Nota: Algunas de las imágenes que aparecen en esta presentación provienen del libro:
Visión por Computador: fundamentos y métodos.
Arturo de la Escalera Hueso. Prentice Hall.

Intelligent Systems Lab

Sistemas de Percepción

Visión por Computador

Arturo de la Escalera
José María Armingol
Fernando García
David Martín
Abdulla Al-Kaff

uc3m | Universidad Carlos III de Madrid

Intelligent Systems Lab

OpenCV: Segmentación: Umbralización, Transformada de Hough y Algoritmo Watershed

Umbralización

- La función se puede utilizar para obtener una imagen binaria (dos niveles de gris).

threshold

Applies a fixed-level threshold to each array element.

C++: double **threshold**(InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**)

Parameters:
- **src** – input array (single-channel, 8-bit or 32-bit floating point).
- **dst** – output array of the same size and type as `src`.
- **thresh** – threshold value.
- **maxval** – maximum value to use with the `THRESH_BINARY` and `THRESH_BINARY_INV` thresholding types.
- **type** – thresholding type (see the details below).

Umbralización

- Parámetro **Type**
  - **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

  - **THRESH_BINARY_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

  - **THRESH_TRUNC**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

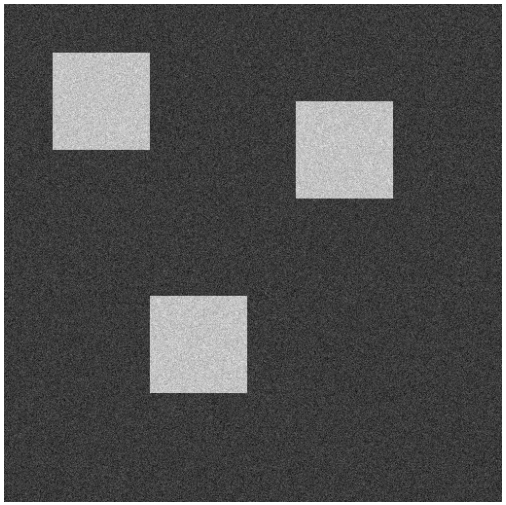
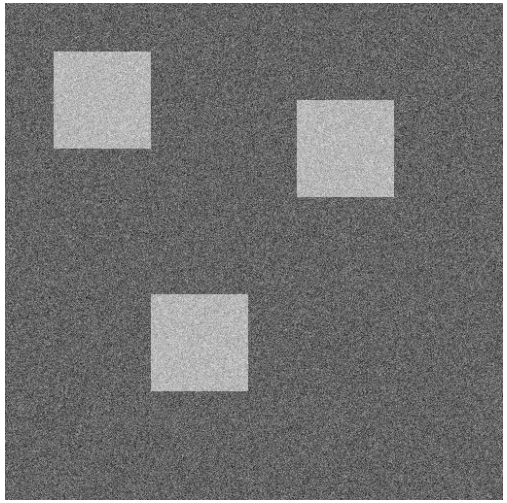
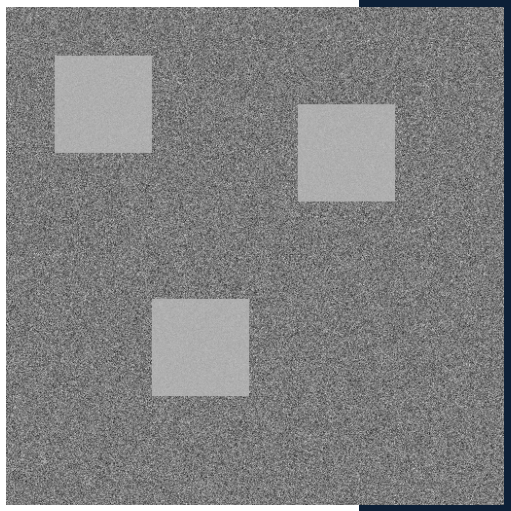
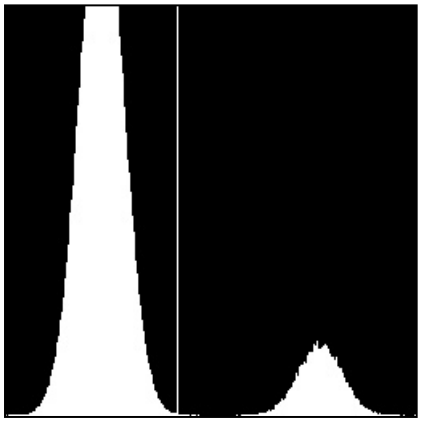
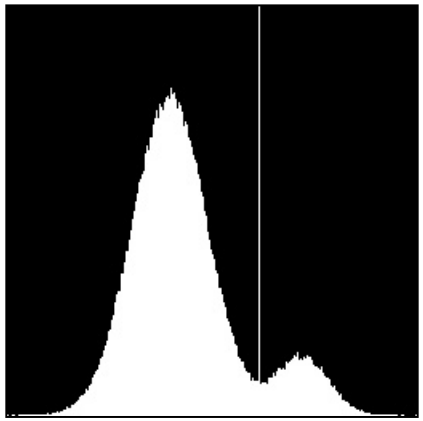
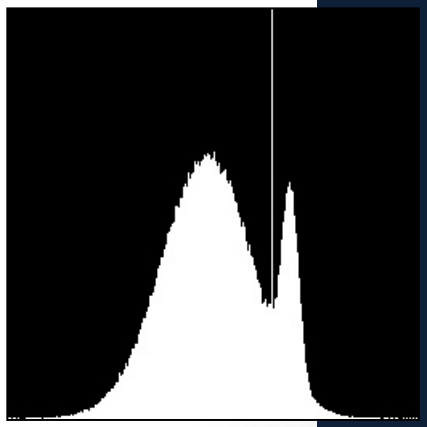
  - **THRESH_TOZERO**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

  - **THRESH_TOZERO_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Umbralización

Umbralización

- Cargar la imagen.
- Calcular el histograma.
- Calcular el valor de umbral (threshold).
- Umbralizar la imagen.
- Mostrar las imágenes y el histograma.
- Esperar a que se pulse una tecla.
- Liberar la memoria.
- Finalizar el programa.

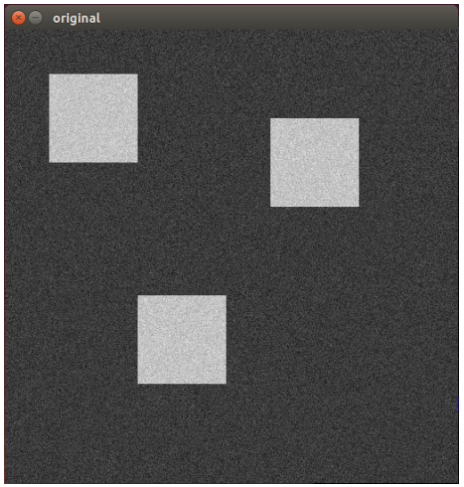
Umbralización

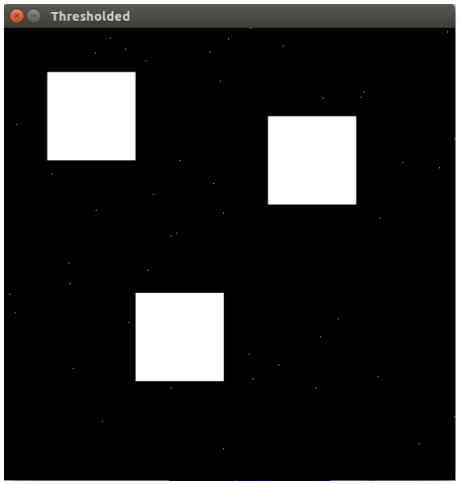
- **Calcular el umbral**

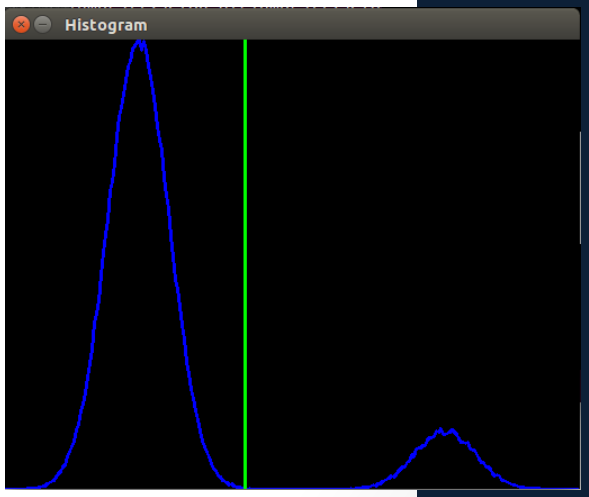
```
int MinMax(long* Histograma)
{
        float HistogramaAux[256];
        int i,maximo1,maximo2,minimo;

        maximo1=0;
        for(i=1;i<256;++i){
                if(Histograma[maximo1]<Histograma[i]){
                maximo1=i;
                }
        }
        for(i=0;i<256;++i){
                HistogramaAux[i]= Histograma[i]*abs(i-maximo1);
        }

        maximo2=0;
        for(i=1;i<256;++i){
                if(HistogramaAux[maximo2]<HistogramaAux[i]){
                        maximo2=i;
                }
        }
        cout << "Maximo1 " << maximo1 << " Maximo2 " << maximo2 << endl;

        if(maximo1<maximo2){
                minimo=maximo1;
                for(i=minimo+1;i<maximo2;++i)
                        if(Histograma[i]<Histograma[minimo]) minimo=i;
        }
        else{
                minimo=maximo2;
                for(i=minimo+1;i<maximo1;++i){
                        if(Histograma[i]<Histograma[minimo]){
                                minimo=i;
                        }

                }
        }
        cout << "Minimo " << minimo << endl;
        return (minimo);
}
```

Umbralización

```
/// Draw
hist_vector[0] = 0;
for (int i = 1; i < histSize; i++)
{
        hist_vector[i] = hist.at<float>(i);
        line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(hist.at<float>(i - 1))),
                Point(bin_w*(i), hist_h - cvRound(hist.at<float>(i))),
                Scalar(255, 0, 0), 2, 8, 0);
}

threshold_value = MinMax(hist_vector);
threshold(img_src, im_thresholded, threshold_value, 255,THRESH_BINARY);
```







Transformada de Hough

- La transformada de Hough es una transformada utilizada para detectar líneas rectas, círculos ...
- Para aplicar la transformada, primero se realiza una detección de bordes sobre la imagen.

Transformada de Hough

Intelligent Systems Lab

- Transformada de Hough estándar

HoughLines

Finds lines in a binary image using the standard Hough transform.

C++: void **HoughLines**(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **srn**=0, double **stn**=0, double **min_theta**=0, double **max_theta**=CV_PI)

Parameters:

- **image** – 8-bit, single-channel binary source image. The image may be modified by the function.
- **lines** – Output vector of lines. Each line is represented by a two-element vector (ρ, θ) . ρ is the distance from the coordinate origin $(0, 0)$ (top-left corner of the image). θ is the line rotation angle in radians ($0 \sim \text{vertical line}, \pi/2 \sim \text{horizontal line}$).
- **rho** – Distance resolution of the accumulator in pixels.
- **theta** – Angle resolution of the accumulator in radians.
- **threshold** – Accumulator threshold parameter. Only those lines are returned that get enough votes ($> \texttt{threshold}$).
- **srn** – For the multi-scale Hough transform, it is a divisor for the distance resolution `rho` . The coarse accumulator distance resolution is `rho` and the accurate accumulator resolution is `rho/srn` . If both `srn=0` and `stn=0` , the classical Hough transform is used. Otherwise, both these parameters should be positive.
- **stn** – For the multi-scale Hough transform, it is a divisor for the distance resolution `theta`.
- **min_theta** – For standard and multi-scale Hough transform, minimum angle to check for lines. Must fall between 0 and max_theta.
- **max_theta** – For standard and multi-scale Hough transform, maximum angle to check for lines. Must fall between min_theta and CV_PI.

Transformada de Hough

- Transformada de Hough Probabilística

HoughLinesP

Finds line segments in a binary image using the probabilistic Hough transform.

C++: void **HoughLinesP**(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **minLineLength**=0, double **maxLineGap**=0)

Parameters:

- **image** – 8-bit, single-channel binary source image. The image may be modified by the function.
- **lines** – Output vector of lines. Each line is represented by a 4-element vector (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) are the ending points of each detected line segment.
- **rho** – Distance resolution of the accumulator in pixels.
- **theta** – Angle resolution of the accumulator in radians.
- **threshold** – Accumulator threshold parameter. Only those lines are returned that get enough votes ($>$ `threshold`).
- **minLineLength** – Minimum line length. Line segments shorter than that are rejected.
- **maxLineGap** – Maximum allowed gap between points on the same line to link them.

Transformada de Hough

- Transformada de Hough Estándar: Programa

```
// Objects
Mat original_img;
// Load image from disk
original_img = imread("building.jpg", CV_LOAD_IMAGE_GRAYSCALE);
if (!original_img.data){
        cout << "error loading image" << endl;
        return 1;
}


// Canny: 2:1 3x3
int low_threshold = 50;
int high_threshold = low_threshold * 4;
int kernel_size = 3;

Mat filtered_img(original_img.size(), original_img.type());
Mat img_color;

Canny(original_img, filtered_img, low_threshold, high_threshold, kernel_size);
cvtColor(filtered_img, img_color, COLOR_GRAY2BGR);
```

Transformada de Hough

- Transformada de Hough: Programa (continuación)

```
// Hough transform (Standard)
vector<Vec2f> lines;
HoughLines(filtered_img, lines, 1, CV_PI/180, 100, 0, 0 );

for( size_t i = 0; i < MIN(lines.size(),50); i++ )
{
        float rho = lines[i][0], theta = lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        pt1.x = cvRound(x0 + 1000*(-b));
        pt1.y = cvRound(y0 + 1000*(a));
        pt2.x = cvRound(x0 - 1000*(-b));
        pt2.y = cvRound(y0 - 1000*(a));
        line(img_color, pt1, pt2, Scalar(0,0,255), 3, CV_AA);
}

// Free memory
original_img.release();
filtered_img.release();
img_color.release();
```

Transformada de Hough

- Transformada de Hough Probabilística: Programa

```
// Hough transform (Probabilistic)
vector<Vec4i> lines;
HoughLinesP(filtered_img, lines, 1, CV_PI/180, 80, 30, 10);
for( size_t i = 0; i < lines.size(); i++ )
{
        Vec4i l = lines[i];
        line(img_color, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, LINE_AA);
}
```

Algoritmo Watershed