeffs;
        cout << "Camera Calibration Matrix A: " << endl <<  cameraMatrix << endl;
        cout << "Distortion Coefs: " << distCoeffs << endl;

        // configure internal ARUCO cameraParameters object
        cameraParameters.setParams(cameraMatrix, distCoeffs, camSize);
}
```

Detección de Marcas: ARUCO

- Ejemplo (**ej6.cpp**)
  - Bucle principal:

```
while (camera.read(capture))
{
        cv:: cvtColor( capture, gray_image, CV_BGR2GRAY );// transforms to gray level

       MDetector.detect(gray_image, Markers);  // detects markers on image
       markerWasFound = (Markers.size()>0);    // checks if some marker was found

       if(markerWasFound)
       {        //for each marker calculates POSE and draw 3D cube
              for (unsigned int i=0; i<Markers.size(); i++)
              {
                     Markers[i].calculateExtrinsics(markerSize, cameraParameters, false);
                     cout << "Marker id:" << Markers[i].id << "  Rvec: " << Markers[i].Rvec <<
                                          "  Tvec: " << Markers[i].Tvec << endl;
                     Markers[i].draw(capture, cv:: Scalar(0,0,255), 1);
                     aruco::CvDrawingUtils::draw3dCube(capture, Markers[i], cameraParameters);
              }
              // Draws coordinate axis for first marker
              aruco::CvDrawingUtils::draw3dAxis(capture, Markers[0], cameraParameters);
       }
       cv::imshow(WINDOW_CAMERA1, capture);     // show image in a window
}
```