

The background of the slide is decorated with a collection of triangles in various shades of green and grey. These triangles are scattered across the slide, with a higher concentration in the top-left corner and a more sparse distribution towards the bottom-right. The triangles vary in size and opacity, creating a layered, geometric effect.

# Visión Artificial

## Práctica 3: detección de objetos

# Pasos para el desarrollo

## ▼ Creación de una colección:

- ▼ El usuario puede seleccionar hasta 3 objetos distintos añadiendo para cada uno una ventana de imagen.
- ▼ Se asume que los objetos son planos y tienen suficiente textura.

## ▼ Detección de objetos (si se ha creado una colección):

- ▼ Por cada nueva captura de cámara, el programa detecta y muestra los objetos detectados de la colección.
- ▼ La visualización debe mostrar el objeto detectado con un marco rectangular.

# Creación de una colección

- ▼ El usuario selecciona una ventana de imagen y la añade a la colección utilizando el botón “*Add object image*”.
- ▼ Por cada objeto:
  - ▼ La ventana de imagen seleccionada por el usuario debe representarse a diferentes escalas (p.e., 0.75, 1. y 1.25)
  - ▼ Por cada escala, deben almacenarse los puntos característicos y los descriptores correspondientes: métodos ***detect*** y ***compute*** (o ***detectAndCompute***) de la clase ***ORB***.
  - ▼ Los descriptores obtenidos para cada escala deben añadirse a la colección: método ***add*** de la clase ***BFMatcher***.

# Creación de una colección

## Representación de la imagen de objeto a diferentes escalas

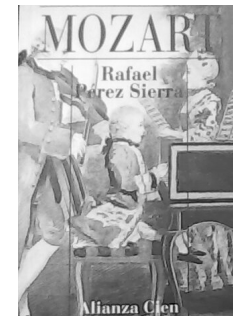
Escala 1.25



Imagen original



Escala 0.75



# Creación de una colección

## Cálculo de puntos característicos y descriptores de cada escala

Escala 1.25



Imagen original



Escala 0.75

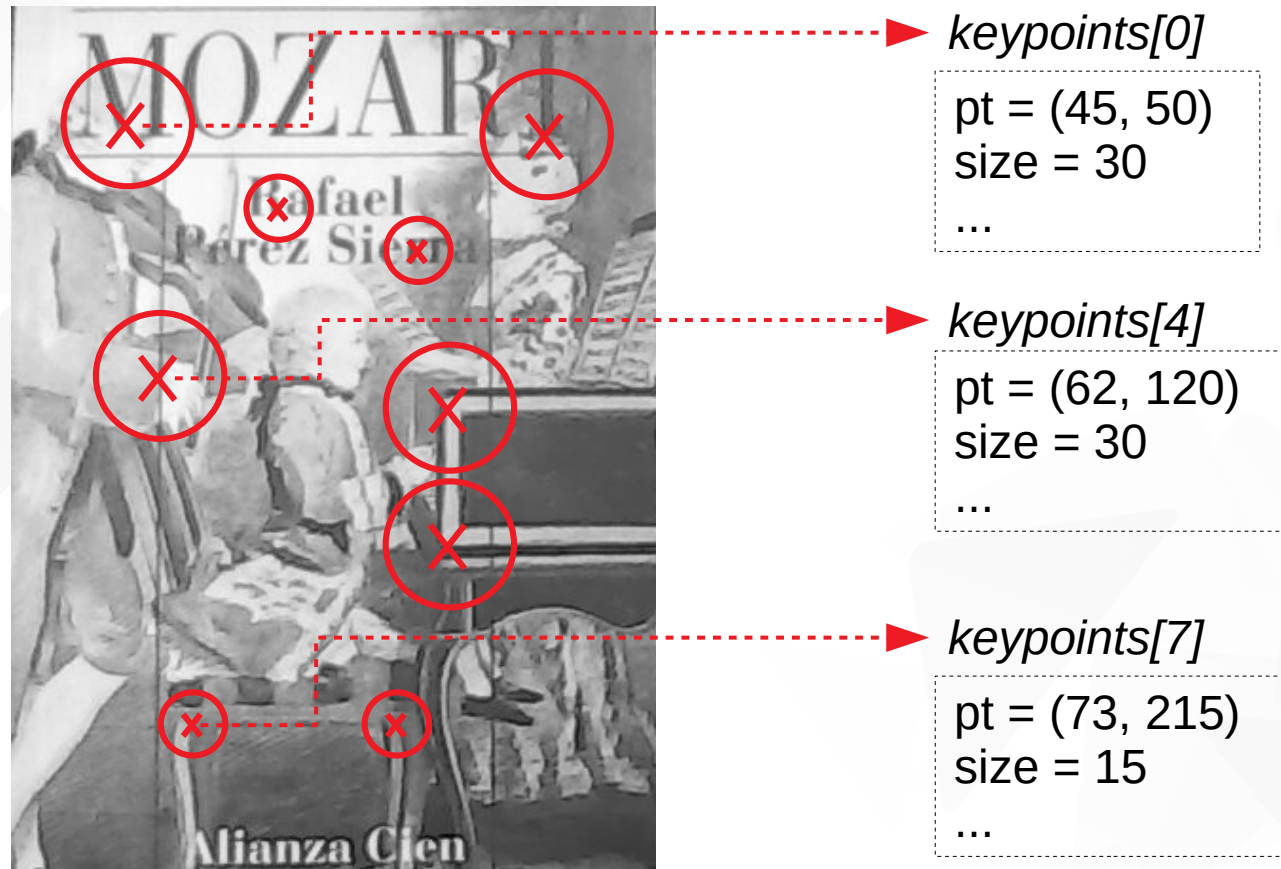


ORB —   
 ➤ `detect(img, keypoints)`   
 ➤ `compute(img, keypoints, descriptors)`

# Creación de una colección

## Cálculo de puntos característicos y descriptores de cada escala

`orbDetector->detect(img, keypoints)`



# Creación de una colección

Cálculo de puntos característicos y descriptores de cada escala

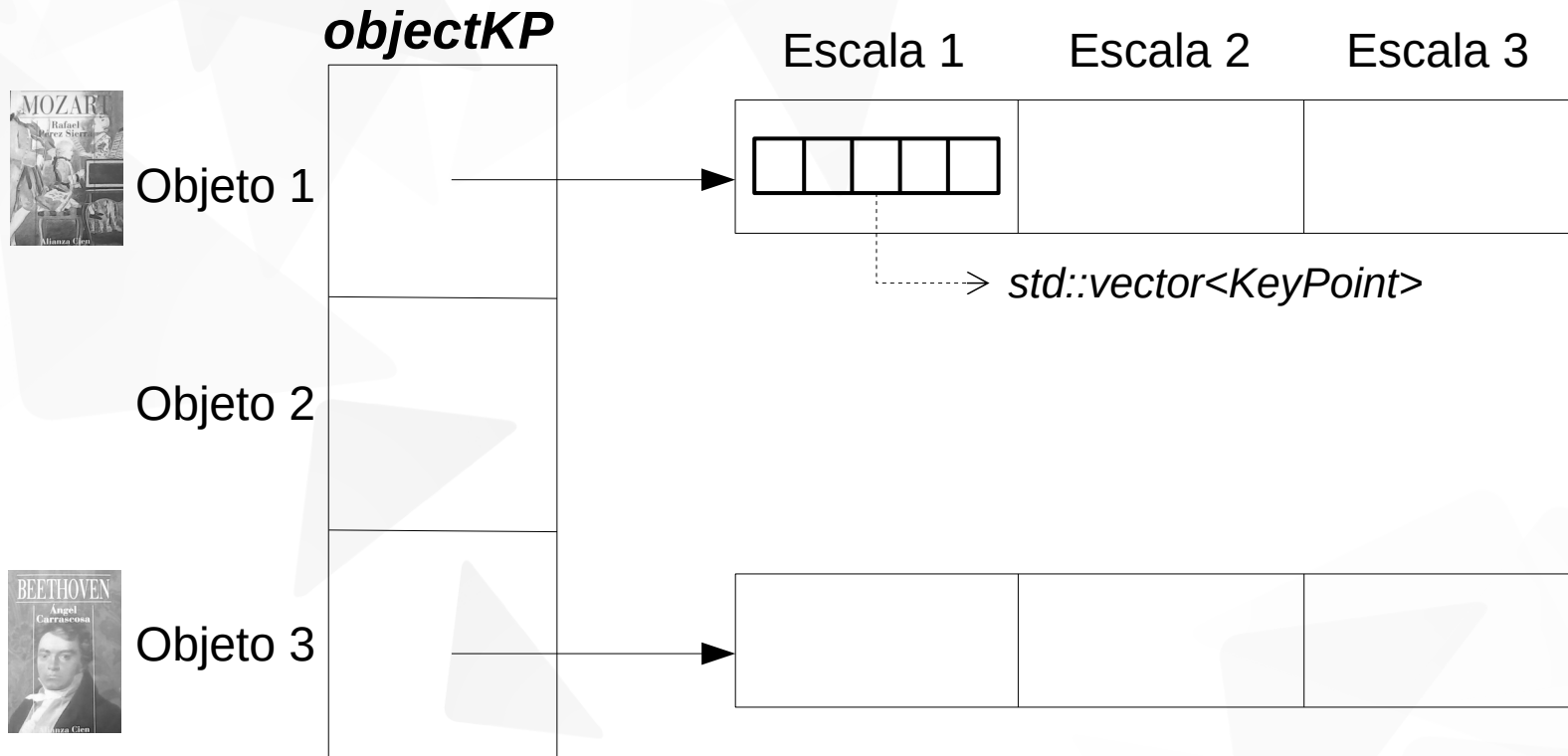
`orbDetector->compute(img, keypoints, descriptors)`





# Creación de una colección

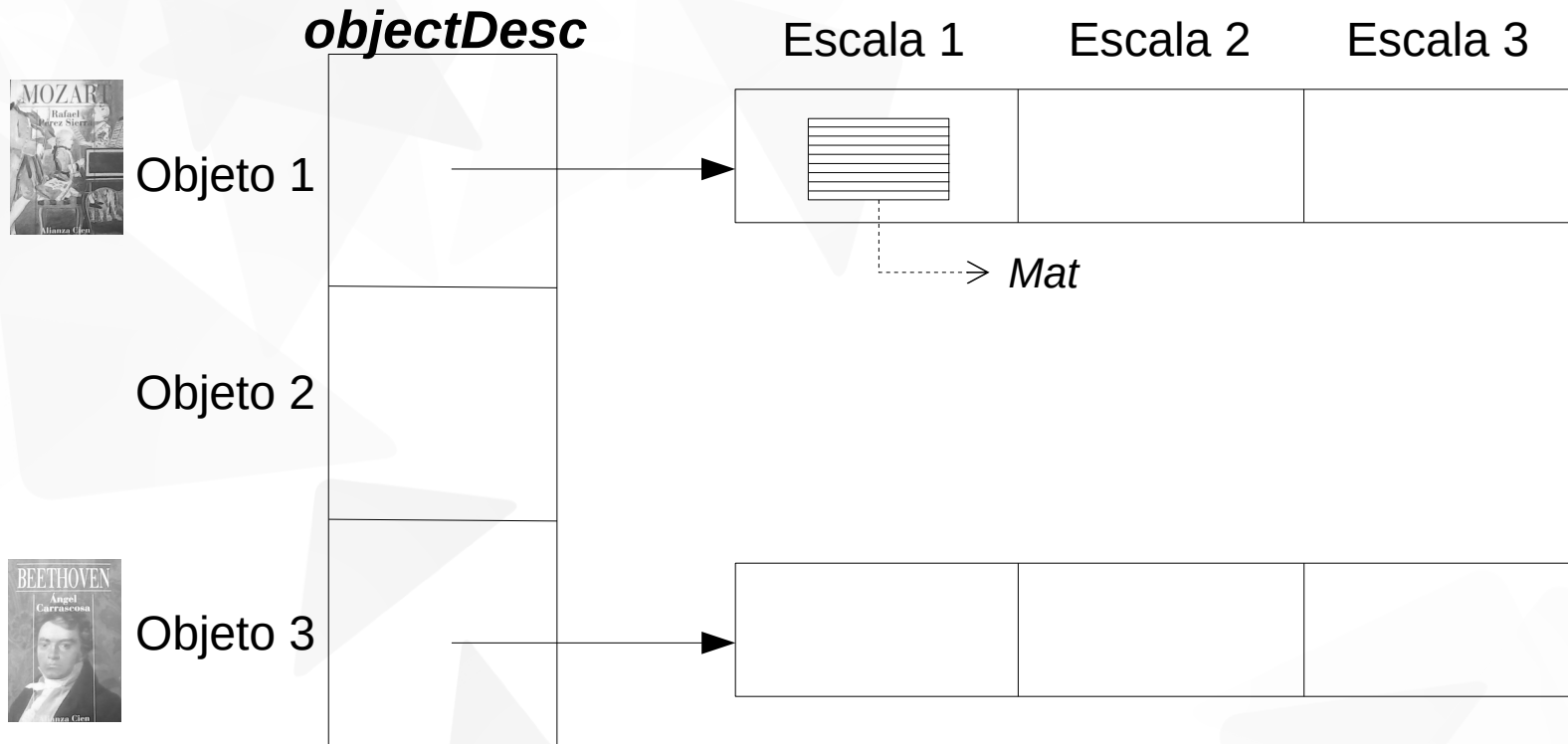
## Almacenamiento de puntos característicos (*objectKP*)





# Creación de una colección

## Almacenamiento de descriptores (*objectDesc*)



**Borrado de la colección previa:** `matcher->clear();`

**Creación de la colección:** `matcher->add(objectDesc[i]);` // *i*=índices de los objetos existentes

**Relación entre índice de colección e índice de objeto**

<i>colect2Object</i>			
0	1	2	→ Índice de colección
0	2	-	→ Índice de objeto

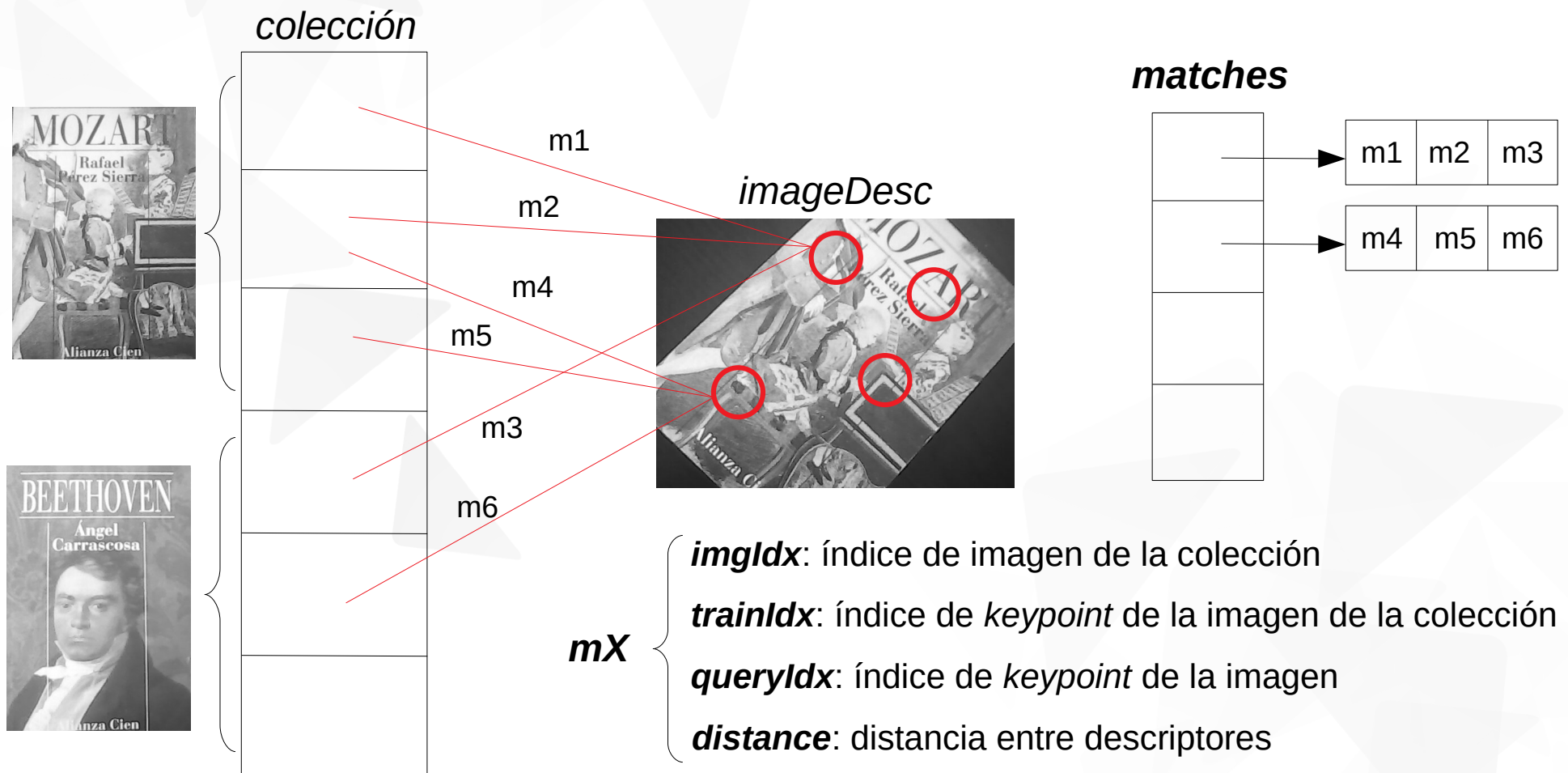
# Detección de objetos

- ▼ Obtener puntos característicos (*imageKP*) y descriptores (*imageDesc*) de la imagen actual.
- ▼ Poner en correspondencia los descriptores de imagen con los de la colección (método ***knnMatch*** de la clase ***BFMatcher***.)
- ▼ Organizar las correspondencias (con distancia menor que un umbral) por objeto y escala.
- ▼ Por cada objeto:
  - ▼ Seleccionar la escala con mayor número de correspondencias.
  - ▼ Organizar las correspondencias en 2 listas de puntos (una lista para puntos de imagen y otra para puntos de objeto)
  - ▼ Obtener la homografía a partir de las 2 listas de puntos.
  - ▼ Aplicar la homografía a las 4 esquinas.

# Detección de objetos

## Cálculo de correspondencias entre la imagen y la colección

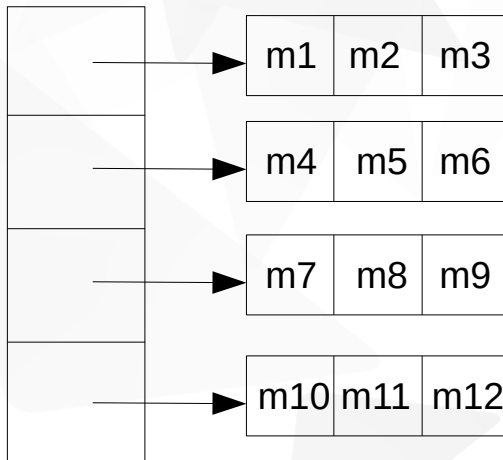
*matcher->knnMatch(imageDesc, **matches**,3)*



# Detección de objetos

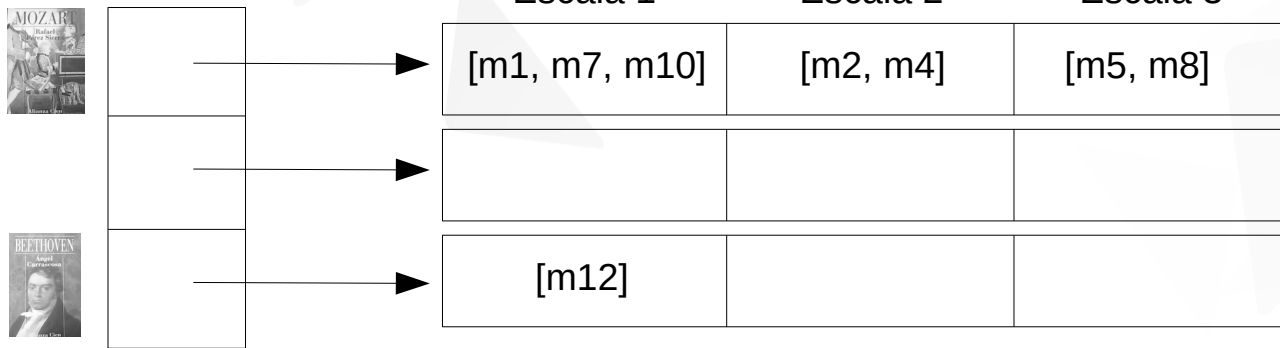
## Almacenamiento de correspondencias por objeto y escala (*objectMatches*)

*matches*



	m1	m2	<del>m3</del>	m4	m5	<del>m6</del>	m7	m8	<del>m9</del>	m10	<del>m11</del>	m12
<i>imgIdx</i>	0	1	3	1	2	4	0	2	5	0	2	3
<i>trainIdx</i>	3	4	10	8	9	2	1	3	7	5	6	4
<i>queryIdx</i>	0	0	0	1	1	1	2	2	2	3	3	3
<i>distance</i>	20	25	40	25	30	50	20	30	50	25	50	20

*objectMatches*



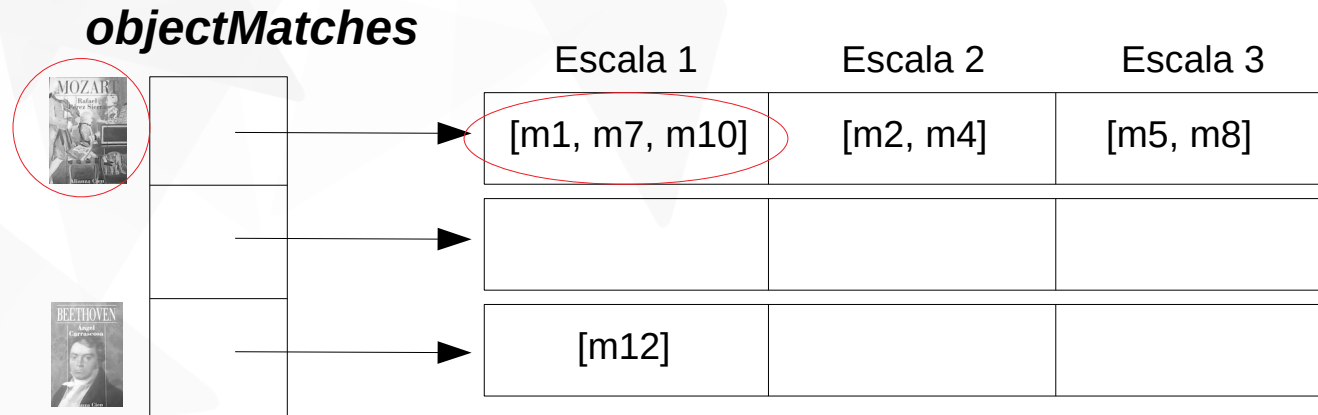
*distance* ≤ 30

objeto = *colect2object*[*imgIdx*/3]

escala = *imgIdx*%3

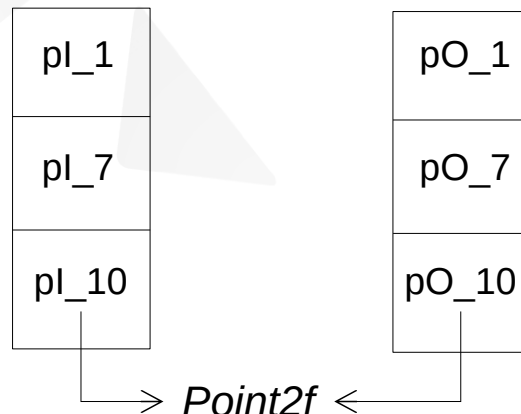
# Detección de objetos

## Selección de escala y generación de las listas de puntos en correspondencia



**imagePoints**

**objectPoints**



$pI\_X = \text{imageKP}[mK.\text{queryIdx}].pt$

$pO\_X = \text{objectKP}[0][0][mK.\text{trainIdx}].pt$

Índice de objeto

Escala seleccionada

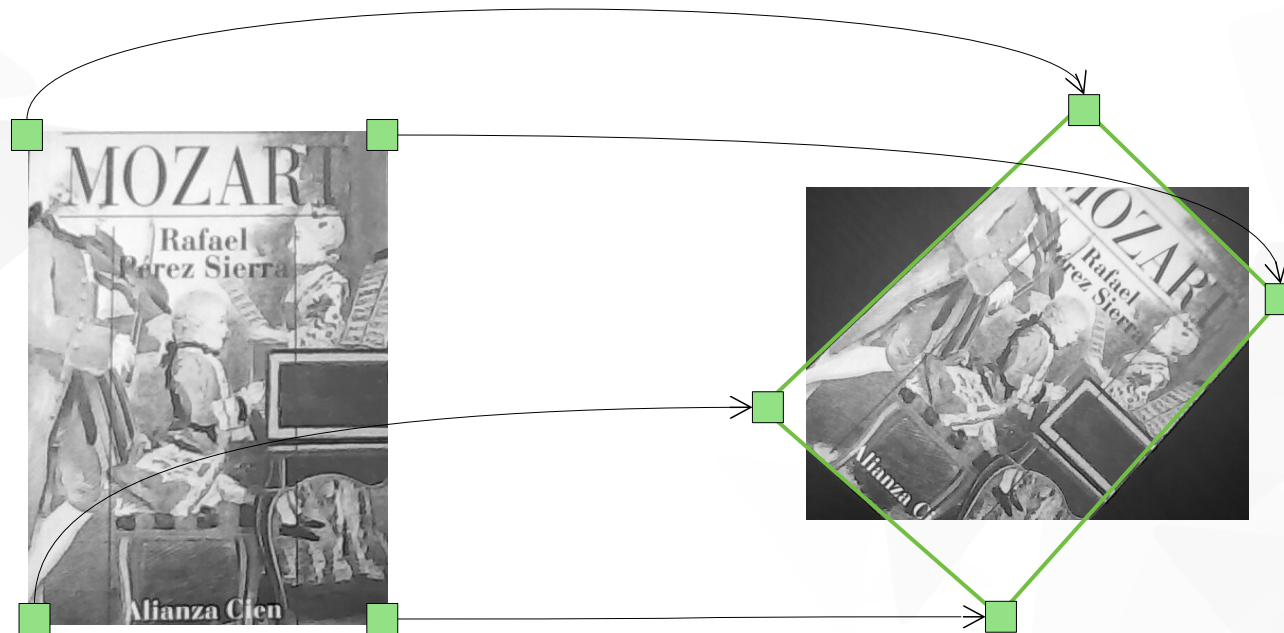
# Detección de objetos

Obtención de homografía a partir de las 2 listas de puntos

$H = \text{findHomography}(\text{objectPoints}, \text{imagePoints}, \text{LMEDS})$

↓  
Método de estimación

Aplicación de la homografía a las 4 esquinas del objeto

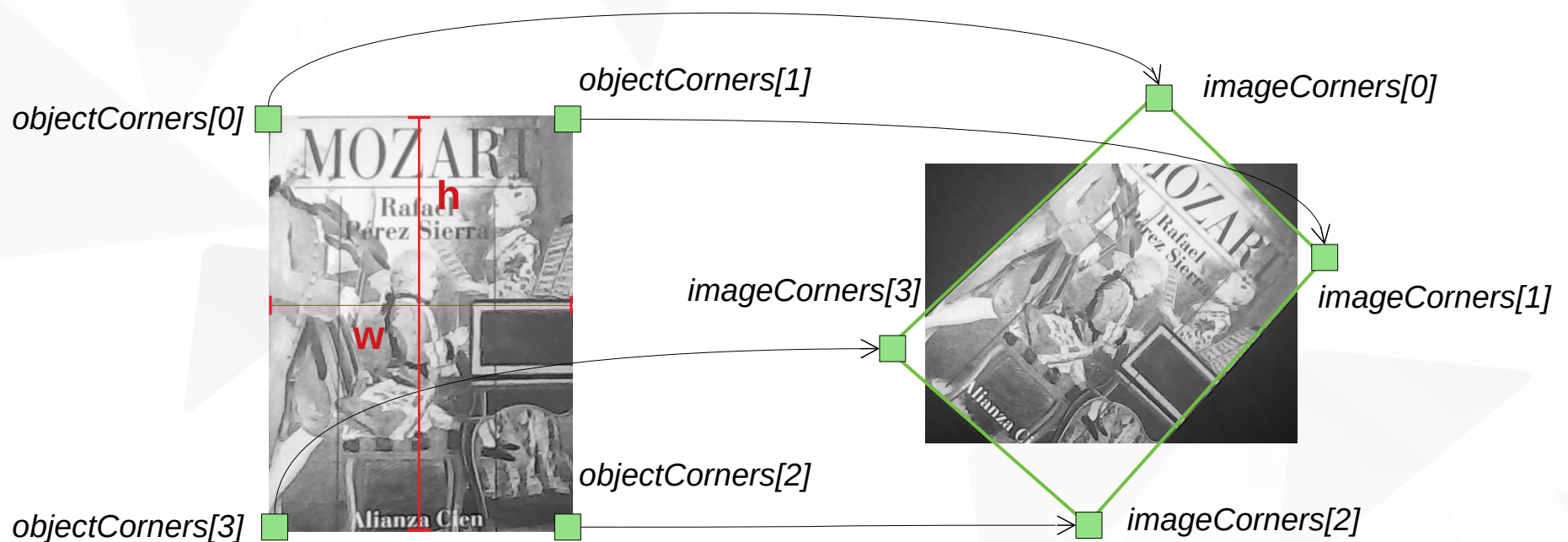


# Detección de objetos

## Aplicación de la homografía a las 4 esquinas del objeto

*perspectiveTransform(objectCorners, **imageCorners**, H)*

$\text{std::vector}<\text{Point2f}>$



***objectCorners*** = [(0, 0), (w-1, 0), (w-1, h-1), (0, h-1)]

**w** y **h**: dimensiones del objeto para la escala seleccionada



The background of the slide is decorated with a collection of triangles in various shades of green and grey. These triangles are of different sizes and are scattered across the slide, with a higher concentration in the top-left corner.

# Visión Artificial

## Práctica 3: detección de objetos