EXTRACCIÓN DE CONTORNOS

Módulo imgproc (ej4)

Listas de puntos enlazados

https://docs.opencv.org/4.5.5/d3/dc0/group__imgproc__shape.html

https://docs.opencv.org/4.5.5/df/d0d/tutorial_find_contours.html

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej4.cpp**): extracción de contornos (listas de puntos enlazados) (partiremos del código de e1b.py)
- 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**

0-based indices in contours

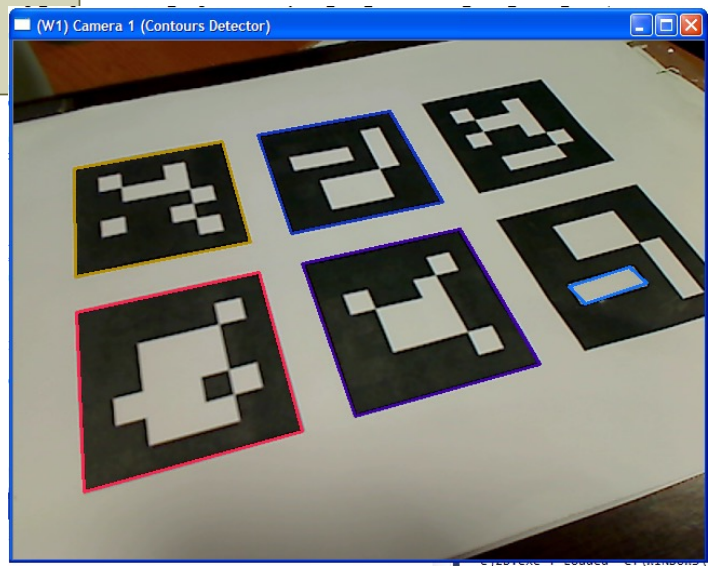
```
cv::cvtColor( capture,  gray_image, cv::COLOR_BGR2GRAY );   // transforms to gray level
cv::Canny (gray_image,  edge_image, 50, 200, 3);      // extracts edges (binary)

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 (**ej4.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=LINE_8, cv::InputArray hierarchy=noArray(), int maxLevel=INT_MAX, cv::Point offset=cv::Point())

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



EJERCICIO 4b

Interpolación y filtrado de contornos (ej4b.cpp)

- Aproximación poligonal
- Propiedades de contornos
- Filtrado de contornos
- Persistencia: almacenamiento

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej4b.cpp**): 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_filtered;

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_filtered.push_back (approxCurve);
}

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

Extracción de Contornos: módulo imgproc

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

```
// Writing the file if there are contours
if(contours_filtered.size() > 0 )
{
        cv::FileStorage fs("contours.yml", cv:: FileStorage::WRITE);
        if(fs.isOpened())
        {
                fs << "size" <<  (int)contours_filtered.size();
                fs << "contours" << "[:";// first level (vector of contours)
                for( unsigned int i = 0; i< contours_filtered.size(); i++ )
                {
                        fs << "[:";  //second level (vector of points)
                        for( unsigned int j = 0; j< contours_filtered[i].size(); j++ )
                        {
                                fs << "{:";// third level (class Point)
                                fs << "x" << contours_filtered[i][j].x << "y" << contours_filtered[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 4c (módulo imgproc)

Detección de rectas y círculos mediante transformada de Hough (ej4c.cpp)

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

https://docs.opencv.org/4.5.5/d9/db0/tutorial_hough_lines.html
https://docs.opencv.org/4.5.5/d4/d70/tutorial_hough_circle.html

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej4c.cpp**): (partiremos del código de ej1.cpp)
- Extraer líneas rectas de una imagen binaria (**Canny**):
  - cv::**HoughLines**(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **srn**=0, double **stn** =0, double **min_theta** =0, double **max_theta** =0)

image: Binary image

lines: vector<cv::**Vec2f**> una línea por fila, 2 elementos: (ρ,θ)
ρ distance from the coordinate origin (0,0) (top-left corner of the image).
θ is the line rotation angle in radians (0~vertical line,π/2~horizontal line).

rho: Distance resolution of the accumulator in pixels.

theta: Angle resolution of the accumulator in radians.

threshold: Only those lines are returned that get enough votes (`>threshold`).

$$\rho = x \cdot \cos(\theta) + y \cdot sen(\theta)$$

```
cv::Mat gray_image, edge_image;

cv::cvtColor( capture,  gray_image, cv::COLOR_BGR2GRAY );   // transforms to gray level
cv::Canny (gray_image,  edge_image, 80, 150, 3);      // extracts edges (binary)

// find straight lines (Hough Transform)
cv::Mat lines;
cv::HoughLines(edge_image, lines, 1, CV_PI/180, 150);
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej4c.cpp**):

- Visualización Líneas:

```
// Draw the lines
for (int i = 0; i < lines.size(); i++)
{
        float rho = lines[i][0], theta = lines[i][1];     // rho = x*cos(theta)+ y*sin(theta)
        cv::Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a * rho, y0 = b * rho;                //  central point (rho is orthogonal to the line)
        pt1.x = cvRound(x0 + 1000 * (-b));                // two far away points of the line
        pt1.y = cvRound(y0 + 1000 * (a));
        pt2.x = cvRound(x0 - 1000 * (-b));
        pt2.y = cvRound(y0 - 1000 * (a));
        // cv::line  do  automatic clipping to the image size

        cv::Scalar color = cv::Scalar(rand() & 255, rand() & 255, rand() & 255);
        cv::line(capture, pt1, pt2, color, 2, cv::LINE_AA);
}
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej4c.cpp**):
- Transformada Hough Círculos:
  - cv.**HoughCircles**(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**=100, double **param2**=100, int **minRadius**=0, int **maxRadius**=0)

image: **grayscale image**

circles vector<cv::**Vec3f**> una línea por fila, 3 elementos (xc,yc,ρ)
(xc,yc); centro del círculo, ρ radio

rmethod: cv::**HOUGH_GRADIENT**, cv::**HOUGH_GRADIENT_ALT**.

dp: Inverse ratio of the accumulator resolution to the image resolution.

minDist: Minimum distance between the centers of the detected circles.

param1: higher threshold of Canny edge detector (the lower one is twice smaller).

param2: the accumulator threshold for the circle centers

$$r^2 = (x - x_c)^2 + (y - y_c)^2$$

```
// find circles (Hough Transform)
vector<cv::Vec3f> circles;
cv::HoughCircles(gray_image, circles, cv::HOUGH_GRADIENT, 1, 20, 150, 30, 20, 100);
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**ej4c.cpp**):

- Visualización Círculos:

```
# Draw  detected circles
for (int i = 0; i < circles.size(); i++)
{
        cv::Vec3i c = circles[i];  // integer approximation
        cv::Point center = cv::Point(c[0], c[1]);
        int r = c[2];

        cv::Scalar color = cv::Scalar(rand() & 255, rand() & 255, rand() & 255);
        cv::circle(capture, center, 1, color, 2, cv::LINE_AA);
        cv::circle(capture, center, r, color, 2, cv::LINE_AA);
}
```

EJEMPLO 5

Segmentación de regiones por Color

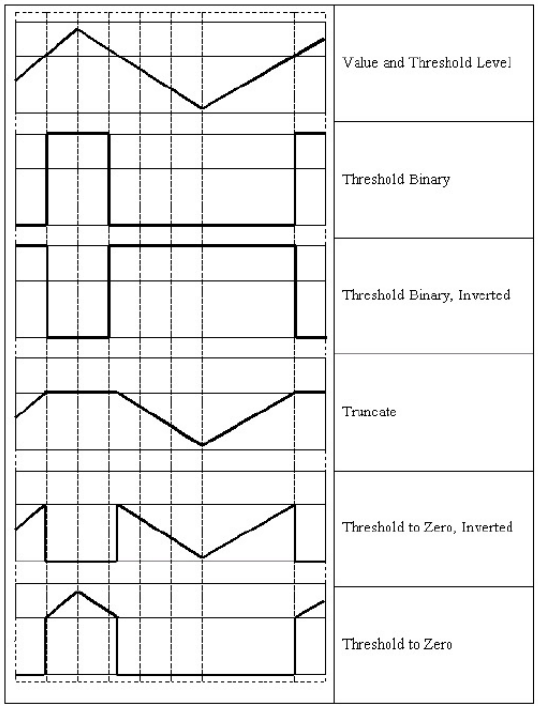
- Conversión de color
- Extracción de canales de una imagen
- Umbralización
- Filtrado morfológico
- Extracción de contornos
- Cálculo de momentos

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::**COLOR_BGR2GRAY**, cv::**COLOR_GRAY2BGR**
      - cv::**COLOR_BGR2HSV**, cv::**COLOR_HSV2BGR,** cv::**COLOR_BGR2HSV_FULL,** cv::**COLOR_HSV2BGR_FULL**
      - cv::**COLOR_BGR2HLS**, cv::**COLOR_HLS2BGR,** cv::**COLOR_BGR2HLS_FULL,** cv::**COLOR_HLS2BGR_FULL**
      - cv::**COLOR_BGR2XYZ**, cv::**COLOR_XYZ2BGR**
      - cv::**COLOR_BGR2Lab,** cv::**COLOR_Lab2BGR** **(escalado 0-1.0 float32 CV_32F)**
      - cv::**COLOR_BGR2Luv,** cv::**COLOR_Luv2BGR** **(escalado 0-1.0 float32 CV_32F)**
      - **…..**
  - void cv::**normalize**(InputArray **src**, **InputOutputArray dst**, double **alpha** = 1, double **beta** = 0, int **norm_type**=NORM_L2, int **dtype**=-1, InputArray **mask**=noArray())
  - 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**
      - **maxval:** valor alto de salida (BINARY)
      - **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

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

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_xor**(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

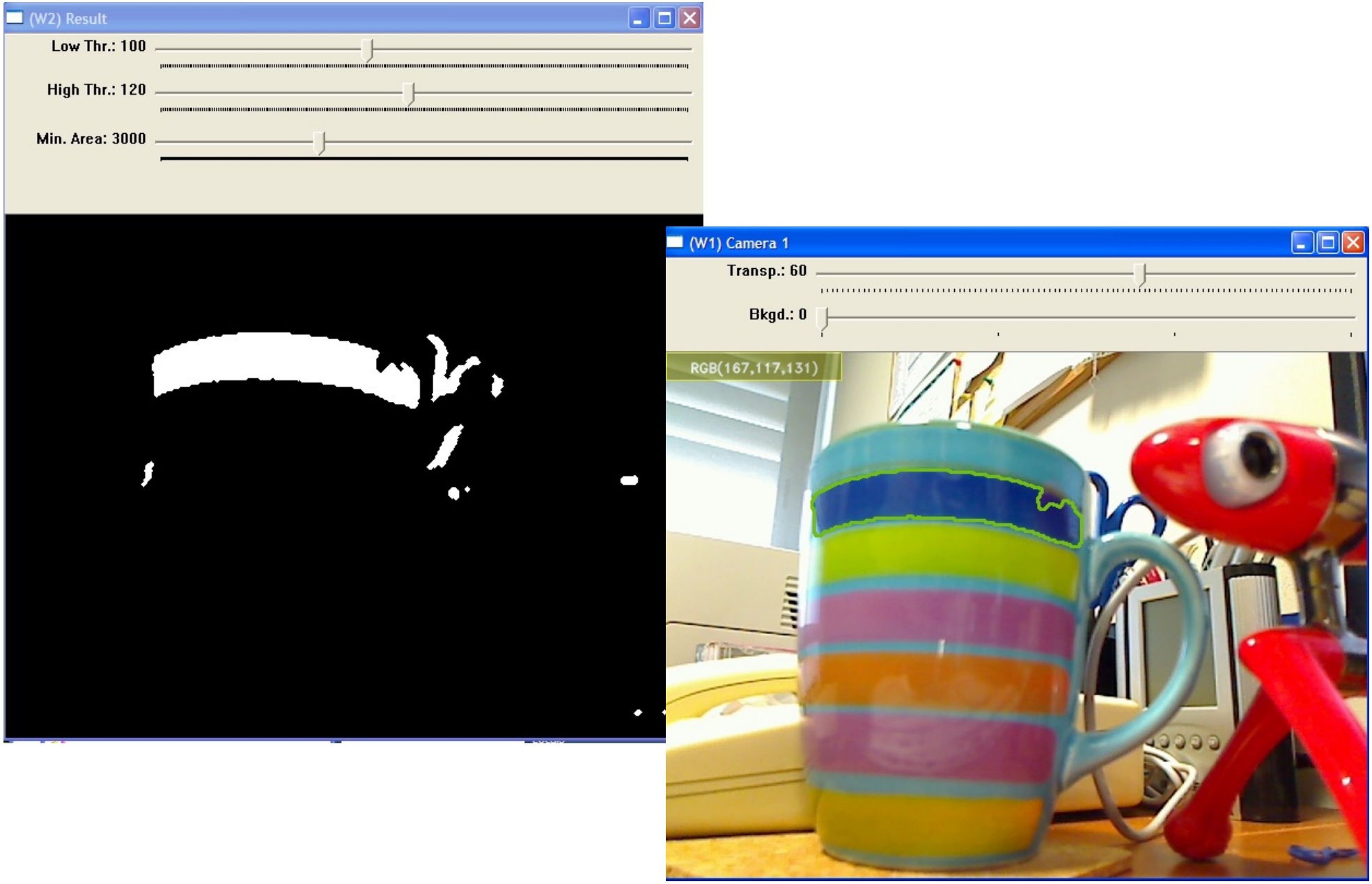
- Módulo imgproc
- Funciones:
  - Momentos:
    - cv::**Moments** cv::**moments**(cv::InputArray **array**, bool **binaryImage**=false)
    - cv::**Moments** class:
      - spatial moments -> **double** m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;
      - central moments -> **double** mu20, mu11, 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**)

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

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

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

Ejercicio: segmentación de regiones por color



Ejercicio: segmentación de regiones por color

- Programa base: **ej5.cpp** (partiremos del código de ej3b.cpp)

```
int AREA_MIN = 1000;          // min area
int TH_LOW = 100;             // Low and High  Hue threshold
int TH_HIGH = 120;
int SI_MIN = 20;              // Filter out regions with low saturation or intensity
```

```
cv::cvtColor( capture, hsv, cv::COLOR_BGR2HSV_FULL ) ; // transforms to HSV

vector<cv::Mat> channels;
cv::split(hsv, channels)
cv::Mat hue = channels[0];
cv::Mat sat = channels[1];
cv::Mat intensity = channels[2];

cv::inRange(hsv, cv::Scalar()TH _LOW, TH _HIGH, res);     // Hue segmentation

// Filter out regions with low saturation or intensity
cv::threshold(sat ,mask1, SI_MIN, 255, cv::THRESH_BINARY); //saturation threshold
cv::threshold(intensity, mask2, SI_MIN, 255, cv::THRESH_BINARY); //intensity threshold
cv::bitwise_and(mask1, mask2, mask);          // useful region
cv::bitwise_and(res, mask, res);              // filters out noisy regions

// morphological filter (opening/closing)
cv::Mat kernel = cv::getStructuringElement(cv::MORPH_CROSS, cv::Size(3, 3));
cv::morphologyEx(res, res, cv::MORPH_OPEN, kernel, cv::Point(-1,-1), 2);
cv::morphologyEx(res, res, cv::MORPH_CLOSE, kernel, cv::Point(-1,-1), 2);
```

Ejercicio: segmentación de regiones por color

- Programa base: **ej5.cpp**

```
# find contours
edge_image = cv.Canny(res, threshold1=50, threshold2=200)

# find contours
cv::findContours ( edge_image , contours_image, hierarchy, cv::RETR_EXTERNAL,
                                cv::CHAIN_APPROX_NONE );

// Filter out small contours and finds bigger contour
vector<vector<cv::Point> > contours_draw;
unsigned int id_max=0;
double area_max=0.0;

for ( unsigned int i=0, j=0; i<contours_image.size();i++ )
{
        double area = cv::contourArea(contours_image[i]);
        if ( area > AREA_MIN )
        {
                contours_draw.push_back(contours_image[i]);
                if(area>area_max)       { id_max= j; area_max = area; }
                j++;
        }
}
```

Ejercicio: segmentación de regiones por color

- Programa base: **ej5.cpp**

```
// Calculates Hu-moments for bigger contour (if any)
if(contours_draw.size() >0)
{
        cv::Moments m = cv::moments(contours_draw[id_max]);
        cv::Mat hu;
        cv::HuMoments(m, hu);
        //cout << "Hu-moments: " <<  hu << endl;
}
```

```
// Draw  contours
for( unsigned int i = 0; i< contours_draw.size(); i++ )
{
        cv::Scalar color = cv::Scalar( rand()&255, rand()&255, rand()&255 );
        cv::drawContours( dispImage, contours_draw, i, color, 2, cv::LINE_AA );
}
```

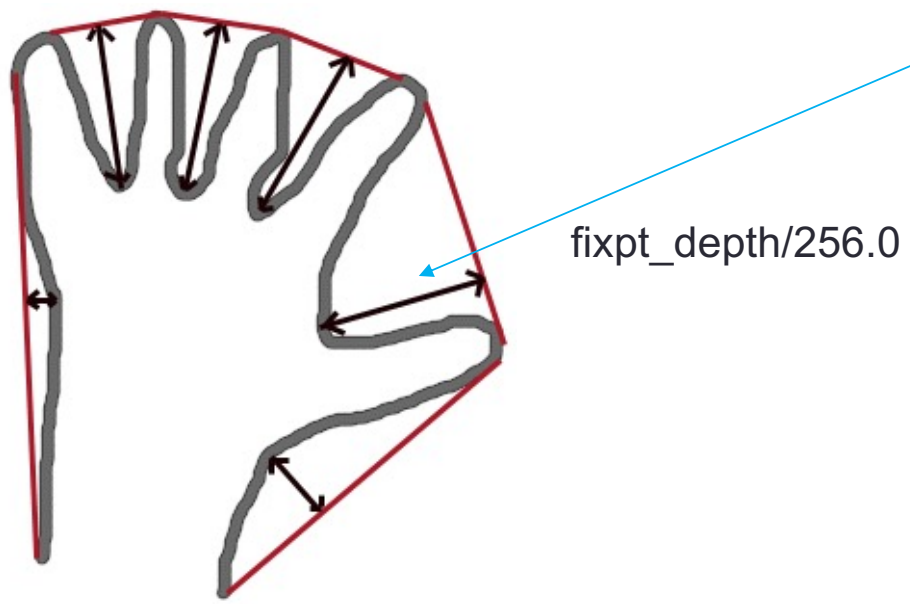
EJERCICIO 5b

Segmentación de regiones por Color

- Incorporar al ejemplo previo el procesamiento del convexHull para la detección de formas

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)



Ejercicio: segmentación de regiones por color

- Programa base: **ej5b.cpp** (partiremos del código de ej5.cpp)
  - LUT: cv::**LUT**(InputArray **src**, InputArray **lut**, OutputArray **dst**) `dst(I) ← lut(src(I))`

```
// Normalize HUE channel (selected hue value in the middle(128))
int HUE_CENTER = 10; // Hue valued to be centered in HUE normalization

// Normalize HUE channel (selected hue value in the middle(128))
cv::Mat lookUpTable(1, 256, CV_8U);
uchar* p = lookUpTable.data;   // faster that using at<> method
for (int i = 0; i < 256; ++i)
      p[i] = (i + 128 - HUE_CENTER) % 256;

cv::LUT(hue, lookUpTable, hue);
```

```
// Calculates Convex Hull of the bigest contour (if any)
vector<int>  convexhull;
vector<cv::Vec4i> convexitydefects;
double convexity_defect = 0.0;
if (contours_image.size() > 0)
{
      cv::convexHull(contours_image[id_max], convexhull, true, false); // Convex Hull indexes clockwise
      cv::convexityDefects(contours_image[id_max], convexhull, convexitydefects);

      // calculates  mean of convexity  defects
      for (unsigned int i = 0, j = 0; i < convexitydefects.size(); i++)
            convexity_defect += convexitydefects[i][3] / 256.0;

      convexity_defect /= convexitydefects.size();
      cout << "Convexity Defect: " << convexity_defect << endl;
}
```

Ejercicio: segmentación de regiones por color

- Programa base: **ej5b.cpp**
  - Visualiza contorno y Convex-Hull

```
// Draw  contours and convexhull
if (contours_image.size() > 0)
{
        // Draws selected contour
        cv::drawContours( dispImage, contours_image, id_max, cv::Scalar(0,255,0), 2, cv::LINE_AA );

        // Draws convex hull for the bigest contour
        for (unsigned int i = 0; i < convexhull.size() - 1; i++)
                cv::line(dispImage, contours_image[id_max][convexhull[i]],
                        contours_image[id_max][convexhull[i + 1]], cv::Scalar(0, 0, 255), 2, cv::LINE_AA);
}
```